Parallel computing for real-time simulation and condition monitoring of fluid power systems

Pollmeier, Klemens

Award date:
1997

Awarding institution:
University of Bath

PARALLEL COMPUTING FOR REAL-TIME SIMULATION AND CONDITION MONITORING OF FLUID POWER SYSTEMS

Submitted by

Klemens Pollmeier

for the degree of PhD.

1997

UNIVERSITY OF BATH

COPYRIGHT

Klemens Pollmeier

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

SUMMARY

Fluid power systems are employed in many large-scale engineering applications. The analysis, synthesis, design and automatic monitoring of such, in general highly nonlinear hydraulic systems, is very difficult without high performance simulations.

The work in this thesis is aimed at developing a general parallel simulation methodology to facilitate the rapid and accurate dynamic analysis of large and complex hydraulic systems as well as establishing parallel simulations for condition monitoring purposes of these systems. It has been demonstrated that considerable improvements in simulation speed are possible using the parallel transmission line modelling method (TLM) method. Depending upon the numerical nature of the equivalent ordinary differential equation simulation, the speed increase achieved employing TLM could exceed more than two orders of magnitude. Because the TLM method is entirely general, the simulation and identification techniques proposed in this research should be readily extendible to accommodate a very wide range of complex engineering systems.

The concepts of processor load balancing and reduction of communication overheads are presented and evaluated for specific test-system simulations. Constraints imposed when configuring a parallel implementation of particular systems are addressed and solutions are evaluated. Genetic algorithms are investigated to solve the difficult process-processor mapping problem where computation and communication time needs to be accounted for. Additionally a new extrapolation-interpolation method is developed enabling the reduction of communication between processors.

Existing monitoring systems largely rely on range, or limit checking to initiate automatic corrections, or warnings. This may be appropriate for overall plant supervision, but incipient internal component faults often go undetected until breakdown occurs. This thesis investigates a different approach where neural networks are trained for the condition monitoring process. It is demonstrated that it is possible to train diagnostic networks with simulation data only enabling the identification of faults obtained from an experimental electrohydraulic servo system.

ACKNOWLEDGEMENTS

I would like to thank Prof. Kevin A Edge and Prof. Clifford R Burrows for their guidance throughout this project. Without their excellent and valuable supervision this thesis would never have been possible.

My sincere thanks are also due to colleagues and friends at the Mechanical Engineering Department, both past and present, many of whom have made this a most memorable and rewarding part of my career.

I am grateful to the Science and Engineering Research Council for their financial support of this project.

Finally, I would like to thank my parents, sister and brothers for their support, and encouragement during my time at university.

Contents

SUMMARY ii
ACKNOWLEDGEMENTS iii
CONTENTS iv
NOTATION viii

1 Introduction 1

1.1 Background 3
1.1.1 Real-time parallel simulation using transmission line modelling (TLM) 3
1.1.2 On-line condition monitoring 4
1.2 Organisation of this thesis 5

2 The transmission line modelling method 7

2.1 Introduction 7
2.2 The historic development of the transmission line modelling method 7
2.3 TLM modelling of fluid power and mechanical systems 9
2.3.1 Basic equations for transmission-line modelling of fluid power systems 11
2.3.2 Modelling example 12
2.3.3 Compressible line 13
2.3.4 Stability of TLM simulation 16
2.3.5 TLM compared to other numerical techniques 17
2.4 Component modelling 18
2.4.1 Directional control valves 19
2.4.2 Pressure relief valve model 21
2.4.3 Differential area actuator model 22
2.4.4 Filter model 22
2.4.5 Positive displacement pumps and motors 23
2.5 Closure 24
TABLES FOR CHAPTER 2 25
FIGURES FOR CHAPTER 2 25

3 Simulation with variable time steps 30

3.1 Introduction 30
3.2 Variable time step TLM 30
3.2.1 How to control the variable time step 31
3.2.2 Modifications necessary for variable timestep TLM 33
3.2.3 Variable line volume and bulk modulus 33
3.3 Comparison between fixed and variable time step simulations 34
3.3.1 Simulation Example: Two-actuator Circuit 34
3.3.2 Further problems associated with variable time step TLM 36
3.4 Closure 37
FIGURES FOR CHAPTER 3

4 Simulations with fixed time steps on parallel platforms 41

4.1 Introduction 41
4.2 Parallel simulations using TLM 41
4.3 Hardware description 43
4.3.1 Description of the new T9000-based system 43
4.3.2 Memory arrangement 43

4.3.3 The implemented data acquisition system 45
4.3.4 Granularity 46
4.4 Parallel implementation of TLM on T9000-based platform 46
4.4.1 Program generator 46
4.4.2 Simulation process 47
4.4.3 Other files required by the make file 49
4.4.4 TLM numerical algorithm 52
4.4.5 Communication and synchronisation between processors 53
4.5 Portability of the program generator concept to other platforms 54
4.5.1 Message passing interface (MPI) 54
4.5.2 Porting to MPI 55
4.6 Closure 57
TABLES FOR CHAPTER 4 57
FIGURE FOR CHAPTER 4 58

5 Process-processor mapping 64

5.1 Introduction 64
5.2 Partitioning and scheduling (process-processor mapping) in general 64
5.2.1 Partitioning 65
5.2.2 Scheduling or assignment of tasks to processors 67
5.2.3 Graph theoretic model of the mapping problem 69
5.2.4 Heuristic and approximate static allocation methods 71
5.3 Genetic algorithm solutions to the mapping problem 71
5.3.1 The fitness/cost function 74
5.3.2 How to calculate the fitness function 75
5.3.3 Software support 77
5.4 Deadlock-free communication scheme 77
5.5 Partitioning examples 79
5.5.1 Performance measures of parallel systems 79
5.5.2 Small-scale partitioning example 80
5.5.3 Small-scale example: Simulation performance 82
5.5.4 Medium-scale partitioning example 82
5.5.5 Medium-scale example: Simulation performance 83
5.5.6 Large-scale partitioning example 84
5.5.7 Large-scale example: Simulation performance 85
5.5.8 Cost and scalability 85
5.6 Simple heuristics to improve GA performance 85
5.6.1 Pre-grouping of components 86
5.6.2 Simulation results 86
5.6.3 Runtime validation 87
5.7 Differences to the TLM implementation in HOPSAN 88
5.8 Closure 89
TABLES FOR CHAPTER 5 89
FIGURES FOR CHAPTER 5 90

6 Reduction of inter-processor communication 104

6.1 Introduction 104
6.2 New extrapolation-interpolation method 104
6.2.1 New filter for approximation of friction 104
6.2.2 Extrapolation 105

6.2.3 Interpolation 106
6.2.4 Adjustments of the characteristic impedance 107
6.3 Simulation results and performance improvements 107
6.3.1 Two-actuator circuit 107
6.3.2 Complex hydrostatic transmission circuit 109
6.4 How to choose the communication time step 110
6.5 Estimated improvements 111
6.6 Other methods to increase computation efficiency and speed up simulation 112
6.7 Closure 113
TABLES FOR CHAPTER 6 113
FIGURES FOR CHAPTER 6 114

7 Non-linear system identification using genetic algorithms 119

7.1 Introduction 119
7.2 System identification 119
7.2.1 Non-linear system identification 121
7.2.2 Identification of fluid power systems 123
7.2.3 System identification and optimisation using genetic algorithms 125
7.3 Genetic algorithms for the identification of fluid power systems 125
7.3.1 Hooke Jeeves search 126
7.3.2 Experimental hydraulic actuator system 127
7.3.3 Simulation model of the hydraulic actuator system 128
7.3.4 Investigation of different cost functions 128
7.4 Parameter identification 130
7.4.1 Choice of parameters to be identified 130
7.4.2 Details on the GA used for the parameter identification 133
7.4.3 Importance of individual parameters for the optimisation 134
7.4.4 Confidence measures and quality of the simulation 135
7.4.5 Fault level identification using GAs 136
7.4.6 Condition monitoring using GAs 137
7.5 Closure 138
TABLES FOR CHAPTER 7 139
FIGURES FOR CHAPTER 7 141

8 Condition monitoring of fluid power systems using neural networks 153

8.1 Introduction 153
8.2 Fault monitoring techniques 153
8.2.1 Limit checking 154
8.2.2 Statistical methods 155
8.2.3 Condition monitoring based on vibration measurement 156
8.2.4 Expert systems 157
8.2.5 Fault tree analysis 158
8.2.6 Model based diagnostics 158
8.2.7 Parameter estimation methods 159
8.3 Condition monitoring of fluid power systems 160
8.3.1 Vibration and noise 160
8.3.2 Temperature 161
8.3.3 Pressure 161
8.3.4 Flow and leakage 161
8.3.5 Contamination and fluid condition 161

8.3.6 Power consumption and efficiency measurements 162
8.3.7 Other 162
8.4 Artificial neural networks for condition monitoring 162
8.5 Description of the neural networks used for condition monitoring 164
8.5.1 The main units of feedforward neural networks 164
8.5.2 Network training and validation 165
8.5.3 Determination of network topology 166
8.5.4 Data processing 167
8.6 Simulation study using reference model 168
8.6.1 Training and validation data 169
8.6.2 Single fault networks 170
8.6.3 Neural networks to identify several faults 172
8.6.4 Modular neural network approach 172
8.7 Monitoring of experimental systems using reference models 174
8.7.1 Training and validation data 174
8.7.2 Single fault networks 175
8.7.3 Single fault networks using residuals 176
8.7.4 Interpreting weights 178
8.8 Monitoring of experimental systems without reference model 179
8.8.1 Single fault networks trained with measured data 179
8.8.2 Single fault networks trained with simulated data 180
8.8.3 Modular neural network approach 180
8.9 Improvements of the neural network approach and real-time performance 181
8.10 Closure 182
TABLES FOR CHAPTER 8 184
FIGURES FOR CHAPTER 8 186

9 Conclusions and further work 208

9.1 Conclusions 208
9.2 Recommendations for further work 211

REFERENCES 213

Appendix A The basic equations for TLM modelling of fluid power systems 222

A1 Continuity 222
A2 Momentum 223
A3 Solving the differential equations 225

Appendix B The TLM method as a general method for integration 230

Appendix C Adjustment of the characteristic impedance 232

Appendix D Component modelling 234

D1 Directional control valves 234
D2 Pressure relief valve model 236

Notation

A	pipe cross-sectional area; assignment vector
a	node weights, speed of sound
B	fluid bulk modulus; connection matrix
B_e	effective fluid bulk modulus (including pipe wall compliance)
b	edge weight; binary string length
C	characteristic pressure; capacitance; communication time matrix; velocity proportional friction
c	acoustic velocity
D	calculation times vector
E	set of edges; efficiency
e	system noise; output error
F	force
f	viscosity pressure dependent factor
G	transfer function; undirected graph
G_p	undirected program graph
G_s	undirected system graph
G_f	filter transfer function
g	activation function
H	help variable
h	Volterra kernel
I	inertance
i	current
J_0	Bessel function of the order 0
J_2	Bessel function of the order 2
j	complex number
K	relief valve flow characteristic
K_I	time step controller coefficient
K_P	time step controller coefficient
k	flow characteristic; transfer function coefficients
L	line length; inductance
M	mass
MSE	mean square error
m	integer number; ratio of error terms
N	viscous friction coefficient; number of nodes
n	integer factor
$obfn$	objective function; cost function
p, P	pressure
q	flow rate; shift operator
Q	flow rate; radius
R	resistance
R_e	Reynolds number
r	pipe inner radius; gradient
res	resolution
S	speedup
s	Laplace operator
t	time
Δt	timestep
T	global time step; time delay
U	input vector
u	demand signal
$\bar{u}$	averaged velocity
V	line volume; vertice; set of nodes
v	node
w	neural network weights
$\mathbf{X}$	network input vector
x	spool fractional displacement; longitudinal pipe coordinate; network input units
y	longitudinal coordinate; observation sequence; network output units
Z	impedance
z	network hidden units
α	filter coefficient
γ	impedance relationship
ρ	fluid density
Φ	potential
μ	dynamic viscosity; mean values
ν	kinematic viscosity
Θ	set of all possible mapping functions

σ variance
θ mapping function
ω_n natural frequency

SUBSCRIPTS

a line end 'a'
b line end 'b'
c characteristic
f faulty
in incoming values
l leakage
leak leakage
n time step
o operating point
p process
par parasitic
r return line; reference
res resistive
s supply line; system
spec specified value
stic stiction
tr trapezoidal rule
x x-direction

SUPERSCRIPTS

′ compensated; adjusted
* altered
e execution time
c communication time

1 Introduction

Fluid power systems are employed in many large-scale engineering applications, including marine and aircraft systems, automobile applications and steel making plant. In general, safety, integrity and efficient operation of these systems is of great importance. The analysis, synthesis and design of such, in general highly nonlinear hydraulic systems, is very difficult or impossible without high performance simulations.

The simulation of continuous systems began more than forty years ago using analogue computers. Such simulations were usually performed in the time domain and are restricted to solving sets of ordinary differential equations. The development of the digital computer soon caused a diminishment in the popularity of the analogue computer for simulation purposes. One reason for this is that analogue computers are unable to cope efficiently with multi-variable function generation, a simple procedure for a digital computer. Since the fifties the speed of digital computers has been improved by about six orders of magnitude.

Nevertheless, the computational demands of many complex scientific and engineering problems cannot be met by a single computer. Furthermore, sequential processing must reach its limit. The advances in microelectronics technology have made parallel computing possible offering the potential for solving very large problems that could not be solved a few years ago. However, in order to use parallel computers for a specific application, existing algorithms need to be restructured for the particular architecture and new algorithms developed. In fact, conventional algorithms need to be re-examined, since the best algorithm for a sequential computer may not be the best for a parallel computer [Özgüner & Erçal, 1991].

Computer modelling of hydraulic systems is non-trivial and involves solving a system of differential equations incorporating significant non-linearities, including discontinuous operation (valve opening/closing), saturation (actuator end-stops) and hysteresis (stiction). Numerical techniques using a multi-step, multi-order integrator coupled with an appropriate re-start capability at discontinuity points, can improve simulation efficiency dramatically. However, the simulation of a complex system involving many components can still be time consuming even when using a high performance processor. Real-time performance cannot be achieved.

Detailed and sophisticated simulations have many merits. They can reduce the number of costly prototype tests and they may be repeated to allow for sensitivity case studies or for extremes of operating conditions. Savings due to simulations often result in savings of overall project cost as a consequence of a simulation study at the design stage. Parallel processing enables the simulation of more complex models incorporating non-linear effects. For multi-variable plant optimisation where many (thousands) simulations of the same plant model have to be calculated parallel processing can efficiently lead to effective solutions. On the other hand, assuming sufficient speed-up of the simulation this can be used for on-line condition monitoring and/or model reference adaptive control. Critical operating conditions can then be detected and warnings or maintenance can be initiated. Reliance may also be placed on the use of on-condition maintenance, as opposed to fixed maintenance schedules. Condition monitoring is especially important in applications such as steel manufacture, where unexpected failures result in costly production losses and/or operation hazardous to personnel.

Existing monitoring systems largely rely on range, or limit checking to initiate automatic corrections, or warnings. This may be appropriate for overall plant supervision, but incipient internal component faults often go undetected until breakdown occurs. An increase in the number of sensors to measure directly plant parameters indicative of certain failure conditions does not necessarily improve plant reliability. Furthermore, many faults remain undetectable through the use of existing transducer technology.

It is possible to perform condition monitoring through a comparison of the measured performance with that predicted by a steady-state model. This may be very effective and generally does not require substantial processing power. However, many systems are self-compensating (for example pressure compensated pumps) and conventional steady-state limit checking techniques of plant states may not detect a developing fault early enough.

The use of a dynamic plant reference model allows the identification of parameters which cannot be determined through the use of simple steady-state models. Careful selection of the plant states to be monitored can allow a reduction in the number of sensors required to identify unknown system parameters. Furthermore, functionally related measurements may allow the verification of some sensor outputs.

1.1 Background

1.1.1 Real-time parallel simulation using transmission line modelling (TLM)

Parallel simulation of systems offers the benefit of increased speed of execution, but requires the system model to be partitioned effectively to enable numerical tasks to be performed concurrently on individual processors. Hydraulic systems are characterised by a transport delay in the pipelines connecting physical components, which is due to the propagation of waves at the speed of sound through the fluid medium. The transmission delay allows component models to be decoupled, enabling a parallel solution; the inputs to each component model are delayed outputs from connected models. This transmission line modelling (TLM) approach has the potential of real-time simulation of large and complex hydraulic plant, which is currently possible only with limited linear system models. All important real plant dynamics including non-linearities can be modelled in a parallel processor scheme, resulting in significantly improved reference models.

The current availability of low cost hardware platforms and the recent development of software tools for parallel programming allows the transmission line modelling method (TLM) to be developed in a true multiprocessor environment. Technology is constantly improving in terms of ever-increasing processor speed and as such this research is concerned with real-time performance as well as relative performance improvements. The concepts of processor load balancing and reduction of communication overheads are presented and evaluated for specific test-system simulations. Constraints imposed when configuring a parallel implementation of particular systems are addressed and solutions are evaluated.

The research described in this thesis is a continuation of a previous project on parallel simulations of hydraulic systems using TLM [Burton, 1994]. Packages with automatic code generation facilities are intended to take computer programming and simulation coding out of the hands of the user. Such packages have been developed for particular industries but mainly use single processor computing. The work in this thesis is aimed at developing a general parallel simulation methodology to facilitate the rapid and accurate dynamic analysis of large and complex hydraulic systems as well as establishing parallel simulations for condition monitoring purposes of these systems.

The performance of a specific computation on a multiprocessor is affected by the processor and communication architecture, the interconnection network topology, the I/O subsystems, and the parallel algorithm and communication protocols. Each of these aspects is a complex problem in itself and solutions require an understanding of the interactions among them. An important part of this research addresses the problem of automatic program generation for parallel computing platforms. This includes the investigation of genetic algorithms to solve the difficult process-processor mapping problem.

1.1.2 On-line condition monitoring

Man has always been interested in preventing catastrophes, being a consequence of their works or of natural causes. As technology advances, it seems that unfortunately, the risk of catastrophic events occurring, increases. The latter fact is the result of bigger and more complicated plants, which makes it impossible for human operators to manage or control them. Thus the need for automatic or operator-aiding fault monitoring systems [Pouliezos & Stavrakakis, 1994].

There has been substantial effort devoted in recent years to the development of system identification techniques for both linear and non-linear systems (Thau [1973], Chow & Willsky [1984], Sato et al. [1991], Zhou et al. [1991]). In most schemes, a model of the plant is employed together with a (limited) number of measured plant states in order to establish unknown parameters. A fault condition is indicated when an identified parameter is determined to be outside a recommended tolerance limit at a given operating condition. Little work appears to have been published on the use of real-time non-linear reference models for condition monitoring of fluid power systems. Whatever type of identification scheme is employed it is important to avoid over-complexity and consequently, full use should be made of a priori information.

Recently, neural networks (NNs) have been found to be very powerful in solving control problems. Many applications use the NNs to build non-linear process models. The difference between model and plant output is then employed to adjust controllers. This thesis investigates a different approach where neural networks are trained for the condition monitoring process. It is demonstrated that it is possible to train diagnostic networks with simulation data only enabling the identification of experimental faults.

1.2 Organisation of this thesis

Tables and figures are always shown at the end of each chapter. The reminder of this thesis is divided into a further eight chapters, structured in the following way:

Chapter 2: Some of the current literature in several fields of the transmission line modelling method (TLM) is reviewed in this chapter. The historic development of TLM is recalled and the basic equations are derived. The method is compared with other numerical techniques and its stability is investigated. Possible alternatives to the parallel, dynamic simulation of hydraulic systems are described in Burton [1994], hence they are omitted in this thesis. A simple example circuit is investigated using the basic equations in order to illustrate the TLM modelling of hydraulic systems. Some new component models are developed and details about these are given. Additionally modelling methods enabling simplified model development are investigated.

Chapter 3: This chapter recounts variable time step methods derived for the simulation of thermal diffusion problems. A similar approach is then taken for hydraulic circuit simulation in order to improve computing efficiency. Problems associated with variable time step simulation are investigated and the simulation results of a stiff example system are compared with fixed time step simulations.

Chapter 4: The TLM method is inherently parallel and has been developed for parallel platforms in order to speed up simulation. In this chapter the process of automatic system model generation is described for the case of multi-processor TLM simulations. Advantages of the new T9000-based platform over the previously employed T800-based system are shown. The TLM simulation process and the TLM numerical algorithm by which the TLM models are solved are described. Channels as a concept to communicate and synchronise between processors are introduced. The principal way of porting the software to other platforms is also described.

Chapter 5: Using the TLM method, decoupling system components is straightforward, but the partitioning of the tasks onto processors cannot be done easily. This so called assignment problem is known to be NP-complete, i.e. optimal partitioning strategies lead to very large computation times. In this chapter the process-processor mapping problem has been solved using an automatic method based on genetic algorithms. The aim of the

partitioning scheme is to reduce the overall computation time of a simulation including both the computation and communication time.

Chapter 6: For parallel simulation the main aim of reduction of execution time can be achieved by even load balancing, by minimising CPU contention and by reduction of the communication between different processors [Sunter & Bakkers, 1994]. This chapter reports an investigation on the latter for simulations with fixed time step algorithms. A new extrapolation-interpolation method is developed enabling the reduction of communication between processors. This method also enables the use of more mainstream computers for efficient and large scale parallel TLM simulations.

Chapter 7: In this chapter it is shown how to optimise the parameters of a simulation in order to match simulated and measured values. A genetic algorithm (GA) is developed and employed enabling the identification and optimisation of several parameters simultaneously. This requires the non-linear TLM component models in order to simulate the highly non-linear hydraulic systems correctly. The identified parameters, describing physical properties of the system, may then be used for condition monitoring purposes.

Chapter 8: This chapter investigates the condition monitoring process of fluid power systems. Through the use of artificial neural networks (NNs) different faults can be identified correctly. An approach is developed that enables the detection of different levels of faults and system deterioration in real-time. A parallel implementation of the approach is also suggested. The purpose of this section is not to determine causes but rather to discuss techniques which identify the presence of failure characteristics and its source.

Chapter 9: Finally, conclusions are drawn from the various aspects of this research and some possible areas for future investigations are suggested.

2 The transmission line modelling method

2.1 Introduction

This Chapter reviews some of the current literature in several fields of the transmission line modelling method (TLM). The historic development of TLM is recalled and the basic equations are derived. The method is compared with other numerical techniques and its stability is investigated. An example circuit is investigated in detail in order to illustrate the TLM modelling of hydraulic systems. In all but the simplest component models, the TLM approach can result in rather complex algebraic equations. Therefore the simplification of the modelling process of some component features is developed and some model details are outlined.

2.2 The historic development of the transmission line modelling method

TLM is characterised by its close relationship to the physical process being simulated. The general approach to the solution of engineering problems is to formulate continuous models which can then be solved by transformation to the discrete time-domain. TLM can also be used to derive discrete time-domain models directly from the physical systems [Hui & Christopoulos, 1991]. In other words, the modelling process conceptually resembles the actual propagation process. The basic calculations required are very simple, and may be implemented efficiently on a computer. Difficulties of implementing the technique are also independent of the model size. The model can be developed and tested on small problems, and scaled to accommodate larger problems without becoming unstable or invalid. Depending on the process being modelled, this can be in one, two or three dimensions.

Transmission line modelling (TLM) was developed in the early seventies as a time-domain, numerical technique for simulating electromagnetic wave propagation and later for other electronic and electrical problems. The first significant impetus to the development of TLM was a paper by Johns & Beurle [1971]. The space to be modelled is represented as a Cartesian mesh of electrical transmission lines. Each node in the mesh corresponds to a junction between a pair of transmission lines and this forms an impedance discontinuity in each line. The mesh is represented by lumped inductors and capacitors. If the inductance and capacitance, per unit length, Δx, for an individual line, are L and C, respectively, the junction between a pair of lines at a mesh node point can be

represented by a basic elementary network. In Figure 2.1 such a network is given for one-dimensional problems.

In two dimensions each node communicates with its neighbours through four ports. Figure 2.2 shows a section of a two-dimensional mesh, and illustrates the scattering and reflection process. One circle (shaded grey) is shown at different instances in time. At instance I an impulse is incident on the node, which scatters into four reflected impulses at instance II. These voltage pulses incident on ports 1 to 4 are scattered at each time step and the resulting reflected pulses are incident on the relevant ports of adjacent nodes (instance III) [Christopoulos, 1991]. Hence, the simulation process is discretised in time and space so that impulses reflected from a node in one iteration are incident on its neighbours in the next. An accurate numerical model for the propagation of voltage impulses can then be implemented in software. In electrical terms the parameters of the connecting transmission lines and their manner of connection are governed by Kirchoff's laws. This gives rise to a reflection coefficient of -1/2, resulting in a negative impulse of half the incident amplitude being reflected in the incident element. Positive impulses of the same amplitude are transmitted in the other three directions. Boundaries in the propagating medium are achieved by placing resistive loads at nodes [Pomeroy, 1991]. If impulses are injected at a number of adjacent nodes a wavefront can also be generated as shown in Figure 2.2.

The TLM models wave propagation and is ideally suited to simulating sound waves travelling through a medium. Hence, it has been applied to a wide variety of wave propagation problems in homogeneous and inhomogeneous media. Over the years, new versions of the TLM method have been developed in order to improve the modelling of arbitrary inhomogeneous media. These versions usually involved changes to the topology of the transmission lines used to approximate the physical problem. The TLM method was further developed to tackle problems other than electromagnetic, namely: circuit simulation (Johns & O'Brien, 1980], [Hui et al., 1993a,b]), thermal problems (Pulko & Newton [1991], Webb [1991], de Cogan & Enders [1991], Pulko & Olashore [1989], Pulko et al. [1990]), acoustic propagation problems (Saleh [1991], Zhang et al. [1991], Pomeroy et al. [1993]) and mechanical problems (Partridge et al. [1987], Krus et al. [1990], Krus [1995]).

There have been many applications of TLM in the area of heat and mass diffusion and related phenomena [Pulko & Newton, 1991]. These include modelling of the soaking of rice grains, modelling the behaviour of thin film fuses, predicting the temperature fields in

glass mould tools, simulating thermal effects in power semiconductors and modelling the melting of ice. The recently acquired ability to represent equations involving cross derivatives enables the modelling of heat transfer in anisotropic materials [Pulko & Wilkinson, 1996]. Saleh [1991] describes the application of TLM in sonar and under water acoustics where compressional waves can be accounted for. The method may be used for the analysis of radiation patterns of transducer arrays, beam forming and ray tracing techniques. A free space boundary for the TLM modelling of acoustics is described by Pomeroy et al. [1993] and the TLM modelling of the behaviour of ultrasonic in air is investigated by Zhang et al. [1991].

One of the important features of the TLM method is that it determines the response to a delta-function impulse at the input. This means that one calculation contains all the necessary information about the frequency response for frequencies from zero to infinity [Johns & Beurle, 1971]. Not only is the impulse response of a structure obtained, yielding in turn its response to any excitation, but the characteristics of the dominant and higher order modes are also accessible in the frequency domain through a Fourier transform [Sadiku & Agba, 1990].

Currently, specialised hardware for the solution of TLM problems is also developed. An application specific processor for the solution of two-dimensional acoustics of electromagnetic TLM models has been investigated by Stothard & Pomeroy [1996]. By optimising the design to perform only the TLM algorithm, removing the need for instruction fetching and decoding, a very high throughput can be achieved using a simplified processing element. The system can be expanded if faster processing or greater storage capacity is required.

2.3 TLM modelling of fluid power and mechanical systems

A simple analogy exists between electrical systems and hydraulic systems, where voltage = pressure and current = flow(rate), i.e. methods developed for electrical problems can be transformed and used to solve hydraulic problems. A simple example is given in Figure 2.3 where a resistance, an inductance and a capacitance are connected in series. A large volume of oil can be described as a capacitance, for instance.

Transmission line modelling of fluid power systems is a relatively new concept. Boucher & Kitsios [1986] investigated fluid pipe network dynamics using TLM. Similar to acoustic waves, decoupled left- and right going waves carrying pressure and flow can be separated.

Using a variant of these waves representing power, it can be shown that such decoupled waves permit fluid transmission lines to be modelled simply as pure time delays. It is also demonstrated that lumped inertance and capacitance may be modelled as transmission line stubs. Thus all dynamic elements in a fluid power circuit are represented as pure time delays and the only computation required relates to wave scattering junctions.

Different fluid friction models and approximations for the simulation of fluid pipe lines have been investigated by Krus & Palmberg [1987], Burton [1992] and Krus et al. [1994a] based on work from Viersma [1980] and D'Souza & Oldenburger [1964]. Recently the TLM method has been developed to simulate general fluid power components and hydraulic systems (Krus et al. [1990], Jansson et al. [1992], Burton [1992], Burton et al. [1993a,b], Burton [1994]).

A variable time step TLM was derived by Jansson et al. [1992]. The timestep is controlled by a Proportional plus Integral (PI) controller according to the pressure 'error' between two line ends. This is explained in detail in Chapter 3. The analysis of electro-hydraulic position control servo-systems using TLM is investigated in a paper by Burton et al. [1993a]. The method described enables TLM simulation of hydraulic circuits including systems with control elements. Component models have to be calculated in a certain order to ensure that electrical signals are propagated instantaneously between component models. The TLM method can also be extended to pipes with elastic walls. Krus et al. [1991a] describes the simulation of the human arterial tree, using transmission line elements with visco-elastic walls.

It appears to be profitable to use one type of model to study complete systems [Partridge et al., 1987] such as hydromechanical systems [Krus et al., 1990] and it is possible to use much larger time steps than with conventional simulation methods. An earlier paper from Partridge et al. [1987] investigates TLM modelling of shaft system dynamics. Non-linear elements are replaced by discrete models and an explicit procedure is used whereby torque and angular velocity at one time step are expressed in terms of values at previous time steps only. Another paper on the modelling of mechanical systems using rigid bodies and transmission line joints was presented by Krus [1995]. The general TLM model can be used to represent both lines in a hydraulic system and springs in mechanical systems. In order to simplify the modelling of rigid mechanical link systems the rigid joints can be replaced with joints with some flexibility. This will increase the number of degrees of freedom but the structure of the mathematical model becomes much simpler. Using this,

the iso-flexible joints between any two rigid bodies are modelled as stiff springs with the same stiffness in all directions (i.e. joints with isometric stiffness). The spring is then modelled as a transmission line element, introducing a time delay in the communication between the connected bodies.

In the original work on TLM in electromagnetics [Johns & Beurle, 1971] the properties being modelled were voltage and current. The equivalent qualities in fluid power simulations are flow and pressure. Force and either speed or displacement each translational or rotational are the respective quantities for simulations of mechanical systems. The availability of a TLM model for mechanical problems also makes a unified approach to the treatment of electromechanical systems possible.

2.3.1 Basic equations for transmission-line modelling of fluid power systems

Real fluid power systems are characterised by a transport delay in the pipeline connecting components in the systems, due to the propagation of waves at the speed of sound through the fluid medium. This transmission delay is invaluable in decoupling component models forming part of a distributed or parallel numerical simulation.

Time-domain transmission-line equations may be derived from the continuity equation and the Navier-Stokes equations. One assumes laminar flow (Re < 2000) in a cylindrical pipe with non-elastic walls where the tangential flow is neglected. For the symmetrical hydraulic pipeline the four-pole equation can be derived (see Appendix A).

$$\begin{pmatrix} -Q_b \\ P_b \end{pmatrix} = \begin{pmatrix} \cosh\left(sT\sqrt{N}\right) & -\dfrac{\sinh\left(sT\sqrt{N}\right)}{Z_c\sqrt{N}} \\ -Z_c\sqrt{N}\sinh\left(sT\sqrt{N}\right) & \cosh\left(sT\sqrt{N}\right) \end{pmatrix} \begin{pmatrix} Q_a \\ P_a \end{pmatrix} \tag{2.1}$$

Flows into the line are defined as positive and the viscous friction factor, $N(s)$, is defined as

$$N(s) = -\frac{J_0\left(j\cdot\sqrt{\frac{s}{\nu}}\cdot r\right)}{J_2\left(j\cdot\sqrt{\frac{s}{\nu}}\cdot r\right)} \tag{2.2}$$

The characteristic impedance is given by

$$Z_c = \frac{\rho\cdot c}{\pi\cdot r^2} = \frac{\rho\cdot c}{A} \tag{2.3}$$

with the sound velocity

$$c = \sqrt{\frac{B}{\rho}} \tag{2.4}$$

For pipe or hose pipe elasticity can be accounted for by using an effective bulk modulus $B=B_e$. Re-arranging equation (2.1) for a lossless transmission line, where $N(s)$ in equation (2.2) is unity and transforming it into the time domain leads to

$$P_a(t) - Z_c Q_a(t) = P_b(t-T) + Z_c Q_b(t-T) = C_b(t-T) \tag{2.5}$$

$$P_b(t) - Z_c Q_b(t) = P_a(t-T) + Z_c Q_a(t-T) = C_a(t-T) \tag{2.6}$$

The right hand sides of equations (2.5) and (2.6) are referred to as characteristic pressures, C, representing delayed information from the opposite pipe end.

2.3.2 Modelling example

To illustrate the solution of the TLM wave equation in the time domain the simple circuit shown in Figure 2.4 is investigated (Pollmeier [1996a]). This system consists of a constant flow source connected to a fluid filled transmission line, which is also connected to a pressure relief valve. The output of the valve is linked to a constant pressure reservoir. The flow source and the inlet port of the pressure relief valve are connected by characteristic pressure waves, which propagate through the line at a finite speed. Hence there is a finite transmission delay that decouples both right and left travelling characteristic pressure waves.

The source flow, Q_1, is defined as positive into the transmission line. The corresponding pressure at the source exit (transmission line end) is given by direct substitution of this flow into the transmission line end (equation 2.5) to obtain the following expression:

$$P_1(t) = C_2(t-T) + Z_c Q_1(t) \tag{2.7}$$

where C_2 is the characteristic pressure wave propagated from the inlet port of the relief valve calculated at the previous time step. Using the flow Q_1 and the pressure P_1 (determined from the transmission line end equation) the flow source characteristic pressure wave is calculated according to the following equation:

$$C_1(t) = P_1(t) + Z_c Q_1(t) \tag{2.8}$$

This characteristic pressure is received by the inlet of the relief valve at the next time step. The simplest model of a relief valve assumes that the valve dynamics are instantaneous.

For this type of model the valve is uni-directional and no flow across the valve takes place until the differential pressure exceeds the cracking pressure, P_c. A linear flow-pressure characteristic of gradient K is assumed once the valve has cracked open. Transmission line end equations at the valve inlet and outlet port are given in equations 2.9 and 2.10.

$$P_2(t) = C_1(t-T) - Z_c Q(t) \tag{2.9}$$

$$P_3(t) = P_3 = const \tag{2.10}$$

The equations relating valve flow to differential pressure are

$$(P_2 - P_3) > P_c \qquad Q(t) = K(P_2(t) - P_3 - P_c) \tag{2.11}$$

$$(P_2 - P_3) \le P_c \qquad Q(t) = 0 \tag{2.12}$$

Flow is then determined explicitly by substitution of the pipe-end (characteristic pressure) equation into the valve equations, as follows:

$$(P_2 - P_3) > P_c \qquad Q(t) = K(C_1(t-T) - Z_c Q(t) - P_3 - P_c) \tag{2.13}$$

$$\therefore Q(t) = -Q_2(t) = Q_3(t) = \frac{K(C_1(t-T) - P_3 - P_c)}{1 + KZ_c} \tag{2.14}$$

$$(P_2 - P_3) \le P_c \qquad Q(t) = Q_2(t) = Q_3(t) = 0 \tag{2.15}$$

These equations take into account that flow is defined positive into the line, or out of the component, i.e. in the opposite direction to the valve inlet port. The valve inlet port pressure and characteristic are then readily determined from the flows. If the applied pressure difference is larger than the cracking pressure this leads to:

$$P_2(t) = C_1(t-T) - Z_c \frac{K(C_1(t-T) - P_3 - P_c)}{1 + KZ_c} \tag{2.16}$$

Equations 2.14, 2.15 and 2.16 are explicit equations for the valve inlet/outlet flow and pressure. The characteristic pressure to be propagated back to the flow source must be computed as follows:

$$C_2 = P_2(t) + Z_c Q_2(t) \tag{2.17}$$

This characteristic pressure is received by the outlet from the flow source at the next time step. Initial conditions for the characteristic pressure are set to the initial line pressure. Thus at simulation start time the characteristic pressures are the same at each end of the

line. It is clear that the numerical calculations relating to both the flow source (equation 2.7) and the relief valve (equations 2.14 to 2.16) can be computed simultaneously.

2.3.3 Compressible line

For many industrial hydraulic applications, compressibility effects dominate the dynamic behaviour of the pipelines and fluid volumes (Tomlinson [1987], Ellman et al. [1993]). The modelling errors introduced by such lumped parameter compressible fluid volume approximations are usually small when compared with the magnitude of errors arising from simplifications made in the modelling of other component models used in the simulation. For example relief valve models as used in the modelling example often employ a simplified linearised flow-pressure characteristic (Palmberg [1991], Ellman et al. [1993]).

In the simulation of a system containing several pipe lines of different length all of the delays in the TLM must be equal to achieve synchronisation. Hence an approximation to a compressible pipeline is obtained by setting the transmission delay, T, to be equal to the integration time-step, Δt. This implies the line length calculated by

$$L = \frac{\Delta t}{c} \tag{2.18}$$

and consequently the pipe diameter must be adjusted to yield the desired volume V, thereby ensuring correct modelling of compressibility effects, such that the capacitive term C remains constant.

$$C = \frac{V}{B_e} \tag{2.19}$$

Where B_e is the effective bulk modulus. If T is changed to match the required time-step, the line impedance, Z, must also be adjusted to maintain the same line capacitance, given by:

$$Z = \frac{\rho c}{A} = \frac{\rho c L}{V} = \frac{\rho c^2 \Delta t}{V} = \frac{\Delta t}{V/B_e} \tag{2.20}$$

with

$$V = LA \qquad L = c\Delta t \qquad c = \sqrt{\frac{B_e}{\rho}} \tag{2.21}$$

This line model introduces a distortion of the fluid inertance I (equation 2.22), due to the adjustment of the pipeline transmission delay, T.

$$I = \frac{\rho L}{A} = \frac{\rho L^2}{V} = \frac{\rho(c\Delta t)^2}{V} = \frac{\Delta t^2}{V/B_e} \tag{2.22}$$

From the latter equation, the parasitic inertia term is proportional to the square of the time step, Δt. Re-writing the transmission line equations using lumped parameters results in:

$$p_b - p_a + I\frac{dq}{dt} + Rq = 0 \tag{2.23}$$

$$q_b - q_a + C\frac{dp}{dt} = 0 \tag{2.24}$$

If the line element is modelled as lossless (R=0) then the line inductance, I, is the only source of pressure change between the line ends. This pressure change is the product of the inertia term and the total rate of change of flow into the line. Hence, rapid flow transients, coupled with high fluid inertance, results in a correspondingly large and distorted pressure change. This pressure difference can be treated as a parasitic pressure error, which indicates how much the line behaviour has diverged from that of an ideal compressible volume. For easy recall this pressure will be called P_{par}, i.e. parasitic pressure. These pressure values can be used as a measure of the simulation accuracy as well as for the control of the time step in a variable time step scheme as described in Chapter 3.

A completely loss-less capacitive transmission-line (equations 2.5 and 2.6) may introduce unrealistic high frequency resonances into the system model, consistent with the undamped propagation of plane waves [Krus et al., 1990]. Laminar and turbulent friction can be incorporated into a friction factor, which reduces the characteristic pressures accordingly. This approach gives correct steady state losses, but ignores frequency-dependent friction. Low-pass filtering of the characteristic pressures approximates the effect of frequency-dependent damping evident in real oil-filled pipelines. The unrealistic resonances can be suppressed by the same low-pass filter. Krus et al. [1990] proposed a filter which is given by the following equations.

$$P_{a,b}(t) - Z'Q_{a,b}(t) = C'_{b,a}(t-T) \tag{2.25}$$

$$C'_{b,a}(t-T) = (1-\alpha)\cdot C_{b,a}(t-T) + \alpha \cdot C'_{b,a}(t-2T) \tag{2.26}$$

$$Z' = \frac{Z}{1-\alpha} \tag{2.27}$$

The filter coefficient, α, is taken to be 0.2 from numerical experiments. The use of this recursive algorithm means that the line impedance, Z, must be corrected to achieve the same capacitance of the line without filtering.

2.3.4 Stability of TLM simulation

TLM is a simple, explicit and unconditionally stable method for the modelling of wave propagation through a medium [Pomeroy, 1991], [Boucher & Kitsios, 1986] as well as for the modelling of passive RLC lumped networks [Johns & Butler, 1983], [Christopoulos, 1995]. In practice, the TLM network itself has a filtering effect [Johns & Beurle, 1971]. Errors in heatflow/diffusion TLM modelling are dissipative rather than cumulative, hence the TLM technique is stable [de Cogan & Enders, 1991]. The same has been found for TLM modelling of fluid power lines. As an example the circuit in Figure 2.5 has been investigated. 15 lines of one metre length are connected to a pump (constant flow source) and a laminar orifice. A pressure error inserted at any node in the system will propagate through the circuit until its effects are smeared out over the entire system.

Well known implicit methods can be realised as transmission line models but the important new feature of the TLM method is that it can result in procedures which are explicit. This means that values at one timestep can be expressed in terms of values at previous steps only, thus avoiding the need to solve simultaneous equations at each iteration. Explicit TLM models do not in general correspond to known explicit routines. The good stability properties of the method are extremely useful in the solution of stiff networks where instability in explicit methods always causes problems. Transmission line modelling also provides considerable knowledge about the errors introduced by the discretisation process. This means that the step size necessary for a certain accuracy of representation of the circuit can be assessed before calculation [Johns & O'Brien, 1980].

There is another advantage to the TLM method which again arises from its simple properties. When a lumped network is solved in the time domain using any numerical method there is an error in the result due to the process of discretisation. This error can be seen immediately when forming the transmission line model. Since subsequent solution of the TLM model is exact, the errors lie in forming the model not in solving it. However, when dealing with lumped networks it may be more useful to describe the modelling error in terms of parasitic lumped components rather than the exact error description given by

the transmission line. The parasitic pressure error is caused by adjusting the line length/diameter according to the used time step. These adjustments distort the system, but every integration will do so and with the TLM method it is clear in what way the system is distorted [Krus et al., 1990].

The example in section 2.3.2 makes it clear that in TLM modelling, errors are due to the modelling process only and not due to the approximate solution of an approximate calculus model. This can also be seen when looking at the block diagram representation of the example circuit in Figure 2.6. The flow Q_2 and the pressure P_2 in equations (2.14) and (2.16), respectively, are calculated exactly according to the model equations. Only numerical rounding errors may occur. Figure 2.6 also indicates the modularity of the TLM approach. All simulation blocks associated with the pump and the valve, respectively, can be combined as shown in Figure 2.7. Each combined function simultaneously calculates the characteristic pressure, C, for one time step and then the functions exchange this values. Then the next time step is calculated, etc. For completeness Figure 2.8 depicts the low-pass characteristic filter according to equation 2.26.

In Hui et al. [1993a,b] it is shown that the numerical solution is still stable even when a time step three times greater than the smallest time constant of the system is used. TLM simulations of fluid power systems have similar properties. Figures 2.9 and 2.11 (detailed in Figures 2.10 and 2.12, respectively) show some simulation results of the modelling example from section 2.3.2 (Figure 2.4) using the filter described in equations (2.25) to (2.27). The cracking pressure P_c of the instantaneously opening relief valve is set to 120 bar and the flow-pressure characteristic gradient K is assumed to be 600 l/min/bar. All simulation parameters are given in Table 2.1. The pump (constant flow source model) supplies the system with a constant flow of 10 l/min. The pressure relief valve opens when the pressure difference becomes larger than the cracking pressure. This happens after about 0.1 s simulation time.

Flow into the transmission line is defined as positive, thus the figures show negative flows. As expected, the simulation with the smallest time step ($\Delta t=10\mu s$) leads to the highest accuracy. An increase in the time step leads to an increased overshoot and decaying oscillations, but the simulation never becomes unstable. Here the TLM simulation can cope with severe nonlinearities without stability problems.

2.3.5 TLM compared to other numerical techniques

The transmission line matrix method can be considered as a discrete form of Huygen's principle [Tan & Fusco, 1993] which is a localised recursive definition of electromagnetic wave propagation in the time domain [So et al., 1995]. TLM can also be seen as a physical model of mathematical finite differencing [Vetri & Simons, 1993]. In the latter paper the authors also derive a class of TLM-type algorithms directly from Maxwell's equations.

A paper from Johns & O'Brien [1980] describes a new way of viewing the numerical solution of lumped electrical networks. In the special case of the RC networks associated with the diffusion equation it has been possible to give the transmission-line model corresponding to the state equations integrated by the explicit forward Euler method. In this application it appears that the TLM algorithm is a general one which includes the forward Euler method as a particular case. Furthermore the solution of an RC network by the TLM method is the same as solving by the trapezoidal algorithm.

Krus et al. [1990] interpret the TLM method as a general method for integration. They also found the TLM corresponds to the trapezoidal rule of integration, but with double the time step. Details on the TLM method as a method of integration are explained more fully in Chapter 6 and Appendix B. Partridge et al. [1987] investigate TLM modelling of shaft systems dynamics. Again solutions turn out to be equivalent to forming the state equation for the lumped model and integrating by the trapezoidal rule.

2.4 Component modelling

In general a component model is implemented by solving simultaneously the component equations (system equations) and the transmission line end equations (equations 2.5 and 2.6) at each component port. Mathematically this is represented by the following equations:

$$\mathbf{Q} = f(\mathbf{P}) \tag{2.28}$$

$$\mathbf{P}(t) - \mathbf{Z}\mathbf{Q}(t) = \mathbf{C}(t - \Delta t) \tag{2.29}$$

Where $\mathbf{Q}$ and $\mathbf{P}$ are vectors with all the flows and pressures in all the nodes connected to the component. $\mathbf{Z}$ is a diagonal matrix with the characteristic impedances in the diagonal and $\mathbf{C}$ is a vector containing the characteristics. With the approach taken here each transmission line end is incorporated into a separate component model, i.e. there are no separate line models as in lumped parameter simulations. Each component model is decoupled in time and is thus numerically isolated from its connecting component models

by the finite delay introduced by the transmission lines. For components that involve differential equations, suitable time-difference forms can be obtained by numerical integration; in effect this transforms the ODE into an algebraic equation, which is then solved simultaneously with the transmission line ends. A Bi-linear transformation, equivalent to trapezoidal integration can be used to solve equations of motion for the mechanical parts of models. This is adequate for most simulations, because when compared with the fluid transients the dynamic motion of, for example, actuators is often considerable slower (Burton [1994]).

It is possible to construct a pseudo-dynamic model, by incorporating a typical first/second order response into the instantaneous dynamic model [Viersma, 1980]. Values for time constants to approximate the dynamics may then be estimated. In this research, instantaneous and pseudo-dynamic models are used when possible in order to simplify the system simulations undertaken. This does not exclude the future use of other, more detailed, component model types. The main exceptions are linear and rotary hydraulic actuators (these include motors), where movement is modelled by numerical integration of the equations of motion within the component models.

It is not the intention to present details of all the new models developed by the author. A comprehensive description of some models developed previously is given in Burton et al. [1992], [1993a,b], Burton [1994]. Several of these models were corrected and improved but due to space limitations the details cannot be presented for all of these models. The following sections illustrate some aspects of the approach to modelling particular hydraulic components and features. These are models and modelling techniques not described previously. A directional control valve model is described to illustrate superposition of flows using standard orifice equations. Furthermore, it indicates how the spool movement can be modelled with a pseudo-dynamic model for a second order response. A more exact model of the spool dynamics is used with the investigated pressure relief valve model. It becomes impossible to derive explicit algebraic equations when spool valve dynamics and square-law orifice equations are to be solved simultaneously. This can be overcome by linearisation of the square-law equation around the operating point. A way of simplifying the modelling of internal leakage is shown for a differential area actuator. This leads to a good approximation of the system behaviour and makes the coding easier. Furthermore, it leads to faster models. A new filter model is also derived and finally the simplified modelling of external leakage in pumps and motors is described.

2.4.1 Directional control valves

This section describes a general purpose model of a four port, three position closed centre electrically-modulated directional control valve, i.e. servo or proportional valve. The model includes valve dynamics and takes into account underlap as well as internal leakage. Leakage is assumed to vary linearly with pressure differential. Figure 2.13 depicts a schematic of such a valve, where the underlap, u, is assumed to be symmetrical, i.e. in centre position u is assumed to be the same for all four circled edges. The characteristics of a particular valve are defined using both steady state and frequency response performance data. Compressibility flow loss effects are not included.

The valve has two gain regions; a central null region where the effective flow gain is modified to account for valve underlap. Proportional control of the valve is incorporated by assuming a linear variation in orifice coefficient, equivalent to a linear change in annular flow area across the valve spool. With the notation in Figure 2.13 and using superposition (see Appendix D1) the flows can be derived for spool displacements smaller than the underlap:

$$Q_s = -\left(k_{sa}\sqrt{P_s - P_a} + k_{sb}\sqrt{P_s - P_b} - Q_{1stleak}\right) \tag{2.30}$$

$$Q_r = k_{ar}\sqrt{P_a - P_r} + k_{br}\sqrt{P_b - P_r} + Q_{1stleak} \tag{2.31}$$

$$Q_a = k_{sa}\sqrt{P_s - P_a} - k_{ar}\sqrt{P_a - P_r} \tag{2.32}$$

$$Q_b = k_{sb}\sqrt{P_s - P_b} - k_{br}\sqrt{P_b - P_r} \tag{2.33}$$

where k_{ij} describes the flow coefficient from port i to j. These coefficients are proportional to the spool displacement and they also take underlap into account. In order to generalise the equations some of the coefficients are set to zero when the spool displacement becomes larger than the underlap. The sum of all four flows equals zero according to the flow balance. First stage leakage is calculated with the user supplied leakage flow coefficient k_l.

$$Q_{1stleak} = k_l\left(P_s - P_{a,b}\right) \tag{2.34}$$

Valve spool position is assumed to respond as a critically damped second-order system driven by the valve drive signal I.

$$\ddot{x} = \omega_n^2(I - x) - 2\omega_n\dot{x} \tag{2.35}$$

where x is the spool fractional displacement and I is calculated by the following equation

$$I = \frac{valve\ current}{rated\ current} \tag{2.36}$$

The natural frequency, ω_n, is set by the user. Spool motion is assumed to be unaffected by flow, pressure or stiction forces. Fluid compressibility and temperature changes are not modelled. All features of the valve are assumed time invariant. The model may be used as a conventional 3 way, 4 port electrically operated modulating servo or proportional valve. It is intended for use with manufacturer's data. The user specifies the characteristics of a particular servo valve with steady state flow rate/pressure drop information, null region details and transient performance (frequency response). The valve null region modelling is very simple, hence, it is unlikely to be suitable for simulations where the null region characteristics are crucially important. With the trapezoidal rule of integration for the spool velocity and displacement the algebraic system equations can be derived as shown in Appendix D1.

2.4.2 Pressure relief valve model

This section describes the model of a pressure relief valve. The most significant dimensional parameters are assumed to be available. These are spool diameter, spring stiffness and the spool stroke. If the inlet pressure exceeds the set cracking pressure, flow is let through to the outlet port. Fluid compressibility effects are not accounted for. The spool dynamics are described by Newton's second law. With the notation in Figure 2.14 this leads to

$$M\ddot{x} = P_1A_1 - P_2A_1 - P_cA_1 - C_v\dot{x} - K_sx \tag{2.37}$$

where C_v describes the velocity proportional friction, M is the spool mass and K_s is the spring stiffness. The flow rate to the outlet port is assumed to follow the square-law orifice model, i.e.

$$Q = k\sqrt{P_1 - P_2} \tag{2.38}$$

where k is linearly dependent on the spool position. In order to achieve a set of solvable equations the square-law equation needs to be linearised around the operating point n. This leads to equation 2.39 where the flow is only linearly dependent on values at time n+1.

$$Q_{n+1} = Q_n + k\sqrt{P_{1,n} - P_{2,n}} \cdot (x_{n+1} - x_n) + \\ + k\frac{x_n}{2\sqrt{P_{1,n} - P_{2,n}}} \cdot (P_{1,n+1} - P_{2,n+1} - P_{1,n} + P_{2,n}) \tag{2.39}$$

Explicit equations for displacement, flows and pressures can be derived from equations 2.37, 2.39 and the transmission line end equations as detailed in Appendix D2. The derived model is suitable for general use as pressure reducing or relief valves where the spool dynamics are of importance.

2.4.3 Differential area actuator model

Burton [1994] derived a dynamic model of an unequal area linear hydraulic actuator. External load forces are supplied to the actuator model from a separate load model. Friction forces are represented as constant stiction, Coulomb and a velocity proportional term, no provision is made for position or time dependency of any of these terms. Fluid compressibility is modelled in each chamber by application of the TLM line equations. Seal wear and temperature effects are not taken into account. The model has been improved by modelling leakage across the piston which is assumed to be laminar and is represented by a constant pressure/flow gradient. Figure 2.15 shows the simplified actuator indicating the cross-port leakage flow. Including this leakage when solving the transmission-line, flow and relevant equation of motion leads to very complicated formulas. Hence, a different approach has been developed where the system behaviour is approximated with simpler equations. The actuator displacement, velocity and flows are calculated as described by Burton [1994]. Before the pressures are calculated the leakage flow is derived and added/subtracted to/from the respective flows. Assuming normal extension of the actuator, i.e. $P_1 > P_2$, the new pressures are calculated with the corrected flows using the following equations.

$$P_1 = C_i - Z_1(Q_1 - Q_{leak}) \tag{2.40}$$

$$P_2 = C_j - Z_2(Q_2 + Q_{leak}) \tag{2.41}$$

In these equations i and j are chosen according to the respective connected port. The leakage flow changes direction for $P_2 > P_1$. This simplified approach not only makes it easier to derive the model equations it also leads to faster component models.

The modified actuator model covers a wide range of practical linear actuator applications. Stiction, Coulomb friction, leakage terms and piston rod diameter may all be set to zero if

desired resulting in the simplest of actuator models. Nonzero values represent a more realistic linear actuator, assuming of course that suitable values are chosen. Note that if all the friction terms (velocity dependent term as well) are set to zero the model will have zero damping and resonate continuously, i.e. there should always be some friction.

2.4.4 Filter model

This section describes a hydraulic filter pressure loss model. The pressure loss over the filter element is assumed to vary linearly with flow rate and viscosity according to user defined data. The pressure/flow rate characteristic is based on experimental data [Pall, 1985]. Variations in local viscosity with local operating pressure are modelled, in fact this is taken as the only influence on changes in local viscosity. The effects of density variations, fluid compressibility and filter ageing are not modelled. No bypass valve is modelled. With the notation in Figure 2.16 and assuming flow from port one to two, the viscosity at mean system pressure is calculated using pressure values from previous time steps, i.e.

$$\nu_{actual} = \nu_0 10^{f_v(P_1-P_2)/2} \tag{2.42}$$

and the outlet flow is determined as

$$Q = -Q_1 = Q_2 = (P_1 - P_2)\frac{Q_{spec}}{P_{spec}}\frac{\nu_{test}}{\nu_{actual}}\frac{\rho_{test}}{\rho_{actual}} \tag{2.43}$$

[Richards, 1993]. The parameter f_v is the viscosity pressure dependent factor and Q_{spec} and P_{spec} are the specified flow rate and corresponding pressure drop across the filter element, respectively. Test values for the fluid viscosity and density are also supplied by the user. In equation 2.43 the factors behind the pressure difference can be combined to one constant, i.e.

$$Q = -Q_1 = Q_2 = (P_1 - P_2)k_f \tag{2.44}$$

This is an equation equivalent to the one used for a laminar orifice. The TLM equations can then be derived as shown in Appendix D1 for the directional control valve (equation D13). The model provides a typical filter flow rate/pressure relationship. It may be used wherever it is not necessary to accurately model rapid pressure transients. A filter housing pressure drop, that is assumed to behave as a fixed hydraulic orifice, can be implemented if better accuracy is required.

2.4.5 Positive displacement pumps and motors

A similar approach, like the one taken with the linear actuator model in section 2.4.3 can also be applied to the modelling of external leakage in positive displacement machines. The TLM models for these components have been derived in Burton [1994] where linearised loss coefficients according to McCandish & Dorey [1983] were considered. Figure 2.17 shows the schematics of a variable displacement pump and a variable displacement motor including inertial load. These models have been extended by the modelling of external leakage flow which is assumed to be laminar and proportional to the constant pressure/flow gradient C_{sd}, i.e. the leakage terms are calculated by

$$Q_{leak,1} = C_{sd}\left(P_1 - P_{tank}\right) \tag{2.45}$$

$$Q_{leak,2} = C_{sd}\left(P_2 - P_{tank}\right) \tag{2.46}$$

Instead of re-deriving the formulas for the flows Q_1 and Q_2, the previously derived equations are calculated. The leakage flow is then superimposed, i.e. the new flows are calculated with

$$Q'_{1,2} = Q_{1,2} - Q_{leak} \tag{2.47}$$

All line pressures are then calculated using the superimposed flow. Again, this approach leads to simpler equations and faster component models.

2.5 Closure

In this Chapter the development of the transmission line modelling method is reviewed. The basic equations are derived and the approach is compared with other numerical techniques. TLM is a very powerful approach for domain decomposition and hence parallel operation, as all models are self-contained and are inherently decoupled by a transmission delay. Both the computation of component models and the integration process may therefore be fully distributed. The method is simple, explicit and unconditionally stable. Furthermore, it generally enables larger time steps which lead to fast simulations. The wider range of applications was indicated, i.e. many of the methods and features developed in this thesis may be transferred to other fields. Some models are described in detail using superposition and pseudo-dynamic models to approximate real system behaviour. Using linearisation the modelling process was simplified and faster models enabling leakage modelling were developed. The widely-spread time constants in

hydraulic systems led to the development of variable time step simulations. This approach and the problems associated with it are investigated in the following chapter.

TABLES FOR CHAPTER 2

Parameter	Value
line length	3 m
line diameter	0.025 m
fluid density	890 kg/m2
kin. viscosity	60*10-6 m2/s
print interval	0.01 s
tank pressure	0 bar
cracking pressure	120 bar
Q-P gradient	600 l/min/bar
source flow	10 l/min
bulk modulus	8900 bar

Table 2.1 Simulation parameters

FIGURES FOR CHAPTER 2

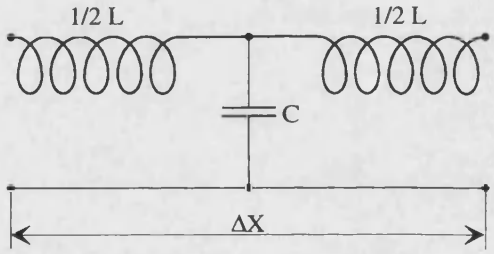


Figure 2.1 Equivalent network of a one-dimensional transmission-line junction

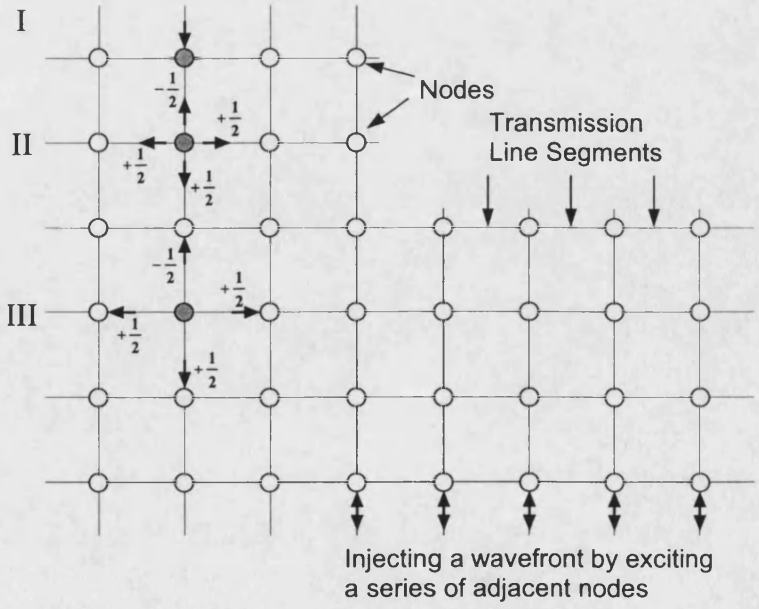


Figure 2.2 A section of a two-dimensional TLM mesh demonstrating various aspects of the modelling process

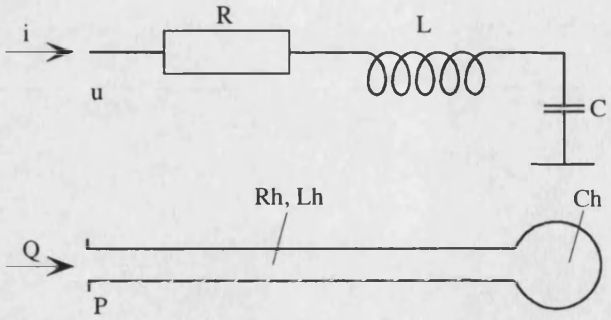


Figure 2.3 Analogy between electrical and hydraulic systems (example)

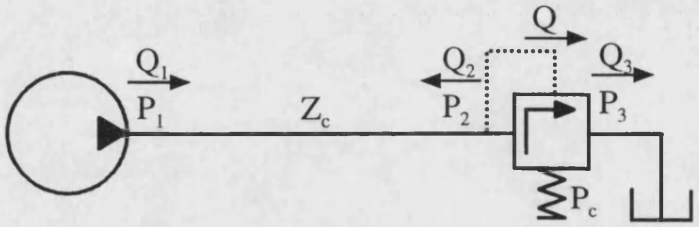


Figure 2.4 Simple hydraulic circuit for the modelling example

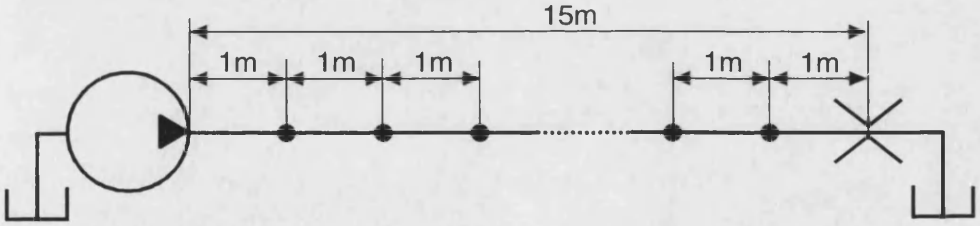


Figure 2.5 Circuit with several connected lines

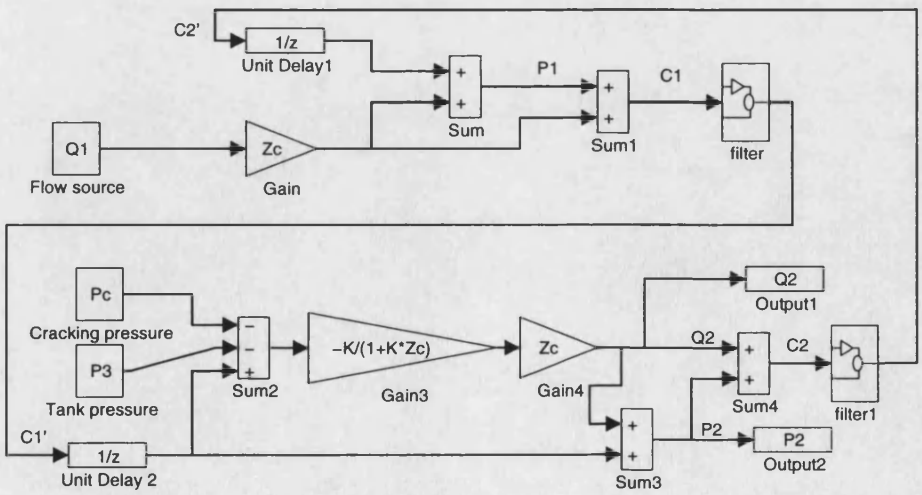


Figure 2.6 Block diagram for the simulation of the circuit in Figure 2.4

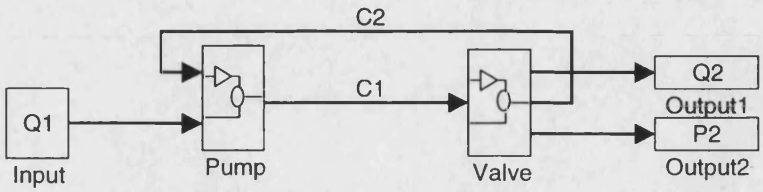


Figure 2.7 Modular block diagram of Figure 2.6

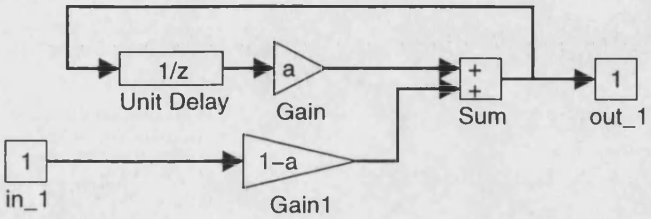


Figure 2.8 Block diagram of filter

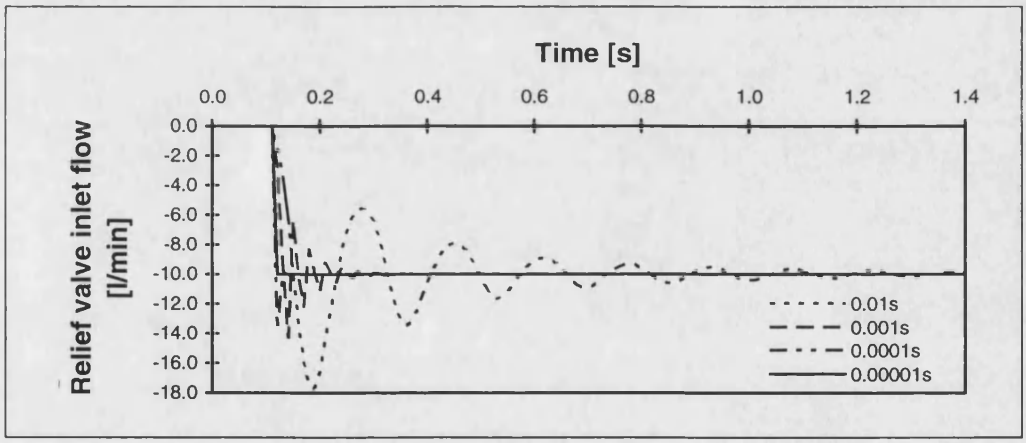


Figure 2.9 Simulation results: relief valve inlet flow

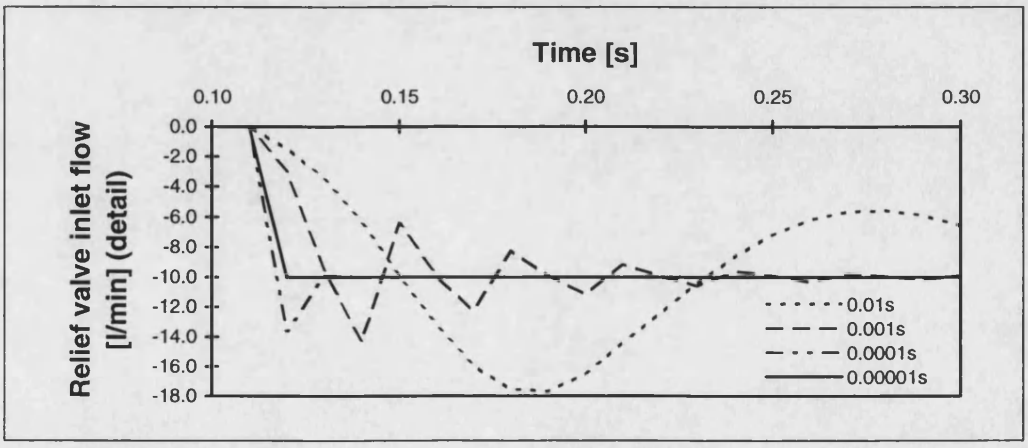


Figure 2.10 Simulation results: relief valve inlet flow (detail)

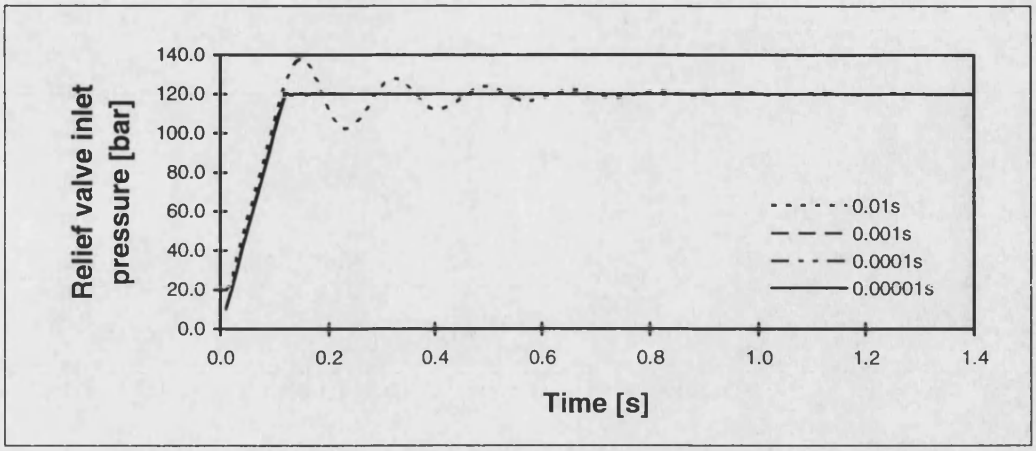


Figure 2.11 Simulation results: relief valve inlet pressure

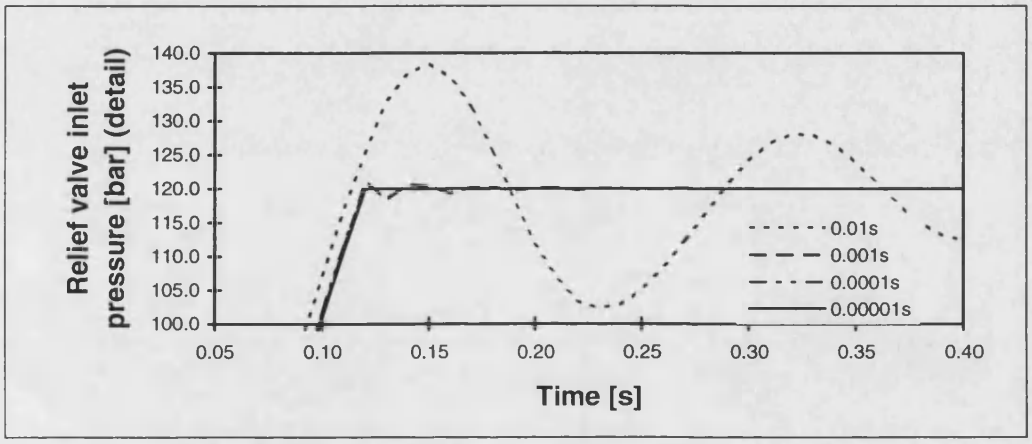


Figure 2.12 Simulation results: relief valve inlet pressure (detail)

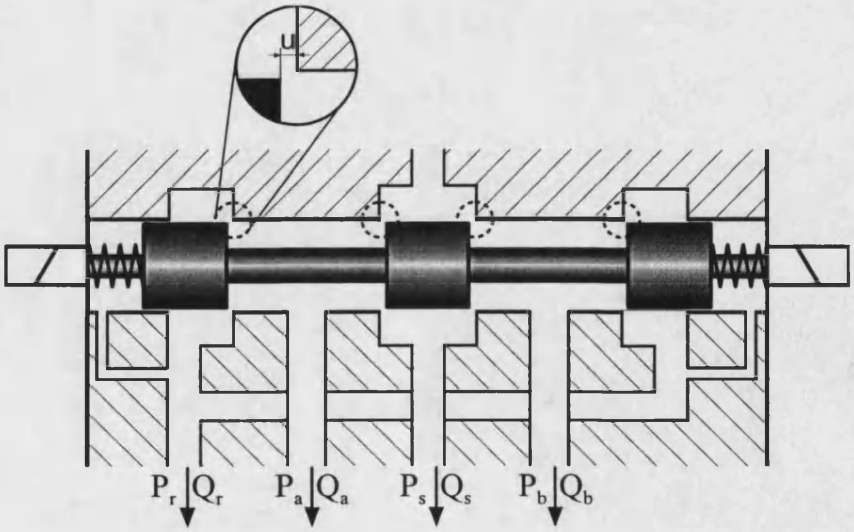


Figure 2.13 Schematic of directional control valve

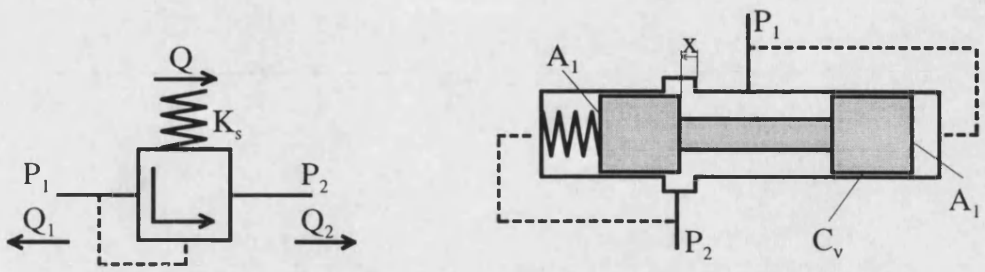


Figure 2.14 Symbol and TLM schematic of pressure relief valve

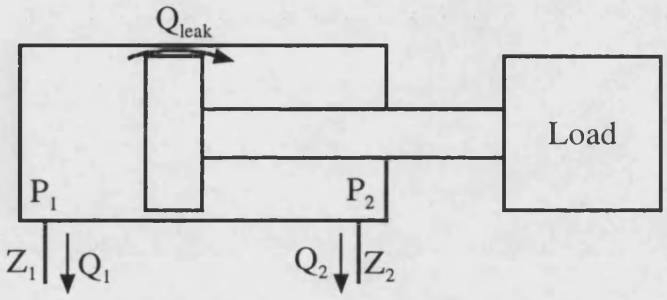


Figure 2.15 Simplified TLM schematic of differential area actuator

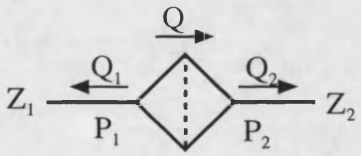


Figure 2.16 TLM Filter symbol

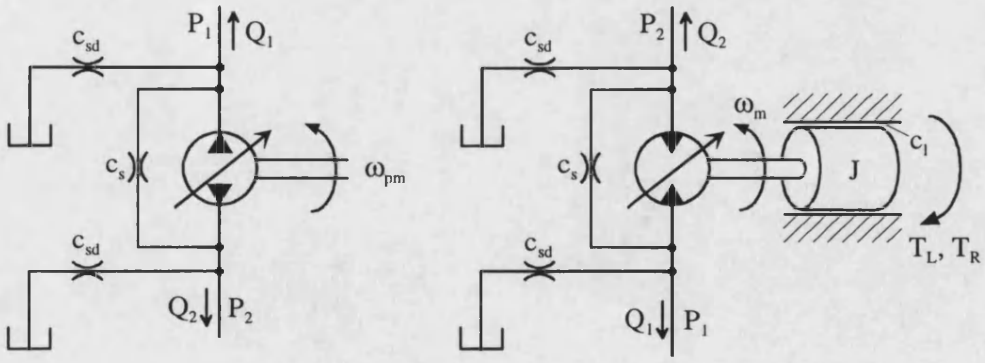


Figure 2.17 TLM schematic of positive displacement pump and motor (including load)

3 Simulation with variable time steps

3.1 Introduction

One problem encountered in modelling some circuits is their widely-spread time constants. Normally, the size of the time step is restricted by the smallest time constant of the system. If a system has widely-spread time constants (i.e. numerically stiff hydraulic systems), constant time step operation of the numerical method can become inefficient. Therefore variable time step TLM simulations have been derived to increase the computing speed even further. This chapter recounts variable time step methods derived for the simulation of thermal diffusion problems. A similar approach is then taken for hydraulic circuit simulation in order to improve computing efficiency. Problems associated with variable time step simulation are investigated and the simulation results of a stiff example system are compared with fixed time step simulations.

3.2 Variable time step TLM

Throughout the course of most fluid power system modelling situations, as the rate of flow change alters, the time step required for a pre-set overall accuracy will vary. It is, therefore, desirable to be able to alter the iteration timestep during modelling, so that the accuracy requirements of the problem may be met with the minimum of computing resource.

Webb [1991] deals with variable time step TLM modelling of thermal diffusion problems and includes a facility to make the mesh coarser at less interesting points of the structure. Hence, the time step was allowed to increase there and shorter time steps are only used in layers where the temperature gradients in time and space are greatest. A similar approach can be adapted for hydraulic systems where long lines need to be simulated with high accuracy. The long lines can be split into several shorter lines leading to the method of characteristics (MOC). This enables a more detailed simulation of the particular line dynamics. Webb [1991] also found that the maximum timestep that could be used at the start of a typical transient was of the same order as that permitted by the stability requirement for an explicit finite difference approach. In Hui et al. [1993a] the variable time step method is extended to TLM short circuited and open circuited stubs. A unified approach for general stubs has been developed by Hui et al. [1993b].

Fundamental to any automatic time-stepping procedure is a rapidly implemented method of some kind of error estimation. Two approaches have been suggested by Pulko et al. [1990] being termed the difference method and the potential drop method. In their paper a two-dimensional thermal diffusion problem described by equation 3.1 is studied.

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=\frac{S\rho}{K}\frac{\partial T}{\partial t} \tag{3.1}$$

This was simulated by using the telegrapher's equation which, in two dimensions, is given in equation (3.2), i.e. the diffusion problem is solved using electrical circuit equivalents.

$$\frac{\partial^2 \Phi}{\partial x^2}+\frac{\partial^2 \Phi}{\partial y^2}=4RC\frac{\partial \Phi}{\partial t}+2LC\frac{\partial^2 \Phi}{\partial t^2} \tag{3.2}$$

The difference method ensures that the ratio of the error term (final term in equation (3.2)) is kept to a small fraction of the wanted term, i.e.

$$m=\frac{2LC\dfrac{\partial^2 \Phi}{\partial t^2}}{4RC\dfrac{\partial \Phi}{\partial t}}<<1 \tag{3.3}$$

This formula can be extended to an explicit form [Pulko et al., 1990]. In the potential drop method the voltage across the transmission lines is calculated and kept below a certain value. This potential drop method has been adopted and successfully applied to hydraulic systems by Jansson et al. [1992] and Burton [1994] as described in the following section.

3.2.1 How to control the variable time step

In order to incorporate automatic timestepping into a TLM model, bounds have to be placed on the values that an error parameter may take. This can be done by performing complete 'dummy runs' with arbitrarily short timesteps and monitoring the values of error parameters. The acceptable values determined in this way may then be used in simulations where externally applied transients and material or system properties are known. However, for problems which are to be modelled over a relatively long period, it is probably preferable to model for a certain simulation time using a very short timestep and to use the corresponding maximum error parameter value as a limit for the remainder of the modelling period. These methods were described by Pulko et al. [1990] and for the latter, the timestep was changed ± 10 per cent for any departure of the error parameter from this 'set-point' by ± 10 per cent. If rapidly changing dynamics are encountered these

approaches will lead to problems when simulating hydraulic systems due to the lack of restart capabilities.

Jansson et al. [1992] describe a more sophisticated step size controller for hydraulic system simulations. Here the parasitic pressure difference P_{par} described in section 2.3.3 is used to control the step size. At first sight it would seem that an estimate of pressure error at each iteration can be calculated quite simply by subtracting $P_a(t)$ from $P_b(t)$. Jansson et al. [1993] compares estimated errors and actual errors for a loss-less pipeline with blocked outlet. For this case the estimated error turns out to be slightly larger that the exact one. In practice it is advantageous to use the average pressure difference from two time steps since this will reduce the influence from the high frequency oscillations of the pressure in neighbouring nodes in a volume. This method was suggested by Pulko et al. [1990]. The estimated difference P_{par} is thus calculated as

$$P_{par}(t) = \frac{P_b(t) - P_a(t) + P_b(t-T) - P_a(t-T)}{2} \tag{3.4}$$

This pressure difference is then used for the step size controller. In diffusion or electrical wave phenomena simulation the mesh of nodes is formed regularly and only one node has to be monitored. Whereas in hydraulic systems the maximum difference can only be obtained by looking at the pressure errors for all used models or lines. The problem is to choose the maximum allowable pressure difference which is to be used for adjusting the time step in the simulations. A good value will depend on the step size controller used. Jansson et al. [1992] used a step size controller which is split into two separate parts. Equation (3.5) is used if P_{par} is reasonably small.

$$\Delta t(t+T) = \left(\frac{P_{ref}}{P_{par}(t)}\right)^{K_I} \left(\frac{P_{par}(t-T)}{P_{par}(t)}\right)^{K_P} \Delta t(t) \tag{3.5}$$

where $P_{par}(t)$ is the parasitic difference in the current time step and P_{ref} is the maximum allowed pressure difference (reference pressure error). A large error is handled by rejecting the result at the current time step and selecting a much reduced step size according to

$$\Delta t(t+T) = \left(\frac{P_{ref}}{P_{par}(t)}\right)^{\frac{1}{K}} \Delta t(t) \tag{3.6}$$

The constants for both control laws are $K_P = 0.001$, $K_I = 0.004$ and $K = 1.5$ [Jansson et al., 1992]. In section 3.3.1 a simulation example is presented to investigate different allowable pressure errors with the step size controller described above.

3.2.2 Modifications necessary for variable timestep TLM

The basic modification that must be made to the integration to allow a varying timestep is to compensate the characteristic for varying Z_c values (equation 2.3) that will be the result of varying time steps. Since timestep changes are implemented as the pressure waves propagate along the transmission-line, these characteristic pressures must be adjusted to maintain "total pressure" and "total flow" in the line [Pulko et al., 1990], [Jansson et al., 1992]. The total pressure is constant if the sum of the characteristic pressures at each line end is the same both before and after the change in line impedance Z and Z'. For total pressure:

$$C_a + C_b = C'_a + C'_b \tag{3.7}$$

Total flow through the line is conserved if the difference in characteristic pressures divided by line impedance remain unchanged. For total flow:

$$\frac{C_a + C_b}{Z} = \frac{C'_a + C'_b}{Z'} \tag{3.8}$$

To meet both constraints the "compensated" characteristic pressures become [Burton, 1994]:

$$C'_a = C_a\left(\frac{1+\gamma}{2}\right) + C_b\left(\frac{1-\gamma}{2}\right) \tag{3.9}$$

$$C'_b = C_b\left(\frac{1+\gamma}{2}\right) + C_a\left(\frac{1-\gamma}{2}\right) \tag{3.10}$$

with

$$\gamma = \frac{Z'}{Z} \tag{3.11}$$

3.2.3 Variable line volume and bulk modulus

Many hydraulic components (e.g. actuators and accumulators) will change the working volume of fluid in a transmission-line. In addition, variations in effective bulk modulus with pressure, for example, will have an effect on system performance. Using TLM the

basic approach to account for changes in volume or bulk modulus is to modify the line characteristic impedance, Z, according to equation (2.20). Hence the same modifications applied for the variable time step TLM (section 3.2.2) may be used for simulations with variable line volumes and changing bulk modulus.

3.3 Comparison between fixed and variable time step simulations

3.3.1 Simulation Example: Two-actuator Circuit

The two-actuator circuit, shown in Figure 3.1, is used as an example with which to demonstrate the application of the variable time step TLM algorithm. Briefly, the purpose of this hydraulic system is to use the coupled motors to divide the flow equally between each of the two actuators, both during extension and retraction. The two pumps initially operate together, supplying maximum flow to the actuators simultaneously. However, once a pre-defined pressure is reached the large capacity pump is unloaded, via the pilot operated relief valve. The smaller pump continues to supply flow independently at the higher pressure until either the directional control valve position is altered, or the high pressure relief is activated. This arrangement is intended to limit the maximum power required from the prime mover. A potential drawback of this circuit design is the inability of the flow-divider to supply precisely equal flows. Different leakages in the separate gear motors, which constitute the flow-divider, result in different supplied flows. It is the function of the relief valves connected to the actuator piston ends to facilitate re-synchronisation of position, should one actuator stop before the other [Burton, 1994].

Figures 3.2 to 3.4 show a selection of computed results obtained from the transmission-line simulations of the two-actuator circuit. The maximum allowed pressure differences (P_{ref}) of 0.05 bar and 0.1 bar and a 1 μs minimum timestep were used.

Figure 3.2 shows the variation in the simulated actuator displacement with time, for both pressure differences considered. There is a change in velocity gradient (after about 1.45 seconds) as a result of the flow redirected to the reservoir from the primary pump, when the pilot operated relief (unloading) valve is opened. The other change in gradient occurs when the actuator reaches the spring after about 0.75 seconds (1.7 m displacement). Only at this point do the different reference pressures lead to very small differences in the simulation results.

Figure 3.3 illustrates the corresponding velocity transients of actuator 1. After initial start-up transients the actuator reaches the steady state velocity of about 0.138 m/s. As the actuator contacts the spring (after 0.75 seconds) a large discontinuity is visible. The simulation results computed with the larger reference pressure (0.1 bar) show an undershoot to the next steady state velocity. Here the deviation in pressure difference leads to large differences in simulation results, but the steady state value is always reached.

Between 1.3 and 1.4 seconds the velocity computed with $P_{ref} = 0.05\text{bar}$ shows some transients, which are probably caused by the time step controller. This will be described later.

Figure 3.4 shows the flow-divider shaft speed plotted against time. The same is shown on a larger scale in Figure 3.5. Differences between both simulations ($P_{ref} = 0.05\text{bar}$, $P_{ref} = 0.1\text{bar}$) are again visible in the time interval 0.75 to 1.0 seconds.

Figures 3.6 and 3.7 reproduces the magnitude of the time step and the corresponding maximum parasitic pressure difference, respectively, plotted against time. The pressure differences are calculated for all lines according to equation 3.4 and then the largest values at every time step are shown. Figure 3.8 again shows the time step plotted on a larger scale. The time step of the simulation computed with $P_{ref} = 0.05\text{bar}$ is generally smaller than the respective values of the simulation with $P_{ref} = 0.1\text{bar}$. Several times between 1.35 and 1.65 seconds the time step controller causes the computation to use the minimum time step which was set to 1 µs for both calculations (see Figure 3.8).

If the error controller requires a time step less than the minimum allowable of 1 µs then this will cause spikes in the maximum parasitic pressure difference transient. This explains the parasitic pressure spikes above 0.1bar and 0.5bar between 1.3 and 1.7 seconds (Figure 3.7). An unexplained situation caused by the time step controller is found in the time period between 2.2 and 2.4 seconds (Figure 3.7). The smaller time step used by the simulation with $P_{ref} = 0.05\text{bar}$ results in larger pressure differences.

The largest differences in simulation of the velocity of actuator 1 (see also Figure 3.3) are detected between 0.75 and 0.9 seconds. At the beginning of this time period the simulation with $P_{ref} = 0.05\text{bar}$ uses a much smaller time step. This time step also changes to a smaller value much more quickly when the discontinuity (described previously) occurs. In

this case the time step for the simulation with $P_{ref} = 0.1\text{bar}$ should decrease much faster in order to obtain better results; i.e. the time step controller does not decrease the time step fast enough for the simulation with $P_{ref} = 0.1\text{bar}$.

The same system was simulated using the fixed time step approach. In Figure 3.9 the maximum parasitic pressure differences are shown for time steps of 100 μs and 10 μs, respectively. These parasitic pressures are smaller then the one obtained when using the variable time step approach (Figure 3.7), i.e. the changing time step leads to higher pressure errors and lower simulation accuracy.

3.3.2 Further problems associated with variable time step TLM

Due to the variable time step controller, unrealistic oscillations can appear in the simulation results. Inconsistently, larger parasitic pressure spikes can occur with smaller maximum allowed pressure difference. Furthermore the switch between the two different controllers represents a discontinuity and this may present problems. High pressure error spikes corresponding to low simulation accuracy will also occur if the time step controller requires a time step smaller than the minimum possible time step. This is only expected at a few points in the simulation but some measure is needed to detect this case. Here the pressure error in itself can be used, with a flag indicating that a very small time step is required.

Implementing the variable time step scheme on to a parallel platform leads to another problem. Changing the step length in one subsystem disturbs the simulation of other subsystems, as described in Jansson et al. [1992]. The synchronisation of the variable time steps on a parallel computer leads to an extra communication overhead. In addition to the characteristic pressures and either the pressure or the flow values, the pressure errors also have to be exchanged. On one of the processors (called master processor) the new time step can then be calculated, but this information has to be propagated to all processors once again increasing the communication between processors. Furthermore, for every system the allowable pressure differences have to be chosen and different systems may require different pressure differences in order to achieve the same accuracy.

Most real-time simulations consist of lumped-parameter systems described by ordinary differential equations. They may either be interfaced with actual physical systems (hardware-in-the-loop) or with human operators (man-in-the-loop). Common to both

applications is the need for the computer system to process the information within certain time limits. The overall speed of the system must be high enough to meet the real-time bandwidth requirements of the problem. Besides the problems found by using variable time steps there are other reasons to simulate with fixed time steps. If real-time simulation has to be achieved then the minimum time step cannot be smaller than a certain limit. Even if the worst case occurs, i.e. simulation with the smallest time step at all times, the computation has to be fast enough to give results in real time. Hence, the mathematical step size must not become smaller than the computer execution time for the calculations.

Another reason for using a fixed time step in real-time simulation is the compatibility with fixed sample rates when dealing with real-time inputs and outputs, e.g. if the simulation is used for on-line condition monitoring. In other words, if a dense output is required, variable time step simulation can be inefficient. Howe [1993], [1994a,b] and Deiss et al. [1994] describe some methods to synchronise asynchronous multi-processor simulations by using interpolation. These methods enable synchronisation of integrations with different variable-time steps on different processors. They depend on the integrators used and cannot easily be used with the TLM. The TLM trapezoidal-rule-like integrator uses previous values which are filtered and these filters are difficult to include in the method described by Howe and Deiss. Furthermore, only small changes in the time step can be accounted for.

Hence, in this thesis, where the main concern is condition monitoring, from now on only fixed time step simulations will be examined in detail. Independent of the duty cycle this always leads to the same runtime. By using the fixed time step TLM algorithm the step size has to be chosen carefully. It must be small enough in order to obtain sufficiently accurate results, but a very small time step could lead to unacceptably long computer execution times. If the time step is restricted to a minimum value then the parasitic pressure can be used as a measure of the simulation accuracy. A new method enabling different time steps for different parts of the circuit will be described in Chapter 6.

3.4 Closure

In this chapter the variable time step TLM simulation of fluid power systems was investigated. Similar to the potential drop method developed for thermal difference problems a parasitic pressure difference can be used to adjust the step size. The approach seems to be suitable for numerically stiff hydraulic circuit simulations but it leads to

several problems. Unrealistic oscillations and parasitic pressure differences can appear in simulation results as shown with a numerically stiff example circuit. Implementing the scheme on to a parallel platform can lead to synchronisation problems and it also increases the communication overhead.

Concentrating on real-time performance from now on only fixed time step simulations will be investigated. In order to speed up these simulations parallel processing can be used. This is investigated in the following chapters. First, the process of automatic circuit model generation using modular components is explained in Chapter 4.

FIGURES FOR CHAPTER 3

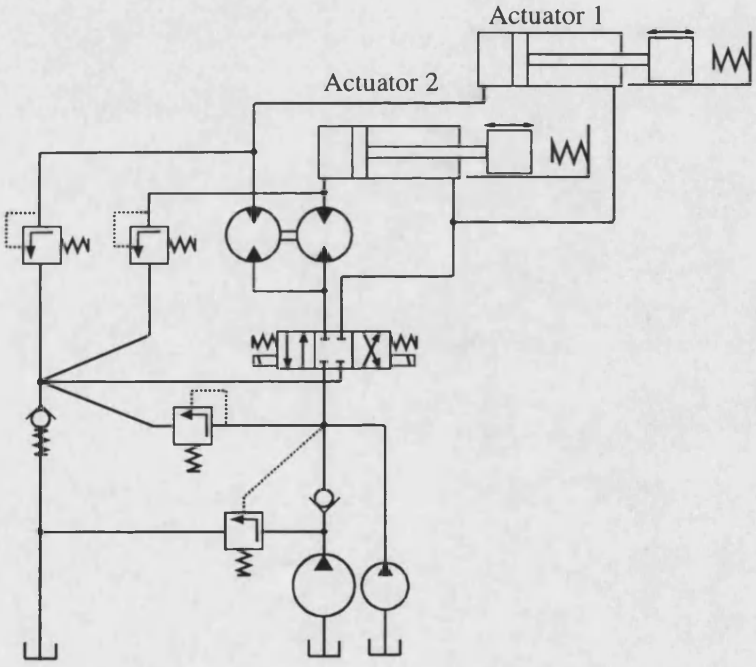


Figure 3.1 Two-actuator circuit

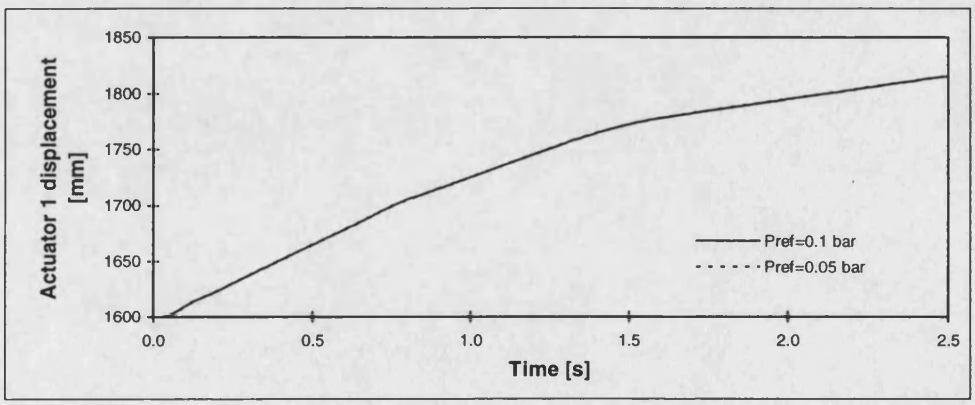


Figure 3.2 Actuator 1 displacement transients

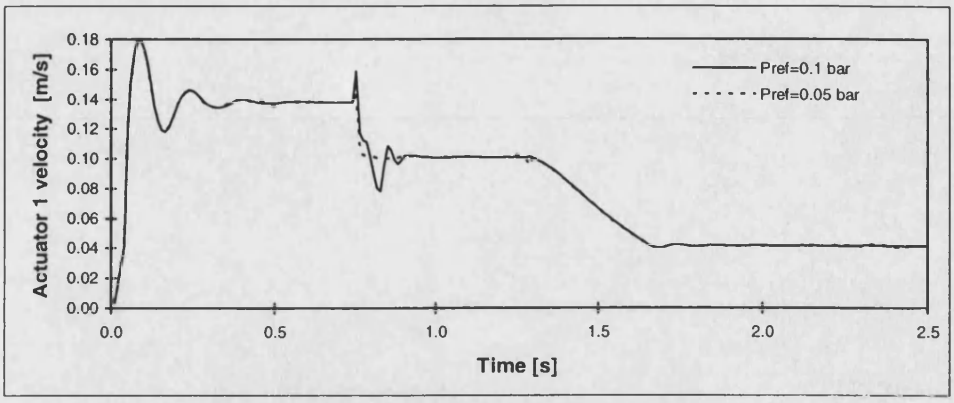


Figure 3.3 Actuator 1 velocity transients

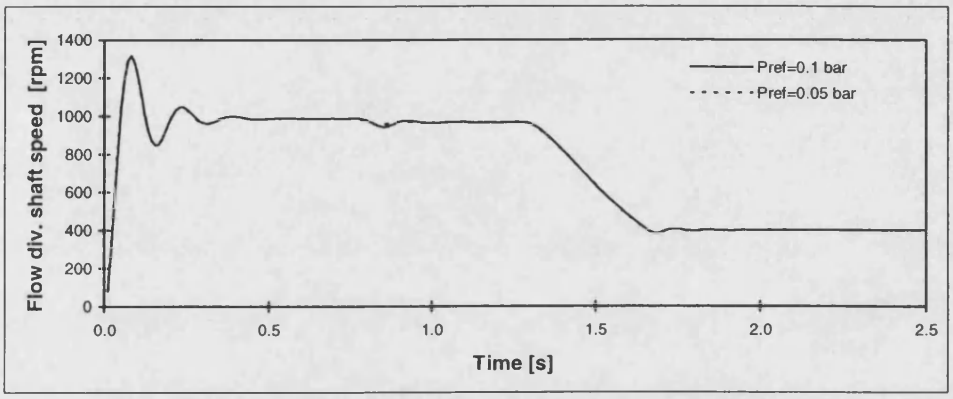


Figure 3.4 Flow-divider shaft speed transients

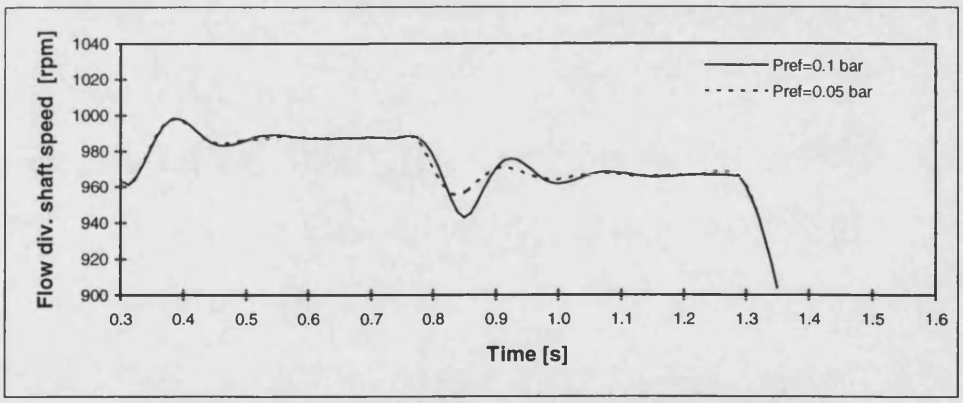


Figure 3.5 Flow-divider shaft speed transients (detail)

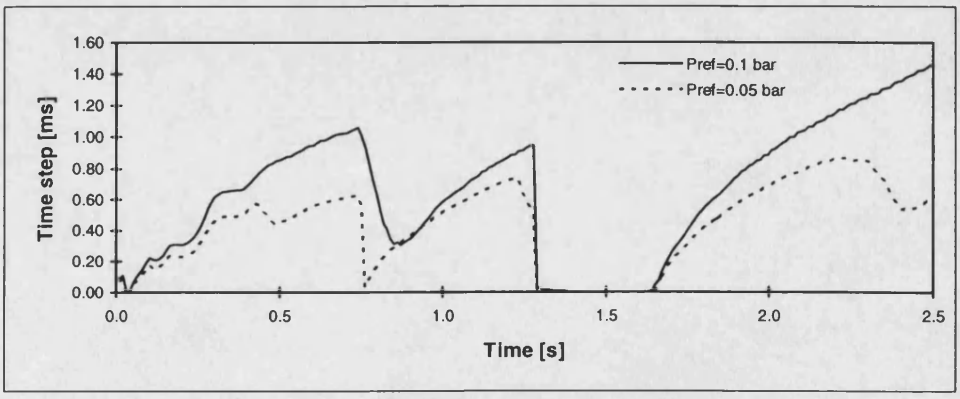


Figure 3.6 TLM time step against time

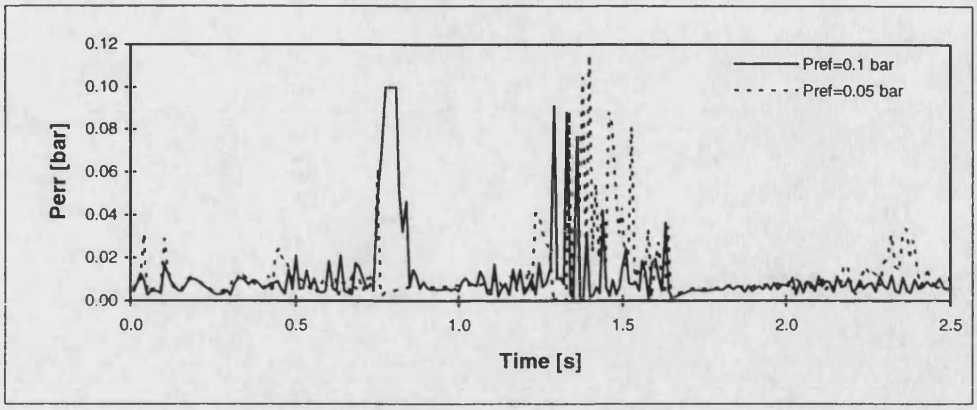


Figure 3.7 TLM parasitic pressure difference against time

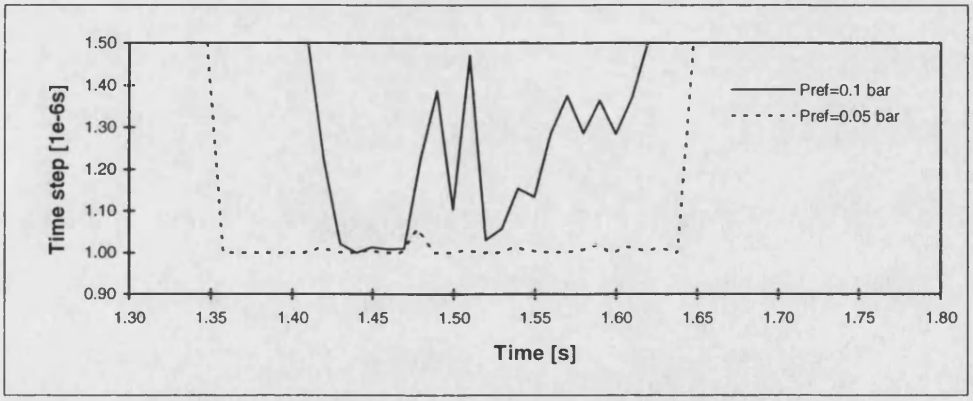


Figure 3.8 TLM timestep against time (detail)

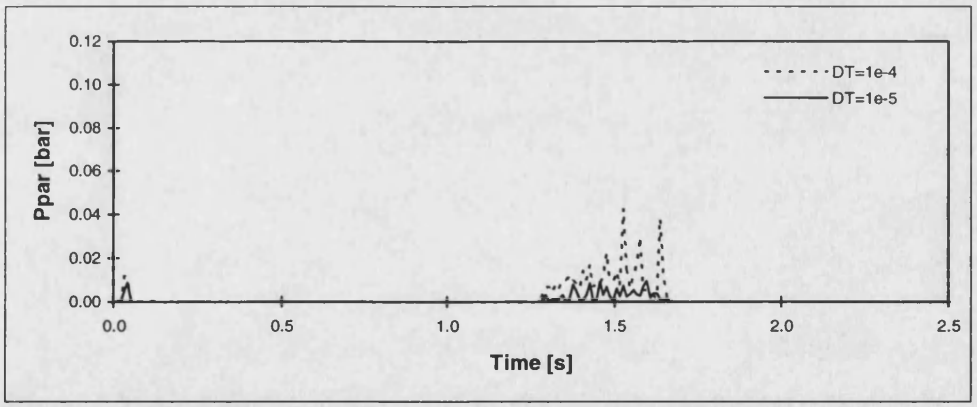


Figure 3.9 Fixed time step TLM parasitic pressure difference against time

4 Simulations with fixed time steps on parallel platforms

4.1 Introduction

In this chapter the hardware used in this research is described. The process of automated program (software) generation using pre-compiled component models to represent the separate circuit elements is explained for the development of multi-processor TLM simulations. Therefore the TLM simulation process and the TLM numerical algorithm by which the TLM models are solved are described. Channels are used as a concept to communicate and synchronise between processors. Using the message passing interface library the portability of the program generator concept is also investigated.

4.2 Parallel simulations using TLM

Parallel simulation of systems offers the benefit of increased speed and execution, but requires the system model to be partitioned to enable numerical tasks to be performed concurrently. The TLM method is inherently suitable for implementation in parallel architectures [Pulko & Olashore, 1989] where the transmission delay allows component models to be decoupled for the current time step. A large circuit can be decoupled into sub-circuits which can then be simulated concurrently.

Parallel simulation of a simple hydromechanical system was first described by Krus et al. [1990] using two Apple Macintosh II micro computers in parallel. For a particular circuit, they achieved a reduction in simulation time of about 40 per cent, enabling real time simulations to be undertaken with a time step of 0.015 seconds. For this simulation on two processors, the time taken for communication between the processors must be smaller than the time for the calculation of one simulation time step in order to improve computational performance.

TLM modelling of electromagnetic structures and fields has also been implemented on parallel platforms. The highly localised nature of the TLM algorithm (any change in the state of a TLM node affects only its immediate neighbours at the next time step) is perfectly suited for parallel processing [Dubard et al., 1991]. Parallel TLM of electrical problems has been implemented on massively parallel computers (DECmpp 12000 [So et al., 1995] and Connection Machine [Dubard et al., 1991]), on a transputer mother board with five T805 transputer modules [Fung et al., 1993] and on an SIMD computer (AMT

DAP 510) [Tan & Fusco, 1993]. In the latter paper the authors show that the computation time is reduced considerably for large mesh sizes and particularly if the mesh size is an integer multiple of the basic processing element array size. In parallel hydraulic system simulations this cannot be achieved because the different models at the nodes require different times for calculation. Such systems cannot be represented by a regular mesh of nodes. For the simulation of electrical fields on massively parallel computers it is most advantageous to use a processing system whose multiprocessor interconnection is geometrically equivalent to the TLM node interconnection [Tan & Fusco, 1993].

Another TLM parallel simulation method for the simulation of fluid power systems has been proposed by Jansson et al. [1992] where different time steps are used in two different subsystems. Both subsystems are connected by a volume in the middle that performs calculations using the smaller of the timesteps of the two subsystems. The subsystem that has the larger timestep is replaced by a constant flow when the subsystem with the shorter timestep is calculating the intermediate points. On a single processor this method can reduce computational cost when one part of a system requires a smaller timestep than other parts of the system. On the other hand this approach is dependent on the partitioning of the system and on the work-cycle that the system is performing. It also only works for dynamically independent systems and systems separated by large volumes. Furthermore, oscillations can occur due to the synchronisation of the two subsystems.

The partitioning problem of a particular hydraulic circuit was also investigated by Burton et al. [1993] and Burton [1994b]. A study was undertaken to assess the most appropriate number of processors and the best circuit partitioning strategy for a particular system. Various arbitrarily partitioned topologies on two to seven processors were investigated and the efficiency of each distributed simulation was compared. Some simplified partitioning guidelines were also given.

The load balancing problem of another circuit has been studied by Burton et al. [1993c]. Computational load balancing is a very important factor when partitioning TLM based simulations, because of the nature of the simulation technique. TLM computations require all component models in the simulation to be executed before progressing to the next time step, thus no single partition may be more than one time step ahead of any other partition. Consequently, all partitions in the distributed simulation are 'locked-linked', that is the computational speed is dependent upon the slowest partition [Burton et al. 1993c]

4.3 Hardware description

The parallel computing surface used for this research was selected after an extensive survey of available systems in 1994 [Pollmeier, 1996b, pp. 4-15]. After benchmarking of the different processors and considering the system requirements a T9000-based system was purchased.

4.3.1 Description of the new T9000-based system

Figure 4.1 shows the architecture of the T9000-based SN 9400 system. All SN 9400 SP systems require a host computer which provides a development environment for the user. The host also has to provide an interface to the system, through which the code may be loaded into the T9000 network and the results (or debug information) may be collected and displayed or stored.

A Pentium® (486 PC compatible) running Windows 3.11 with a large hard disk, 32 Mbytes of RAM and an Inmos B108 motherboard (host card) was chosen as the development platform. The transputer host card has to provide one or two transputer links to the outside world for connection to the SN 9400 equipment, in addition to a subsystem reset port. The B108 has two C101 chips (parallel DS[1]-Link adapter), each interfaces between the PC bus and a DS Link. The C101 provides an inter-networking solution for mixed processor systems and it converts between the serial DS-Link format and external systems such as busses, peripheral devices or microprocessors. One C101 is for the control links and the other is for data links.

4.3.2 Memory arrangement

In general there are three possible memory arrangements for a parallel computer as shown in Figure 4.2 When developing specialised parallel software applications the particular memory arrangement needs to be considered. With the shared memory arrangement the processors are connected to the memory via a bus or multistage network. Contention of the bus limits the scalability of this system but the processors share equal access to the memory. In the distributed memory case each processor is attached to local memory. The processors are connected with switches or routers which enable "point to point" links.

[1] DS - Data and Strobe signal wires are used to make up the link

Virtual shared memory brings together the two benefits but the time of memory access is not uniform. The T9000-based platform purchased uses the distributed memory arrangement as described in more detail in the following section. With this the scalability is superior. The total memory available is much greater than using shared memory, particularly when it is necessary to add more processors in order to upgrade the system.

The SN 9400 has 8 Size 2 HTRAMs[1] in sites 0-3 and 8-11, i.e. the system is extendible to up to 12 HTRAMs. Here each HTRAM consists of a T9000 transputer (20 MHz CPU) with 8 Mbytes of memory. Links 1 and 3 of each T9000 are connected to the C104 Asynchronous Packet Switch, and links 0 and 2 connect the HTRAMs in a pipeline. The C104 allows communication between devices, such as microprocessors, that are not directly connected. A single C104 can be used to connect up to 32 microprocessors. It can also be connected to other C104s to make larger and more complex switching networks, linking any number of processors, link adapters, and any other devices that use the link protocol.

Burton [1994] employed a variable time-step TLM method on a T800-based system which leads to a requirement for the placement of components onto processors: only one processor can be allowed to determine the time-step. Consequently the simulation must be partitioned such that one subsystem exchanges information with all others. This leads to the restriction of master-slave topologies (Figure 4.3). The master process governs the operation of the slaves, by defining the simulation time-step used in the computation.

Figure 4.4 shows a comparison between the T800-based system (System T8) used by Burton [1994] and the new T9000-based system (System T9). Both systems contain 8 processors, here numbered from P1 to P8. Communication between e.g. processor P1 and P3 on System T8 has to be established via processor P2 and P4. This leads to unacceptable latencies and communication delays. Hence in Burton [1994] only nearest neighbour communication between processors was permitted. System T9 can communicate from any processor to every other processor with small latencies via the C104 switch, i.e. it is more flexible with much faster communication between all processors. All four links of each processor can be used concurrently and the C104 switch enables any four pairs of processors to communicate at the same time. This leads to many more possible process-

[1] HTRAM - High performance TRAnsputer Module

processor mappings including arbitrary master-slave configurations. But this also makes automatic mapping (described in chapter 5) more difficult.

Additional to the flexibility and communication improvements, the new T9-based platform fulfils the following system requirements:

- Double precision is implemented in hardware. Tests indicated that double precision is necessary to achieve acceptably accurate results.
- The system is well designed for fine grained problems. Granularity is explained in more detail in section 4.3.4.
- Greater processing power and communication bandwidth then the existing T8-based platform.
- The system is flexible and can be upgraded from 8 processors to more and faster processors at a later stage if necessary.
- The system supports the programming language 'C' enabling the reuse of parts of previously developed code.
- 8 megabytes of memory per processor are available.
- A good system support is available in the UK
- Many different topologies are possible due to the C104 switch which can be changed during runtime.
- Furthermore, the system can easily be connected to a data acquisition system as described in the following section.

4.3.3 The implemented data acquisition system

Two OS[1] Links from the SN 9400 are interfaced via a 44-way compact D-type connector to the special system containing an ecm_7720 TRAM mother board and an adt108C A-to-D converter TRAM (Figure 4.1). The adt108 series is a multiple input, 12 bit, 100 kHz sampling Analogue to Digital converter module built to the Inmos TRAM format (based on a T425). Its signal part comprises an input multiplexer, instrumentation amplifier with software programmable gain per channel, sample and hold amplifier and Analogue to Digital converter. The TRAM is configured with 16 differential inputs and the input

[1] OS - Over-Sampling technique used to extract the data from the link

voltage ranges are selectable from one of +/-5V, +/-10V, 0 to 5V, 0 to 10V and are overvoltage protected. Input signals are taken through a 50-way IDC connector.

4.3.4 Granularity

TLM simulations of hydraulic systems lead to fine grained problems, i.e. simulations where a relatively long time is spent for communication between processors. In general, the transfer of delayed characteristic pressures between decoupled component models either takes place locally on the same processor via 'on-chip' memory, or for parallel simulations with connected processors via serial data links or shared memory. With this approach pressure and flow have to be exchanged every time-step, hence the simulation is a fine-grained computation. In fine-grained parallelism, what is normally thought of as a single, indivisible calculation is partitioned among processors. This commonly requires relatively frequent communication between programs running on different CPUs (e.g. different iterations of a subdivided loop are executed by different processors). In coarse-grained problems, each calculation is conceptually nearly independent of the others and normally involves relatively infrequent communication among the individual calculations (e.g. database management).

4.4 Parallel implementation of TLM on T9000-based platform

4.4.1 Program generator

In this section, the process of automated program generation using pre-compiled component models, is described for the development of multi-processor TLM simulation. The pre-compiled component models represent the separate circuit elements and the source code is generated for the T9000-based platform described in section 4.3. Automated generation of the simulation program enables the system modeller to develop large and complex circuit configurations from much simpler modular elements.

In general, the use of automatic program generation leads to improvements in:

Efficiency: It takes less time to build and modify simulation programs.

Reliability: New component models may be tested in isolation for correct operation. Large circuits can then be developed with increased confidence.

Simplicity: The analyst is separated from the task of creating models and writing source code directly using a 4th generation language such as C. Even the

implementation of the simulation in parallel is achieved automatically. This task is normally very complicated, time consuming and prone to errors.

Notwithstanding the potential benefits indicated, the development of a program generator is a detailed and complex task. The program generator described here and all TLM component models have been written in the C programming language using the Borland C++ compiler and the INMOS T9000 ANSI C tools, respectively.

For the TLM code generator, the models used in the simulation and how they are connected together are defined by the user in an ASCII text file, subsequently named the *link* file. Details on the program generator and how to create a link file from a hydraulic circuit drawing are given in [Pollmeier, 1996c].

4.4.2 Simulation process

Figure 4.5 shows schematically the complete simulation process. Firstly, the user-supplied link file (extension .lnk) is interpreted by the link file generator. This module then writes a new link file for each processor. The user-defined link file contains the following five sections:

1. System parametric data. This section contains data on the fluid properties, simulation start and end times, results print interval, time step and the number of processors used with this simulation.
2. Transmission line data. Here the program reads in the individual link numbers and the link type (fluid or signal link). For hydraulic transmission line links, the pipe diameter, length, effective bulk modulus and initial pressure are also specified.
3. Component model data. Here the following information is detailed for every component model used in the circuit: model name, links to transmission lines followed by a list of component parametric data. The latter data have to be entered into the link file in a specific sequence corresponding to the parameter argument list for the model.
4. System partitioning data. This section contains information about the mapping of the simulation on several processors. Each model is assigned to one of the transputers used.
5. Inter-processor communication data. In this final section the expected order in which the processors finish their computational load is included.

The method used to get the information for the latter two sections is described in Chapter 5. When interpreting the link file the program makes a number of checks to ensure that a valid circuit model can be created. Therefore the data in the model attribute file (containing e.g. the number of ports and parameters of each available model) are read and compared with the component model data (section 3). In addition, links between models are checked to ensure that they are defined only once and connected correctly. The link file generator cannot validate the hydraulic functionality of a circuit description, it can only check whether a certain arrangement is physically possible (e.g. hydraulic lines cannot be connected to signal lines). The loaded link file data is then used to write a new link file for each processor. From the sections described above the first three are just copied (without the comments) to the new link files. The latter two sections are used to calculate the order and the direction of communication between the connected partitions. This information is then also written to the new link files which all have to contain the same parametric data from section 1 [for details see Pollmeier, 1996c].

For numerical simulations many of the source files written by the program generator have similar structure and many common functions. It is for this reason that template files are read by the code generator line-by-line and used to construct the simulation program source, header and make files, by inserting relevant code excerpts into the file templates. The use of template files as a means of simplifying the code generation process is not uncommon, and has been adopted in the **BATHfp** package and a previously-developed single processor TLM program generator [Burton, 1994].

In all source programs the appropriate model call argument list must be written for every occurrence of a component, corresponding to each model entry in the link files. Moreover, simulation data associated with each component model must be stored by the control program during execution to ensure data integrity. This is necessary, as there may be multiple occurrences of the same model using the same component model code, but having different data.

The automatically-generated header files contain the particular variable declarations, required by the particular source code files. Finally, the system make file (called *makefile*) contains all of the arguments necessary to build an executable file from all of the constituent elements. The files additionally required by the make command are explained in section 4.4.3. These files are the files not written by the main code generator. The make command can be automatically invoked upon the successful generation of the system

model source and header files by the program generator. A single executable file with the extension .btl is produced which can be executed on the command line within a DOS-window on the PC host computer. Furthermore *make* generates the network initialisation file (.nif).

A hosted T9000 network is initialised by sending control commands to the transputer and router connected to the control network. The code is then loaded using the data network. The data used to generate the control network commands is held in the network initialisation file. This contains all the information needed to initialise a system through the control links prior to loading the application code. It can be automatically generated by the initialisation file generator tool (*inif* tool in the toolset), from the network description and memory configurations. To initialise the network, the network initialisation file is used by the initialisation software to generate the correct sequence of commands to send to the control link network. In a hosted system, the initialisation software runs on the host and sends the initialisation commands to the control network, as given by the initialisation file. The simulation is then started calling the executable file with the network initialisation file as its argument.

Following completion of the simulation, binary and/or text (ASCII) results files are written containing simulation data saved by the separate component models during the simulation. Each model also writes an entry into an ASCII reference file (extension .ref) during the initialisation phase of the simulation. It details every component model used in the circuit, the parameters saved by each model for user output and a unique integer number corresponding to the parameter position in the results files. A description of the model the parameter came from and the model "instance" number is also included. This unique integer number corresponds to the model position (from first to last) in the respective link file. The use of a graphical post-processing utility in conjunction with the reference files enables the system modeller to inspect the simulation results of each partition.

4.4.3 Other files required by the make file

Pre-compiled model object files (.TCO)

The source code for the component models is divided into two parts for each model. An initialisation function is used to pre-set certain variables and to write the aforementioned entry into the reference file. The second part (the main function) contains the component equations, i.e. it is used to calculate pressure, flow, speed, displacement, etc. for the next

time step. Both functions are written in the C programming language. All model object files are then included in the make file instructions using a library indirect file. This is a text file containing a list of all compiled models.

Global header file (TGLOBAL.H)

The global header file contains only very specific variable declarations, accessible by all sub-programs in the source code. These are maximum array sizes, transmission line structures and data structures used for the inter-processor communication. In addition, some software-specific header files are included.

Multiplexer (KMUX.C)

This file is an additional source code file and its function is to enable all processors to communicate with the host processor. With this, results from each processor can be displayed on the screen during the parallel simulation. Furthermore, at the end of the simulation each partition can write its own results files to the hard disk of the host processor.

Hardware network description file (.NDL)

The hardware is described using the Network Description Language (NDL), which uses simple textual descriptions of device attributes. The NDL supports the setting up of attributes to configure the subsystems of the IMS T9000 to the user's requirements. These include the memory interface, cache and links. This description is used for two purposes: for the software configuration (allocating code and data to processors, and channels to links) and for the hardware initialisation. The use of NDL means that there is a single system description for use by all the development tools.

The hardware description includes a definition of all the nodes in the network. A node in a T9000-series network is a device or part of a device which is initialised by means of a control link or a ROM. The T9000 transputers, asynchronous packet switches (C104) and system protocol converters (C100), for example, are all nodes.

Each node has attributes. Attributes are of one of the following two types:

- A value attribute has a numerical value which can be set. The value gives information about the node, such as the amount of memory present. Most attributes represent a value which will be written to a configuration register during initialisation.

- An edge attribute is a communications port of the node. Edge attributes do not have values, but they specify whether an edge is connected.

Connections to nodes and to devices external to the network are referred to as edges. An edge for an external device is also known as a network edge or external edge. The complete network description also has to define the connections (or arcs) between nodes and to the outside world. Arcs always connect edges. Arcs between nodes need not be named.

These concepts are illustrated in Figure 4.6

It is important to note that the configuration tools allow a complete separation of the hardware description and the mapping of software onto that hardware. The network description exports the names of the nodes, arcs and routes, making the objects available to software configuration descriptions. These are then used when mapping a particular piece of software onto the hardware [INMOS, 1994a,b].

Software network description file (.CFS)

The software network is described in the configuration description file (.CFS). A configuration description states how code is to be run on a network of transputers. It contains a definition of the software network and a mapping description defining how the software processes are to be placed on the transputer network. The software and mapping descriptions are written in the INMOS configuration language.

When preparing an application for a network, each software process is constructed as though it were a separate program. All the source files for it are compiled, the object modules are then linked to form a file known as a linked object file. The linked object file embodies the program in a way that can be used as a process in a network.

The software network is composed of nodes of the predefined type *process* connected by input and output channels. The software description consists of a series of process declarations that define the network's interface with the outside world, and the connections between them. A separate statement is used to assign linked object files to the software processes.

The description of the hardware network is contained in a separate file, the network description file, as explained above. This file is referenced by the configuration description. The separation of hardware and software descriptions facilitates their

independence and extends their functionality. By modifying the mapping descriptions a variety of software configurations can reuse the same hardware network description and a software description can be mapped onto different hardware networks. In addition to describing the configuration, the description references the linked object files which form the application [INMOS, 1994a,b]. The object files are the compiled multiplexer source code, the compiled source code files of each partition and the pre-compiled model object files.

4.4.4 TLM numerical algorithm

A key part of the TLM simulation is the algorithm by which the TLM models are solved. The parallel algorithm developed and used here is illustrated by the flow-chart of Figure 4.7. Here only two processors/processes are shown but the chart can easily be extended to more processors. The TLM algorithm is based on two basic sets of equations - those which define the propagation of information between component models (equations 2.5 and 2.6 for a loss-less line with equations 2.25 to 2.27 for the approximated friction), and those which form the component models (e.g. equation 2.14 and 2.16 for the relief valve in the modelling example, section 2.2.2)

The first stage of a simulation is to set various global parameters and initial values for the main simulation and for each model used. A simulation loop is then entered until the simulation is completed. In the first part of the simulation loop each model calculates pressures and flows at each end of each line from the known characteristic pressures (determined at the previous time-step). From these pressure and flow values the new characteristic pressures are calculated which are then exchanged between all connected processors. At the same time the new pressure values are also propagated. The communication between processors is designed to be dead-lock free and as efficient as possible. Details on the inter-processor communication are given in section 4.4.5.

After the communication between the processors, the pressure difference across each line is calculated and filtered according to equation (3.4) These values are then used to determine the maximum parasitic pressure difference. At this point low-pass filtering of the characteristic pressures using equation (2.26) is applied. The filtering algorithm introduces a modification to the line impedance (as do changing volumes e.g. in an actuator). Hence, the characteristic pressures must be compensated to allow for the change in impedance, using equations (3.9) to (3.11). The procedure above must allow for the

total volume of fluid associated with each line, and this may include fluid contained within the components. However, as this volume is subject to change it is important that the total line volume is updated at the start of each loop, and that at the end of each loop the line volumes are reset to the pipe volumes only.

The final part of the loop is to transfer the newly-calculated characteristic pressures between connected components on the same processor. If the time has advanced beyond the last storage time by the user-defined print interval results may be stored to memory. The simulation time is then incremented by the time-step and, if the end-time has not been reached, the loop starts again. Finally, the simulation results of each partition are copied from local memory to files on the host processor hard disk.

The package has been tested against experimental data and against a standard package which employs an equivalent system of ordinary differential equations (Bathfp) and hence provides a suitable "control".

4.4.5 Communication and synchronisation between processors

Parallel processing is widely accepted as an important way of improving software performance on any given processor architecture. The transputer supports parallel processing directly by incorporating into its design a process scheduler which is responsible for scheduling parallel tasks, and by providing the means for connecting processors (transputer links) to create a multi-processor network.

Parallel programming is supported in the INMOS C toolset by extra library functions. These functions allow processes to be defined and created, to communicate with one another via channels and to synchronise using semaphores. Parallel processing in transputer based systems is based on the idea of Communicating Sequential Processes (CSP) developed by Hoare [1985]. CSP is an abstract generalised model of concurrency based on the idea of independently executing processes exchanging data by synchronised communications. The model can be used to describe software applications in an intuitive way reflecting the parallelism of the real world [INMOS, 1994a,b].

Concurrent C processes are independent, they can be nested within each other, and they are linked together by channels. Any C function can be defined as a concurrent process using a special set of functions provided in the runtime library. Processes can

communicate either unidirectionally (one process passing data to another) or bi-directionally (two processes exchanging data and working in a co-operative manner)

Three new data types are introduced for the concurrency support. Data structures are used to hold information about processes and semaphores, and a pointer type is used to implement channels.

- Process: A structure type to hold information about a declared process.
- Channel: A data type used to implement channels.
- Semaphore: A structure type that holds information about a semaphore.

Processes which exchange messages and data with each other must do so via a pair of channels. Channels have two functions. They provide the communication path between independently executing processes, and serve to synchronise the communication between the two processes. Processes which send data cannot do so until the receiving process is ready. Neither process can continue until the communication is completed. In this way synchronisation between the two processes is assured: no data is passed until both partners in the operation are ready.

Semaphores are the traditional way of co-ordinating activity in shared memory environments. They are also implemented within the toolset using channel functions, and are therefore subject to a slightly greater overhead than if the intrinsic synchronising ability of channels were used directly. Hence, channels are used for communication and synchronisation between different processors.

4.5 Portability of the program generator concept to other platforms

The program generator was developed for TLM simulations on specialised hardware using a particular development tool (INMOS C toolset). In general using specialised libraries one cannot put the code on any other machine. Nevertheless, the program generator has been designed in such a way that it can easily be adapted to any parallel platform supporting the message passing interface library (MPI). MPI is described in the following section. The general concept used for the program generation does not need to be changed.

4.5.1 Message passing interface (MPI)

During 1993, a broadly based group of parallel computer vendors, software writers, and application scientists collaborated on the development of a standard portable message-

passing library definition called MPI. MPI is a specification for a library of routines to be called from C and FORTRAN programs. The programs that the user writes in FORTRAN 77 and C are compiled with ordinary compilers and linked with the MPI library [Gropp et al., 1994]. In the same reference it is also predicted/assumed that the majority of programmers of parallel computers will, in the long run, access parallelism through libraries. Indeed, enabling the construction of robust libraries is one of the primary motives behind the MPI effort, and perhaps its single most distinguishing feature when compared with other parallel programming environments. Major goals of MPI are portability across different machines and architecture, efficiency and reliability of code and language independency.

It is a sign of the maturity of parallel computing that there already exist many explicitly parallel application programs written for the message passing model. Old programs needing to be ported to new hardware and software platforms used to be called "dusty decks". Now they are more respectfully called "legacy applications". In general, porting them to MPI will be straightforward, because MPI is for the most part a functional superset of existing message-passing libraries. Therefore the changes to existing programs will be largely syntactical changes and, in some cases, can be done automatically [Gropp et al., 1994].

The six fundamental functions of MPI are given in Table 4.1. One can write complete message passing programs with just these six functions. The remaining MPI functions (about 120) 'just' add flexibility, robustness, efficiency, modularity or simply convenience.

4.5.2 Porting to MPI

The parallel TLM simulation contains a set of processes that have only local memory but are able to communicate with other processes by sending and receiving messages. In the INMOS C toolset this point-to-point communication is realised with channels (compare section 4.4.5). Only two processes are involved, one process sends and the other receives. The basic channel input and output commands are ChanIn and ChanOut, respectively, and they are called the following format:

```
ChanIn (Channel *c, void *cp, int count);
```

With the arguments:

Channel *c A pointer to the input channel.

void *cp A pointer to the array where the data will be stored.

int count The number of bytes of data.

```
ChanOut (Channel *c, void *cp, int count);
```

With the arguments:

Channel *c A pointer to the output channel.

void *cp A pointer to an array containing the output data.

int count The number of bytes of data.

These channel communications can be rewritten in the equivalent basic (blocking) MPI send and receive operation.

```
MPI_Recv(buf, count, datatype, source, tag, comm,
status, ierror)
```

With the arguments:

buf Starting address of buffer to receive.

count Number of elements to receive.

datatype Datatype of each element.

source Process to receive from.

tag Message tag.

comm Communicator.

status Status object.

ierror Error handler.

```
MPI_Send(buf, count, datatype, dest, tag, comm,
ierror)
```

With the arguments:

buf Starting address of data to send.

count Number of elements to send.

datatype Datatype of each element.

source Process to send to.

tag Message tag.

comm Communicator.

ierror Error handler.

With the functions in Table 4.1 for initialisation, termination, etc. the TLM simulation can be completely ported to MPI format. In general the computation remains a collection of

processes communicating with messages. Using MPI the previously necessary network description files (.NDL/.CFS) and the network initialisation file (.NIF) become superfluous. Their task is automatically performed by the particular MPI implementation.

The program generator was generally designed for homogenous systems only, whereas MPI can also deal with heterogeneous systems, i.e. systems with different processors. For these systems only the automatic partitioning described in chapter 5 needs to be updated so that it can take different processors into account. The program generator can still be used with the same inputs, i.e. the mapping of processes to processors. But using MPI the program generator will then only create source code that defines the different processes and communication between them. The processes will automatically be assigned to processors by the MPI implementation. This should lead to good parallel efficiencies and speedups because the program generator will only create source code for as many processes as there are processors available. MPI does not need to find the best process processor mapping, that is done before the program generator starts, i.e. the described approach can increase the efficiency of MPI.

MPI does not include I/O functions, i.e. the multiplexer needs to be rewritten so that it allows the different processes to write to the screen or to the host processors hard disk without using channel communication functions.

4.6 Closure

In this chapter the process of automatic system model generation has been described for the case of multi-processor TLM simulations. Advantages of the new T9000-based platform over the T800-based system were shown. The TLM simulation process and the TLM numerical algorithm by which the TLM models are solved were described. Channels as a concept to communicate and synchronise between processors were introduced. The principal way of porting the software to other platforms was described using the message passing interface library.

An automatic process-processor mapping method is developed in the following chapter. This method leads to the information required by the program generator.

TABLES FOR CHAPTER 4

MPI command	Description of the command
MPI_Init	Initialise MPI
MPI_Comm_size	Find out how many processes there are
MPI_Comm_rank	Find out which process I am
MPI_Send	Send a message
MPI_Recv	Receive a message
MPI_Finalize	Terminate MPI

Table 4.1 The six basic functions of MPI

FIGURE FOR CHAPTER 4

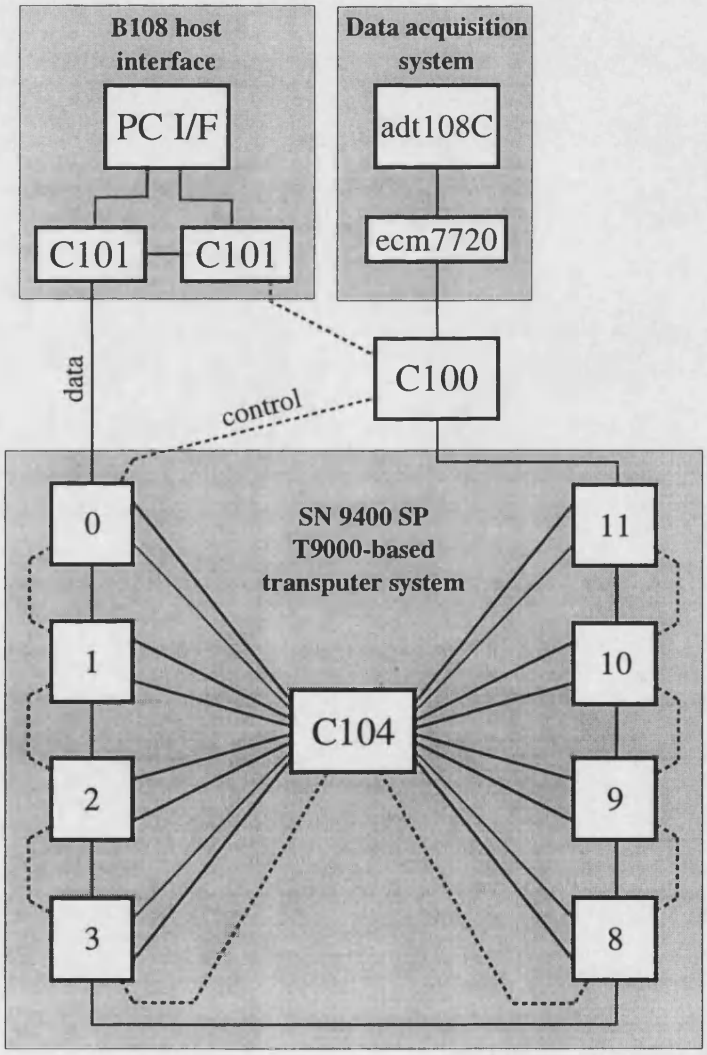


Figure 4.1 Architecture of an 8 node SN9400 SP system

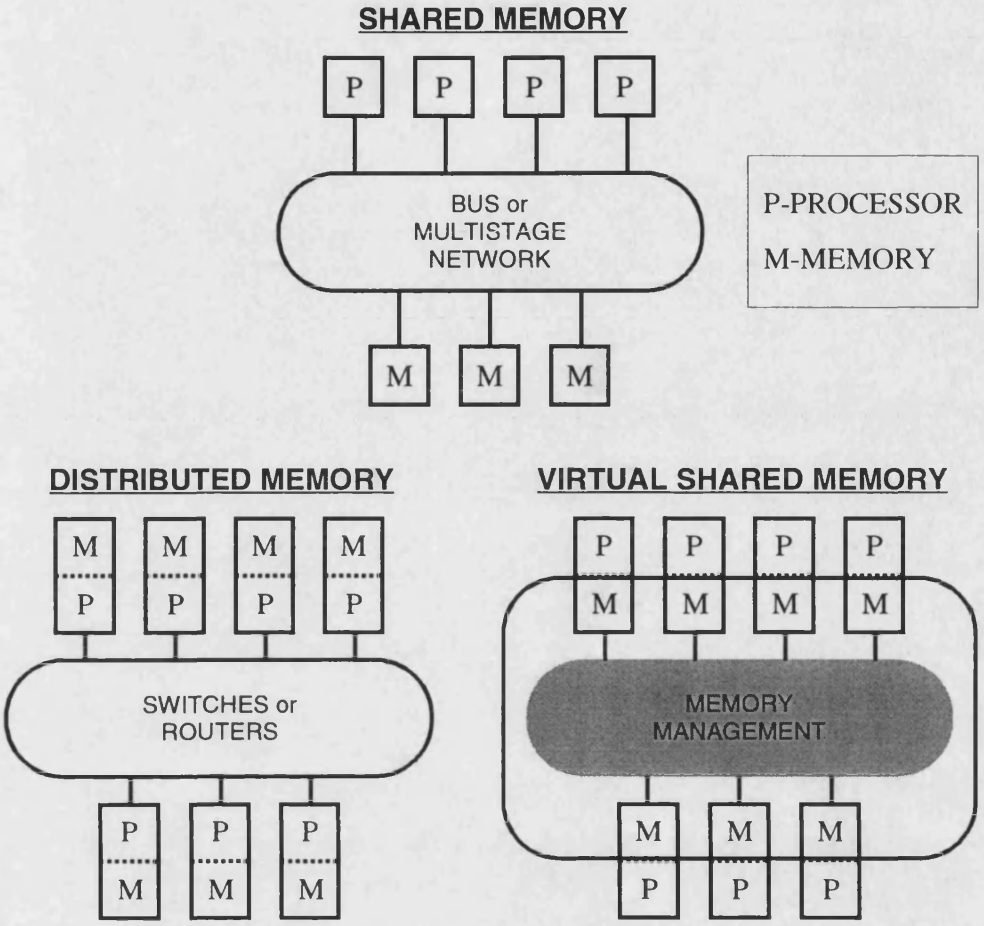


Figure 4.2 Memory arrangements

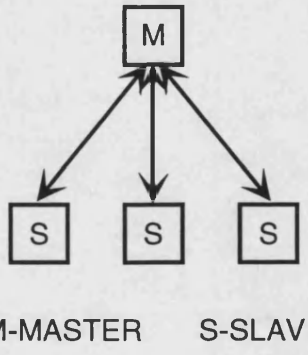


Figure 4.3 Master-slave topology

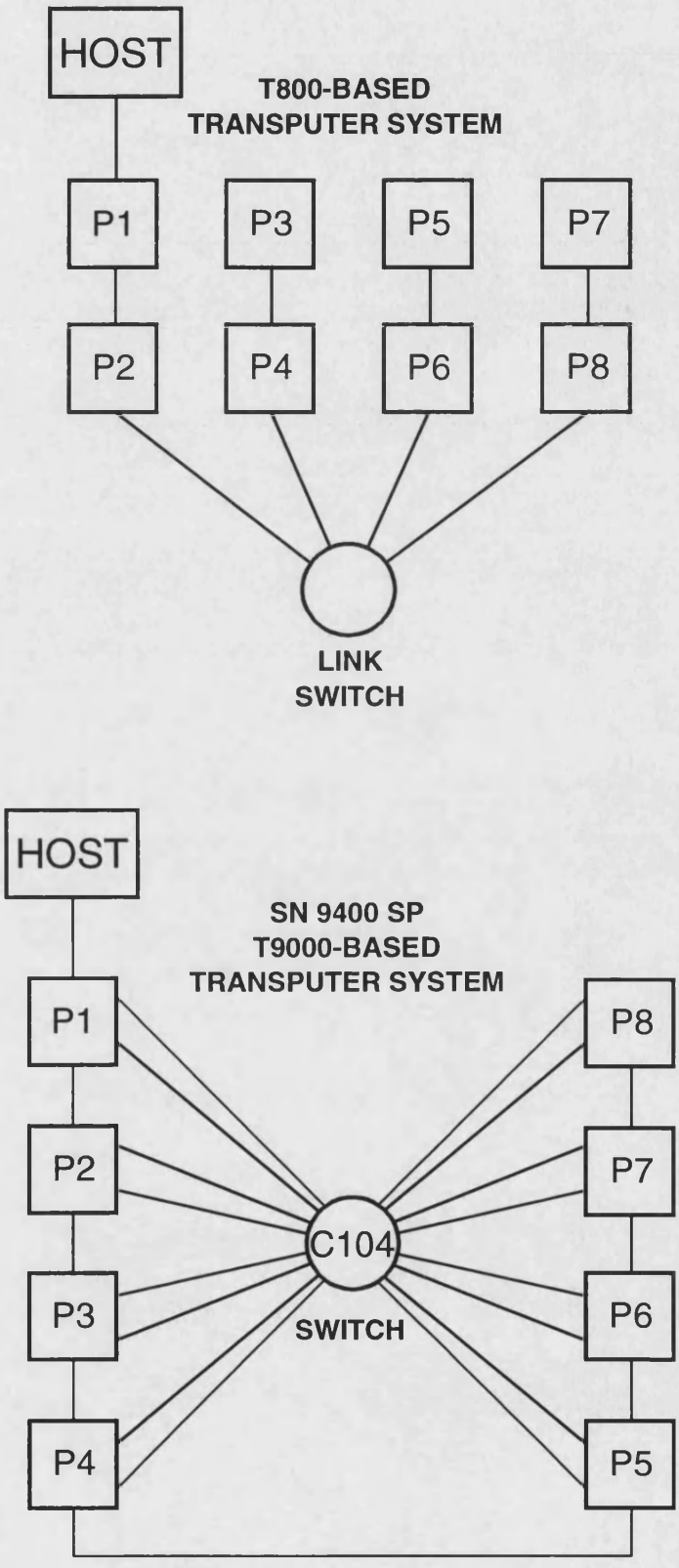


Figure 4.4 Comparison between T800- and T9000-based system

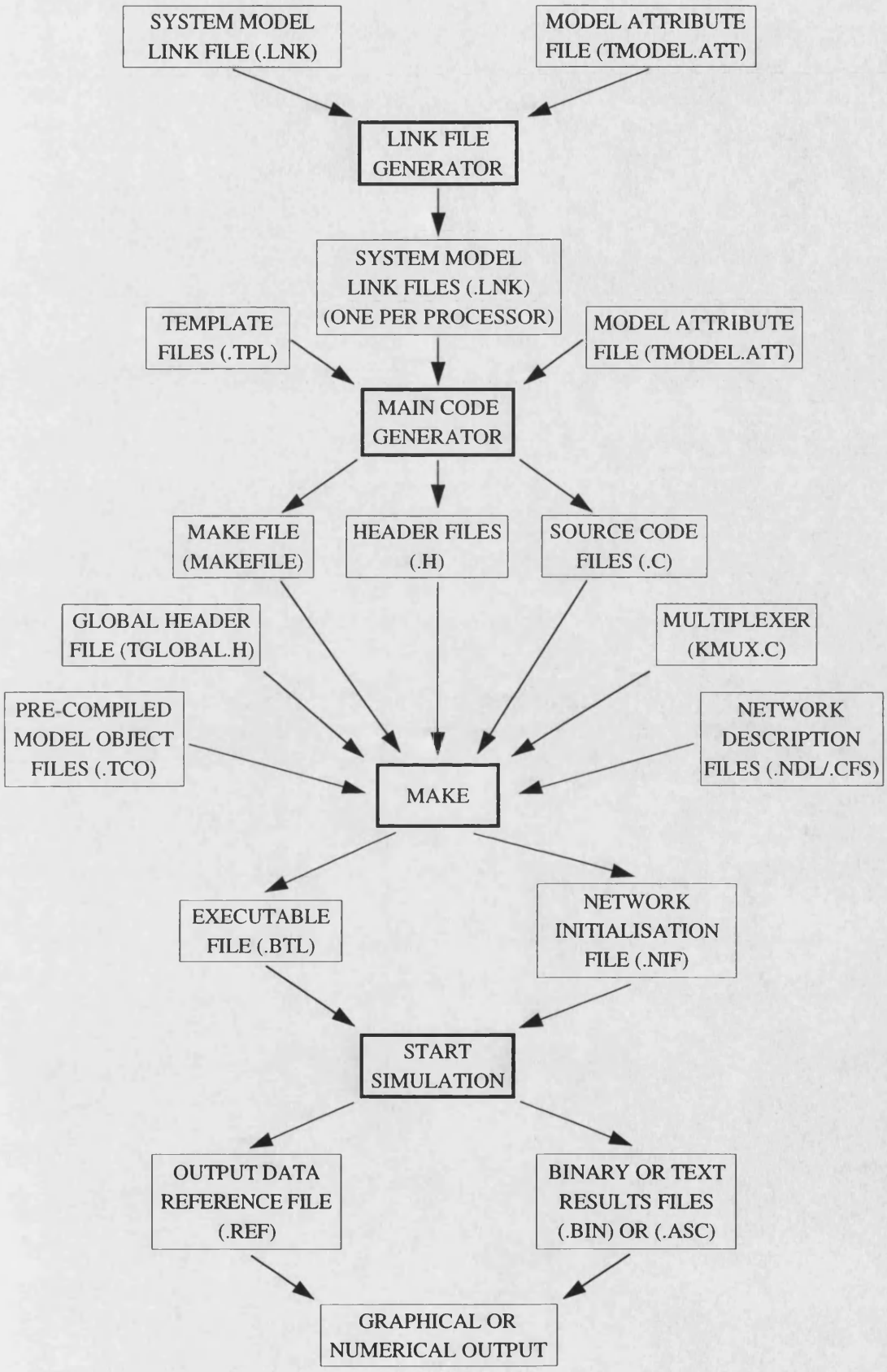


Figure 4.5 Simulation process: Automatic code generation

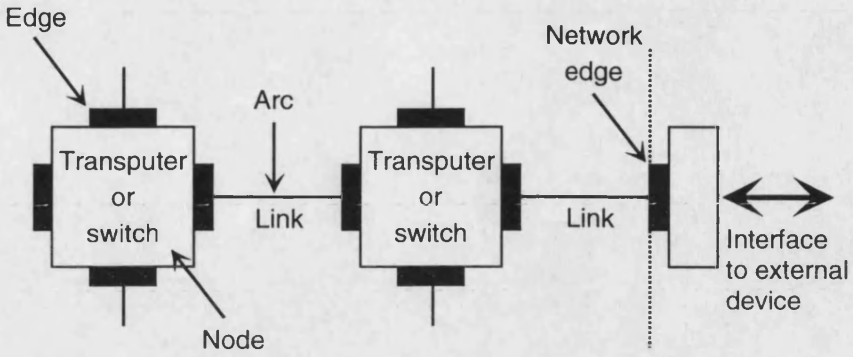
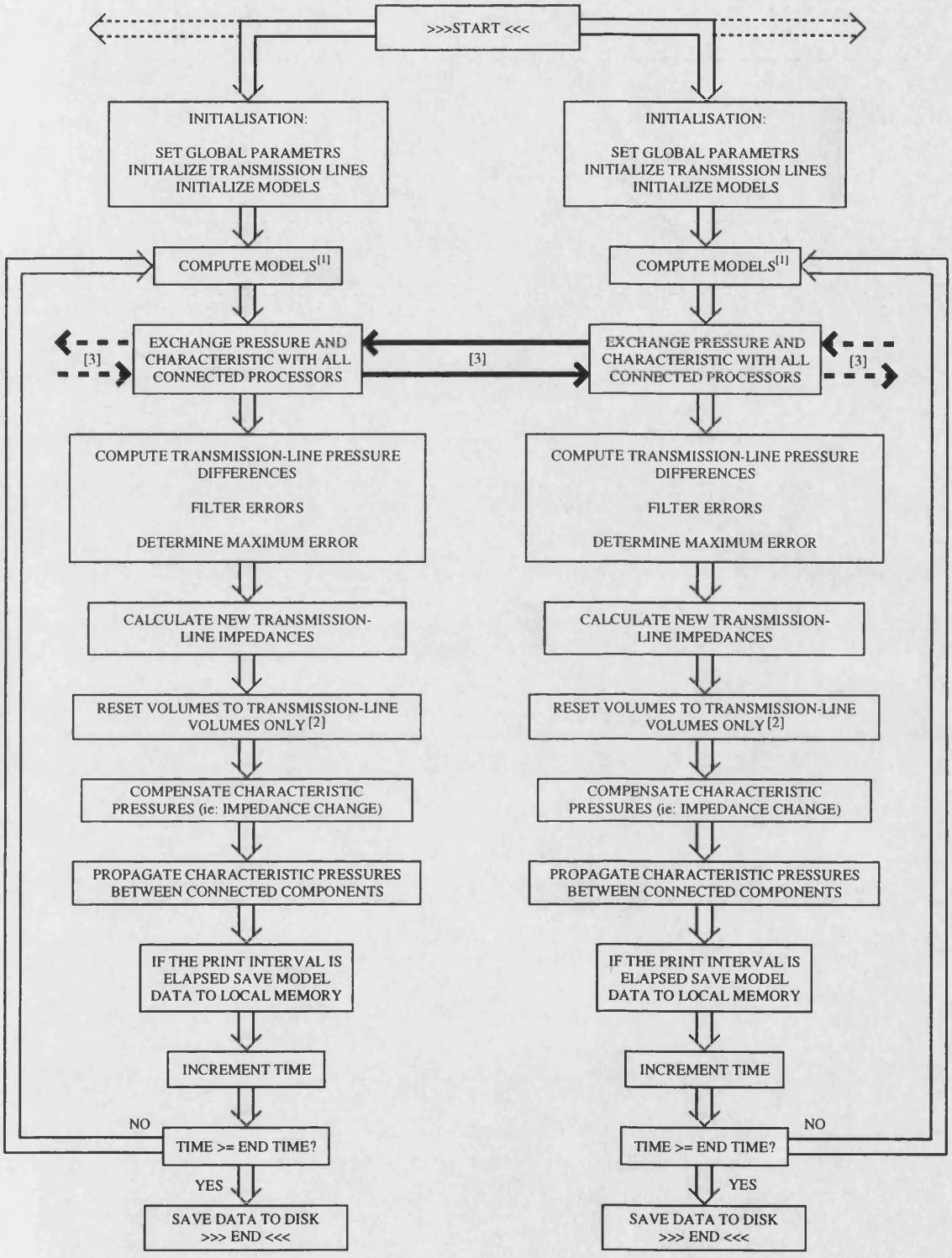


Figure 4.6 Basic elements of a network description [INMOS, 1994b]



[1] Models must execute in the correct sequence in the case of control/duty-cycle

[2] Individual components may add variable volumes to transmission lines (e.g. trapped volumes in an actuator). Volumes must be reset to line volumes only at the end of each model call sequence

[3] Inter-processor communication (a deadlock free communication scheme is described in section 4.5)

Figure 4.7 Flow chart of partitioned TLM algorithm

5 Process-processor mapping

5.1 Introduction

In this chapter the development of effective techniques for the distribution of the processes of parallel TLM programs onto multiple processors is investigated. Parts of the shown results have been presented previously [Pollmeier et al., 1995]. The problem is how to distribute (or schedule) the processes among processing elements to achieve some performance goal(s), such as minimising execution time, maximising resource utilisation and/or achieving real time simulations.

The highest level of decomposition that can be applied to a circuit is to place each component onto its own processor. However, this is generally undesirable due to the communication overheads that may be imposed. It is preferable to group components together on a processor; each group of components representing a subsystem of the circuit.

Local scheduling performed by the operating system of a processor consists of the assignment of processes to the time-slices of the processor. Global scheduling, on the other hand, is the process of deciding where to execute a process in a multiprocessor system. This chapter will address (global) static scheduling, i.e. information regarding task execution times and processing resources are assumed to be known in advance.

5.2 Partitioning and scheduling (process-processor mapping) in general

There are three fundamental problems to be solved in the execution of a parallel program on a multiprocessor: identifying the parallelism in the program, partitioning the program into tasks and scheduling/assigning the tasks on to processors. Whereas the problem of identifying parallelism is a programming language issue, the partitioning and scheduling problems are intimately related to parameters of the target multiprocessor, such as the number of processors and the synchronisation and communication overhead [Sarkar, 1989]. It is desirable for the partitioning and scheduling to be performed automatically in order to achieve optimal or near-optimal parallel performance.

There are three possibilities for automatic partitioning and scheduling:

1. Partitioning and scheduling both at run-time
2. Compile-time partitioning and run-time scheduling

3. Partitioning and scheduling both at compile-time

In the first approach, both the partitioning and the scheduling decisions are postponed until run-time. The major disadvantage in doing everything at run-time is the extra overhead incurred during program execution. The advantage is the availability of run-time information which may lead to a better partition and schedule. Sarkar [1989] investigates the second approach with a so-called macro-dataflow model. The task partition is restricted so that the inter-task dependencies are acyclic. With this restriction, the model of compile-time partitioning and run-time scheduling is like the dataflow model at the granularity of tasks, rather than machine instructions. However, the large overhead of run-time analysis necessitates very simple partitioning and/or scheduling algorithms. This research study did not pursue the run-time partitioning and scheduling approaches because of the problems with excessive run-time overhead, and the corresponding limitations on the partitioning and scheduling algorithms. Furthermore, for real-time simulations the time for the simulation must be predictable at compile-time.

In the third approach, both partitioning and scheduling are performed automatically at compile time. Compile-time scheduling is attractive because it eliminates the scheduling overhead entirely at run-time. The disadvantage is that the compile-time estimates of execution time and overheads may be inaccurate for some program inputs, leading to inefficient schedules. Considering real-time simulations in this thesis only the latter approach is investigated.

Partitioning is necessary to ensure that the granularity of the parallel program is coarse enough for the target multiprocessor, without losing too much parallelism. Scheduling is necessary to achieve a good processor utilisation and to optimise inter-processor communication in the target multiprocessor.

5.2.1 Partitioning

There are two ways of partitioning on parallel computers, partitioning data or partitioning the program. The former is easier to do and it is used for the simulation of the same system with for example different input data. It has been used amongst other things for the optimisation of hydraulic systems, where the same system is simulated with different parameters in order to find the best parameter set for a desired system performance [Donne, 1993]. Unfortunately this cannot be used for real time simulations of a particular simulation of a single hydraulic system.

In the second approach the partitioning of a parallel program specifies the sequential units of computation in the program. For convenience, we call such a sequential unit of computation a 'task', though it does not necessarily have the characteristics of an operating system task. According to Sarkar [1989] the properties of a task which are of interest are:

1. The task's sequential execution time (also called the task's size).
2. The task's total overhead, which includes scheduling overhead and communication overhead for the task's inputs and outputs.
3. The task's precedence constraints, which specify the parallelism in the partitioned program.

As mentioned earlier the TLM method is inherently parallel. The transmission delay allows component models to be decoupled for the current time step, enabling parallel calculation of all model functions. A large circuit may be partitioned into sub-circuits which can then be simulated concurrently, i.e. appropriate grouping of components will solve the partitioning problem of parallel TLM simulations. TLM computation requires all component models in the simulation to be executed before progressing to the next time step; thus the task's precedence constraints are easily described. All models simply have to be calculated before the exchange of characteristics between them can happen.

For the scheduling or assignment method to be described later the communication overhead (point 2 above) will be considered independently of any single tasks. It will be calculated as the total time needed for communication by a sub-circuit which is due to be placed on one processor. Hence, for the TLM simulation each model (component) function is considered as one task. The sequential execution time of each model function can be measured on the particular hardware leading to good run-time estimations. The extra time for the calculation of parasitic pressure differences, line impedances, characteristic compensation, etc. (as described in section 4.4.4) can be included depending on the number of ports for each model.

The time needed for the propagation of a certain amount of data between processors can also be measured. With this information, the time needed for the exchange of characteristics between sub-systems on different processors can be estimated. Furthermore, the time for the data exchange between models on the same processor can also be measured.

Assuming these times to be constant for all possible assignments, the total calculation time for a particular assignment can be estimated. This is explained in more detail in sections 5.3.1 and 5.3.2. The calculated execution time is then dependent on the particular partition and schedule of the assignment. It should be mentioned that the presence of the overhead in a multiprocessor can make it impossible to achieve ideal speed-up, i.e. doubling the number of processors does not simply double the calculation speed. Furthermore, it may not be possible to partition a real parallel program into tasks of equal size. Different models have varying complexity and hence different execution times.

5.2.2 Scheduling or assignment of tasks to processors

The scheduling problem (also called the assignment problem) is a fundamental aspect of distributed computing. It arises whenever the procedures or modules of a program are distributed over several interconnected computers so that program activity moves among processors as execution proceeds. The program may be serial on each processor (only one module active on each processor at a time) or parallel (several modules concurrently active on each processor). Here only the former approach is considered, the latter can be of use when a parallel system is shared by more than one user. In general, the assignment problem deals with the question of assigning modules to processors so as to minimise the cost of running a program. The cost may be time, money or some other measure of resource usage [Bokhari, 1987].

For the real-time TLM simulation the scheduling problem is to assign tasks to processors, so as to minimise the parallel execution time. The parallel execution time depends on processor utilisation and on the overhead of inter-processor communication. Minimising the execution time can be done by maximising and balancing the utilisation of resources while minimising the communication between processors [Efe, 1982].

Unfortunately, the scheduling problem becomes more difficult when considering communication overhead with arbitrary data sizes. While minimising inter-processor communication tends to assign all tasks to a single processor, load balancing tries to distribute the program modules (tasks) evenly among the processors. Therefore, there exists conflict between these two criteria and a compromise must be made to obtain an optimal policy for task assignment [Shen & Tsai, 1985]. Furthermore the scheduling problem with communication overhead is NP-complete in the strong sense, even if there is an infinite number of processors available [Sarkar, 1989]. NP-completeness means that a

problem is "inherently intractable" [Garey & Johnson, 1979]. For example consider an extensive search problem with n states that requires 2^n calculations of a particular object function. Furthermore assume that the calculation of each object function requires 1ms CPU-time. A system with $n = 20$ states requires 1049 seconds CPU-time to be calculated whereas a system with $n = 40$ states would take about 35 years. Even with a considerable increase of processing speed the problem cannot be solved in reasonable time and it gets worse for systems with more states.

Timesharing on a single processor has been eminently successful but the scheduling problem becomes more difficult on multiple processors due to frequent context switching[1], necessary for synchronisation and communication. If context switches occur too frequently (e.g. using dynamic partitioning/scheduling), the overhead incurred in task scheduling may well undo the potential speed-up obtained by multiprocessing. The task scheduling problem is different for parallel processing. For tasks within the same parallel program this must address the new problems of task synchronisation and communication, while ignoring some classical considerations from scheduling for timesharing on single processors, like fairness and response time.

Several approaches to task assignment in distributed computing systems (process-processor mapping) have been suggested. They are based on: mathematical programming, graph theory, queuing theory and heuristics. Figure 5.1 shows the taxonomy of static allocation methods. The first three approaches give optimal solutions but are very time consuming, given that the problem is NP-complete.

The automatic placement of processes on processors of a distributed memory parallel machine is a combinatorial optimisation problem which can be reduced to the graph partitioning problem. Again, this problem was shown to be NP-complete [Garey & Johnson, 1979]. Graph theory is a branch of mathematics that has found numerous applications in many different fields. It deals with entities ('nodes'), connections between entities ('edges') and the consequences of these connections. Graph theoretic techniques have been successful in modelling many problems of assignment and partitioning in distributed systems. This is not surprising, since the notions of 'node' and 'edge' from graph theory are very similar to the concepts of 'module' ('task') and 'communication' in

[1] Context switch is a general term covering the situation in which a process initiates a new type of activity.

distributed programs. The idea of partitioning a program is analogous to the concept of partitioning a graph and so on.

There is usually a very clear relationship between a problem and its graph theoretic model. This gives insight into the structures and properties of the problem. This is in contrast to other mathematical modelling techniques where the model is essentially a set of equations of abstract algebraic structures which require deep understanding of mathematics to appreciate [Bokhari, 1987].

5.2.3 Graph theoretic model of the mapping problem

A parallel program is represented by a weighted undirected graph $G_p = (V_p, E_p)$, where $V_p = (v_1, v_2, \ldots v_N)$ is a set of N nodes and $E_p \subseteq V \times V$ is a set of edges. The nodes of the program graph represent sequential program modules (processes) and the edges represent a fixed communication pattern between processes. Node weights a $(a: V \rightarrow Z^+)$ characterise computation costs of the processes and edge weights b $(b: E \rightarrow Z^+)$ characterise communication cost between given pairs of processes located in neighbouring system nodes. For example, a graph with 20 nodes developed from the particular hydraulic circuit in Figure 5.2 is shown in Figure 5.3 (the circuit is described in more detail in section 5.5.2). It is assumed that the weights of the program graph represent worst case properties of the processes, i.e. the time it takes to calculate one timestep with the most complicated equations used in the particular model. The time is given by the number of required clock cycles needed to do the calculation of one time step. Edge weights of the program graph are given by the number of characteristics to be propagated between different models, i.e. the number of transmission lines connecting two models. The models connected by signal lines indicated by 's' (nodes 1 & 2 and 19 & 20) can be combined to form single modules (i.e. they are to be placed on the same processor). For example this leads to the same edge weights for all remaining edges.

A graph is called 'connected' if a path exists between every pair of its nodes. A graph that is not connected (a disconnected graph) has two or more components which are simply smaller constituent graphs that are connected [Bokhari, 1987]. The graph in Figure 5.3 is connected. Disconnected graphs can be used to describe the concurrent simulation of more than one hydraulic circuit. One parallel processing platform may be used to simulate several hydraulic systems in real-time for condition monitoring purposes.

The parallel processing system (Figure 4.1) is represented by a second undirected graph $G_s = (V_s, E_s)$ called a system graph, where V_s represents the nodes of the system and E_s represents the interconnection pattern (a set of edges) of the system (Figure 5.4).

Various assumptions made about the distributed computing system considered here are first described in the following:

1. The processors in the system are homogeneous. That is, a single program module, if executed on different processors, will require the same amount of running time.
2. Identical communication links are used by the processors for message transmission. Direct links between processors are slightly faster than links via the C104 asynchronous packet switch.
3. The link between any two processors is symmetric, i.e. the time to transmit a certain length of message from one processor to another is identical to that to transmit the same message in the reverse direction.

The interconnection pattern of the system graph describes the time it takes to propagate the characteristics of n transmission lines between models via the particular connection. These times are dependent on the hardware and measurements with example simulations show they can be expressed by three different times. Communication on the same processor leads to the fastest propagation, $t_1 \cdot n$. For the inter-processor communication the time consists of an initialisation time (to establish the communication link) and the actual time for the data transfer. The initialisation time is assumed to be independent of the amount of data transferred. Communication via the C104 switch leads to only slightly longer times (compare section 5.3.2). The times are split into two parts, one from a processor to the C104 and another from this switch to the next processor.

Let θ be a mapping function from the vertex set of the program graph to the vertex set of the system graph and Θ the set of all mapping functions, i.e. $\Theta = \{\theta: V_p \rightarrow V_s\}$. The problem of static mapping can then be formulated as the problem of seeking a mapping function $\theta \in \Theta$ that minimises the total computation time. Each graph match then corresponds to a specific task assignment. The above described graph theoretic model illustrates the mapping problem in a very clear form. If will be useful for the understanding of the fitness function described later.

5.2.4 Heuristic and approximate static allocation methods

All optimal allocation methods are very time consuming, hence, in order to speed up the search, approximate algorithms have been used. Approximate solutions to the problem can be obtained by using heuristics. They may be divided in two classes: greedy and iterative. The greedy algorithms are initialised by a partial solution and search to extend this solution until a complete mapping is achieved. At each step, one process assignment is done and it is not possible to change this decision in the remaining steps. Iterative algorithms are initialised by a complete mapping and search to improve it by moving a process to another processor or by exchanging the mapping of two or more processes. Iterative heuristics may be divided in two main classes. On one hand, general purpose optimisation algorithms independent of the given application and, on the other hand, heuristic approaches especially designed for the mapping problem.

Two widely-used optimisation techniques are the hill-climbing algorithm and simulated annealing. Hill-climbing is sure to find the global minimum only in convex spaces. Otherwise, most often it is a local instead of a global minimum which is found. Simulated annealing offers a way to overcome this major drawback of hill-climbing but the price to pay is a huge computation time. Worst, as the simulated annealing algorithm is of a sequential nature, its parallelisation is quite a difficult task [Muntean & Talbi, 1991].

More inherently parallel distributed optimisation techniques may also be considered. Some of them are closely related to neural network algorithms [Ackley, 1987]. The training of a neural network (NN) with many different partitions of parallel hydraulic TLM simulations appears as an intractable complex task. For the NN training data each circuit needs to be simulated and timed several times in order to establish optimal or sub-optimal partitions. It seems to be impossible to train a NN with this (few) known good partitions for the many different hydraulic systems likely to be simulated. Genetic algorithms (GA) are investigated here because they offer the advantage of finding a good solution for any circuit and they can be enhanced using heuristics (specific knowledge about hydraulic systems) as described later. The most remarkable feature of the genetic approach is its generality as a highly powerful and flexible tool.

5.3 Genetic algorithm solutions to the mapping problem

Genetic algorithms (GAs) are motivated by the theory of evolution and date back to the early work of Rechenberg [1973] and Holland [1975]. Recently, GAs have been

successfully applied to various optimisation problems. Several approaches for solving partitioning problems using GAs are considered by Jones & Beltramo [1991]. Curatelli [1995] investigates GAs for graph partitioning where nodes are separated into two groups with a minimum number of edges connecting nodes between different groups and with roughly the same number of elements for each group. Graph theoretical problems have also been tackled using GAs (Pirkul & Rolland [1994], Von Laszewski & Mühlenbein [1990]).

The scheduling problem has also been solved with GAs. Benten & Sait [1994] found GAs to be very effective in determining the minimal time of completion compared to optimal schedules. However when Glass et al. [1994] compared simulated annealing (SA) and taboo search with GAs, the GA on its own led to poor results. But the incorporation of neighbourhood search into the GA improves its performance substantially. This indicates that sometimes the GA needs to be improved using other methods or heuristics. Hou et al. [1994] also investigate GAs for the scheduling problem based on a deterministic model, i.e. execution time and relationship between tasks is known.

Recently, some papers have dealt with the process-processor mapping problem. Muntean & Talbi [1991] use a simplified cost function and they only look at cases where the communication cost between processes is relatively small compared with the computation cost. This is not the case for hydraulic TLM simulations. Hurley [1993], on the other hand, investigates different cost functions where parallelism in communication is ignored. Neuhaus [1990] investigates the mapping of asynchronous processes to an arbitrary parallel architecture but fails to give any practical results or example solutions. In a relatively early paper by Mühlenbein et al. [1987] the assignment of processes to processors is reduced to the graph partitioning problem and solved by an evolution method derived from biology. In their paper only systems where all processes have the same size are considered.

Genetic algorithms differ from traditional optimisation methods in the following ways [Goldberg, 1989]: GAs use a coding of the parameter set rather than the parameters themselves. Unlike local search methods such as simulated annealing and taboo search which manipulate a single feasible solution, genetic algorithms consider a population of feasible solutions, known as chromosomes. Furthermore GAs use probabilistic transition rules, i.e. new solutions are calculated from parts of previous solutions using different probabilities based on random numbers.

After an initial population of solutions has been constructed at random (or by using problem specific information), the fitness of each chromosome in the population is calculated (a high fitness would indicate a better solution than a low fitness value). The fitness is evaluated as the reciprocal of the cost function. Cost functions (also called object functions) are to be minimised. The fitter chromosomes are then selected to undergo transformation to produce offspring for the next generation, which inherit the best characteristics of both parents. After a user-defined number of generations, the result is normally a population that is substantially fitter than the original. The main transformation operators used in GAs are crossover and mutation.

In general a genetic algorithm contains the following steps:

1. Initialisation: generation of a population of random individuals
2. Evaluation: calculate the fitness of each individual of the whole population
3. Selection: select the fittest individuals
4. Replacement: form a new population by replacing worst individuals by best ones
5. Reproduction: apply genetic operators to selected pairs

Steps 2 to 5 are repeated until the maximum number of generations is reached or until the algorithm converges. The construction of a genetic algorithm for any problem can be separated into four distinct and yet related tasks [Hou et al., 1994].

1. the choice of the representation of the strings,
2. the design of the genetic operators,
3. the determination of the fitness/cost function, and
4. the determination of the probabilities controlling the genetic operators

Some of the characteristics of the GA developed for the example systems documented in this thesis are briefly described in the following sections. The fitness function is explained in more detail in sections 5.3.1 and 5.3.2.

String representation:

In this work the partitioning onto four and eight processors is considered. Hence, for each component/model the processor number it is assigned to can be represented by binary strings of two and three bits length, respectively. For the example system with 18 components this binary string can be combined to form chromosomes of 36 and 54 bits

length, respectively. If one likes to partition a system onto a different number of processors that cannot be expressed by a power of 2 then the GA needs to be altered. Using e.g. 3 bits and restricting the processor numbers from 1 to 5,6 or 7 would lead to different probabilities for some of the processors. Without these restrictions the GA needs to reject values larger than the number of available processors. This also requires some recalculations which can lead to longer run times. Another option is to use string representations not based on binary strings. These options are not further investigated in this thesis.

Genetic operators crossover and mutation:

Crossover between parent binary strings occurs with a probability P_c (typically between 0.6 and 0.8). If crossover is to take place, the actual position is randomly chosen. If for example, crossover is to occur after the seventh bit:

```
Parent A 1100101101  Child A 1100101111
                ↑                   ↑
Parent B 1010110111  Child B 1010110101
                ↑                   ↑
```

This scheme was extended to multi-point crossovers as described in Donne et al. [1995]. The GA mutates one bit of the binary string (its value is changed from 1 to 0 or vice versa) with probability P_m (typically between 0.001 and 0.01).

Subpopulations and migration:

To reduce the chance of convergence to a local minimum, each generation is divided into several smaller independent subpopulations. In a process called migration, poorly performing individuals in a subpopulation are periodically replaced by fit individuals from another. The use of different subpopulations leads to an inherently parallel GA, even though so far this has been implemented as a serial GA only. In the GA used to calculate the partition for the example circuits, 12 subpopulations were used and 4 individuals were migrated every third generation.

5.3.1 The fitness/cost function

Most assignment methods (not only GAs) adopt some type of cost function to evaluate the effectiveness of task assignment algorithms. The most commonly used cost function is defined as the sum of the inter-processor communication cost and the processing cost. The cost function used is described in the following section.

After a particular simulation code is partitioned into suitable modules, let $t_p^e(\theta)$ denote the total time spent for execution of all models, and $t_p^c(\theta)$ be the total time for inter-processor communication, both in some processor (partition) p according to a certain task assignment θ. Let

$$t_p(\theta) = t_p^e(\theta) + t_p^c(\theta) \tag{5.1}$$

which is the total time spent in processor p for task assignment θ. We call $t_p(\theta)$ the processor turnaround time of p. This turnaround time is in general different for each distinct processor. Let

$$t(\theta) = \max_p t_p(\theta) \tag{5.2}$$

which can be called the task turnaround time of θ. Sometimes this is also called the parallel run time. It is easy to see that $t(\theta)$ is the total time required to complete all tasks according to assignment θ under the assumption of negligible processor idleness. Therefore, $t(\theta)$ is, from the point of reducing total processing time, used as a cost measure for the effectiveness of task assignment θ. The smaller $t(\theta)$ is, the better θ is. An optimal task assignment thus may be defined as the one θ_0 which minimises the task turnaround time $t(\theta)$, i.e.,

$$t(\theta_0) = \min_{\Theta} t(\theta) = \min_{\Theta} \max_p t_p(\theta) \tag{5.3}$$

where Θ is the set of all possible assignments. This means that we want to minimise the maximum processor turnaround time, resulting in the so-called minimax criterion. $t(\theta_0)$ is called the minimum task turnaround time [Shen & Tsai, 1985].

5.3.2 How to calculate the fitness function

The task turnaround time for task assignment θ in equation (5.2) can be calculated using the vectors and matrices defined by the following:

Assignment vector *A*:

$$A = A(m) \tag{5.4}$$

where m equals the number of component models (number of nodes in the undirected graph). A describes the particular task assignment θ and it can contain values between 1

and the number of processors used for a particular simulation, e.g. $A(1)=2$, $A(2)=5$ means the models encountered firstly and secondly in the system link file (see Pollmeier [1996c]) are placed on processor 2 and 5, respectively.

Connection matrix *B*:

$$B = B(m,m) \tag{5.5}$$

(with m as described above) is a symmetric matrix and specifies the number of connections (transmission lines) between components, i.e. $B(4,1)=B(1,4)=1$ indicates one connection between model 1 and 4. There can be more than one line between components, for example if a flow divider is connected to a proportional valve. Figure 5.5 shows the connection matrix derived from the example graph in Figure 5.3. Here nodes 1 & 2 and nodes 19 & 20 are compound as described earlier and all nodes are renumbered from 1 to 18.

Communication time matrix *C*:

$$C_{1,2} = C_{1,2}(n_c, n_c) \tag{5.6}$$

where n_c is the number of processors. These matrices are shown in Figure 5.6. They give the times for communication between processors obtained by experiments with the particular T9000-based parallel platform, i.e. $n_c = 8$. The notation is chosen according to the system graph in Figure 5.4. C_2 describes the initialisation time and C_1 represents the time needed for the data transfer of one transmission line. All times are given in the number of clock cycles required.

Calculation times vector *D*:

$$D = D(m) \tag{5.7}$$

Again m represents the number of nodes in the undirected graph. D contains the number of clock cycles needed to calculate one timestep of a model (or group of models). The numbers correspond to the node weights in the particular circuit graph. Figure 5.7 shows D for the example graph in Figure 5.3.

Finally, the task turnaround time for a particular assignment can be calculated. Figure 5.8 shows part of the FORTRAN 77 code used to calculate the fitness function. The turnaround time is returned as the variable called *obtotal*.

5.3.3 Software support

A program called 'l2ga' (link file to GA) has been developed that creates the above-described matrices *B* and *D* automatically. Using the user-defined system link file (as described in section 4.4.2 and in Pollmeier [1996c]) the necessary information is automatically extracted. Furthermore, heuristics can be accommodated by specifying certain models that must be placed on the same processor. For example, the components connected by signal links as seen in Figure 5.3 have to be placed on the same processor. The program automatically combines these models and it also calculates the new node weights for the combined groups of models.

5.4 Deadlock-free communication scheme

Deadlock will arise when members of a group of processors which hold resources are blocked indefinitely from access to resources held by other processes within the group [Hwang & Briggs, 1987]. According to Welch & Peel [1994] there are three possible ways of reaching deadlock:

Primitive deadlock:

This happens when a process executes the stop and refuses any further communication with its environment.

Structured deadlock:

This is the more common design fault. Assuming a sub-system of two processes called P1 and P2, if process P1 commits to its output at the same time as process p2, this sub-system refuses any further communication with its environment. In other words, if both processes have to communicate with each other and both try to send data first this will lead to a deadlock. (See section 4.4.5 for details about the features of the channel connection used for communication and synchronisation between processors.)

Primitive livelock:

This happens when a process commits itself to an infinite loop, hence, it refuses any further communication with its environment. Livelock is a less common design fault, and is usually caused by mis-programmed loops that fail to terminate.

To their environment, deadlocked and livelocked systems seem the same in that they do not respond. To the processor, a deadlocked system imposes zero load whereas a livelocked system imposes continuous demands [Welch & Peel, 1994]. Exhaustive testing

to prevent deadlock is usually impossible. The verification would involve exhaustive "case-by-case" check through all "categories" of state. Furthermore, the choice of "categories" for the above reasoning requires some knowledge. For well-engineered designs the above choices should be "obvious".

Many researchers have investigated the detection and avoidance of deadlock situations, e.g. Spirakis [1986], Beauquier et al. [1991] and Zhong & Lo [1992]. For the parallel TLM simulation deadlock-free simulations can be guaranteed if communication always follows a certain pattern. A deadlock-free method has been designed particularly for the TLM simulation and is described in the following section.

With the information stored in the *D* vector (described in section 5.3.2) the estimated total time spent for execution of all models can be calculated for every processor/partition. This relies on reasonably accurate estimates. The partitions can then be named in an increasing order, i.e. the partition expected to finish calculation first may be called partition 1 and the one expected to finish second may be called partition 2, etc. up to partition 8 with the highest computational load:

partition 1	lowest computational load
partition 2	second lowest computational load
⋮	⋮
partition 8	highest computational load

If some of the partitions have equal execution times the order of these partitions is arbitrary. Partition 1 is the first to finish with its calculations for the current time step and thus it can initialise communication first. This partition should then first communicate with partition 2, then it should communicate with partition 3, etc. (if there is any data exchange between the respective two partitions).

After partition 2 finishes its communication with partition 1 it should immediately communicate with partition 3 and afterwards with partition 4, etc. The data exchange between two independent pairs of partitions/processors (e.g. partitions 1 & 4 and 2 & 3) can be done concurrently, hence up to four processor pairs can communicate independently at the same time. This is an important feature of the T9000-based platform where the C104 asynchronous packet switch enables concurrent operation.

Tests show that initialising the communication from the less-loaded partition leads to the most time efficient implementation. The data exchange between two partitions is always in both directions (bi-directional) using two consecutive channel communications as

described in section 4.4.5. The scheme described here is guaranteed to be deadlock-free and it is automatically implemented by the program generator described in section 4.4.1.

5.5 Partitioning examples

5.5.1 Performance measures of parallel systems

In this section, some metrics commonly used to measure the performance of parallel systems are introduced. The execution time of a parallel computer depends not only on input size but also on the architecture of the parallel computer and the number of processors. Hence, a parallel algorithm cannot be evaluated in isolation from a parallel architecture [Kumar et al. 1994].

Run time

The serial run time 'T_S' of a program is the time elapsed between the beginning and the end of its execution on a sequential computer using the fastest known sequential algorithm. The parallel run time 'T_P' is the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

Speedup

One is often interested in knowing how much performance gain is achieved by parallelising a given application over a sequential implementation. Speedup 'S' describes the relative benefit of solving a problem in parallel and it is defined by

$$S = \frac{T_S}{T_P} \tag{5.8}$$

This definition assumes p identical processors for the parallel implementation. The maximum speedup (also called linear speedup) is given by the number of processors. However, in practice the speedup achieved is often less than this due to for example communication overheads.

Efficiency

Efficiency is a measure of the fraction of time for which a processor is usefully employed, as it is defined as the ratio of speedup to the number of processors 'p' [Kumar et al. 1994]

$$E = \frac{S}{p} \tag{5.9}$$

Cost

The cost of solving a problem on a parallel system is the product of parallel runtime and the number of processors used. A parallel system is cost-optimal if the cost of solving a problem on a parallel computer is proportional to the execution time of the fastest known sequential algorithm on a single processor.

Scalability

The scalability of a parallel system is a measure of its capacity to increase speedup in proportion to the number of processors. It reflects a parallel system's ability to utilise increasing processing resources effectively. For a given problem instance, the speedup does not increase linearly as the number of processors increase. The speedup tends to become saturated and the speedup curve flattens. A larger instance of the same problem yields higher speedup and efficiency for the same number of processors, although speedup and efficiency continue to drop with increasing number of processors. Given that increasing the number of processors reduces efficiency and that increasing the size of the computation increases efficiency, it should be possible to keep the efficiency fixed by increasing both the size of the problem and the number of processors simultaneously. The size of a problem can be expressed in terms of the total number of basic operations (basic computation steps) required to solve the specific problem.

A scaleable parallel system is one in which the efficiency can be kept fixed as the number of processors is increased, provided that the problem size is also increased. For different parallel systems, the problem size must increase at different rates in order to maintain a fixed efficiency as the number of processors is increased. This rate determines the degree of scalability of the parallel system [Kumar et al. 1994]. A parallel system is highly scaleable if small increments in the problem size are sufficient for the efficient utilisation of an increasing number of processors.

5.5.2 Small-scale partitioning example

The hydrostatic transmission in Figure 5.2 was chosen as a small-scale example system. A variable displacement pump drives a motor on which a predefined changing load acts. The boost pump supplies the circuit with oil from the reservoir.

Some simulation results are given in Figure 5.9. The main pump displacement is linearly increased from 0 to 100 cm^3 in the first 0.6 seconds of the simulation. After 0.15 seconds

the motor load torque is also increased. The load duty cycle is given in Figure 5.9. Furthermore, the corresponding motor torque, motor shaft speed and the mechanical efficiency of the motor are given. Due to stiction effects, the motor only starts moving after about 0.1 seconds. Once the motor reaches its desired speed of about 1000 rpm this speed only drops slightly when the load torque increases.

The hydraulic circuit in Figure 5.2 was partitioned using the above-described GA with the probabilities shown in Table 5.1 [Donne, 1993]. For this study, the number of individuals in each subpopulation was set to 60.

Figures 5.10 and 5.11 show the cost function (parallel run time in CPU clock cycles per time step) over the number of populations for the partitioning onto 4 and 8 processors, respectively. On the left hand sides the average of 10 runs are shown for some different crossovers. For the partitioning onto 4 processors the chromosome (string) length is 36 bits; 1 crossover led to the best results. After about 75 to 100 generations further generations improve the task turnaround time only marginally. On the other hand, for the partitioning onto 8 processors (54 bits string length) 2 crossovers led to the best results. Here good results are obtained after about 150 to 200 generations. Hence, doubling the number of processors requires twice as many generation evaluations although the string length is only increased by 50 percent. Arriving at a general guideline for the optimum number of crossovers proved difficult but one crossover point every 20 to 30 bits was found to work well.

On the right hand sides of Figures 5.10 and 5.11 the minimum, maximum and average cost function of 10 GA runs are shown for 1 and 2 crossovers, respectively. It can be seen that the difference between minimum and maximum cost function is slightly larger for the 8 processor partition. This might depend on the different string lengths and the relatively small model calculation times. The latter are in the same order of magnitude as the communication times required for inter-processor communication. Furthermore, as one would expect, the 8 processor partition leads to smaller task turnaround times. Additionally, the cost function starts at a smaller value, i.e. a random distribution of the components onto 8 processors leads to better mappings than placing the same components randomly onto 4 processors.

In this research a near optimum solution is sought which does not need to be the absolute minimum, as long as real time performance is achieved. Because the run-times of the

different models are measured for the worst case, optimal parallel performance would depend on system inputs and is not achievable in the general case.

Three good partitions onto both 4 and 8 processors are shown in Figures 5.12 and 5.13, respectively. The eight processors are named from P1 to P8. An experienced user or system designer might be able to find similar solutions for the 4 processor partition but an increasing number of components and processors would make the search very tedious and time consuming, i.e. it would require several test runs of the parallel simulations. In Figure 5.12 mapping A and mapping B show exactly the same partitions which are only placed on different processors. An extensive search of all possible partitions was carried out and both mappings were found to be optimal. Further tests showed that the best GA results were always optimal for the partitioning of small systems onto up to 4 processors where an extensive search was possible. This assumes the calculation of a sufficient number of generations. The GA needed less than 5 minutes for the calculation of 200 generations on a SUN 20.

5.5.3 Small-scale example: Simulation performance

The system simulation was implemented on the T9000-based transputer platform using the above-described deadlock-free scheme. In order to measure the speedup the simulation was run on a single processor and then with the partitions given in Figure 5.12 and Figure 5.13 on 4 and 8 processors, respectively. A speed-up of 2.6 was achieved with the 4 processor simulation and this was improved to a speed up of 3.6 for the implementation on 8 processors. This corresponds to efficiencies of 0.65 and 0.45, respectively, i.e. the simulation on 4 processors leads to a better efficiency. Real-time performance was possible on a single processor with a time step of 1.1 ms but for accurate simulation the system dynamics require smaller time steps. Time steps of 41 μs and 30 μs are possible for real-time simulations of the 4 and 8 processor simulation, respectively. These time steps are sufficiently small for accurate simulations and they lead to results as described above and presented in Figure 5.9.

5.5.4 Medium-scale partitioning example

Here two of the hydraulic transmissions described in section 5.5.2 (Figure 5.2) were simulated concurrently. This could be used for the simultaneous simulation of two independent systems. Using the same circuit makes it possible to compare mappings of

this larger system with the optimal solution obtained from the partitioning of the small-scale system. The combination of the two circuits leads to 40 component models. Models connected by signal lines were again pre-grouped together and placed onto the same processor. All settings of the genetic partitioning scheme were left as described earlier, i.e. 12 subpopulations with 60 individuals were used. Results are given in Figure 5.14 where the cost function is shown over the number of populations for the partitioning onto 4 and 8 processors, respectively. The maximum, average and minimum of 10 runs is given for both mappings. Three crossovers were used for the mapping onto 4 processors where 36 components (40 minus 4 pre-grouped components) lead to a string length of 72 bits. After about 150 to 200 generations the cost function decreases only marginally by further generations. Hence, compared with the partitioning of the small-scale example (Figure 5.10) about twice as many generations needed to be calculated. The 8-processor mapping was calculated with 4 crossovers and the respective string length was 108 bits. Again the number of generations had to be doubled in order to achieve good results.

For the combined circuits the component run times were also measured and they turned out to be slightly larger then the respective values for the single hydrostatic transmission. Hence, the absolute value of the cost function cannot be compared directly. Figure 5.15 shows a near-optimal mapping onto four processors (P1 to P4) where both hydraulic transmissions are partitioned onto a separate set of processors. Furthermore, both circuits are partitioned identically. The solution in Figure 5.15 is only 2 percent slower than the optimum mapping found by calculating all possible mappings.

This ideal process-processor mapping is given in Figure 5.16. Increasing the number of generations to 300 the GA also found this optimum. The mapping onto 8 processors leads to results slightly inferior to the mapping of one circuit onto 4 processors as described in the previous section. Nevertheless, the GA automatically places both circuits onto different sets of processors. For the partitioning of a real system containing two similar circuits the group of processors would be restricted for each of the circuits. For example, the top circuit would be placed onto processors P1 to P4 whereas the bottom circuit would be placed onto the remaining processors.

5.5.5 Medium-scale example: Simulation performance

Parallel simulations of different partitions were again implemented on the T9000-based transputer platform. For reference, the simulation was again run on a single processor.

Using the mappings onto 4 processors shown in Figures 5.15 and 5.16 a speedup of up to 4.25 was achieved (corresponding to an efficiency larger than 1.0). This speedup is larger than the linear speedup which is given by the number of processors. In this example a phenomenon known as superlinear speedup is observed. This is due to hardware characteristics that put the sequential algorithm at a disadvantage. The data for the problem are too large to fit into the main memory (cache) of a single processor, thereby degrading its performance due to the use of secondary storage. On the transputer platform each of the processors contains 16 Kbytes of cache which is the maximum amount of data that can be accessed very swiftly. When partitioned among several processors, the individual data-partitions are small enough to fit into their respective processors' main memories. The performance gained due to the extra memory is large enough to compensate for the additional communication overhead. A speedup of up to 5.9 was achieved for the partitioning onto 8 processors. In this case the additional communication overhead leads to an efficiency smaller than one (0.74).

5.5.6 Large-scale partitioning example

A complex hydraulic system is detailed schematically in Figure 5.17. This circuit layout is typical of a dual-channel safety system, but is used here principally to test the performance of the partitioning scheme on large systems. The hydraulic circuit features two boosted, closed-loop hydrostatic transmissions which are mechanically cross-linked, and also have cross-linking pressure relief valves. Duplication of some of the main components in this circuit leads to a certain amount of fault tolerance due to redundancy in the system. The complete circuit contains 70 components connected by 80 transmission lines. Components encircled by dotted lines are connected by signal lines and they are pre-grouped in order to be placed onto the same processor. This reduces the number of tasks to be partitioned to 64. The above-described GA is again used to partition the system onto 4 and 8 processors using 5 and 7 crossovers, respectively. Figure 5.18 shows the maximum, average and minimum of the cost function again using 10 runs against the number of generations. The calculation of about 300 generations are required for the mapping onto 4 processors. This increases to about 500 for the mapping onto 8 processors. Two partitions onto 4 processors with similar cost function values (16739 and 16686) are given in Figure 5.19. These mappings and several similar mappings were found in different runs of the GA. This shows the repeatability of the mapping algorithm, i.e. GAs differently initialised by random partitions evolve similar efficient mappings.

5.5.7 Large-scale example: Simulation performance

Both mappings shown in Figure 5.19 were implemented as parallel simulations on 4 processors using arbitrary duty cycles. Again a superlinear speedup of 4.24 is observed for both systems. Running the system on 8 processors with mappings obtained from the same GA leads to a speedup of up to 7.8. This almost linear speedup is equivalent to an efficiency of 0.98.

5.5.8 Cost and scalability

The achieved runtimes, speedups and efficiencies of the three example systems are summarised in Table 5.2. Additionally, the cost (defined in section 5.5.1) has also been calculated and displayed. The size of the medium-scale example is twice the size of the small-scale example. An efficiency of 0.65 was obtained for the simulation of the small-scale system on 4 processors. On the other hand the respective simulation of the medium-scale system on 8 processors leads to an efficiency of 0.74. This means doubling the problem size and doubling the number of processors leads to a better efficiency, i.e. the parallel TLM system is highly scaleable on the T9000-based platform. This indicates the suitability of the hardware for the particular fine grained TLM simulations. Comparing the large-scale example with the smaller examples also leads to good scalability.

A cost-optimal parallel system has an efficiency of O(1), i.e. the parallel TLM simulation is cost-optimal for the partitioning onto 4 processors. The efficiency of the 8 processor simulation is increasing with problem size, i.e. in this case the system is also very cost-effective.

5.6 Simple heuristics to improve GA performance

In order to get reasonably good mappings the GA needs to run for a certain number of generations. In general it is found that the number of generations needs to be increased about linearly with the number of components and also with the number of processors. This can lead to long calculation times of the GA for large mapping problems. For example the calculation of 400 generations for the partitioning of the large scale system onto 4 processors takes about 69 minutes on a SUN20. With increasing problem size the difference between minimum and maximum of the cost function also increases. Hence in order to achieve sufficiently good mappings for time critical applications the GA needs to

be run several times. The following section introduces some simple heuristics that can reduce the GA runtimes considerably.

5.6.1 Pre-grouping of components

All partitions shown so far indicate that the GA mapping scheme tries to map connected components together onto the same processor in order to reduce inter-processor communication. This can be used as a heuristic in order to speed up the GA. For example in Figure 5.19 the tank model and its connected component are always grouped and then placed onto the same processor. In general one can place any component that is only connected to one other component onto the same processor. The so received group of components can again be pre-grouped if it is connected to one other component only. Furthermore, groups of components that are likely to be assigned to the same processor can be pre-grouped as well. Figure 5.20 illustrates this concepts for two different pre-groupings where groups of components are identified by a shaded box.

On the left hand side all tank models are pre-grouped with their adjacent components. Additionally, components 12 & 13 are combined with component 11 and components 25 & 26 are grouped in the same way with component 24. On the right hand side the pre-grouping is extended to the filter components with their adjacent node and relief valve models. Using this grouping reduces the number of tasks to be partitioned to 58 and 46, respectively. Here it is assumed that components linked by signal lines are also grouped together. The number of tasks can be reduced even further by grouping some of the transmission components numbered 27 to 39 and similarly 42 to 54. On the other hand, this might reduce the quality of achievable mappings, i.e. the pre-grouped system cannot be partitioned as efficiently as the system without pre-grouping. In general it is found that the larger the problem size the more pre-groupings that can be assigned without reducing the final mapping quality. This is partly because the run times for the different models are the same order of magnitude. If at a later stage one develops very sophisticated models of certain components then these should not be grouped together with any other components.

5.6.2 Simulation results

The GA partitioning algorithm was applied to both pre-grouped systems using 5 and 4 crossovers for mapping onto 4 processors. 7 and 6 crossovers were used for the mapping onto 8 processors. The larger number of crossovers was used each time for the mapping of pre-grouping 1. Figure 5.21 shows the average cost function (average of 10 runs) over the

number of generations. For comparison the previously-obtained results are also shown where only components linked by signal lines are grouped together. This case is described in section 5.5.6 and here it is called 'grouping 0'. For pre-grouping 1 the GA finds good mappings with fewer generations than for the standard case. The number of generations required is even smaller for pre-grouping 2. Additionally, the pre-groupings reduce the number of tasks and hence the chromosome string length used by the GA. This leads to a considerable reduction in calculation time. For pre-grouping 2 the calculation time is reduced to less than 60 percent of the time it takes to calculate the standard case. Furthermore, the quality of the mappings is sometimes even improved for the pre-grouped case. One mapping result onto 8 processors is shown in Figure 5.22. This partitioning arrangement leads to a speedup of 7.8 and shows some interesting features. The mapping would not have been possible if certain components of the closed loop transmissions had been grouped together, e.g. components 27 to 29 and 34 to 36. There are more groups of components than there are processors. Hence, sometimes it can be advantageous to put disconnected groups of components onto the same processor in order to improve the balance between processors. This is because the communication overhead is accounted for when load balancing the parallel simulation. Figure 5.23 clarifies this aspect. For two similar good mappings the estimated computation and communication times are given for each of the 8 processors (P1 to P8). Mapping B corresponds to the mapping shown in Figure 5.22. It can clearly be seen that only balancing the computational load would not lead to an efficient simulation. Although the size of the structure used to propagate information between processors is optimised according to the amount of data to be exchanged, the communication time can still be a considerable part of the load on several processors. In Chapter 6 this problem is solved. Figure 5.23 also indicates that different mappings can lead to similar useful results. Mapping A and mapping B lead to about the same speedup.

One could think of another way of improving the GA performance by increasing the number of individuals in each generation. This was investigated and even with 100 individuals in each of the 12 subpopulations this did not lead to any better results.

5.6.3 Runtime validation

The estimated and measured model calculation times are compared in Figure 5.24 in order to validate the runtime estimations. Therefore the time it takes to calculate one time step of all models placed onto one processor is estimated. This time is then expressed in

percentage of the time it takes to calculate one time step of all models together. The same times are measured when running the simulation using Mapping B (Figure 5.22). For the estimation a simplified model of the parallel platform is used, i.e. changes in computation speed due to available cache are not accounted for. Furthermore the runtimes are estimated for the worst case and in a real simulation some of the models might be calculated faster than this. Nevertheless, a very good agreement between estimation and measurement has been found.

5.7 Differences to the TLM implementation in HOPSAN

The simulation package HOPSAN developed at Linköping University (Sweden) is also based on the TLM method and so-called Q-type and C-type components are used. In the Q-type, flows are calculated and the characteristics are adjusted to become the effort variables and in the C-type, characteristics which have the same dimension as the effort variables are calculated. Effort variables include pressure, force and electrical potential. The flow variables on the other hand include such variables as flow, speed and electric current. In order to achieve numerical stability all the Q-type components for a simulation have to be executed before the C-type components (or vice versa) [Krus et al., 1990]. With this arrangement there have to be two data transfers per calculated time step. Firstly all characteristics in the fluid transmission lines are calculated and propagated (C-type). After that components such as pumps and valves are evaluated (Q-type) and the respective pressures and flows are propagated. E.g. for the modelling example in Chapter 3.2.2 equations (3.8) and (3.17) would be calculated by the C-type components and equations (3.14) and (3.16) by the Q-type components. Implementing this scheme onto a parallel processing platform would lead to a higher communication overhead than the method developed in this thesis. Here data is only exchanged once per calculated time step.

Another difference is the programming language. HOPSAN models and simulations are written in FORTRAN whereas the above-described systems is based on the INMOS ANSI C toolset. Furthermore an automatic program generator for a parallel platform was developed here which is not available with HOPSAN.

On the other hand there are several similarities between both implementations, e.g. both packages use double precision and they are based on the same physical equations.

5.8 Closure

In this chapter the process-processor mapping problem has been solved using an automatic method based on genetic algorithms. The GA fulfilled the expectation in general as it computed good solutions to the sample problems. To compensate the influence of probabilistics the algorithm often has to be run more than once (in some cases an additional run brought a much better mapping). This can lead to long runtimes and some heuristics were developed that overcame this problem. The quality of the achieved mappings was very good and led to ideal or near ideal speedups and efficiencies. Communication was found to be a considerable part of the load on the processors. In the next chapter a method is developed which reduces this communication overhead.

TABLES FOR CHAPTER 5

Subpopulation	1	2	3	4	5	6
P_c	0.6	0.5	0.7	0.8	0.6	0.7
P_m	0.05	0.005	0.02	0.001	0.05	0.01

Subpopulation	7	8	9	10	11	12
P_c	0.9	0.8	0.5	0.6	0.7	0.8
P_m	0.005	0.008	0.05	0.02	0.008	0.01

Table 5.1 Crossover and mutation probabilities

Problem size		small	medium	large
Runtime on single processor [s]		4.36	10.31	22.89
Partitioning onto 4 processors	Runtime [s]	1.68	2.43	5.40
	Speedup [-]	2.60	4.24	4.24
	Efficiency [-]	0.65	1.06	1.06
	Cost [s]	6.72	9.72	21.60
Partitioning onto 8 processors	Runtime [s]	1.20	1.75	2.92
	Speedup [-]	3.60	5.90	7.80
	Efficiency [-]	0.45	0.74	0.98
	Cost [s]	9.60	14.00	23.36

Table 5.2 Performance measures

FIGURES FOR CHAPTER 5

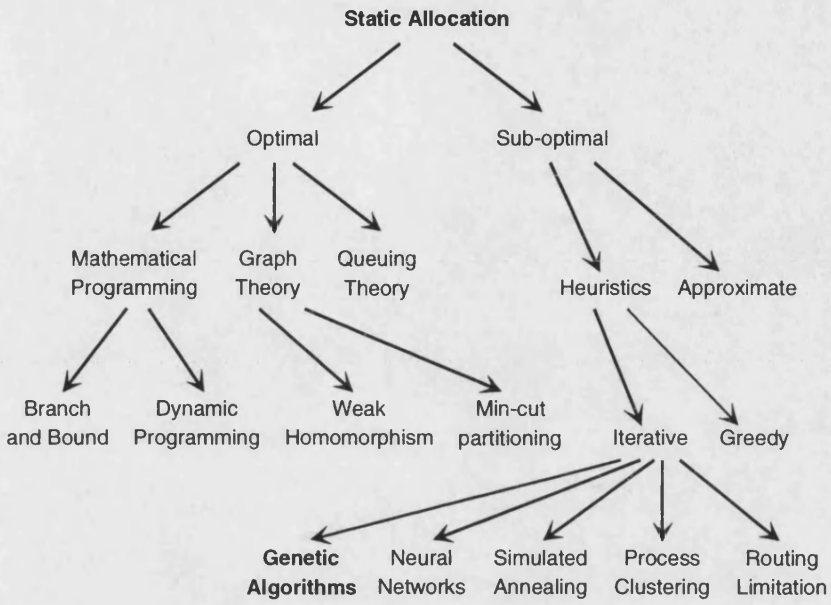


Figure 5.1 A taxonomy of static allocation

[Muntean & Talbi, 1991]

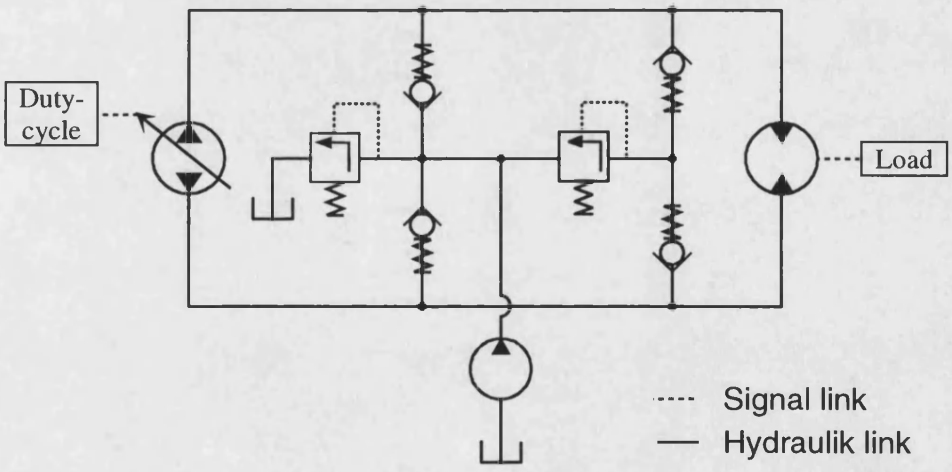


Figure 5.2 Hydraulic circuit for graph partitioning example

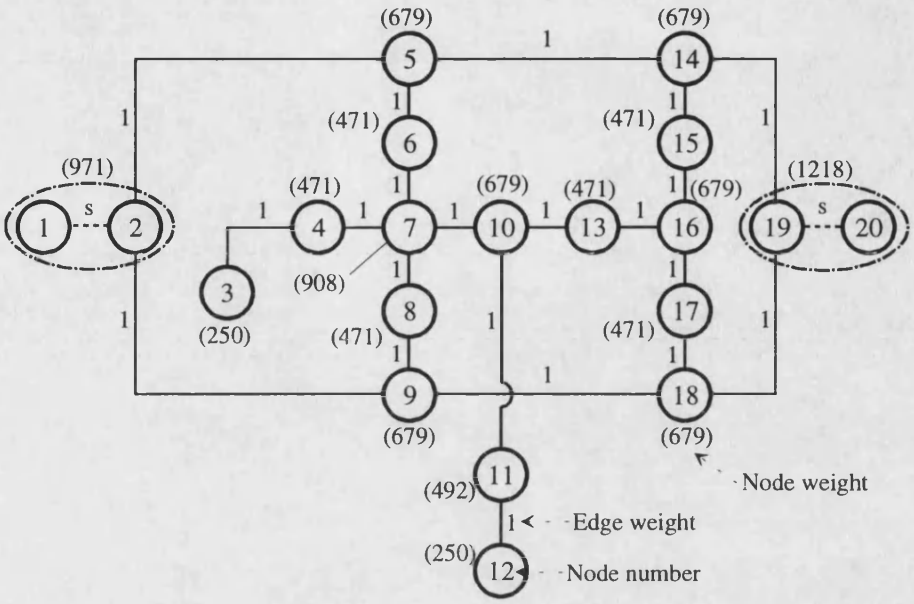


Figure 5.3 A graph with 20 nodes according to the hydraulic circuit in Figure 5.2

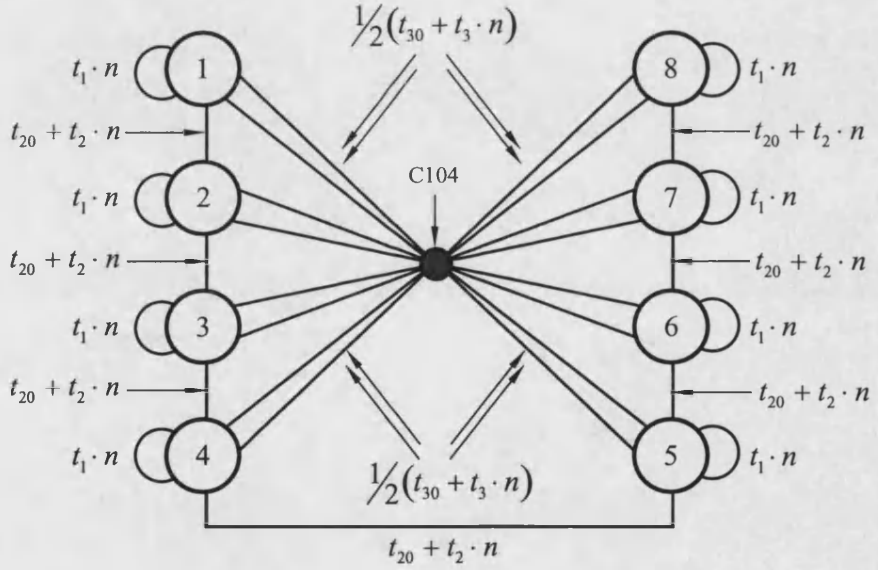


Figure 5.4 System graph of the T9000-based parallel processing platform

(The parameters are defined in Figure 5.6 and in the text)

$$B = \begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

Figure 5.5 Connection matrix *B* for the example graph in Figure 5.3

$$C_{1,2} = \begin{pmatrix}
t_1 & t_2 & t_3 & t_3 & t_3 & t_3 & t_3 & t_3 \\
t_2 & t_1 & t_2 & t_3 & t_3 & t_3 & t_3 & t_3 \\
t_3 & t_2 & t_1 & t_2 & t_3 & t_3 & t_3 & t_3 \\
t_3 & t_3 & t_2 & t_1 & t_2 & t_3 & t_3 & t_3 \\
t_3 & t_3 & t_3 & t_2 & t_1 & t_2 & t_3 & t_3 \\
t_3 & t_3 & t_3 & t_3 & t_2 & t_1 & t_2 & t_3 \\
t_3 & t_3 & t_3 & t_3 & t_3 & t_2 & t_1 & t_2 \\
t_3 & t_3 & t_3 & t_3 & t_3 & t_3 & t_2 & t_1
\end{pmatrix}$$

$$C_1 \begin{cases} t_1 = 3 \\ t_2 = 277 \\ t_3 = 287 \end{cases}$$

$$C_2 \begin{cases} t_1 = 0 \\ t_2 = t_{20} = 432 \\ t_3 = t_{30} = 552 \end{cases}$$

Figure 5.6 Communication time matrices for the system graph in Figure 5.4

$$D = \begin{pmatrix} 971 \\ 250 \\ 471 \\ 679 \\ 471 \\ 908 \\ 471 \\ 679 \\ 679 \\ 492 \\ 250 \\ 471 \\ 679 \\ 471 \\ 679 \\ 471 \\ 679 \\ 1218 \end{pmatrix}$$

Figure 5.7 Calculation time vector *D* for the system in Figure 5.3

```
      obtotal=0.0
      do 100 i=1, NoT
         t(i) = 0
         do 110 j=1, NoC
            if (A(j) .eq. i) then
               t(i) = t(i) + D(j)
               do 120 k=1, NoC
                  if (B(j,k) .ne. 0) then
                     ah = A(k)
                     t(i) = t(i) + B(j,k)*C1(i,ah) + C2(i,ah)
                  endif
120            continue
            endif
110      continue
         if (obtotal .lt. t(i)) then
            obtotal = t(i)
         endif
100   continue
```

NoT - number of transputers (processors)
NoC - number of components (models/nodes)

Figure 5.8 FORTRAN 77 code used to calculate fitness functions

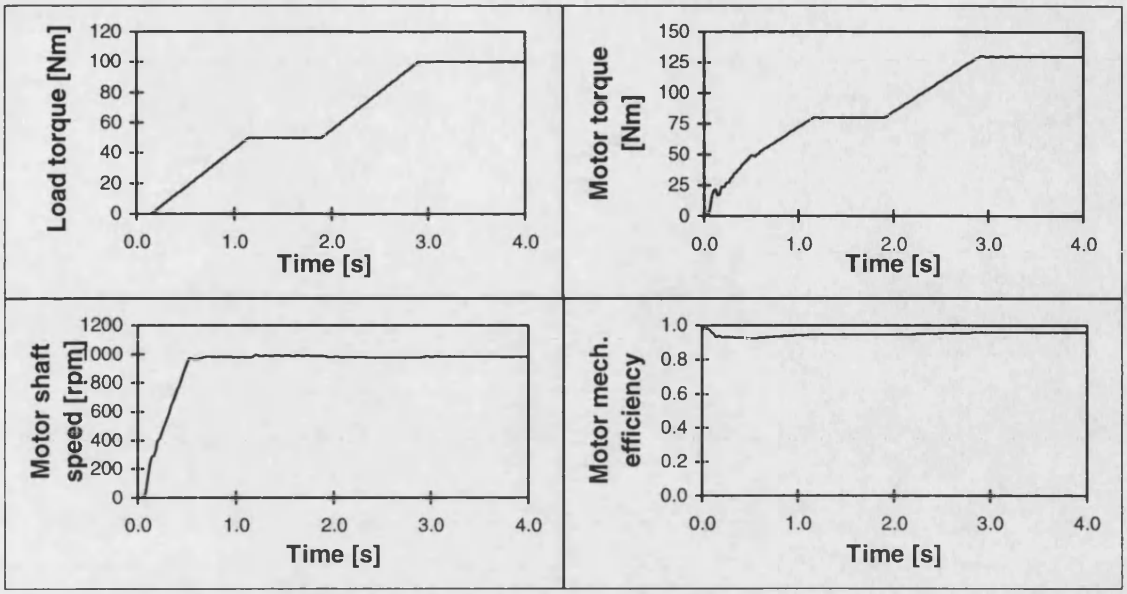


Figure 5.9 Simulation results

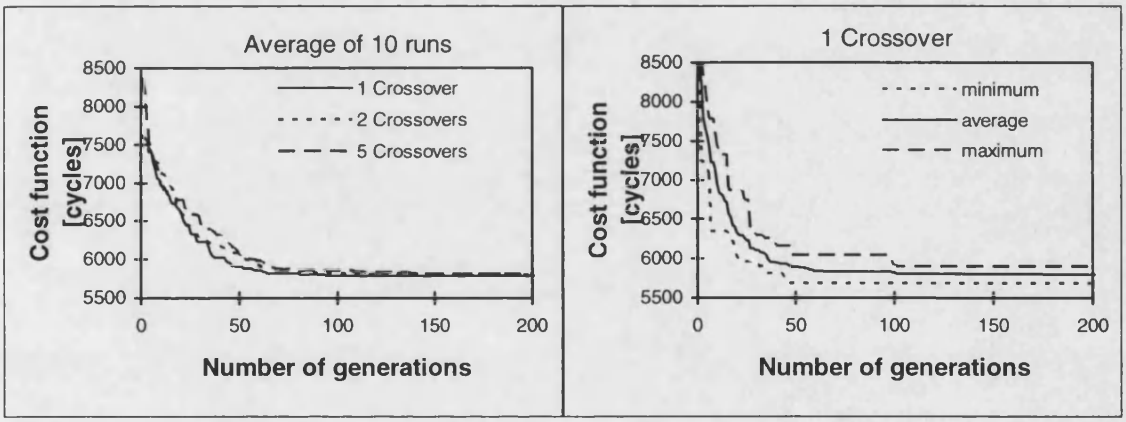


Figure 5.10 GA results for partitioning onto 4 processors (P1 to P4)

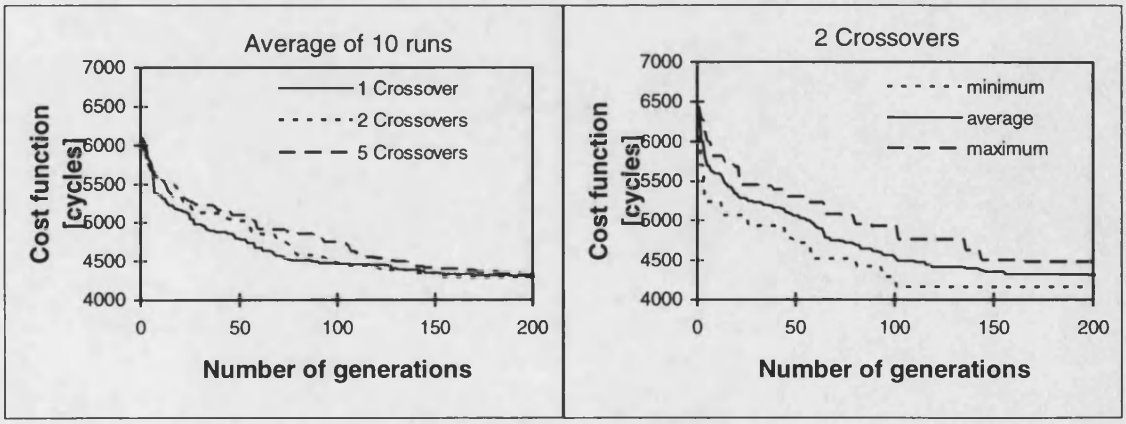


Figure 5.11 GA results for partitioning onto 8 processors (P1 to P8)

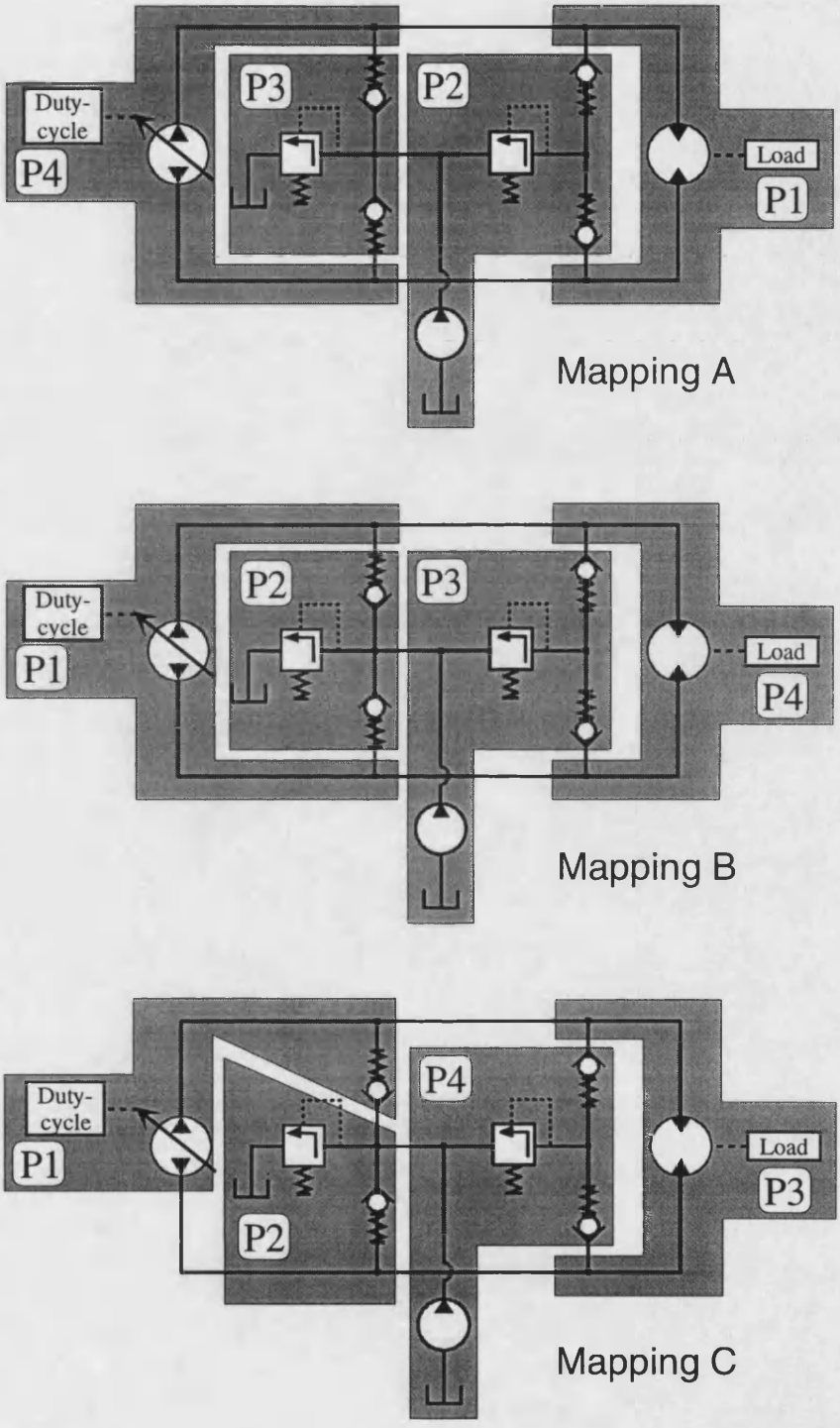


Figure 5.12 Partitioning onto 4 processors (P1 to P4)

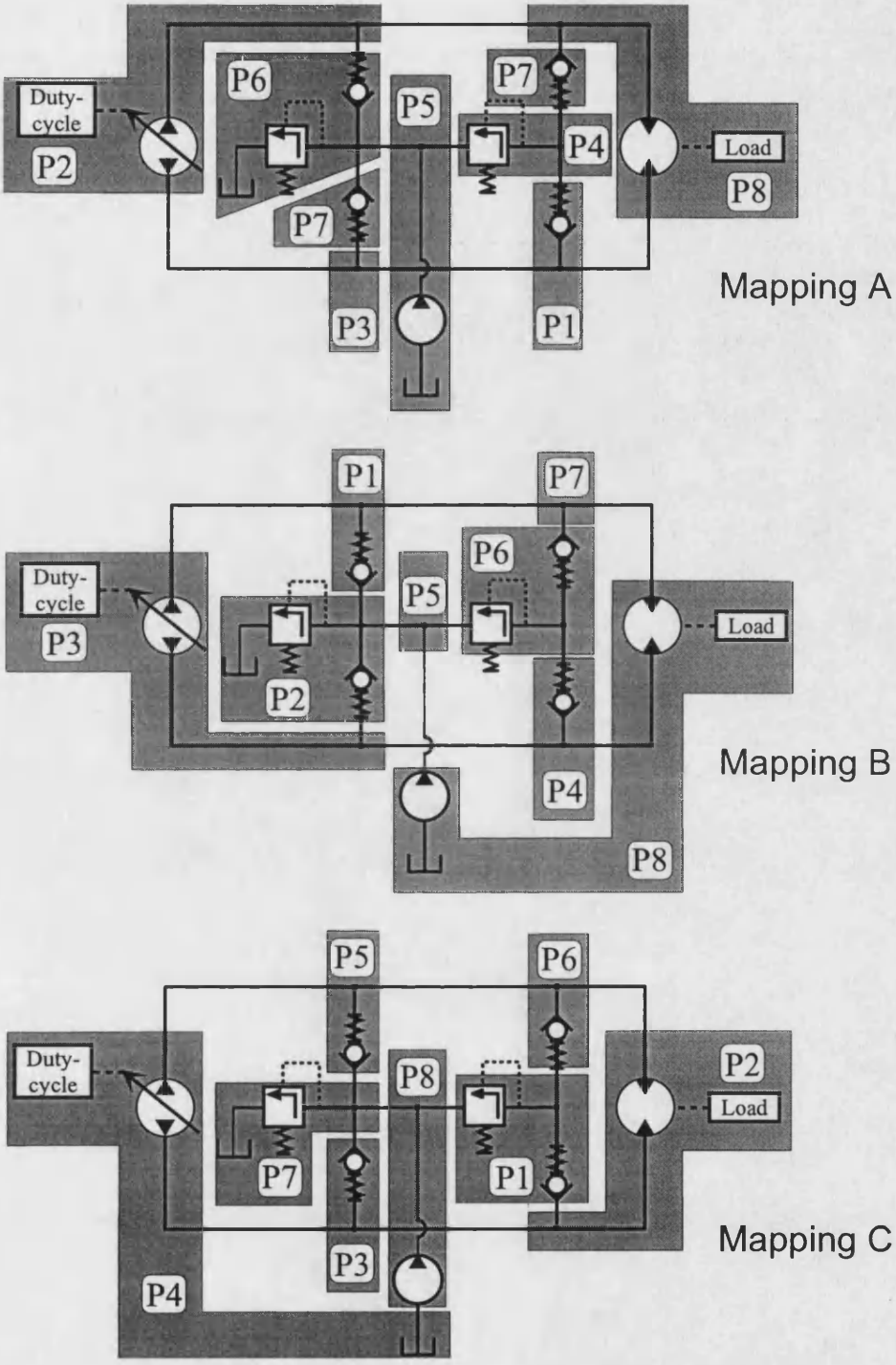


Figure 5.13 Partitioning onto 8 processors (P1 to P8)

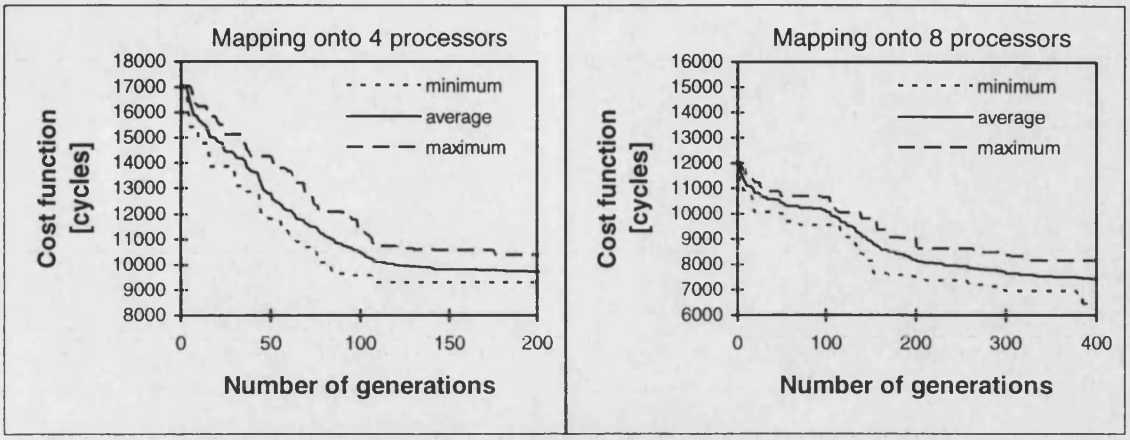


Figure 5.14 GA results for partitioning of two identical systems

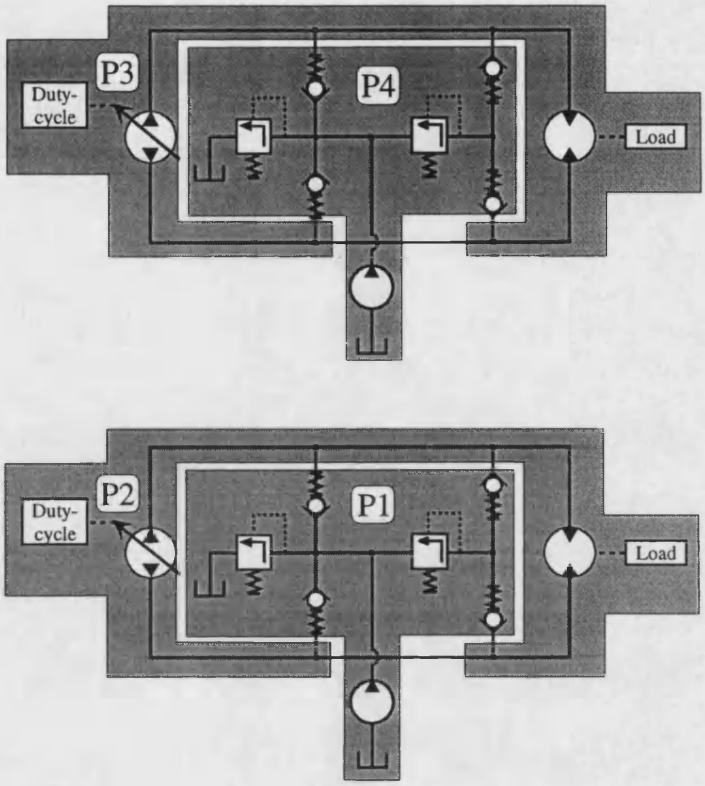


Figure 5.15 Partitioning of two systems onto 4 processors (P1 to P4)

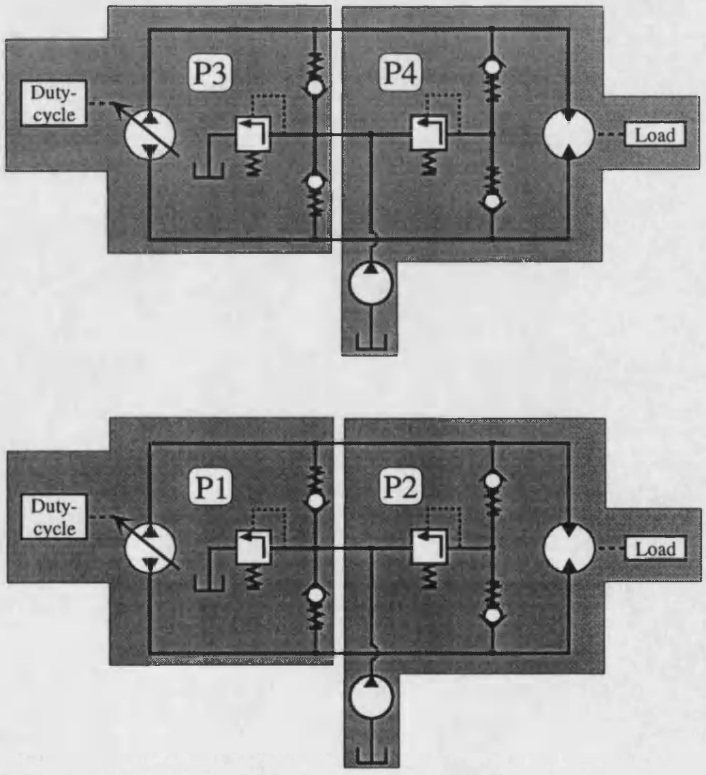


Figure 5.16 Ideal partitioning of two systems onto 4 processors (P1 to P4)

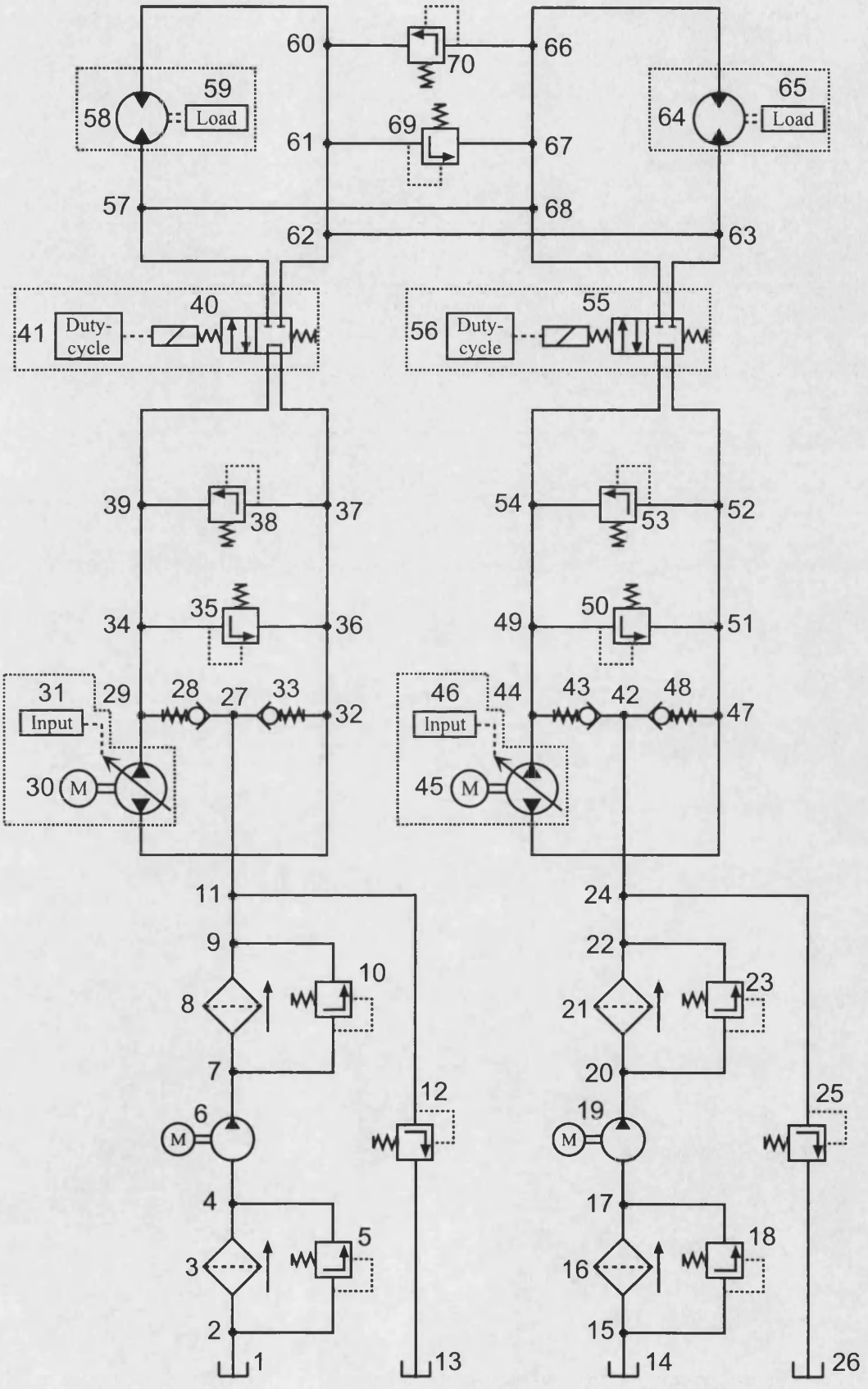


Figure 5.17 A complex hydrostatic transmission circuit

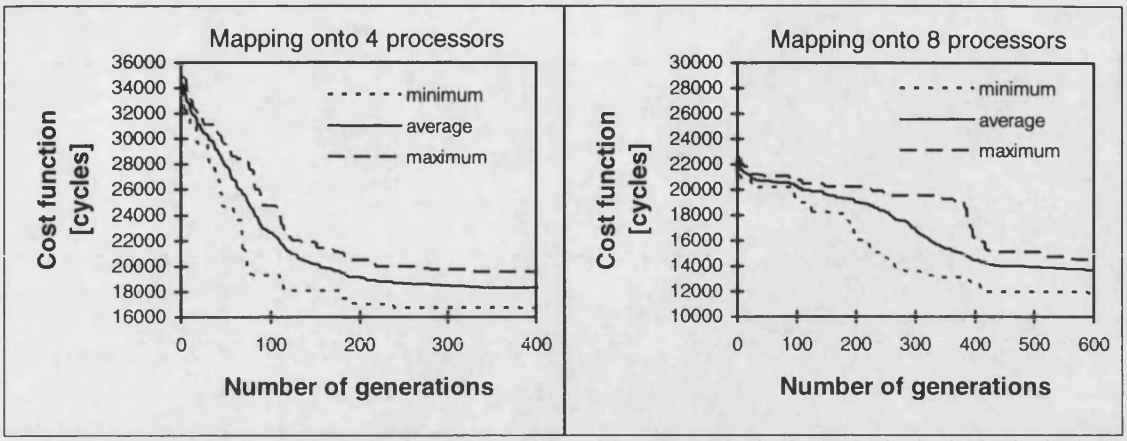


Figure 5.18 GA results for partitioning of large-scale systems

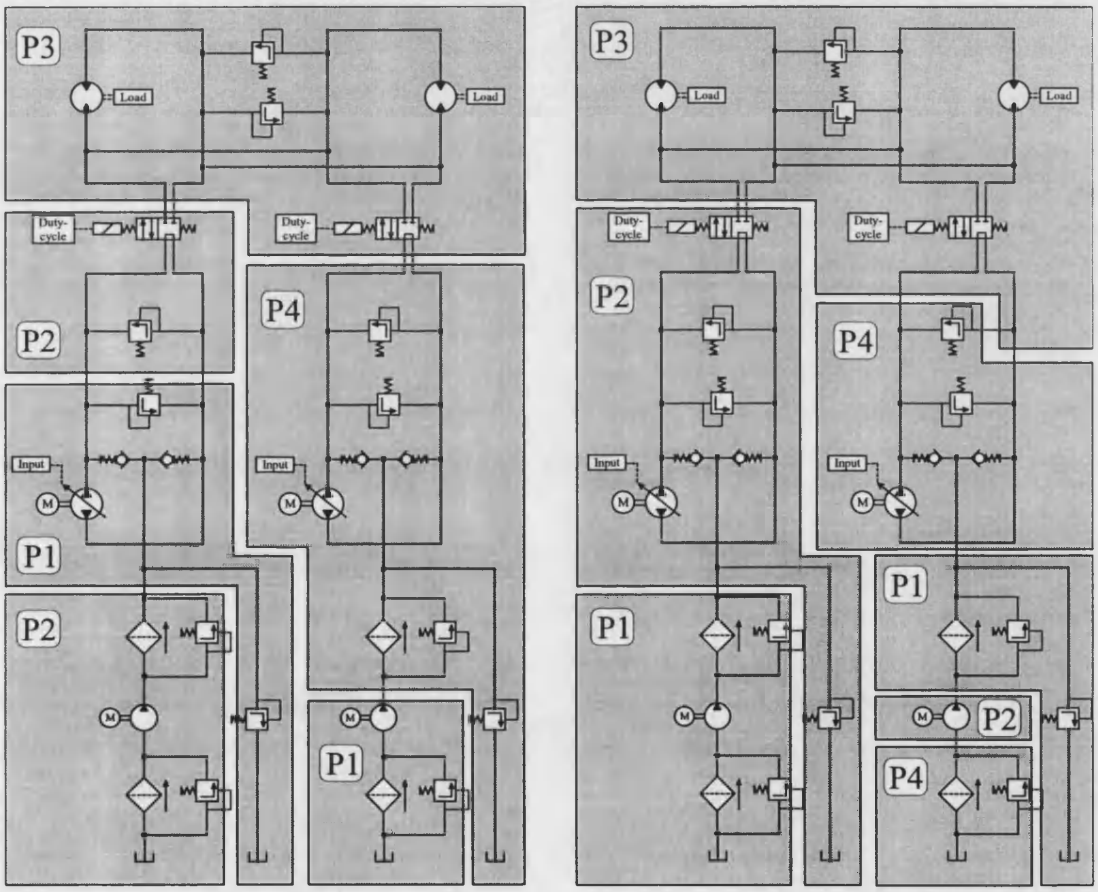


Figure 5.19 Partitioning of large-scale system onto 4 processors (P1 to P4)

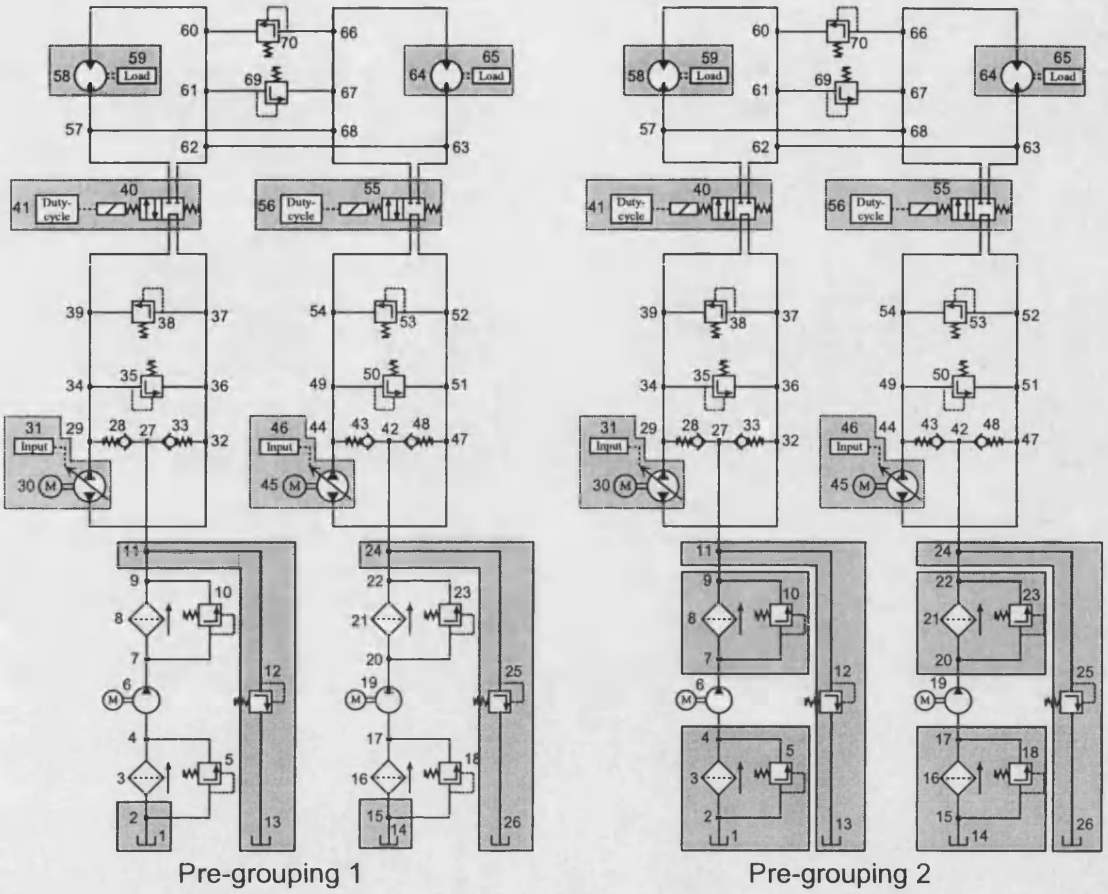


Figure 5.20 Pre-grouping of components

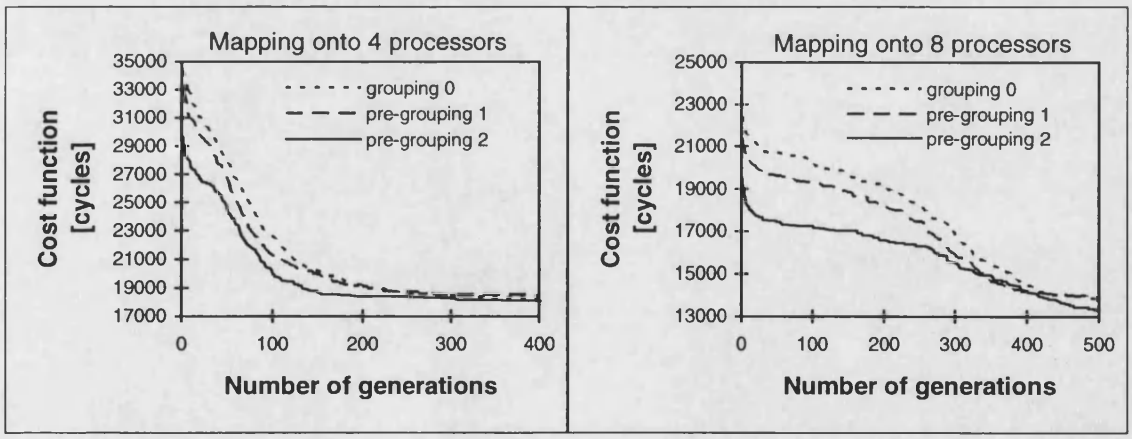


Figure 5.21 GA results comparison

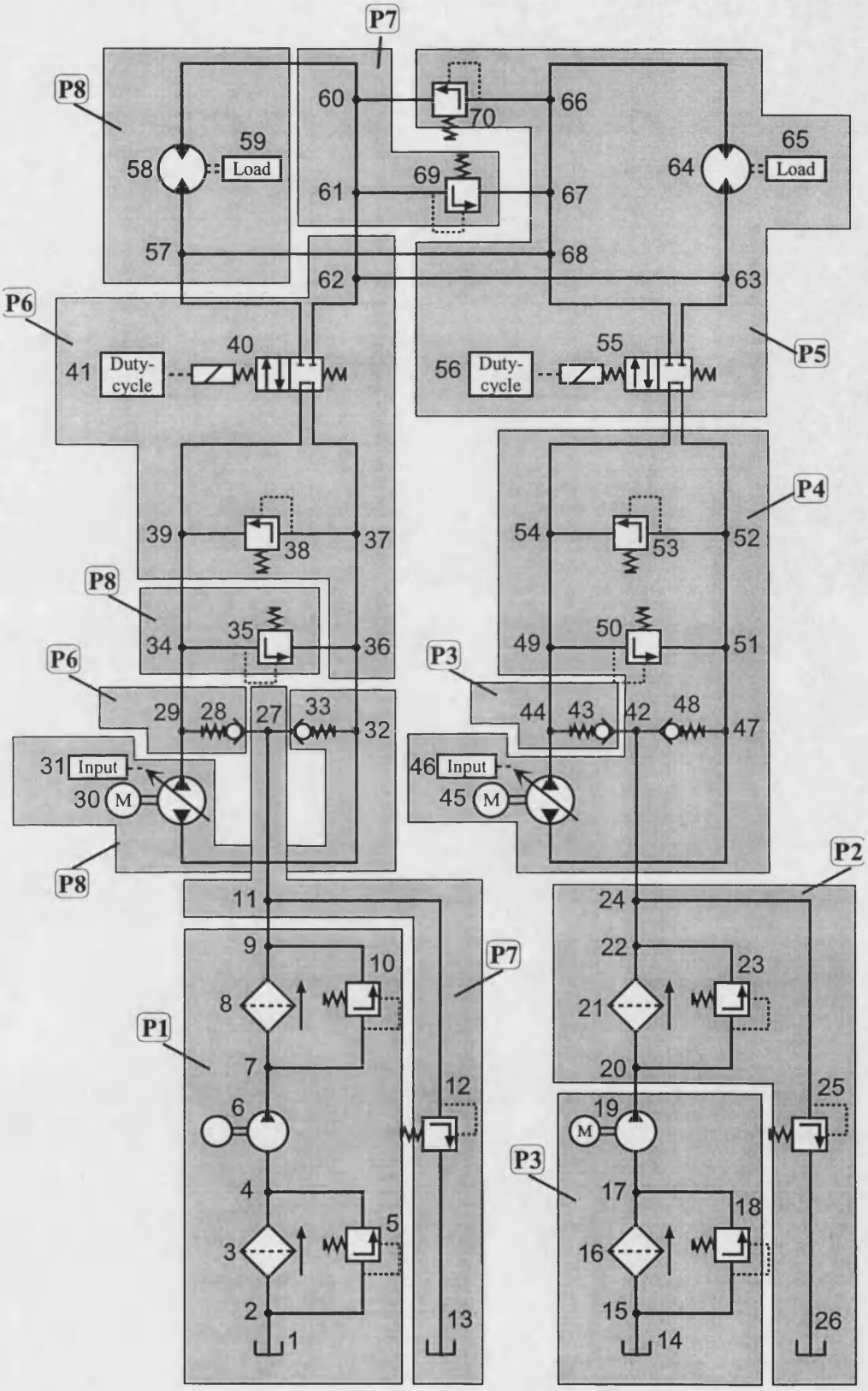


Figure 5.22 Partitioning of large-scale system onto 8 processors (P1 to P8)

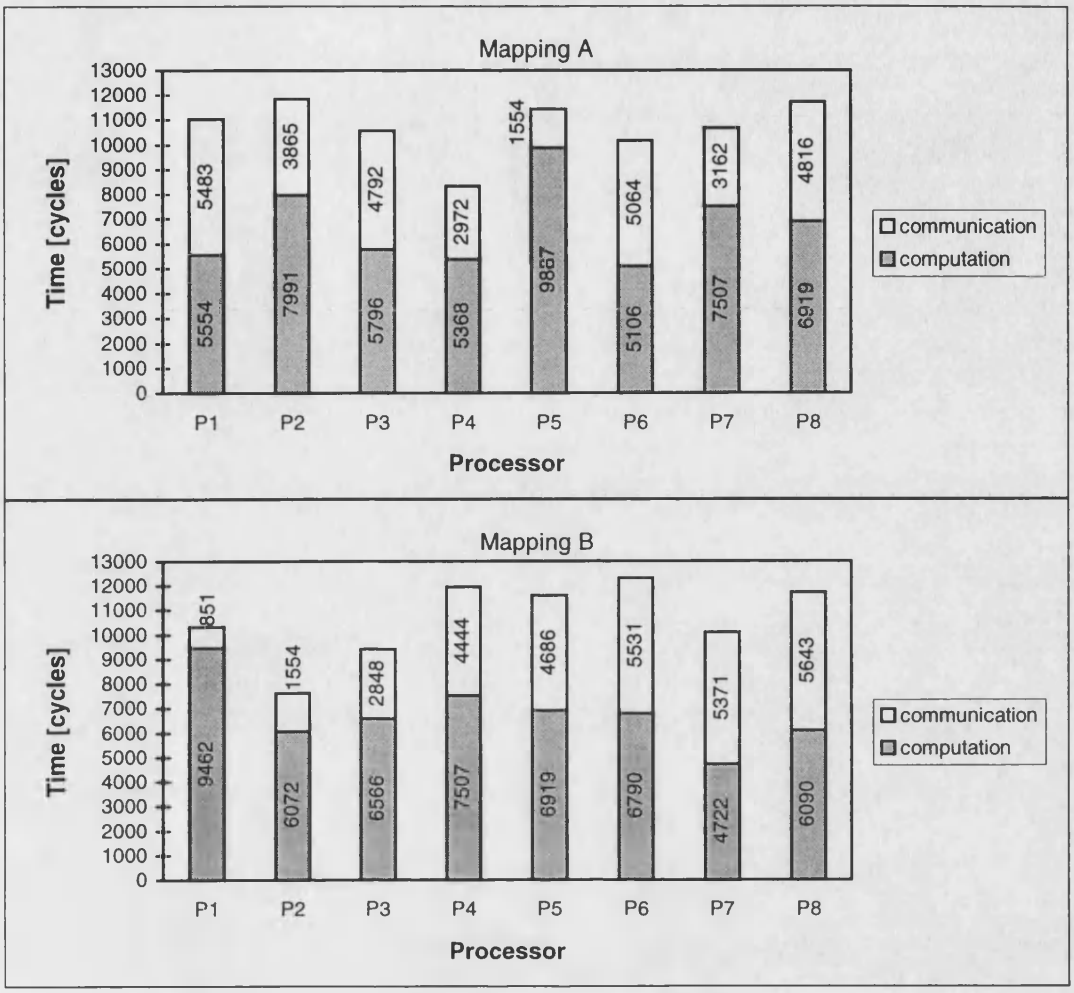


Figure 5.23 Communication and computation times

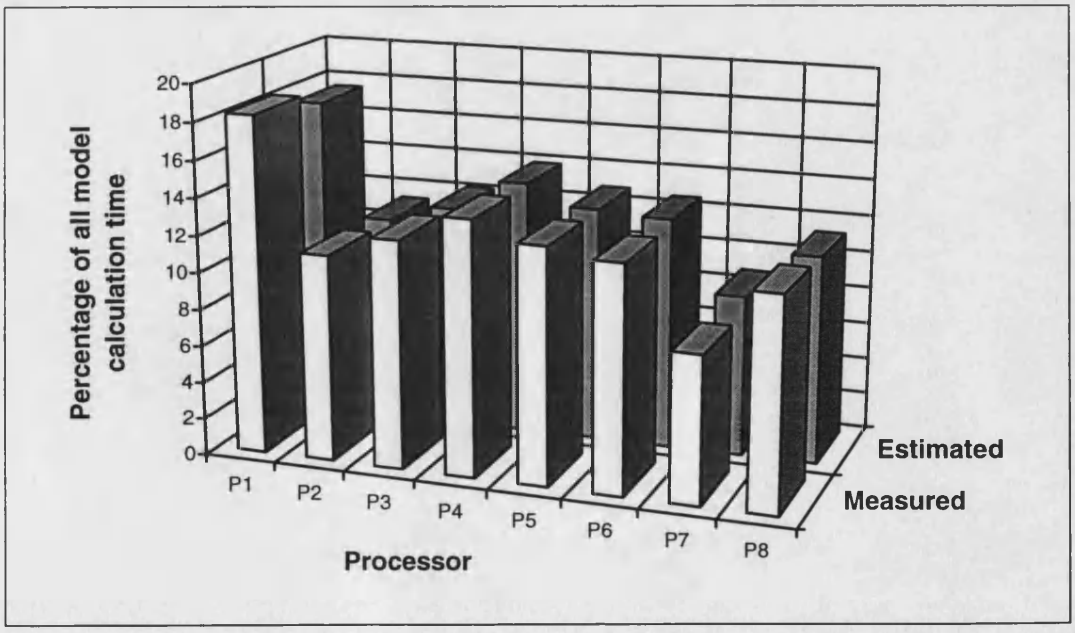


Figure 5.24 Runtime validation

6 Reduction of inter-processor communication

6.1 Introduction

For parallel simulation the main aim of reduction of execution time can be achieved by even load balancing, by minimising CPU contention and by reduction of the communication between different processors [Sunter & Bakkers, 1994]. This chapter reports an investigation on the latter for simulations with fixed time step algorithms. Most parallel computer hardware is generally designed for medium or coarse grained problems [Burton, 1994]. Parallel TLM simulations on such machines will lead to large communication overheads and real time performance can only be achieved for very simple systems. A new extrapolation-interpolation method is developed here enabling the reduction of communication between processors. This method also enables the use of more mainstream computers for efficient and large scale parallel TLM simulations.

6.2 New extrapolation-interpolation method

For the numerical scheme described in the following sections it has been found necessary to derive a new filter for the approximation of frequency-dependent friction. This new filter is described in the following section.

6.2.1 New filter for approximation of friction

In section 2.2.3 (equations 2.25 to 2.27) low-pass filtering of the characteristic pressure is used to approximate friction and to suppress unrealistic resonances. A different low-pass filter is suggested here where only the pressure is filtered which is more consistent with the physical behaviour. This approach is given by the following equations.

$$P_{a,b}(t) - Z'Q_{a,b}(t) = C'_{b,a}(t-T) \tag{6.1}$$

$$C'_{b,a}(t-T) = P'_{b,a}(t-T) + Z'Q_{b,a}(t-T) \tag{6.2}$$

$$P'_{b,a}(t-T) = (1-\alpha)\cdot P_{a,b}(t-T) + \alpha \cdot P'_{b,a}(t-2T) \tag{6.3}$$

$$Q_{b,a}(t-T) = Q_{a,b}(t-T) \tag{6.4}$$

$$Z' = \frac{Z}{1-\alpha} \tag{6.5}$$

The line impedance, Z, must be corrected by the same factor used with the standard filter (see equation 2.27) and α is again set to 0.2. This correction factor is derived in detail in

Appendix B. Appendix B also examines the TLM method as a general method for integration through comparison with the trapezoidal rule. The numerical behaviour of the new filter is closer to the trapezoidal rule of integration then the standard filter. This probably leads to better numerical properties of the simulations.

The pressure and flow propagation from one model to another is illustrated graphically in Figure 6.1. $P_b(t)$ is the pressure used by model B for the calculation for the next time step. It is calculated from the pressure at A and the filtered pressure at B both from the previous step. $Q_b(t)$ is the corresponding flow and is equal to the flow at the previous time step at A. The simultaneous propagation of pressure and flow from model B to model A happens in a similar manner.

Simulations performed on a variety of sample circuits showed that the new pressure filter suppresses resonances more effectively than the standard characteristic filter and it describes the frequency dependent friction equally well. As an example the two-actuator circuit previously described in section 3.3.1 (see Figure 3.1) was simulated with both filters. Figure 6.2 shows the velocity of one of the actuators against time in comparison for both filters. The results were calculated with a fixed time step of 0.1 ms. The new filter leads to smaller oscillations between 1.3 and 1.7 seconds. In this period the global time step needs to be smaller for both filters in order to obtain accurate results. Nevertheless, the new filter leads to better results. For the same system Figure 6.3 shows the parasitic pressure difference against time using time steps of 1 ms, 0.1 ms and 10 µs. Results obtained with the new filter and the standard filter are given on the left and right side, respectively.

In general the new filter leads to smaller parasitic pressure differences, i.e. it leads to better simulation results. This improvement is independent of the chosen time step. Hence, the new filter enables the use of larger time steps for the same simulation accuracy. Furthermore the new filter does not introduce unrealistic numerical effects. As expected the parasitic pressure difference is smaller for simulations with smaller time steps.

6.2.2 Extrapolation

In order to simulate hydraulic circuits on different processors they can be split into two or more subsystems as described in Chapter 5. Normally the pressure and flow has to be passed between the different subsystems once every time step. With the method developed

here this can be reduced, i.e. the time step for the communication between the systems can be increased. This is done by increasing the characteristic impedance

$$Z' = \frac{\rho \cdot c}{A} \cdot \frac{1}{1-\alpha} = \frac{T}{V/B_e} \cdot \frac{1}{1-\alpha} \tag{6.6}$$

to

$$Z_n = n \cdot Z' \tag{6.7}$$

in order to extrapolate pressure and flow values n time steps in advance. This corresponds to the use of a larger time step for the trapezoidal rule of integration (see Pollmeier et al. 1996d). The extrapolated values are then propagated to the other end of the line. Hence, pressure and flow values are only exchanged every n-th time step.

6.2.3 Interpolation

Taken together with the previous propagated values, the actual pressure and flow values are then formed by interpolation. In Figure 6.4 the pressure and flow propagation between two subsystems is illustrated. Every n-th time step the shaded circle values are propagated. The linear interpolation between these values (white circle values) is then performed on the other processor. For example after the propagation from subsystem A to B the following interpolations are computed on the processor where subsystem B is running.

$$P_a(t-T) = P_a(t-mT) + \{P_a(t+(n-m)T) - P_a(t-mT)\} \cdot \frac{m-1}{n} \tag{6.8}$$

$$Q_a(t-T) = Q_a(t-mT) + \{Q_a(t+(n-m)T) - Q_a(t-mT)\} \cdot \frac{m-1}{n} \tag{6.9}$$

The pressure is then filtered with

$$P_b^*(t-T) = (1-\alpha) \cdot P_a(t-T) + \alpha \cdot P_b^*(t-T) \tag{6.10}$$

and the flow used at B for the next calculation equals the interpolated flow

$$Q_b^*(t-T) = Q_a(t-T) \tag{6.11}$$

This leads to the new characteristic pressure

$$C_b^*(t-T) = P_b^*(t-T) + Z^* Q_b^*(t-T) \tag{6.12}$$

hence the TLM equation

$$P_a^*(t) - Z^* Q_a^*(t) = C_b^*(t-T) \tag{6.13}$$

can be applied. Similar equations can be derived for the pressure and flow propagation from subsystem B to A by changing the subscript 'a' to 'b' and vice versa.

6.2.4 Adjustments of the characteristic impedance

The characteristic impedance of the line connecting the subsystems is set to

$$Z^* = n \cdot Z' \tag{6.14}$$

when the extrapolated values are calculated, i.e. this is applied only every n-th time step. In between the characteristic impedance needs to be adjusted to

$$Z^* = Z' \frac{5}{5n - 4m + 4} \tag{6.15}$$

where n and m are chosen according to the notation in Figure 6.4. Equation 6.15 is derived in Appendix C. This approach leads to the recalculation of the characteristic impedance at every time step, i.e. it is an expensive method. Numerical experiments show that a different adjustment with fewer computations can be used. Simply by using equation 6.14 at every time step the simulation leads to similar good results. Using this approach the characteristic impedance needs to be calculated only once. Physically this means the line length is increased by the factor n but information is exchanged as if it was an unchanged line length. All the models and lines in the subsystems are calculated as usual with equations (6.1) to (6.5). Pollmeier et al. [1996d] gives an analytical comparison between this new method (equations 6.8 to 6.14) and the standard method (equations 6.1 to 6.5) for the case of a simple hydraulic circuit. It indicates that the systems should be partitioned at large volumes or long lines. Some simulation results for much more complex and realistic circuits are given in the following sections.

6.3 Simulation results and performance improvements

6.3.1 Two-actuator circuit

The two-actuator circuit, described in section 3.3.1, is shown again in Figure 6.5 partitioned into two sub-circuits. It is used as the first example with which to demonstrate the application of the new TLM algorithm. This circuit was selected because previous simulation studies had revealed it to possess high numerical stiffness which could lead to

very long simulation times even when using high performance numerical integration algorithms. The general system purpose and performance is described in section 3.3.1.

The circuit was split into two parts namely the computationally demanding actuator dynamics and the rest of the system. Figure 6.5 also gives the line dimensions. In Figures 6.6 and 6.7 some simulation results are shown. In the top of the figures the solid lines depict the reference results achieved by using a fixed time step of 10 μs for both the communication and the global calculation of the subsystems. These results match, very closely, those obtained from a lumped parameter simulation. Figure 6.6 shows the torque transients of the flow divider in detail and Figure 6.7 shows the corresponding results of the flow going into one of the actuators. The flow values are negative due to the sign convention (flow into a line is defined as positive). Increasing the communication time step to 1 ms, hence exchanging pressure and flow only every 100-th time step leads to very close matching results (dashed line). For comparison the same system was simulated with a global time step of 1 ms (dotted line). The difference between the reference case and the two other cases is also given in both figures for clarification. It can easily be seen that a global time step of 1 ms produces misleading results. The start-up transients are over or under estimated and other unrealistic transients are introduced. In order to simulate the dynamics in the subsystems it is necessary to use a small time step but the new method enables the use of larger time steps for the inter-processor communication.

The measured runtimes of the two-actuator example simulations are given in Table 6.1. On a single processor it takes 727 seconds to simulate the system for a certain time using a time step of 10 μs. The implementation of the same system onto two processors leads to a reduction in runtime to 511 seconds. This is equivalent to a speedup of 1.4. Using the new scheme and reducing the communication to every 10th time step only, i.e. increasing the communication time step to 0.1 ms, leads to an increase in runtime. When reducing the communication to every 100th time step the runtime is marginally reduced but it is still longer than the parallel implementation without using the new scheme. On the T9000-based platform the extra calculations (equations 6.8 and 6.9) lead to slower parallel simulations. This means that the communication overhead is smaller than the time it takes to interpolate the pressure and flow values. This is valid when using five or more transmission lines to partition the system. The two-actuator circuit was chosen to demonstrate the accuracy of the new scheme and it does not improve the speed-performance on the T9000-based platform. The method was developed for coarse grained

computers with faster processors and/or slower communication between the processors. Therefore in the next section the implementation of a complex circuit on a workstation is described.

6.3.2 Complex hydrostatic transmission circuit

The complex hydrostatic transmission circuit in Figure 5.17 (described in section 5.5.6) is used as an example with which to demonstrate the application of the new scheme on a SUN20. Therefore the circuit was partitioned symmetrically, i.e. the four transmission lines between the component pairs 60/70, 69/67, 57/68 and 62/63 were used to split the circuit into two halves (compare Figure 5.17). Using multiple threads and shared memory the new scheme was implemented under the UNIX operating system, i.e. the pressure and flow exchange between processors was realised via shared variables. Table 6.2 gives the measured runtimes of the circuit simulation. On a single processor it takes 613 seconds to simulate the system for a certain time using a time step of 10 μs. The implementation of the same system onto two processors leads to a reduction in runtime to 503 seconds. This is equivalent to a speedup of only 1.2. Reducing the communication to every 10th time step only, i.e. increasing the communication time step to 0.1 ms, leads to a considerable decrease in runtime. The measured runtime of 324 s corresponds to the very high speedup of 1.9. For this application the time needed to calculate the interpolations is much smaller then the time it would take to communicate the pressure and flow values every time step. Assuming an ideally balanced simulation, the time associated with the communication between processors can be calculated by subtracting halve the runtime on a single processor from the parallel runtime, i.e. $503s - 613/2s = 196.5s$. This divided by 10 (exchange every 10th time step only) and added to half the single processor runtime leads to $613/2s + 196.5/10s = 326.15s$. The measured runtime of 324 seconds is very close to the theoretical value. All results clearly show the possible improvements using the new scheme on platforms with faster processors and slower communication times. For comparison the same circuit simulation was also implemented on the T9000-based platform. The measured runtimes are given in Table 6.3. On a single processor the runtime of 2501 s is about 5 times longer than on the SUN20. Implementing the simulation onto two processors reduces the runtime to 1356 s, i.e. a speedup of 1.8 can be achieved, but the runtime is still longer than on a single processor on the UNIX-based system. Implementing the new scheme on to the T9000-based platform improves the performance only marginally. Using four transmission lines for the partitioning of the circuit (compared

to five with the two-actuator circuit) leads to slightly faster simulations. For the T9000-based platform up to four transmission lines can be used to split a circuit into sub-systems enabling faster simulations with the new scheme. On faster processors with slow communication this can be increased to several tens or even hundreds. When using networked computers, e.g. when using several processors connected by the internet the time used for the calculation of the interpolation is negligible compared to the communication time.

6.4 How to choose the communication time step

The maximum time step for the inter-processor communication is restricted by the simulation accuracy. As a measure of the simulation quality the parasitic pressure difference can be used (compare section 2.3.3). Figure 6.8 depicts the parasitic pressure difference against time for the two-actuator circuit using global time steps of 100µs and 10µs, respectively. On the left side the maximum parasitic pressure difference from both sub-circuits is shown. For easy recall it is referred to as 'internal error', i.e. the maximum pressure difference inside the partitions. On the right side the respective values are given for the five separated lines only. This pressure differences are referred to as 'split error', i.e. the maximum pressure difference between the partitions. Exchanging information between the partitions every 10th time step leads to maximum split errors that are smaller than the maximum internal errors. This is the case for the investigated global time steps of 100µs and 10µs. The parasitic pressures are again smaller with smaller global time steps. A reduction of communication to every 100th time step (using a global time step of 10µs) leads to larger split errors. The peaks of these errors are still of the same order of magnitude as the peaks of the internal errors. Further reduction of the communication leads to relatively large split errors. Particularly the starting transients lead to large errors. Using this information the time step for the communication between partitions of the particular circuit should be restricted to 1ms, i.e. exchange every 100th time step. A different circuit can lead to slightly different maximum time steps. More importantly, the error comparison also indicates whether the selected lines are suitable for partitioning. If very high dynamics have to be simulated and propagated along a certain line it cannot be used for partitioning. The above described method will indicate this with larger split errors. For this purpose the pressure differences can be calculated and displayed for each line independently. It should be mentioned that less frequent communication does

influence the internal errors only marginally. For example, the maximum internal error was the same for the simulations with data exchange every 100th and 500th time step.

When simulating a system containing partitions with different dynamics the reduced communication approach can also be used to simulate the sub-circuits with different time steps. For example, with the two-actuator circuit the actuators can be simulated using a time step of 10μs and the rest of the system can be simulated using 100μs. If the actuator partition exchanges data every 10th time step then the remaining partition needs to exchange data every time step. This method can also be implemented on a single processor in order to speed up the simulation, i.e. only the demanding parts of the system are simulated with a small time step.

6.5 Estimated improvements

The above described method is particularly useful if the simulation time required for the calculation of the largest task is in the same order of the respective time needed for the communication between the tasks. Estimated gains in simulation performance are considerable. If one considers a coarse grained parallel computer with the following features

- latency for communication between the processors of about 0.09 ms
- bi-directional communication bandwidth of 100 Mbits/sec = 12.5 Mbytes/sec

it takes about 0.1 ms to exchange a message of the size 125 bytes.

Consider now a system with message sizes of 125 bytes (typical for this numerical scheme), the global time step is chosen to be 0.2 ms and the simulation of one time step on one processor takes 0.2 ms. In this case real time performance can just be achieved. Applying the standard method the simulation would take the same time on two processors. The calculation time of 0.2/2 ms = 0.1 ms plus the communication time of 0.1 ms leads to 0.2 ms total time, assuming perfect balancing of the system onto two processors. The time to calculate the interpolation given in equations 6.8 & 6.9 is considered to be negligible. Using the new method with exchange every 100-th time step enables the calculation of 100 time steps in 10.1 ms, hence 0.101 ms on average for each time step. This is a considerable increase in computation speed.

Implementing the simulation on more than two processors would make the improvement even greater. The new method also enables the use of smaller time steps for real time

simulations because the communication overhead associated with the latency is largely reduced. Simulations with very small messages to be exchanged can also be computed more effectively.

6.6 Other methods to increase computation efficiency and speed up simulation

Jin & Vahldieck [1992] describe a frequency-domain TLM (FDTLM) that combines the flexibility of the conventional TLM method with the computational efficiency of frequency-domain methods. The theoretically infinite frequency range before Fourier transformation leads to the processing of much more information than needed if only a certain frequency range is of interest. Errors due to the transformation from the time-domain into the frequency-domain can be avoided with the new concept. The basis for the new technique is the excitation with an impulse train of sinusoidally modulated magnitude. Using appropriate excitation the same network can still be used in the time-domain as well as in the frequency-domain. FDTLM cannot easily be adapted to hydraulic system simulations because system inputs are defined by the particular application. It might be possible to implement it for the frequency analysis of special systems.

Christopoulos et al. [1991] describe a TLM method that permits more efficient modelling of non-uniformities and fine features. The "graded mesh" is the approach used to deal with non-uniformities and irregular shapes, whilst the "multi-grid" technique is designed to deal with problems where fine spatial resolution is required only in certain areas. For the latter the spatial resolution within a region is increased by dividing the space length Δl typically by a factor of two. Both approaches still lead to constant time step algorithms. For hydraulic systems long lines can be split into several lines transforming the TLM method into the method of characteristics (MOC) leading to another distributed simulation.

The considerable computer time and memory required to model realistic electromagnetic structures with time and space discrete methods, such as TLM, call for measures to reduce the computation count to an acceptable level [Dubard et al., 1991]. Their paper describes an enhancement of the TLM method through signal processing. The Prony-Pisarenko signal processing method is used to extract characteristic parameters from a TLM response which can be much quicker than if the Fourier transform is used. This method may be combined with parallel processing to accelerate TLM simulation even further.

A recent paper by So et al. [1995] describes the combination of TLM analysis with Prony's method as well as with autoregressive moving average (ARMA) digital signal processing for electromagnetic field modelling. By combining these advanced computation techniques, typical electromagnetic field modelling of microwave structures can be accelerated by some orders of magnitude. The number of computation time steps needed to obtain steady-state frequency domain results is reduced considerably.

The latter two methods described in this section cannot easily be applied to the TLM simulation of hydraulic systems but they might be of use for simulations where more than one domain is considered. For example the simulation of electro-hydraulical or hydromechanical systems.

6.7 Closure

In this chapter an extrapolation-interpolation method for the reduction of communication between processors has been derived. By adjusting the line impedance it is possible to use a larger time step for the communication between processors than the global time step. This approach requires a new filter for the approximation of friction. A measure to estimate the maximum communication time step has been proposed. Simulation results of realistic example circuits show good accuracy when exchanging data only every 100th time step. The new scheme leads to valuable reductions in run time on medium and coarse grained computers without compromising accuracy. It can also be used to simulate different partitions with different time steps, according to the required accuracy and dynamics of the subcircuits. Furthermore, the extrapolation-interpolation method improves the portability, i.e. the parallel TLM simulation can efficiently be implemented on a wider range of parallel computers. The above-described TLM implementations still require the system analyst to specify all simulation parameters. In the next chapter an approach based on genetic algorithms is investigated that enables automatic selection of appropriate parameters.

TABLES FOR CHAPTER 6

Runtimes in seconds (global time step = 1e-5 s)			
Number of T9000 processors	exchange every time step	exchange every 10th time step	exchange every 100th time step
1	727	-	-
2	511	520	512

Table 6.1 Two-actuator circuit runtime in seconds on T9000-based platform

Runtimes in s (global time step = 1e-5 s)		
Number of T9000 processors	exchange every time step	exchange every 10th time step
1	613	-
2	503	324

Table 6.2 Complex circuit runtime in seconds on SUN20

Runtimes in seconds (global time step = 1e-5 s)			
Number of T9000 processors	exchange every time step	exchange every 10th time step	exchange every 100th time step
1	2501	-	-
2	1356	1315	1311

Table 6.3 Complex circuit runtime in seconds on T9000-based platform

FIGURES FOR CHAPTER 6

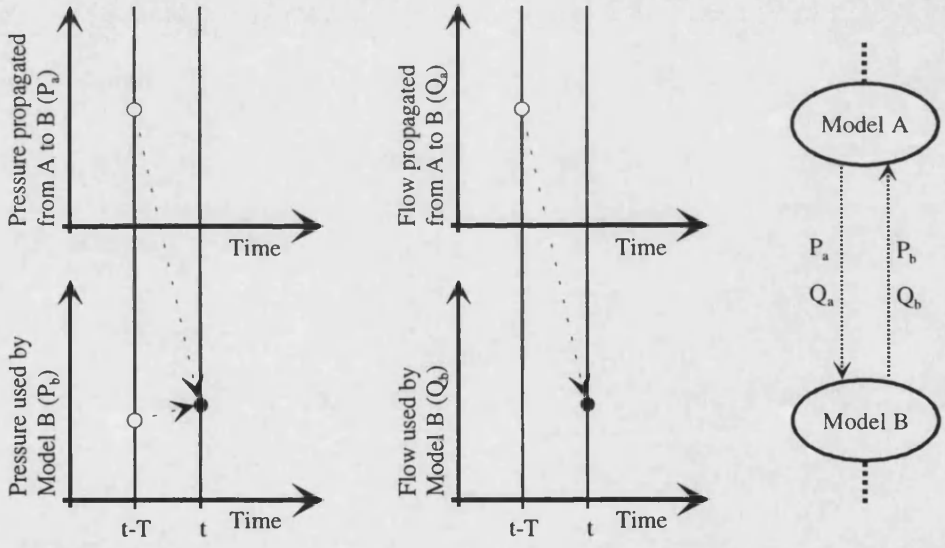


Figure 6.1 Pressure and flow propagation from Model A to Model B

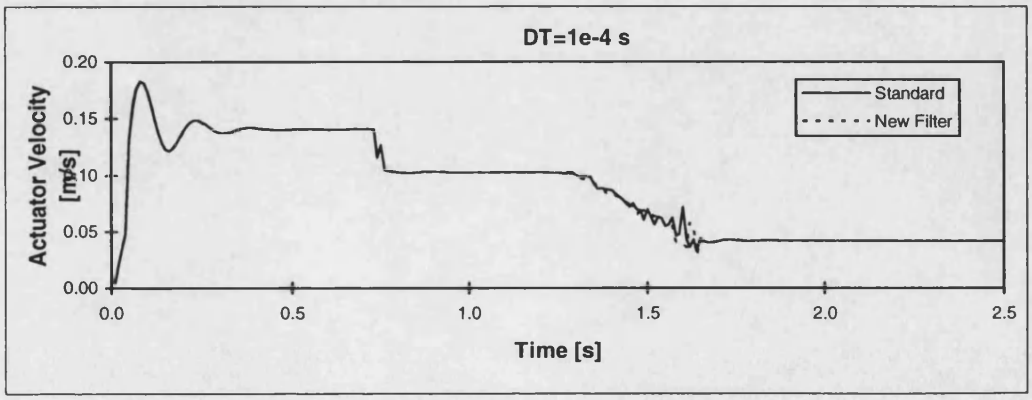


Figure 6.2 Simulation results for different filters

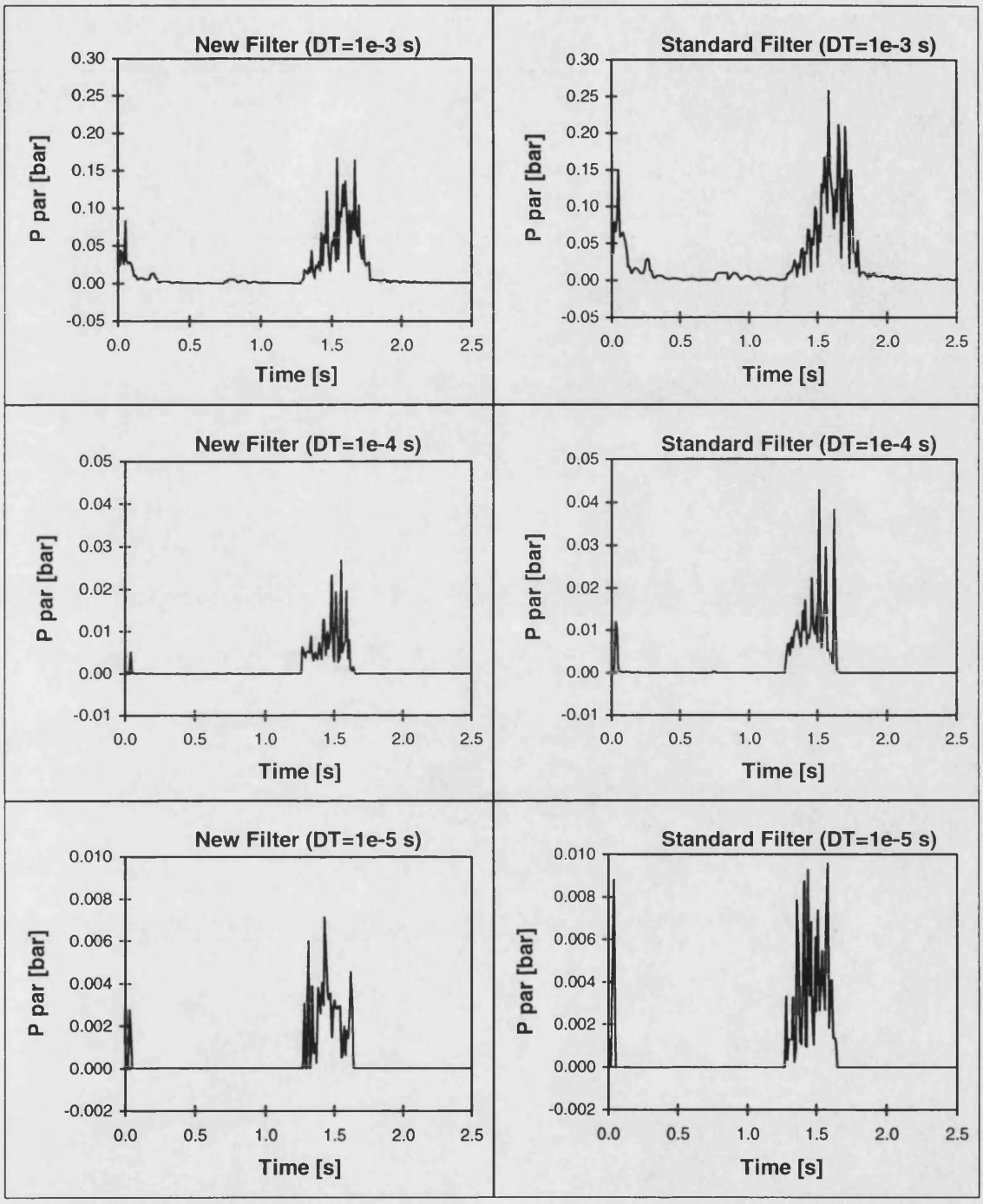


Figure 6.3 Parasitic pressure difference with different filters and different time steps

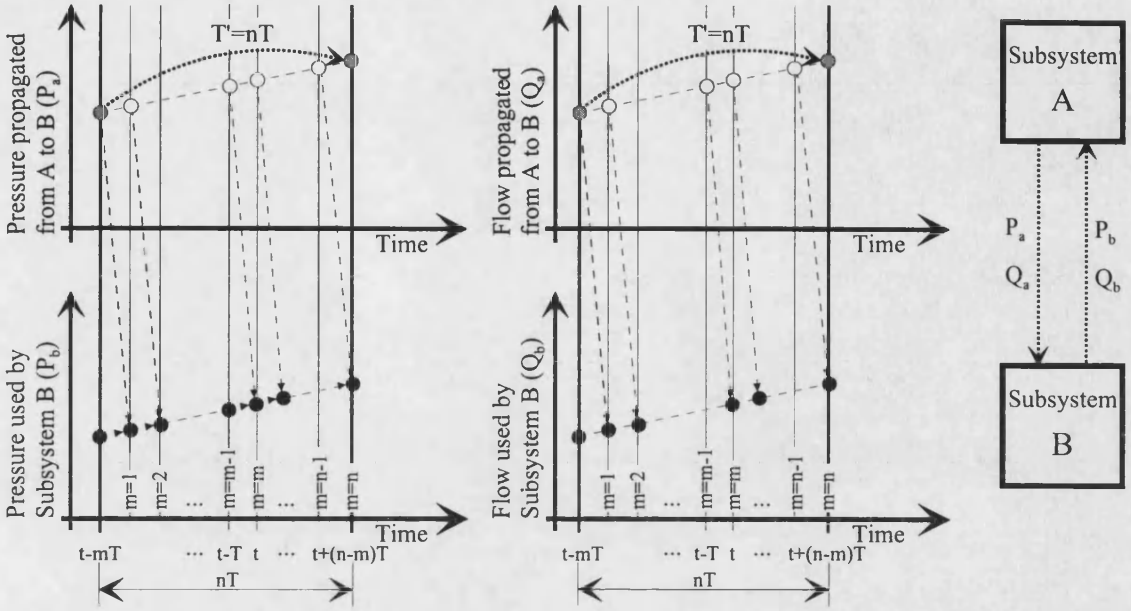
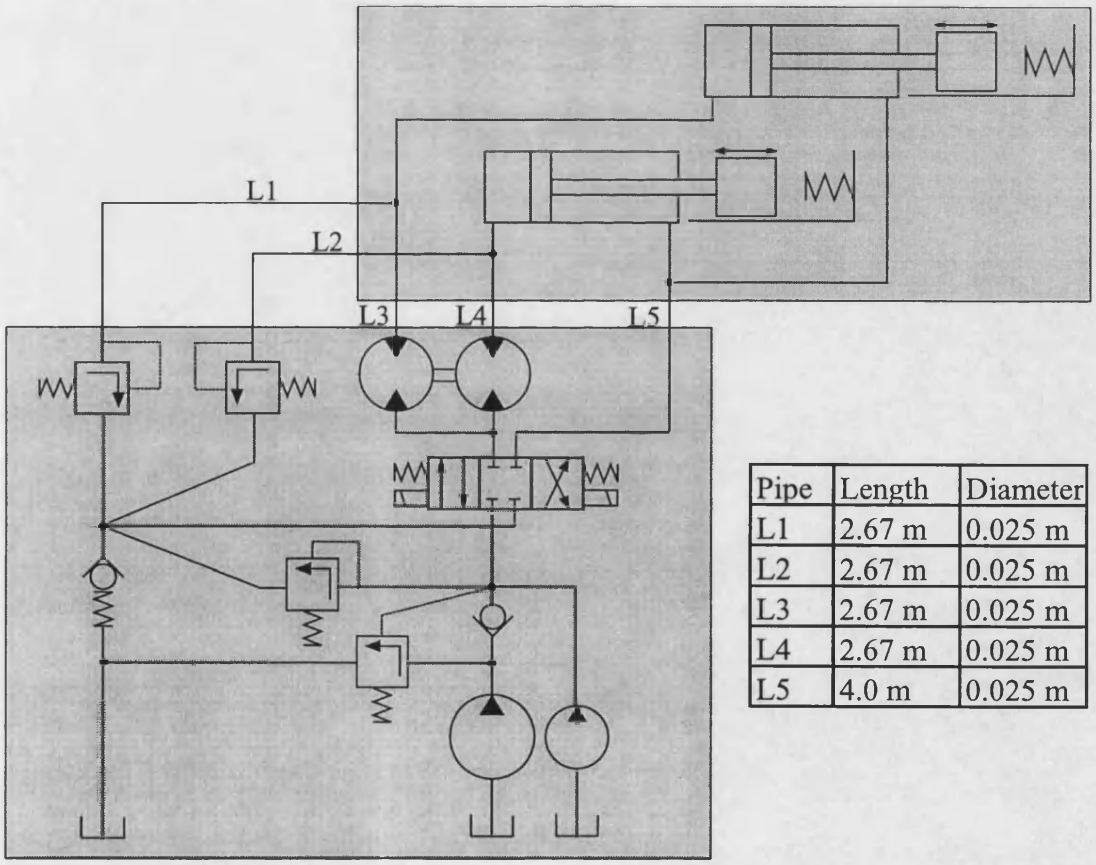


Figure 6.4 Pressure and flow propagation from Subsystem A to Subsystem B



Pipe	Length	Diameter
L1	2.67 m	0.025 m
L2	2.67 m	0.025 m
L3	2.67 m	0.025 m
L4	2.67 m	0.025 m
L5	4.0 m	0.025 m

Figure 6.5 Synchronised actuator hydraulic circuit

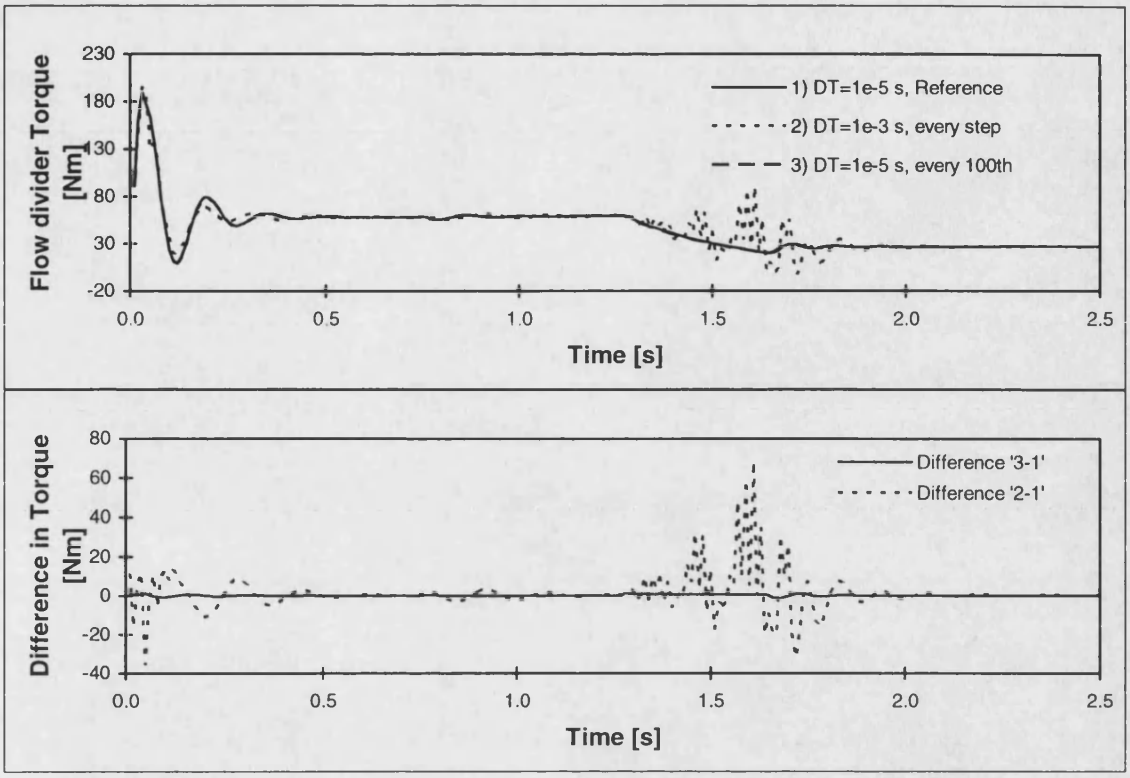


Figure 6.6 Flow divider torque and difference in torque against time

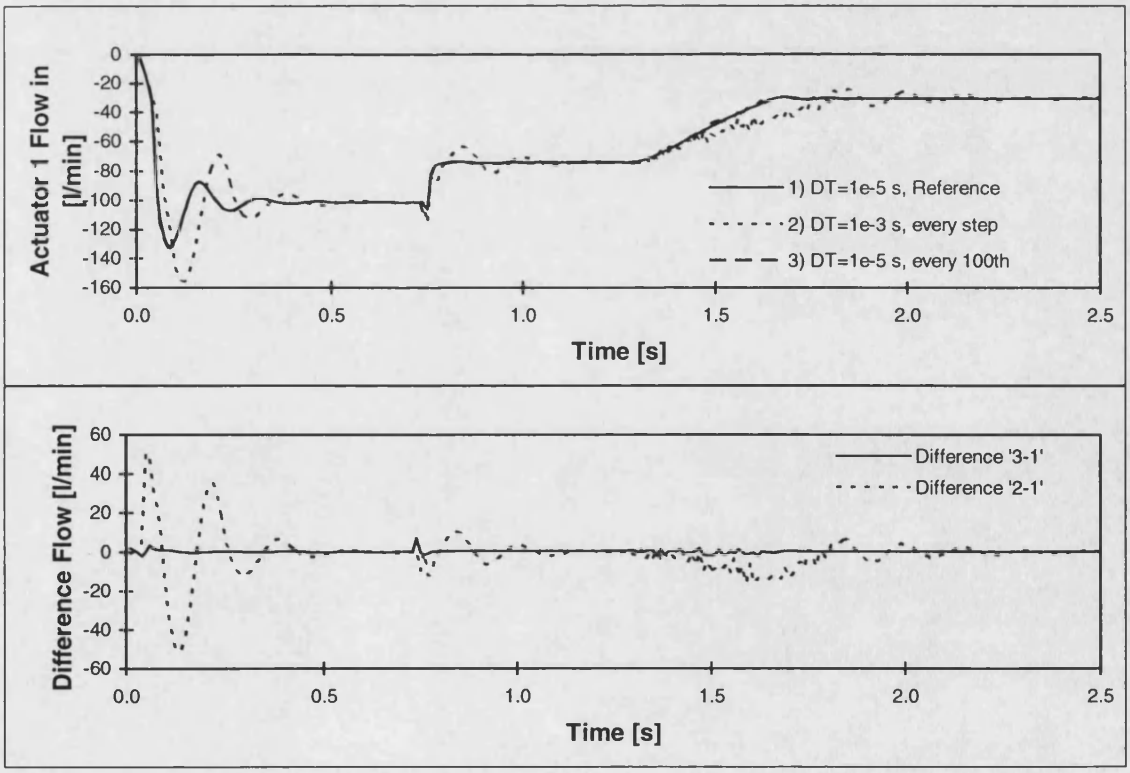


Figure 6.7 Actuator flow and difference in flow against time

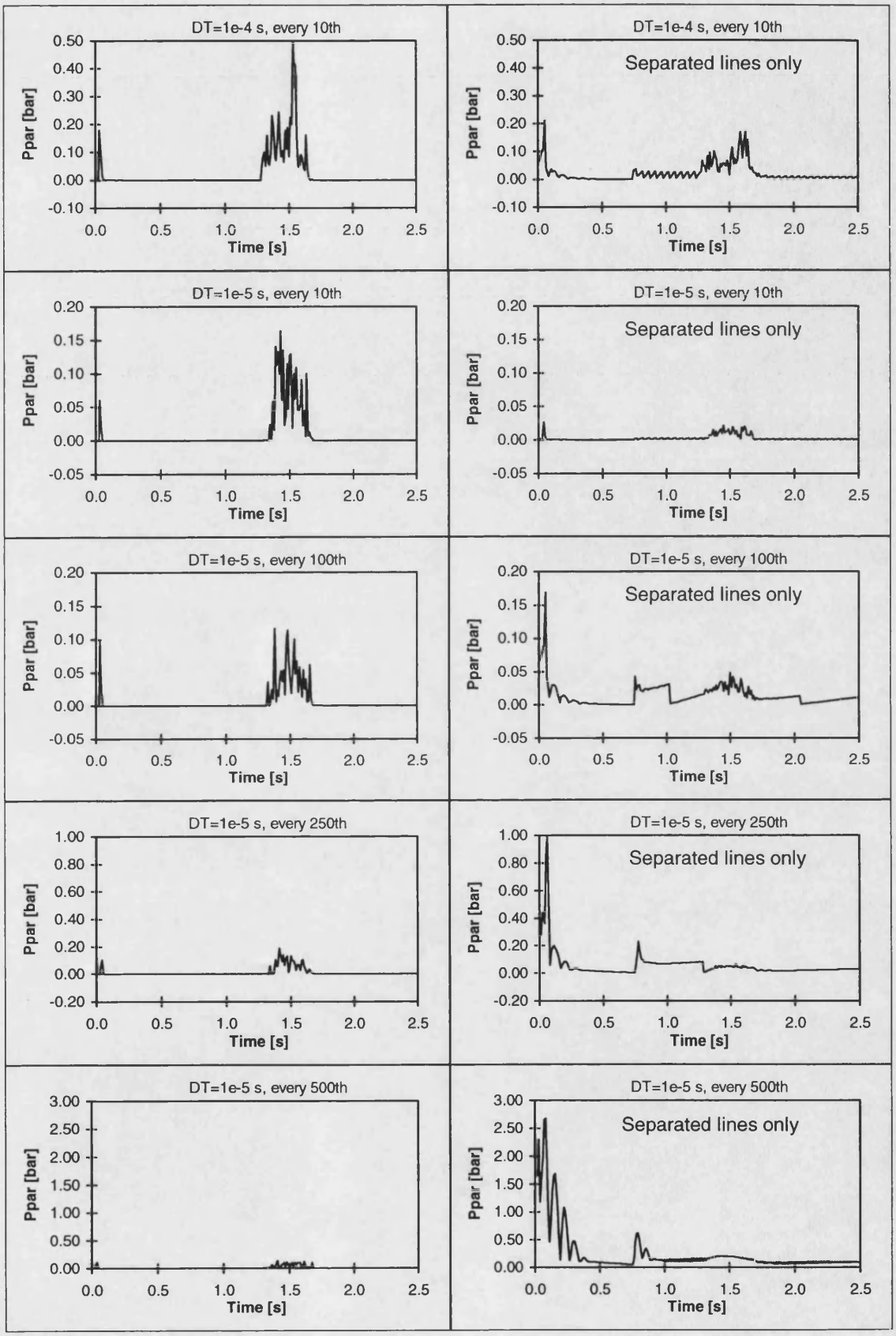


Figure 6.8 Parasitic pressure differences for different groups of lines

7 Non-linear system identification using genetic algorithms

7.1 Introduction

In this chapter it is shown how to optimise the parameters of a simulation in order to match simulated and measured values. In the literature there are only very few authors tackling this problem. Very often appropriate parameter values are only found after a long period of trial and error. A genetic algorithm (GA) is developed and employed enabling the identification and optimisation of several parameters simultaneously. A close agreement between simulation and measurement can be achieved. This requires non-linear component models in order to simulate the highly non-linear hydraulic systems correctly. The identified parameters, describing physical properties of the system, may then be used for condition monitoring purposes.

7.2 System identification

The construction of a model for system identification involves three basic actions. These are recording of system data, selection of suitable models and determining the best model [Ljung, 1987]. In this chapter it is assumed that the input-output data can be recorded during a specifically designed identification experiment, where one can determine which signals to measure and when to measure them. Hence, the input signals are chosen so that the data become sufficiently informative. Before addressing the model selection process the two different types of system identification, namely methods with parametric and non-parametric models, are described. Non-parametric system identification routines return transfer functions which are not described by an equation, i.e. they do not (explicitly) employ a finite-dimensional parameter vector. Most widely used is the Fourier analysis which returns complex frequency domain transfer functions indicating the significance of certain frequencies for example in a noise signal. In practical applications where data is recorded from a test rig, noise and disturbance signals are inevitably recorded at the same time. These noise signals cover a wide frequency range with random gain and phase lag values. This leads to considerably changing values from one frequency step to the next one. A Bode plot is commonly used to visualise the obtained frequency domain gain and phase lag.

On the other hand the result of parametric system identification is a time domain equation (transfer function) for the system output. A generalised model structure for linear time-invariant systems is given in the following equation [Ljung, 1987]:

$$A(q)y(t) = \frac{B(q)}{F(q)}u(t - n_k T) + \frac{C(q)}{D(q)}e(t) \tag{7.1}$$

where y(t), u(t) and e(t) are the time domain output, input and noise signals, respectively. T is the sample interval and the factor n_k accounts for the delay of dynamics from u to y. The polynomials A to F contain the following parameters:

$$A(q) = 1 + a_1 q^{-1} + \cdots + a_{n_a} q^{-n_a} \tag{7.2}$$

$$B(q) = 1 + b_1 q^{-1} + \cdots + b_{n_b} q^{-n_b} \tag{7.3}$$

$$C(q) = 1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c} \tag{7.4}$$

$$D(q) = 1 + d_1 q^{-1} + \cdots + a_{n_d} q^{-n_d} \tag{7.5}$$

$$F(q) = 1 + f_1 q^{-1} + \cdots + f_{n_f} q^{-n_f} \tag{7.6}$$

n_a to n_f are the number of parameters in the respective polynomials and the shift operator q^{-n} specifies values n sample times, T, earlier as for example in the following equation:

$$q^{-2}y(t) = y(t - 2T) \tag{7.7}$$

Equation 7.1 describes a general family of model structures and it may give rise to 32 different model sets, depending on which of the five polynomials A to F are used. Several of these model sets are described in the literature, for example see Ljung [1987]. The factor '1' in the polynomials B and C are left out if the polynomials F and D are also used. System identification is the determination of all parameters in order to achieve the best possible fit between measured and predicted values. However, the most important task is the selection of the model structure and the degree of the polynomials. It is here that a priori knowledge and engineering intuition may be of help in finding the right model, but one normally has to test different orders and/or delays to get the best results. The best model structure is a trade-off between flexibility and parsimony.

- Flexibility: Employing model structures that offer good capabilities of describing different possible systems. Flexibility can be obtained either by using many parameters or by placing them in "strategic positions".
- Parsimony: Not to use unnecessarily many parameters: to be "parsimonious" with the model parametrization [Ljung 1987].

Having arrived at a particular model it then remains to test whether this model is good enough, i.e. deciding whether the model is valid for its designed purpose. The linear model sets in equation 7.1 do not refer to the physical background and are often employed as so called black box models.

7.2.1 Non-linear system identification

Most identification techniques to date are restricted to linear models. Non-linear systems are often assumed to be linearizable in some manner. The linearized model is then used to analyse the behaviour of the non-linear system. For example a general state space system is given in equation 7.8:

$$\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X}, \mathbf{U}) \tag{7.8}$$

where $\mathbf{X}$ is an n-dimensional state vector and $\mathbf{U}$ is a p-dimensional input vector. The behaviour of this system near the operating point B_0 can be described by a Taylor row expansion with truncation after the first term:

$$\dot{\mathbf{x}} = \left.\frac{\partial \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{X}}\right|_{B_0} \cdot \mathbf{x} + \left.\frac{\partial \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{U}}\right|_{B_0} \cdot \mathbf{u} \tag{7.9}$$

with $\mathbf{x} = \mathbf{X} - \mathbf{X}_0$, $\mathbf{u} = \mathbf{U} - \mathbf{U}_0$ and at the operating point B_0: $\mathbf{X} = \mathbf{X}_0$, $\mathbf{U} = \mathbf{U}_0$. The Jacobian matrices $\left.\frac{\partial \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{X}}\right|_{B_0}$ and $\left.\frac{\partial \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{U}}\right|_{B_0}$ can be seen as constant coefficient matrices $\mathbf{A}$ and $\mathbf{B}$, i.e. the system can be approximated as a linear state space system:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u} \tag{7.10}$$

The problem with this simplification is that it is only valid for small perturbations in a small region around the operating point. Hence it is desirable to have a more general model that is still simple and can be computed in reasonable time.

A more advanced method of linearisation is the bilinearisation approach. A bilinear model can also be obtained by Taylor row expansion. In contrast to the normal linearisation the mixed term of elements of the second order will be considered as well (Naujoks & Wurmthaler, [1988]):

$$\dot{\mathbf{x}} = \left.\frac{\partial \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{X}}\right|_{B_0} \cdot \mathbf{x} + \left.\frac{\partial \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{U}}\right|_{B_0} \cdot \mathbf{u} + \sum_{i=1}^{p} \left.\frac{\partial^2 \mathbf{F}(\mathbf{X}, \mathbf{U})}{\partial \mathbf{X}\, \partial \mathbf{U}}\right|_{B_0} \cdot \mathbf{x} \cdot u_i \tag{7.11}$$

The matrix $\left.\frac{\partial^2 \mathbf{F}(\mathbf{X},\mathbf{U})}{\partial \mathbf{X}\,\partial \mathbf{U}}\right|_{B_0}$ contains constant coefficients that will be designated $\mathbf{N_i}$, hence the bilinear system can be written as:

$$\dot{\mathbf{x}} = \mathbf{A}\cdot\mathbf{x} + \mathbf{B}\cdot\mathbf{u} + \sum_{i=1}^{p} \mathbf{N}_i \cdot \mathbf{x} \cdot u_i \qquad (7.12)$$

With the additional bilinear term on the right hand side the system model is more accurate but it is still valid only in the region around the operating point. Another method to improve model accuracy is to develop local linear or bilinear models for the system at various points in the space of states and inputs (or around different equilibrium points). Then these local models are pieced together into a compatible global non-linear model. This can improve the model accuracy but in general, non-linear processes can only be adequately characterised by a non-linear model [Billings, 1980].

In Thomson et al. [1996] a NARMAX[1] polynomial model representation was investigated. This model is similar to equation 7.1 setting the order of the polynomials D and F to zero (ARMAX[2] model) and extending it by non-linear terms. The NARMAX model has the desirable property of being linear in its parameters and has the following general form:

$$y(t) = f\big(y(t-1)\cdots y(t-n_y), u(t-1)\cdots u(t-n_u), e(t-1)\cdots e(t-n_e)\big) + e(t) \qquad (7.13)$$

where $y(t)$ denotes the output, $u(t)$ denotes the input and $e(t)$ represents system noise; n_y, n_u and n_e are the maximum lags in the output, input and noise, respectively. The noise $e(t)$ is assumed to be a white sequence. $f(.)$ is expanded as a polynomial which produces a non-linear difference equation model. The model components are linear and non-linear polynomial functions of the input, output and iteratively computed residuals. These models, different to hierarchical models, offer simpler structure-identification algorithms and easier incorporation of a priori knowledge into the model [Thomson et al. 1996]. In their paper the adding of second-degree and cubic non-linear terms improved modelling performance of a heat exchanger significantly.

Another attempt of non-linear identification is based on Volterra series. The relation between the response, $y(t)$, of the model and its input, $x(t)$, can be expressed as the pth-order Volterra series

[1] NARMAX = Nonlinear AutoRegressive Moving Average model with eXogenous inputs
[2] ARMAX = AutoRegressive Moving Average model with eXogenous inputs

$$y_p(t) = \sum_{n=0}^{p} H_n[x(t)] \tag{7.14}$$

in which

$$H_0[x(t)] = h_0 \tag{7.15}$$

and

$$H_n[x(t)] = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} h_n(\tau_1, \cdots \tau_n) x(t-\tau_1) \cdots x(t-\tau_n) d\tau_1 \cdots d\tau_n \tag{7.16}$$

For causal systems the Volterra kernel, h_n, is zero if any of its arguments is negative; that is,

$$h_n(\tau_1, \cdots \tau_n) = 0 \quad \text{for any } \tau_i < 0,\ i = 1,2,\cdots,n \tag{7.17}$$

The identification problem thus is to determine the Volterra kernels, h_n, in equation 7.14 so that the mean-square modelling error between model and measurement is a minimum. This turns out to be a complicated task and the calculation of the multidimensional convolution integrals in equation 7.16 is quite time consuming. Furthermore the Volterra series approach does not give insights about physical parameters of the system. Other non-linear identification methods, partly also based on functional expansions, are reviewed in Mehra [1979], Billings [1980, 1985] and Natke et al. [1988]. Recently fuzzy relations have also been developed to model non-linear dynamic systems since they are universal approximators and can perform non-linear mappings [Branco & Dente, 1993].

7.2.2 Identification of fluid power systems

Fluid power systems which often contain non-linear memory elements such as hysteresis or backlash are excluded from the description of equation 7.14. The Volterra series cannot represent systems in this class, since the characteristic subharmonics associated with double-valued nonlinearities are not generated by the Volterra expansion [Billings, 1980, 1985]. In polynomial model representations such as in equation 7.1 the identified parameters do not give insights about the physical parameters of the system and hence, these approaches are not considered in this research.

Boes [1992] describes a valve controlled actuator circuit by a 4th order ODE in state space form. The model parameters are identified analytically with a recursive method, i.e. using past values. A disadvantage with the described method is the necessary linearisation which is again only valid for a small working range of the system. Conrad et al. [1993]

investigate the experimental identification of flow and torque losses for hydraulic motors. Based on the measurement of differential pressure, rotational speed, effective flow and effective torque the parameters for models of gerotor type motors are estimated. Using the least-square approximation technique two non-linear models for flow- and torque-loss, respectively, are derived. The method requires the collection of data over a large range of operating conditions and the final model is only valid for a particular motor. Furthermore, the model coefficients are not physically meaningful. Hence, the optimised model can be used to simulate a particular component behaviour but it cannot directly identify faults or deterioration of the system.

Some authors have used models based on physical parameters, i.e. models derived from first principles for the identification task. Schothorst et al. [1995] present a nonlinear dynamic model of a servo-valve, which is based on theoretical model relations. The model parameters are estimated from experimental data which are obtained using sinusoidal inputs. For the investigated system a linear model could be fitted on the measured closed loop response. This approach cannot be used for general (non-linear) fluid power systems. In del Re & Keusch [1989] the determination of a model of a hydrostatic drive was investigated using only easily obtainable measurement data. The main idea was to divide the system in small blocks, whose external quantities were measurable, and to try to tune the most simple model. Again the model was linearised. Lin & Kortüm [1992] proposed a non-linear identification algorithm which was successfully applied to a vehicle suspension system subject to random excitation. This algorithm was adapted to a hydraulic actuator system [Pollmeier et al. 1996]. The method is based on the minimisation of quadratic costs functions which are composed from the time-domain measurements and are constructed directly from the differential equation dynamics. The complete structure, that means all relevant physical equations which describe the system behaviour have to be known a priori. Estimations of the parameters can then be obtained directly from the derived formula without any iteration. The method leads to fast identification results but the formula needs to be rederived for different system topologies. Furthermore the approach does not work as well with real noisy data.

A more general approach is developed here where the idea is to make use of the set of component models derived previously (see Chapter 2). For the simulation of hydraulic systems using the pre-developed component models several parameters have to be specified. Some of these can be measured or taken from catalogues (for example physical dimensions) but others are difficult to obtain and need to be identified. For example,

leakage coefficients, friction factors and in particular a bulk modulus are difficult to determine correctly. The increased speed of the simulation using TLM modelling enables the use of genetic algorithms (GAs) for the system identification problem, i.e. many more dynamic simulations of the complete system can be achieved in reasonable time. GAs are not problem specific and particularly suited to optimising multi-parameter problems. In the following sections GAs combined with a direct search method are investigated to solve the identification problem. First the use of GAs for general system identification and optimisation is reviewed.

7.2.3 System identification and optimisation using genetic algorithms

Genetic algorithms are used in many fields for the identification of parameters and also for the optimisation of control systems. Krishnakumar & Goldberg [1992] investigate GAs as a technique for solving aerospace-related control system optimisation problems. Starkweather et al. [1990] use distributed GAs for optimisation problems. They found that distributed GAs, i.e. using several subpopulations as described in section 5.3, can outperform a serial GA using the same number of total recombinations and the same total population size. In Kristinsson & Dumont [1992] it is shown how GAs can be applied to system identification of both continuous and discrete time systems. They can be applied to directly identify physical parameters but the GA is only looking for a good solution not necessarily the best. GAs have also been developed for the estimation of kinetic and non-linear parameters in Yao & Sethares [1994]. The latter paper also investigates the application of neural networks for the identification problem. This approach will be investigated in detail in Chapter 8.

7.3 Genetic algorithms for the identification of fluid power systems

In Sepehri et al. [1994] the compliance in heavy-duty hydraulically-actuated manipulators is identified using genetic algorithms. A constant compliance is assumed and the values that lead to the best fit between simulation and measurement are considered to be correct. This cannot be a realistic system description because the compliance must change during extraction of the respective actuators due to the change in oil volume on either side of the actuator.

Donne et al. [1994, 1995] investigate methods to automate the parameter definition stage of the design process. The methods are applied to the design of fluid power systems, which primarily involve the sizing of components and the setting of controller gains. It

appears that the investigated parallel genetic algorithm is the most effective optimisation algorithm available. Testing with GAs showed that they are excellent at locating the region of a minimum, but not so effective at finding its exact location. The speed and accuracy of the GA can be improved by carrying out several direct searches once the final generation has been completed. A particular GA has already been described in section 5.3. This GA was extended by applying a version of the Hooke Jeeves method in order to fine tune the best GA solutions. This simple direct search method is employed as described in Donne [1993]. The method is described in the following section for completeness.

7.3.1 Hooke Jeeves search

Hooke & Jeeves [1961] describe direct search methods that enable the location of optima where gradient techniques are unsuitable, i.e. the method does not require the calculation of derivatives. Starting with an initial parameter set $\mathbf{X}_i$, each parameter in the set is individually varied by a certain amount (according to the step size) and the effect on the objective function of this variation is monitored. In this application the objective function is the same as the cost function which is to be minimised by the GA. If a parameter variation improves the objective function value, the particular parameter is immediately set to that value. Once all the parameters have been varied, the objective function of the new parameter set $\mathbf{X}_n$ is compared with the value due to $\mathbf{X}_i$. If there is no improvement, the size of the step is reduced. If there is an improvement, a pattern move is made according to the following equation:

$$\mathbf{X}_p = \mathbf{X}_i + k(\mathbf{X}_n - \mathbf{X}_i) \tag{7.18}$$

The reasoning behind the pattern move is that, if $\mathbf{X}_i$ and $\mathbf{X}_n$ lie on a slope and $\mathbf{X}_n$ lies below $\mathbf{X}_i$, which it must do if it has a lower objective function value, moving from $\mathbf{X}_i$ through $\mathbf{X}_n$ will lead to an even better parameter set, $\mathbf{X}_p$. The pattern move amplifies the change between $\mathbf{X}_i$ and $\mathbf{X}_n$. If the objective function value of $\mathbf{X}_p$ or its value after all its parameter values have been varied is not an improvement on that due to $\mathbf{X}_n$, the parameter values are reset to $\mathbf{X}_n$. This process is continued until the step sizes for each parameter drop below preset values. At this point, the current parameter set is considered to be optimal. An initial and minimum step size can be specified for each parameter. This allows the different order of the parameters to be taken into account. The standard algorithm was improved by specifying upper and lower bounds for each parameter. If a parameter value exceeds the bounds it is set to that boundary value.

There are several constants in the Hooke Jeeves algorithm that effect its performance, these are the initial parameter step size, the pattern move factor k (equation 7.18), the step reduction factor and the step size at which the program is terminated. The most important of the four factors is the initial step size. It is vital that this value is large enough to ensure that a substantial portion of the search space is covered by the initial steps. This will enable the algorithm to rapidly locate parameter sets with low objective function values, even if they differ significantly from the initial parameter set. Best results were achieved by setting the initial step value to approximately 20 percent of the difference between the upper and lower boundary values of the parameters [Donne, 1993]. This setting was used for the case where the Hooke Jeeves search is applied on its own. In this chapter the best parameter sets obtained from each sub-population of the GA are used as starting points for the direct search. In this case the initial step size was set to 64 times the minimum step size. If a stage of the Hooke Jeeves search fails to locate an improved point, the step sizes are reduced by a factor of four. Due to the proximity of the starting points to good minima, only a few Hooke Jeeves search steps are necessary in order to improve the GA solution considerably. The combination of GA and Hooke Jeeves search also increases the chances of locating the global minimum.

7.3.2 Experimental hydraulic actuator system

Figure 7.1 shows the schematic of the test rig used to demonstrate the GA-based approach of parameter identification. A trolley-mounted mass of 46 Kg is moved by a 0.1 m stroke actuator controlled by an electrically-operated servo valve. The trolley position is measured with a displacement transducer and its signal is used for a positional control loop. Leakage between the annulus and piston sides of the actuator can be introduced by opening bleed valve 1. A second, passive actuator is employed to apply a load to the trolley. The dynamic friction characteristics of the load can be modified by adjusting bleed valve 2 which connects the two sides of the load actuator. Several sensors on the rig enable the measurement of the servo valve current, i, the differential pressure between the piston and annulus sides of the drive actuator, Δp, and the trolley position, x. The system pressure was measured with a pressure gauge next to the servo valve and the oil temperature was measured at the ring main system supplying this test rig with constant pressure. Dynamic data was recorded using the data acquisition system described in section 4.3 and alternatively with a PC-based system.

7.3.3 Simulation model of the hydraulic actuator system

A schematic of the model used to simulate the hydraulic actuator system is shown in Figure 7.2. In contrast to the real system the load actuator is not simulated. The externally applied load is accounted for by a velocity dependent friction term in the actuator model. Furthermore the same model includes a leakage term to simulate leakage flow between the annulus and piston sides. This enables the simulation of a worn out or broken seal where the flow is from high to low pressure side of the actuator. External leakage is not considered in this investigation. Details about the actuator and servo valve models are given in section 2.4. Flow into the system is supplied by a constant flow source and the pressure relief valve is used to adjust the system pressure. The settings of the control loop are the same as used during the measurement at the test rig, i.e. the feedback and controller gain K1 and K2 are set to 100 [V/m] and 0.5 [A/V], respectively.

7.3.4 Investigation of different cost functions

It is essential that the objective or cost function of a system accurately reflects the design requirement. Because the GA is concerned with function minimisation, the objective function value must reduce as the system performance approaches the required performance. If an objective is not achieved, the function value must be proportional to the degree of failure. A good performance of the GA relies on having a well defined objective function. Here the aim is to minimise the difference between measured and simulated system behaviour. In particular the differences in actuator displacement and the differences in differential pressure Δp are of interest. The objective function describing the average absolute difference between measured and simulated displacement (x_i^m and x_i^s) is calculated according to the following equation

$$obfn_x = \frac{1}{n}\sum_{i=1}^{n}\left|x_i^m - x_i^s\right| \tag{7.19}$$

Where n specifies the number of data samples. A similar equation can be derived for the differential pressure:

$$obfn_p = \frac{1}{n}\sum_{i=1}^{n}\left|\Delta p_i^m - \Delta p_i^s\right| \tag{7.20}$$

With the measured and simulated differential pressures Δp_i^m and Δp_i^s, respectively. The RMS[1] error was not used in equations 7.19 and 7.20 because this would give peaks and overshoots, for example after step changes, too much influence on the optimisation. It is more important to simulate the general trend correctly than to estimate the amplitude of individual pressure peaks exactly. In this investigation a time step of 0.4 ms was used for all simulations. This time step was chosen in order to enable real time simulation of the whole circuit on one T9000 processor. The actual time interval between samples in equations 7.19 and 7.20 of 2.5 ms was determined by the sample rate of 400 Hz used for the measurement. Simulated values between two time steps were linearly interpolated if necessary. About 1720 samples were used for the calculation of the obfn-values. Figure 7.3 shows the first four seconds of the demand duty cycle used to excite the system sufficiently. In order to reduce the influence of start up transients it starts with a zero demand. Zero displacement is defined when the actuator is in its middle position. Furthermore the demand signal contains steady state periods to check the simulation quality for this case as well. With this demand used as system input several simulation parameters were optimised with the cost function in equation 7.19. The different parameters optimised will be described in section 7.4.1. Figure 7.4 shows the comparison between measurement and optimised simulation of the displacement and the pressure difference, respectively. The agreement between measured and simulated values is very good for the displacement but not for the pressure difference. In this case the difference in displacement is not a good measure for the general simulation performance. The displacement is dependent on the pressure differences in an integral manner. A set of simulation parameters can be found that enables the correct simulation of the displacement with larger pressure differences. In Figure 7.5 the same results are given for the simulation parameters optimised with the cost function in equation 7.20. Here a very good agreement between simulation and measurement was achieved for both compared signals. A correct pressure difference automatically leads to matching displacement values. This means only certain signals are useful for the evaluation of an objective function. In general one can say that values representing higher derivatives are best to use. For example, if the velocity or even the acceleration signal is available it should be used for measures that describe the system performance. Unfortunately these signals are often not available.

[1] RMS-error = Root Mean Square-error

Another option is the use of multi-objective optimisation. With the particular example circuit the differences in displacement and differential pressure can be optimised simultaneously. The proposed cost function is given with

$$obfn_{5x+p} = 5 \cdot obfn_x + obfn_p = 5 \cdot \frac{1}{n}\sum_{i=1}^{n}\left|x_i^m - x_i^s\right| + \frac{1}{n}\sum_{i=1}^{n}\left|\Delta p_i^m - \Delta p_i^s\right| \qquad (7.21)$$

This means the GA tries to minimise the weighted sum of both averaged differences. The weighting factor is included because the different units and quantities need to be accounted for. Initial tests indicated that the optimised cost function values for equation 7.19 are about 5 times smaller than the respective values obtained using equation 7.20. The GA optimisation was again applied, this time using equation 7.21 as cost function. Figure 7.6 gives the respective results. Similar to Figure 7.5 a very good agreement for both quantities was achieved, indeed the optimised parameters turn out to have very similar values. Changes of less than 6 percent are obtained for the important simulation parameters. The disadvantage with equation 7.21 is that it takes longer to be calculated; hence from now on only equation 7.20 is used as objective function.

7.4 Parameter identification

This section investigates the different parameters that need to be identified in order to achieve good simulation accuracy. Details on the applied GA settings are given and different fault levels are identified.

7.4.1 Choice of parameters to be identified

Using the above described GA with the Hooke Jeeves search several different sets of up to eleven parameters were optimised. In Table 7.1 the results of 6 different optimisations, called opt. 1 to opt. 6, are given. Each row in the table gives the identified values of the following parameters:

- Q_{sa}, Q_{br}, Q_{sb} and Q_{ar} describe the pressure-flow characteristics of the different ports of the servo valve. The subscripts indicate the ports using the notation in Figure 2.13. These characteristics are calculated by $k_i = Q_i/\sqrt{P_i}$ where P_i is the respective pressure difference. P_i was set to 1 bar for all four ports, i.e. the characteristics could be described by the four flows only with $k_i = Q_i/\sqrt{bar}$.

- The servo valve spool underlap is specified as a percentage of its spool stroke, i.e. the spool displacement leading to the maximum flow through the valve. A symmetrical

underlap was assumed, i.e. in its middle position the lap between the respective edges is identical.

- First stage leakage of the servo valve, Q_{leak}, was also included in the valve model as described in section 2.4.1. The leakage flow per bar pressure difference is accounted for by this parameter.
- $F_{res.}$ represents the total resistive force acting against the movement of actuator and load.
- F_{stict} is the respective stiction force preventing the initial movement of the load.
- The velocity dependent friction (also called viscous load) acting on the load and the actuator is described by the parameter 'vis. load'.
- Leakage from piston to annulus side of the actuators and vice versa is included by a parameter called 'cross leak'. Again the leakage is specified in flow per pressure difference.
- B_e is the effective bulk modulus, here given in the unit bar.

All other parameters are assumed to be known. These are mainly dimensions of component parts and the oil properties like density (864 kg/m^3) and kinematic viscosity (46e-6 m^2/s). In Table 7.1 the crossed boxes indicate values that are not optimised, i.e. they are not modelled and set to zero. The last column gives the achieved objective function value, called obfn, in bar. This value represents the average of the difference between measured and predicted differential pressure calculated according to equation 7.20. Using the optimised parameter values from opt. 1 to opt. 6 six system simulations were performed. In Figures 7.7 to 7.12 the simulated and measured pressure differences are displayed against time for these cases. Again the demand duty cycle from Figure 7.3 was applied. All simulations lead to a very close agreement between simulated and measured actuator displacement, hence these results are not displayed.

Initially, in the first optimisation attempt (opt. 1), the pressure-flow characteristics of the servo valve were assumed to be the same for all four ports. Leakage was not accounted for and a zero lap valve was assumed. There is already a good agreement achieved between prediction and measurement in parts of the investigated time interval (Figure 7.7). Unfortunately not the complete duty cycle can be simulated correctly. These results can be improved by accounting for first stage leakage and underlap of the servo valve (opt. 2, Figure 7.8). The reduced objective function value indicates a small improvement of the

simulation accuracy. A slightly better objective function can be achieved by accounting for cross port leakage of the actuator (opt. 3, Figure 7.9). The improvement gained is relatively small, i.e. it was concluded that some details of the circuit were not modelled correctly. Here the servo valve was most likely to work differently to the predicted behaviour. Next the pressure-flow characteristics of the servo valve were assumed to be different for the edges connected to supply and return (opt. 4, Figure 7.10). In this case the optimisation leads to different values for the two pairs of pressure-flow characteristics. Even though no leakage and underlap was accounted for the simulation accuracy was improved. Mainly in the interval between 2.4 and 2.6 seconds the difference between simulation and measurement is unsatisfactory. Including the leakage terms and valve underlap can improve the simulation quality in this time interval (opt. 5, Figure 7.11). An even better system model can be obtained by optimising four different pressure-flow characteristics for the four edges of the servo valve (opt. 6, Figure 7.12). For completeness the simulated and measured valve current of this optimisation is given in Figure 7.13. Again a very good agreement between measurement and prediction was achieved. The valve current saturates at 250 mA, hence this signal does not contain sufficient information to be used for the calculation of an objective function.

If not all factors are used for the optimisation (for example leakage terms are not included) the other parameters are adjusted in order to achieve the best possible results. This can lead to a null resistive force (opt. 1, 3, 4 & 5 in Table 7.1). In a real system there must be some resistive force present, i.e. this force is negligible or the system description is inaccurate. Assuming the latter, an experienced simulation designer knows that some important features of the system are missing in the simulation model. Thus one should include more details into the models until all parameter values get reasonable values. This indicates how detailed the models of the different components need to be in order to achieve the required simulation accuracy. It also indicates whether certain model simplifications are permissible. For example it might be sufficient to approximate some components by linearised models as described in section 7.2.1. Some detailed knowledge about the system (a priori information) can be useful for the optimisation process. For example the actuator moving the load is more than 10 years old and probably significantly worn, hence leakage had to be included in its model. Furthermore there could be some air in the actuator leading to the identification of relatively low values for the effective bulk modulus.

7.4.2 Details on the GA used for the parameter identification

Similar to the GA described in Chapter 5 twelve subpopulations were used and the same probabilities for crossover and mutation were applied (see Table 5.1). Further details are given in Table 7.2 for the different optimisations. Each parameter was represented by a string of 6 bits. The different number of parameters leads to different length binary strings for each chromosome. Normally a GA working with longer chromosomes needs to calculate more generations before the obfn-values converge. This cannot be seen in Figure 7.14 where the optimisation progress of the six different GA runs is displayed over the number of generations. For opt. 6 the increased number of parameters leads to faster improvements of the optimisation. An additional improvement of the obfn-value after the 40 th generation can clearly be seen. Figure 7.15 gives the best obfn-value for each of the 12 subpopulations. The random starting values lead to very different best values after the first generation. When the GA finishes (here the GA is stopped after 40 generations) the Hooke Jeeves search is employed and details of the achieved results are given in Figure 7.16. This additional Hooke Jeeves search at the end of the GA can improve the results considerably. One can also see that it is useful to apply the direct search to all 12 subpopulations as the best result obtained by the GA is not necessarily closest to the best achievable result. Here the obfn-value of subpopulation 12 which is only the third best obtained from the GA leads to the overall best result.

For all investigated identifications the number of generations and the number of individuals in each subpopulations was set to 40 and 30, respectively. These settings lead to runtimes of 592 to 1154 minutes on a Sun 20 workstation applying the combined GA-Hooke Jeeves search. The complete TLM simulation of the circuit (required once per calculation of the objective function) takes about 2 seconds on a Sun 20. This is about 100 times faster than a standard lumped parameter simulation of the same circuit. The many oscillations and rapid changes slow down these traditional simulations (due to the small time step required for the simulation of rapid transients) but they do not influence the run-time of the fixed time step TLM simulation. Considering the total optimisation runtime one can see that only the large speed up enables the calculation of sufficiently many simulations in reasonable time.

Although the parameters are optimised automatically with the described GA approach some expert knowledge is required in order to achieve good and reliable results with this method. Firstly, one has to decide whether the simulation accuracy is good enough, i.e.

which average pressure difference (obfn-value) is acceptable. A visual comparison between simulation and measurement is unavoidable if the 'shape' of the signal needs to be similar, i.e. the obfn-value on its own is not the only reliable performance measure. Secondly, one has to specify upper and lower values for each of the parameters. If these values are not known very large intervals may need to be set for an initial GA run. The intervals can then be reduced for a second run. Another option is to use floating point coding where the bits are used to represent the mantissa and the exponent of a floating point number. Particularly for the latter approach it is necessary to check whether identified parameters are of reasonable size.

In Table 7.3 the upper (u_i) and lower (l_i) limits used for all investigated parameters are given. The resolution of each parameter can be calculated using these limits and the number of bits, b, employed to represent each parameter:

$$res = \frac{u_i - l_i}{2^b - 1} \tag{7.22}$$

Table 7.3 also gives these values. If the required resolution is known in advance, the following equation can be used to provide an estimate of the appropriate binary string length:

$$b = \frac{log_{10}\left\{\frac{u_i - l_i}{res} + 1\right\}}{log_{10} 2} \tag{7.23}$$

It is highly unlikely that the value of b will be an integer, as it must be, so it is rounded up or down and either the resolution or the boundary values of the parameters are adjusted using equation 7.22. Here the same number of bits $b = 6$ is used for all parameters. Due to the small parameter intervals obtained by initial tests this leads to sufficiently small resolutions.

7.4.3 Importance of individual parameters for the optimisation

The problem of specifying a sufficient resolution for each parameter can be solved by looking at the sensitivity of a simulation to changes in its parameters. This has been investigated for the sixth optimisation (opt. 6). Table 7.4 gives the change in obfn-value for changes in the parameters. During the tests all parameters were set to their optimised values except one which was changed by ±10 and ±20 percent. The new objective function value is compared to the original one of 2.987 bar. Changing the pressure-flow coefficients of the spool valve leads to large changes in obfn-value, i.e. these factors are

very important for a correct simulation. The same is valid for the viscous load factor. These values should be optimised with a high resolution. For the system investigated the simulation is not as sensitive to changes in cross port leakage and bulk modulus. These parameters can be optimised with a smaller binary string length. The least-important parameters are the underlap and first stage leakage of the valve as well as the resistive and stiction forces. An even smaller resolution might be sufficient for these parameters. In general parameters which lead to a large change in obfn-value should be represented by more bits than parameters which do not lead to any considerable changes. This should enable a better (faster) and more efficient optimisation process. Another approach to speed up the optimisation is to use less samples for the calculation of the obfn-value in equation 7.20. This might lead to worse optimisation results; it is not further investigated in this thesis.

7.4.4 Confidence measures and quality of the simulation

In the above-described method fixed time step simulations are applied. Thus one needs a measure indicating whether differences between simulation and measurement are caused by an insufficiently small time step. For this the parasitic pressure difference P_{par} described in section 3.2.1 can be used. Figure 7.17 shows the measured and simulated pressure difference simultaneously with the respective parasitic pressure difference for opt. 6. High values of P_{par} indicate low simulation accuracy or an insufficiently small time step. Immediately after rapid changes in pressure level P_{par} increases, i.e. the transients are simulated with a relatively low accuracy. This accuracy can still be sufficient but this is dependent on the objective of the simulation. In the time interval between 1.8 and 1.9 seconds there is an offset between measurement and simulation. Because P_{par} is small in this time interval the simulation time step cannot be the reason for this deviation. It also indicates that some parts of the system are still not modelled correctly.

Once all simulation parameters are identified the performance can also be tested with a different demand duty cycle. In Figure 7.18 the results are given for a more rapidly changing random input signal (using the parameters from opt. 6). Again a very good agreement between simulation and measurement is achieved. In general a different system input should always be used to check the simulation performance. One should consider that different duty cycles lead to different objective function values. Hence, the quality of different optimisations described by the obfn-values can only be compared for the same duty cycle.

7.4.5 Fault level identification using GAs

In this section three different faults in the hydraulic actuator circuit are investigated. It was decided to focus on the identification of faults that are common in practice and that can be introduced into the rig without causing damage. Different levels of the following faults were introduced into the test rig (Figure 7.1):

1. decreased supply pressure (by setting the relief valve pressure)
2. increased actuator cross leakage (by incrementally opening bleed valve 1)
3. increased load dynamic friction (by incrementally closing bleed valve 2)

All the signals described in section 7.3.2 were measured when running the rig with four different levels of each fault. The levels of the faults were then identified by changing only one parameter of the optimised system (opt. 6). For the supply pressure fault a minimum obfn-value was obtained by incrementally reducing the pressure in different simulations (steps of 0.5 bar were used). The value of the pressure used to calculate the minimum was considered to be the correct value for the particular fault level. In a similar manner the cross leakage level was identified by changing the leakage coefficient of the actuator model and the viscous load factor was changed for the dynamic friction fault.

In Figure 7.19 the simulated and measured pressure differences are shown over time for four different system pressure faults. Normal operation of the system is assumed to be at a pressure of 100 bar. The different pressure fault levels 1 to 4 investigated are set to 80, 60, 40 and 20 bar, respectively. Using the above described approach the pressures are identified to be 80.0, 58.5, 41.0 and 17.0 bar, respectively. The large difference between the lowest identified and measured fault is partly due to the low accuracy of the system pressure gauge. For fault level 1 the agreement between simulated and measured pressure difference is even better than for the fault free case. The respective obfn-value of 2.63 bar is smaller than the value calculated for opt. 6 (2.987 bar). Table 7.5 gives the values of all identified parameters and the respective obfn-values. Reducing the pressure to 60 bar (fault level 2) still leads to very good agreement between simulation and measurement although the obfn-value increases. With increasing level of fault the differences between prediction and measurement become larger (Figure 7.19), i.e. the obfn-values increase. This indicates that some of the simulation parameters assumed to be constant need to be modelled dependent on the system pressure. The simulation can only approximate the system behaviour accurately for a limited range of pressures. This implies that the parameters cannot be optimised for the complete operating range of the system. As a test

all eleven simulation parameters (compare Table 7.1) have been identified using the GA described in section 7.3 with the data obtained by setting the pressure to 80 bar (fault level 1). The identified parameter values are slightly different to the values obtained by the fault free case, but a very small obfn-value of 2.11 bar was achieved. This also indicates the pressure dependency of the parameters.

For the faults in supply pressure the actual fault levels can be measured whereas the leakage and friction faults cannot be measured directly. The leakage fault is introduced into the rig by opening bleed valve 1 in increments of 1/12-th of a turn (compare Figure 7.1). This leads to a flow across the lines and not across the actuator as it is modelled in the simulation. Due to the difference between the rig and the model, larger errors are expected for the identified leakage fault level. In Figure 7.20 the simulated and measured pressure differences are shown over time for four different leakage faults. Although the system was not modelled like the real model the 'shape' of the pressure difference signal is simulated most accurately. The approximated leakage seems to describe the changes in pressure difference correctly. All identified parameters and the respective obfn-values are again given in Table 7.5. Measurements with an artificially deteriorated actuator where for example the sealing is damaged, might lead to an even better agreement. Another option is to simulate the external flow path through the bleed valve but this would not represent a real faulty system.

The third investigated fault is the friction fault which is introduced into the rig as a change in viscous load due to an increased restriction in bleed valve 2 (Figure 7.1). Again increments of 1/12-th of a turn are used where the load increases when the valve is closed. For fault levels 1 and 2 the agreement is very good whereas larger errors lead to worse results. This can be seen in the obfn-value as well as in Figure 7.21 where a comparison between simulation and measurement is given. Again an increasing fault level leads to an increase of the obfn-value. Thus in general the simulation accuracy deteriorates with increasing fault magnitudes.

7.4.6 Condition monitoring using GAs

There is no clear cut between system identification and condition monitoring. The identification of system parameters can be used for condition monitoring purposes, for example an identified leakage coefficient that is increasing can indicate the deterioration of a sealing. A big advantage of the GA is that several parameters can be identified simultaneously. Thus it should be possible to identify several simultaneous faults, i.e. the

deterioration of several parts and components of the system. This requires sophisticated simulation models for all important components. The approach might be suitable for the monitoring of gradually changing properties such as leakage and friction. If a system is available say once a day for a few minutes a special duty cycle could be run. The obtained data can then be used during the day to identify appropriate parameters. Due to the long runtimes of several hours this approach cannot be used for real time applications.

7.5 Closure

In this chapter a GA-based parameter identification method is developed and applied to an example fluid power system. The performance of the GA is improved by a Hooke Jeeves direct search method. Different cost functions were investigated and it was found that signals representing higher derivatives like velocity and differential pressures are useful for the design of these objective functions. Several parameters could be identified simultaneously leading to a very good agreement between simulation and measurement. The optimisation approach investigated here uses pre-developed non-linear component models to identify its parameters. This leads to a very flexible method, i.e. for different systems the parameter optimisation can easily be adapted. Furthermore the method automatically indicates whether certain parameters are relevant and whether the component models contain sufficient detail. Different fault levels could be identified successfully. The method also enables the identification of friction and leakage coefficients which cannot be measured directly. This may then be used for the condition monitoring of fluid power systems. Due to the long run time of the optimisation process there is no real-time capability of this method. Therefore in the next chapter an approach based on artificial neural networks will be investigated enabling real time condition monitoring. The fault levels identified in this chapter will support the neural network training.

TABLES FOR CHAPTER 7

	Q s-a [l/min]	Q b-r [l/min]	Q s-b [l/min]	Q a-r [l/min]	underlap [percent]	Q leak [l/min/bar]	F res. [N]	F stict. [N]	vis. load [N/m/s]	cross leak [l/min/bar]	Be [bar]	obfn [bar]
opt.1	3.540	3.540	3.540	3.540			0.0	561.9	5428.6		15238	4.812
opt.2	3.413	3.413	3.413	3.413	0.571	0.143	42.9	600.0	5262.9		8000	4.785
opt.3	3.603	3.603	3.603	3.603			0.0	507.4	5880.0	0.004	16000	4.648
opt.4	3.921	3.286	3.921	3.286			0.0	600.0	5491.4		13714	4.255
opt.5	4.452	3.486	4.452	3.486	0.568	0.144	0.0	521.1	6714.3	0.027	8127	3.672
opt.6	4.107	2.787	3.570	3.717	0.915	0.548	50.9	335.0	7428.6	0.02617	8000	2.987

Table 7.1 Optimisation results

	Number of parameters	No. of bits per parameter	Total No. of bits	Number of generations	No. of indiv. per subpopulation
opt. 1	5	6	30	40	30
opt. 2	7	6	42	40	30
opt. 3	6	6	36	40	30
opt. 4	6	6	36	40	30
opt. 5	9	6	54	40	30
opt. 6	11	6	66	40	30

Table 7.2 Details on the GA settings

	lower value	upper value	resolution using 6 bits	unit of parameter
Q s-a	1.0	5.0	6.35E-02	[l/min]
Q b-r	1.0	5.0	6.35E-02	[l/min]
Q s-b	1.0	5.0	6.35E-02	[l/min]
Q a-r	1.0	5.0	6.35E-02	[l/min]
underlap	0.0	3.0	4.76E-02	[percent]
Q leak	0.0	3.0	4.76E-02	[l/min/bar]
F resist.	0.0	300.0	4.76E+00	[N]
F stict.	300.0	600.0	4.76E+00	[N]
visc. load	3000.0	12000.0	1.43E+02	[N/m/s]
cross leak	0.0	0.08	1.27E-03	[l/min/bar]
Be	6000.0	16000.0	1.59E+02	[bar]

Table 7.3 Resolutions and upper & lower parameter values

reference values obfn = 2.987 bar		(-)20%	obfn [bar]	change [%]	(-)10%	obfn [bar]	change [%]
Q s-a	4.1071	3.2857	4.48	49.98	3.6964	3.56	19.18
Q b-r	2.7873	2.2298	4.58	53.33	2.5086	3.49	16.84
Q s-b	3.5697	2.8558	4.11	37.59	3.2127	3.29	10.14
Q a-r	3.7167	2.9734	4.00	33.91	3.3450	3.29	10.14
% underlap	0.9152	0.7322	3.13	4.78	0.8237	3.01	0.77
Q leak	0.5481	0.4385	3.08	3.11	0.4933	2.99	0.10
F resist.	50.905	40.724	3.09	3.44	45.8145	3.03	1.44
F stict.	335	268	3.08	3.11	301.5	3.02	1.10
visc. load	7428.6	5942.88	4.48	49.98	6685.74	3.45	15.50
cross leak	0.02617	0.02093	3.17	6.12	0.02355	3.05	2.11
Be	8000	6400	3.05	2.11	7200	2.99	0.10

reference values obfn = 2.987 bar		(+)10%	obfn [bar]	change [%]	(+)20%	obfn [bar]	change [%]
Q s-a	4.1071	4.5178	3.47	16.17	4.9285	4.10	37.26
Q b-r	2.7873	3.0660	3.42	14.49	3.3448	4.04	35.25
Q s-b	3.5697	3.9267	3.14	5.12	4.2836	3.37	12.82
Q a-r	3.7167	4.0884	3.31	10.81	4.4600	3.89	30.23
% underlap	0.91524	1.0068	3.00	0.43	1.0983	3.00	0.43
Q leak	0.5481	0.6029	3.02	1.10	0.6577	3.03	1.44
F resist.	50.905	55.9955	3.03	1.44	61.086	3.03	1.44
F stict.	335	368.5	3.02	1.10	402	3.02	1.10
visc. load	7428.6	8171.46	3.40	13.82	8914.32	4.16	39.27
cross leak	0.02617	0.02878	3.09	3.44	0.03140	3.17	6.12
Be	8000	8800	3.09	3.44	9600	3.18	6.46

Table 7.4 Sensitivity of simulation to changes in parameters

	Supply pressure faults			Actuator cross leakage faults			Viscous load/friction faults		
Fault level	measured [bar]	identified [bar]	obfn [bar]	measured [turns]	identified [l/min/bar]	obfn [bar]	measured [turns]	identified [N/m/s]	obfn [bar]
1	80	80.0	2.63	1/12	0.0306	3.12	2/3	12160	3.60
2	60	58.5	4.21	1/6	0.0520	3.47	1/2	16730	3.70
3	40	41.0	4.37	1/4	0.0660	5.05	5/12	22320	5.09
4	20	17.0	4.80	1/3	0.0902	5.84	1/3	27760	7.24

Table 7.5 Fault level identification

FIGURES FOR CHAPTER 7

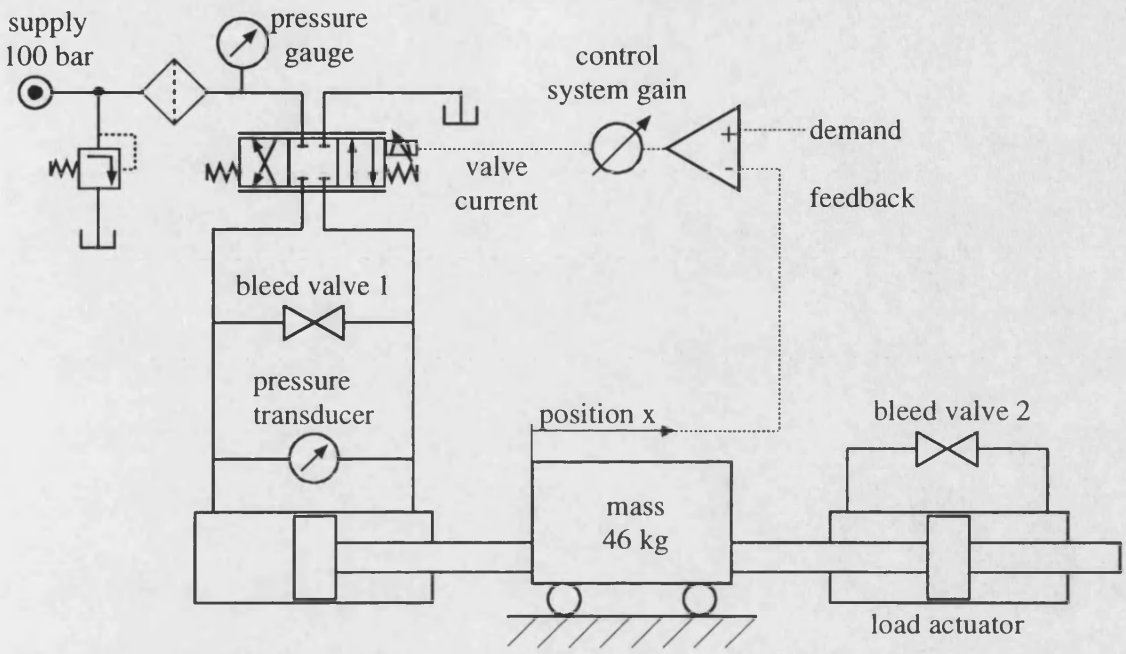


Figure 7.1 Schematic of the experimental actuator test rig

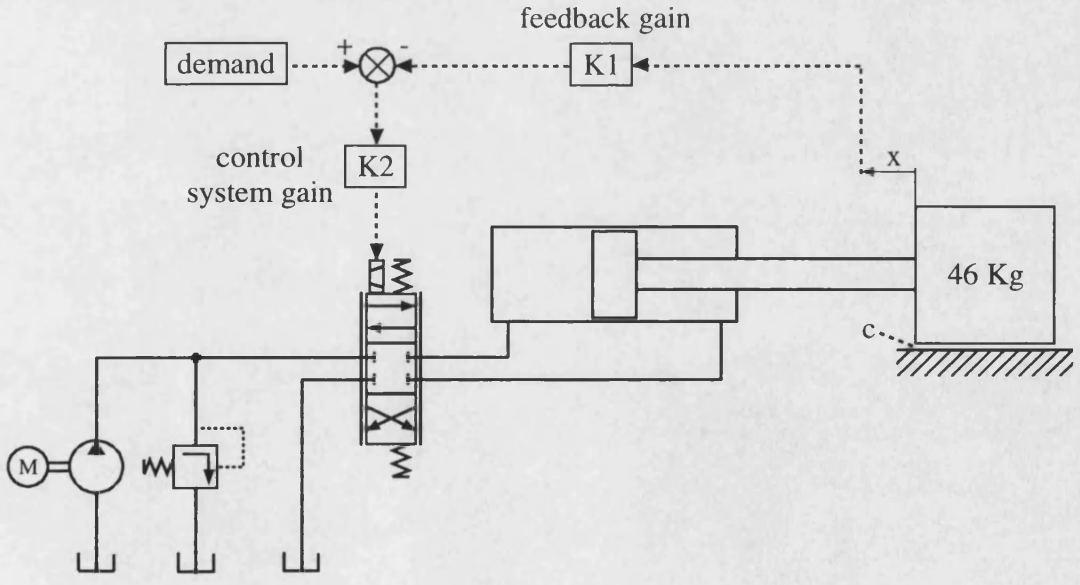


Figure 7.2 Schematic of simulation model

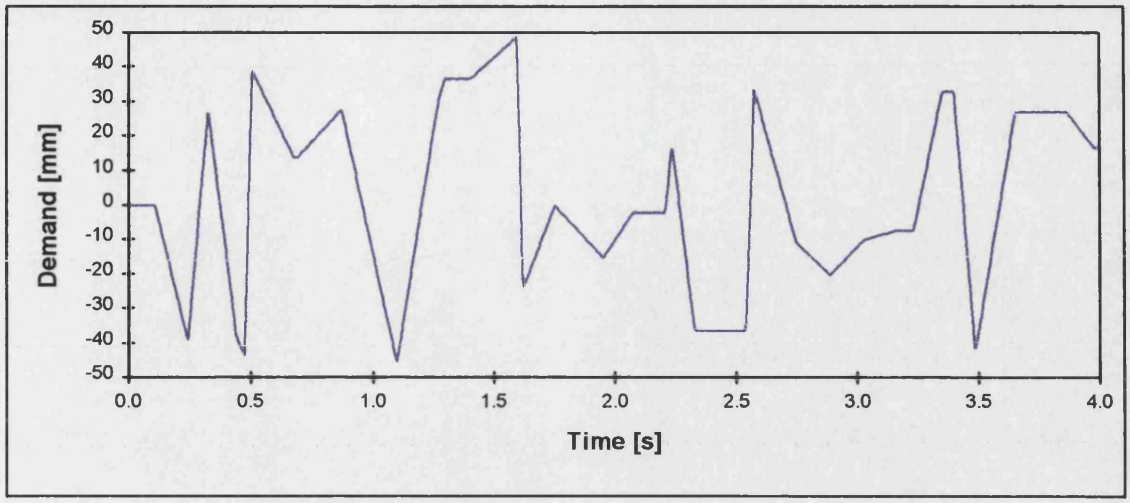


Figure 7.3 Demand duty cycle

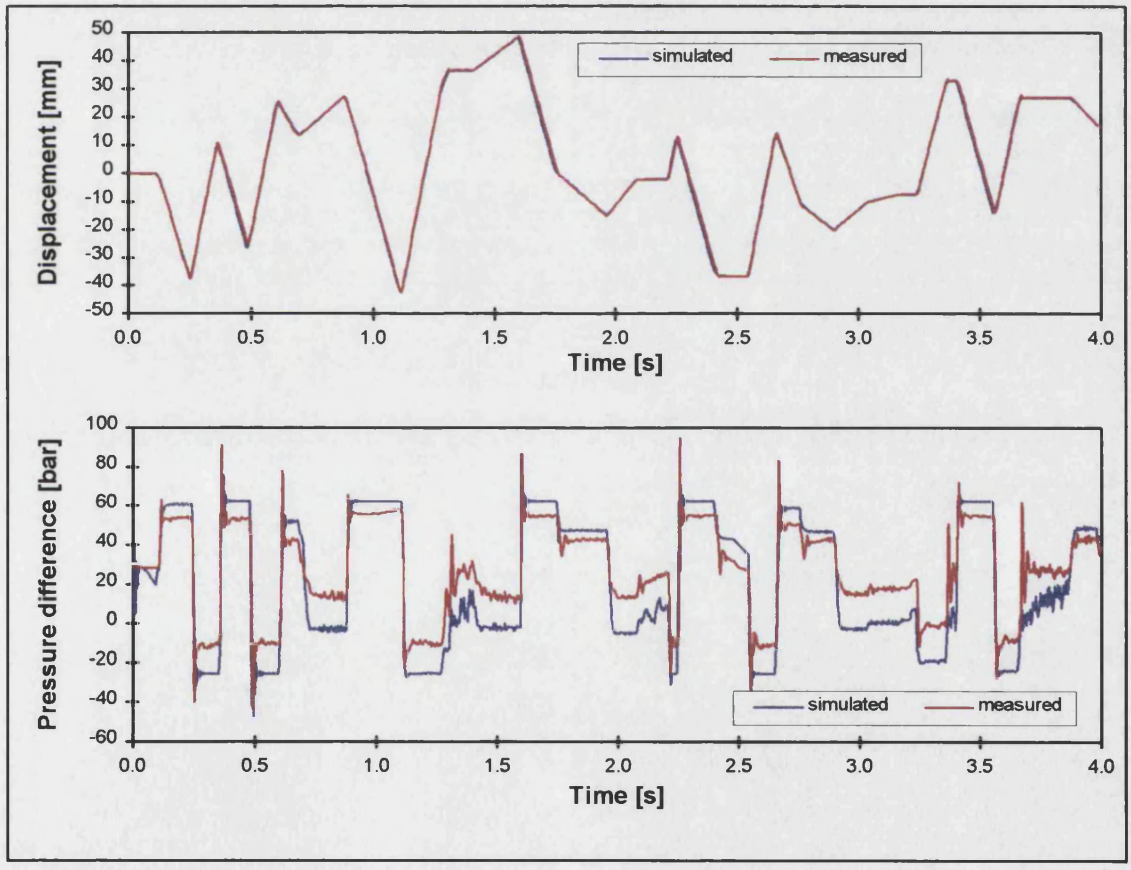


Figure 7.4 Comparison between measurement and optimised simulation ($obfn_x$)

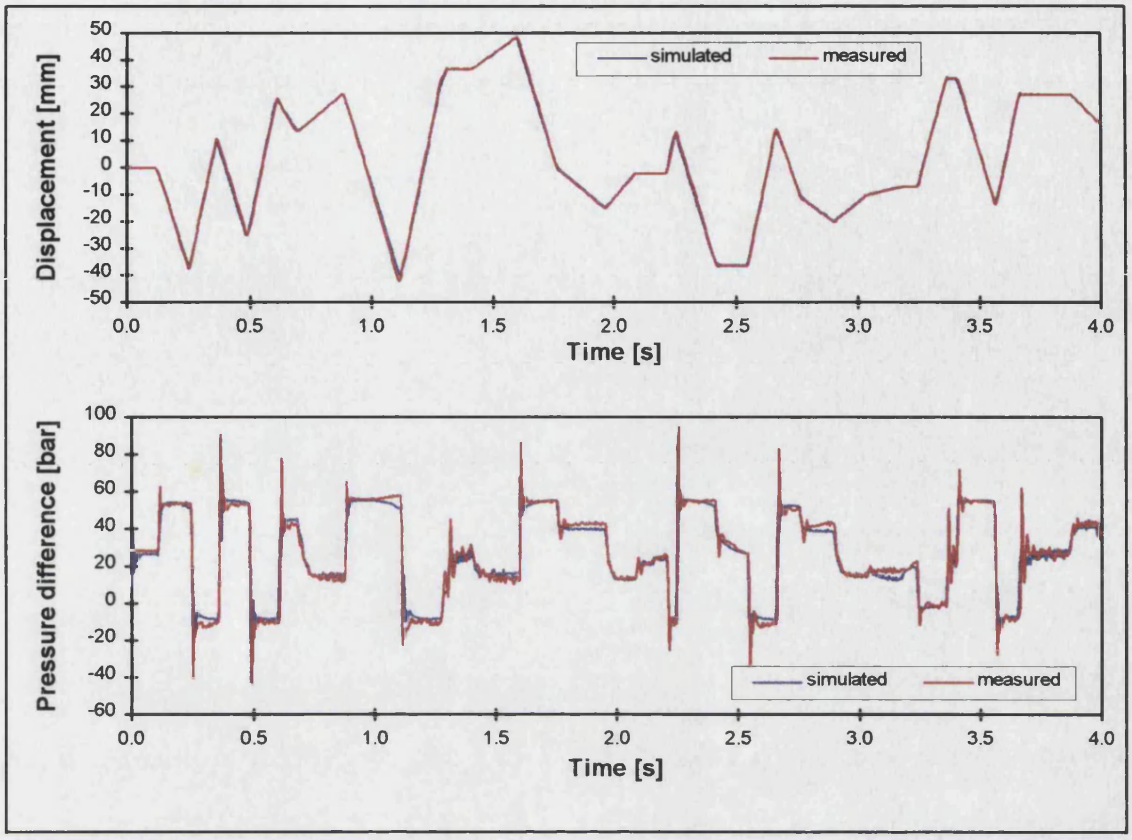


Figure 7.5 Comparison between measurement and optimised simulation ($obfn_p$)

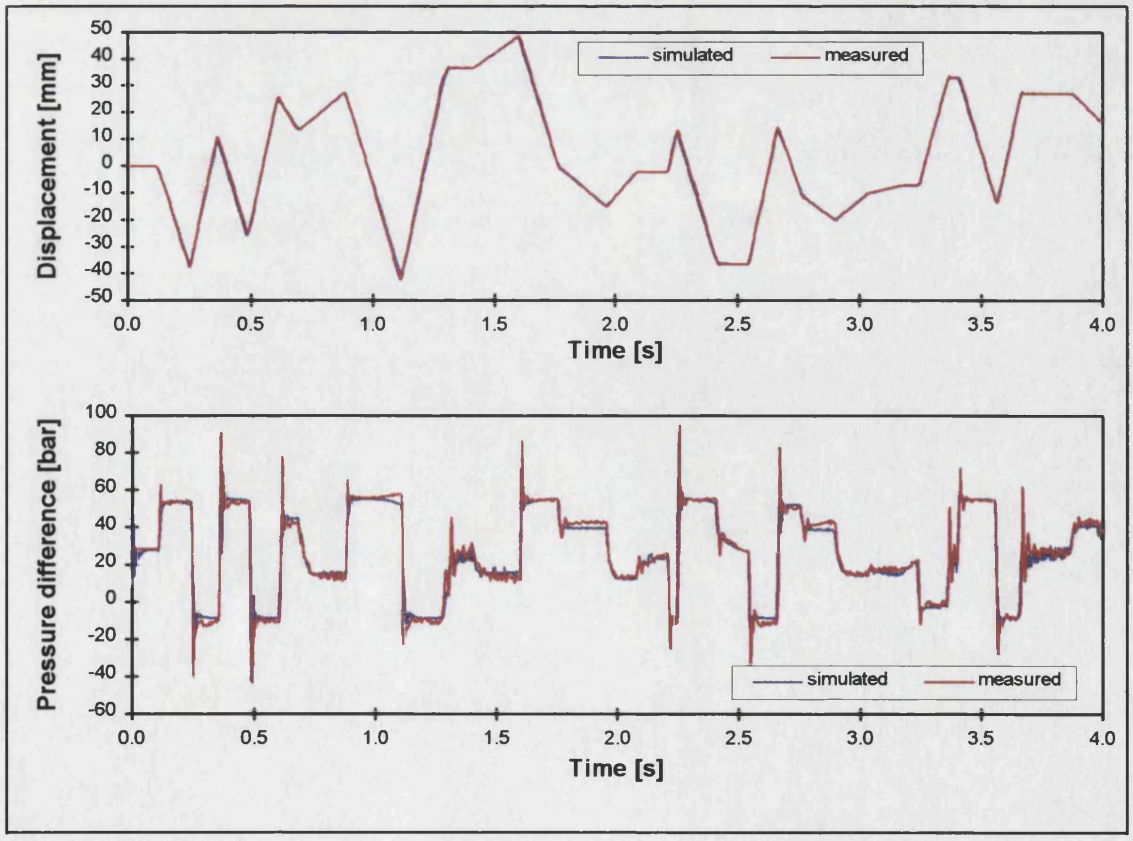


Figure 7.6 Comparison between measurement and optimised simulation ($obfn_{5x+D}$)

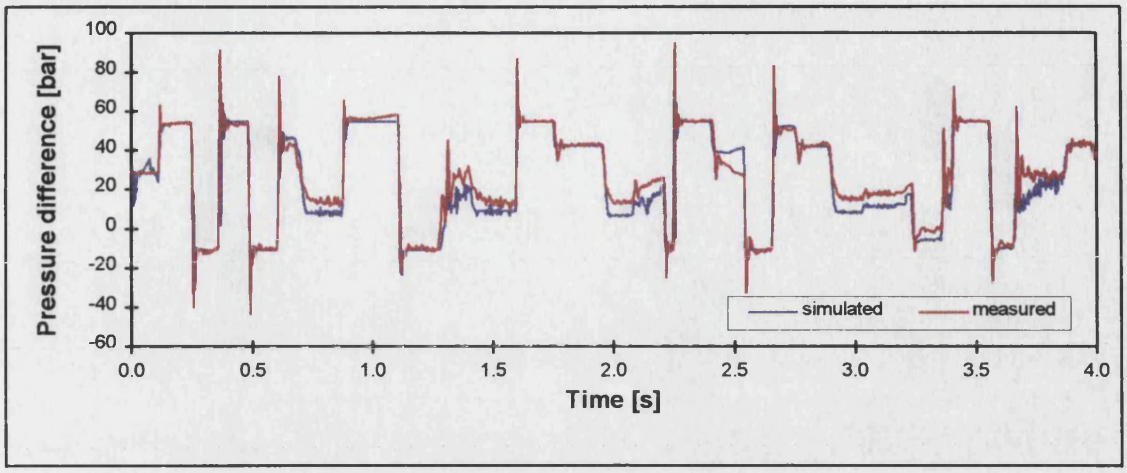


Figure 7.7 Optimisation results (opt. 1)

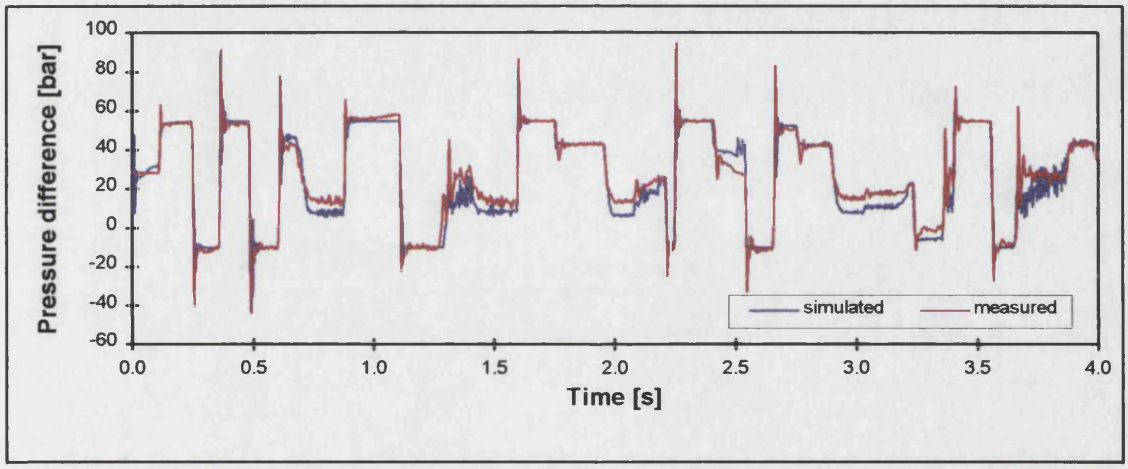


Figure 7.8 Optimisation results (opt. 2)

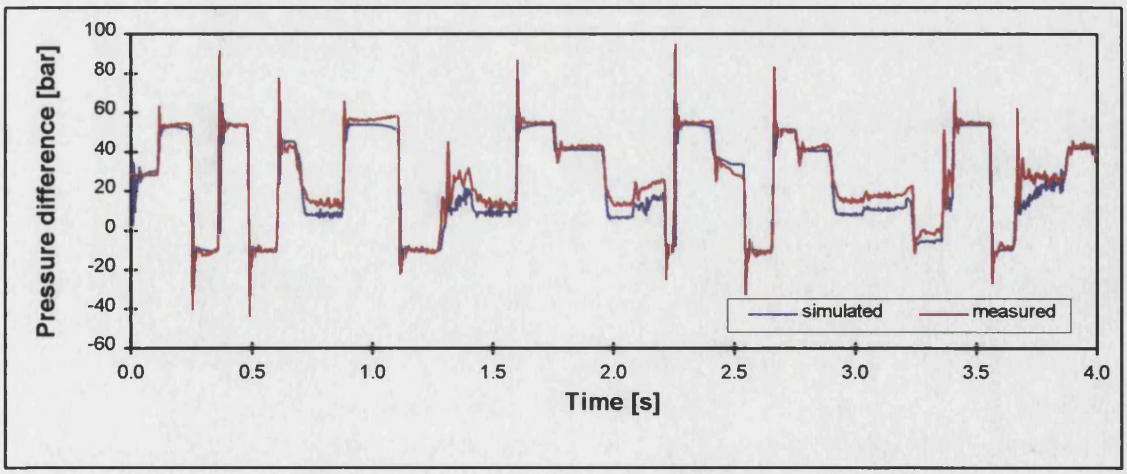


Figure 7.9 Optimisation results (opt. 3)

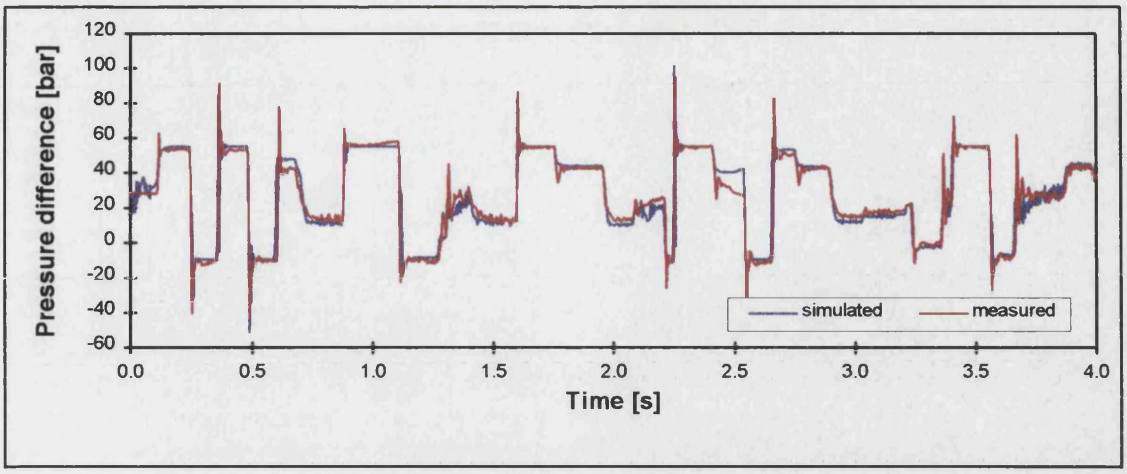


Figure 7.10 Optimisation results (opt. 4)

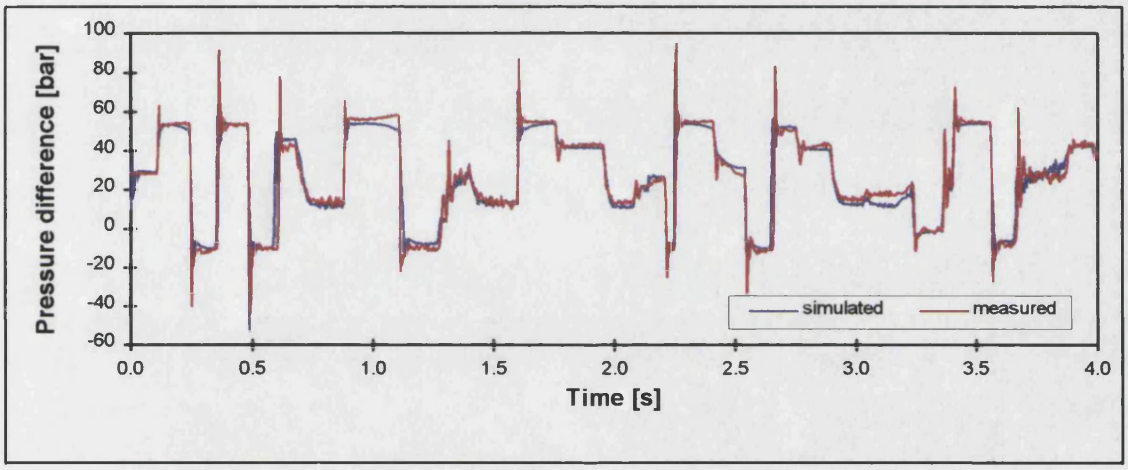


Figure 7.11 Optimisation results (opt. 5)

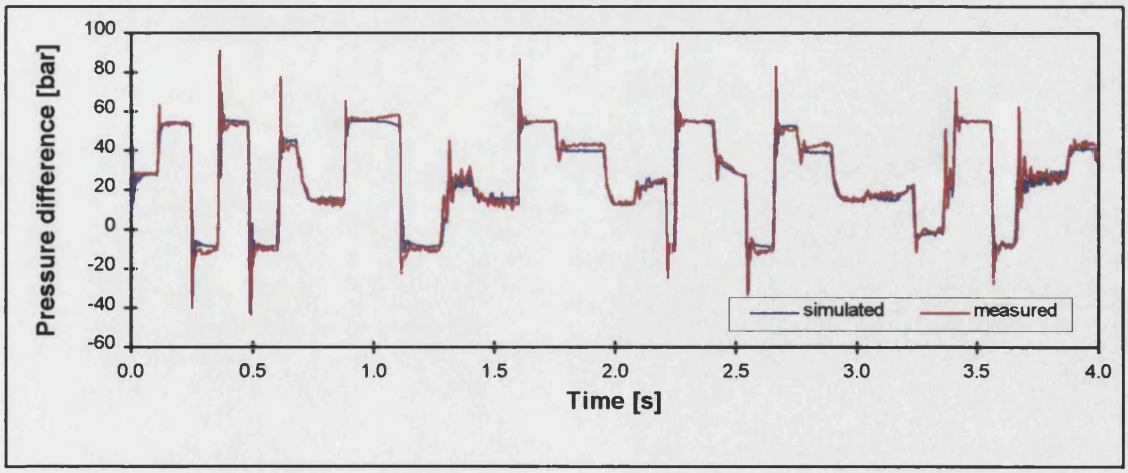


Figure 7.12 Optimisation results (opt. 6)

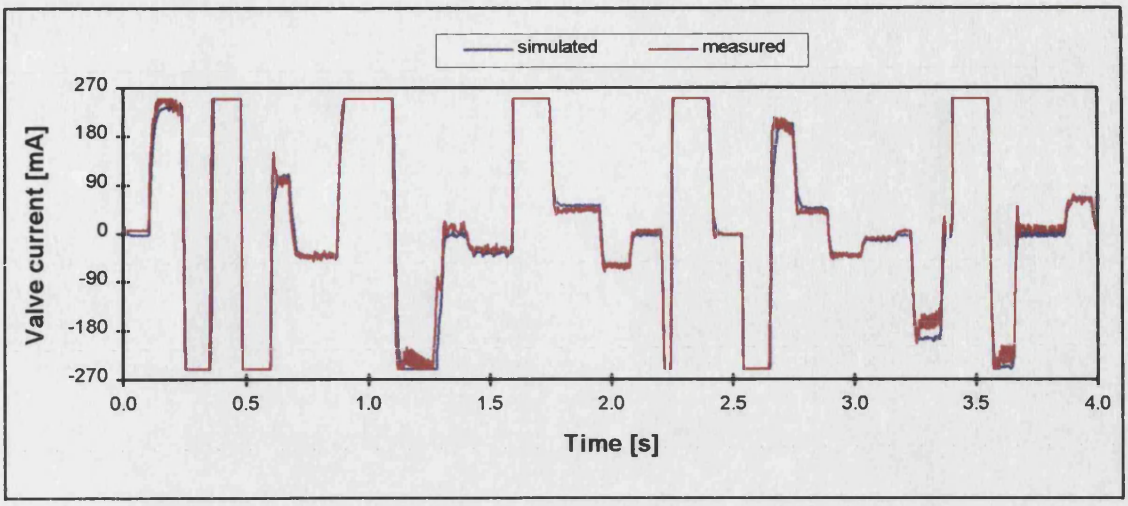


Figure 7.13 Additional optimisation results (opt. 6)

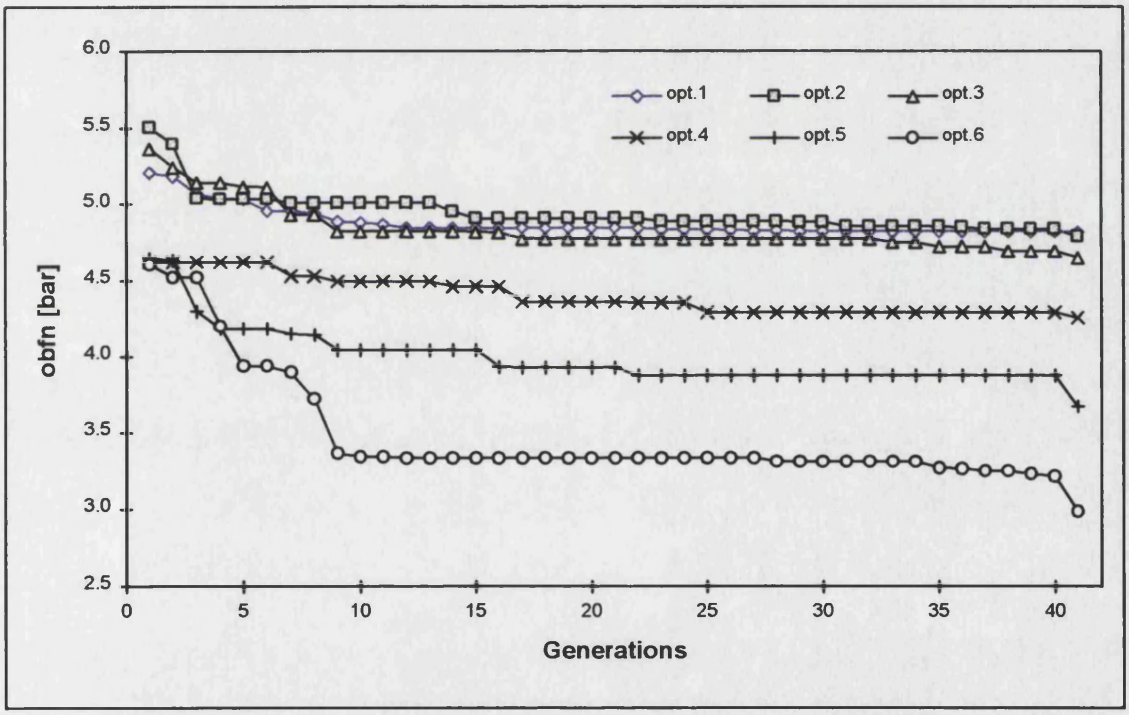


Figure 7.14 Improvement of obfn over the number of generations for 6 optimisations

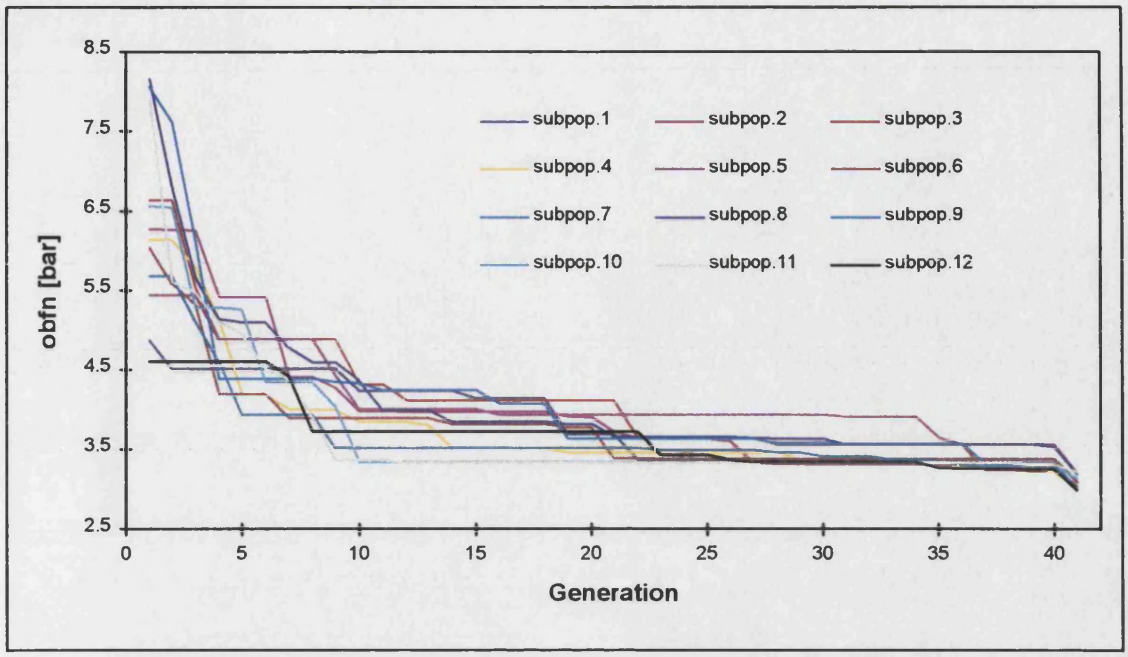


Figure 7.15 Comparison of obfn-value between different subpopulations

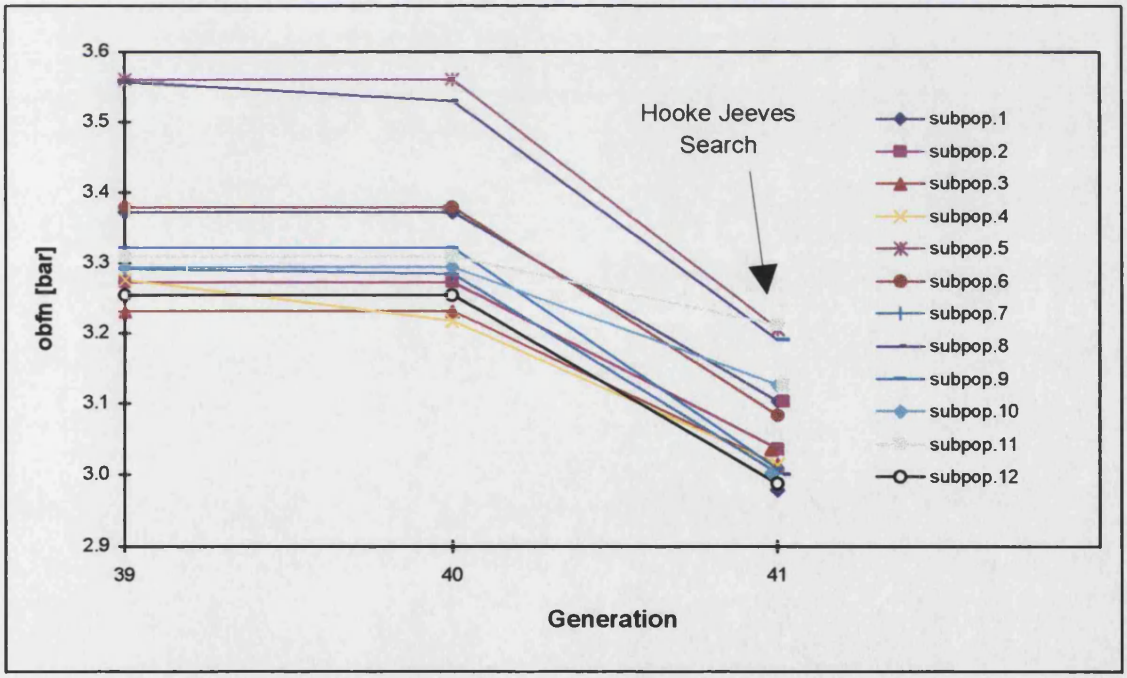


Figure 7.16 Details of Figure 5.15

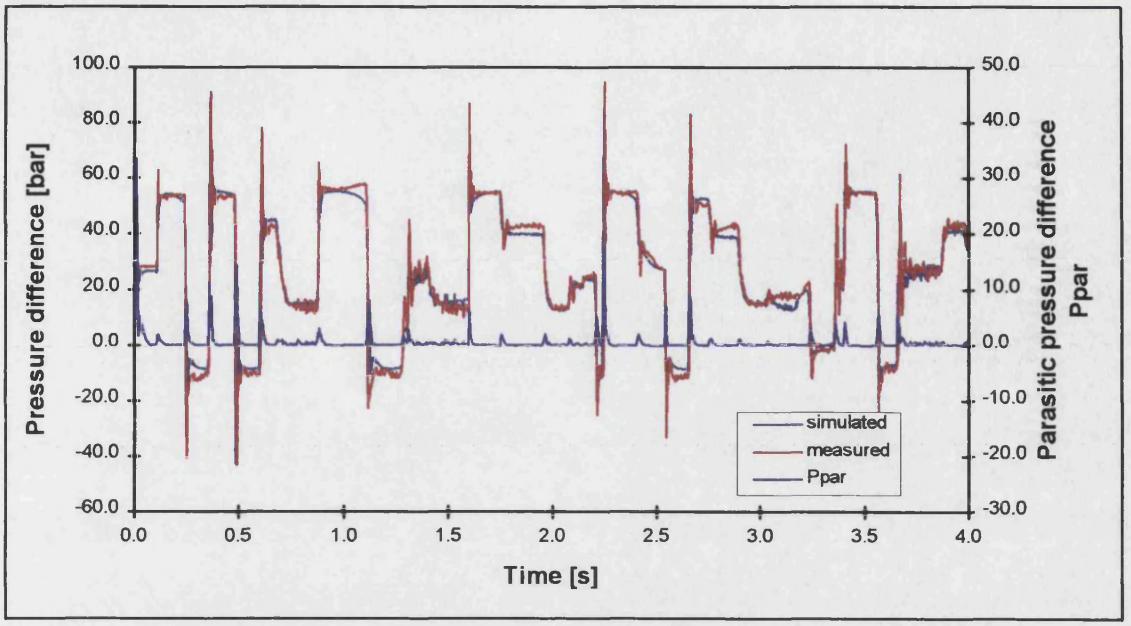


Figure 7.17 Simulation results and parasitic pressure difference

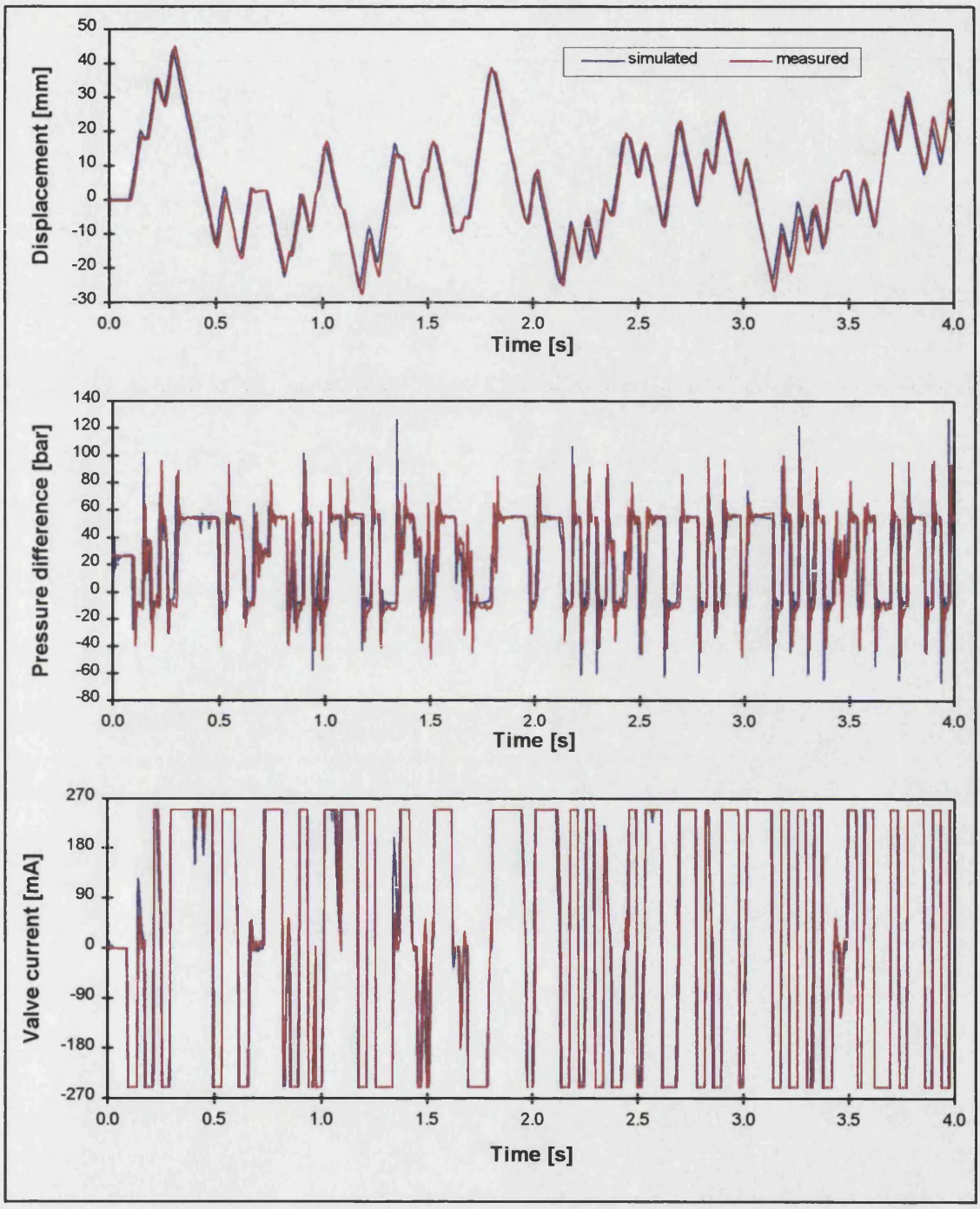


Figure 7.18 Results using different duty cycles

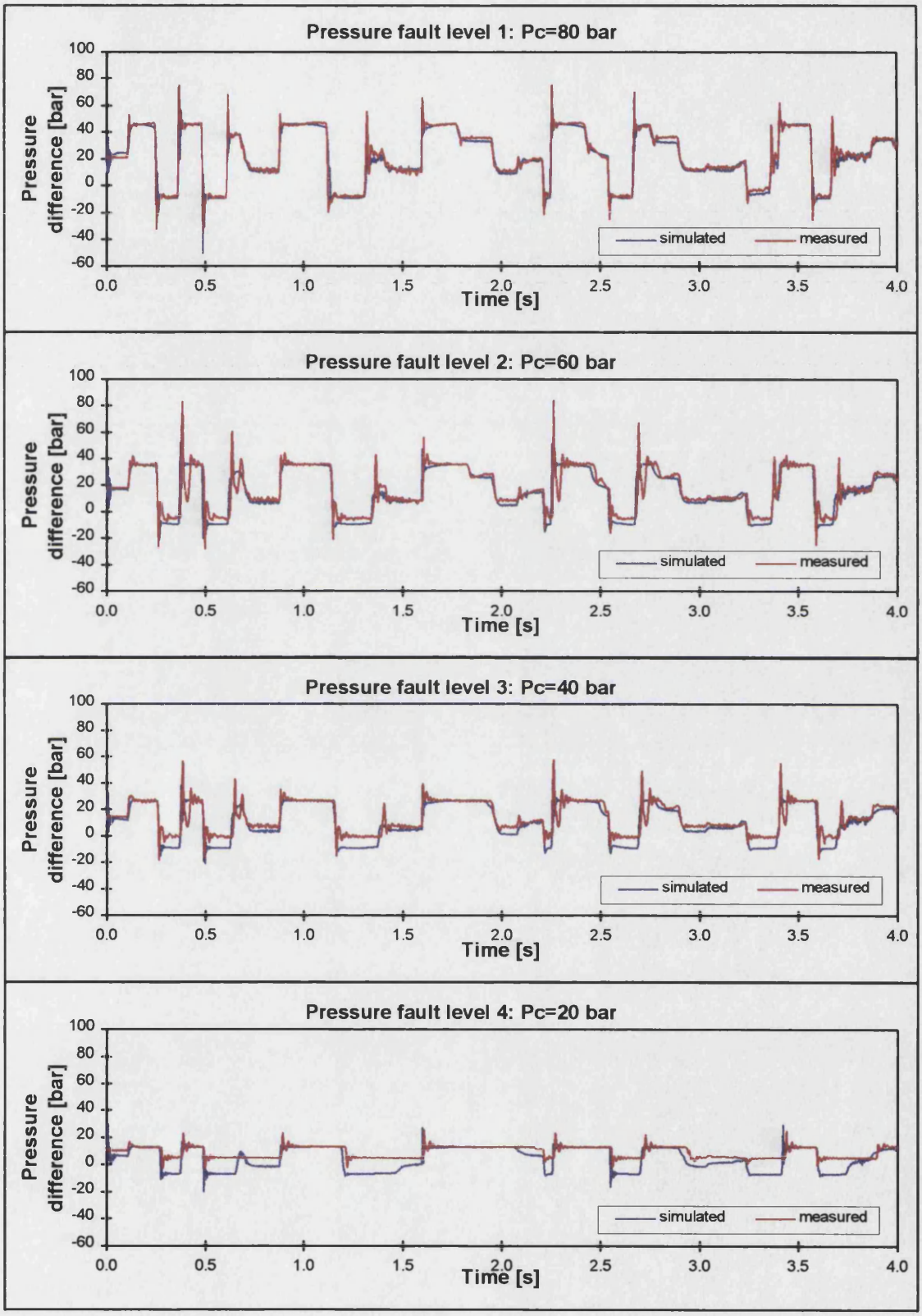


Figure 7.19 Pressure fault level identification results

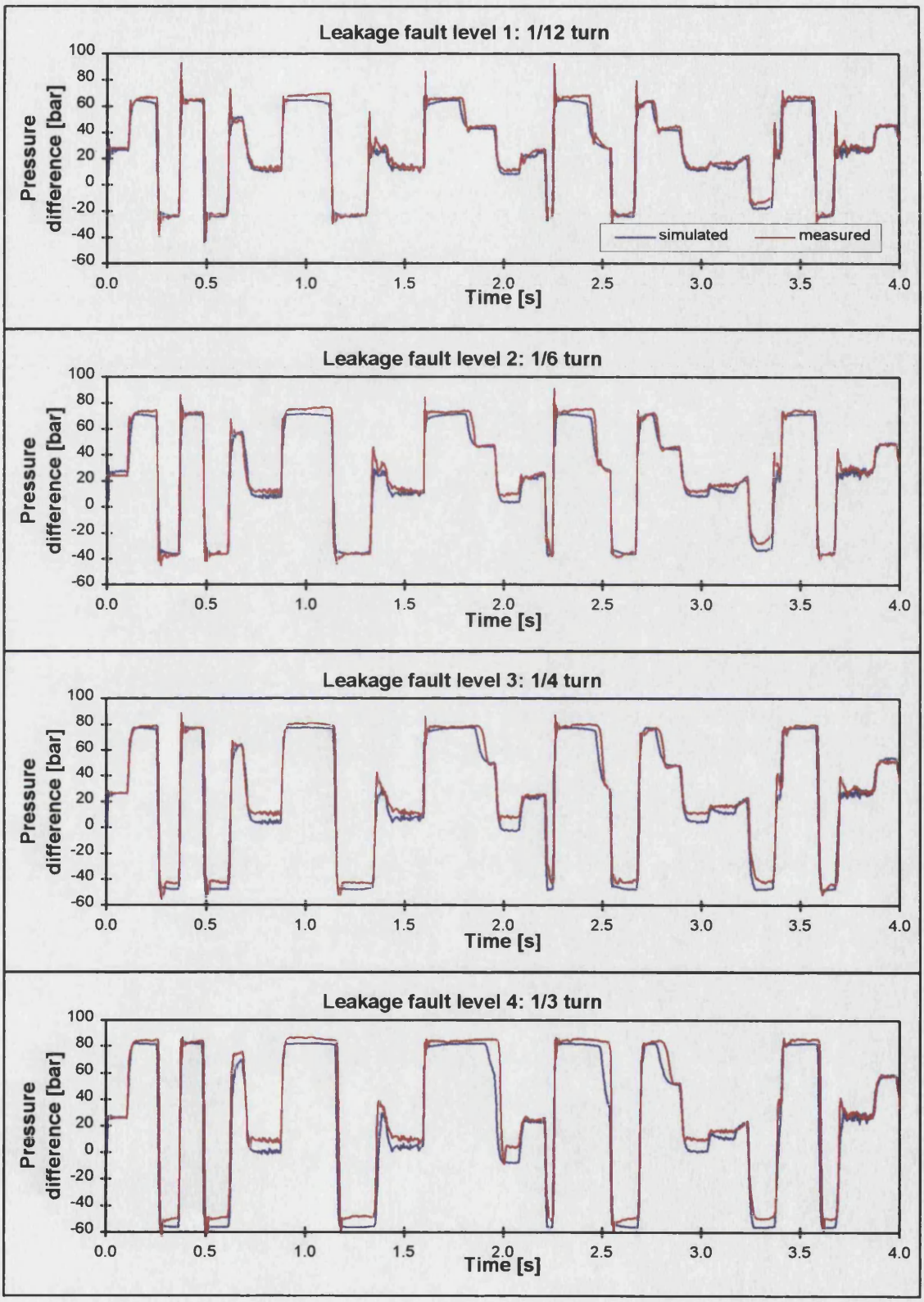


Figure 7.20 Leakage fault level identification results

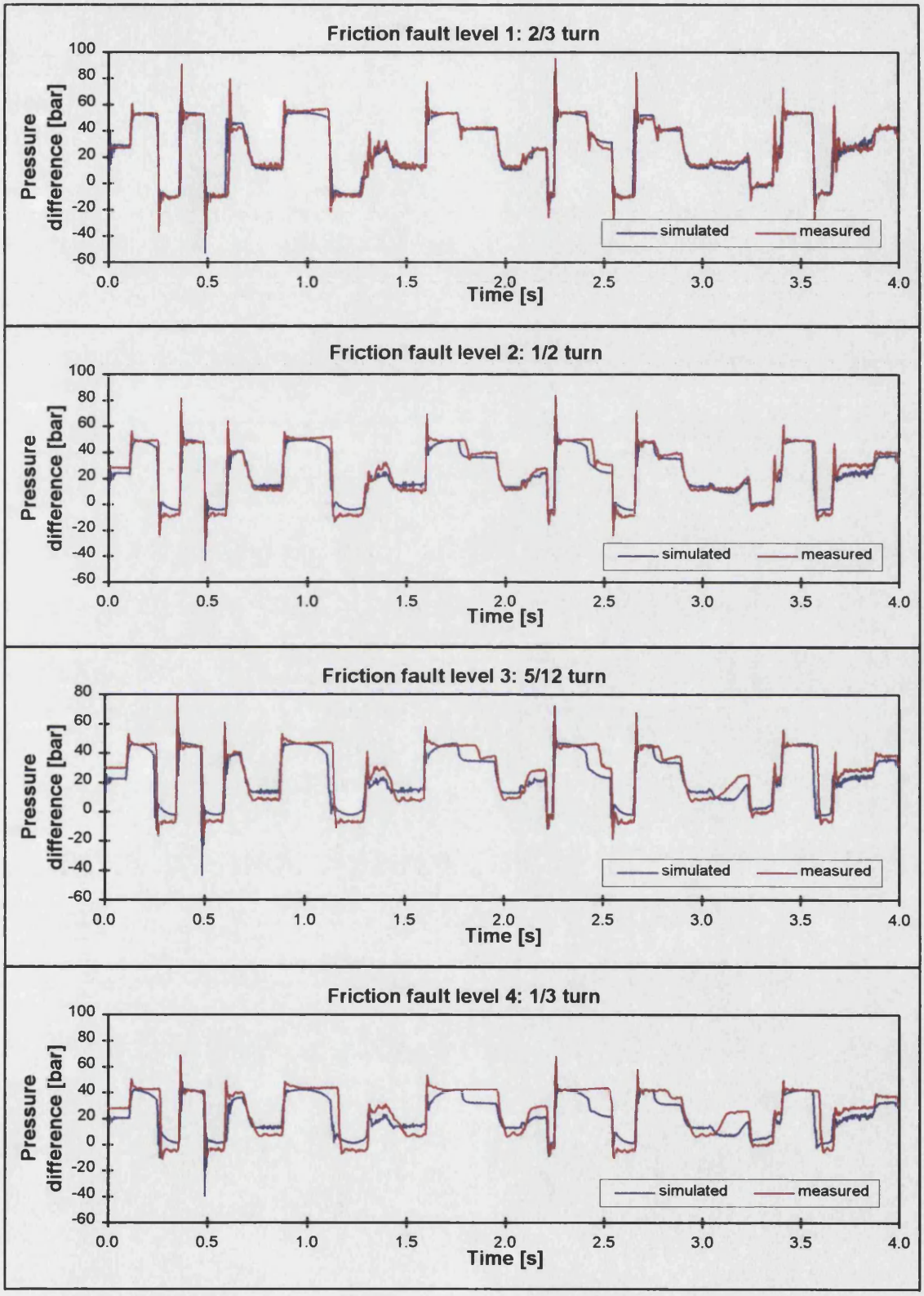


Figure 7.21 Friction fault level identification results

8 Condition monitoring of fluid power systems using neural networks

8.1 Introduction

Malfunctions of hydraulic components increase the operating costs of any fluid power system. Furthermore the consequences of a gross accident can be very serious for system operators and the environment. A complete failure of equipment is usually relatively easy to detect, but when failure has occurred, considerable damage may have taken place. Therefore, it is desirable to have a monitoring system enabling the identification of small deteriorations in order to predict incipient faults which might otherwise lead to a complete break down or other catastrophic events.

This chapter investigates the condition monitoring process of fluid power systems. Employing artificial neural networks different faults can be identified correctly. An approach will be developed that enables the detection of different levels of faults and deterioration in real-time. The purpose of this section is not to determine causes but rather to discuss techniques which identify the presence of failure characteristics and its source. First a review of condition monitoring techniques is given and then theoretical and practical applications of neural networks are investigated.

8.2 Fault monitoring techniques

Generally, faults can occur in sensors, actuators, controller hardware or software, the process itself and to structures (pipes, beams etc.). In this thesis only the component fault detection is investigated, i.e. instrument faults and other faults are not considered. The terms fault and failure are used interchangeably, but a subtle difference does exist. Faults occur when the components in question operate incorrectly in a permanent or intermittent manner whereas failure denotes a complete operational breakdown. Related to this terminology is the notion of fault tolerance which describes the ability of a system to maintain tolerable performance during the occurrence of malfunctions.

In general, fault monitoring systems should perform the following tasks:

- fault detection indicating when the failure happened
- fault isolation which leads to a list of possible faults
- fault identification which indicates the cause and severity of a particular fault

Based on the results of the fault detection and diagnostic procedure, actions have to be released. For example an automatic protection system may be initiated, or maintenance or repair steps may be started [Isermann & Freymuth, 1993]. Another option is the reconfiguration or restructuring of appropriate control laws, to effect tolerable operation of the system if possible. The performance of the above described monitoring tasks, should also meet the following requirements:

- Low false-alarm and low missed-alarm rate, i.e. high rate of correct detections.
- The delay time between a fault occurrence and a fault detection should be small, i.e. the system needs to satisfy real-time requirements.
- Early detection of developing deterioration and warning off incipient faults. The earlier an impending fault is detected the longer one has to plan maintenance.
- A high accuracy of the estimated fault parameters is desired, i.e. the ability to correctly distinguish (isolate) faults, to characterise the size and time of occurrence of faults.
- The employed method must be insensitive to model inaccuracies (if a mathematical model is used). It should also be able to work with noisy data and unknown disturbances.

Incorporating a fault monitoring system into an industrial process results in improved reliability, maintainability and survivability. Reliability deals with the ability to complete a task satisfactorily and within the period of time over which that ability is retained. Maintainability concerns the need for repair and the ease with which repairs can be done and survivability relates to the likelihood of conducting an operation safely whether or not the task is completed. The following sections give an overview of some fault monitoring techniques.

8.2.1 Limit checking

In this research a fault is to be understood as nonpermitted deviations of characteristic properties of a fluid power circuit. If these deviations influence the measurable variables of the system, they may be detected by an appropriate signal evaluation. The corresponding fault detection consists of checking the measurable variables with regard to a certain tolerance of the normal values.

Often the plant measurements are compared to preset limits and exceeding a limit indicates a fault situation. In many systems, there are two levels of limits: the first level

serves for prewarning only, while the second level triggers emergency action [Pouliezos & Stavrakakis, 1994]. The above described (classical) way of limit value checking may be appropriate for the overall supervision of a complete system. However, developing internal (component) faults are only detected at a rather late stage and the available information does not allow an in-depth fault diagnosis. Since a single component fault may cause many plant variables to exceed their limits, fault isolation is very difficult. While very simple, this approach has another serious drawback. Since the plant variables may vary widely due to variations in its duty cycles and inputs, the test thresholds have to be set quite conservatively.

If the supervision is going to be improved and automated, a first step consists of adding more sensors. It is usually desirable to add such sensors which directly indicate faults. Because the number of sensors, transmitters and cables increases, the cost goes up and the overall reliability is not necessarily improved. Furthermore many faults cannot be detected directly by available sensor technology [Pouliezos & Stavrakakis, 1994].

8.2.2 Statistical methods

In order to improve operational safety of a system preventive maintenance can be applied. Based on the statistical life expectancy of individual components, all parts which are likely to fail in the near future are replaced. The expected component life time will be much smaller than the actual time that several of the components could work safely. This means the method (which needs to apply conservative life time thresholds in order to achieve high safety standards) leads to the replacement of many fault-free components, i.e. the approach is quite costly.

Other statistical methods are based on the measurement of system states. Under fault conditions, the movement of the system states are usually complex such that statistical distinctions are sufficient to detect the variation from normal behaviour. A set of several successive measurements called a window, is collected. In real-time applications the window must be sliding. On this window, statistical properties of the measured signals are studied. For example let $\gamma(k)$ be a white noise sequence with variance σ^2 and let $y(k)$ be the observations sequence such that:

$$y(k) = \mu(k) + \gamma(k) \tag{8.1}$$

with the known mean $\mu(k)$. Any kind of fault occurrence makes the residual

$$\gamma(k) = y(k) - \mu(k) \qquad (8.2)$$

depart from its zero mean (variance σ^2). Further details about statistical methods can be found in Pouliezos & Stavrakakis [1994].

8.2.3 Condition monitoring based on vibration measurement

The operating condition of a machine can be monitored by analysing the generated vibration and/or noise. These vibrations and noise are produced by mechanical forces and can be used to reveal faults in the mechanism itself or some changes in the vibration path. The fault revealed may be an actual malfunction of a change in an operating parameter of the machine. In the latter case, the operating condition may be changed by a control system, but if a fault is detected, the machine needs to be scheduled for maintenance. Changes in the vibratory path may signal the need for the repair or replacement of structural elements.

The traditional vibration analysis methods are essentially energy methods and therefore only detect faults that generate significant amounts of changes in energy. When vibration is measured from a transducer mounted on the casing of a machine, what is actually measured is the original force signal from the source of the signal, modified by the characteristics of the transmission path from the source to the measurement point. Expressed in terms of frequency this modification is a multiplication by the mobility of the transmission path. This means the transducer should be mounted as close as possible to the particular component that is to be monitored. A developing fault will show up as a change in vibration and noise. If the system is linear, a relative change in the vibration at the source will give the same relative change in the vibration at the measuring point. Hence it is the relative change which is important and this is mainly used for fault identification. Different faults will manifest themselves at different frequencies in the vibration spectrum. A Fourier transform can be applied to identify the respective frequencies. The obtained frequency domain information can then be related to periodic events in the system, for example events related to shaft speeds. A disadvantage of the vibration and noise measurement methods is that in order to achieve a performance data base one needs to measure the system at many different working points. These measurements include faulty and fault-free cases at for example different system pressures, shaft speeds, temperatures etc. The change in the vibration or noise spectrum needs to be recorded for all these system states and then particular changes may be associated with certain faults.

8.2.4 Expert systems

An expert system, in its most basic sense, is no more than a tool to organise and codify for the computer the experience and thought processes of a human with expertise concerning the operation of a technological process or an industrial plant. The process of building an expert system consists of two main activities which usually overlap: acquiring the knowledge and implementing the system. The acquisition activity involves the collection of knowledge about facts and reasoning strategies from the domain experts. Most of the past applications involving diagnosis have been implemented as rule-based systems. That is, they use simple production rules to provide a mapping between the possible causes and inputs of a system and the possible faults [Pouliezos & Stavrakakis, 1994].

Rule-based systems may be very successful in finding faults when applied to systems from which extensive operational experience has been learnt and stored in a (large) data base. However, due to the nature of their knowledge representation (in terms of empirical symptom-failure associations), the rule-based approaches to diagnosis are usually restricted to familiar systems. The structural properties of the physical system under consideration are only implicit and, hence, they do not support the development of a diagnostic system based on fundamental theoretical principles. For many actual physical systems, a large amount of theoretical design knowledge is usually available and may be very useful for diagnosis. Unfortunately, rule-based systems fail to provide a foundation for using such knowledge without experiencing particular faults or foreseeing potential failures in the physical systems to be diagnosed [Shen & Leitch, 1992]. Despite the usefulness of some expert systems their efficiency may be further limited by the following factors [Rodriguez et al. 1996]:

- It is difficult to provide the correct diagnosis with insufficient or noisy input information.
- The formulation and maintenance of production rules is quite difficult. Eliciting rules from experts is far from being trivial. Moreover, expert systems tend to be rigid in the sense that adaptability of a working system to new functionalities of the elements modelled is very costly. In most practical cases the system has to be rebuilt.
- The effort, in terms of manpower and time, required to design and develop an expert system may be considerable.

- Finally, the need to exhaustively search for all the possible fault hypotheses considerably slows down the global functioning of such systems. The fact that the chain of rules to be fired cannot be predicted in advance makes it difficult to foresee the response time and practically excludes a parallel implementation.

8.2.5 Fault tree analysis

Fault tree analysis has been widely used for assessing the safety and reliability of engineering systems. An established design is considered and the failure of each component assumed. The interactions of these faults are then analysed. This approach may simplify the failure analysis of complex systems and is considered particularly useful as a supplementary analysis to a failure mode and effects analysis. Fault tree diagrams may also be very useful in the training of operating personnel [Collacott, 1977]. Despite its advantages the method also has some drawbacks. There are no positive mechanisms for identifying the important system failure modes to study or the potential component faults within the system, and so the analysis tends to rely heavily on the expertise and experience of the team involved. Consequently, although attempts have been made to formalise the procedure, the technique remains a fundamentally subjective analysis method [Hogan et al., 1993].

8.2.6 Model based diagnostics

This approach is based on models, unlike expert systems and fault trees which are rule-based diagnostic systems. To determine why a physical system has not worked correctly compared with its designed behaviour it is useful to know how it was supposed to work in the first place. It is this simple observation that underlies the considerable interest in the development of model based diagnostic systems (MBD) [Shen & Leitch, 1992].

Model-based diagnostics consists of two stages: residual generation and decision making based on these residuals. In the first stage, outputs and inputs of the system are processed by an appropriate algorithm to generate residual signals which are nominally near zero and which deviate from zero in characteristic ways when particular faults occur. In the second (decision making) stage, the residuals are examined for the likelihood of faults. A decision process may be based on a simple threshold test, on the instantaneous values of moving averages of the residuals, or it may consist of methods of statistical decision theory, such as sequential probability testing or likelihood ratio testing. One residual is sufficient to detect the occurrence of a fault and a set of residuals is required for fault isolation. To

achieve on-line fault diagnosis in the presence of transient behaviour, the system dynamics have to be considered. [Benkhedda & Patton, 1996].

A fundamental issue in the generation of residuals is their robustness (insensitivity) relative to unavoidable modelling errors. Various techniques have been proposed to make the failure detection process more robust. Design methods are proposed with the goal of making the detection filter very much more sensitive to one fault than others. These methods have been shown to be specific cases of the unknown input observer approach. In this approach noise, disturbances, parameter uncertainties and unmodelled dynamics are modelled as 'fault events' of the system, along with the fault events arising from actual system failures. An observer is then designed to be sensitive to a fault event of interest, while insensitive to as many other real and pseudo-fault events as possible. In general, observers are dynamic systems that are aimed at reconstructing the state of a state-space model on the basis of the measured inputs and outputs. The state estimation error is then used as the residual for the detection of the faults.

To avoid complexity in system modelling qualitative reasoning techniques can be employed to describe the system model [Shen & Leitch, 1992]. The physical nature of the investigated signals is, in general, the same for quantitative or qualitative modelling but its description differs severely. In quantitative models the signals are described by time functions which can assume any real value, In qualitative modelling, the value of a single signal or several signals in combination are described by attributes or symptoms, each of which refers to a set of signal values. In diagnosis, an alarm message is a typical qualitative description of a measurement value. The alarm is alerted if the signal exceeds a given bound. Hence, it is only known whether the signal is below or above the prescribed bound. Qualitative models are, in general, relational. Since the system input is known merely qualitatively, the system output cannot be determined unambiguously. As qualitative values can only be combined via qualitative algebra, qualitative models also yield solutions that the system under consideration cannot perform [Lunze, 1992].

8.2.7 Parameter estimation methods

Fault detection via parameter estimation relies on the principle that possible faults in the monitored process can be associated with specific parameters and states of a mathematical model of a process given in general by an input-output relation. Therefore it is necessary to have an accurate theoretical dynamic model of the process in order to apply parameter estimation methods. The problem of parameter estimation has already been described in

detail in Chapter 7. With the described GA-based method multiple fault detection is possible, since the procedure is estimating several values simultaneously. Further details and examples of fault diagnosis with parameter estimation can be found in Isermann & Freyermuth [1991a,b] and Isermann [1993].

8.3 Condition monitoring of fluid power systems

All of the above described fault monitoring techniques can also be applied to fluid power systems. Sato et al. [1990] developed a non-linear observer as part of a control scheme for an hydraulic servo-motor, based on the work of Thau [1973]. This could be readily adopted for condition monitoring purposes, although the system studied by Sato is fairly simple. A fault detection method based on the generation of robust residuals for a bilinear system with unknown inputs was proposed by Yu et al. [1994]. The method was applied to a simple hydraulic system and simulation results are given. These show that the monitoring performance can be improved compared to linear approaches. Yongxiang & Zhangwei [1995] investigate a knowledge-based diagnostic model of an electrohydraulic servo-system. Utilising the hierarchical classification principle the approach leads to efficient condition discrimination and diagnosis.

Two extensive overviews of condition monitoring techniques applicable to fluid power systems are given in Hunt [1986] and Watton [1992]. There are many different parameters that can be monitored in fluid power systems and based on these parameters some monitoring techniques are described in the following sections.

8.3.1 Vibration and noise

The intensity of structure borne noise is about three orders of magnitude higher than the intensity of air borne noise. Hence, the measurement of vibration is more often used for monitoring purposes. By measuring the structure acceleration high frequency contents can be measured very precisely. Several attempts have been made to use vibration analysis to monitor pumps and motors. In Langen [1981] vane pumps have been investigated. In this case frequencies up to 50 kHz were examined and the overall vibration level was recorded. Although pressure pulses were also recorded it was the external accelerometer response which appeared to rise with a vane fracture and later when the control ring developed a small crack. This work was extended to gear pumps by Backé & Langen [1983a,b] where cavitation could be detected in a frequency range of 20 to 30 kHz.

8.3.2 Temperature

Basically the idea of the thermodynamic technique is to measure the temperature rise of the working fluid as it passes through a pump or component. Certain rises in temperature are normal due to pressure changes but, if a change in flow characteristic occurs, the rise or drop will be modified. In pumps this method can be used to measure their efficiencies. The measurement of temperatures is seen as an effective means of monitoring the condition of a pump but care would be needed in environments with varying atmospheric temperature. The absolute temperature of the oil can also influence the actual temperature rise [Hunt, 1986].

8.3.3 Pressure

Pressures can conveniently be monitored by pressure transducers. A fault in the system can then by identified by the loss of pressure indicating for example a fracture in the pipework. This is not primarily used as a monitor of specific components but rather of the complete system. There are several users of such monitoring systems [O+P, 1978]. Pressure pulsations when considered as fluid borne noise can be analysed similar to the vibration measurements. Furthermore, a pressure drop across a filter indicates the condition of the filter element. By looking at this pressure drop over time it can also indicate the increase in particles in the fluid.

8.3.4 Flow and leakage

The measurement of leakage flow is an effective method of monitoring fluid power components such as pumps and motors. Similar to pressure transducers flow meters can also be used to monitor complete systems. A disadvantage of flow rate measurements is that rapid changes cannot be readily measured. Hence, these measurements are mainly used to monitor normally steady or slowly changing flow rates. By measuring the change in oil level in the main supply system tank one can also identify increasing external leakage of the system as mentioned in O+P [1978].

8.3.5 Contamination and fluid condition

The contamination of fluids is a major cause of failure in hydraulic systems. Monitoring the number of particles in the oil can indicate when the filter needs to be changed. This can reduce the wear of parts and prevent or postpone the occurrence of faults. For the actual condition monitoring task the particles can also be analysed in detail. This might indicate whether the particles are externally added or whether they are from specific components of

the system. Even when using this detailed information the problem of identifying faulty components remains very difficult. Physical and chemical changes in the fluid are also important as these might lead to corrosion of surfaces or swelling of seals. Changes might be detected for example in water content, viscosity and loss of additives. Until recently the monitoring of such properties was restricted to off-line monitoring. O+P [1995] describe a system developed for the automatic monitoring of the above described fluid properties. This enables on-line fluid monitoring supported by software which automatically adjusts the required measurement intervals.

8.3.6 Power consumption and efficiency measurements

The measurement of power consumption is very much dependent on the work duty cycle. Hence this method is only feasible where work cycles are know or where the cycles are similar over the period of monitoring. The same problems apply to the measurement of the efficiency of a system or component which, in addition is complicated and can be costly

8.3.7 Other

Depending on the actual application monitoring of the function of a system itself can be used for the monitoring process. The state of hydraulic components can be concluded by investigating for example torque, actuator displacement or velocity measurements.

8.4 Artificial neural networks for condition monitoring

As the name suggests an artificial neural network (NN) is inspired from the intelligence processing that occurs in biological systems. It is the properties of the brain, such as sensing, recognition, weighting of simultaneous inputs, learning, abstraction and generalisation, which NNs attempt to mimic. These properties are attractive to apply in many engineering and scientific areas. [Liu et al. 1992]. Furthermore, NNs are noise tolerant and enable efficient parallel distributed processing.

Recently, NNs have been found to be very powerful in solving control problems of industrial systems. Many control applications make use of a process model abstracting the relationships between inputs and outputs. By using NNs as non-linear (black-box) models it is not necessary to have explicit knowledge about this relationship and a very good representation of processes can be derived. Burton et al. [1991] examined the possibility of using a NN in a simple hydraulic control application. The ability of the system to deal with pattern reproduction, non-linearities and drift was demonstrated. NN-based adaptive

controllers of fluid power drives were investigated by Liu et al. [1992]. Other applications of NN-based control strategies can be found in Govind & Ramamoorthy [1992] and Goh & Noakes [1993]. Aldrich & von Deventer [1993] used NNs to form an internal representation of the relationship between the distributions of the measurement residuals and the residuals of the process constraints. The major advantage of using NNs instead of conventional statistical methods, is that NNs can also be used to detect systematic errors in process systems subject to non-linear process constraints.

The literature gives several more examples where processes are modelled and identified by NNs (Tan et al. [1995], Delgado et al. [1995]). Pollard et al. [1992] demonstrated that improvements in NN identification are possible by building upon established identification techniques. The behaviour of fluid power components was simulated using feedforward NNs by Xue & Watton [1995].

In Dransfield & Stecki [1991] the monitoring of hydraulic systems are seen as another useful application of NNs. So far, mainly simulation studies are published on fluid power applications. A monitor for a simulated hydraulic rotary drive system is investigated in Daley & Wang [1993] where the NN is again used to model the system behaviour. The difference between simulated and predicted performance is then investigated in order to identity faults. Two previous values of shaft speed and pressure difference were used as inputs to the net. Ramdén et al. [1995a] and Ramdén [1995] investigated a hydraulically operated gear box control system. Different types of, and combinations of, directional control valve faults were considered. Usually, the net did not detect all faulty conditions completely, but at least one fault was detected. In this case pressures and piston velocities were used as monitoring parameters. Esser [1992] used the amplitudes of several harmonics of the measured vibration signal for the diagnosis of hydraulic pumps. This means the relevant information about the system and faults is reduced to fewer values enabling a faster network training with smaller networks. The extra time required to calculate the spectrum will slow down this approach considerably.

The classification of individual measurement patterns is, in general, a very straightforward fault diagnosis method. In Sorsa & Koivo [1993] a heat exchanger continuously stirred tank reactor system is studied. For the diagnosis, 14 signals are measured and 10 fault situations are investigated. It was found that multilayer feedforward networks can very reliably classify the training patterns and the performance with the test data is usually satisfactory.

8.5 Description of the neural networks used for condition monitoring

Several specific neural network models have been developed. For this research only feedforward networks (multilayer perceptrons) are investigated. These networks can approximate any continuous function (linear or non-linear) to arbitrary accuracy given sufficient hidden neurons [Masters, 1993]. This class of networks has also been used as the basis for the majority of practical applications of neural networks to date [Bishop, 1994].

8.5.1 The main units of feedforward neural networks

A feedforward network consists of units and directed, weighted links (connections) between them. In analogy to activation passing in biological neurons, each unit receives a net input that is computed from the weighted outputs of prior units with connections leading to this unit. Figure 8.1 shows such a network. Depending on their function in the net, one can distinguish three types of units. The units whose activations are the problem input for the net are called input units; the units whose output represent the output of the net output units. The remaining units are called hidden units, because they are not visible from the outside [SNNS, 1995]. Here only nets with one set of hidden units (one hidden layer) are considered. The net in Figure 8.1 contains n input, m hidden and c output units. These units are often called neurons in analogy to biological neurons. In short the network topology will be described by an n:m:c-network.

In Figure 8.2 details on a single unit are given. These kind of units are used in the hidden and output layers only, i.e. the inputs x_i are propagated directly to the hidden units. The output of the hidden and output units, z and y, respectively, are given by the following equations.

$$z_j = g\left(\sum_{i=0}^{n} w_{ij} x_i\right) \tag{8.5}$$

$$y_k = g\left(\sum_{j=0}^{m} w_{kj} z_j\right) \tag{8.6}$$

where w_{ab} is the weight from input a to unit b, x_0 and z_0 are bias parameters which are set to one and $g(\ )$ is a non-linear activation function. Most networks of practical interest make use of sigmoidal (meaning S-shaped) activation functions. A common choice is the logistic sigmoid given by equation 8.7 and plotted in Figure 8.3.

$$g(a) = \frac{1}{1+e^{-a}} \tag{8.7}$$

With sigmoidal hidden units, the universal approximation properties of the network hold even if the output units have linear activation functions [Bishop, 1994]. For the applied networks sigmoidal units were used for both, hidden and output units.

8.5.2 Network training and validation

A neural network is (usually) trained either supervised or unsupervised. The most common is supervised training. One collects many samples to serve as exemplars. Each sample (also referred to as patterns) in this training set completely specifies all inputs, as well as the outputs that are desired when those inputs are presented. Then the patterns are presented to the network one at a time. For each sample, the outputs obtained by the network are compared with the outputs one likes to obtain. After the entire set of training samples has been processed, the weights that connect the neurons in the network are updated. One pass through the training samples, along with an updating of the network's weights, is called an epoch. In unsupervised as in supervised training, one uses a collection of sample inputs. But the network is not provided with outputs for those samples. One assumes that each input arises from one of several classes, and the network's output is an identification of the class to which its input belongs. The process of training the network consists of letting it discover salient features of the training set, and using these features to group the inputs into classes that the network finds distinct [Masters, 1993]. Unsupervised training will not be investigated in this thesis as it is generally not as useful for the fault monitoring process.

The error function describing the quality of a set of weights is given by the sum-of-squares error. Each input vector

$$x^q = \left(x_1^q, \ldots, x_n^q\right) \tag{8.8}$$

from the data set has a corresponding target vector t_q. The error for output k when the network is presented with pattern q is given by

$$e_k = y_k\left(x^q; w\right) - t_k^q \tag{8.9}$$

where y_k and w are the predicted network output and the weights, respectively. The total error for the whole pattern set can then be defined as the squares of the individual errors

summed over all c output units and over all d patterns. This leads to the following mean square error (MSE):

$$MSE = \frac{1}{d}\sum_{q=1}^{d}\left[\frac{1}{c}\sum_{k=1}^{c}\left\{y_k\left(x^q; w\right) - t_k^q\right\}^2\right] \tag{8.10}$$

NN functions depend non-linearly on their weights and so the minimisation of the corresponding error function is substantially more difficult than in the case of polynomials. It generally requires the use of iterative non-linear optimisation algorithms [Bishop, 1994]. The training algorithms used to find values for the weights were standard backpropagation and backpropagation with momentum term. A description of these algorithms can be found in Rummelhart & McClelland [1986] and SNNS [1995] describes the particular implementation. All neural network results shown in this thesis were obtained using the SNNS simulator[1] [SNNS, 1995].

The quality of a trained network needs to be evaluated before it is applied. This process is called validation. Therefore a second set of data samples is used in order to check that the training process did not overfit the training data. In general, the expected error on the validation set exceeds slightly that on the training set. If the difference is large, one must suspect that either the two sets are not representative of the same population, or the model has been overfitted. It is important that the validation set is not used as part of the training procedure.

8.5.3 Determination of network topology

In Figure 8.4 a schematic plot of the test error with respect to the training and validation set as a function of the number of hidden units in a neural network is given. In most applications, the goal of network training is to find a network mapping function which makes the best possible predictions for new data. This corresponds to the network having the minimum test error, given by $m = m_m$ in Figure 8.4. A detailed description of the determination of the best network topology is given in [Bishop, 1994].

This leads to an analogy with polynomial curve fitting or parametric system identification where the stage of model selection is equivalent to finding the number of hidden units for

[1] The SNNS simulator (Stuttgart Neural Network Simulator) was obtained via anonymous ftp from the host ftp.informatik.uni-stuttgart.de (129.69.211.2), subdirectory: /pub/SNNS

the net. The stage of parameter estimation can be compared to the computation of the network weights.

8.5.4 Data processing

Feedforward networks with logistic activation functions (as described in section 8.5.1) are theoretically limited to an output range of 0 to 1. The scaling method applied to enable the use of measured data is the common linear mapping where the extremes of the variables are scaled in the range from 0.1 to 0.9. An observed value V is scaled to a presentable value A with the following formula:

$$A = r(V - V_{min}) + A_{min} \tag{8.11}$$

with

$$r = \frac{A_{min} - A_{max}}{V_{min} - V_{max}} \tag{8.12}$$

where A_{min}=0.1, A_{max}=0.9 and V_{min} and V_{max} are the variable's minimum and maximum values, respectively. By inverting equation 8.11 a respective formula can be derived for unscaling the network output. There is another reason for scaling the network training data: In a typical application very different physical quantities may be used as inputs which could differ by several orders of magnitude. For example a load might be of the order 10^5 (N) whereas a displacement might be only of the order 10^{-2} (m). This causes difficulties in network training, since the optimal values for the weights would need to span a similar range, i.e. order 10^7. The scaling of data represents an invertible transformation in which no information is lost.

If a net is designed with n inputs, m hidden and c output neurons (compare Figure 8.1), there will be $(n+c)\cdot m$ weights. The only way to prevent the network from learning unique characteristics of the training set, to the detriment of learning universal characteristics, is to flood it with so many examples that it cannot possibly learn all of their idiosyncrasies. As a rule of thumb, double the number of weights in the network is the minimum number of training samples required [Masters, 1993].

Higher derivatives were important for the identification process, therefore these are also likely to be useful for condition monitoring purposes. As mentioned earlier these signals are not always available in real applications. Hence, several previous time steps are used as additional inputs to the nets. These will enable the network to infer derivatives, if necessary.

8.6 Simulation study using reference model

An extensive simulation study was undertaken based on the hydraulic actuator circuit described in section 7.3.3 (Figure 7.2). Normally, model based diagnostic systems are slow due to the time-consuming calculation of sophisticated models. Furthermore, residuals need to be calculated and only then the process of decision making can be applied. This problem can be overcome by a combination of NNs and the transmission line modelling method which enables the calculation of real-time reference models. Instead of using the calculated residuals for network inputs a different approach is suggested. By supplying the net with data from the faulty system as well as the fault free system, the net can automatically calculate residuals if necessary. This might be particularly useful for the example system. If there occurs an error in the system the control loop automatically tries to adjust for it, i.e. faults in the system may not always be detected in signals from the plant output data itself.

In order to decide what faults to investigate detailed studies about faults of particular components may be performed. These are already available for many systems, for example Heinonen et al. [1995] investigated the behaviour of electro-hydraulic servo valves under different kinds of malfunctions. In this section five different faulty cases of the actuator system (Figure 7.2) have been investigated. These are the following:

- Fault 1: increased leakage across the actuator
- Fault 2: a change in load, i.e. a force applied externally on the actuator/mass
- Fault 3: a change in the mass moved by the actuator
- Fault 4: a drop in system supply pressure
- Fault 5: an increase in friction between the mass and the ground

All faults were simulated for four different fault levels. By accounting for the fault-free case this leads to a 5-class classification problem. The parameter values for the five levels are given in Table 8.1 where level 1 represents the fault-free case.

8.6.1 Training and validation data

In order to obtain useful training and validation data the hydraulic system must be excited with a sufficiently rich input signal. It is also important to obtain data from the whole operating range. The demand duty cycle used as input signal is the same as already described in Chapter 7 (Figure 7.3). In this simulation study, both the training and the

validation data were obtained using this duty cycle. The complete system was simulated with the above described fault levels and the following signals were recorded:

- the demanded actuator displacement (u)
- the valve driving current (i)
- the pressure difference between the lines connecting the actuator and the servo-valve (p)
- the achieved actuator displacement (x)

The simulation time step was 0.4 ms (enabling real-time performance) and the sample time of the recorded data was arbitrarily chosen to $T = 0.01$ s. Some tests and visual inspection of the data suggested that this sample rate and data from five consecutive samples would contain sufficient information for each pattern. To produce a training data set for each fault level, 380 training patterns were obtained. This leads to a complete set of 1900 training vectors, i.e. five times 380. For the validation sets, a gradual change and a step change in the respective parameters were simulated. Because the system was simulated for the same time interval, this also gives 380 patterns for each validation data set. Figure 8.5 shows a schematic of the training process. The training input vector is given by

$$\mathbf{X} = \begin{pmatrix} \mathbf{y}_f \\ \mathbf{u} \\ \mathbf{y}_r \end{pmatrix} \tag{8.13}$$

where $\mathbf{y}_f$, $\mathbf{u}$ and $\mathbf{y}_r$ are the faulty plant output, the demand duty cycle and the estimated (reference) fault-free plant output, respectively, as given in equations 8.14.

Estimated values are indicated by a hat above the variable. For simplicity, the same number of samples was used from all signals, i.e. $n_u = n_p = n_x = n_i = 4$. The order of the different variables is not important and was chosen arbitrarily. Once a network is trained the same variables are used for the actual identification process as shown schematically in Figure 8.6. Linear scaling of the input data was employed as described in section 8.5.4 using the limits in Table 8.2.

$$\mathbf{y}_f = \begin{pmatrix} p(t) \\ p(t-T) \\ \vdots \\ p(t-n_pT) \\ x(t) \\ x(t-T) \\ \vdots \\ x(t-n_xT) \\ i(t) \\ i(t-T) \\ \vdots \\ i(t-n_iT) \end{pmatrix} \quad \mathbf{u} = \begin{pmatrix} u(t) \\ u(t-T) \\ \vdots \\ u(t-n_uT) \end{pmatrix} \quad \mathbf{y}_r = \begin{pmatrix} \hat{p}(t) \\ \hat{p}(t-T) \\ \vdots \\ \hat{p}(t-n_pT) \\ \hat{x}(t) \\ \hat{x}(t-T) \\ \vdots \\ \hat{x}(t-n_xT) \\ \hat{i}(t) \\ \hat{i}(t-T) \\ \vdots \\ \hat{i}(t-n_iT) \end{pmatrix} \tag{8.14}$$

8.6.2 Single fault networks

As an initial simplification, five separate networks were trained, one for each investigated fault case. The identical topology of the networks used can be described by a 35:15:1-network. This means 35 input units, 15 hidden units and 1 output unit were used. The number of input units was derived from the number of samples taken from each network input variable whereas the number of output units depends on the representation of the desired output of the net. Using one output per fault level, indicating only the occurrence of a certain fault level, would not enable the identification of intermediate levels. Hence, only one output was chosen to represent the different fault levels. Several tests indicated that nets with 15 hidden units led to satisfactory performances. The standard backpropagation learning function was applied with the gradient descent parameter $\eta = 0.2$. After 50,000 epochs MSE errors between 0.0005 and 0.0091 were achieved for the different networks. This led to runtimes of about 13 hours on a SUN 20 workstation. Details about the final training errors can be found in Table 8.3.

Unscaled results of the five trained networks queried with the training data are given in Figure 8.7. To improve clarity, these and the following graphs do not show every consecutive neural network output, but no other signal processing is used. Furthermore, if a network output is shown in physical units then the results are unscaled, i.e. the linear scaling process (equation 8.11) is reversed in order to achieve meaningful outputs.

It can be seen that the different nets can classify the data correctly. The prediction quality of the net trained on the fault in mass is not as good as the others. For this initial trials no tests were applied to check whether the system was sufficiently excited. Hence, the chosen

input duty cycle might not provide sufficient excitation of the system, i.e. making it difficult to identify the correct mass. Restricting the training patterns such that actuator acceleration is above a certain limit only can improve the situation.

Four of the nets were trained on fixed output levels, i.e. the network 'only' has to identify five different fault levels. In another test, the network which was trained on a fault in system supply pressure was treated differently. For that, the pressure in the supply line was also recorded (p_{sys}) in order to be used as the desired network output. Here, the net was successfully trained to predict the actual system pressure. This approach actually led to the smallest final MSE error. It appeared to be easier to train the network on the supply pressure than on the actual relief valve setting used to obtain the respective pressures.

In order to evaluate the performance of the network at intermediate levels two validation cases were investigated. Figure 8.8 shows the results for all five networks. Table 8.3 also gives the MSE error for these validation cases. These errors are partly smaller than the training errors indicating a very good generalisation of the trained nets. For validation the networks were queried with data they had not seen before, i.e. using different fault levels. All networks can identify a step change immediately and the agreement for the gradual change in parameter is also excellent. Even individual peaks in the supply pressure were estimated correctly. The net trained on a fault in mass also led to good results, indicating that final training errors of about MSE = 0.01 are sufficiently small, i.e. lead to reasonable network performances. This might enable much shorter training times if the training process is stopped as soon as the MSE error value becomes smaller than this threshold.

8.6.3 Neural networks to identify several faults

To reliably diagnose single faults using individual networks for each fault in a multiple fault environment, it is necessary to train networks using data for all the faults. This has been investigated for faults in leakage and load. It is still assumed that only one fault can occur at a time. By simply adding the two training data sets described above a training set of 3800 patterns was obtained. The two validation cases were added similarly leading to 760 patterns, equivalent to 15.2 seconds in real time. In this case 25 hidden neurons were found to be sufficient, i.e. a 35:25:2-network was used. The two outputs independently indicate fault levels in leakage and load, respectively. It took 36 hours to calculate 50,000 epochs leading to final MSE errors of 0.0044 and 0.0048 for the training and validation data, respectively. Figure 8.9 shows the network output queried with training data. Again, the different faults and fault levels can be identified. The same was found for the

validation cases as given in Figure 8.10 where gradual changes as well as step changes were predicted correctly. This shows that NNs can be trained to be sensitive to more than one fault.

Unfortunately, there are some major drawbacks of this approach. In order to train the net for all possible faults a larger number of training patterns is needed. Furthermore, more hidden neurons are required to enable the classification task. This leads to much longer training times (here 36 h compared to 13 h). Alternatively, assuming the same time spend for training, the accuracy is much lower for nets trained on several faults. These disadvantages become even worse with an increasing number of faults to be monitored. A method to overcome these problems is the following modular approach.

8.6.4 Modular neural network approach

Da & Lin [1995] train two NNs, one for failure location and another for identifying when it happened and how serious the failure is. Here a slightly different approach is proposed. As already described in section 8.6.2 one network can be trained for each fault. Querying all networks with the input training data for every fault case leads to the outputs given in Figures 8.11 to 8.15. The results are shown in scaled form, i.e. all values are between 0 and 1. It was thought that statistical methods could be used in order to identify whether a fault in the particular parameter has occurred. Scattered data could be rejected as 'no fault' or as 'find fault in different parameter'. Unfortunately, this cannot be done because several faults appear as different faults when presented to nets trained on these different faults. For example in Figure 8.14 (the net trained on the fault in supply pressure) the load training data (second from the top) suggests an increasing pressure although the pressure was constant for the fault in load. Statistical methods may pick up this signal as a fault in pressure.

The new approach suggested here is to use the outputs of all networks as input to another (filtering) network which then identifies the correct fault. This method is schematically shown in Figure 8.16. A 25:10:5-network was trained to perform this filtering task. 25 inputs were obtained by using 5 consecutive outputs of the individual (first stage) networks. The training and validation data sets were constructed from 100 samples at each fault and fault level leading to 2480 patterns. After 10,000 epochs (taking 105 minutes on a SUN 20) final MSE errors of 0.009 and 0.06 were obtained for the training and validation cases, respectively. Figure 8.17 gives the 5 network outputs when queried with the validation data set. After every 496 patterns a different fault is introduced, starting with

a fault in leakage (pattern number 1 to 496). Again 5 fault levels are used beginning with the fault free case for the first 100 patterns. The largest faults are only represented by 96 patterns. Due to the use of 5 consecutive samples the last 4 patterns are not complete. During a fault only one of the five outputs should show a higher value for this particular fault. None of the outputs should give unusual values if the system is working normally (fault-free). The obtained validation outputs are already quite meaningful but statistical methods may be used to improve these results. As an example a simple arbitrarily chosen criteria was applied in order to classify the five different fault levels. First, the average of 5 consecutive outputs was calculated (called y_a) from the scaled outputs as returned from the net. Then the following limits are applied:

$$y = \begin{cases} 0.1 & \text{if } 0.0 \le y_a < 0.2 \\ 0.3 & \text{if } 0.2 \le y_a < 0.4 \\ 0.5 & \text{if } 0.4 \le y_a < 0.6 \\ 0.7 & \text{if } 0.6 \le y_a < 0.8 \\ 0.9 & \text{else} \end{cases} \tag{8.15}$$

The employment of this simple criteria led to the results given in Figure 8.18. Here the predictions match the desired fault level quite well although the actual fault level is normally underestimated. The reason for this can be found in the network training data. Out of the 2480 training patterns only about 100 represent each fault level, i.e. there are more examples for fault-free cases. The learning algorithm will expend considerable effort reducing the error for these fault-free cases, while neglecting to fine tune the different fault level predictions. The only way to overcome this problem it to change the NN error function calculation, i.e. errors at fault levels need to be given a (higher) weighting factor. This could not be investigated in this research but considerable improvements are anticipated.

The described modular approach is much more flexible than using one large network for all faults. If a new fault needs to be included in the monitoring process only two (simple) neural networks need to be trained. One is the network sensitive to the new fault and the second is the filtering network. It is expected that more faults (and therefore additional networks) lead to additional information for the filtering network, making it easier to be trained. Another advantage of the modular approach is that it is faster to train several small networks than to train one very large network to the same degree of accuracy.

8.7 Monitoring of experimental systems using reference models

An important goal of this section is to demonstrate the feasibility of using NNs to diagnose faults in experimental data. The study was again undertaken based on the hydraulic actuator circuit shown in Figure 7.1. Some changes were made compared to the simulation approaches presented in the previous sections.

8.7.1 Training and validation data

Although Master [1993] gave the rule of thumb for neural network data with "If in doubt, throw it in.", redundant information makes it more difficult to adjust the weights correctly. In the actuator system, the valve current, i, is proportional to the difference between actuator displacement and its demand, x-u. By supplying the net with all three signals some redundant information is supplied, hence from now on only the current, i, and the displacement, x, are used. The pressure difference across the actuator, p, is still included as before.

Figure 8.19 shows the two demand duty cycles used for this investigation. The problem of insufficient excitation has been avoided by driving the system with a persistently exciting position demand signal. A series of random step inputs with a maximum magnitude of 50 mm was chosen. The signal switching time of 40 ms was determined on the basis that the system should not reach steady state. Visual inspection of the experimental data showed features of interest up to frequencies of around 50 Hz. To give adequate resolution of the signals, a sampling rate of 400 Hz was chosen. Here only faulty cases which could be introduced into the rig without causing damage were investigated. These are the three faults already described in section 7.4.5, i.e. changes in friction, leakage and pressure. The parameter identification results obtained in the same section also become very useful for the network training process. These enable the expression of the different friction and leakage fault levels by actual values.

Another difference to the simulation study is the sample time and the number of values used. Both, simulated and measured data were taken (linearly interpolated if necessary) at time intervals of $T = 0.005$ s but 10 consecutive samples were used. A schematic of the training process is shown in Figure 8.20 and a complete training vector is given by

$$\mathbf{X} = \begin{pmatrix} \mathbf{y}_t \\ \mathbf{y}_r \end{pmatrix} \tag{8.16}$$

where $\mathbf{y_f}$ and $\mathbf{y_r}$ are again the faulty plant output and the reference (fault-free) plant output, respectively, as given in equations 8.14 ($n_p = n_x = n_i = 9$). These settings mean that each pattern still contains information about a period of 50 ms. Training and validation data sets were obtained by taking 700 patterns for each of the fault levels given in Table 8.4. This leads to 3500 patterns which were also linearly scaled with the limits already given in Table 8.2. Duty cycle 1 (Figure 8.19) was used as input when recording the training data set and duty cycle 2 was used for the respective validation data. The simulated reference data were calculated for the fault-free cases only. A second set of validation data was obtained by manually changing the respective components during the measurements. Friction and leakage faults were inserted by an approximate step change via opening the bleed valves (compare Figure 7.1). A more gradual change of the pressure was introduced via the supply relief valve.

8.7.2 Single fault networks

Three separate 60:20:1-networks were trained to monitor the different faults. Once again one output was used to represent the different fault levels and 20 hidden neurons were found to work well. The learning function 'backpropagation with momentum term' [SNNS, 1995] was applied (step width of the gradient descent: $\eta = 0.8$, momentum term: $\mu = 0.2$). To reach the MSE errors given in Table 8.5 took 1000 epochs and between 66 and 84 minutes on a SUN 20. Using real measured data and the different learning function enable a faster training process although the net used contained more than twice as many weights (1220 compared to 540). The trained nets were first evaluated with the validation patterns obtained by using a different duty cycle but containing faults at the same fault levels. Figure 8.21 gives the outputs of the three trained nets. All faults are classified correctly, although the predicted faults particularly for the leakage faults are slightly scattered. It seems to be more difficult to extract the leakage coefficient from the supplied data. One explanation for this can be found by looking at Figures 7.19 to 7.21 where the pressure differences are given for the different faults and fault levels. The pressure amplitude and shape is very similar for the leakage fault (Figure 7.20). Furthermore the differences between (reference) simulation and measurements at all fault levels are also relatively small. This indicates that there is insufficient information in the training data.

A second evaluation of the trained nets is given in Figure 8.22 where the manually introduced faults are investigated. All three faults can clearly be identified from the measured data. Note that during the insertion of the faults the network is still able to give

sensible outputs. The fault transients are on a larger time scale than the data window of 50 ms used to make the diagnosis. However, for the fault in pressure, the additional (unlearned) dynamics, due to the change in pressure when the error was introduced, may contribute to the slightly scattered predictions. The leakage fault prediction was expected to lead to worse results, nevertheless, the step change was still clearly identified. It may be noticed that the friction fault output shows a curious behaviour after about 2.4 seconds. The manually introduced fault was leading to a friction outside the range the net was trained for. Due to the scaling process the values cannot exceed the limit of 30844 N/m/s. It remains to be investigated whether faults larger than the maximum value trained for, always lead to predicted values close to the same side of the scaled interval.

8.7.3 Single fault networks using residuals

An approach based on residuals is presented here which is similar to model based diagnostic decision making. Data from the faulty plant and additionally, these signals subtracted form the reference output are supplied to the net as shown in Figure 8.23. A complete training vector is then given by

$$\mathbf{X} = \begin{pmatrix} \mathbf{y}_f \\ \mathbf{y}_d \end{pmatrix} \tag{8.17}$$

where $\mathbf{y}_f$ and $\mathbf{y}_d$ are the faulty plant output and the residuals, respectively, as given further down in equations 8.18 ($n_p = n_x = n_i = 9$).

The reference plant output may be the measured fault-free plant if the working duty cycle is known in advance. In general this is not the case, hence, the simulated reference model is used. By building residuals it was anticipated to achieve more distinct signals for each faulty case. Furthermore, for the fault-free case the network was supplied with the difference between simulation and measurement. These may enable the neural network to account for modelling inaccuracies, i.e. filtering out differences between simulation and measurements.

$$\mathbf{y_f} = \begin{pmatrix} i(t) \\ i(t-T) \\ \vdots \\ i(t-n_iT) \\ p(t) \\ p(t-T) \\ \vdots \\ p(t-n_pT) \\ x(t) \\ x(t-T) \\ \vdots \\ x(t-n_xT) \end{pmatrix} \quad \mathbf{y_d} = \begin{pmatrix} \hat{i}(t)-i(t) \\ \hat{i}(t-T)-i(t-T) \\ \vdots \\ \hat{i}(t-n_iT)-i(t-n_iT) \\ \hat{p}(t)-p(t) \\ \hat{p}(t-T)-p(t-T) \\ \vdots \\ \hat{p}(t-n_pT)-p(t-n_pT) \\ \hat{x}(t)-x(t) \\ \hat{x}(t-T)-x(t-T) \\ \vdots \\ \hat{x}(t-n_xT)-x(t-n_xT) \end{pmatrix} \tag{8.18}$$

A disadvantage of residuals is that they are normally more sensitive to noise, whereas this might not be so important with this approach because the reference signal is noise free. There is no loss of information due to the subtraction of the two signals because the faulty signal is also supplied. All residuals were linearly scaled with the limits in Table 8.6. Training and validation pattern files were obtain with the same settings given in the previous section. The calculation of 1000 epochs took between 66 and 71 minutes (SUN 20) and led to the final MSE errors given in Table 8.7. Again separate 60:20:1-networks were trained for each fault using the same learning function as before.

Figures 8.24 and 8.25 give the respective evaluation results. All faults are again classified correctly. The step changes in friction and leakage are predicted more clearly whereas the fault in pressure somehow led to worse results. It is not clear why this is the case. Furthermore, it is not conclusive whether the residuals improve the network performances. Both approaches might lead to more distinctive results when the nets are trained for more epochs, but this was not investigated. The calculation of residuals leads to a computational overhead which can be avoided by using the approach described in section 8.7.2.

8.7.4 Interpreting weights

The question of which signals in the training sets are used by the networks may be answered by looking at the weights obtained by the training process. Small weights may indicate the unimportance of a certain input. In general, weights near zero do not necessarily mean that the input is unimportant. Several small weights for that input, each leading to a different hidden neuron, can add up to a significant effect. Also, a small weight may mean that the input affects a hidden neuron only slightly. But that hidden

neuron may be connected to an output with a large weight, causing the input to have a significant effect on the output [Masters, 1993]. In spite of these concerns the weights are relatively easy to interpret in the networks investigated here.

A way of clearly visualising weights (neuron connection strengths) is the so-called Hinton diagram. Hinton diagrams are named after the neural network pioneer Geoffrey Hinton, who first popularised this method of displaying weights. There are several variations, but they all have in common the attribute that individual squares are used to represent weights. The intensity of colour of each square is proportional to the strength of the connection weight. Here a bright red is used for large negative values and a bright green is used for positive values. Intermediate numbers have a lighter colour and the value zero is represented by white. For a better orientation the numbers of the units are printed around the display. In this numbering the units under the diagram represent source units, while numbers to the right represent target units. Two pairs of source and target units are present in the investigated networks using one hidden layer. These are input & hidden layers as well as hidden & output layers. The units are numbered consecutively from input to output, i.e. in the 60:20:1 networks 81 units can be found and the hidden units are numbered from 61 to 80.

Figure 8.26 gives the Hinton diagram for the network using the fault-free simulation as reference input. It presents the weights from the net trained to identify friction faults. Reference data is supplied to the net at inputs 1 to 30 whereas the measured data (faulty plant) is supplied at inputs 31 to 60. The weights for the measured data are clearly stronger than the weights for the reference data. This indicates the lower importance of the reference data for this particular classification task. For comparison, the respective diagram is given for the network using the residual data as reference input (Figure 8.27). The strength of the weights are more evenly distributed. This indicates that the net can make more use of the residuals than the plain reference data. Hence, in this case training the net for more epochs might lead to improved prediction performances. A similar weight distribution was obtained for the networks trained on faults in leakage and pressure.

The network trained with fault-free simulation data led to reasonable performance. Due to the low strength of weights in this case the training without any reference data is also investigated in the following sections.

8.8 Monitoring of experimental systems without reference model

By omission of the reference model fewer network inputs are used. Assuming the same number of hidden neurons this reduces the number of weights to be evaluated during the training process. This in turn is leading to a faster training process and additionally, the trained network is faster when performing the actual monitoring process. Again the same levels of friction, leakage and pressure faults were investigated. Figure 8.28 shows a schematic of the training process. The network input vector is given by

$$\mathbf{X} = \mathbf{y}_f \tag{8.19}$$

where $\mathbf{y}_f$, the data from the faulty plant is given in equation 8.18 ($n_p = n_x = n_i = 9$). Faulty plant data can either be measured or simulated. Both approaches will be investigated in the following sections. Input data sets were again obtained by taking 700 patterns for each of the fault levels. All 3500 patterns were then linearly scaled as described above. Duty cycles 1 and 2 (Figure 8.19) were used as inputs when recording data for training and validation, respectively. The second set of validation data obtained by manually introducing faults was also available.

8.8.1 Single fault networks trained with measured data

Again three 30:15:1-networks were trained (for 1000 epochs) to monitor the respective faults. This took only about 26 minutes (on a SUN 20) leading to the final MSE errors given in Table 8.8. Training a neural network with measured data and also querying it with measured data was assumed to be straightforward. Figure 8.29 shows the trained nets enquired with the first validation data set. All fault levels are predicted correctly and in particular the quality of the predicted leakage fault has improved. A second evaluation of the trained systems is given in Figure 8.30 where the manually introduced faults are investigated. Here the results look very promising and only very little scattering of the predicted fault levels can be seen. The network performance is much better although it did not make use of any reference model.

8.8.2 Single fault networks trained with simulated data

For practical applications, it can be expected that measurements for the normal operation of a plant are available. These data can be used to optimise the fault-free model of the plant as described in Chapter 7. However, measurements on a faulty system are usually hard to obtain, particularly for faults that make it dangerous to operate the plant. Hence, the main goal of this section is to demonstrate the feasibility of using NNs trained on

simulation data to diagnose faults in experimental data. Again three 30:15:1-networks were trained for 1000 epochs. Table 8.9 gives the final MSE errors. The final errors for the nets trained on experimental data are larger than those of the nets trained on simulation data. These higher errors in the experimental data are to be expected due to the presence of additional noise and unmodelled higher dynamics. The trained nets were first queried with the (simulated) validation sets and Figure 8.31 gives the respective results. A very good agreement between actual fault level and prediction was obtained. Faults in friction are identified correctly and the pressure faults in particular are predicted very closely. Figure 8.32 shows results of the second evaluation where the network is validated on measured data. The manually introduced faults are clearly identified and even the fault levels are the same as in Figure 8.30 (identified using the network trained on measured data). It is notable that the validation data was obtained using a different duty cycle input for training and validation. This clearly demonstrates the feasibility of networks trained on simulated data to diagnose faults in experimental data.

In the latter evaluation case the predicted leakage fault sometimes gives curious outputs (Figure 8.32). These might be caused by an actual friction level that is higher than the maximum level the net has been trained for. The network trained with measured data did not lead to such behaviour. Probably the lack of simulation accuracy caused these outputs. Another explanation could be that the simulation data did not contain any noise. Artificially superimposing some noise to the simulated training data might improve the network performance.

8.8.3 Modular neural network approach

The modular neural network approach described in section 8.6.4 will be adapted to systems where only simulated training data is available. In Figure 8.33 a schematic of the altered approach is given where $\mathbf{y}_f$ contains data from a simulated faulty plant as given in equation 8.18. Again the outputs of all networks can be used as inputs to a filtering network. A 30:15:3-network was trained for this purpose. 10 consecutive outputs of the first stage networks lead to 30 input units. This time only three outputs are required to indicate faults in friction, leakage and pressure and 15 hidden units were found sufficient. The training and validation data sets were constructed from 350 samples at each fault. Overall 5250 patterns formed a complete data set. Different duty cycles (Figure 8.19) were used to obtain the training and validation data. After 1000 epochs (taking 42 minutes on a SUN 20) final MSE errors of 0.00455 and 0.00852 were obtained for the training and

validation cases, respectively. Figure 8.34 gives the three network outputs when queried with the validation data set. The correct levels for faults in friction and pressure are predicted. Intermediate faults in leakage appear as slightly scattered predictions whereas the large fault level is also identified correctly. In real systems faults in leakage may develop slowly, for example due to a deteriorating seal. The application of statistics can then help to identify all levels correctly. Even without statistics the large faults will be predicted and recognised immediately. This can be seen in Figure 8.35 where the manually introduced faults are predicted correctly. The filtering network enables the recognition of all three faults in experimental data.

8.9 Improvements of the neural network approach and real-time performance

In general, it helps the network to learn if the contributions of the variables are as additive as possible. Multiplicative relationships can be changed to additive by taking the logarithmic values of the respective entities. For example, the leakage across the actuator is inversely proportional to the pressure difference. For this case the logarithmic values may increase the prediction accuracy and/or speed up the training process considerably. Furthermore fewer hidden neurons can then be used, leading to an even faster training process. In the example circuit, the friction force is proportional to the actuator velocity. It might be advantageous to supply the network with additional velocity data if available. Another option is to calculate the velocity from measured displacement data.

As already mentioned, the robustness of neural networks can sometimes be increased by supplementing the training set with noisy samples. A similar effect might be achieved by training the nets for fewer epochs. This will lead to a slightly scattered output when evaluated but this scattering may also be able to represent noise. Assuming the modular approach then noisy outputs can be dealt with by the filtering network. Furthermore, statistical methods may be applied to conclude the correct results.

A modular NN approach to fault diagnosis was described in Rodríguez et al. [1996] where it was found that one large network was impractical to monitor huge electrical grids. In order to increase scalability and dynamic adaptability smaller sub-systems of the grid were monitored separately. This approach might be useful for very large fluid power systems where different sub-systems can be monitored by the above described modular approach.

The modular approach can be further improved by using optimised networks for each fault. Different faults are dependent on different variables, hence, it might be useful to

determine the most appropriate parameters for the network inputs as well as the most appropriate network topologies. For the example circuit a smaller demand amplitude might lead to better results due to the current signal which then will not be saturated for such long periods. Furthermore, the first stage networks may all be trained to the same MSE errors. It is anticipated that this would improve the overall performance of the filtering network.

Once a network is trained the obtained weights can be used to calculate predictions from any inputs. The employed neural network tool [SNNS, 1995] contains a function that automatically compiles an executable C source code from a trained network. For the modular approach all nets can be turned into separate programs. These can then be run in parallel on different processors. Figure 8.36 gives the complete monitoring setup in schematical form. This enables simultaneous monitoring of friction, leakage and pressure. 5 processors are needed, one for each neural net and one for the data acquisition task. Although this has not been implemented, real-time performance can be achieved. Initial tests indicate that all sub-systems can be run in real-time. The final monitoring results can then be printed to the screen in numerical or graphical form.

8.10 Closure

In this chapter condition monitoring of fluid power systems has been investigated. Several monitoring techniques were reviewed and neural networks were examined in detail. An extensive simulation study was performed indicating the feasibility of neural networks for the monitoring process. Two sets of data were supplied to the network input. These are data from simulations of the faulty and the fault-free plant, respectively. The latter input was used as a reference input. Due to the increased speed of the TLM simulation method real-time reference models were enabled. A very good agreement between predicted and actual fault levels was achieved. Different faults were identified by different networks, all using the same topology. It was also shown that neural networks can be trained to be sensitive to more than one fault. Networks for the identification of several faults need to be larger and require more training patterns. This leads to much longer training times. Additionally, this approach leads to problems with scaleability, i.e. it cannot easily be adapted to several faults. Therefore a modular approach was developed. Several networks were trained to identify individual faults. The parallel outputs of these nets were then used as inputs to another network. This additional network was able to filter out the correct faults as well as the actual fault levels.

The simulation study was adapted in order to work with experimental data obtained form a hydraulic actuator system. A considerable speed-up of the training process could be achieved by using fewer parameters as network inputs. Different reference models were investigated and small performance improvements could be achieved by using residuals as network inputs. By interpreting the strengths of the weights it was found that the network could perform the fault classification task without a reference model. That is fewer weights have to be evaluated leading to an even faster training process. Additionally, the trained network is faster when performing the actual monitoring task. Individual networks were trained with measured data only and a very good fault prediction was achieved. Manually introduced step changes could clearly be identified. Finally, diagnostic networks were trained with simulation data but queried with experimental data. Whilst the predictions showed increased scatter, the network was clearly able to successfully diagnose the faults in the experimental data. Again, separate networks were trained for each fault alone allowing quick and accurate training. However, these nets do not give meaningful outputs when queried with data from faults other than the fault for which they were specifically trained. Therefore, the modular approach was also adapted to networks trained using simulated data only. The filtering network was again able to detect the correct faults and fault levels in the experimental data.

In general, the network training showed similar trends for simulation and experimental data, but the same number of training epochs led to higher final errors in the experimental data. This is due to the unavoidable noise in the measured data. Some possible improvements to the neural network approach have been proposed and the parallel implementation of it have also been outlined. Although the networks have not been optimised real-time capability is anticipated.

TABLES FOR CHAPTER 8

Fault parameter	Fault level 1	Fault level 2	Fault level 3	Fault level 4	Fault level 5	parameter unit
Leakage	0.0	0.1	0.2	0.3	0.4	L/min/bar
Load	0	500	1000	1500	2000	N
Mass	50	100	150	200	250	kg
Pressure	100	80	60	40	20	bar
Friction	1000	8000	15000	22000	29000	N/m/s

Table 8.1 Parameter fault levels

Parameter	Minimum value	Maximum value	Units
Demand: u	-50	50	mm
Current: i	-250	250	mA
Pressure: p	-40	100	bar
Displacement: x	-50	50	mm

Table 8.2 Scaling limits

Network trained on fault in	MSE training data	MSE validation data 1	MSE validation data 2
Leakage	0.001693	0.001337	0.002184
Load	0.001246	0.001376	0.001115
Mass	0.009069	0.008272	0.007102
Pressure	0.000495	0.000412	0.001011
Friction	0.002898	0.002763	0.003004

35:15:1-networks trained for 50,000 epochs

Table 8.3 Training and validation MSE errors

Faulty parameter	Fault level 1	Fault level 2	Fault level 3	Fault level 4	Fault level 5	Unit
Leakage	0.0262	0.0306	0.0520	0.0660	0.0902	L/min/bar
Pressure	100	80	60	40	20	bar
Friction	7428.6	12160	16730	22320	27760	N/m/s

Table 8.4 Parameter fault levels (identified in section 7.4.5

Network trained on fault in	MSE training data	MSE validation data
Friction	0.00195	0.00597
Leakage	0.00350	0.01490
Pressure	0.00200	0.00560

60:20:1-networks trained for 1000 epochs; simulation as reference input

Table 8.5 Training and validation MSE errors

Parameter	Minimum value	Maximum value	Units
Current: i	-250	250	mA
Pressure: p	-30	75	bar
Displacement: x	-25	25	mm

Table 8.6 Scaling limits for residuals

Network trained on fault in	MSE training data	MSE validation data
Friction	0.00162	0.00574
Leakage	0.00335	0.02210
Pressure	0.00185	0.00469

60:20:1-networks trained for 1000 epochs; residuals as reference input

Table 8.7 Training and validation MSE errors

Network trained on measured data	MSE training data	MSE validation data
Friction	0.00287	0.00390
Leakage	0.00515	0.00777
Pressure	0.00211	0.00211

35:15:1-networks trained for 1000 epochs on measured data

Table 8.8 Training and validation MSE errors

Network trained on simulated data	MSE training data	MSE validation data
Friction	0.00190	0.00280
Leakage	0.00450	0.00640
Pressure	0.00064	0.00083

35:15:1-networks trained for 1000 epochs on simulation data

Table 8.9 Training and validation MSE errors

FIGURES FOR CHAPTER 8

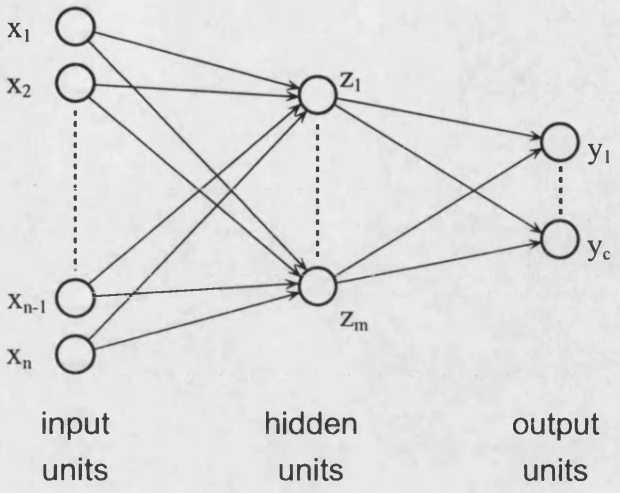


Figure 8.1 A multilayer perceptron neural network having one hidden layer

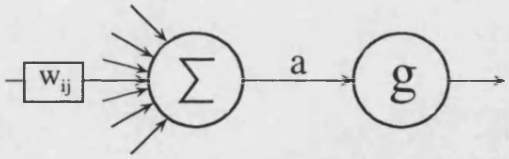


Figure 8.2 Units (neurons) of neural networks

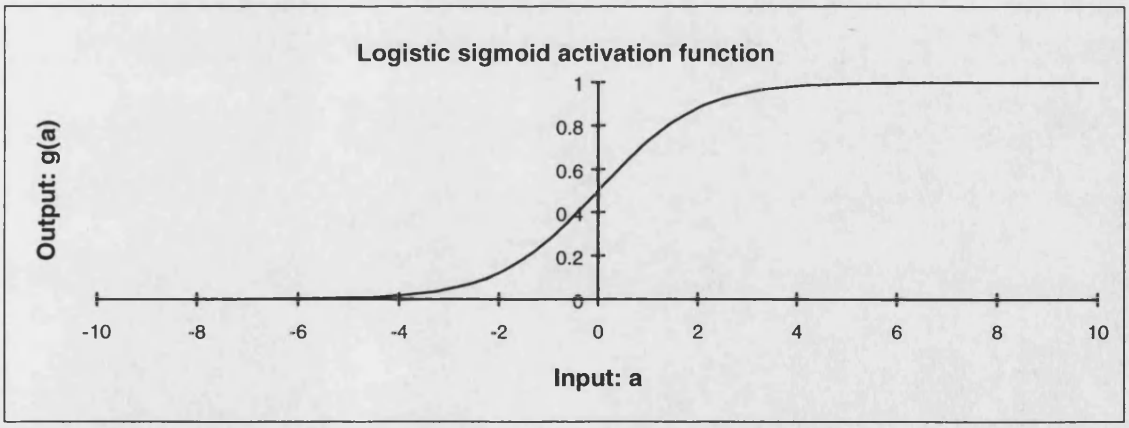


Figure 8.3 Activation function

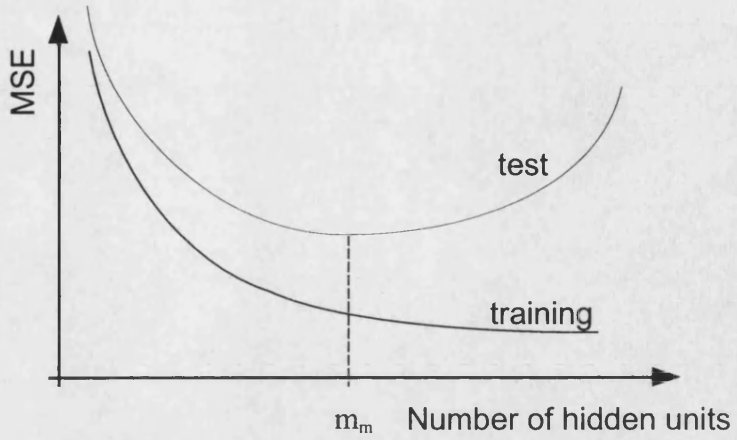


Figure 8.4 Schematic plot of the residual errors

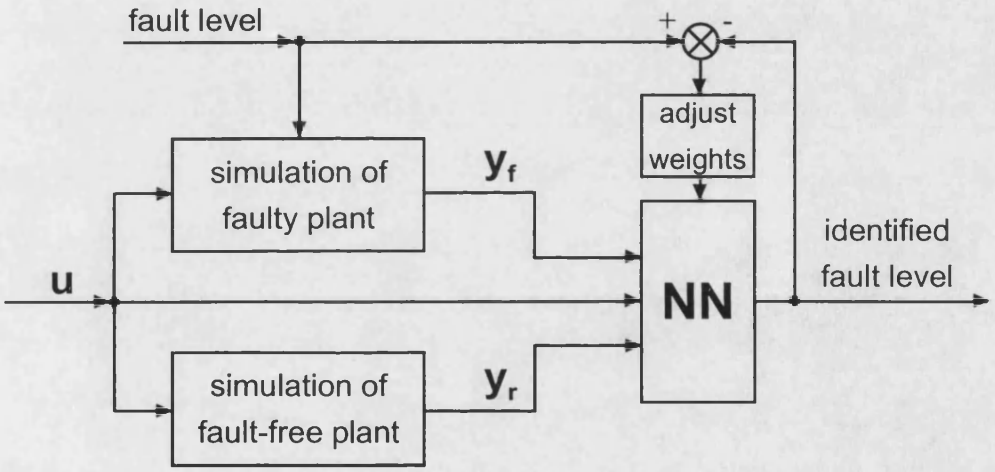


Figure 8.5 Schematic of network training process using reference model

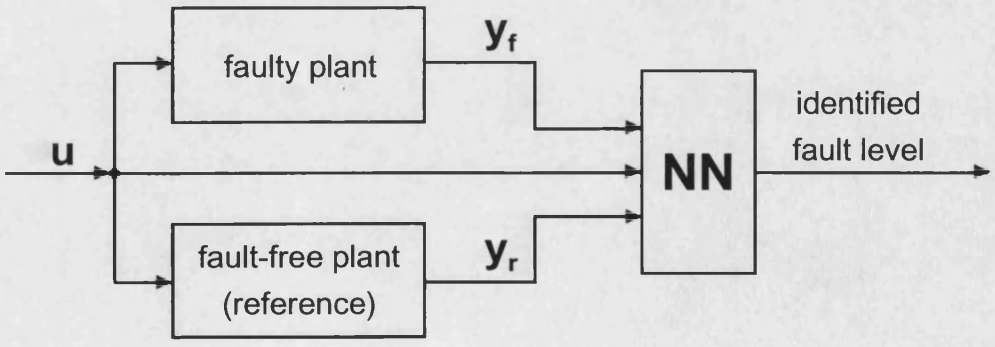
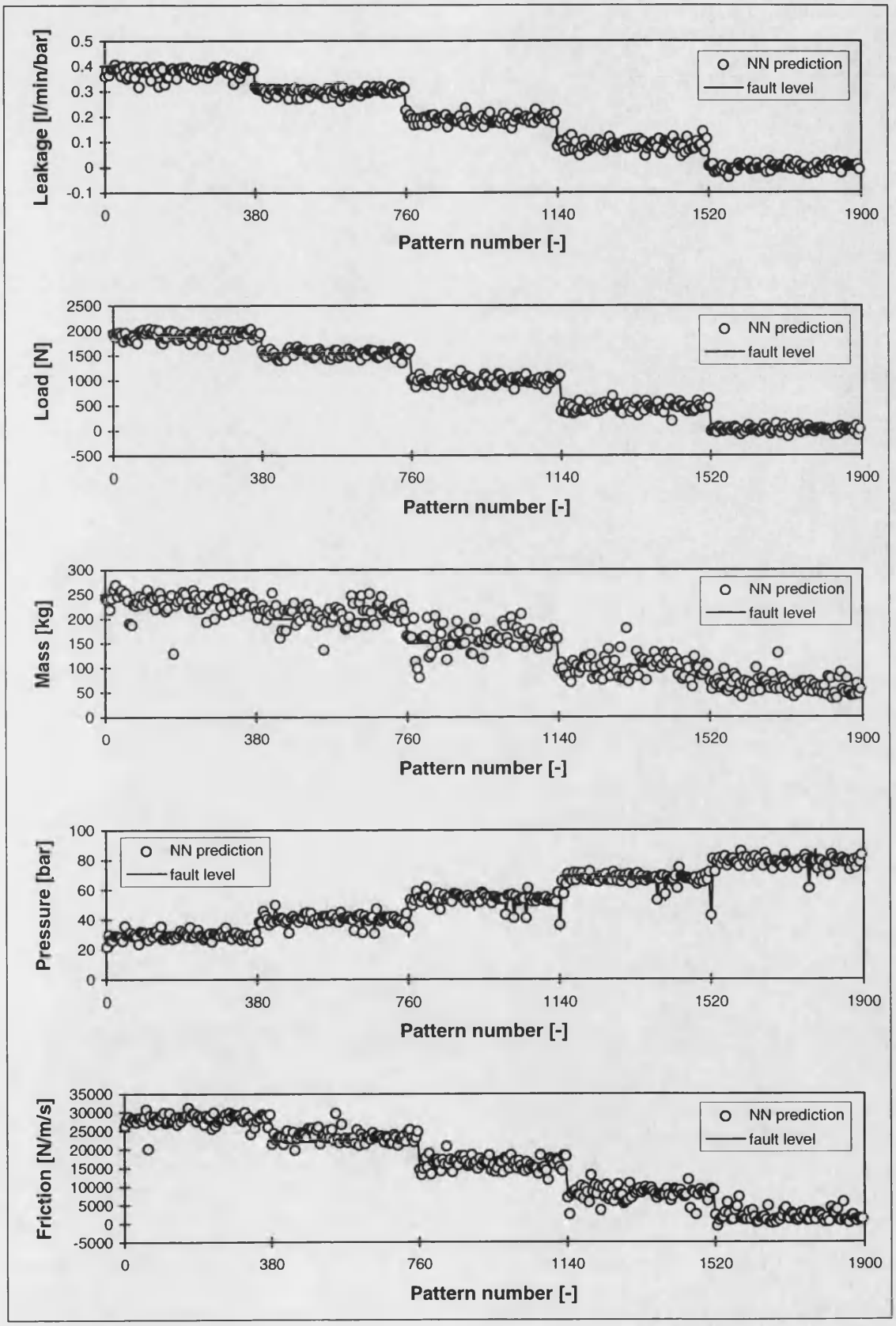
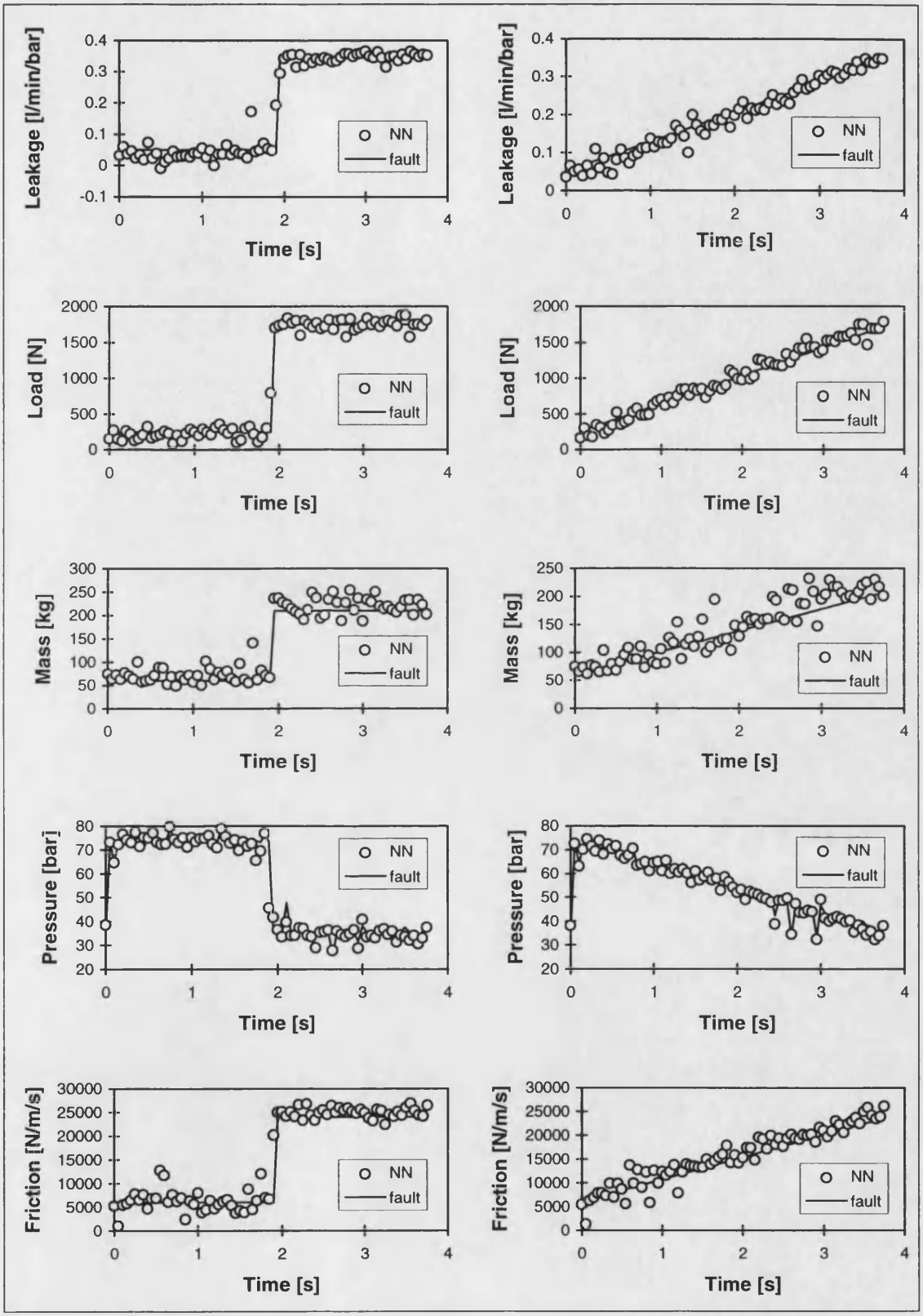


Figure 8.6 Schematic of monitoring process using reference model



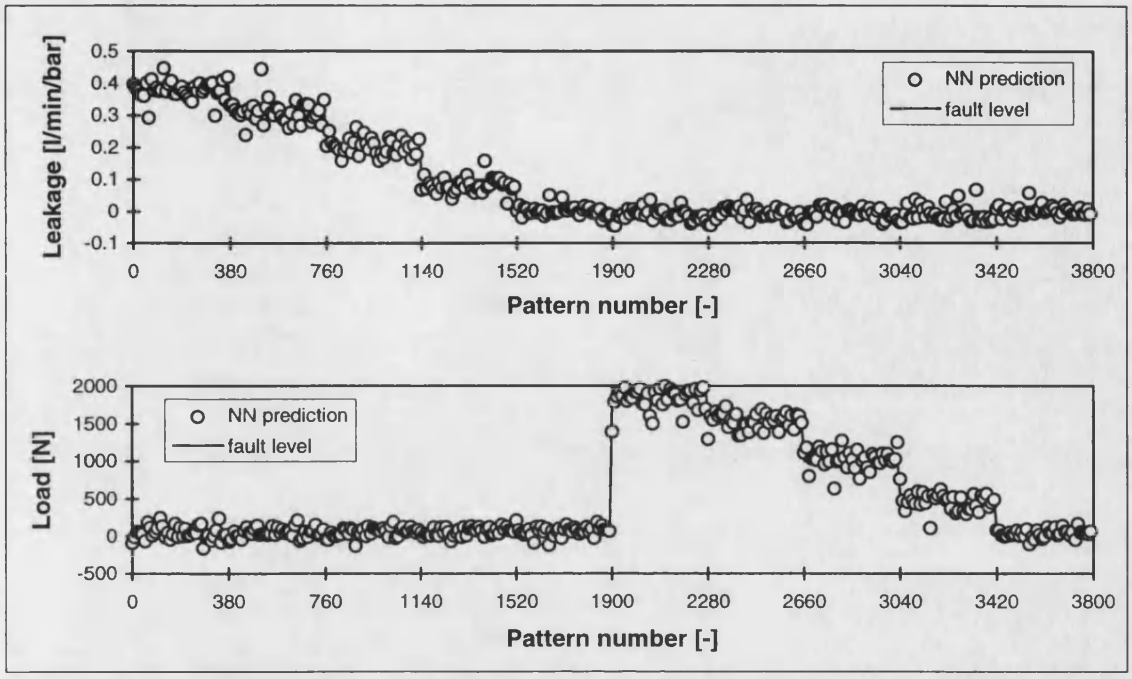
5 individual 35:15:1-networks, trained for 50,000 epochs, queried with training data

Figure 8.7 Neural network classification results



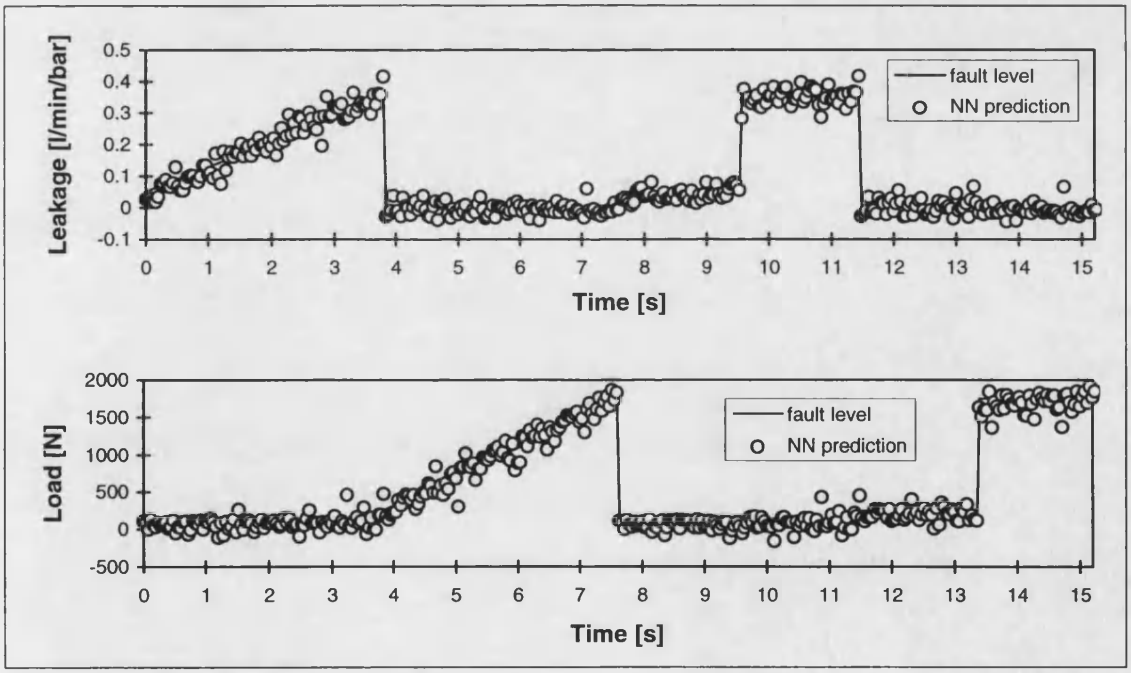
5 individual 35:15:1-networks, trained for 50,000 epochs, queried with validation data

Figure 8.8 Validation results



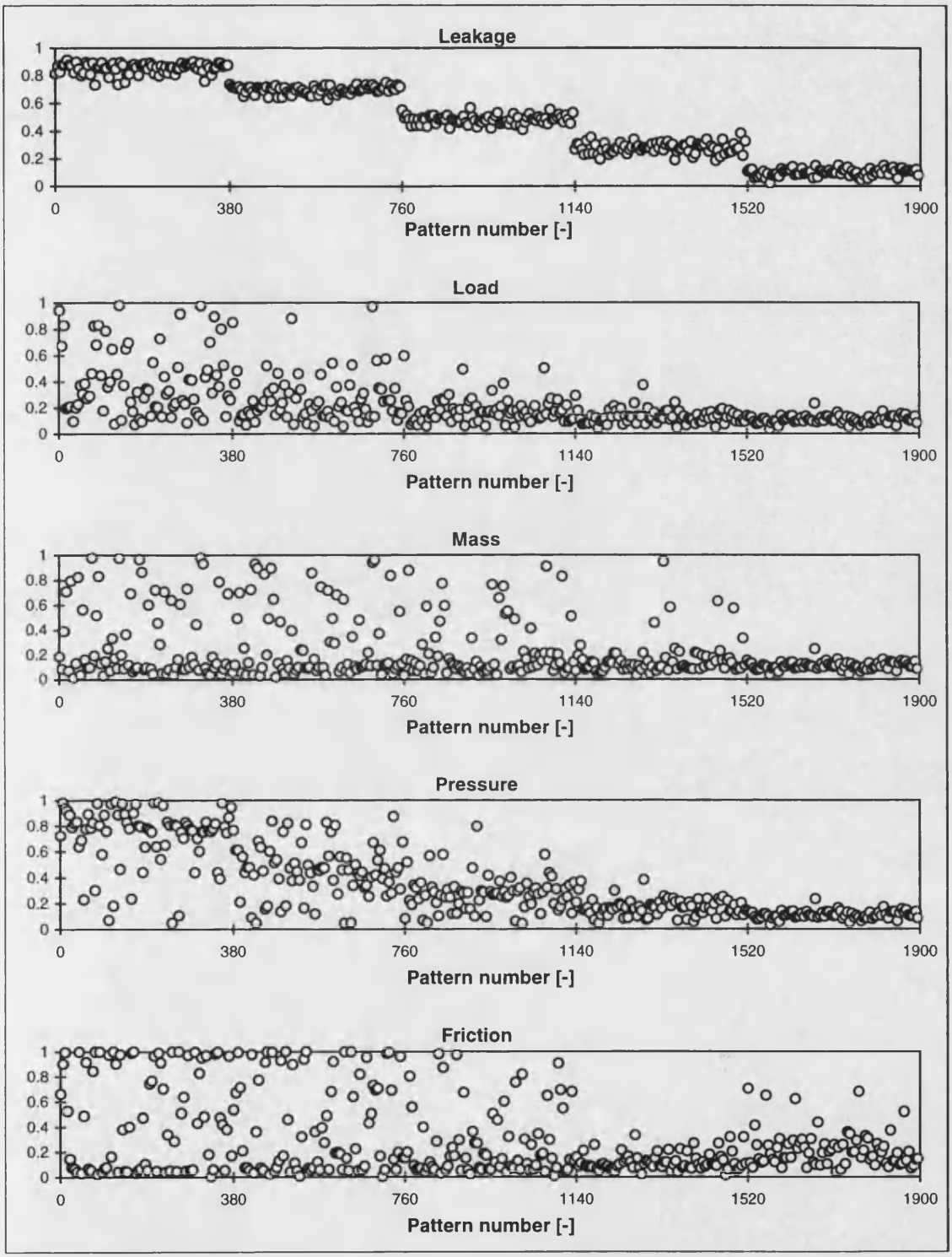
One 35:25:2-network, trained for 50,000 epochs, queried with training data

Figure 8.9 Classification results for two network outputs



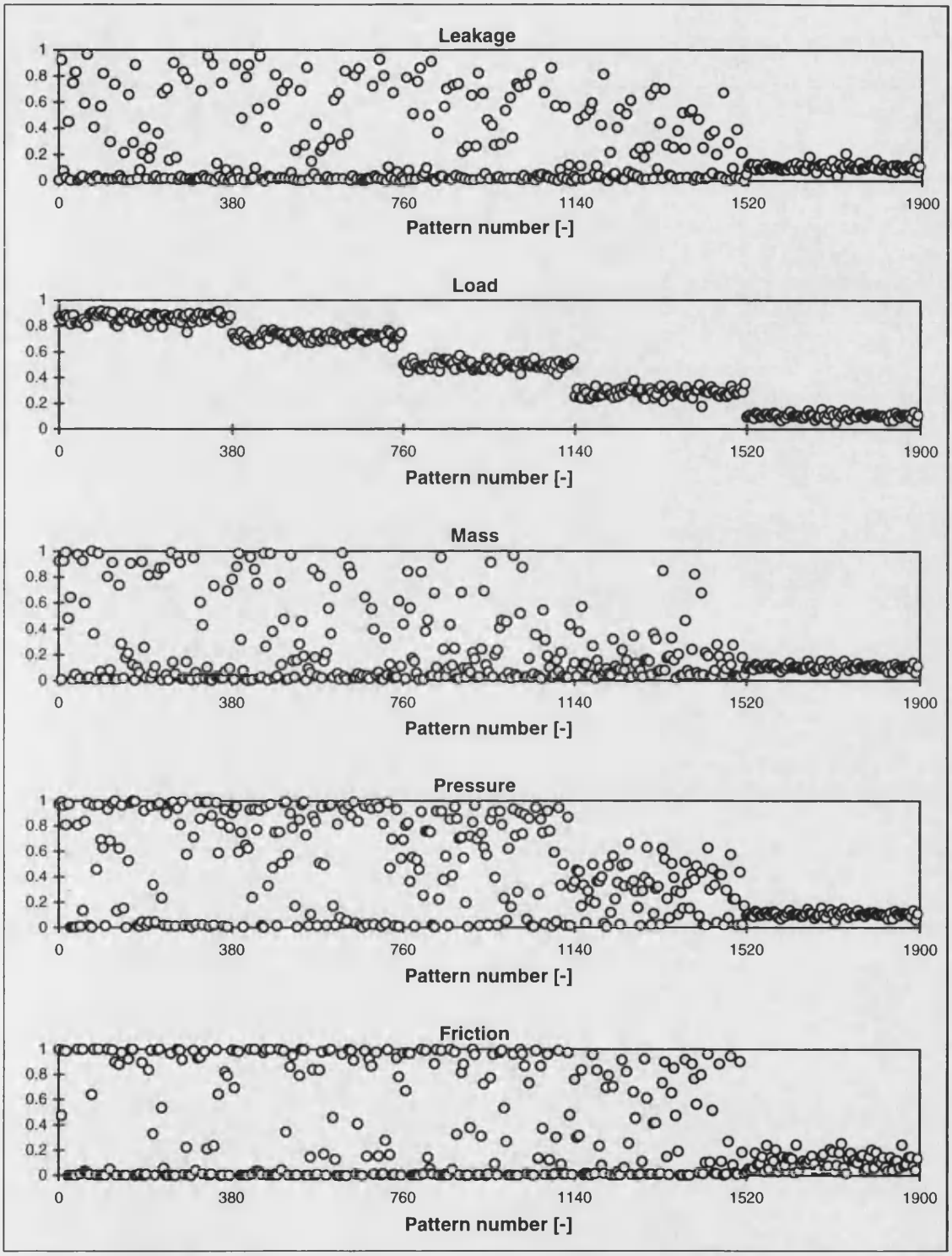
One 35:25:2-network, trained for 50,000 epochs, queried with validation data

Figure 8.10 Validation results for two network outputs



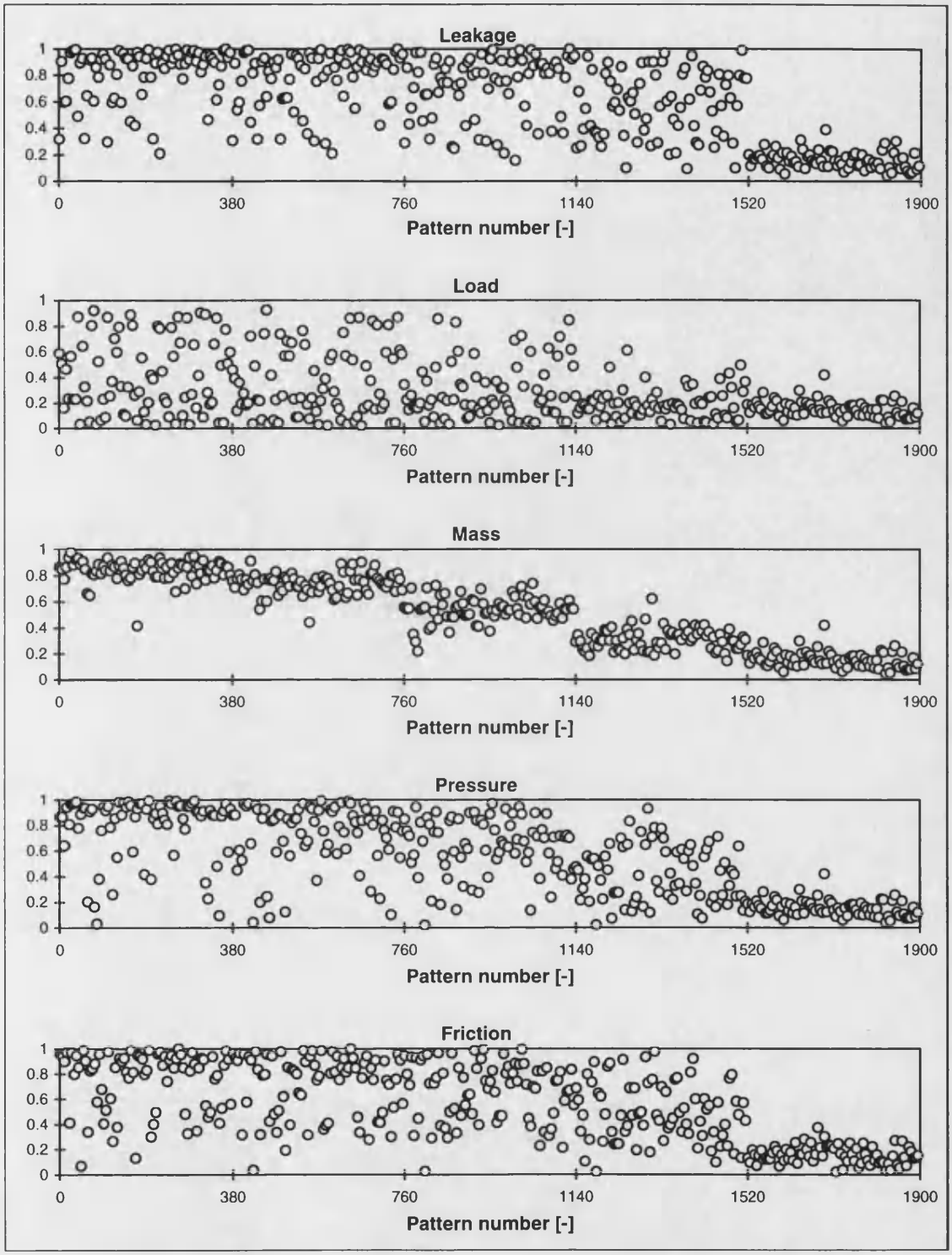
One 35:15:1-network, trained for 50,000 epochs
queried with training data from 5 different faults

Figure 8.11 Network trained to identify leakage fault



One 35:15:1-network, trained for 50,000 epochs
queried with training data from 5 different faults

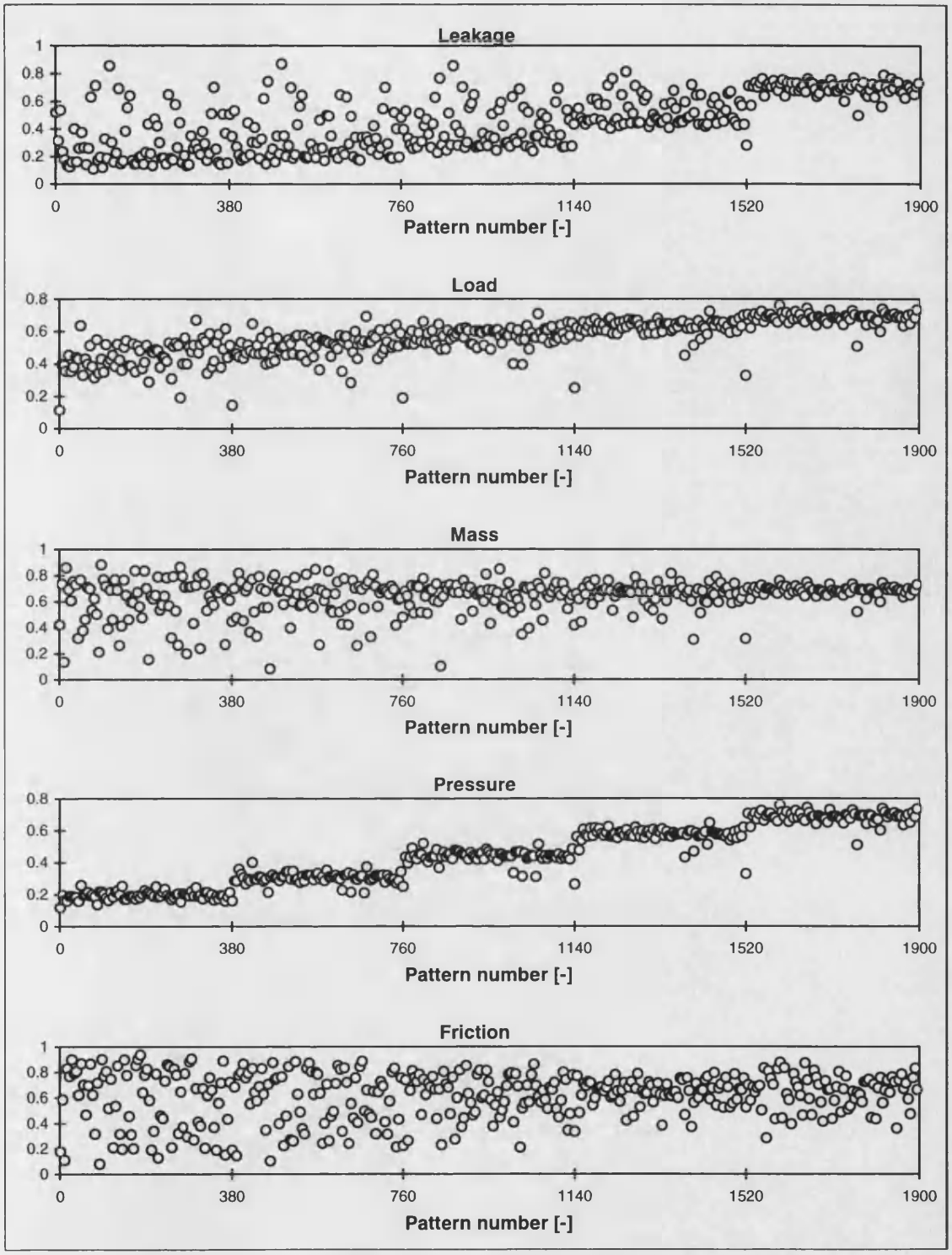
Figure 8.12 Network trained to identify load fault



One 35:15:1-network, trained for 50,000 epochs

queried with training data from 5 different faults

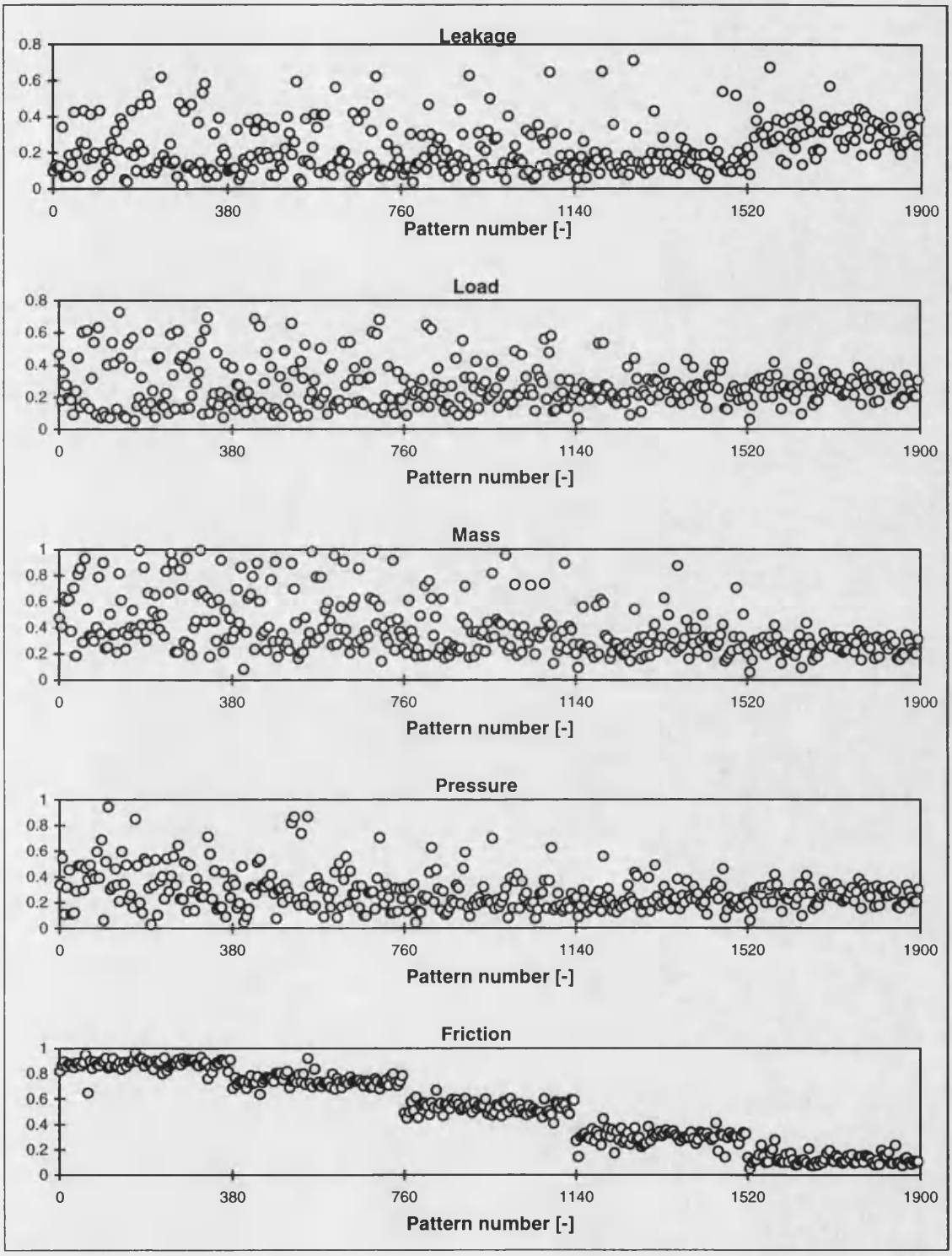
Figure 8.13 Network trained to identify mass fault



One 35:15:1-network, trained for 50,000 epochs

queried with training data from 5 different faults

Figure 8.14 Network trained to identify pressure fault



One 35:15:1-network, trained for 50,000 epochs

queried with training data from 5 different faults

Figure 8.15 Network trained to identify friction fault

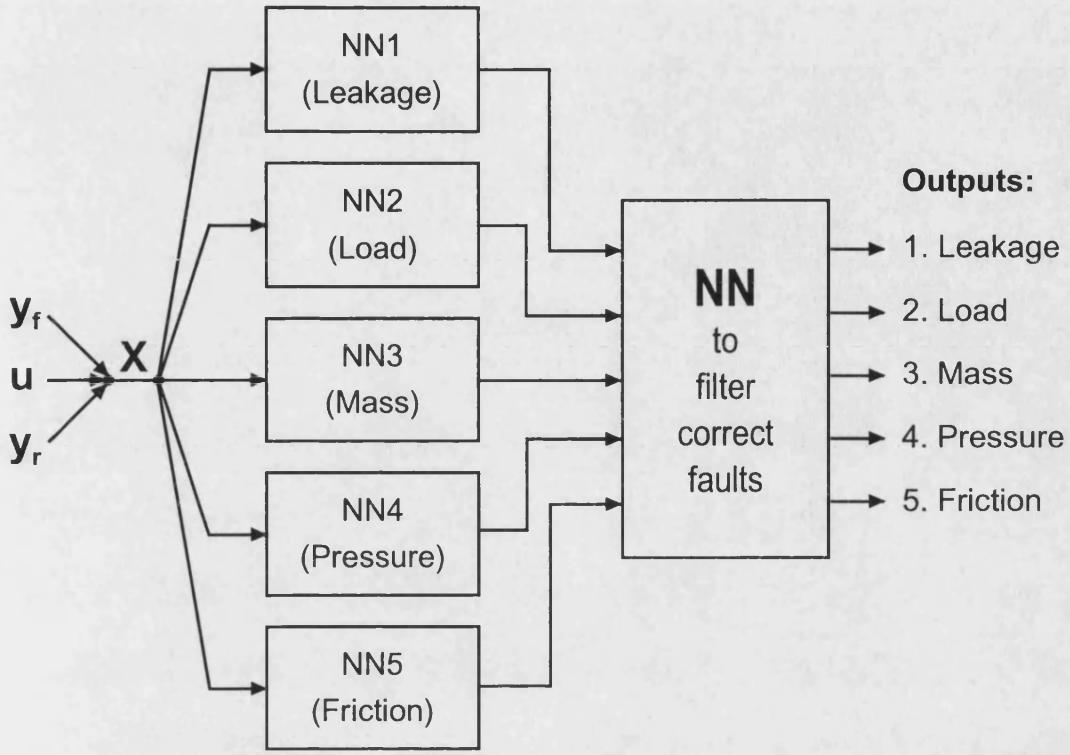
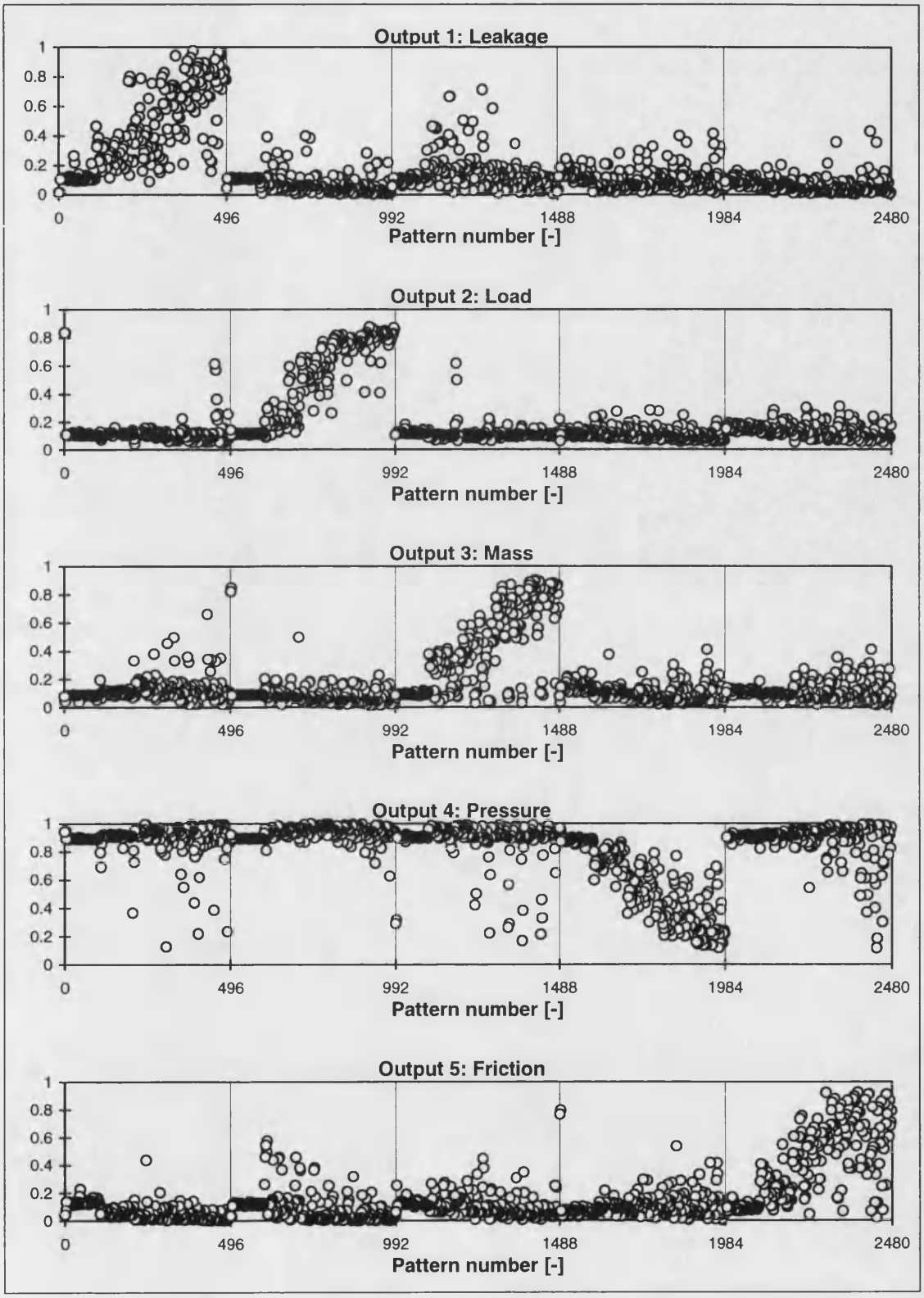


Figure 8.16 Schematic of modular neural network approach



One 25:10:5-network, trained for 10,000 epochs, queried with validation data

Figure 8.17 Five fault classification outputs from filtering network (1)

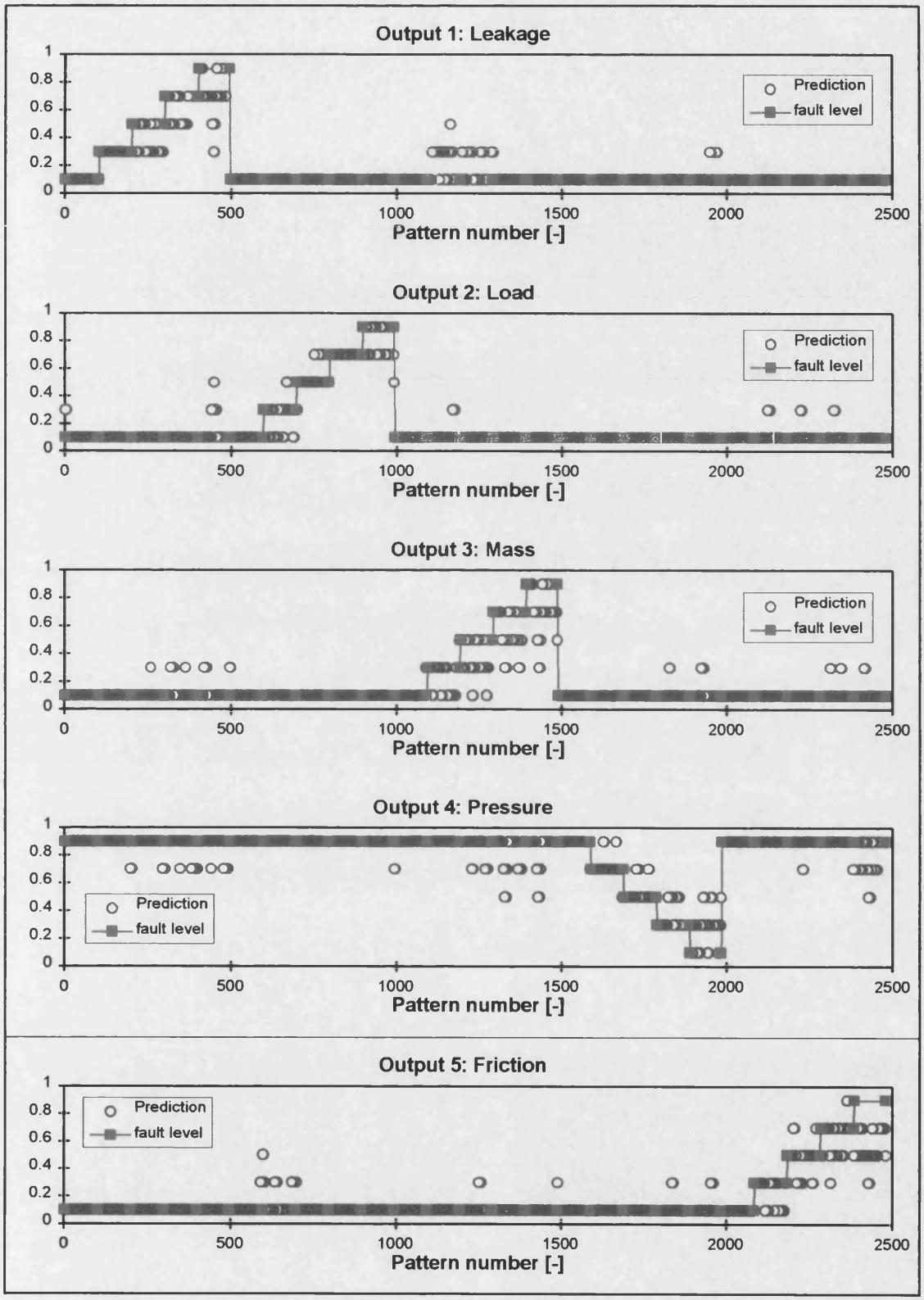


Figure 8.18 Five fault classification outputs from filtering network (2)

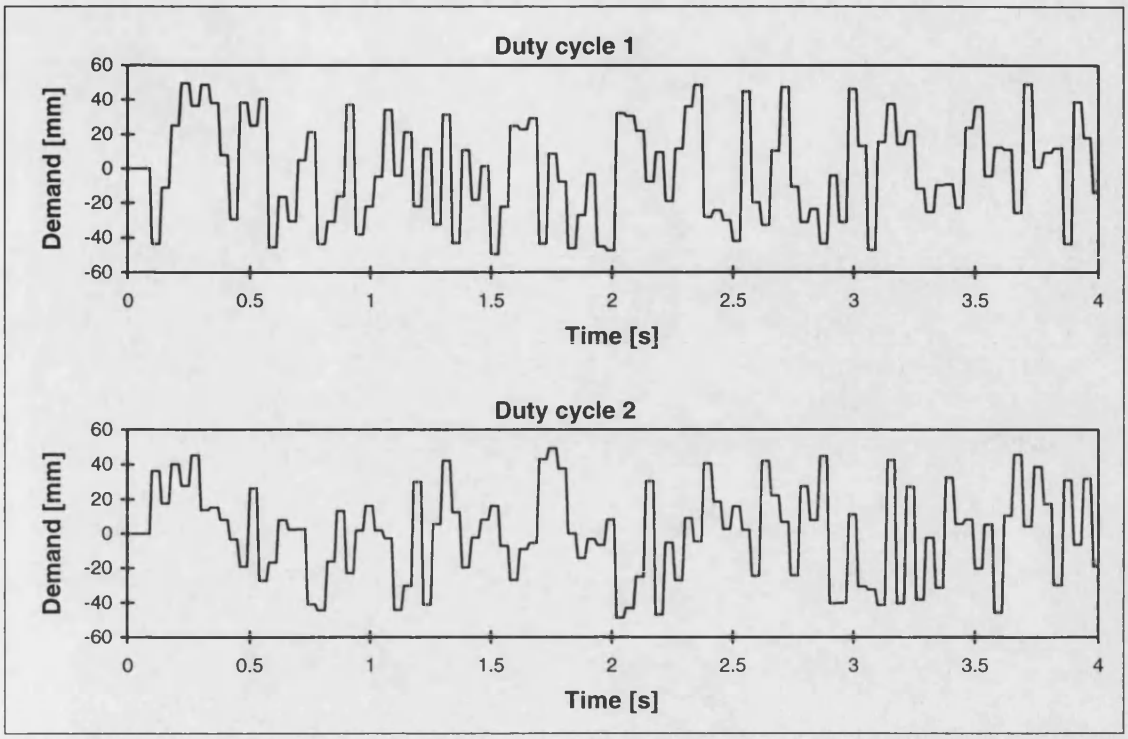


Figure 8.19 Displacement demand input signals

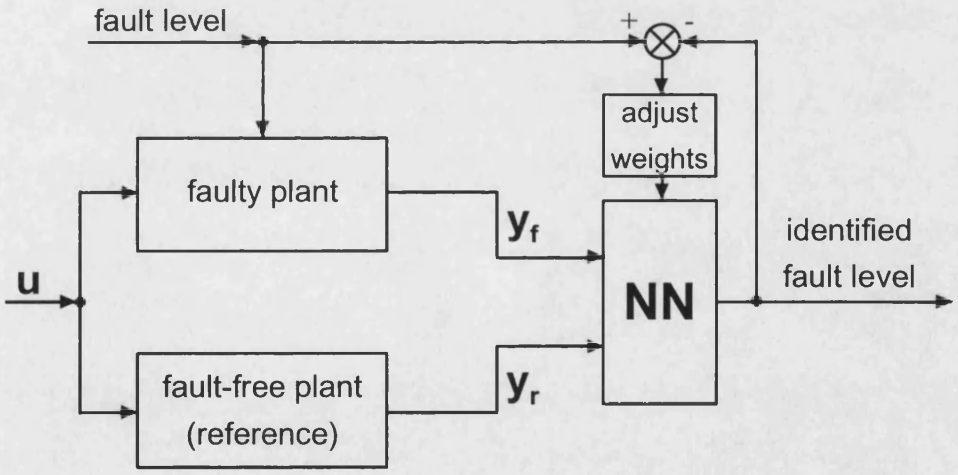
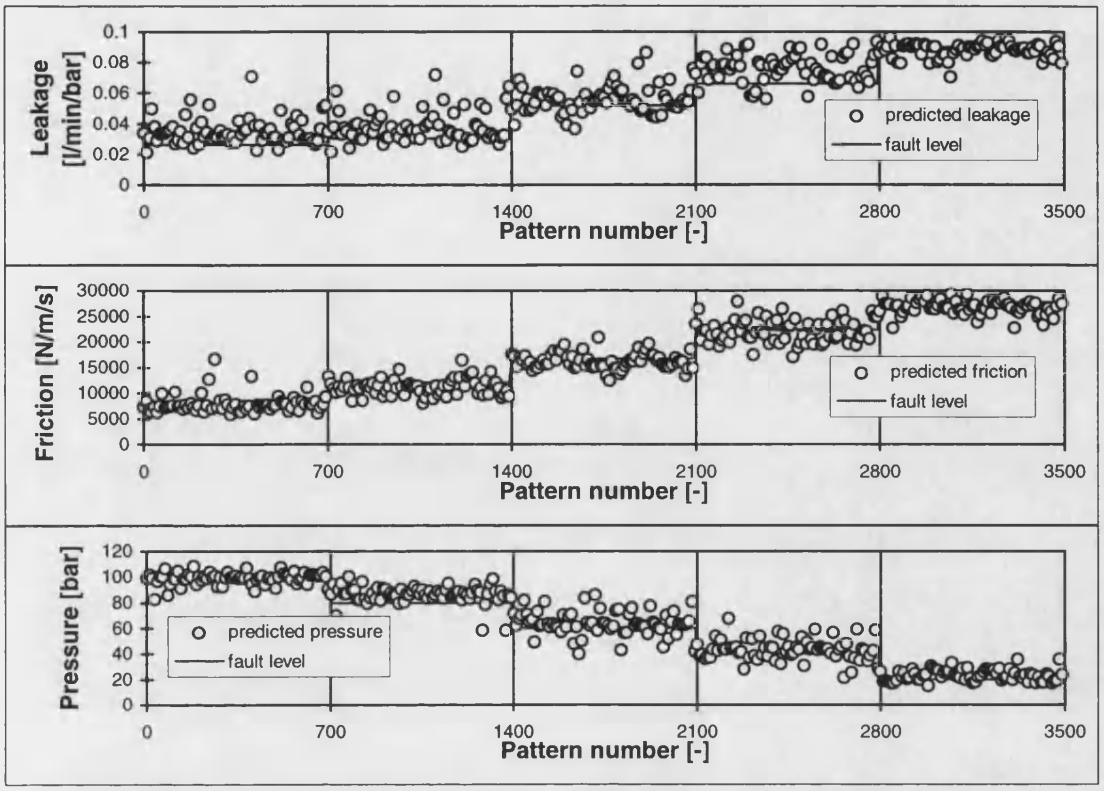
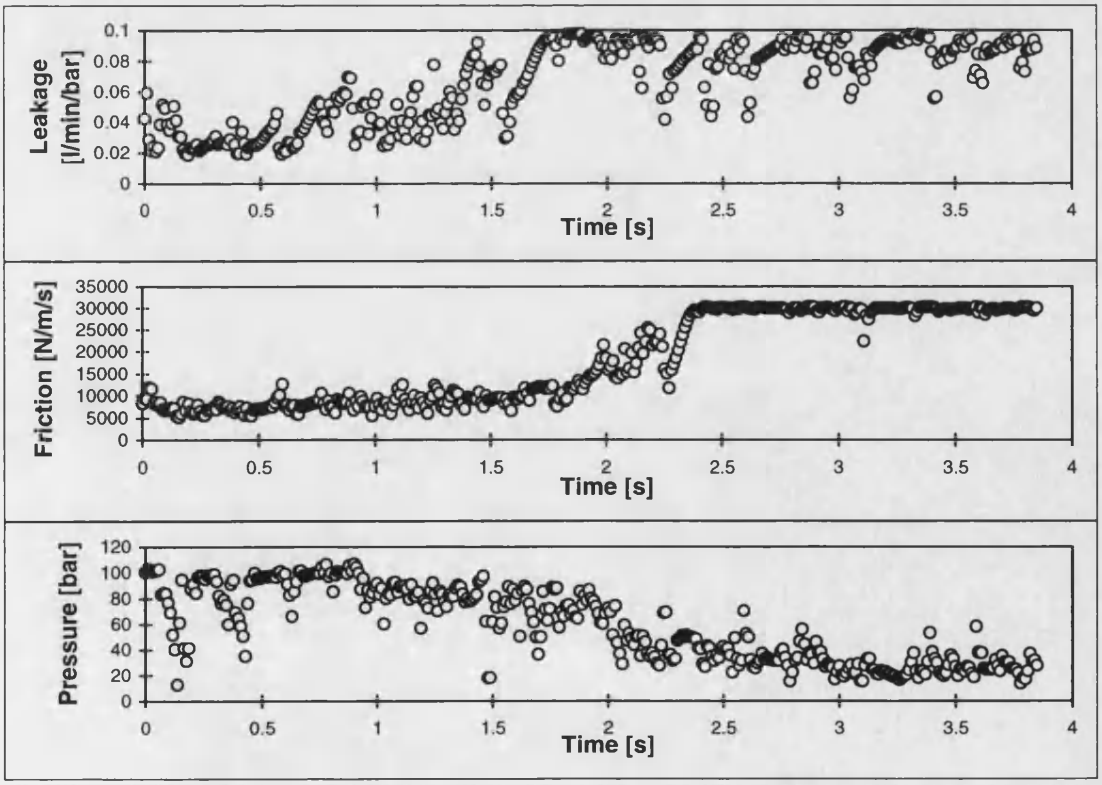


Figure 8.20 Schematic of network training process using reference model



3 individual 60:20:1-networks, trained for 1000 epochs, queried with validation data

Figure 8.21 Neural network results using reference model



3 individual 60:20:1-networks, trained for 1000 epochs, queried with real faults

Figure 8.22 Neural network validation results using reference model

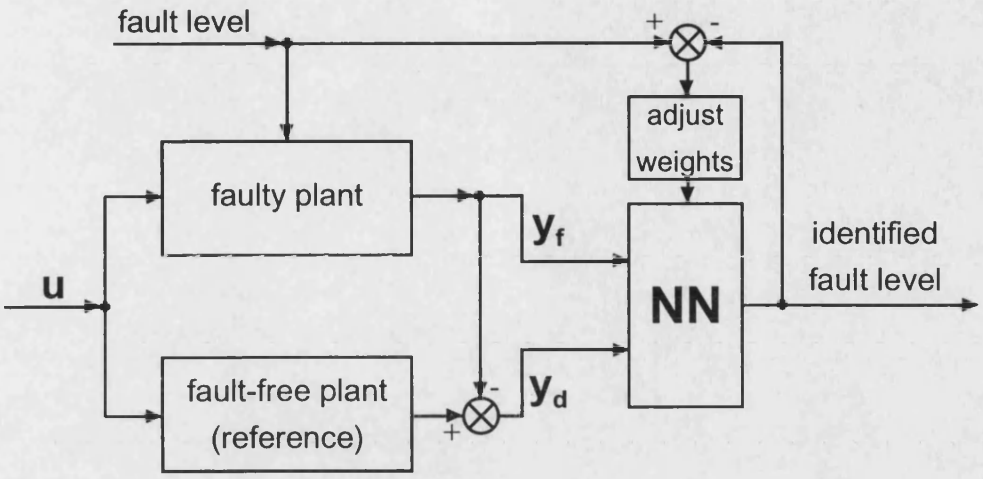
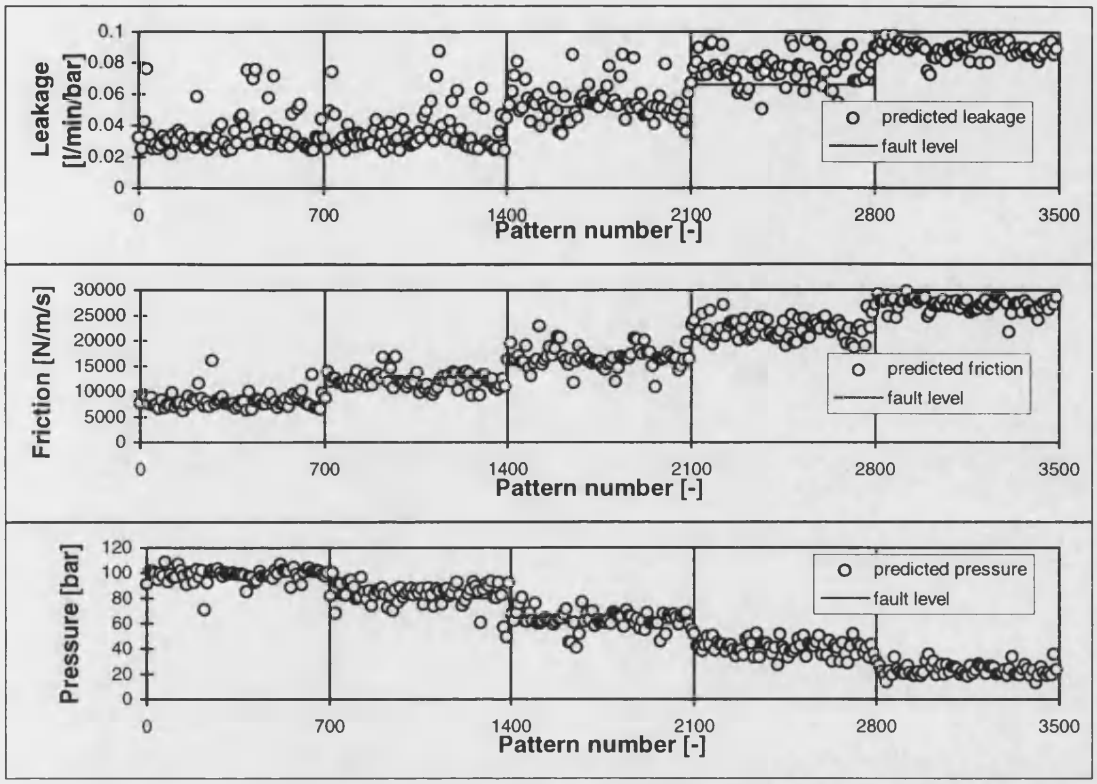
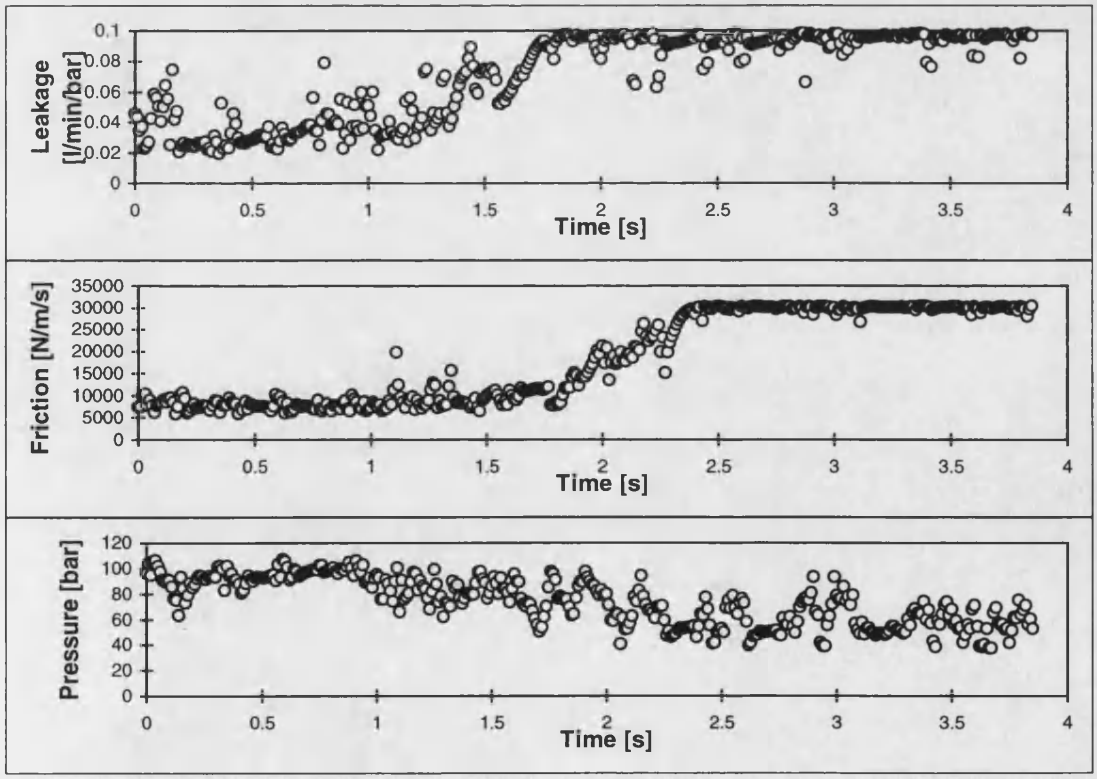


Figure 8.23 Schematic of network training using residuals



3 individual 60:20:1-networks, trained for 1000 epochs, queried with validation data

Figure 8.24 Neural network results using residuals as reference



3 individual 60:20:1-networks, trained for 1000 epochs, queried with real faults

Figure 8.25 Neural network validation results using residuals as reference

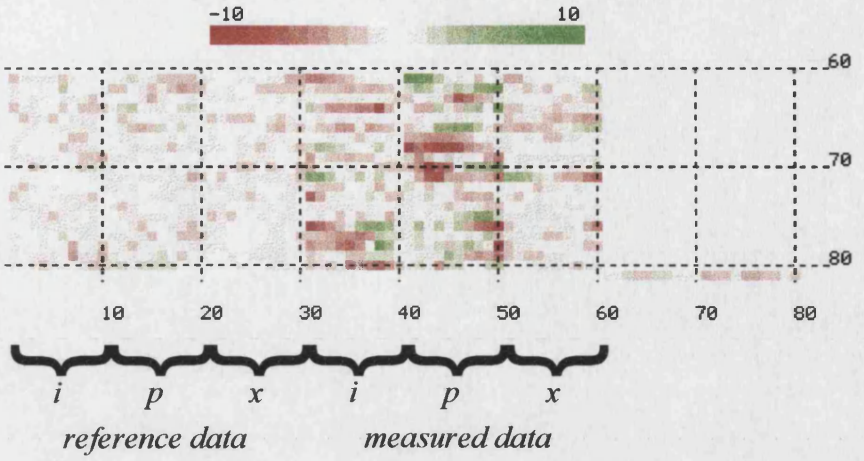


Figure 8.26 Hinton diagram for 60:20:1-network using reference data

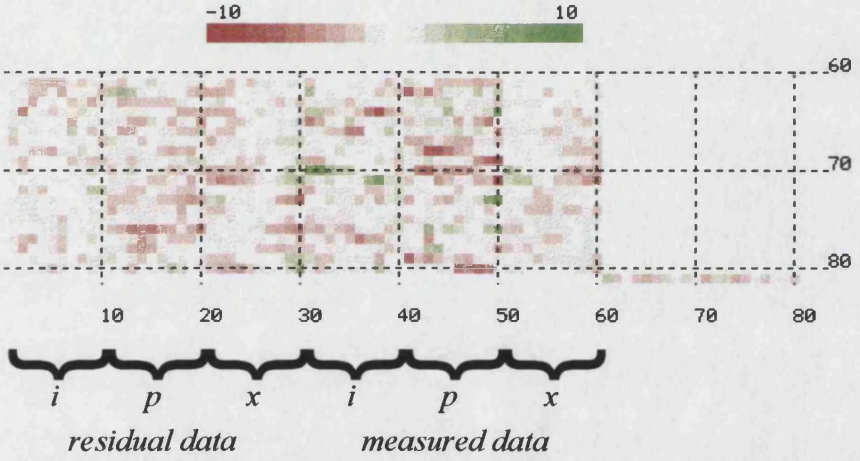


Figure 8.27 Hinton diagram for 60:20:1-network using residuals as reference data

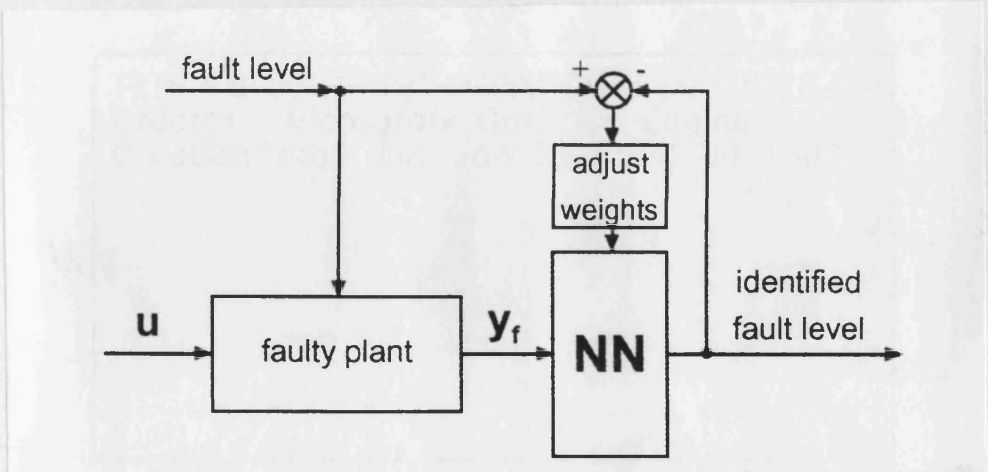
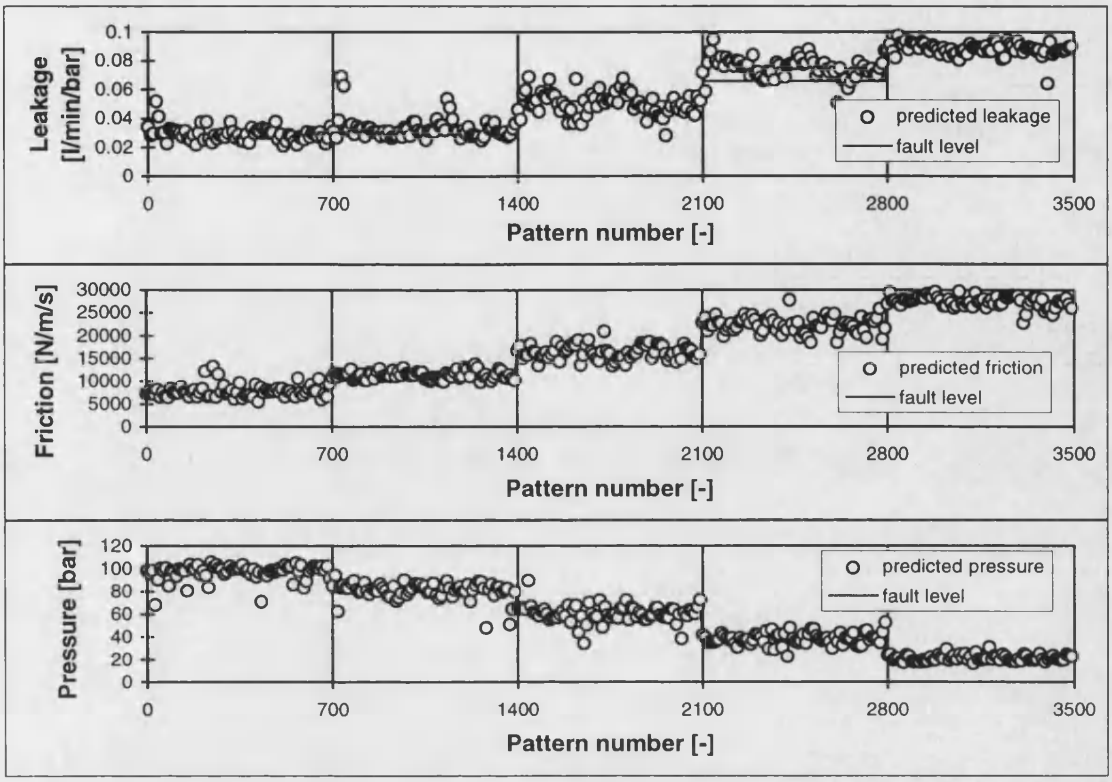
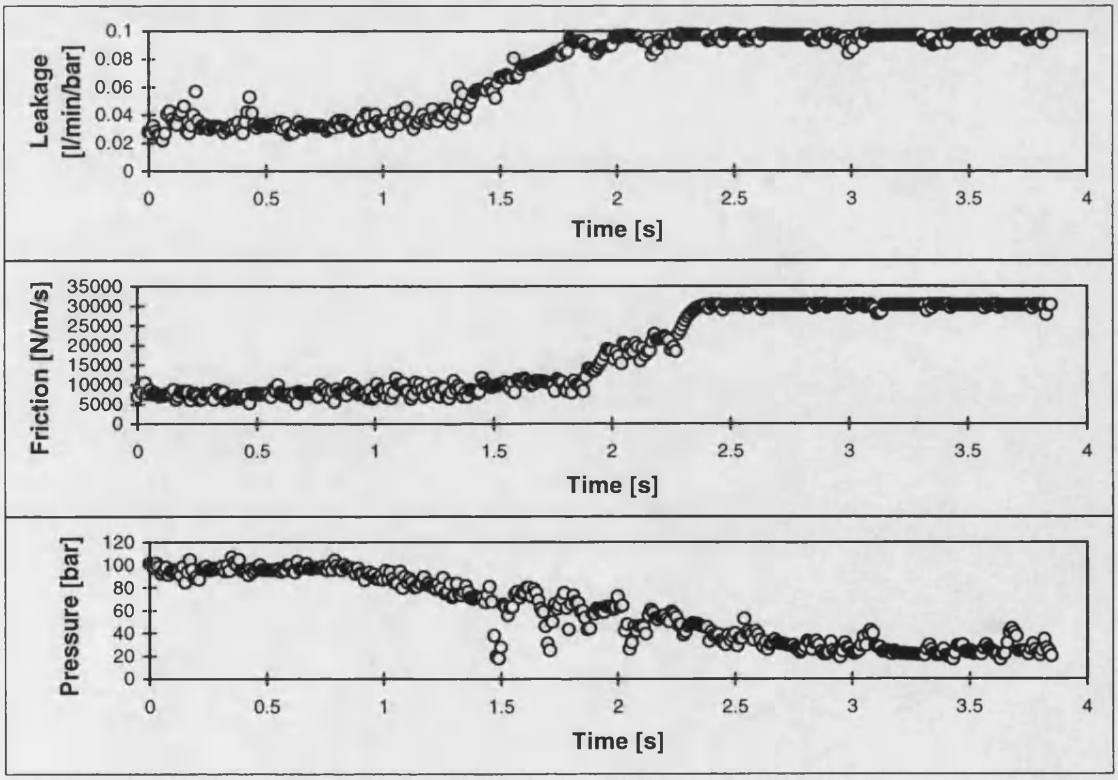


Figure 8.28 Schematic of network training using faulty plant data only



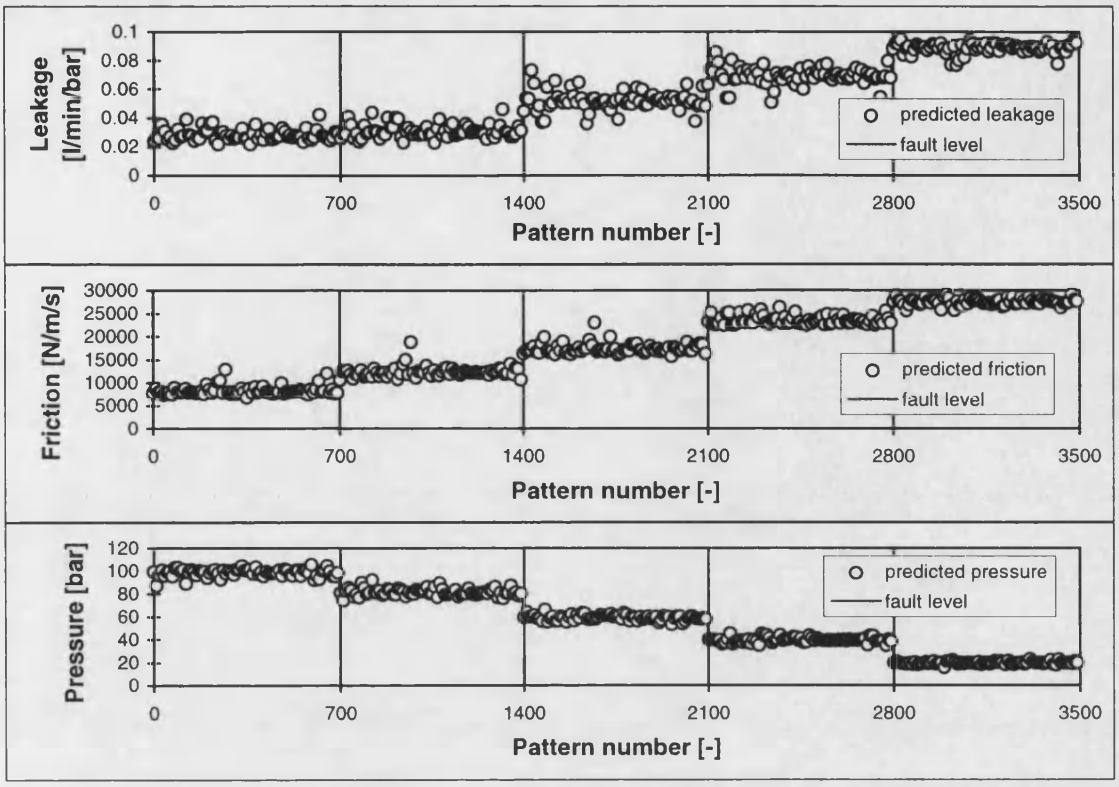
3 individual 60:20:1-networks, trained for 1000 epochs, queried with validation data

Figure 8.29 Neural network results (trained with measured data)



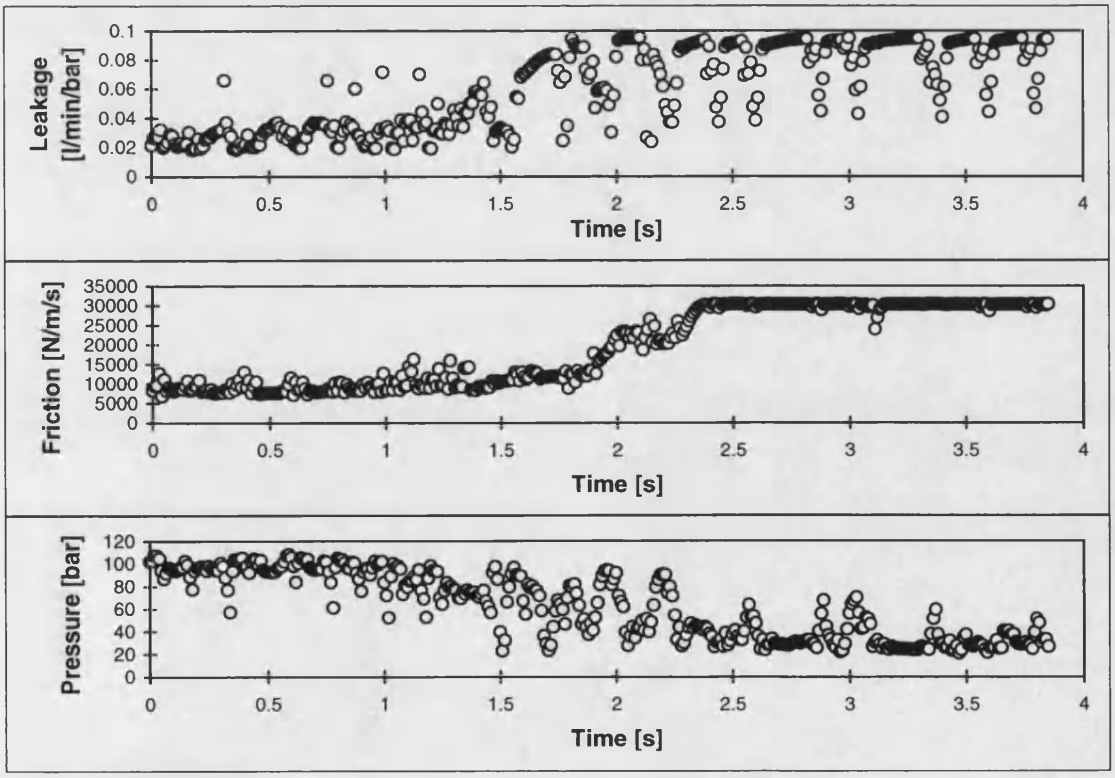
3 individual 60:20:1-networks, trained for 1000 epochs, queried with real faults

Figure 8.30 Neural network validation results (trained with measured data)



3 individual 60:20:1-networks, trained for 1000 epochs, queried with validation data

Figure 8.31 Neural network results (trained with simulation data)



3 individual 60:20:1-networks, trained for 1000 epochs, queried with real faults

Figure 8.32 Neural network validation results (trained with simulation data)

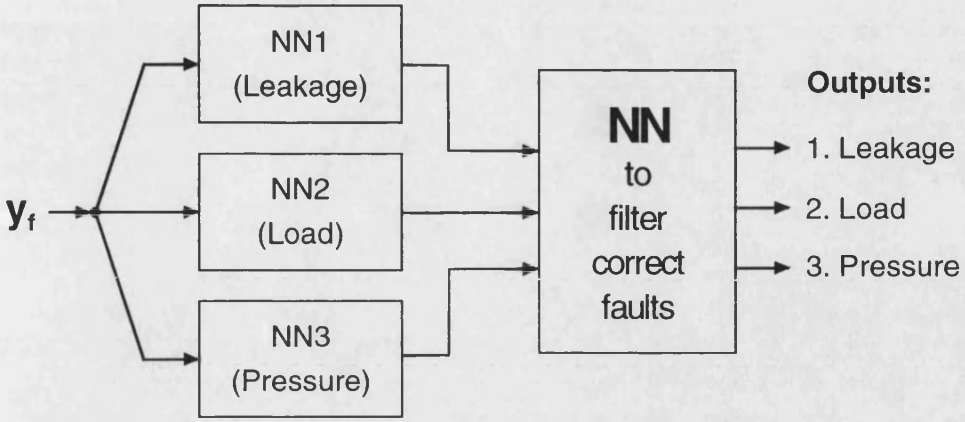
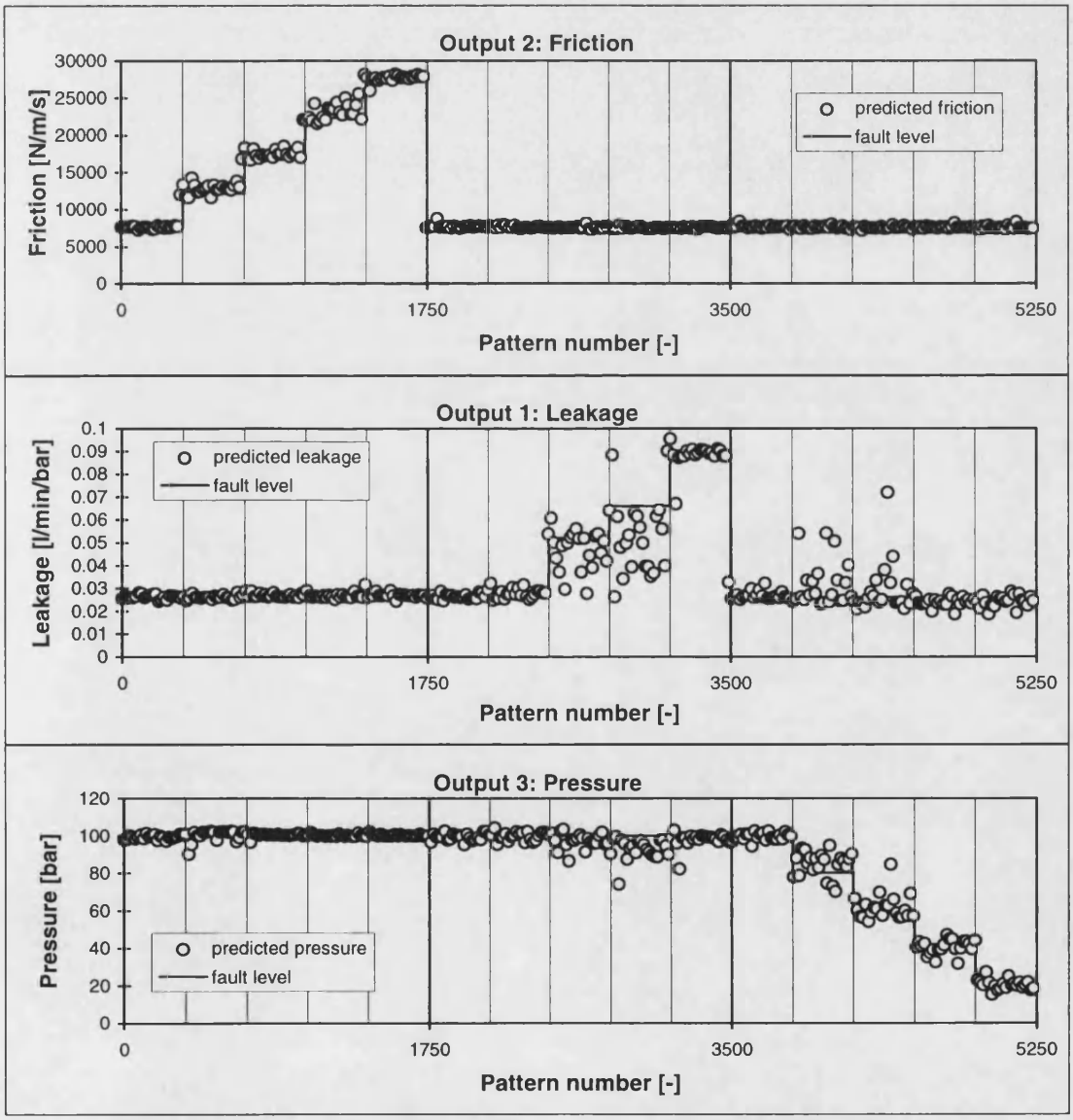
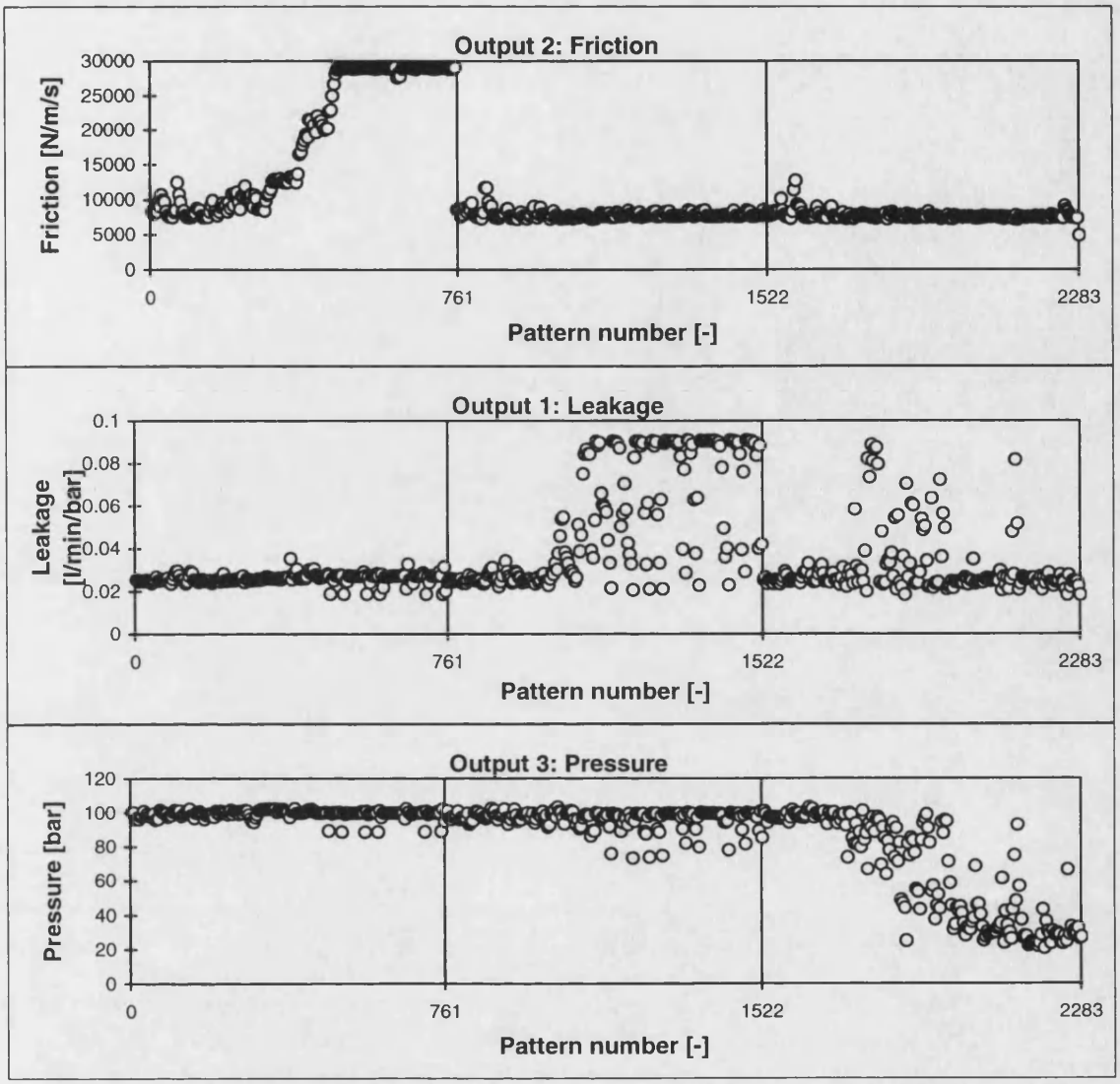


Figure 8.33 Schematic of modular neural network approach



30:15:3-network, trained for 1000 epochs, queried with validation data

Figure 8.34 Three fault classification outputs from filtering network (1)



30:15:3-network, trained for 1000 epochs, queried with real faults

Figure 8.35 Three fault classification outputs from filtering network (2)

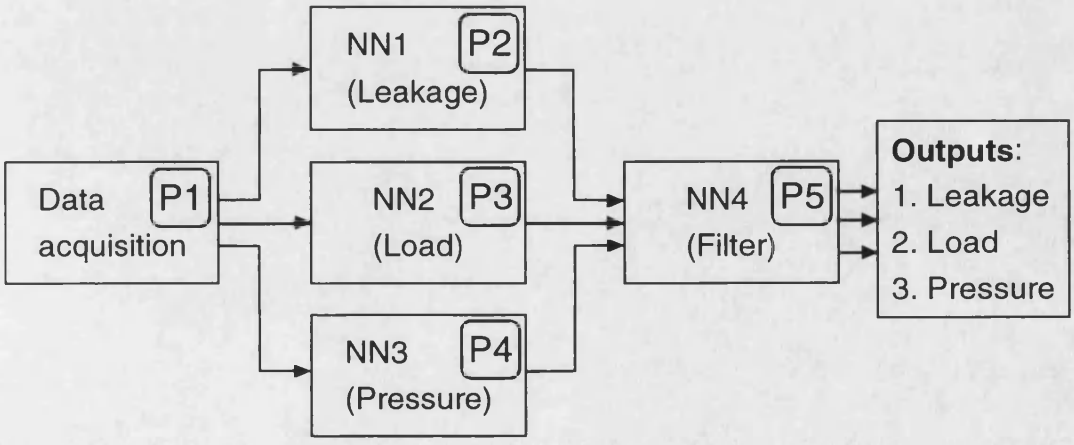


Figure 8.36 Schematic of monitoring process using parallel processing

9 Conclusions and further work

9.1 Conclusions

This thesis demonstrates the application of distributed-parameter transmission-line modelling (TLM) techniques to complex hydraulic systems simulation. Very significant improvements in execution time are achieved compared to conventional-lumped parameter modelling. For stiff numerical computations more than 100 times faster simulations are achieved. The TLM method was found to be a simple, explicit and unconditionally stable method for the modelling of wave propagation problems. If was found that one can interpret TLM as a general method of integration. This approach is used in a wide range of applications, i.e. many of the methods and features developed in this thesis may be transferred to other fields.

The widely-spread time constants in hydraulic systems has led to the development of a variable time step algorithm in order to speed up simulations. This approach was recounted and investigated in chapter 3. Variable-time step TLM seems to be suitable for numerically stiff hydraulic circuit simulations but it also leads to several problems. Unrealistic oscillations and parasitic pressure differences can appear in the simulation results as shown with a numerically stiff example circuit. For further investigations this approach is not recommended. Besides the problems found in the use of variable time steps, it is necessary to use fixed time steps for real-time simulations. There the mathematical step size must not become smaller than the computer execution time for the calculation. For fixed time step TLM, the step size has to be chosen carefully. Several results in this work show that the parasitic pressure difference can be used to estimate simulation accuracy, indicating whether the time step is sufficiently small.

The TLM method is inherently parallel and was developed for parallel platforms in order to speed up simulations. An automatic code generator for multi-processor simulations is developed in Chapter 4 using pre-compiled component models. Automated generation of the simulation program enables the system modeller to develop large and complex circuit configurations from much simpler module elements. Even the implementation of the simulation in parallel is achieved automatically. This task is normally very complicated, time consuming and prone to errors. A large fluid power circuit can be decoupled into sub-circuits which can then be simulated concurrently. Decoupling system components is straight-forward, but the apparently simple task of process-processor mapping is actually rather complicated. The aim of a partitioning scheme is to reduce the overall computation

time of a simulation, including both the computation and communication time. An automatic mapping method based on genetic algorithms was developed in Chapter 5. Results show that the method computes optimal or near optimal solutions to the sample problems. When several processors communicate, deadlock can occur, i.e. two processors refuse any further communication with each other. A scheme was developed that guarantees deadlock-free inter-processor communication. This scheme is already implemented in the automatic program generator using the output from the GA.

In general the GA fulfilled the expectation as it achieved mappings leading to good speed ups and efficiencies. To compensate for the influence of probabilistics the algorithm often has to be run more than once. This can lead to long runtimes and some simple heuristics are developed that reduce the large runtimes. Several results demonstrate that even with good mappings communication is still a considerable part of the load on the processors.

In order to reduce this communication overhead a new TLM interpolation-extrapolation method with less inter-processor communication was developed in Chapter 6. This approach requires a newly developed filter for the approximation of frequency dependent friction. Simulation results of realistic example circuits show good accuracy when exchanging data only every 100th time step. The new scheme leads to valuable reductions in run time on medium and coarse grained computers without compromising accuracy. It can also be used to simulate different partitions with different time steps, according to the required accuracy and dynamics of the subcircuits. Analytical analysis indicates that the system should be partitioned at long lines in order to maximise simulation accuracy.

This thesis also demonstrates the application of GA-based techniques to the parameter identification problem of complex hydraulic systems simulation. In chapter 7 such a method was developed and the performance of the GA was improved by an additional Hooke Jeeves direct search. It was found that signals representing higher derivatives like velocity and differential pressures are most useful for the design of objective functions. As the results in chapter 7 illustrate, several parameters can be identified simultaneously leading to a excellent agreement between simulation and measurement. Due to the use of pre-developed TLM component models the proposed approach leads to a very flexible method, i.e. for different systems the parameter optimisation can easily be adapted. If was found that the method automatically indicates whether certain parameters are relevant and whether the component models contain sufficient detail. Furthermore, different fault levels could be identified successfully, although the simulation accuracy deteriorates with

increasing fault magnitude. This may have some general implications on the use of simulations when predicting system behaviour. Precise estimations can only be made for a limited operating range of the plant. This is due to the approximations used during the component modelling process. An important point to emerge from this is that by optimising the parameters for a 'medium' operating point the useful range can be maximised. The identified fault levels may then be used for the condition monitoring of fluid power systems. Due to the long run time of the optimisation process (several hours for a small example circuit) there is no real-time capability of this method.

In Chapter 8 the feasibility of neural networks for the real-time monitoring process is shown. Due to the increased speed of the TLM simulation method real-time reference models are enabled. Hence, data from the faulty and the fault-free plant, respectively, can be used as network inputs. In an extensive simulation study a very good agreement between predicted and actual fault levels was achieved when training one network for each fault. The results in chapter 8 also indicate that neural networks can be trained to be sensitive to more than one fault. But these networks need to be larger and require more training patterns leading to much longer training times. Additionally, this approach cannot easily be scaled up to several faults. Therefore a modular approach was developed. The output of several networks, trained to identify individual faults, were used as inputs to another network. As the results show this additional network was able to successfully filter out the correct faults as well as the actual fault levels.

A considerable speed-up of the training process could be achieved by using simulated reference data of the fault-free plant and experimental data of the faulty plant as network inputs. It was found that redundant information slows down the network training and fewer parameters were sufficient as network inputs. By interpreting the strengths of the weights it was concluded that the network could also perform the fault classification task without a reference model. This was shown when individual networks were trained with measured data only and a very good fault prediction was achieved. Diagnostic networks were also trained with simulation data only but queried with experimental data. Whilst the predictions showed increased scatter, the network was clearly able to successfully diagnose the faults in the experimental data. The accuracy of the simulation was only sufficient due to the GA-based identification process where the fault free simulation was optimised to match measured data. Separate networks can be trained for each fault allowing quick and accurate training. It was found that these nets do not give meaningful outputs when queried with data from faults other than the fault for which they were specifically trained.

Statistical methods to identify the correct faults were dismissed due to the misleading output patterns. In chapter 8 it is shown that the modular approach also works with networks trained using simulated data only. A filtering network was again able to detect the correct faults and fault levels in the experimental data. Another advantage of the modular approach is its flexibility. If a new fault needs to be included in the monitoring process only two small networks need to be trained (one for the particular fault and a filtering network). This is found to be much faster than training one very large network. The modular approach also enables parallel processing of each network and the data acquisition task. Hence, a parallel implementation of the schemes was also outlined where the different parts (data acquisition, fault identification networks, filter networks, output of results) are placed in a pipeline distribution. This means every sample step data is passed from one part to the next leading to a new output indicating the condition of the plant. Although the networks were not optimised real-time capability is anticipated.

9.2 Recommendations for further work

Throughout the progress of this research a number of areas for further investigation and development were found. In the short-term this could include an increase in the diversity of component models (although new models are mainly developed as the need arises). For this research project the transputer was the only affordable parallel device on the market offering a high communication bandwidth. Current technological developments may lead to alternatives that may be more disposed to efficient operation, given the often fine-grained nature of TLM simulations. A standard portable message-passing library definition like MPI may then be used to implement the TLM method onto the new platform.

Presently, the creation of TLM simulation code is based on a manually written link file. This can be a time consuming task and it is prone to errors. Ideally, a graphical interface would automatically carry out the linking procedure and additionally provide for a parameter input. This then enables changes of circuits as well as parameters in a more convenient way.

As the results in this thesis show, the GA-based partitioning scheme requires a significant amount of time to produce good results. Although it is thought that the continuous improvement in computer hardware will play the largest part in reducing run times, it would be beneficial to further investigate heuristics enabling faster and/or better mappings. For very large hydraulic circuits the inherently parallel GA might be partitioned onto several processors.

Parallelising of the GA-based identification method can also reduce calculation time or, assuming the same runtime, can increase parameter accuracy. If this does not lead to good results in reasonable time other newly developed optimisation techniques may be investigated (for example tabu search). The identification process requires some representative plant data measurements. These are dependent on the particular duty cycle and the actual time interval considered. Further research is required to optimise these entities, i.e. leading to the specification of inputs that enable the measurement for all plant states in the shortest possible time. For some systems the frequency transform of the measured signal may be useful for the identification process in order to increase the achieved accuracy.

The work in this thesis also indicates that the approach of training NNs for the monitoring process of fluid power systems is viable. If this investigation is to be continued, it is considered that research into several areas would be worthwhile. For individual faults it will be useful to know how many transducers are necessary and where they should be placed in order to obtain sufficient information for the fault classification task. The importance of signals required by the training process may be investigated by looking at the strength of the weights obtained. Each network identifying individual faults can then be optimised in terms of the number of inputs as well as the number of hidden units. Logarithmic scaling may be investigated for the classification case when the fault is dependent on the input signal in a multiplicative manner. These optimisations can lead to much faster training and faster application of the trained networks if required for on-line condition monitoring.

The issue of system excitation is crucial to successful fault diagnosis of dynamic systems. For weakly excited systems, diagnosis is still possible, but the problem is numerically ill-conditioned and will lead to unacceptably long network training times. Therefore, it may be useful to investigate the minimum excitation requirements. Furthermore, the filtering network may be enabled to indicate insufficient excitation by an output stating 'no identification possible'. Statistical methods may be investigated for this task and also to clarify the filtering network output. So far only simple feed forward networks were investigated. Different networks like recurrent or radial basis function networks may lead to superior performance.

REFERENCES

Ackley D H, 1987; "A connectionist machine for genetic hillclimbing"; Kluwer Academic Pub., Boston, referred to in Muntean T & Talbi E-G, 1991

Aldrich C & van Deventer J S J, 1993; "The use of neural nets to detect systematic errors in process systems"; International Journal of Mineral Processing, Vol. 39, pp. 173-197

Backé W & Langen H J, 1983a; "Einsatz der Körperschallmeßmethode zur Schadensfrüherkennung in der Hydraulik, Teil 1: Grundlagen"; (Use of vibration measurement for early damage diagnosis in hydraulics, part 1: basics); O+P, Ölhydraulik und Pneumatik, 27, Nr. 10, pp. 675-682

Backé W & Langen H J, 1983b; "Einsatz der Körperschallmeßmethode zur Schadensfrüherkennung in der Hydraulik, Teil 2: Bauteilüberwachung"; (Use of vibration measurement for early damage diagnosis in hydraulics, part 2: monitoring of components); O+P, Ölhydraulik und Pneumatik, 27, Nr. 11, pp. 745-752

Backé W, 1988; "Grundlagen der Ölhydraulik"; Undruck zur Vorlesung, 7. Auflage, Institute für hydraulische und pneumatische Antriebe und Steuerungen der RWTH Aachen

Beauquier J, Choquet A, Petit A & Vidal-Naquet G, 1991; "Detection of deadlocks in an infinite family of nets"; Lecture Notes in Computer Science, No. 480, STACS 91, 8th Annual Symposium on theoretical Aspects of Computer Science, Hamburg, February, pp. 334-347

Benkhedda H & Patton R J, 1996; "Fault Diagnosis Using Quantitative and Qualitative Knowledge Integration"; UKACC International Conference on Control '96, 2-5 September

Benten M S T & Sait S M, 1994; "Genetic scheduling of task graphs"; Int. J. Electronics, Vol. 77, No. 4, pp. 401-415

Billings S A, 1983; "Structure detection and model validity tests in the identification of nonlinear systems"; IEE Proceedings, Vol. 130, Pt. D. No. 4, pp. 193-200

Billings S A, 1980; "Identification of nonlinear systems - a survey"; IEE Proc., Vol. 127, Pt. D, No. 6, pp. 272-285

Billings S A, 1985; "Introduction to Nonlinear Systems Analysis and Identification"; SERC Vacation School, "Signal Processing for Control", University of Warwick, 15-20 September, pp. 263-294

Bishop C M, 1994; "Neural networks and their applications"; "Rev. Sci. Instrum., Vol. 65, No 6, American Institute of Physics, pp. 1803-1832

Boes C, 1992; "Regleradaption an Zylinderantrieben mittels OnLine-Systemidentifikation"; 10th Aachener Fluidtechnisches Kolloquium, 17-19 März 1992, Aachen, Germany

Bokhari S H, 1987; "Assignment problems in parallel and distributed computing"; Kluwer Academic Publishers, Boston/Dordrecht/Lancaster, ISBN 0-89838-240-8

Boucher R F & Kitsios E,1986; "Simulation of Fluid Network Dynamics by Transmission-Line Modelling"; Proc. IMechE, Vol 200, No C1, pp 21-29

Branco P J & Dente J A, 1993; "A New Algorithm for On-Line Relational Identification of Nonlinear Dynamic Systems"; 2nd Int. Conf. Fuzzy Systems / Int. Conf. Neural Networks, pp. 1173-1178

Burton J D, 1992; "Parallel simulation of hydraulic systems"; MSc Transfer Report, University of Bath

Burton J D, Edge K A & Burrows C R, 1992; "Modelling Requirements for the Parallel Simulation of Hydraulic Systems"; ASME Winter Annual Meeting, Anaheim, California, USA.

Burton J D, Edge K A & Burrows C R, 1993a; "Analysis of an Electro-Hydraulic Position Control Servo-System Using Transmission-line Modelling"; 2nd JHPS International Symposium on Fluid Power, Tokyo, Japan.

Burton J D, Edge K A & Burrows C R, 1993b; "Partitioned Simulation of Hydraulic Systems Using Transmission-line Modelling"; ASME Winter Annual Meeting, New Orleans, LA, USA.

Burton J D, Edge K A & Burrows C R, 1993c; "Computational Load Balancing of Parallel Hydraulic Circuit Simulations Employing Variable Timestep Transmission-line Modelling"; 3rd Scandinavian Conference on Fluid Power, Linköping, Sweden.

Burton J D, 1994; "Parallel Simulation of Hydraulic Systems using Transmission Line Modelling (TLM)"; PhD thesis, University of Bath, England

Burton R, Harte A-M, Sargent C & Schoenau G, 1991; "An Attempt to Control a Hydraulic Circuit using an Artificial Neural Network"; 4th Bath

Chow E Y & Willsky A S, 1984; "Analytical Redundancy and the Design of Robust Failure Detection Systems". IEEE Trans. Automatic Control, Vol 29. pp 603-614

Christopoulos C, 1995; "The Transmission-Line Modeling Method TLM"; Oxford University Press, The Institute of Electrical and Electronics Engineers, Inc. New York, ISBN 0-7803-1017-9, IEEE Order Number: PC3665

Christopoulos C, 1991; "The historical development of TLM"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Christopoulos C, Herring J L & Scaramuzza RA, 1991; "Electromagnetic simulation using transmission-line modelling"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

de Cogan D & Enders P, 1991; "Microscopic effects in TLM heat flow modelling"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Collacott R A, 1977; "Mechanical Fault Diagnosis and condition monitoring"; Chapman and Hall, London, ISBN: 0 412 12930 2

Conrad F, Trostmann E & Zhang M, 1993; "Experimental identification and modelling of flow and torque losses in gerotor hydraulic motors"; 2nd JHPS, pp. 677-682

Curatelli F, 1995; "Implementation and evaluation of genetic algorithms for system partitioning"; Int. J. Electronics, Vol. 78, No. 3, pp 435-447

Da R & Lin C-F, 1995; "Failure Diagnosis System Using ARTMAP Neural Networks"; Journal of Guidance, Control, and Dynamics, Vol. 18. No. 4, pp. 696-701

Daley S & Wang H, 1993; "On the Application of Neural Networks to the Monitoring of a Simulated Hydraulic Rotary Drive System"; 6th Bath international Fluid Power Workshop, Burrows C R & Edge K A (Editors), pp. 253-264

Deiss W H, Fadden E J, Howe R M. 1994, (Paper distributed at the one day seminar in Northampton, 22th April 1994); "An Asynchronous Simulation Methodology for Improving the Performance of Training Simulators"; Applied Dynamics International, Ann Arbor, Michigan 48108

Delgado A, Kambhampati C & Warwick K, 1995; "Identification of nonlinear systems with a dynamic recurrent neural network"; Artificial Neural Networks, 26-28 June, Conference Publication No. 409, IEE, pp. 318-322

Donne M S, Tilley D G & Richards W, 1995; "The use of multi-objective parallel genetic algorithms to aid fluid power system design"; Proc Instn Mech Engs, Vol. 209, pp. 53-61

Donne M S, Tilley D G & Richards W, 1994; "Adaptive search and optimization techniques in fluid power system design"; Conference on Adaptive computing in engineering design and systems control, University of Plymouth, UK, 21-22 September, pp. 67-76

Donne M S, 1993; "Development of an optimization technique to aid fluid power system design"; PhD Thesis, University of Bath

Dransfield P & Stecki J, 1991; "Bond Graphs and Neural Nets - The Potential of each for Fluid Power Control"; 4th Bath Int. Fluid Power Workshop, pp. 174-189

D'Souza A F & Oldenburger R, 1964; "Dynamic Response of Fluid Lines"; Trans. ASME J. of Basic Eng., pp 589.

Dubard J L, Benevello O, Pompei D, Le Roux J, So P P M & Hoefer W J R, 1991; "Acceleration of TLM through signal processing and parallel computing"; Int. conf. on computation in electromagnetics, IEE, Vol. 350, Ch. 97, pp. 71-73

Efe K, 1982; "Heuristic models of task assignment scheduling in distributed systems"; IEEE Computer, June, pp. 50-56

Ellman A U, Lindberg I, Vilenuis M J, 1993; "Simulation in the Design of Hydraulic Driven Machines: New Approach and Aspects"; 3rd Scandinavian Int. Conference of Fluid Power, Vol 2, pp 29-41.

Esser J, 1992; "Diagnosesystem für Hydraulikpumpen"; (Diagnostic system for hydraulic pumps); O + P, Ölhydraulik und Pneumatik, 36, Nr. 3, pp. 176-181

Fung K K, Hui S Y R & Christopoulos, 1993; "Parallel simulation of electrical circuits using the TLM technique"; Software Appls. in Elec. Eng., CH 46, Computational Mech. Pubs., Ashurst Lodge, Southampton, pp. 21-28

Garey M R & Johnson D S, 1979; "Computers and intractability: A guide to the theory of NP-completeness"; Freeman, San Francisco, ISBN: 0716710447

Glass C A, Potts C N & Shade P, 1994; "Unrelated Parallel Machine Scheduling Using Local Search"; Math. Comput. Modelling, Vol. 20, No. 2. pp. 41-52

Goh C J & Noakes L, 1993; "Neural Networks and Identification of Systems With Unobserved States"; Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control, Vol. 115, pp. 196-203

Goldberg D E, 1989; "Genetic Algorithms in search, optimisation & machine learning"; Addison-Wesley, ISBN 0-201-15767-5

Govind G & Ramamoorthy P A, 1992; "A new online adaptive algorithm for nonlinear system identification and control"; SPIE Vol. 1706 Adaptive and Learning Systems, pp. 187-197

Gropp W, Lusk E & Skjellum A, 1994; "Using MPI - Portable Parallel Programming with the Message-Passing Interface"; MIT Press, Cambridge, ISBN: 0-262-57104-8

Heinonen M, Rinkinen J & Sassali T, 1995; "Behaviour of electro-hydraulic two-stage servo valves under different kinds of malfunctions"; 4th Scandinavian International Conference on Fluid Power, 26-29 September, Tampere, Finland, Vol. 2, pp. 696-705

Hoare C A R, 1985; "Communicating Sequential Processes"; Prentice-Hall International Series in Computer Science, Hoare C A R (Series Editor), Prentice-Hall International, UK, ISBN 0-13-153271-5

Hogan P A, Burrows C R, Edge K A, Atkinson R M, Montakhab M R & Woollons D J, 1993; "Automated fault tree analysis for hydraulic systems"; Presented at the ASME Winter Annual Meeting, New Orleans, Luisiana, Nov. 28-Dec. 3

Holland J H, 1975; "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor

Hooke R & Jeeves T A, 1961; "Direct Search Solution of Numerical and Statistical Problems"; J. Assn. Comp. Mach, No. 8, pp. 212-229

Hou E S H, Ansari N & Ren H, 1994; "A Genetic Algorithm for Multiprocessor Scheduling"; IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 2, pp. 113-120

Howe R M, 1994a; "Emulation of Real-time Asynchronous Multi-processor Simulation Using a Spread Sheet"; Ann Arbor, Michigan, Applied Dynamics International

Howe R M, 1994b; "Real-time Simulation of Dynamic Systems with Friction Using Variable-step Integration"; Applied Dynamics International, Ann Arbor, Michigan

Howe R M, 1993; "Real-time Asynchronous Multi-processor Simulation Using Variable-step Integration"; Ann Arbor, Michigan, October, 1993, Applied Dynamics International

Hugh M, 1995; "The Design of Hydraulic Components and Systems"; Ellis Horwood Limited, ISBN 0-13-297194-1

Hui S Y R & Christopoulos C, 1991; "The use of TLM-based discrete transforms for general numerical analysis"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Hui S Y R, Fung K K, Zhang M Q, Christopoulos C, 1993a; "Variable time step technique for transmission line modelling"; IEE Proceedings-A, Vol. 140, No. 4, July 1993, pp 299-302

Hui S Y R, Fung K K, Zhang M Q & Christopoulos C, 1993b; "Variable Time Step TLM Algorithm: A Unified Approach"; Electronics Letters, 27th May, 1993, Vol. 29, No. 11, pp 978-979

Hunt T M, 1986; "A Review of Condition Monitoring Techniques Applicable to Fluid Power Systems", 7th International Fluid Power Symposium, University of Bath, UK.

Hurley S, 1993; "Taskgraph mapping using a genetic algorithm: A comparison of fitness functions"; Parallel Computing 19, pp. 1313-1317

Hwang K & Briggs F A, 1987; "Computer Architecture and Parallel Processing"; McGraw-Hill International Editions, ISBN 0-07-Y66354-8

INMOS, 1994a; "T9000 ANSI C toolset user guide"; SGS-Thomson Microelectronics Limited, Document Number: 72 TDS 408 02

INMOS, 1994b; "T9000 toolset hardware configuration manual"; SGS-Thomson Microelectronics Limited, Document Number: 72 TDS 427 00

Isermann R, 1993; "Fault Diagnosis of Machines Via Parameter Estimation and Knowledge Processing"; IFAC Symposium Safeprocess, Baden-Baden; Automatica, Vol. 29, No 4, pp 815-835

Isermann R & Freyermuth B 1993; Editorial to "Special Section on Fault Detection, Supervision and Safety for Technical Processes"; Automatica, Vol. 29, No. 4, pp. 813-814

Isermann R & Freyermuth B 1991a; "Process Fault Diagnosis Based on Process Model Knowledge - Part I: Principles for Fault Diagnosis With Parameter Estimation"; Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control, Vol. 113, December, pp. 620-626

Isermann R & Freyermuth B 1991b; "Process Fault Diagnosis Based on Process Model Knowledge - Part II: Case Study Experiments"; Transactions of the ASME, Journal of Dynamic Systems, Measurement, and Control, Vol. 113, December, pp. 627-633

Jansson A, Krus P & Palmberg J-O, 1992; "Variable Time Step Size applied to Simulation of Fluid Power Systems using Transmission Line Elements"; 5th Bath Int. Fluid Power Workshop.

Jansson A & Krus P, 1991; "Real-Time Simulation Using Parallel Processing"; 2nd Tampere Int. Conference on Fluid Power, pp. 27-39

Jin H, Vahldieck R, 1992; "The Frequency-Domain Transmission Line Matrix Method - A New Concept"; IEEE Transactions on Microwave Theory and Techniques, Vol. 40, No. 12, December 1992, pp 2207-2218

Johns P & Butler G, 1983; "The consistency and accuracy of the TLM method for diffusion and its relationship to existing methods"; Int. Journal for Numerical Methods in Engineering, Vol. 19, pp. 1549-1554

Johns P B & O'Brien M, 1980; "Use of the Transmission Line Modelling (TLM) Method to Solve Non-linear Lumped Networks"; The Radio and Electronic Engineer, Vol 50, No 1/2, pp 59-70.

Johns P B & Beurle R L, 1971; "Numerical solution of 2-dimensional scattering problems using a transmission-line matrix"; Proc. IEEE, Vol. 118, No. 9, September, pp. 1203-1208

Jones R D & Beltramo M A, 1991; "Solving partitioning Problems with Genetic Algorithms"; in Belew R K & Booker L B (editors), Proceedings of the Fourth International Conference of Genetic Algorithms, San Diego, 13-16. July, Margan Kaufmann Publishers

Krishnakumar K & Goldberg D E, 1992; "Control System Optimization Using Genetic Algorithms"; Journal of Guidance, Control, and Dynamics, Vol. 15, No. 3, pp. 735-740

Kristinsson K & Dumont G A, 1992; "System Identification and Control Using Genetic Algorithms"; IEEE Transactions on Systems, Man, and Cybernetics, Vol. 22, No. 5, September/October, pp. 1033-1046

Krus P, 1995; "Modelling of Mechanical Systems using Rigid Bodies and Transmission Line Joints"; ASME, FPST-Vol. 2, Fluid Power Systems and Technology, pp. 107-114

Krus P, Weddfelt K & Palmberg J-O, 1994; "Fast Pipeline Models for Simulation of Hydraulic Systems"; ASME Winter Annual Meeting, December 1-6, Atlanta, GA, Transaction of the ASME, Vol. 116, March, pp 132-136

Krus P, Karlsson M & Engvall J, 1991; "Modelling and Simulation of the Human Arterial Tree, Using Transmission Line Elements with Visco-Elastic Walls"; ASME, Atlanta

Krus P, Jansson A, Palmberg J-O & Weddfelt K, 1990; "Distributed Simulation of Hydromechanical Systems"; Proc. 3rd Bath Int. Fluid Power Workshop, Research Studies Press, UK.

Krus P & Palmberg J-O, 1987; "Simulation of fluid power systems with long lines"; Modelling Identification and Control, MIC87, IASTED Conference

Kumar V, Grama A, Gupta A & Karypis G, 1994; "Introduction to parallel computing - Design and Analysis of Algorithms"; Chapter 4: Performance and Scalability of Parallel Systems"; Chapter 7: Graph Algorithms; Chapter 12: Systolic Algorithms and their Mapping onto Parallel Computers; The Benjamin/Cummings Publishing Company, Inc.; ISBN 0-8053-3170-0

Langen H J, 1981; "Schadensfrüherkennung an Hydraulikbauelementen mit Hilfe der Körperschallmeßmethode"; (Early damage diagnosis of hydraulic components using vibration measurement); O+P, Ölhydraulik und Pneumatik, 25, Nr. 7, pp. 568-573

von Laszewski G & Mühlenbein H, 1990; "Partitioning a Graph with a Parallel Genetic Algorithm"; Lecture Notes in Computer Science, 496, Parallel Problem Solving from Nature, 1st Workshop, Editors: Schwefel H-P & Männer R, pp. 165-169

Lin Y & Kortüm, 1992; "Identification of system physical parameters for vehicle systems with nonlinear components"; Vehicle System Dynamics, 20, (SS), pp. 354-365

Liu P, Dransfield P & Stecki J S, 1992; "Neural Nets - Their Potential for Intelligent Control of Fluid Power Drives"; 10th Aachener Fluidtechnisches Kolloquium, 17-19 März

Ljung L, 1987; "System Identification: Theory for the User"; Prentice-Hall, Inc., Englewood Cliffs, New Jersy 07632, ISBN: 0 13 881640 9

Lunze J, 1992; "Qualitative modelling of dynamical systems for on-line diagnosis"; Intelligent Systems Engineering, IEE Conference Publication No. 360

Masters T, 1993; "Practical Neural Network Recipes in C++"; Academic Press, Inc., London, UK., ISBN: 0 12 479040 2

McCandish D & Dorey R E, 1984; "The mathematical modelling of hydrostatic pumps and motors"; Proc Instn Mech Engrs, IMechE, Vol. 198B, No. 10, pp 165-174

Mehra R K, 1979; "Nonlinear System Identification: Selected Survey and Recent Trends"; 5th IFAC Symposium on Identification and System Parameter Estimation, Darmstadt, pp. 77-83

Mühlenbein H, Gorges-Schleuter M & Krämer O, 1987; "New solutions to the mapping problem of parallel systems: The evolution approach"; Parallel Computing, 4, pp. 269-279

Muntean T & Talbi E-G, 1991; "A parallel genetic algorithm for process-processors mapping"; High Performance Computing II, M. Durand and F. El Dabaghi (Editors), Elsevier Science Publishers B. V., pp. 71-82

Natke H G, Juang J-N & Gawronski W, 1988; "A Brief Review on the Identification of Nonlinear Mechanical Systems"; Proc. 6th Int. Modal Analysis Conference, 2(275), pp. 1569-1574

Naujoks T & Wurmthaler C, 1988; "Bilinearer Regler- und Beobachterentwurf für nichtlineare Systeme am Beispiel eines hydraulischen Drehantriebes"; (Bilinear controller and observer design for nonlinear systems with application to a hydraulic rotary drive); Automatisierungstechnik at, 36. Jahrgang, Heft 1, pp. 32-37

Neuhaus P, 1990; "Solving the Mapping Problem: Experience with a Genetic Algorithm"; Lecture Notes in Computer Science, 496, Parallel Problem Solving from Nature, 1st Workshop, Editors: Schwefel H-P & Männer R, pp. 170-175

Özgüner F & Erçal F, (Editors), 1991; "Parallel Computing on Distributed Memory Multiprocessors"; NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 103

O+P, 1995; "Früherkennung von Schäden an hydraulischen Maschinen und Anlagen"; O+P, Ölhydraulik und Pneumatik, 39, Nr. 4, pp. 324-327

O+P, 1978; "O+P Gespräch: Schadensfrüherkennung an hydraulischen Anlagen"; (O+P Discussion: Early damage diagnosis in hydraulic systems); O+P, Ölhydraulik und Pneumatik, 22, Nr. 3, pp. 121-136

Palmberg J-O, 1991; "Fluid Power Engineering: A Field for Computer Application"; Proc. 2nd Int. Conference on Fluid Power, Tampere, Finland, pp 415-431.

Pall 1985; "The Pall 9640 Series Pressure Balanced Duplex Filter", Brochure RPA/MP/1M/885, Pall Industrial Hydraulics Ltd., Europa House, Havant street, Portsmouth, PO1 3PD, UK.

Partridge G J, Christopoulus C & Johns P B, 1987; "Transmission Line Modelling of Shaft System Dynamics"; Proc. IMechE Vol. 201 C4, pp 271-278.

Pirkul H, Rolland E, 1994; "New Heuristic Solution Procedures for the Uniform Graph Partitioning Problem: Extensions and Evaluation"; Computers and Operations Research, Vol. 21, No. 8, pp. 895-907

Pollard J F, Broussard M R, Garrison D B & San K Y, 1992; "Process identification using neural networks"; Computer chem. Engng., Vol. 16, No. 4, pp. 253-270

Pollmeier K, 1996a; "TLM modelling of fluid power and mechanical systems", Proceedings of an informal meeting held at the University of East Anglia, Norwich on 27th June, Ed. Donard de Cogan, ISBN 1 898290 01 6

Pollmeier K, 1996b; "Parallel computing for real-time simulation of complex fluid power systems"; Transfer Report, University of Bath

Pollmeier K, 1996c; "Program generator for parallel TLM simulation - User guide and software description"; University of Bath, School of Mechanical Engineering, Report No 06/1996

Pollmeier K, 1996d; "Parallel simulation of complex fluid power systems - A new method to reduce the communication between processors"; Proceedings Part I, Journal of Systems and Control Engineering, IMechE, Vol. 210, No 14, pp. 221-230

Pollmeier K, Strößenreuther F, Burrows C R & Edge K A, 1996; "Identification of Nonlinear Components for Condition Monitoring in Hydraulic Systems", 12. Aachener Fluidtechnisches Kolloquium, 12.-13. März, Aachen, Germany

Pollmeier K, Burrows C R & Edge K A, 1995; "Parallel simulation of complex fluid power systems: A new method to reduce the communication between processors", Presented at the ASME Winter Annual Meeting, San Francisco, California

Pomeroy S C, 1991; "Introduction to the modelling of wave propagation using TLM"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Pomeroy S C, Zhang G, Wykes C, 1993; "Variable Coefficient Absorbing Boundary Condition for TLM"; Electronics Letters, 24th June 1993, Bol. 29, No. 13, pp 1189-1200

Pouliezos A D & Stavrakakis G S, 1994; "Real Time Fault Monitoring of Industrial Processes"; Kluwer Academic Publishers, Dordrecht/Boston/London, ISBN: 0 7923 2737 3

Pulko S H & Olashore A O, 1989; "Modelling of Heat Transfer From Fluid Sources Using The TLM Technique"; Int. J. Modelling & Sim., Vol 9, No 4, pp 96-100.

Pulko S H, Mallik A, Allen R & Johns P, 1990; "Automatic timestepping in TLM routines for the modelling of thermal diffusion processes"; International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 3, pp. 127-136.

Pulko S H & Newton HR, 1991; "Modelling diffusion based processes using TLM"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Pulka S H & Wilkinson A J, 1996; "TLM at the University of Hull", Proceedings of an informal meeting held at the University of East Anglia, Norwich on 27th June, Ed. Donard de Cogan, ISBN 1 898290 01 6

Ramdén T, Krus P & Palmberg J-O, 1995; "Fault Diagnosis of Complex Fluid Power Systems Using Neural Networks"; 4th Scandinavian International Conference on Fluid Power, 26-29 September, Tampere, Finland, Vol. 2, pp. 706-718

Ramdén T, 1995; "Tillståndskontroll av komponenter och system" ; Hydraulikdager I Linköping, Sweden, 30 - 31 maj, pp. 215-234

del Re L & Rene Keusch, 1989; "Modelling and Simulation of a Hydrostatic Drive in a State Variable Controlled Loop"; Proc. 3rd European Simulation Congress, Edinburgh, pp. 779-785

Rechenberg I, 1973; "Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution", Frommann-Holzboog (Stuttgart, Germany)

Richards C W, 1993; "Bath*fp* Manual Volume 2 - The Bath*fp* Model Reference Guide", University of Bath, Fluid Power Centre

Rodríguez C, Rementería S, Martín J I, Lafuente A, Muguerza J & Pérez J, 1996; "A Modular Neural Network Approach to Fault Diagnosis"; IEEE Transactions on Neural Networks, Vol. 7, No. 2, pp. 326-340

Rumelhart D E & McClelland J L, 1987; "Parallel Distributed Processing - Explorations in the Microstructure of Cognition", Volume 1: Foundations; MIT Press, Cambridge, Massachusetts, London, England, ISBN: 0 262 18120 7

Sadiku M N O & Agba L C, 1990; "A Simple Introduction to the Transmission-Line Modelling"; IEEE Transactions on Circuits and Systems, Vol. 37, No. 8, August 1990, pp 991-999

Saleh A H M, 1991; "The application of TLM in sonar and underwater acoustics"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Sarkar V, 1989; "Partitioning and Scheduling Parallel Programs for Multiprocessors"; Pitman, London, The MIT Press, Cambridge, Massachusetts, ISBN 0-273-08802-5

Sato S, Kume S & Kobayashi K, 1990; "Non-linear Observer of Electro-Hydraulic Servomotor"; Proc. 3rd Bath Int. Fluid Power Workshop, Research Studies Press, UK.

van Schothorst G, Teerhuis P C & van der Weiden A J J, 1995; "Describing-Function-Based Identification of a Nonlinear Servo-Valve Model"; ASME, FPST-Vol. 2, Fluid Power Systems and Technology, pp. 131-138

Sepehri N, Wan F L K, Lawrence P D & Dumont G A M, 1994; "Hydraulic compliance identification using a parallel genetic algorithm"; Mechatronics, Vol. 4, No. 6, pp. 617-633

Shen C-C & Tsai W-H, 1985; "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion"; IEEE Transactions on computers, Vol. C-34, No. 3, March, pp. 197-203

Shen Q & Leitch R, 1992; "Qualitative model based diagnosis of continuous dynamic systems"; Intelligent Systems Engineering, IEE Conference Publication No. 360

SNNS, 1995; "Stuttgart Neural Network Simulator"; User Manual, Version 4.1, Report No. 6/95, University of Stuttgart, Institute for Parallel and Distributed High Performance Systems (IPVR)

So P P M, Eswarappa C & Hoefer W J R, 1995; "Parallel and distributed TLM computation with signal processing for electromagnetic field modelling"; Int. Journal of numerical modelling: Electronic Networks, Devices and Fields, Vol. 8, pp. 169-185

Sorsa T & Koivo H N, 1993; "Application of Artificial Neural Networks in Process Fault Diagnosis"; Automatica, Vol. 29, No. 4, pp. 843-849

Spirakis P, 1986; "The Parallel Complexity of Deadlock Detection"; Lecture Notes in Compute Science, No. 233, Ed. G Goos & J Hartmanis, Mathematical Foundations of Computer Science

Starkweather T, Whitley D & Mathias K, 1990; "Optimization using distributed Genetic Algorithms"; Lecture Notes in Computer Science, No. 496, Parallel Problem Solving from Nature, 1st Workshop, PPSN I, Dortmund, FRG, pp. 176-185

Stothard & Pomeroy,1996; "An application specific TLM array processor", Proceedings of an informal meeting held at the University of East Anglia, Norwich on 27th June, Ed. Donard de Cogan, ISBN 1 898290 01 6

Sunter J P E & Bakkers A W P, 1994; "Performance of Post-Game Analysis on Transputers"; WTC'94 Tutorial, 1994, Villa Erba, Cernobbio, Lake Como, Italy

Tan C C & Fusco V F, 1993; "TLM Modelling using an SIMD Computer"; International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, Vol. 6, pp. 299-304

Tan S, Hao J and Vandewalle J, 1995; "Efficient identification of RBF neural net models for nonlinear discrete-time multivariable dynamical systems"; Neurocomuting, Vol. 9, pp. 11-26

Thau F E, 1973; "Observing the State of Non-Linear Dynamic Systems"; Int. Jnl of Control, Vol 17, pp 471-479.

Thomson M, Schooling S P & Soufian M, 1996; "The Practical Application of a Nonlinear Identification Methodology"; Control Eng. Practice, Vol. 4, No. 3, pp. 295-306

Tomlinson S, 1987; "The Hydraulic Automatic Simulation Package (HASP): Modelling and Simulation Aspects"; PhD Thesis, University of Bath, UK.

Vetri J Lo & Simons RS, 1993; "A Class of Symmetrical Condensed Node TLM Methods Derived Directly from Maxwell's Equations"; IEEE Transactions on Microwave Theory and Techniques, Vol. 41, No. 8, August 1993, pp 1419-1428

Viersma T J, 1980; "Analysis, Synthesis and Design of Hydraulic Servo-systems and Pipelines"; Elsevier spc., ISBN: 0444418695

Watton J, 1992; "Condition monitoring and fault diagnosis in fluid power systems"; Ellis Horwood Limited, Chichester, England, ISBN: 0-13-176405-5

Webb P W, 1991; "Transmission line matrix modelling applied to thermal diffusion problems"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Welch P, Peel R, 1994; "Parallel Design Concepts/Methods for Communicating Process Architectures"; World Transputer Congress, 4. September 1994, Lake Como, Italy, Tutorial

Xue Y & Watton J, 1995; "A self-organizing neural network approach to data-based modelling of fluid power systems dynamics using the GMDH algorithm"; IMechE, Proc Instn Mech Engrs, Vol. 209, pp. 229-240

Yao L & Sethares W A, 1994; "Nonlinear Parameter Estimation via the Genetic Algorithm"; IEEE Transactions on Signal Processing, Vol. 42, No. 4, pp. 927-935

Yongxiang L & Zhangwei C, 1995; "A Knowledge-based Electrohydraulic Servo-System Fault Diagnostic Model"; 4th Scandinavian International Conference on Fluid Power, 26-29 September, Tampere, Finland, Vol. 2, pp. 686-694

Yu D, Shields D N & Mahtani J L, 1994; "A Nonlinear Fault Detection Method for a Hydraulic System"; Control '94, 21-24 March, Conference Publication No. 389, pp. 1318-1322

Zhang G, Wykes C & Pomeroy S C, 1991; "The application of TLM to the modelling of airborne ultrasonics"; IEE "Tutorial Colloquium on Transmission line matrix modelling - TLM", Friday, 18 October 1991, London, Digest No: 1991/175 (Organised by professional groups E15 and C6)

Zhong X & Lo V M, 1992; "Application-Specific Deadlock Free Wormhole Routing on Multicomputers"; Lecture Notes on Computer Science, No. 605, PARLE '92, Parallel Architectures and Languages Europe, 4th Int. PARLE Conference, Paris, June, pp. 193-208

Zhou D H, Sun Y X, Xi Y G & Zhang Z J, 1991; "Real-Time Detection and Diagnosis of "Parameter Blas" Faults for Non-Linear Systems"; IFAC Symposium Safeprocess, Baden-Baden.

Appendix A The basic equations for TLM modelling of fluid power systems

For the analysis of the dynamic behaviour of hydraulic pipelines having distributed parameters the basic equations are derived, based upon the following assumptions:

- laminar flow (Re < 2000)
- cylindrical pipe with non-elastic walls
- the temperature is supposed to be constant, hence the fluid viscosity is considered to be constant
- differences of pressure and density in radial direction are neglected
- radial fluid speed v is also neglected

It is convenient to use cylindrical coordinates where the x-axis is identified with the centre line of the pipe, r is the coordinate in radial direction and t denotes time as shown in Figure A1 and Figure A2.

Summarising the assumptions, we have the following variables

$$\begin{aligned} P(x,r,t) &= P(x,t) \\ \rho(x,r,t) &= \rho(x,t) \\ Q(x,r,t) &= Q(x,t) \\ v(x,r,t) &= 0 \end{aligned} \tag{A1}$$

The flow is calculated by using the mean value of the axial speed $\bar{u}$ in a pipe with the radius R and cross-section A:

$$Q(x,t) = A \cdot \bar{u}(x,t) = \frac{A}{\pi R^2} \int_0^R 2\pi r u(x,t,r) dr = 2\pi \int_0^R r u(x,t,r) dr \tag{A2}$$

A1 Continuity

With the notation in Figure A1 the conservation of mass leads to the following equation

$$\rho Q - \left(\rho + \frac{\partial \rho}{\partial x} dx\right)\left(Q + \frac{\partial Q}{\partial x} dx\right) = \frac{\partial m}{\partial t} = A dx \frac{\partial \rho}{\partial t} \tag{A3}$$

eliminating products of small quantities and dividing by dx

$$-\frac{\partial Q}{\partial x} - \frac{Q}{\rho}\frac{\partial \rho}{\partial x} = \frac{A}{\rho}\frac{\partial \rho}{\partial t} \tag{A4}$$

From the definition of the bulk modulus

$$B = -V\frac{\partial \rho}{\partial V} \tag{A5}$$

the equation of state can be derived [Hugh, 1995]

$$\frac{\partial \rho}{\rho} = \frac{\partial P}{B} \tag{A6}$$

this leads to

$$-\frac{\partial Q}{\partial x} - \frac{Q}{B}\frac{\partial P}{\partial x} = \frac{A}{B}\frac{\partial P}{\partial t} \tag{A7}$$

with the following assumption [D'Souza & Oldenburger, 1964]

$$Q\frac{\partial P}{\partial x} << A\frac{\partial P}{\partial t} \tag{A8}$$

this leads to

$$\frac{A}{B}\frac{\partial P}{\partial t} + \frac{\partial Q}{\partial x} = 0 \tag{A9}$$

Although, cylindrical pipes with non-elastic walls were assumed for pipe of hose pipe elasticity can be accounted for by using an effective bulk modulus.

A2 Momentum

Owing to the conservation of momentum of the mass inside the control volume of Figure A2 we find the equation of motion in x-direction:

$$F_x = \frac{m}{A}\frac{dQ}{dt} \tag{A10}$$

$$\frac{2\pi r dr dx}{A}\rho\frac{dQ}{dt} = 2\pi r dr\left[P - \left(P + \frac{\partial P}{\partial x}dx\right)\right] + 2\pi r dx\tau - 2\pi(r+dr)\left(\tau + \frac{\partial \tau}{\partial r}dr\right)dx \tag{A11}$$

eliminating products of small quantities and dividing by dx leads to

$$\frac{\rho}{A}\frac{dQ}{dt} = -\frac{\partial P}{\partial x} - \frac{\tau}{r} - \frac{\partial \tau}{\partial r} \tag{A12}$$

with the following assumption [Viersma, 1980]

$$\frac{dQ}{dt}=\frac{\partial Q}{\partial t}+Q\frac{\partial Q}{\partial x}\approx\frac{\partial Q}{\partial t} \tag{A13}$$

and Newton's viscosity law [Backé, 1988]

$$\tau=-\frac{\mu}{A}\frac{\partial Q}{\partial r} \tag{A14}$$

this leads to

$$\frac{\rho}{A}\frac{\partial Q}{\partial t}=-\frac{\partial P}{\partial x}+\frac{\mu}{Ar}\frac{\partial Q}{\partial r}+\frac{\mu}{A^2}\frac{\partial^2 Q}{\partial r^2} \tag{A15}$$

For the case where μ is assumed to be zero (ignoring frictional effects), the equation reduces to the basic water-hammer equation

$$\frac{\rho}{A}\frac{\partial Q}{\partial t}+\frac{\partial P}{\partial x}=0 \tag{A16}$$

Laplace transforming the friction-less equations A9 and A16 leads to

$$\frac{As}{B}P(x,s)+\frac{\partial Q(x,s)}{\partial x}=0 \tag{A17}$$

$$\frac{\rho s}{A}Q(x,s)+\frac{\partial P(x,s)}{\partial x}=0 \tag{A18}$$

Equation A15 which includes friction can also be Laplace transformed [Viersma, 1980] to

$$\frac{\rho s}{A}Q(x,s)N(s)+\frac{\partial P(x,s)}{\partial x}=0 \tag{A19}$$

where $N(s)$, the viscous friction factor, is defined as

$$N(s)=-\frac{J_0\left(j\cdot\sqrt{\frac{s}{\nu}}\cdot r\right)}{J_2\left(j\cdot\sqrt{\frac{s}{\nu}}\cdot r\right)} \tag{A20}$$

with J_i the Bessel function of the order $i=0$ and $i=2$, respectively. Differentiating equations A17 and A19 partially w.r.t. x and rearranging

$$\frac{\partial P(x,s)}{\partial x}=-\frac{B}{As}\frac{\partial^2 Q(x,s)}{\partial x^2} \tag{A21}$$

$$\frac{\partial Q(x,s)}{\partial x} = -\frac{1}{N(s)}\frac{A}{\rho s}\frac{\partial^2 P(x,s)}{\partial x^2} \tag{A22}$$

Substituting these results into equations A17 and A19

$$\frac{As}{B}P(x,s) - \frac{A}{\rho s}\frac{\partial^2 P(x,s)}{\partial x^2} = 0 \tag{A23}$$

$$\frac{\rho s}{A}Q(x,s)N(s) - \frac{B}{As}\frac{\partial^2 Q(x,s)}{\partial x^2} = 0 \tag{A24}$$

Introducing the characteristic impedance Z_c and the sound velocity a

$$Z_c = \frac{\rho a}{A} \tag{A25}$$

$$a = \sqrt{\frac{B}{\rho}} \tag{A26}$$

reduces the number of relevant system parameters from three (B, A, ρ) to two (Z_c, a) and equations A23 and A24 can be rewritten to

$$\frac{s}{aZ_c}P(x,s) - \frac{1}{N(s)}\frac{a}{Z_c s}\frac{\partial^2 P(x,s)}{\partial x^2} = 0 \tag{A27}$$

$$\frac{Z_c s}{a}Q(x,s)N(s) - \frac{aZ_c}{s}\frac{\partial^2 Q(x,s)}{\partial x^2} = 0 \tag{A28}$$

rearranging leads to

$$\frac{\partial^2 P(x,s)}{\partial x^2} - \frac{s^2}{a^2}P(x,s)N(s) = 0 \tag{A29}$$

$$\frac{\partial^2 Q(x,s)}{\partial x^2} - \frac{s^2}{a^2}Q(x,s)N(s) = 0 \tag{A30}$$

A3 Solving the differential equations

Replacing $N(s)$ by N to aid clarity, the solution of this second order differential equations is given by

$$Q(x,s) = C_1 e^{\frac{sx}{a}\sqrt{N}} + C_2 e^{-\frac{sx}{a}\sqrt{N}} \tag{A31}$$

$$P(x,s) = C_3 e^{\frac{sx}{a}\sqrt{N}} + C_4 e^{-\frac{sx}{a}\sqrt{N}} \tag{A32}$$

Differentiating equations A31 and A32 w.r.t. x

$$\frac{\partial Q}{\partial x} = \frac{s}{a}\sqrt{N}\cdot C_1 e^{\frac{sx}{a}\sqrt{N}} - \frac{s}{a}\sqrt{N}\cdot C_2 e^{-\frac{sx}{a}\sqrt{N}} \tag{A33}$$

$$\frac{\partial P}{\partial x} = \frac{s}{a}\sqrt{N}\cdot C_3 e^{\frac{sx}{a}\sqrt{N}} - \frac{s}{a}\sqrt{N}\cdot C_4 e^{-\frac{sx}{a}\sqrt{N}} \tag{A34}$$

substituting these into equations A17 and A19

$$\frac{As}{B}P(x,s) + \frac{s}{a}\sqrt{N}\cdot C_1 e^{\frac{sx}{a}\sqrt{N}} - \frac{s}{a}\sqrt{N}\cdot C_2 e^{-\frac{sx}{a}\sqrt{N}} = 0 \tag{A35}$$

$$\frac{\rho s}{A}Q(x,s) + \frac{s}{a}\frac{1}{\sqrt{N}}\cdot C_3 e^{\frac{sx}{a}\sqrt{N}} - \frac{s}{a}\frac{1}{\sqrt{N}}\cdot C_4 e^{-\frac{sx}{a}\sqrt{N}} = 0 \tag{A36}$$

rearranging leads to

$$C_1 = \left[C_2\cdot e^{-\frac{sx}{a}\sqrt{N}} - \frac{a}{\sqrt{N}}\frac{A}{B}P(x,s)\right]\cdot e^{-\frac{sx}{a}\sqrt{N}} \tag{A37}$$

$$C_3 = \left[C_4\cdot e^{-\frac{sx}{a}\sqrt{N}} - \sqrt{N}\frac{a\rho}{A}Q(x,s)\right]\cdot e^{-\frac{sx}{a}\sqrt{N}} \tag{A38}$$

with equations A25 and A26 this leads to

$$C_1 = \left[C_2\cdot e^{-\frac{sx}{a}\sqrt{N}} - \frac{1}{Z_c\sqrt{N}}P(x,s)\right]\cdot e^{-\frac{sx}{a}\sqrt{N}} \tag{A39}$$

$$C_3 = \left[C_4\cdot e^{-\frac{sx}{a}\sqrt{N}} - Z_c\sqrt{N}Q(x,s)\right]\cdot e^{-\frac{sx}{a}\sqrt{N}} \tag{A40}$$

substituting these into equations A31 and A32 and rearranging

$$C_2 = \left(Q + \frac{1}{Z_c\sqrt{N}}P\right)\cdot\frac{1}{2}e^{\frac{sx}{a}\sqrt{N}} \tag{A41}$$

$$C_4 = \left(P + Z_c\sqrt{N}Q\right)\cdot\frac{1}{2}e^{\frac{sx}{a}\sqrt{N}} \tag{A42}$$

substituting equation A41 in A31 and the boundary conditions on the left side of the pipe in Figure A3 ($P = P_a$ and $Q = Q_a$ at $x = 0$)

$$C_1 = \left(Q_a - \frac{1}{Z_c\sqrt{N}}P_a\right)\cdot\frac{1}{2} \tag{A43}$$

$$\therefore C_2 = \left(Q_a + \frac{1}{Z_c\sqrt{N}}P_a\right)\cdot\frac{1}{2} \tag{A44}$$

substituting equation A42 in A32 with the same boundary conditions

$$C_3 = \left(P_a - Z_c\sqrt{N}Q_a\right)\cdot\frac{1}{2} \tag{A45}$$

$$\therefore C_4 = \left(P_a + Z_c\sqrt{N}Q_a\right)\cdot\frac{1}{2} \tag{A46}$$

Flow and pressure can then be calculated as

$$Q(x,s) = \left(Q_a - \frac{1}{Z_c\sqrt{N}}P_a\right)\cdot\frac{1}{2}e^{\frac{sx}{a}\sqrt{N}} + \left(Q_a + \frac{1}{Z_c\sqrt{N}}P_a\right)\cdot\frac{1}{2}e^{-\frac{sx}{a}\sqrt{N}} \tag{A47}$$

$$P(x,s) = \left(P_a - Z_c\sqrt{N}Q_a\right)\cdot\frac{1}{2}e^{\frac{sx}{a}\sqrt{N}} + \left(P_a + Z_c\sqrt{N}Q_a\right)\cdot\frac{1}{2}e^{-\frac{sx}{a}\sqrt{N}} \tag{A48}$$

Using the boundary conditions on the right hand side of the pipe in Figure A3 ($P = P_b$ and $Q = -Q_b$ at $x = L$, i.e. flow into the line is defined as positive) and the wave propagation time along the line T

$$T = \frac{L}{a} \tag{A49}$$

leads to

$$-Q_b = \left(Q_a - \frac{1}{Z_c\sqrt{N}}P_a\right)\cdot\frac{1}{2}e^{sT\sqrt{N}} + \left(Q_a + \frac{1}{Z_c\sqrt{N}}P_a\right)\cdot\frac{1}{2}e^{-sT\sqrt{N}} \tag{A50}$$

$$P_b = \left(P_a - Z_c\sqrt{N}Q_a\right)\cdot\frac{1}{2}e^{sT\sqrt{N}} + \left(P_a + Z_c\sqrt{N}Q_a\right)\cdot\frac{1}{2}e^{-sT\sqrt{N}} \tag{A51}$$

This can be rearranged to produce the 'four-terminal network equation' otherwise known as the four-pole equation:

$$\begin{pmatrix} -Q_b \\ P_b \end{pmatrix} = \begin{pmatrix} \cosh\left(sT\sqrt{N}\right) & -\dfrac{\sinh\left(sT\sqrt{N}\right)}{Z_c\sqrt{N}} \\ -Z_c\sqrt{N}\sinh\left(sT\sqrt{N}\right) & \cosh\left(sT\sqrt{N}\right) \end{pmatrix} \begin{pmatrix} Q_a \\ P_a \end{pmatrix} \tag{A52}$$

For the loss-less line $(N = 1)$ taking equation A51 divided by $(-Z_c)$ and subtracted from and added to equation A50 leads to

$$P_b - Z_c Q_b = (P_a + Z_c Q_a) \cdot e^{-sT} \tag{A53}$$

$$P_a - Z_c Q_a = (P_b + Z_c Q_b) \cdot e^{-sT} \tag{A54}$$

transforming these equations into the time domain becomes

$$P_b(t) - Z_c Q_b(t) = P_a(t-T) + Z_c Q_a(t-T) \tag{A55}$$

$$P_a(t) - Z_c Q_a(t) = P_b(t-T) + Z_c Q_b(t-T) \tag{A56}$$

Hence, pressure and flow on one end of the line are equal to the respective values on the other end one time step earlier.

For more complex friction models the transformation into the time domain cannot easily be done. Some approximations of the Bessel functions needed in equation A20 are derived by Viersma [1980] which enable the transformation into the time domain.

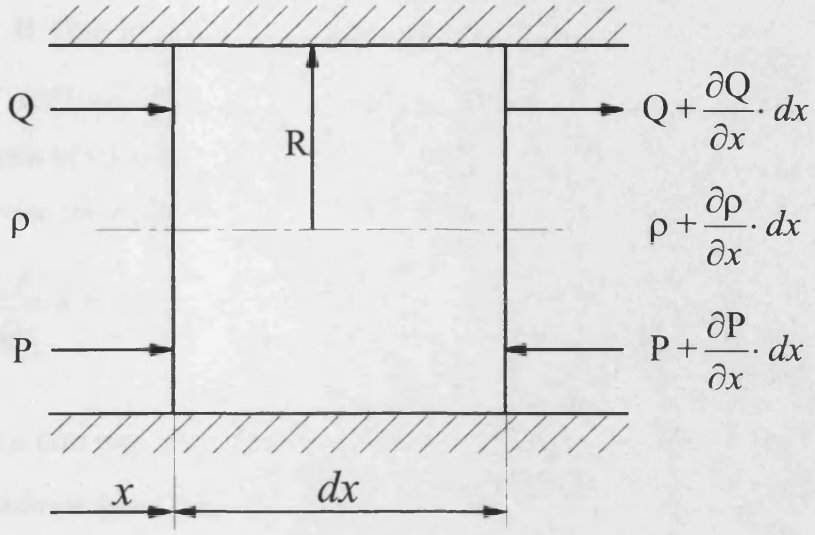


Figure A1 Control volume concerning the conservation of mass

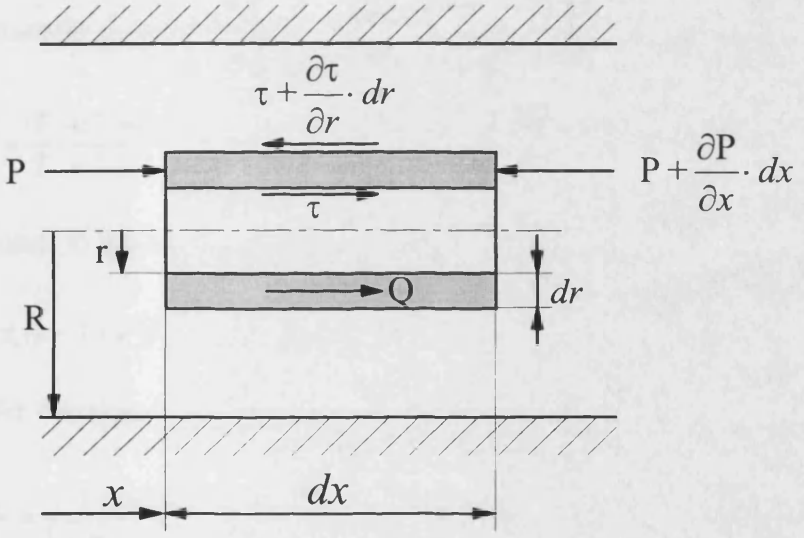


Figure A2 Control volume concerning the conservation of momentum

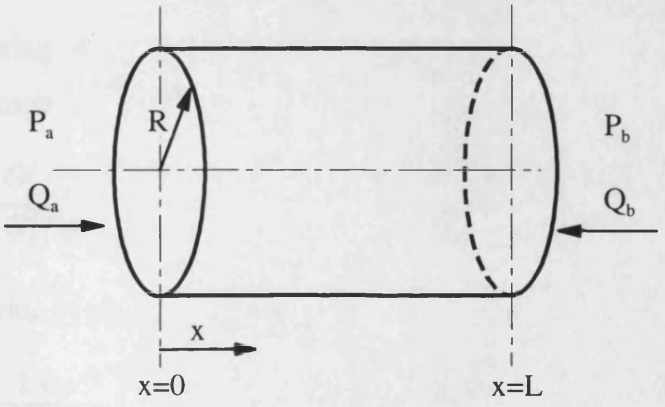


Figure A3 Pipe line with boundary conditions

Appendix B The TLM method as a general method for integration

Krus et al. [1990] compared the TLM method using characteristic pressure filtering with the trapezoidal rule of integration. Examining the integration of pressure in a closed volume the transfer function corresponding to integration can be derived as

$$G_{cr}(s) = \frac{CP_{in}}{Q_{in}} = T \cdot \frac{e^{sT} + e^{-sT}}{e^{sT} - e^{-sT}} \tag{B1}$$

where T is the time step and the hydraulic capacitance $C = V/B_e$. The Laplace transform of the trapezoidal rule for integration

$$y(t+T) = y(t) + \frac{T}{2}\left[\dot{y}(t) + \dot{y}(t+T)\right] \tag{B2}$$

leads to the transfer function

$$G_{tr}(s) = \frac{y}{\dot{y}} = \frac{T}{2} \cdot \frac{e^{sT} + 1}{e^{sT} - 1} \tag{B3}$$

This corresponds to equation (B1) with half the time step. Introducing the low pass filter

$$x_i(t) = \alpha \cdot x_i(t-T) + (1-\alpha) \cdot x_{0i}(t-T) \tag{B4}$$

and its transfer function

$$G_f(s) = \frac{x_i}{x_{0i}} = \frac{1-\alpha}{1-\alpha \cdot e^{-sT}} \tag{B5}$$

(where x_{0i} is the unfiltered value) leads to the following transfer functions for integration:

I TLM with filtering of the characteristic pressure according to Krus et al. [1990], hence filtering of pressure and flow

$$G_I(s) = k_I T \cdot \frac{1 + G_f^2(s) \cdot e^{-2sT}}{1 - G_f^2(s) \cdot e^{-2sT}} \tag{B6}$$

II TLM with filtering of the pressure only

$$G_{II}(s) = k_{II} T \cdot \frac{1 + e^{-2sT}}{1 - G_f^2(s) \cdot e^{-2sT}} \tag{B7}$$

where k_I and k_{II} are obtained using the Final Value Theorem, thus

$$k_{I,II} = \frac{\lim_{s \to 0} sG_{cr}(s)}{\lim_{s \to 0} sG_{I,II}(s)} = \frac{1}{1-\alpha} \tag{B8}$$

in order to achieve correct steady state values. Using the filters with $\alpha = 0.2$ the transfer functions in Figure B1 are obtained where the time step was arbitrarily chosen for $T = 1$. Filtering of the characteristic pressure suppresses the resonance and antiresonance peaks whereas filtering of the pressure suppresses the resonance peak only. The phase characteristic is also closer to that obtained from the trapezoidal rule.

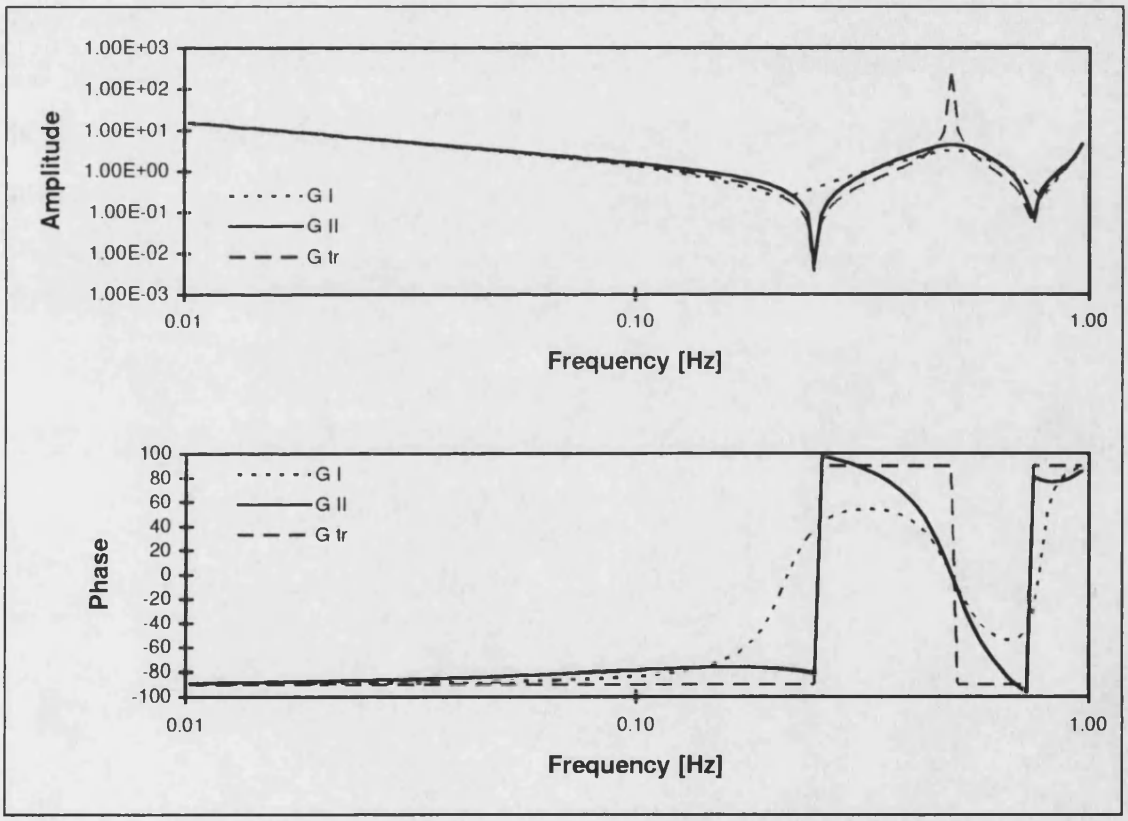


Figure B1 Bode diagram of integration using different filters compared with trapezoidal rule of integration ($\alpha = 0.2, T = 1$)

Appendix C Adjustment of the characteristic impedance

In Appendix B the correction factor for the standard TLM scheme is calculated by comparing transfer functions. The same method can be applied to derive the respective factor for the extrapolation-interpolation method. Using the notation in Figure 6.4 the less frequent pressure and flow propagation can be expressed as filters. For the propagation of pressure from subsystems A to B this leads to the following equations:

$$P(t) = P_p + (P_n - P_p)\frac{m}{n} \tag{C1}$$

$$P_n = (1-\alpha)P_{A0} + \alpha P_p \tag{C2}$$

$$P_p = P(t - mT) = P(t)e^{-smT} \tag{C3}$$

Equation C1 describes the interpolation step. P_p and P_n indicate the previous and the current exchanged pressures, respectively. With equation C2 the pressure is filtered, i.e. it is calculated from the previously exchanged (and filtered) pressure P_p and the unfiltered pressure P_{A0} propagated from subsystem A at time (t-mT). In equation C3 pressure P_p is expressed as a time delayed pressure. Combining equations C1 to C3 leads to the following pressure filter G_p:

$$G_P = \frac{P(t)}{P_{A0}} = \frac{\frac{m}{n}(1-\alpha)}{1+\left(\frac{m}{n}(1-\alpha)-1\right)e^{-smT}} \tag{C4}$$

The equivalent equations for the flow propagation (without the low-pass filter) are given in equations C5 to C7.

$$Q(t) = Q_p + (Q_n - Q_p)\frac{m}{n} \tag{C5}$$

$$Q_n = Q_{A0} \tag{C6}$$

$$Q_p = Q(t - mT) = Q(t)e^{-smT} \tag{C7}$$

Combining these equations leads to the respective flow filter G_Q in equation C8.

$$G_Q = \frac{Q(t)}{Q_{A0}} = \frac{\frac{m}{n}}{1+\left(\frac{m}{n}-1\right)e^{-smT}} \tag{C8}$$

Using the transfer function corresponding to the TLM integration from Appendix B

$$G_{cr}(s) = \frac{CP_{in}}{Q_{in}} = T \cdot \frac{1+e^{-2sT}}{1-e^{-2sT}} \tag{C9}$$

(compare equation B1) the new transfer function can be derived to

$$G_{III}(s) = k_{III} T \cdot \frac{1+G_Q^2(s)\cdot e^{-2sT}}{1-G_P^2(s)\cdot e^{-2sT}} \tag{C10}$$

where k_{III} can again be obtained using the Final Value Theorem, thus

$$k_{III} = \frac{\lim\limits_{s\to 0} sG_{cr}(s)}{\lim\limits_{s\to 0} sG_{III}(s)} = \frac{1}{1-\alpha}\frac{5}{5n-4m+4} \tag{C11}$$

in order to achieve correct steady state values. For the case $m = n = 1$, i.e. exchange of data every time step, the factor k_{III} equals k_I and k_{II} from Appendix B.

Appendix D Component modelling

D1 Directional control valves

In general, the (spool) velocity and displacement can be integrated by the trapezoidal rule of integration.

$$\dot{x}_{n+1} = \frac{\Delta t}{2}\left(\ddot{x}_n + \ddot{x}_{n+1}\right) + \dot{x}_n = \frac{\Delta t}{2}\ddot{x}_{n+1} + \frac{\Delta t}{2}\ddot{x}_n + \dot{x}_n \tag{D1}$$

$$x_{n+1} = \frac{\Delta t}{2}\left(\dot{x}_n + \dot{x}_{n+1}\right) + x_n = \frac{\Delta t}{2}\dot{x}_{n+1} + \frac{\Delta t}{2}\dot{x}_n + x_n \tag{D2}$$

$$\therefore \dot{x}_{n+1} = \frac{2}{\Delta t}\left\{x_{n+1} - \frac{\Delta t}{2}\dot{x}_n - x_n\right\} = \frac{2}{\Delta t}\left\{x_{n+1} - x_n\right\} - \dot{x}_n \tag{D3}$$

Valve spool position is assumed to respond as a critically damped second-order system driven by the valve drive signal *I*, i.e. a pseudo-dynamic model is used to approximate the spool dynamics.

$$\ddot{x}_n = \omega^2\left(I - x_n\right) - 2\omega\dot{x}_n \tag{D4}$$

$$\ddot{x}_{n+1} = \omega^2\left(I - x_{n+1}\right) - 2\omega\dot{x}_{n+1} \tag{D5}$$

Where ω is the user supplied natural frequency. The velocity in equation D2 can be substituted by equation D1:

$$x_{n+1} = \frac{\Delta t}{2}\left(\frac{\Delta t}{2}\ddot{x}_{n+1} + \frac{\Delta t}{2}\ddot{x}_n + \dot{x}_n\right) + \frac{\Delta t}{2}\dot{x}_n + x_n \tag{D6}$$

Combining equations D4 to D6 and rearranging leads to

$$x_{n+1} = \frac{\Delta t^2}{4}\left\{\omega^2\left(2I - x_n - x_{n+1}\right) - 2\omega\dot{x}_{n+1}\right\} - \frac{\Delta t^2}{2}\omega\dot{x}_n + \Delta t\dot{x}_n + x_n \tag{D7}$$

Substituting the velocity in equation D7 from equation D3 and rearranging leads to the displacement of the spool at time n+1 depending on values at time n only.

$$x_{n+1} = \frac{\frac{\Delta t^2}{4}\left\{\omega^2\left(2I - x_n\right) + \omega\frac{4}{\Delta t}x_n\right\} + \Delta t\dot{x}_n + x_n}{1 + \frac{\Delta t^2}{4}\omega^2 + \omega\Delta t} \tag{D8}$$

In order to account for saturation the valve drive signal and the spool displacement are restricted by the following values:

$$\begin{aligned} &\text{if } I_{n+1} \le -1 \rightarrow I_{n+1} = -1 && \text{if } x_{n+1} \le -1 \rightarrow x_{n+1} = -1 \\ &\text{if } I_{n+1} \ge +1 \rightarrow I_{n+1} = +1 && \text{if } x_{n+1} \ge +1 \rightarrow x_{n+1} = +1 \end{aligned} \tag{D9}$$

The velocity of the spool can then be calculated using equation D3, i.e.

$$\dot{x}_{n+1} = \frac{2}{\Delta t}\{x_{n+1} - x_n\} - \dot{x}_n \tag{D10}$$

This velocity needs to be calculated and saved for the next time step (see equation D8). Adding/subtracting the underlap from the spool displacement enables the calculation of the annular flow area for the respective ports. The flow between each port is then simulated as a square-law orifice model by superposition of the different flows. For example, the supply flow rate (equation 2.30) is composed from three flows. These are the flow from supply to port *a*, the flow from supply to port *b* and the first stage leakage flow. The latter flow is calculated using the laminar orifice model equation. With the notation in Figure D1 this can be derived using the transmission line equations (see equations 2.5 and 2.6)

$$\begin{aligned} P_1 &= C_2 - Z_1 Q \\ P_2 &= C_1 + Z_2 Q \end{aligned} \tag{D11}$$

and the laminar pressure flow relationship

$$Q = k(P_1 - P_2) \tag{D12}$$

this leads to equation D13 enabling the calculation of flow from previous values only.

$$Q = -Q_1 = Q_2 = \frac{k(C_2 - C_1)}{1 + k(Z_1 + Z_2)} \tag{D13}$$

In this case the indices 1 and 2 represent the supply and return port, respectively. The other two flows in equation 2.30 are calculated using the TLM orifice equation. Again with the notation in Figure D1 this can be derived to the following equation (Burton [1994]).

$$Q = -Q_1 = Q_2 = -\frac{k^2}{2}(Z_1 + Z_2) + \frac{1}{2}\sqrt{k^4(Z_1 + Z_2)^2 + 4k^2(C_2 - C_1)} \tag{D14}$$

This equation is applied twice, i.e. with the indices 1 and 2 representing supply *s* and port *a* as well as supply *s* and port *b*, respectively. After all the flows have been calculated and

added according to equations 2.30 to 2.33 the respective port pressures are calculated from the following equation:

$$P_i = C_j - Z_c Q_i \tag{D15}$$

where i and j are chosen accordingly.

D2 Pressure relief valve model

In order to clarify the derivation of the following equations again the notation in Figure 2.14 is used. Equation 2.37 in the main text describes the spool dynamics according to Newton's second low. The trapezoidal rule of integration for displacement and velocity of the spool are the same as already given in equations D1 and D2, respectively. Substituting the acceleration in equation D1 by the one derived from equation 2.37 and rearranging leads to

$$\dot{x}_{n+1} = \frac{\frac{\Delta t}{2}\ddot{x}_n + \dot{x}_n + \frac{\Delta t}{2M}\left\{(C_2 - C_1 - P_c)A_1 - (Z_1 + Z_2)\cdot Q_{n+1}A_1 - K_s\left(\frac{\Delta t}{2}\dot{x}_n + x_n\right)\right\}}{1 + \frac{\Delta t}{2M}\left(C_v - K_s\frac{\Delta t}{2}\right)} \tag{D16}$$

The latter equation can then be combined with equations 2.39, D1 and the transmission line end equations in order to derive an explicit formula for the spool displacement (equation D17). This can then be used to calculate the flow (equation D18) leading to the velocity and acceleration, equations D19 and D20, respectively.

$$x_{n+1} = \frac{H_4 + H_{11}\left(H_3 + H_2\left(H_9 - H_{12}H_{13} - K_s H_4\right)\right)}{1 + H_{11}H_2H_{12}k_v H_5} \tag{D17}$$

$$Q_{n+1} = \frac{x_{n+1}k_v H_5 + H_7\cdot(C_2 - C_1) - H_6\cdot H_5}{H_8} \tag{D18}$$

$$\dot{x}_{n+1} = \frac{H_3 + H_2\left\{H_9 - (Z_1 + Z_2)\cdot Q_{n+1}A_1 - K_s H_4\right\}}{H_{10}} \tag{D19}$$

$$\ddot{x}_{n+1} = \dot{x}_{n+1} = \frac{1}{M}\left\{(C_2 - C_1 - (Z_1 + Z_2)\cdot Q_{n+1} - P_c)A_1 - C_v\dot{x}_{n+1} - K_s x_{n+1}\right\} \tag{D20}$$

with the following constants

$$H_1 = \frac{\Delta t}{2} \qquad H_2 = \frac{\Delta t}{2M} \qquad H_3 = \frac{\Delta t}{2}\ddot{x}_n + \dot{x}_n \qquad H_4 = \frac{\Delta t}{2}\dot{x}_n + x_n$$

$$H_5 = \sqrt{P_{1,n} - P_{2,n}} \qquad H_6 = k_v \frac{x_n}{2} \qquad H_7 = \frac{H_6}{H_5} \qquad H_8 = 1 + H_7(Z_1 + Z_2)$$

$$H_9 = (C_2 - C_1 - P_c)A_1 \qquad H_{10} = 1 + \frac{\Delta t}{2M}\left(C_v - K_s \frac{\Delta t}{2}\right) \qquad H_{11} = \frac{\Delta t}{2H_{10}}$$

$$H_{12} = \frac{(Z_1 + Z_2)A_1}{H_8} \qquad H_{13} = H_7 \cdot (C_2 - C_1) - H_6 \cdot H_5$$

During the simulation the spool displacement is restricted to a maximum, i.e. the spool stroke. The flow across the valve, the spool displacement, velocity and acceleration are set to zero as long as the pressure difference between input and output port is smaller than the cracking pressure.

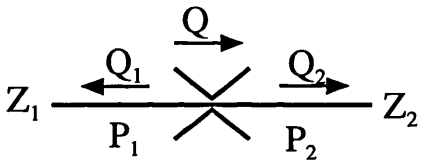


Figure D1 TLM schematic of orifice model