

University of Bath



**PHD**

**Distributed memory diesel engine simulation using transputers**

Shamail, Shafay

*Award date:*  
1990

*Awarding institution:*  
University of Bath

[Link to publication](#)

**General rights**

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

**Take down policy**

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. May. 2019

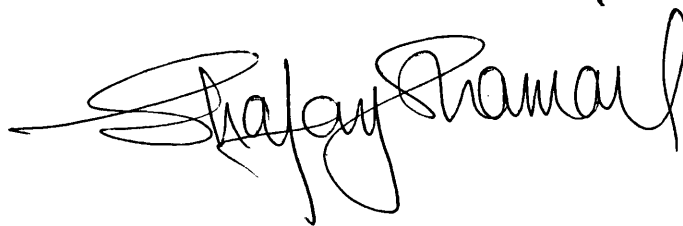
**DISTRIBUTED MEMORY  
DIESEL ENGINE SIMULATION  
USING TRANSPUTERS**

**Submitted by Shafay Shamail,  
B.Sc.(Electrical), M.Sc.(Electronics)  
for the degree of  
Doctor of Philosophy  
of the University of Bath  
1990**

**COPYRIGHT**

Attention is drawn to the fact that copyright of this thesis rests with its author. This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and no information derived from it may be published without the prior written consent of the author.

This thesis may be made available for consultation within the University library and may be photocopied or lent to other libraries for the purpose of consultation.

A handwritten signature in black ink, appearing to read 'Shafay Shamail', written in a cursive style.

UMI Number: U029804

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U029804

Published by ProQuest LLC 2014. Copyright in the Dissertation held by the Author.  
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against  
unauthorized copying under Title 17, United States Code.



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

UNIVERSITY OF BATH LIBRARY	
20 AUG 1991	33
PHD	

5054070.

**To My Parents**

# Summary

In this thesis, a multicylinder turbocharged Diesel engine simulation using a distributed memory, transputer based parallel computer is presented. The new simulation has a flexible data entry mechanism which allows the user to change different parameters of the engine and which allows their effects observed without changing the actual code. The simulation results are compared with the results of an equivalent simulation. A comparison is also made between the simulated results from an engine. Different parallel methods to increase computation speed are discussed.

# Acknowledgements

I would like to express my gratitude to Mr. A. R. Daniels and Dr. R. W. Dunn for their encouragement and support in supervising this work. I would also like to thank Dr. S. J. Charlton for his invaluable help on Diesel engine models.

I would also like to acknowledge the assistance of my colleagues in the Department of Electrical Engineering, University of Bath, in particular Dr. T. Berry, Mr. V. S. Gott and Mr. C. G. Selwyn for their advice and help throughout this project.

The permission to study in the School of Electrical Engineering by Professor F. Eastham and the grant by Government of Pakistan is also gratefully acknowledged.

Finally I wish to thank my mother whose patience and prayers were always an inspiration for me during the course of this research.

# Contents

Summary	i
Acknowledgements	ii
List of principal symbols	viii
<b>1 Introduction</b>	<b>2</b>
1.1 The Technological Revolution	2
1.2 The Computer	3
1.3 Parallel Processing	3
1.4 Diesel Engine Simulation	4
1.5 About This Thesis	5
<b>2 The Diesel Engine Model</b>	<b>8</b>
2.1 Introduction	8
2.2 The Control Volume Gas State Equations	11
2.2.1 Rate of Change of Mass	11
2.2.2 Rate of Change of Fuel to Air Ratio	12
2.2.3 Rate of Change of Temperature	12
2.3 The Cylinder Control Volume Equations	12
2.3.1 Scavenge	13
2.3.2 Induction	13
2.3.3 Compression and Power	14
2.3.4 Exhaust	15
2.3.5 Combustion	16
2.4 The Manifold Control Volume Equations	16



2.4.1	Inlet Manifold	17
2.4.2	Exhaust Manifold	17
2.5	Sub Models	18
2.5.1	Gas Properties Model	18
2.5.2	The Heat Release Models	19
2.5.3	The Heat Transfer Models	24
2.5.4	Cylinder Volume and Rate of Change of Volume	27
2.5.5	Junction Flow Models	28
2.6	The Engine Dynamics	33
2.6.1	The Engine Crankshaft	33
2.6.2	The Turbocharger Shaft	35
2.6.3	The Control Actuators	35
2.7	Summary	37
<b>3</b>	<b>The Bath University Transputer Based Parallel Computer (BUTPC)</b>	<b>46</b>
3.1	The BUTPC for Diesel Engine Simulation	46
3.2	The T800 Processing Node	47
3.3	The Input/Output System	52
3.4	The Backplane	53
3.5	Bus Arbitration	54
3.6	Inter Rack Connection	55
3.7	Summary	56
<b>4</b>	<b>The System Software</b>	<b>71</b>
4.1	Introduction	71
4.2	Helios: The Operating System	73
4.2.1	The Nucleus	74

4.2.2	The Server	77
4.2.3	The Posix Library	79
4.2.4	The User Interface	79
4.3	Communication Methods in Helios	79
4.3.1	The Language Level IO	80
4.3.2	The Posix Level IO	80
4.3.3	The System Level IO	80
4.3.4	The Kernel Level IO	81
4.4	New Communication Routines for the PDESIM	84
4.4.1	The Backplane Routines	85
4.4.2	Packets	86
4.5	Data Communication and Synchronisation	91
4.6	Summary	92
<b>5</b>	<b>Parallel Numerical Integration</b>	<b>106</b>
5.1	Introduction	106
5.2	Geometric Parallelism	106
5.3	Algorithmic Parallelism	109
5.4	Power Series Expansions	112
5.5	Single Cylinder Implementation on Transputers	113
5.5.1	Single Cylinder Implementation Using the MEPCM	114
5.5.2	Single Cylinder Implementation Using the BIOSM	116
5.5.3	Single Cylinder Implementation Using the BIPCM	118
5.6	Synchronisation Mechanism for the Block Methods	120
5.6.1	Four Point Synchronisation Using Flags	120
5.6.2	Four Point Synchronisation Using BPSignal and BPWait	121
5.7	Summary	122

<b>6</b>	<b>A Multi Cylinder Diesel Engine Simulation</b>	<b>144</b>
6.1	Introduction	144
6.2	The DISC	144
6.3	The PDESIM on the BUTPC	146
6.3.1	Task Creation and Initialisation	147
6.3.2	Data File Format	148
6.3.3	Data Communication and Synchronisation	149
6.3.4	Open and Closed Loop Connections	152
6.4	Speed Performance of the PDESIM	154
6.5	Summary	155
<b>7</b>	<b>Engine Simulation Results and their Validation</b>	<b>167</b>
7.1	Introduction	167
7.2	The Engine Cycle Parameters	167
7.2.1	The Engine Cycle Variables	168
7.2.2	The Derived Engine Parameters	172
7.3	Collection and Presentation of Engine Cycle Parameters	175
7.4	Validation of Results	178
7.4.1	The Leyland TL11 Diesel Engine	178
7.4.2	The FJH Model	180
7.4.3	The TL11 Engine	181
7.5	Summary	182
<b>8</b>	<b>Application of the Engine Simulation</b>	<b>207</b>
8.1	Introduction	207
8.2	Simulation as a Design Tool	207

8.3	Condition Monitoring and Fault Diagnosis	208
8.4	Engine Control	209
8.5	Summary	209
<b>9</b>	<b>Conclusions</b>	<b>211</b>
<b>10</b>	<b>Further Work</b>	<b>214</b>
	<b>References</b>	<b>217</b>
	<b>Appendices</b>	<b>223</b>
<b>A</b>	<b>Solution of Ordinary Differential Equations Using Power Series</b>	<b>224</b>
<b>B</b>	<b>An Example Data File for Processor Allocation</b>	<b>232</b>
<b>C</b>	<b>Data Preparation for the PDESIM</b>	<b>233</b>
<b>D</b>	<b>An Example Engine Data File</b>	<b>248</b>

# List of Principal Symbols

A	Cross Sectional Area of Valves
BDC	Bottom Dead Centre
BDCST	Broadcast
BIOSM	Block Implicit One Step Method
BIPCM	Block Implicit Predictor Corrector Method
BOOTLOC	Boot Location
BUTPC	Bath University Transputer based Parallel Computer
C	Speed of Sound
C	Thermal Capacitance
calval	Calorific Value
CPU	Central Processing Unit
cr	Compression Ratio
crl	Connecting Rod Length
CSR	Control and Status Register
DISC	Distributed Iteration Step Communication
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
EVENTREG	Event Register
FBR	Fuel Burning Rate
FAEM	Filling and Emptying Method
fmep	Friction Mean Effective Pressure

GSP	General Server Protocol
h	Step Length
ho	Specific Enthalpy
htc	Heat Transfer Coefficient
I <sup>2</sup> C	Inter-Integrated Circuit
IC	Integrated Circuit
IO	Input/Output
IOC	Input/Output Controller
J	Inertia
l	Fuel Pipe Length
LG	Link Guardian
LSB	Least Significant Bit
m	Mass
mf	Mass of Fuel
MCB	Message Control Block
MEPCM	Modified Euler Predictor Corrector Method
MSB	Most Significant Bit
ODEs	Ordinary Differential Equations
P	Pressure
PDESIM	Parallel Diesel Engine Simulation
PNUM	Processor Number
Pm	Mean pressure

Q	Quantity of Heat
r	Stroke/2
R	Gas Constant
R	Thermal Resistance
RGB	Red, Green, Blue
RK	Runge-Kutta Method
ROM	Read Only Memory
SA	Surface Area
SASI	Shugart Associates Systems Interface
SCSI	Small Computer Systems Interface
SEMTAB	Semaphore Table
t	Time
T	Temperature
TAS	Test and Set Semaphore
TDC	Top Dead Centre
TFM	Task Force Manager
Tm	Mean Temperature
u	Specific Internal Energy
V	Volume
VLSI	Very Large Scale Integration
VSC	Video and System Controller

$\beta$	Mode of Burning Factor
$\epsilon$	Maximum Allowable Error
$\gamma$	Ratio of Specific Heats
$\eta$	Efficiency
$\lambda$	Fuel to Air Ratio
$\theta$	Angle
$\tau$	Torque
$v$	Mean Piston Speed
$\omega$	Angular Speed

## Subscripts

amb	ambient
c	compressor
calval	calorific value
cl	closed cycle period
crit	critical
cyl	cylinder
d	downstream
e	engine
e	exhaust
ea	exit port to exhaust manifold
em	exhaust manifold
eng	engine
fb	fuel burnt



g	gas
gw	gas to wall
i	inlet
ia	entry port to inlet manifold
ic	intercooler
igd	ignition delay
im	inlet manifold
in	induction period
inj	injection
max	maximum
pis	piston
r	ratio
sc	scavenge period
sub	subsonic
sup	supersonic
t	turbine
tc	turbocharger
u	upstream
w	wall
wc	wall to coolant

# Chapter 1

# Introduction

"It took five months to get word back to Queen Isabella about the voyage of Columbus, two weeks for Europe to hear about Lincoln's assassination, and only 1.3 seconds to get the word from Neil Armstrong that man can walk on the moon [1]."

## 1.1 The Technological Revolution

It took man thousands of years to progress from the stone age to the atomic age, but it took him less than a century to advance from the early era of valves and tubes to the present day era of electronic technology.

Discovering the power of electricity by Edison was like the "taming of the shrew", once tamed, rapid improvements followed and electrical power moved from the experiments of the laboratories to the essentials of everyday life. New technologies emerged. Germanium and silicon were discovered as bases for thin layer electronic devices. Transistors replaced valves and were themselves then replaced by the integrated circuits [2].

The advent of technology triggered a chain reaction; new fabrication techniques followed new devices, which in turn followed new techniques. Very large scale integration (VLSI) techniques helped pack large number of individual components onto a single chip. The transistor density per package increased from one to one million [3].

## 1.2 The Computer

All these advancements brought a parallel revolution in the world of computing. The helping aids for mathematicians changed their shape from the ABACUS to pocket calculators [4] and from the analytic engine of Charles Babbage [5] to the digital computers of the present day world.

The decade of the 1940s saw the evolution of the first generation of stored program computers based on valve technology, like ENIAC [6] and EDSAC [7]. Later in the late 1970s and early 1980s, emergence of a new type of integrated circuit (IC), called a microprocessor, and developments in the field of memory devices changed the shape of digital computers and made them cheaper, smaller and reliable.

In recent years, it has become possible to pack a central processing unit (CPU), a floating point processor, some memory and input/output (IO) channels on a single IC chip. This kind of microprocessor makes it possible to realise that parallel computers could be constructed from a large number of identical units, each with its own processing unit, memory and communications.

This flood of new technology made it possible to experiment in different directions, diverting from the initially defined Von Neumann computer architecture to other novel architectures [8].

## 1.3 Parallel Processing

Although parallel computing is not a new idea, the earliest reference to parallelism is thought to be by Charles Babbage [5], the first general purpose computer that contained parallel features was ACE [9], whose commercial derivative DEUCE [10]

could perform multiplication or division in parallel with data transfer to or from memory.

As the technology developed, the demand to solve more difficult problems increased. To fulfil this demand, uniprocessor systems were proving to be inefficient and costly. It was thought that a multiprocessor system, consisting of 'n' processors would, in theory, compute 'n' times faster than a uniprocessor system. The problem, however, is to use these processors effectively and efficiently at the same time to solve a variety of different problems.

Parallel processing is finding its way into a number of applications very rapidly; from database search applications [11] to radar control in military applications [12], and, of course, in the field of simulation [13]. In this thesis, for example, parallel processing has been used in the field of Diesel engine simulation.

## **1.4 Diesel Engine Simulation**

The Diesel engine is recognised especially for its efficiency and reliability which explains its well established position in commercial vehicles and industrial applications. In recent years, Diesel engineering is becoming a fast growing applied engineering discipline designed to meet exacting demands for designing Diesel engines optimized in relation to greater efficiency and power, low noise, acceptable emissions, fuel economy and greater maintainability.

Because of the complexity of the physical phenomena involved in the Diesel engine processes, the design of engines has been a complex and demanding problem, and to get necessary 'know how' for this purpose extensive testing of prototypes has been a necessary prerequisite to all engine development. Engine simulation can be used to

systematize the knowledge obtained through this expensive engine testing. It can also reduce the amount of engine testing by narrowing the range of variables that must be studied experimentally. Engine simulation can help predict the behaviour of the engine for those variables for which little is known and in this way new experiments may be carried out and results predicted well before a prototype design is suggested [14]. For example, SPICE, the simulation program for internal combustion engines [15], can be used for a number of prototype designs before an internal combustion engine is physically built. Similarly DEEDS, the Diesel engine expert diagnostic system [16], which is an integration of engine simulation programs and expert system techniques, has been developed for maritime engine condition monitoring and fault diagnosis.

Once designed, a steady performance can be obtained by the Diesel engine if it is maintained properly. For this purpose, new electronic controls are being employed. These may include a single microprocessor control system or a number of microprocessors monitoring a number of parameters of the engine simultaneously, providing a possible parallel computer engine control system [17, 18, 19].

## **1.5 About This Thesis**

In this thesis the work carried out to simulate a Diesel engine on a transputer based parallel computer is described.

Chapter 2 describes the 'filling and emptying' method on which the Parallel Diesel Engine Simulation (PDESIM) developed in this thesis is based. This chapter also describes the state equations relating cylinders, manifolds, various junctions, actuators and shafts. A dynamic engine model is described at the end of the chapter.

In chapter 3 the Bath University Transputer based Parallel Computer (BUTPC) which is used for the PDESIM is outlined.

Chapter 4 is a description of the operating system for the BUTPC. Some of the communication limitations are discussed and new communication mechanisms are proposed to achieve higher data transfer rate for the PDESIM.

Chapter 5 discusses different parallel algorithms. It also describes implementation of these algorithms on the BUTPC.

Chapter 6 describes the simulation of a six cylinder Diesel engine on the BUTPC. It also describes the new technique applied in order to make the simulator more flexible as compared with the previous implementations by Jones [20] and Haysom [21].

In chapter 7 a brief description of a Leyland TL11 experimental Diesel engine is provided. A comparison of the FJH Model [21] results and the simulated results from the PDESIM is also discussed with the help of a number of engine performance maps generated using the data obtained from the FJH Model and the PEDSIM.

Chapter 8 discusses possible applications of the PDESIM developed in this thesis.

Chapter 9 concludes the work carried out for this thesis, and chapter 10 gives suggestions for the possible future work in the Parallel Diesel Engine Simulation area.

# Chapter 2



# The Diesel Engine Model

## 2.1 Introduction

A model is a mathematical representation of a complicated physical situation. It is formulated by selecting the properties of a system which are important in determining the required information. A model should represent the behaviour of the system itself and the behaviour of its boundaries as well. A Diesel engine is essentially a thermodynamic system. Hence to describe the Diesel engine system behaviour, a thermodynamic model is needed.

There are a number of engine models available that range from semi-empirical matching calculations to comprehensive analysis of the fluid and thermodynamic processes that take place in the engine. The usefulness, accuracy and running costs of these models vary tremendously. These models can be classified in three basic groups[22]:

zero dimensional

quasi dimensional

multi dimensional

The zero dimensional models assume that:

- o a homogeneous mixture of ideal gases exists at all times and at all points within the system; perfect mixing.
- o the instantaneous state of the gaseous mixture can be defined by pressure, temperature and fuel to air ratio; thermodynamic equilibrium.
- o fuel mixes and burns instantaneously as it enters the cylinder.
- o unburnt fuel vapours do not affect the ignition delay or the combustion process.

- o quasi steady state flow occurs between the control volumes.

The quasi dimensional models:

- o account for finite rate effects due to injection, vaporization, mixing and combustion.
- o divide the fuel injected into the combustion chamber into several zones.
- o account for the jet deflection by the surrounding air in the combustion chamber.

The multidimensional models:

- o account for the temporal and spatial variations of the fluid flow, temperature, gas composition, pressure and turbulence within the combustion.
- o consider the injected fuel as droplets sprayed inside the combustion chamber and hence the effects of gas phase turbulence on the liquid droplets.

In this thesis, the zero dimensional models are used in conjunction with 'filling and emptying' method (FAEM) to model the Diesel engine to be simulated. The FAEM views a Diesel engine as a set of interconnected thermodynamic control volume models. Each of the control volumes represents one of the cylinder or manifold volumes of the engine. These control volumes are interconnected by the valve and port models. The turbine and compressor are viewed as connections between the manifolds and the atmosphere with the flow through them being modelled by the compressor and turbine characteristics.

The FAEM can be explained as follows:

- o Air enters into the inlet manifold either via an orifice or via a compressor.
- o Heat transfer occurs between the gas and the walls of the inlet manifold.
- o Gas leaves the inlet manifold via the inlet valves to the cylinder.
  
- o When the inlet valve is open, gas enters the cylinder.
- o Fuel is injected into the cylinder when both, the inlet and exhaust valves are closed.
- o Heat transfer occurs between the cylinder walls and the gas mixture in the cylinder.
- o Useful work is transferred to the shaft via the piston.
- o Gases leave the cylinder when the exhaust valve is open to the exhaust manifold.
  
- o The exhaust manifold receives hot gases from the cylinder when the exhaust valves are open.
- o Heat is transferred from the hot gases to the exhaust manifold walls.
- o Exhaust gases leave the exhaust manifold, either via the orifice or through a turbine.

Figure 2.1 shows how the thermodynamic behaviour of a six cylinder turbocharged Diesel engine is represented using a FAEM.

In the following sections, the state equations representing changes in the control volume mass, the fuel to air ratio and the temperature are discussed alongwith the

submodels that describe different chemical and physical processes within a particular control volume.

## 2.2 The Control Volume Gas State Equations

The state of the gas in a control volume can be represented at any instant of time by its temperature, mass and composition. The three state equations that represent rates of change of these parameters, can be derived by applying laws of conservation of mass, energy and fuel species to the mixture contained in a control volume [21]. and are given below.

### 2.2.1 Rate of change of mass

$$\frac{dm}{dt} = \sum_j \frac{dm_j}{dt} + \frac{dm_{fb}}{dt} \quad 2.1$$

where  $dm/dt$  is the overall rate of change of mass of the gases inside a control volume,  $dm_j/dt$  is the rate of change of mass associated with the  $j$ th port connected to the control volume and  $dm_{fb}/dt$  is the rate of change of fuel burnt in the control volume.

### 2.2.2 Rate of change of fuel to air ratio

$$\frac{d\lambda}{dt} = \frac{(1+\lambda)}{m} \left[ (1+\lambda) \left( \sum_j \frac{\lambda_j}{1+\lambda_j} \frac{dm_j}{dt} + \frac{dm_{fb}}{dt} \right) - \frac{\lambda dm}{dt} \right] \quad 2.2$$

where  $d\lambda/dt$  is the rate of change of fuel to air ratio inside the control volume, and  $(\lambda_j/(1+\lambda_j)dm_j/dt)$  is the fractional change in the fuel to air ratio associated with the  $j$ th port connected to the control volume. The terms containing  $dm_{fb}/dt$  and  $dm/dt$

represent change in the fuel to air ratio caused by the burnt fuel and the change in total mass of the control volume respectively.

### 2.2.3 Rate of change of temperature

$$\frac{dT}{dt} = \frac{1}{m\delta u/\delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} + \sum_j h_{oj} \frac{dm_j}{dt} + h_{calval} \frac{dm_{fb}}{dt} - u \frac{dm}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda}{dt} \right] \quad 2.3$$

where  $dT/dt$  is the overall rate of change of temperature within the control volume. The term containing  $PdV/dt$  represents the useful work done by the control volume gases, and  $dQ_w/dt$  is the loss of energy in the form of heat exchange to the walls of the control volume. The term containing  $h_{oj}dm_j/dt$  represents the amount of enthalpy associated with the mass entering or leaving through a port connected to the control volume and  $h_{calval} \cdot dm_{fb}/dt$  is the representation of the amount of enthalpy entering the control volume due to the fuel burnt inside the system. The remaining terms containing  $u$  represent the specific internal energy of the mixture.

## 2.3 The Cylinder Control Volume Equations

The above state equations are general in nature and can be applied to any control volume. But there are some terms which are not needed during certain stages of the Diesel engine cycle. Which can then be simplified for the six stages of the cylinder cycle: scavenge, induction, compression, combustion, power stroke and exhaust.

All these stages depend upon the position of the crankshaft with respect to a reference. In this thesis, the top dead center (TDC) is taken as a reference. The relationship between crankshaft position and time is defined by equation 2.4.

$$\theta = \int \omega_e dt + \theta_0 \quad 2.4$$

where  $\theta_0$  is the angular position of the crankshaft in radians with respect to the TDC at  $t=0$ , and  $\omega_e$  is the engine speed in radians per second.

The equations for individual sections of the Diesel engine cycle are derived for the case of one inlet and one exhaust valve per cylinder and are given in the following sections.

### 2.3.1 Scavenge

The scavenge period is defined when both the inlet and the exhaust valves are open. During this period no combustion takes place. Hence

$$\frac{dm_{fb}}{dt} = 0 \quad 2.5a$$

$$\frac{dm_{sc}}{dt} = \frac{dm_i}{dt} - \frac{dm_e}{dt} \quad 2.5b$$

$$\frac{d\lambda_{sc}}{dt} = \frac{1+\lambda}{m} \left[ (1+\lambda) \left( \frac{\lambda_i}{1+\lambda_i} \frac{dm_i}{dt} - \frac{\lambda_e}{1+\lambda_e} \frac{dm_e}{dt} \right) - \frac{\lambda dm_{sc}}{dt} \right] \quad 2.5c$$

$$\frac{dT_{sc}}{dt} = \frac{1}{m\delta u/\delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} + h_{oi} \frac{dm_i}{dt} - h_{oe} \frac{dm_e}{dt} - u \frac{dm_{sc}}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda_{sc}}{dt} \right] \quad 2.5d$$

### 2.3.2 Induction

During the induction period, only the inlet valve is open and no combustion takes place. Hence

$$\frac{dm_{fb}}{dt} = 0; \quad \frac{dm_e}{dt} = 0 \quad 2.6a$$

$$\frac{dm_{in}}{dt} = \frac{dm_i}{dt} \quad 2.6b$$

$$\frac{d\lambda_{in}}{dt} = \frac{1+\lambda}{m} \left[ (1+\lambda) \left( \frac{\lambda_i}{1+\lambda_i} \frac{dm_i}{dt} \right) - \frac{\lambda dm_{in}}{dt} \right] \quad 2.6c$$

$$\frac{dT_{in}}{dt} = \frac{1}{m\delta u/\delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} + h_{oi} \frac{dm_i}{dt} - u \frac{dm_{in}}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda_{in}}{dt} \right] \quad 2.6d$$

These equations represent normal conditions when gases flow from the inlet manifold to the cylinder via the inlet valve. But, if at some stage, the pressure in the inlet manifold is less than the pressure in the cylinder during the induction period, then reverse flow occurs. Under these conditions

$$\frac{d\lambda_{in}}{dt} = 0 \quad 2.6e$$

### 2.3.3 Compression and Power

The compression and power strokes are defined when both the inlet and exhaust valves are closed and no combustion occurs. Hence the simplified state equations are

$$\frac{dm_{fb}}{dt} = 0; \quad \frac{dm_i}{dt} = 0; \quad \frac{dm_e}{dt} = 0 \quad 2.7a$$

$$\frac{dm_{cl}}{dt} = 0 \quad 2.7b$$

$$\frac{d\lambda_{cl}}{dt} = 0 \quad 2.7c$$

$$\frac{dT_{cl}}{dt} = \frac{1}{m \cdot \delta u / \delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} \right] \quad 2.7d$$

## 2.2.4 Exhaust

Exhaust period is defined when only the exhaust valve is open. There is no combustion during the exhaust period. Hence the simplified state equations that represent normal exhaust stroke are

$$\frac{dm_{fb}}{dt} = 0; \quad \frac{dm_i}{dt} = 0 \quad 2.8a$$

$$\frac{dm_{ex}}{dt} = \frac{dm_e}{dt} \quad 2.8b$$

$$\frac{d\lambda_{ex}}{dt} = 0 \quad 2.8c$$

$$\frac{dT_{ex}}{dt} = \frac{1}{m \delta u / \delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} - h_{oe} \frac{dm_e}{dt} - u \frac{dm_{ex}}{dt} \right] \quad 2.8d$$

But, if reverse flow occurs during an exhaust stroke due a low pressure inside the cylinder compared with the pressure inside the exhaust manifold, then the rate of change of fuel to air ratio and that of temperature are given as

$$\frac{d\lambda_{ex}}{dt} = \frac{1+\lambda}{m} \left[ (1+\lambda) \left( -\frac{\lambda_e}{1+\lambda_e} \frac{dm_e}{dt} \right) - \frac{\lambda dm_{ex}}{dt} \right] \quad 2.8e$$

$$\frac{dT_{ex}}{dt} = \frac{1}{m \delta u / \delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} + h_{oe} \frac{dm_e}{dt} - u \frac{dm_{ex}}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda_{ex}}{dt} \right] \quad 2.8f$$



### 2.3.5 Combustion

Fuel is injected into the cylinder when both the inlet and the exhaust valves are closed. Fuel is considered to be a liquid before combustion and therefore assumed to have no effect on the gas of the cylinder before combustion. Thus, during combustion period, the state equations become

$$\frac{dm_i}{dt} = 0; \quad \frac{dm_e}{dt} = 0 \quad 2.9a$$

$$\frac{dm_{cb}}{dt} = \frac{dm_{fb}}{dt} \quad 2.9b$$

$$\frac{d\lambda_{cb}}{dt} = \frac{1+\lambda}{m} \frac{dm_{fb}}{dt} \quad 2.9c$$

$$\frac{dT_{cb}}{dt} = \frac{1}{m\delta u/\delta T} \left[ -P \frac{dV}{dt} + \sum_w \frac{dQ_w}{dt} + h_{calval} \frac{dm_{fb}}{dt} - u \frac{dm_{cb}}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda_{cb}}{dt} \right] \quad 2.6d$$

## 2.4 The Manifold Control Volume Equations

The general FAEM state equations may be simplified for the inlet and the exhaust manifold control volumes as no combustion is assumed to occur in any of the manifold control volumes, and also a manifold has a constant volume. Hence

$$\frac{dm_{fb}}{dt} = 0; \quad \frac{dV}{dt} = 0; \quad 2.10$$

## 2.4.1 Inlet Manifold

Neglecting heat transfer to the walls of the inlet manifold, the three state equations that represent an inlet manifold are

$$\frac{dm_{im}}{dt} = \frac{dm_{ia}}{dt} - \sum_i \frac{dm_i}{dt} \quad 2.11a$$

$$\frac{d\lambda_{im}}{dt} = \frac{1+\lambda}{m} \left[ (1+\lambda) \left( \frac{\lambda_{ia}}{1+\lambda_{ia}} \frac{dm_{ia}}{dt} - \sum_i \frac{\lambda_i}{1+\lambda_i} \frac{dm_i}{dt} \right) - \frac{\lambda dm_{im}}{dt} \right] \quad 2.11b$$

$$\frac{dT_{im}}{dt} = \frac{1}{m\delta u/\delta T} \left[ h_{oia} \frac{dm_{ia}}{dt} + \sum_i h_{oi} \frac{dm_i}{dt} - u \frac{dm_{im}}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda_{im}}{dt} \right] \quad 2.11c$$

where the subscript ia represents an entry port to the inlet manifold, which may be either an orifice or a compressor.

## 2.4.2 Exhaust Manifold

$$\frac{dm_{em}}{dt} = \sum_e \frac{dm_e}{dt} - \frac{dm_{ea}}{dt} \quad 2.12a$$

$$\frac{d\lambda_{em}}{dt} = \frac{1+\lambda}{m} \left[ (1+\lambda) \left( - \frac{\lambda_{ea}}{1+\lambda_{ea}} \frac{dm_{ea}}{dt} + \sum_e \frac{\lambda_e}{1+\lambda_e} \frac{dm_e}{dt} \right) - \frac{\lambda dm_{em}}{dt} \right] \quad 2.12b$$

$$\frac{dT_{em}}{dt} = \frac{1}{m\delta u/\delta T} \left[ \sum_w \frac{dQ_w}{dt} - h_{oea} \frac{dm_{ea}}{dt} + \sum_e h_{oe} \frac{dm_e}{dt} - u \frac{dm_{em}}{dt} - m \frac{\delta u}{\delta \lambda} \frac{d\lambda_{em}}{dt} \right] \quad 2.12c$$

where the subscript ea represents an exit port from the exhaust manifold, which may be an orifice or a turbine.

## 2.5 Sub Models

In order to evaluate the state equations, their constituent terms are either calculated using the thermodynamic laws or empirical models derived from the experimental data are used where these laws are not directly applicable. The following sections describe the equations representing the terms in the state equations.

### 2.5.1 Gas properties model

This model calculates the values of the following quantities for use by the engine model in the state equations.

- $u$  the specific internal energy of the gas
- $\delta u / \delta T$  partial derivative of specific internal energy with respect to temperature
- $\delta u / \delta \lambda$  partial derivative of specific internal energy with respect to fuel to air ratio
- $R$  the gas constant
- $\gamma$  ratio of specific heat at constant pressure to the specific heat at constant volume
- $h_o$  specific enthalpy of the gas mixture

The empirical relations used to calculate these quantities are derived by Charlton [15] from the tabulated data for the internal energy and the gas constant for the equilibrium combustion products of a hydrocarbon and air. In this model the gas dissociation effects are ignored and the specific internal energy becomes a function of only the temperature and the gas composition. The gas constant is defined as a function of gas composition only. Hence

$$u = f(T, \lambda) \quad 2.13a$$

$$R = f(\lambda) \quad 2.13b$$

The gas model expressions for the lean fuel to air ratio mixtures are:

$$u = \frac{K_a - K_b \lambda}{1 + \lambda} \quad 2.14a$$

$$K_a = 696T + 0.89465e-3T^2 + 102.512e-6T^3 - 45.52e-9T^4 + 6.22e-12T^5 \quad 2.14b$$

$$K_b = 722.903T + 1.57193T^2 - 523.84e-6T^3 + 116.61e-9T^4 - 12.533e-12T^5 \quad 2.14c$$

$$R = \frac{270.21 + 288.294 \lambda}{1 + \lambda} \quad 2.14d$$

$$\frac{\delta u}{\delta T} = (101.4663 - 2.5223T - 1.055e-3T^2 + 1.7569e-7T^3) \lambda + 920.7128 - 0.2806T - 5.4939e-5T^2 - R \quad 2.14e$$

$$\frac{\delta u}{\delta \lambda} = \frac{26.053T + 1.57104T^2 - 626.352e-6T^3 + 162.1283e-9T^4 - 18.75e-12T}{(1 + \lambda)^2} \quad 2.14f$$

$$\gamma = 1 + \frac{R}{\delta u / \delta T} \quad 2.14g$$

$$h_o = u + R.T \quad 2.14h$$

## 2.5.2 The heat release models

Combustion is one of the most complex processes carried out in a Diesel engine and is responsible for the useful work output from the engine. It may be considered to commence when fuel is dynamically injected into the cylinder. It comprises of an ignition delay followed by a period of heat release.

Once ignition takes place, the combustion mechanisms accelerate, resulting in a rapid rate of pressure rise. The first appearance of a flame following ignition is of the non-luminous, pre-mixed type. The highest rate of heat release is normally associated with this relatively short period of uncontrolled burning.

At the high temperatures corresponding to the main period of combustion, which follows the pre-mixed burning stage, the burning rate is controlled by the mixing pattern, which is dependent on purely mechanical and physical processes. This stage is called the diffusion burning stage, subsequent to which the diffusivity of the fuel becomes low and the rate of heat release becomes dependent upon chemical kinetics of the mixture and the amount of oxygen left in the combustion chamber. All these four stages of chemical combustion are shown in figure 2.2.

In the following sections, along with a model for ignition delay, two different models are represented to calculate combustion processes in the Diesel engine.

It should be noted that in the simulations developed by Jones [20] and Haysom [21], a fixed combustion model was used which could not be exchanged with any other model. But in the simulation presented in this thesis two different combustion models are given, either of which can be selected at the start of the simulation as explained in chapter 5.

### **2.5.2.1 The Ignition delay model**

The fuel when injected into the engine cylinder, does not ignite instantaneously. The time interval between the start of fuel injection and the start of its ignition is termed as ignition delay. The duration of this period greatly affects the intensity of the

subsequent burning, hence the rate of pressure rise, maximum cylinder pressure and the indicated mean effective pressure [23].

Wolfer [24] derived a semi-empirical relationship to calculate the ignition delay by considering a number of factors, such as:

amount of air

amount of fuel

shape of combustion chamber

shape of fuel nozzle

injection pressure

turbulence and mixing inside the combustion chamber

fuel temperature

He found that the ignition delay was a function of the mean temperature and pressure during the ignition delay period. This Wolfer ignition delay relation, given in equation 2.15 was used to calculate the ignition delay with coefficients suggested by Watson [25].

$$t_{igd} = \frac{453.4638 e^{2100/T_m}}{P_m^{1.022}} \quad 2.15$$

### 2.5.2.2 Watson heat release model

The first model used to describe the fuel burning rate is due to Watson et al [26]. It assumes that

- o the combustion consists of two main parts, the pre-mixed and the diffusion combustion.

- o both these parts start at the same time at ignition.
- o the overall combustion correlation is due to the individual contributions from these two parts.

Equation 2.16a represents the pre-mixed combustion whereas equation 2.16b represents the diffusion combustion.

$$FBR_p(\tau) = 1 - (1 - \tau^{C_{p1}})^{C_{p2}} \quad 2.16a$$

$$FBR_d(\tau) = 1 - \exp(-C_{d1} \tau^{C_{d2}}) \quad 2.16b$$

The overall combustion correlation is:

$$FBR(\tau) = \beta FBR_p(\tau) - (1 - \beta) FBR_d(\tau) \quad 2.17$$

where  $\beta$  is defined as the 'mode of burning' which expresses the cumulative fuel burnt during pre-mixed combustion as a fraction of the total fuel injected. The constants  $C_{p1}$ ,  $C_{p2}$ ,  $C_{d1}$ , and  $C_{d2}$  are called the shape factors.

Equation 2.17 gives the rate of fuel burning as the sum of two weighted distributions which are non-dimensionalised with respect to the amount of fuel injected into the cylinder and the nominal time of the combustion.

The shape factors,  $C_{p1}$ ,  $C_{p2}$ ,  $C_{d1}$  and  $C_{d2}$ , and the mode of burning,  $\beta$ , are dependent on the engine operating conditions. For the purpose of this model these values were taken from Watson [25] and are given in equation 2.18 as

$$C_{p1} = 2.0 + 44.549 (t_{igd} \omega_e)^{2.4} \quad 2.18a$$

$$C_{p2} = 5000.0 \quad 2.18b$$

$$C_{d1} = 2.5463 \lambda^{-0.644} \quad 2.18c$$

$$C_{d2} = \frac{1.0 - 0.4125 \lambda^{0.37}}{t_{igd}^{0.26}} \quad 2.18d$$

If ' $mf_{inj}$ ' is the amount of fuel injected during a combustion period ' $\Delta_{comb}$ ' then the rate of fuel burning may be given by equation 2.19 in the form

$$\frac{dm_{fb}}{d\theta} = \frac{FBR(\tau) mf_{inj}}{\Delta_{comb}} \quad 2.19$$

where 
$$\tau = \frac{\theta - \theta_i}{\theta_e - \theta_i} = \frac{\theta - \theta_i}{\Delta_{comb}}$$

and  $\theta_i$  = angle with respect to TDC at which ignition starts.

$\theta_e$  = angle with respect to TDC at which ignition ends.

### 2.5.2.2 Wiebe heat release model

The second heat release model is based on the Wiebe function [25], which is a simple algebraic formula for the fuel burnt as a fraction of the total fuel injected. It is given in equation 2.20.

$$FB(\tau) = 1 - \exp(-K2 \tau^{K1+1}) \quad 2.20$$

Differentiating with respect to  $\tau$  gives

$$\frac{dFB(\tau)}{d\tau} = K2 (K1 + 1) \tau^{K1} \exp(-K2 \tau^{K1+1}) \quad 2.21$$



where FB is the fraction of fuel burnt per total fuel injected per cycle per cylinder, FBR is the non-dimensional fuel burning rate, K1 is the shape factor and K2 is the combustion factor.

### 2.5.3 The Heat Transfer Models

Heat is transferred from the gas in a control volume by convection at the control volume walls and by the radiation from flame and luminous carbon particles. Heat entering the walls is passed through the coolant with forced convection. As the forced convective heat transfer at the gas-to-wall interface has a more important impact on the heat transfer than conduction through the wall or transfer from wall to coolant, the heat transfer due to conduction is assumed to be negligible. Also, as the significance of the contribution of radiation may vary between 0% to 30% [27], its effects on the overall heat transfer are ignored in this thesis.

Hence a convective, gas-to-wall heat transfer model may be defined by the equation 2.22.

$$\frac{dQ}{dt} = SA htc (T_g - T_w) \quad 2.22$$

where  $dQ/dt$  is the rate of heat transfer across the surface area SA.  $T_g$  and  $T_w$  are the temperatures of the control volume gas and wall respectively and the proportionality constant 'htc' is known as the heat transfer coefficient.

The surface area, SA, of the control volume is calculated from the knowledge of the geometry of the control volume. In case of a cylinder, shown in figure 2.3, it is given by equation 2.23, whereas it is a constant quantity for a manifold.

$$SA_{cyl} = SA_{TDC} + \pi d(1 + r(1 - \cos\theta) - \sqrt{1 - r^2 \sin^2\theta}) \quad 2.23$$

For a manifold, the htc is taken as a constant quantity [20]. But in case of a cylinder, as the heat transfer primarily depends on the flow conditions within the cylinder which are difficult to predict, the calculation of htc is not a linear problem. In the following sections two different empirical methods are described to calculate the value of the htc.

### 2.5.3.1 Hohenberg heat transfer model

Hohenberg [28] derived a semiempirical heat transfer model assuming only the convective heat transfer conditions. The htc due to Hohenberg, given in equation 2.34, depends upon the cylinder temperature, pressure, volume and the mean piston speed. The two empirical constants,  $C_{h1}$  and  $C_{h2}$ , depend upon the operating conditions inside the cylinder and ideally should be determined from the heat transfer measurements on the engine being modelled.

$$htc = C_{h1} P^{0.8} (v_{pis} + C_{h2})^{0.8} V^{-0.06} T^{-0.4} \quad 2.24$$

where  $v_{pis} = 2r\omega_e/\pi$

### 2.5.3.2 Woschni heat transfer model

The heat transfer model derived by Woschni [29] is based on the experiments on motored and fired engines and a constant volume bomb. It describes two different

heat transfer coefficients; one for the scavenge period and the other for the rest of the cycle. Equation 25 gives the hts as defined by Woschni.

$$htc = C_{w1} P^{0.8} d^{-0.2} T^{-0.53} [C_{w2} v_{pis} + C_{w3} C_{w4} C_{w5}]^{0.8} \quad 2.25a$$

$$C_{w1} = 12.99 \cdot 10^{-3} \quad 2.25b$$

$$C_{w2} = 6.18 \quad (\text{during scavenge}) \quad 2.25c$$

$$C_{w2} = 2.28 \quad (\text{during compression and expansion}) \quad 2.25d$$

$$C_{w3} = 3.24 \cdot 10^{-10} \text{ m/sec } ^\circ\text{C} \quad 2.25e$$

$$C_{w4} = V_{total} T_{at\_inj} / (V_{at\_inj} P_{at\_inj}) \quad 2.25f$$

$$C_{w5} = P - P_{motoring} \quad 2.25g$$

where  $P_{motoring}$  is the cylinder pressure corresponding to the current volume under motoring conditions.

### 2.5.3.3 Control volume surface temperature

A simple model is used to calculate the surface temperature of the wall of the control volume. Figure 2.4 shows the model. The thermal resistance,  $R_{gw}$ , between the gas and the wall is calculated from the heat transfer coefficient, and is given in equation 2.26.

$$R_{gw} = (SA htc)^{-1} \quad 2.26$$

The thermal resistance  $R_{wc}$  and the thermal capacitance  $C_{wc}$ , which represent the heat transfer model for the heat flow from the control volume wall surface to the engine coolant, can be estimated by having a knowledge of the bulk properties of the wall of the control volume. From this model, the rate of change of wall temperature is given by

$$\frac{dT_{sa}}{dt} = \frac{1}{C} \frac{dQ}{dt} - \frac{T_g - T_w}{C_{wc} R_{wc}} \quad 2.27$$

## 2.5.4 Cylinder volume and rate of change of volume

The cylinder trapped volume is a function of the crank angle and is calculated using the information about the cylinder geometry as shown in figure 2.3. Equation 2.28 gives the cylinder trapped volume.

$$V = 0.25\pi d^2 [crl + r(1 - \cos\theta) - \text{sqrt}(crl^2 - r^2 \sin^2\theta) + 2r / (cr - 1)] \quad 2.28$$

where 'crl' is the connecting rod length and 'cr' is the compression ratio of the cylinder.

Differentiating equation 2.28 with respect to t yields the rate of change of volume and is given in equation 2.29.

$$\frac{dV}{dt} = 0.25\pi d^2 \left[ r \sin\theta \frac{d\theta}{dt} + \frac{r^2 \sin\theta \cos\theta}{\text{sqrt}(crl^2 - r^2 \sin^2\theta)} \frac{d\theta}{dt} \right] \quad 2.29a$$

$$\frac{dV}{dt} = 0.25\pi d^2 \omega_e \left[ r \sin\theta + \frac{r^2 \sin\theta \cos\theta}{\text{sqrt}(crl^2 - r^2 \sin^2\theta)} \right] \quad 2.29b$$

where  $d\theta/dt = \omega_e =$  engine speed in radians per second.

## 2.5.5 Junction flow models

In this thesis, three types of junctions are modelled; a poppet valve, a compressor with an intercooler and a turbine. The models representing flow through these junctions are discussed in the following sections.

### 2.5.5.1 The poppet valve

A poppet valve connects a cylinder control volume to a manifold control volume. The rate of mass flow through the valve is a function of the pressure ratio across the valve, the effective cross-sectional area of the valve throat, local geometrical effects, heat transfer, flow separation and other secondary effects. In this thesis, the mass flow through the valves is considered to be quasi-steady, one-dimensional and isentropic. These assumptions highly simplify the resultant model. Some account of secondary flow effects is taken by introducing an empirical quantity called the discharge coefficient into the model.

Two different types of flow occur through the valves; supersonic and subsonic. These depend upon the pressure,  $P_r$ , ratio across the junction, defined by equation 2.30.

$$P_r = \frac{P_{\text{upstream}}}{P_{\text{downstream}}} \quad 2.30$$

The pressure ratio at which the transition between the two flow types occurs is known as the critical pressure ratio, and is given by equation 2.31.

$$P_{\text{crit}} = ((\gamma + 1)/2)^{\gamma/(\gamma-1)} \quad 2.31$$

where  $\gamma$  is the ratio of specific heats of the upstream gas mixture.

If the pressure ratio  $P_r$  is less than the critical value, then the flow is subsonic and is calculated using equation 2.32a. Otherwise the flow is supersonic and is calculated using equation 2.32b.

$$\frac{dm_{\text{sub}}}{dt} = C_d A P_u \text{ sqrt} \left[ \frac{2 \gamma_u}{R_u T_u (\gamma_u - 1)} \left( \frac{1}{P_r^{(2/\gamma_u)}} - \frac{1}{P_r^{(\gamma_u - 1)/\gamma_u}} \right) \right] \quad 2.32a$$

$$\frac{dm_{\text{sup}}}{dt} = C_d A P_u \text{ sqrt} \left[ \frac{\gamma_u}{R_u T_u} \left( \frac{2}{(\gamma_u + 1)} \right)^{(\gamma_u + 1)/(\gamma_u - 1)} \right] \quad 2.32b$$

Here  $C_d$  is the discharge coefficient and the product  $C_d A$  is the effective cross-sectional area of the valve.

The general valve area  $A$  is given to the engine model in tabular form as a function of the crankangle. A constant area valve can be simulated by providing a constant area for all the possible values of the crankangle.

### 2.5.5.2 The turbocharger

In this thesis, a turbocharger consisting of a conventional centrifugal compressor and a fixed geometry turbine, is modelled. The compressor is connected to an intercooler which cools the compressed air before it enters the inlet manifold.

The model evaluates the rate of mass flow through the compressor and turbine units. It also calculates the torque developed by the turbine and that required to drive the compressor via the turbocharger shaft.

As quasi-steady flow of gas is assumed through the compressor and turbine junctions, the flow and efficiency characteristic maps of the compressor and the turbine based on their steady flow performance data, obtained experimentally or provided by the manufacturer, are used to interpolate the required mass flow parameters.

Compressor and turbine models are discussed below.

### The compressor

The steady state performance map of a compressor manufactured by Garrett is shown in figure 2.5. The mass flow rate through the compressor and the compressor efficiency can be calculated by interpolating this map, provided the pressure ratio across the compressor, the rotational speed of the turbocharger and the upstream temperature and pressure known.

$$\frac{dm_{\text{compressor}}}{dt} = \text{CompMassFlowParameter} \frac{P_{\text{upstream}}}{\text{sqrt}(T_{\text{upstream}})} \quad 2.33$$

Once the compressor efficiency is known, the exit temperature can be calculated using equation 2.34.

$$T_d = T_u + \frac{T_u}{\eta_c} \left[ \left( \frac{P_d}{P_u} \right)^{(\gamma-1)/\gamma} - 1 \right] \quad 2.34$$

Equation 2.35 gives the torque required to drive the compressor.

$$\tau_c = \frac{dm_c (h_{0in} - h_{0out})}{dt \omega_{tc}} \quad 2.35$$

where  $h_{o_{in}} - h_{o_{out}}$  is the change in enthalpy of gas after passing through the compressor.

### The intercooler

The air from the compressor passes through an intercooler before entering the inlet manifold, which absorbs heat from the compressed air and hence increases the air density. The efficiency of the intercooler is defined as the ratio between the actual temperature drop across the cooler to the theoretical pressure drop due to the cooling water. Equation 2.36 gives the value of this effectiveness.

$$\eta_{ic} = \frac{T_{in} - T_{out}}{T_{in} - T_{coolant}} \quad 2.36$$

From this equation the exit temperature can be calculated as given in equation 2.37.

$$T_{out} = (1 - \eta_{ic}) T_{in} + \eta_{ic} T_{coolant} \quad 2.37$$

There is also a pressure drop associated with the intercooler due to the resistance offered by the cooling surface. Due to this pressure drop  $\Delta P_{ic}$ , the pressure ratio across the intercooler is given by equation 2.38.

$$P_{ric} = \frac{P_{outic} + \Delta P_{ic}}{P_{inic}} \quad 2.38$$

The total pressure ratio across the combination of the compressor and the intercooler used to calculate compressor mass flow is given by equation 2.39.

$$P_r = \frac{P_{outic} - \Delta P_{ic}}{P_{incomp}} \quad 2.39$$



$$= \frac{P_{im} - \Delta P_{ic}}{P_{amb}}$$

In this research, a value of 0.9 for the intercooler effectiveness and a value of 10000 N/m<sup>2</sup> for the pressure drop across the intercooler was used [21].

### The turbine

The model of the turbine is divided into two imaginary parts, each of which is connected to an exhaust manifold. This assumption is made in order to model the turbocharger of the experimental Leyland TL11 Diesel engine which has two exhaust manifolds and a multi entry turbine.

Both imaginary turbines are constrained to rotate at the same speed, and their mass flow performances are scaled so that their combined mass flow is equal to that of the real turbine.

Figure 2.6 shows the map of Garrett turbine. The rate of mass flow through the turbine can be calculated by interpolating this map and then using equation 2.40.

$$\frac{dm_{turbine}}{dt} = \text{TurbMassFlowParameter} \frac{P_{upstream}}{\text{sqrt}(T_{upstream})} \quad 2.40$$

The exit temperature is calculated using equation 2.41 by using the interpolated turbine efficiency.

$$T_d = T_u - \eta_t T_u \left[ 1 - \left( \frac{P_d}{P_u} \right)^{(\gamma-1)/\gamma} \right] \quad 2.41$$

The resultant torque developed by the turbine is the sum of the individual turbine torques produced by each imaginary turbine. It is given by equation 2.42.

$$\tau_t = \sum_j \frac{dm_{tj}}{dt} \left( \frac{h_{oinj} - h_{ooutj}}{\omega_{tc}} \right) \quad 2.42$$

where  $h_{oinj} - h_{ooutj}$  is the change in enthalpy of gas after passing through the  $j$ th turbine.

## 2.6 The Engine Dynamics

The model described above for combustion, heat transfer, cylinder geometry and junction flows represent steady state engine operation. But if dynamic models that describe parameters such as shaft speed, mass of fuel injected and fuel injection timing are used, then the dynamic response of the diesel engine can be modelled using the FAEM. These dynamic models are described below.

### 2.6.1 The engine crankshaft

The engine crankshaft connects all the cylinders together and is responsible for the transfer of the power, generated by the engine, to the outside world. The acceleration of the crankshaft depends upon the torque produced by the cylinders, the demanded torque from the load on the engine, and the inertia of the crankshaft and that of the load. Equation 2.43a represents the engine acceleration as a function of these parameters, whereas equation 2.43b gives the engine speed calculated using acceleration.

$$\frac{d\omega}{dt} = \frac{\Sigma \tau_{cyl,brake} - \tau_{load}}{J_{eng} - J_{load}} \quad 2.43a$$

$$\omega_e = \frac{d\omega_e}{dt} \quad 2.43b$$

The indicated cylinder torque that is a function of cylinder pressure, is given by equation 2.44.

$$\tau_{cyl,ind} = (P_{cyl} - P_{crankcase}) \frac{dV_{cyl}}{dt} \omega_e \quad 2.44$$

Some of this torque is lost in order to overcome the primary friction which is present between the sliding surfaces of the cylinder and the crankshaft bearings, and some torque is used up by secondary friction which is caused by driving engine auxiliaries such as water and oil pumps.

Chen and Flynn [30] have proposed a friction model for a turbocharged Diesel engine. It is based on the mean piston speed and the maximum cylinder pressure for each engine cylinder and provides the friction mean effective pressure as given in eq 2.45.

$$f_{mep} = C_1 + C_2 P_{max} + C_3 v_{pis} \quad 2.45$$

The constants C1, C2 and C3, which should ideally be obtained from motoring experiments, were used, for this research, as proposed by Chen and Flynn.

$$C_1 = 13.7 \quad C_2 = 0.005 \quad C_3 = 16.2$$

Hence the brake cylinder torque generated by each cylinder is given by equation 2.46.

$$\tau_{\text{cyl,brake}} = (P_{\text{cyl}} - P_{\text{crankcase}}) \frac{dV_{\text{cyl}}}{dt} \omega_e - f_{\text{mep}} \frac{dV_{\text{cyl}}}{dt} \omega_e \quad 2.46$$

Another fixed f<sub>mep</sub> model is also provided which reads a fixed value of f<sub>mep</sub> from the data file and stops dynamic calculation of f<sub>mep</sub>.

## 2.6.2. The turbocharger shaft

This shaft connects the turbine to the compressor and transfers power generated by the turbine to the compressor. It is assumed that the friction offered to the turbocharger rotor is proportional to the turbine torque. Hence the resultant torque that accelerates the turbo rotor is a function of the turbocharger efficiency,  $\eta_{\text{tc}}$ . Therefore the turbocharger acceleration is given by:

$$\frac{d\omega_{\text{tc}}}{dt} = \frac{\eta_{\text{tc}} \Sigma \tau_t - \tau_c}{J_{\text{tc}}} \quad 2.47$$

where  $\eta_{\text{tc}} \Sigma \tau_t$  is the friction adjusted torque produced by all the imaginary turbines and  $\tau_c$  is the torque required to drive the compressor.  $J_{\text{tc}}$  is the inertia of the turbocharger.

## 2.6.3 The control actuators

Three different control inputs were modelled by Jones [20]:

- o to control the fuel rack position,
- o to get time of fuel injection,
- o to define the turbine nozzle restriction for a variable entry turbine.

The third control input is not considered in this research as a constant geometry turbine fixed to the TL11 Diesel engine which was used as a test engine for this research. The first two actuators are discussed below.

### 2.6.3.1 Rack position

Jones found that the hydraulic actuator dynamics were predominantly second order in nature. The general equations that describe the engine actuators are given in equation 2.48.

$$\begin{aligned}
 &\text{if } (x_{\text{actuator}} < x_{\text{slew}}) \\
 &\frac{x_{\text{actuator}}}{x_{\text{demand}}} = \frac{k \omega_n^2}{s^2 + 2 \zeta \omega_n s + \omega_n^2} \\
 &\text{else} \\
 &x_{\text{actuator}} = x_{\text{demand}} \qquad \qquad \qquad 2.48
 \end{aligned}$$

where  $x$  is the displacement from the normal zero position and  $\dot{x}$  is the velocity at which the actuator will move under a disturbance. Table 2.1 gives the values used in these equations.

Table 2.1: The control actuator model constants

Actuator	Gain (k)	Damping ratio ( $\zeta$ )	Natural frequency ( $\omega_n$ , rad/sec)	Slew rate (m/sec)
Fuel rack	0.98	0.55	174	0.1
Fuel injector	1.00	0.70	125	0.2

### 2.6.3.2 Injection timing

The static angle  $\theta_s$  at which fuel is injected, is determined from the actuator dynamics, whereas the dynamic angle at which fuel actually enters the cylinder is determined by considering the length of the fuel pipe, engine speed and the speed of fuel pressure wave which travels at the speed of sound. The dynamics injection angle is given by equation 2.49.

$$\theta_d = \theta_s + l \omega_c / C \quad 2.49$$

Once the dynamic injection angle is known the amount of fuel injected into the cylinder at that angle may be found by interpolating the fuel injector characteristic map shown in figure 2.7.

## 2.7 Summary

In this chapter the state equations that represent the thermodynamic engine processes are described. These state equations are based on a 'filling and emptying' method, which considers a Diesel engine as a thermodynamic system consisting of cylinder and manifold control volumes, junctions that interconnect these control volumes, and shafts that transfer power generated by the engine. Hence, the parallelism is achieved by solving each control volume as a separate task. The next chapter describes the hardware features of a parallel computer on which the above Diesel engine model is implemented.

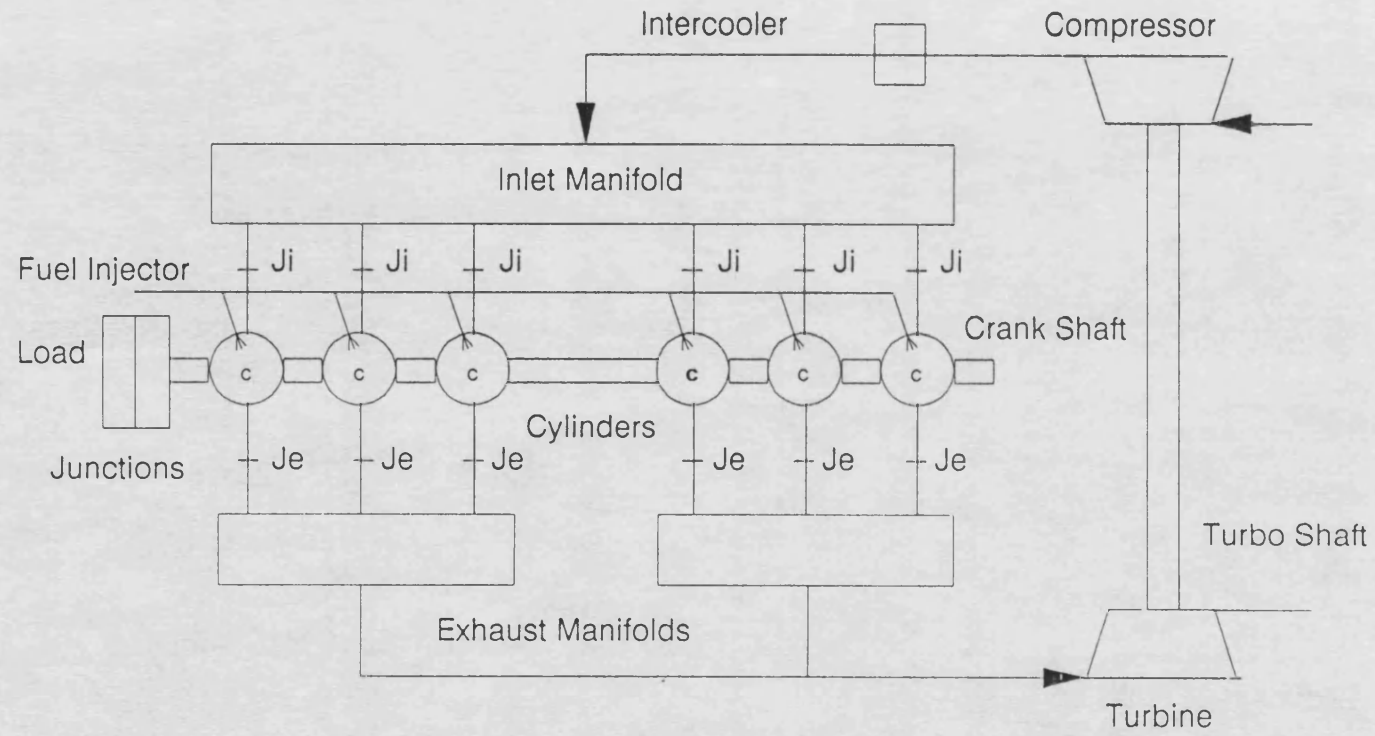


Figure 2.1 Filling and emptying model representation of a six cylinder Diesel engine

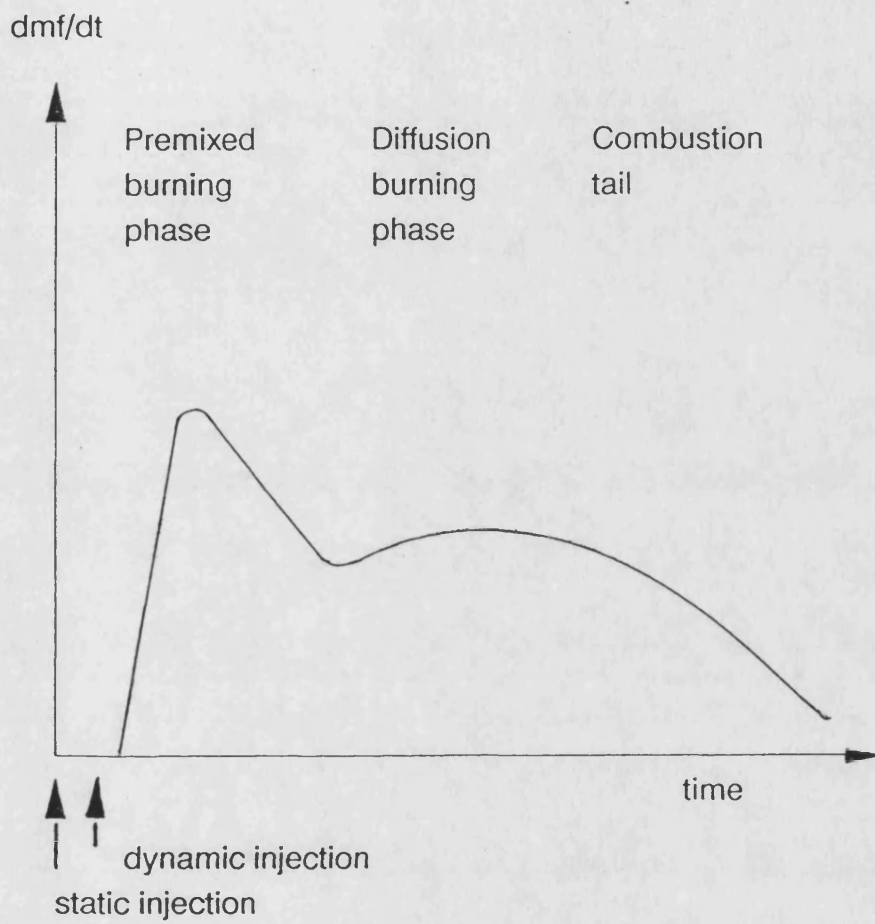


Figure 2.2 The cylinder combustion phases



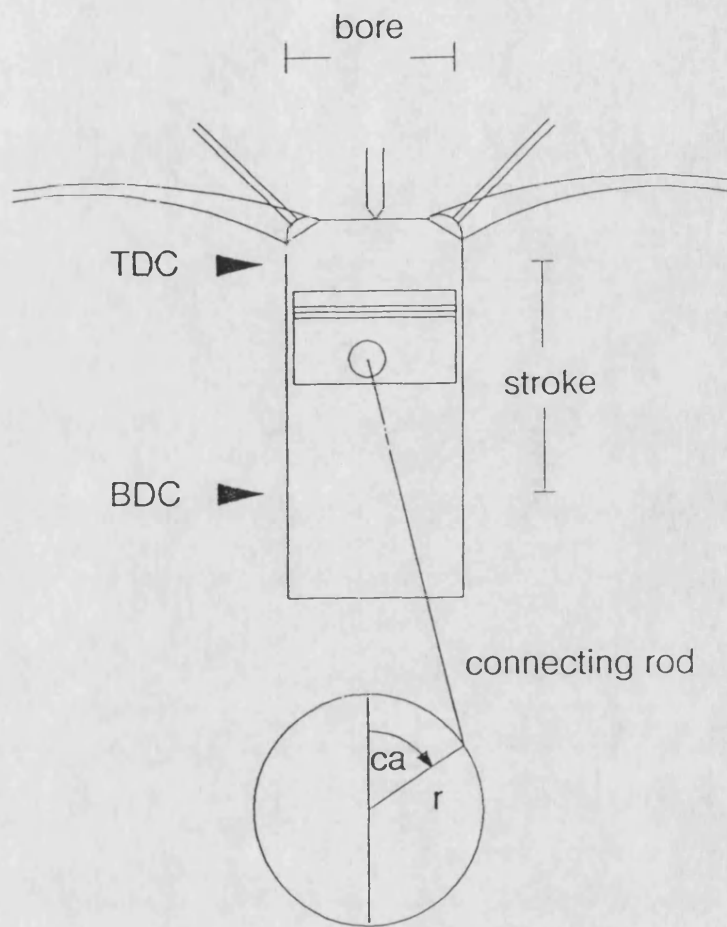
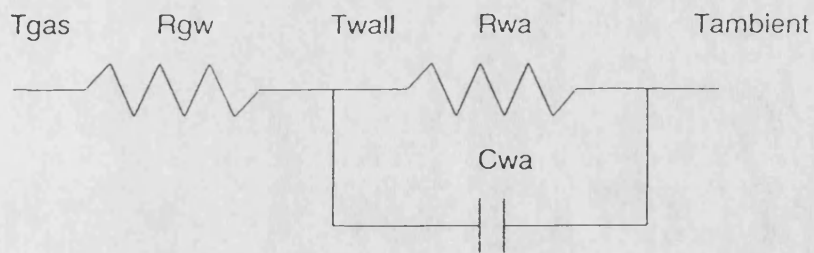


Figure 2.3 Cylinder geometry



$R_{\text{gw}}$  = Thermal resistance between control volume gas contents and wall

$R_{\text{wa}}$  = Thermal resistance between wall and atmosphere surrounding it

$C_{\text{wa}}$  = Thermal capacitance between wall and atmosphere surrounding it

Figure 2.4 A control volume heat transfer model

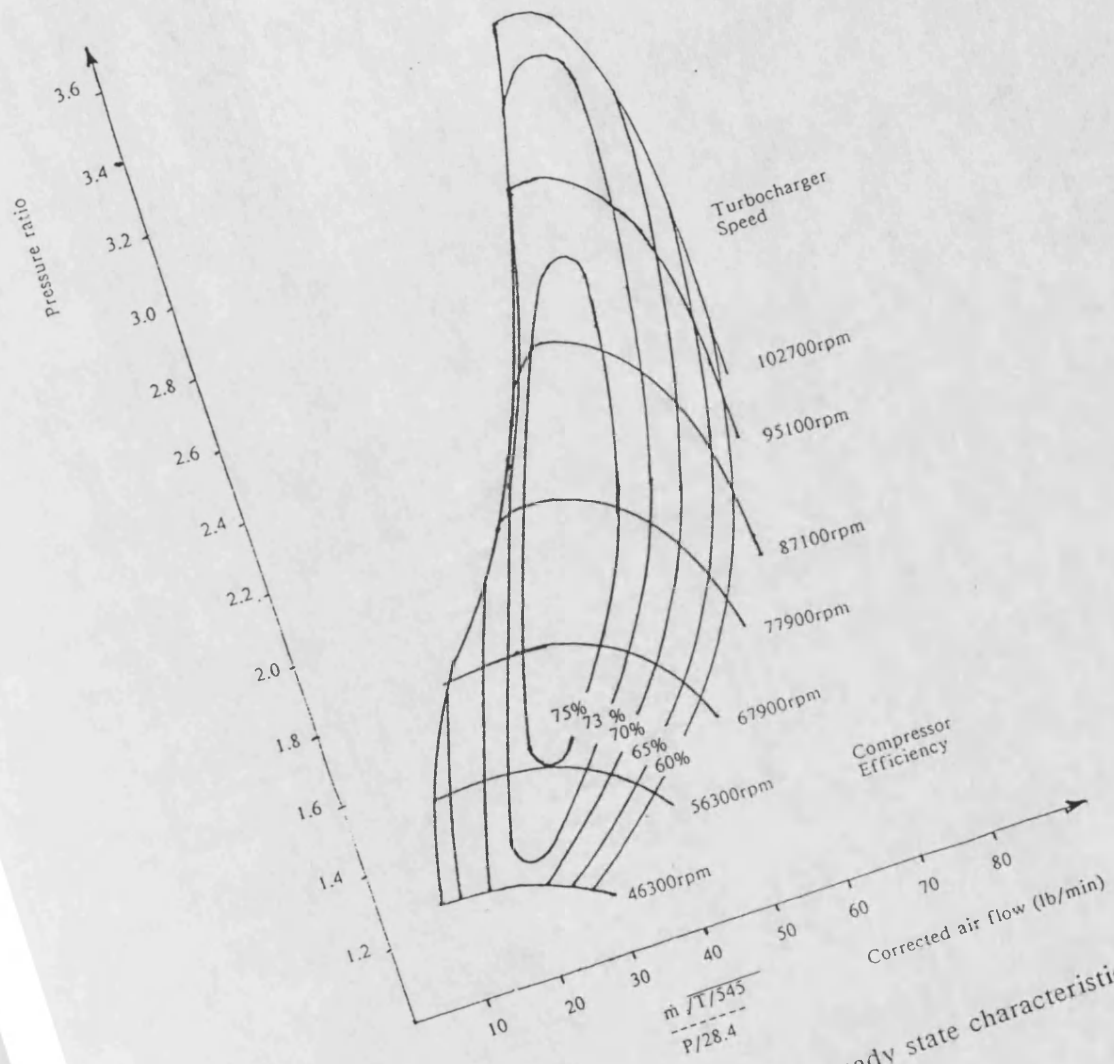


Figure 2.5 The Compressor steady state characteristic map

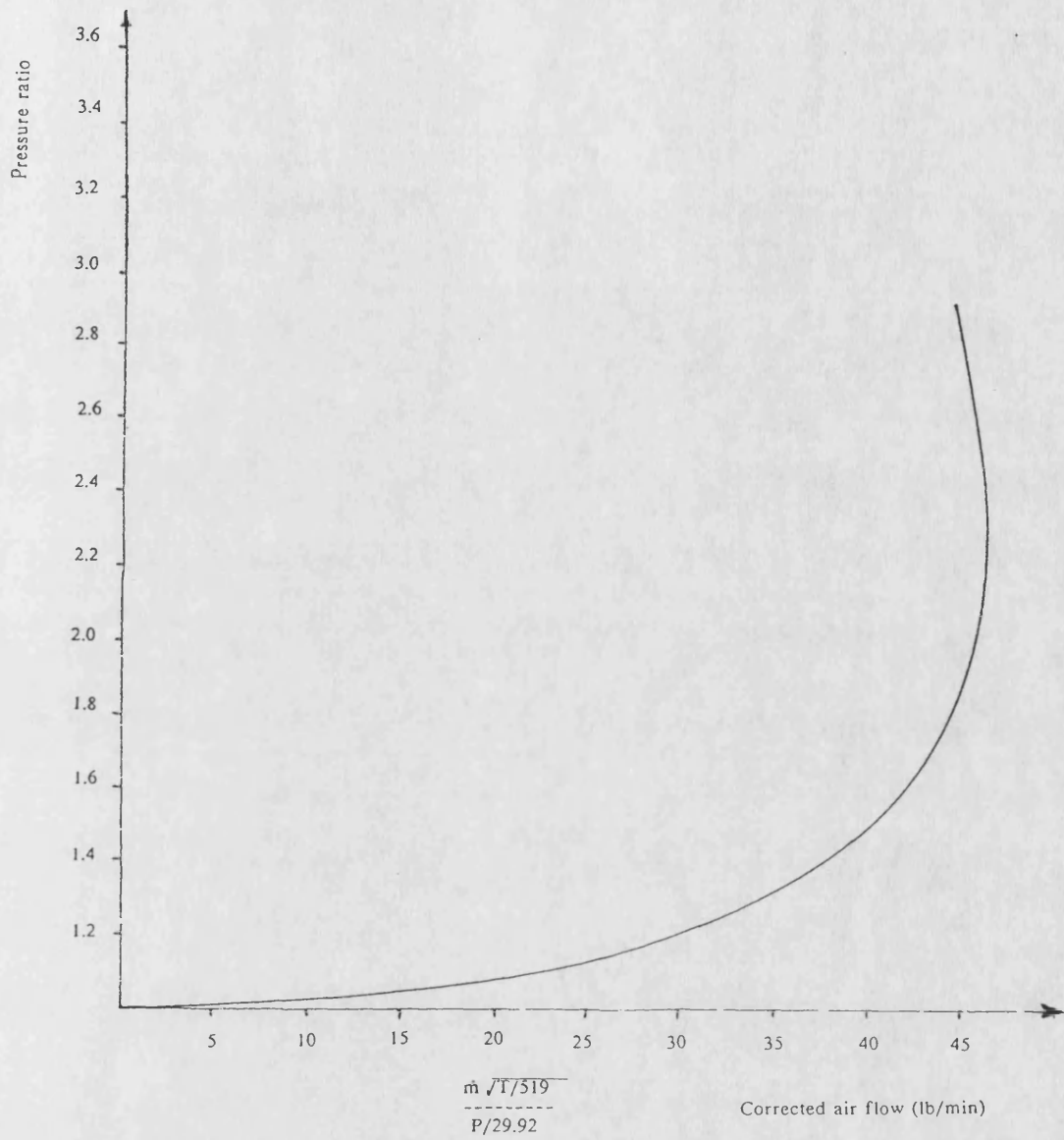


Figure 2.6 The Garrett turbocharger swallowing curve for matched flow

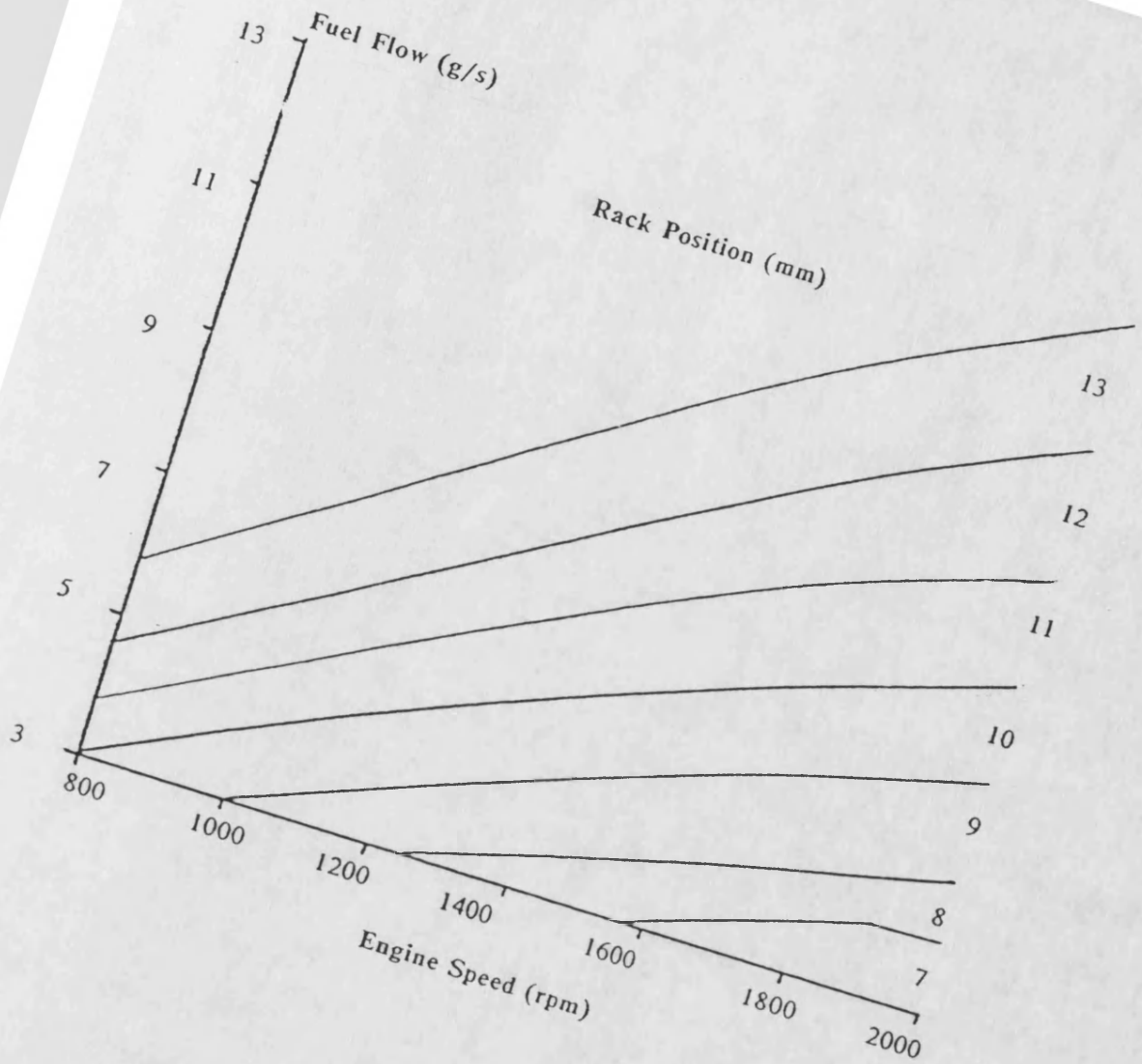


Figure 2.7 Steady state performance characteristic map for the fuel delivery system

# Chapter 3

# The Bath University Transputer Based Parallel Computer (BUTPC)

## 3.1 The BUTPC for Diesel Engine Simulation:

To achieve higher throughput for the diesel engine simulation, a new parallel computer was needed. The options available were:

- redesign the existing Motorola 68020 based processing nodes [31] and the backplane using new high speed components.
- use an upgraded version of the same family of processors, for example a Motorola MC68030.
- use the Intel 80286 processor available at that time.
- use the INMOS transputer and redesign the new parallel computer around it.

The choice of modifying the existing MC68020 processing node or going for the MC68030 did not promise sufficiently higher throughput. Similarly Intel's 80286 in conjunction with a mathematics coprocessor gave lower performance figures compared with that of the transputer. Hence it was decided to build a transputer based parallel computer for the diesel engine simulation.

Now the question was the selection of the communication mechanism for the new BUTPC. The main means of communication available in a transputer system are the transputer links. Although an INMOS T800 transputer has sufficient power, for example a T800-30 running at 30 Mhz clock speed may achieve floating point performance of 2.25 Mflops (for 64 bit operations) and a throughput of possibly 15 Mips [32], it was the communication bandwidth available from the transputer links that was not sufficient to allow efficient use of this processing power for the diesel engine simulation [33, 34]. The other option was to use shared memory. For this

purpose, a fast backplane multiprocessor bus was designed along with the transputer link bus, hence providing a more flexible parallel computer.

Having decided that a T800 transputer based parallel computer with both links and a shared memory architecture was required for the diesel engine simulation, the next question was whether to buy a commercially available system such as the Meiko computing system [35] or buy plug in boards and use a standard host system such as a PC or build our own system at the Bath University. The first two options meant higher costs in terms of initial installation and maintenance, particularly if a shared memory architecture was used. It was, therefore, necessary to build another in-house parallel machine based on the Inmos T800 transputer. A block diagram of this machine is given in figure 3.1.

## **3.2 The T800 Processing Node**

The BUTPC is housed in several 19 inch racks and has sixteen transputer based processing nodes per rack. Each processing node consists of an INMOS T800 transputer, one megabyte of dynamic ram, a multiprocessor bus interface, local and multiprocessor bus arbiter logic and high speed line drivers to connect the transputer links to other boards via the backplane.

A transputer is a single VLSI device with on chip memory, a central processing unit and communication links for direct connections to other transputers [36]. The T800 used in the BUTPC has a 32 bit bus, 4 Kbytes of on chip memory and a 64 bit floating point processor. and four direct memory access (DMA) controlled serial links. Figure 3.2 shows a block diagram of the T800 transputer.

The transputer exploits fast on chip memory by having only a small number of registers and simple instructions. Only six registers are used in the execution of a



sequential process as shown in figure 3.3. The A, B and C registers form an evaluation stack and are the sources and destinations of most of the arithmetic and logical operations. Loading a value into the stack pushes B into C and A into B, before loading A. Storing a value from A, pops B into A and C into B. The workspace register points to an area of store where local variables are kept. The instruction pointer points to the next instruction to be executed and the operand register is used in the formation of instruction operands.

The T800 uses multiplexed address and data signals on its 32 bit memory bus. A built in memory controller provides DRAM control and refresh timing. The internal processor speed is link selectable and is generated by a 5 Mhz external clock.

The T800 transputer can be booted either from a communication link or from a ROM. All the processing nodes in the BUTPC are booted via links. A block diagram of the T800 processing node is shown in figure 3.4. Figure 3.5 shows the T800 processing board itself.

Each T800 processing node consists of two different communication paths. The T800 links are directly available for data communication between adjacent transputers and connected to the backplane link bus, whereas the 32 bit transputer bus is connected to the shared memory bus via bus transceivers.

There are three main types of memory access cycles:

- i) Local memory access by the on-board T800.
- ii) Off-board memory access by the on board T800.
- iii) Local memory access by an off-board T800.

When an on-board memory access cycle is initiated by the local T800, the local decoder signals the local arbiter which enables the local access buffers and disables

the external to local cycle buffers so that during an internal memory access no external transputer should access the local memory.

But if the access cycle is for off-board memory, then the local arbiter disables the local access buffers, so that, if any off-board transputer wants to access the local memory, it could do so. The local decoder hands over the control to the bus arbiter which looks at the bus arbitration signals, and if the bus is available, enables the local to external cycle buffers so that the local to external cycle can be completed. The bus arbitration is explained later in this chapter.

The third type of memory cycle consists of a memory request from an off-board transputer for the local memory. The local arbiter waits if the local memory access is in progress, and then disables the local access buffers while allowing the external to local cycle buffers to put the external data on the local memory bus.

The memory map for the T800 processing node is shown in figure 3.6. It extends from 00000000H to ffffffffH. A 1 Mbyte slot is used by the 'Helios' operating system [37], out of which the 4 Kbytes is the on chip memory. Rest of the 1 Mbyte is paged onto the memory page starting at 80000000H and ending at 800fffffH, because the T800 recognizes the memory address 80000000H as its base address [32]. The 4 Kbytes from 80000000H to 80000fffH is not known to the Helios and can safely be used outside the Helios environment.

The top 4 Kbytes of the transputer 1 Mbyte memory page contain special purpose registers. These are:

- . Boot location (BOOTLOC)
- . Control and status register (CSR)
- . Test and Set semaphore (TAS)
- . Processor number (PNUM)

- . Event register (EVENTREG)
- . Semaphore table pointer register (SEMTAB)

The boot location is the register that is used to provide the jump location if a transputer is to be bootstrapped from ROM. If the BootFromRom pin of the transputer is connected to logical high, the transputer starts bootstrapping immediately by executing code located at the memory address given at BOOTLOC; otherwise it waits for the first bootstrap message to arrive at any of its links [32].

The control and status register is used to flag the system of special functions. These special functions are:

- . Disable the on chip ram of the transputer.
- . Enable the T800 analyse mode.
- . Signal the transputer to boot from ROM.
- . Enable broadcasting, a special feature discussed later in the chapter.
- . Send a signal to the link topology controller to set new link topology of the system [38].
- . Set an error flag if an event failure occurred.
- . Set an error signal if a processor error occurred.

These special flags are shown in figure 3.7. These flags enable the particular operation related to these when the CSR is set to their value. These special operations are disabled when the CSR is cleared to zero.

Another special feature provided is an indivisible Test-and-Set (TAS) bit location on each T800 board. Whenever a read cycle is initiated for TAS location it is set to logical 1 in one indivisible read-test-write cycle. It is cleared by explicitly writing logical zero to this location. This TAS semaphore was one of the main features

exploited extensively for the inter processor communications in the diesel engine simulation, explained in the next chapter.

The PNUM register provides the complete bus which, when read, gives the full backplane address of the processing node. This location has another role as well. When written, it sets up the timeout for all shared memory accesses so that if the multiprocessor bus is not available within that time interval a retry could be initiated.

The ENVENTREG generates an event whenever data is written into it. It blocks any further writes into itself until the previous data has been read, i.e. the previous event has been handled by the transputer.

The last register SEMTAB provides a location at which a pointer to a semaphore table can be held. This semaphore table can be used to simulate a number of TAS semaphores by locking the table using the TAS.

Since the BUTPC has a shared memory architecture, it is necessary to have a unique address for each memory location in the system as seen by an off-board processor. This is achieved by assigning a unique number to the rack that houses the transputer processing nodes. This provides a unique page within the whole memory span of the transputer. Within this page, each processing node has its own unique identity number. The most significant byte of the address bus presents the rack number and the third most significant nibble gives the identity number of the processing node. Since there are 16 processing nodes per rack at the most, hence all of the processing nodes in the system have a unique memory address in the system. For example, for a rack number 2 the memory page would be 02xxxxxxH. The processing node zero within this area would have a 1 Mbyte memory page starting at location 02000000H, the processing node one would have a memory page starting at 02100000H and so on.

The input/output (IO) facilities to the system are provided by a standalone computer developed by Hafeez [39] which consists of an IO board and a graphics board both connected to the same backplane as the other transputer boards via links.

### 3.3 The Input/Output System

The Input/Output (IO) system developed by Hafeez [39] is based on the Philips SCC68070 microprocessor. A block diagram of the IO board is shown in figure 3.8. The SCC68070 has an on chip memory management unit, a two channel DMA controller, a serial interface, an inter-integrated circuit (I<sup>2</sup>C) bus interface and a timer. The IO system built around it exploits these features of the processor to connect to external devices such as disk drives, printer, and the console.

The floppy disk controller DP8474 uses DMA channel 2 on the SCC68070, whereas DMA channel 1 is connected to the Shugart Associates Systems Interface (SASI) bus which is a parallel bus and provides a data transfer rate of approximately 4 Mbytes per second. The SASI bus is the lowest level implementation of the Small Computer Systems Interface (SCSI) and connects a hard disk and a tape streamer to the system. The I<sup>2</sup>C interface provided on the SCC68070 is a serial bus and operates independently of the centralised bus arbiter. It is a bidirectional, two wire bus comprising of the 'Serial Clock' and 'Serial Data' lines. It is used to connect the SCC68070 to a real time clock PCF8583, and also to an IO expander, PCF8574, used to provide a parallel centronics interface for a printer. Since the BUTPC is based on the T800 transputer, in order to connect the IO board to other T800 based boards in the system a bidirectional, two wire data link consisting of an IMSC012 link adapter is provided.

The graphics board consists of a T800 transputer, a Philips SCN66470 Video and System Controller (VSC), a Philips 68070 microprocessor, keyboard and mouse interfaces, a colour palette and two different banks of memory. A block diagram of the graphics board is shown in figure 3.9. The 4 Mbyte local memory is provided to run the 'Helios' operating system and is refreshed by the T800. The other memory bank is controlled by the VSC and is used for video and display data. The T800 reads and writes to this memory via a handshake on the VSC's coprocessor interface [39]. The T800 updates the picture information in the video memory whereas the VSC uses this information from the video memory to display the pixels onto the graphics screen [39]. The colour palette is a lookup RAM that is connected to the lower byte of the T800 data bus. It translates the pixel output into Red, Green and Blue (RGB) signals which may drive the 75 ohm inputs of a video monitor [39].

### **3.4 The Backplane**

The backplane used in the BUTPC has two different physical buses, one is a 32 bit 80 Mbytes per second multiprocessor bus shared by only the T800 processing nodes and the other is the link bus for the transputer links. As mentioned before, the bandwidth offered by the transputer links was not sufficient to handle the communication for the diesel engine simulator so that a new architecture for the shared data bus was implemented. This bus structure does not use any asynchronous handshakes or address or cycle validation signals and is not tailored to any specified processor.

Each processor node in the system is supplied with input and output latching address and data buffers. The information appears on the bus during an inter processor write cycle, only long enough to be latched by all the processor nodes which are ready to latch that information at that instant of time. When an inter processor read is initiated by a processor, the data return is allowed as a separate cycle and can occur at an indeterminate interval after the initiation of the cycle. This allows a variable access

time for read cycles. At a clock speed of 20 MHz, there is about a 500 nanosecond interval between the address and data phases [33]. This allows around 10 other bus accesses to be started and/or finished on the same bus signal wires during this time.

The interleaved bus access can give rise to the possibility of many bus access attempts to the same processor. Two signals, SUCCESS and FAIL, are provided on the backplane bus to cope with this problem. All processors decode the address of a cycle that appears on the backplane. Only one of them responds and, if it has latched the information and is capable of carrying out the cycle, it sets the SUCCESS signal; otherwise it sets the FAIL signal. A failure causes the initiating processor to automatically re-send the cycle 250 nanoseconds later.

All these features do not involve the T800 processor which can carry on its processing. These features are handled by a specialised hardware involving the backplane.

### **3.5 Bus Arbitration**

In order to achieve optimum arbitration speed to decide which processor gets the next access to the bus, the arbitration mechanism has to be as fast as the multiprocessor bus access mechanism. Instead of using a standard centralised bus arbiter, a specialised distributed arbiter was developed. Limited priority shifting is provided for processors that are having difficulty getting time on the bus.

In figure 3.10, the arbiter control signals are shown. The four least significant bits are used for fixed priority. Their level is set by the geographical position of the processor node in a nineteen inch rack. The three most significant bits, DATA, BDCST and FAIR, are provided to change the priority level under certain conditions.

The DATA bit ensures that the returning data phase accesses are always high priority as a processor is waiting to receive the data. The FAIR bit is used to signal that a processor has been unsuccessfully trying to access the bus for more than sixteen time slots and the BDCST bit is used to ensure the quick completion of a new type of cycle, called a broadcast cycle.

The broadcast cycle involves all the processors connected to the bus. When a broadcast cycle is initiated by a processor, all other processors decode the current write cycle as if it is for them and reply on the SUCCESS and FAIL signals. If any FAIL occurs, then the sender resends the cycle. A processor that succeeds in capturing the cycle ignores subsequent broadcast cycles until a completely successful cycle occurs. Because of the involvement of all the processors, a broadcast cycle is always of high priority.

## **3.6 Inter Rack Connection**

There are physical constraints on the number of processing nodes that can simultaneously operate within a single backplane. Therefore the expansion of the system is possible only by connecting racks together by a communication method that does not introduce significant additional overheads.

As described earlier, each rack has its own graphics board plugged into it, with extra links to the outside world. Hence to connect two racks together, the simplest solution is to use two links on the respective graphics boards of each rack. This provides a connection of two clusters of transputers which communicate with each other via transputer links. But this is relatively slow and would introduce large overheads.



Since the BUTPC has shared memory architecture as well, it was necessary to provide a connection between the busses of the two racks as well. This is achieved by providing a high speed buffered optical fibre link running at 125 Mbits per second [33] between the two racks together. Any bus access that does not belong to any one of the local processors is passed through the optical link to another rack. If the system consists of more than two racks then, if the cycle does not belong to a given rack, it is passed higher up the rack tree structure until it reaches a rack that has the desired processing node down another branch of the tree. The bus access is then passed down the tree by the direct path to the desired processing node. The maximum latency to traverse all the optical fibre links for a tree structure of three levels is three micro seconds provided there is no other load on the system bus [33]. The figure 3.11 shows a two rack system.

For a 32 bit parallel bus the total address space is 4 Gbyte. In the current system each processor has an address space of 1 Mbyte thus giving a total number of 4095 shared memory slots. This provides a total of 256 racks that can be connected in a three level tree structure as shown in figure 3.12. But increasing the number of racks affects the bus bandwidth. Having a high bandwidth of 80 Mbytes per second on each backplane increases the overall bus bandwidth in proportion to the number of processors. The advantage of having a number of racks can be achieved by implementing programmes which use clusters of processors, with frequent access to the processors local to a given cluster and infrequent access to the processors in remote clusters via the optical fibre system.

## **3.7 Summary**

In this chapter, the hardware features of the Bath University Transputer based Parallel Computer used to simulate the diesel engine simulator are described. The next chapter

introduces the operating system implemented on the BUTPC and describes the communication mechanisms developed for the diesel engine simulator.

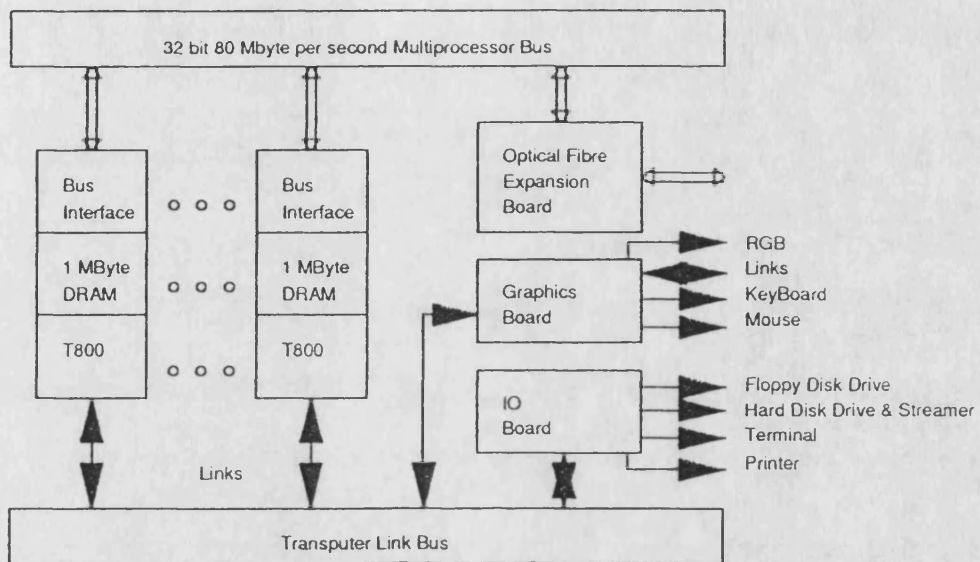


Figure 3.1 Block diagram of the BUTPC

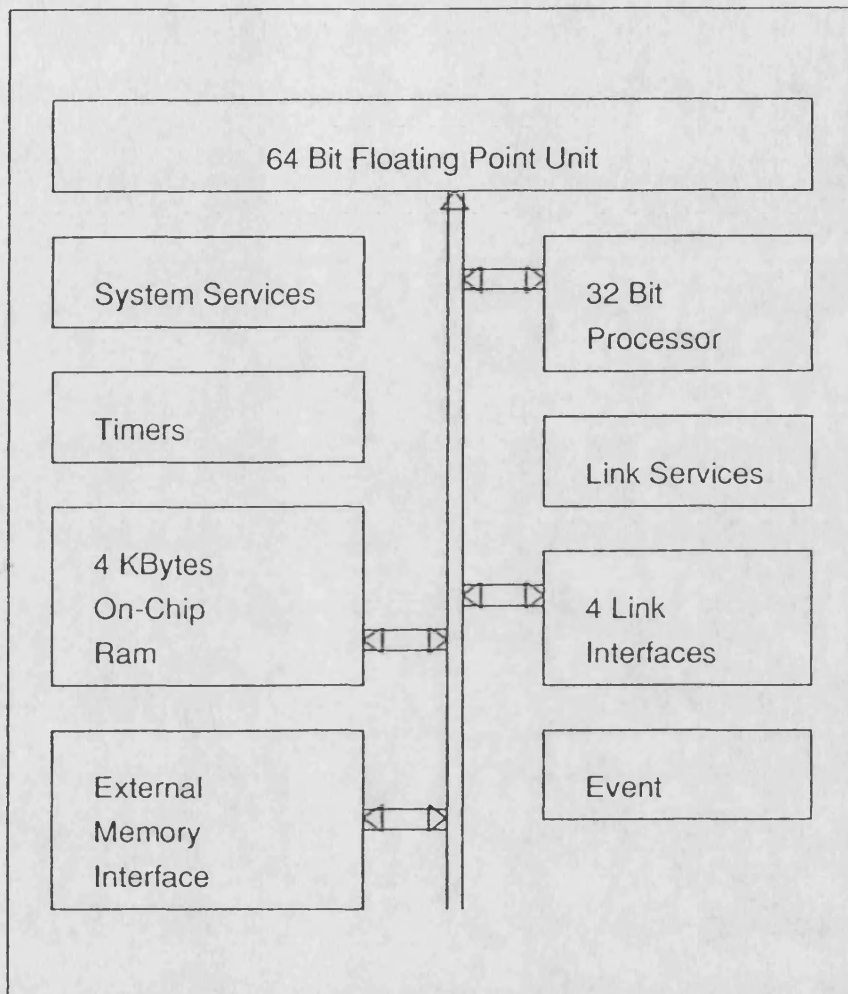


Figure 3.2 Block diagram of the INMOS T800 transputer

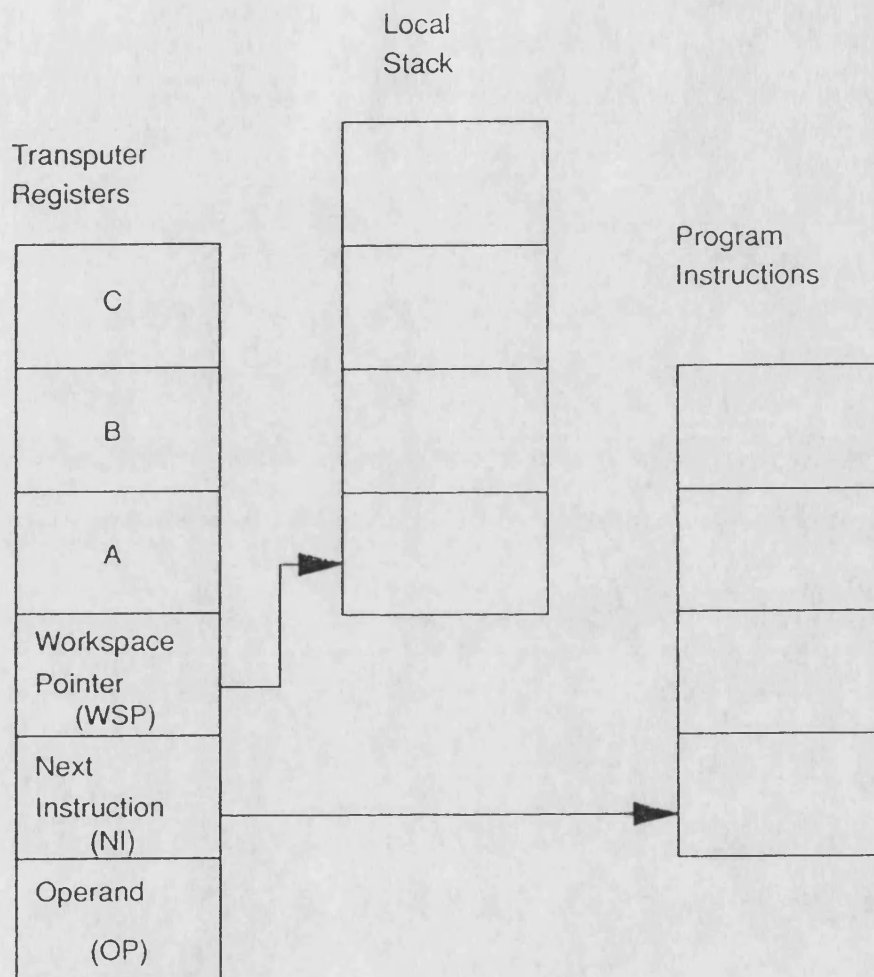


Figure 3.3 The T800 registers

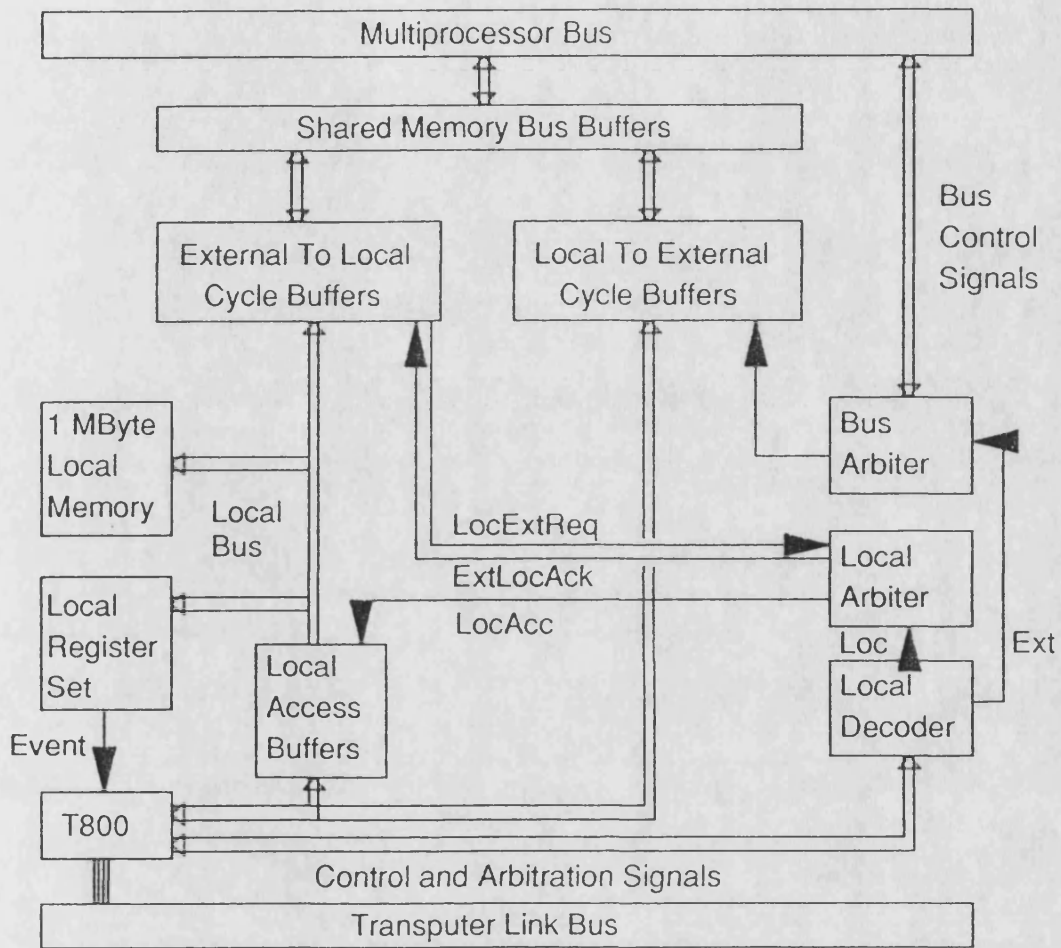
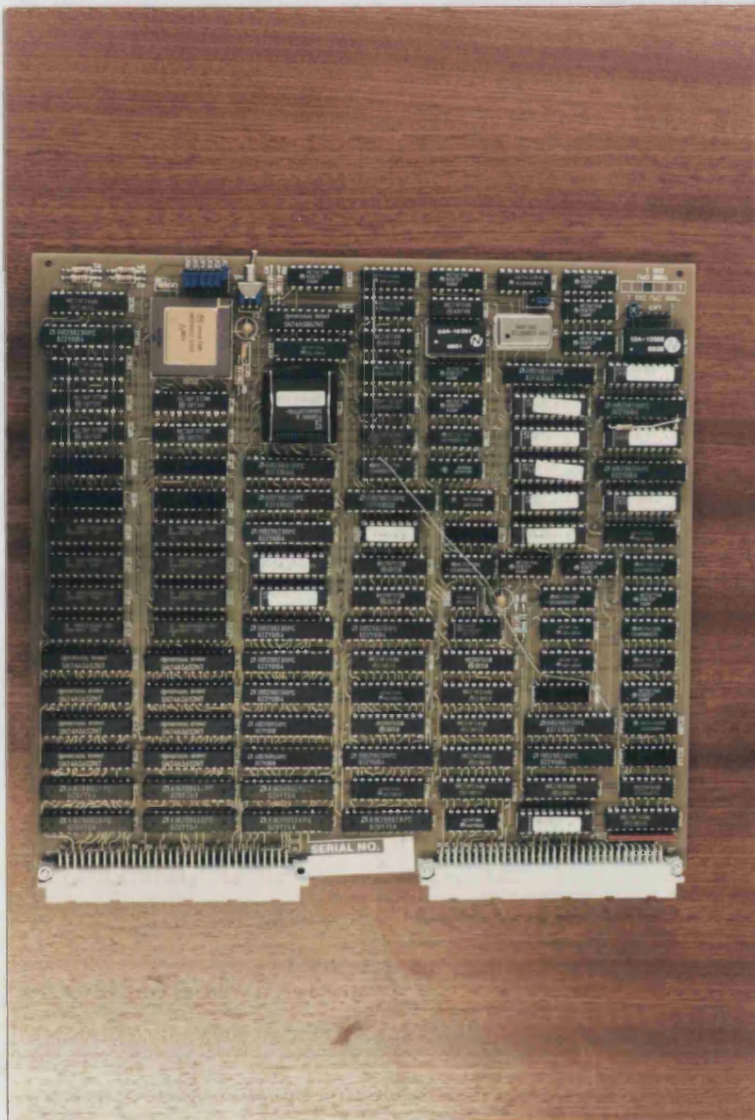


Figure 3.4 Block diagram of the T800 processing node.

Figure 3.5 The T800 processing node



Page Number

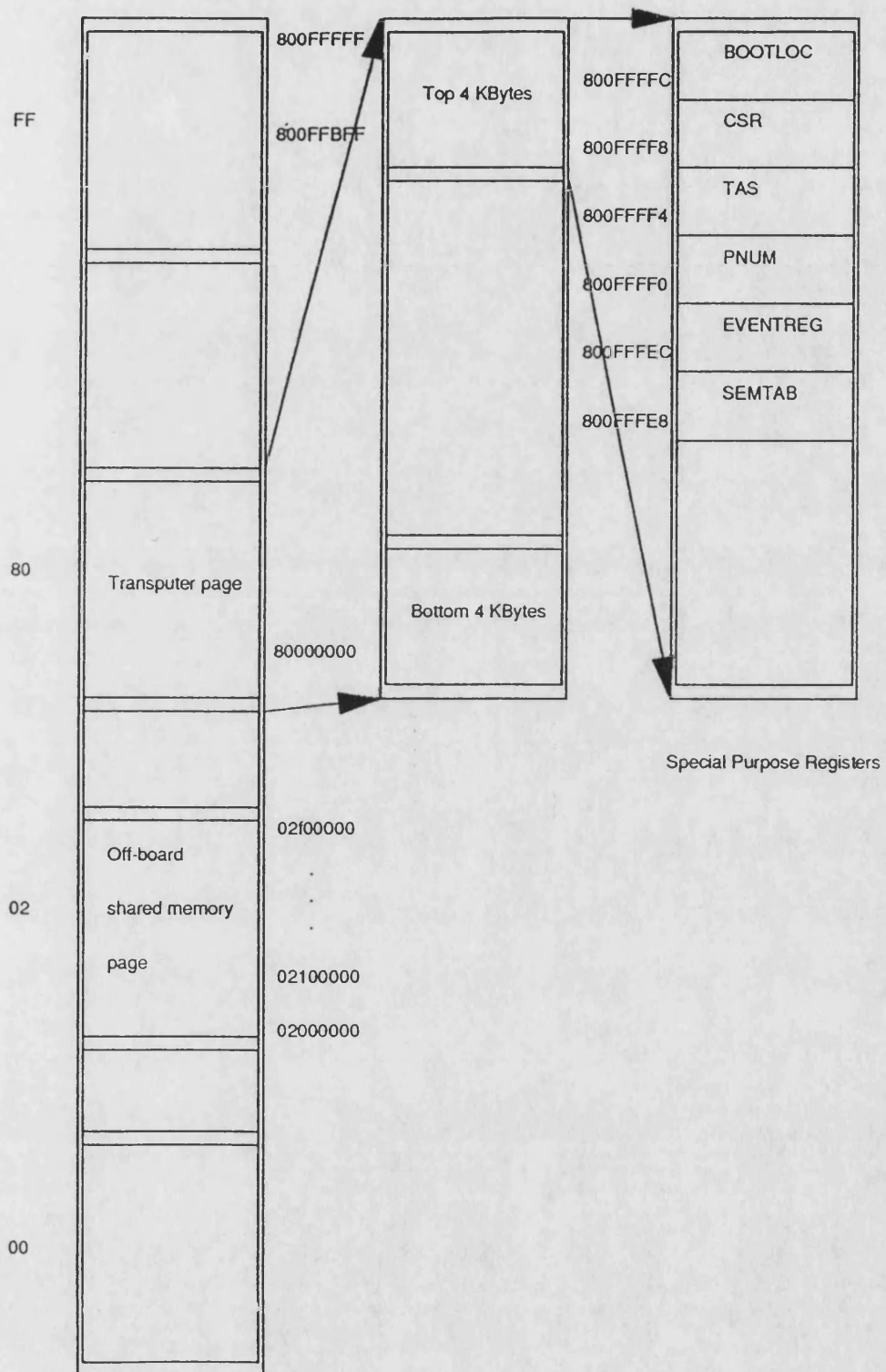


Figure 3.6 The memory map



Masking Flags	Values (Hex)
IRAMENB_M	01
ANALYSE_M	02
RAMBOOT_M	04
SHRMEMENB_M	08
BRDCSTENB_M	10
LNKTOPREQ_M	20
EVNTERR_M	40
PROCERR_M	80

Figure 3.7 The Control and Status Register Masking Flags.

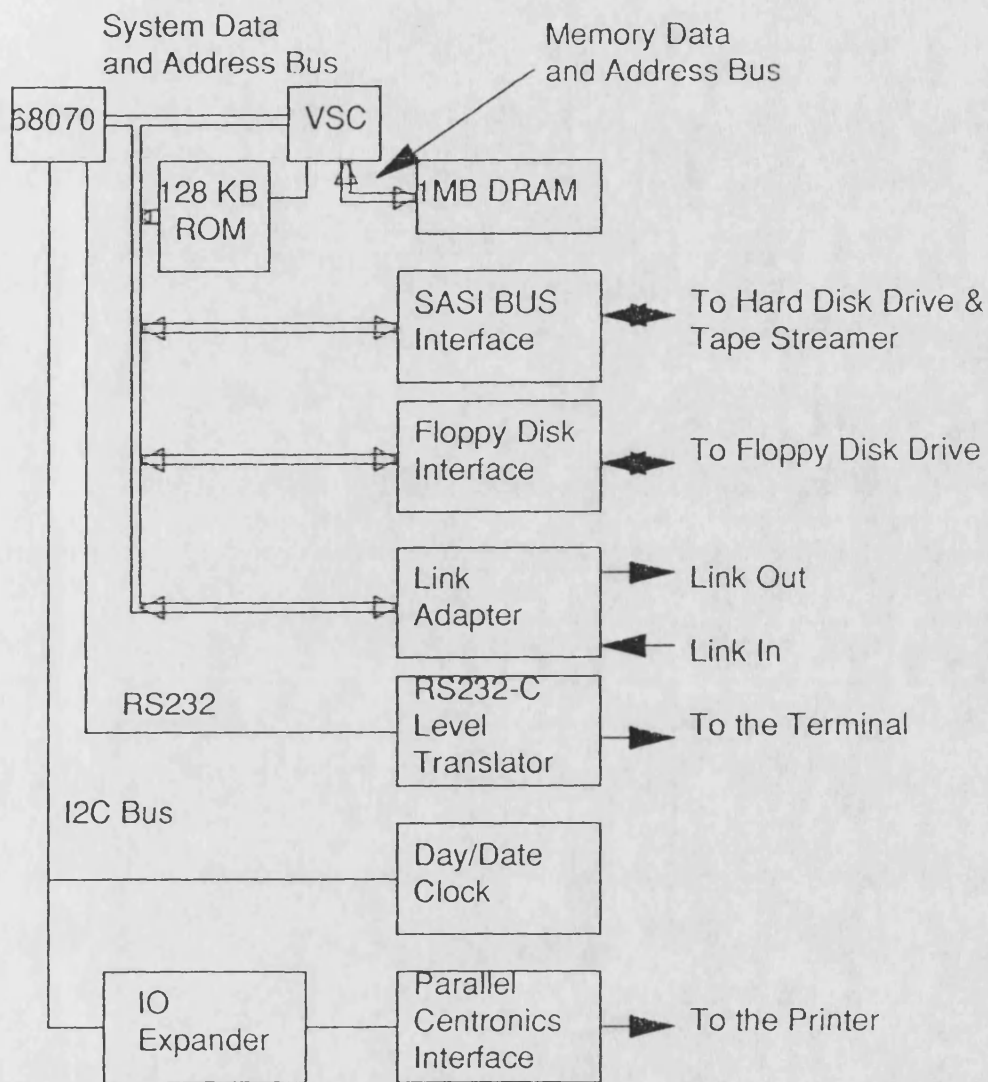


Figure 3.8 Block diagram of the IO board

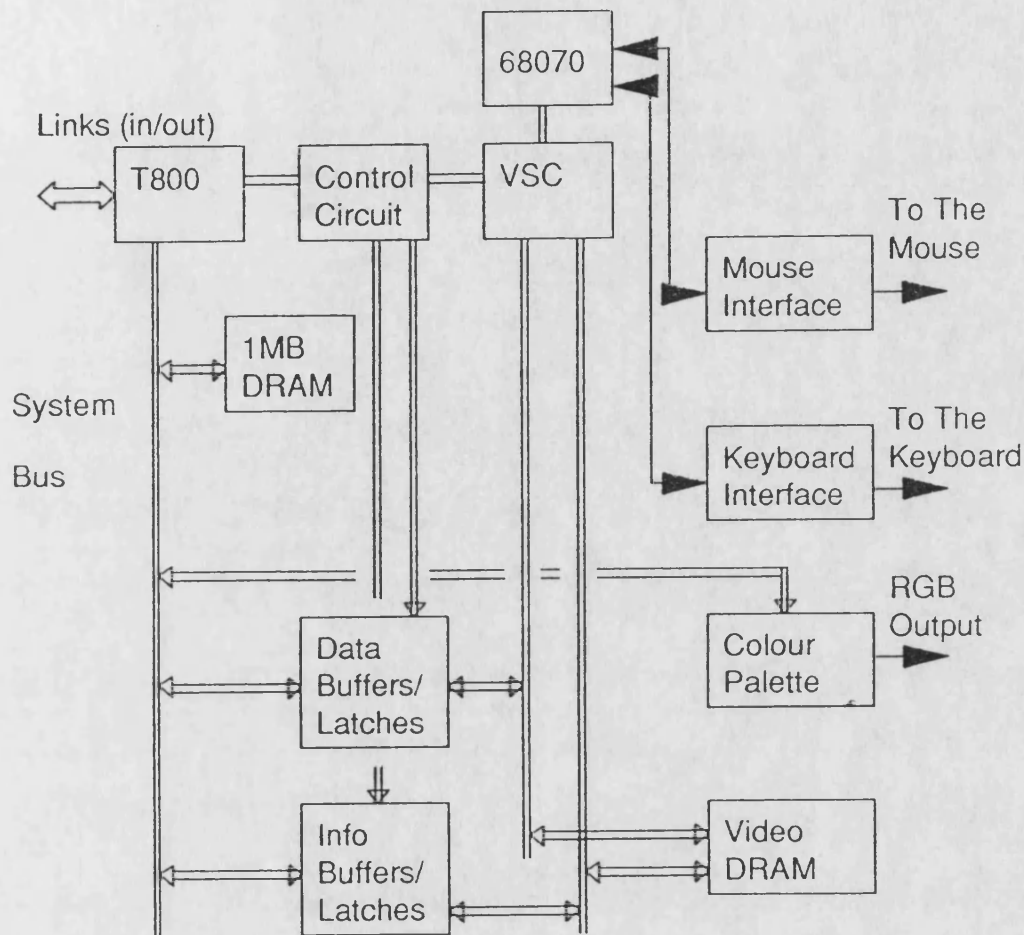


Figure 3.9 Block diagram of the graphics board

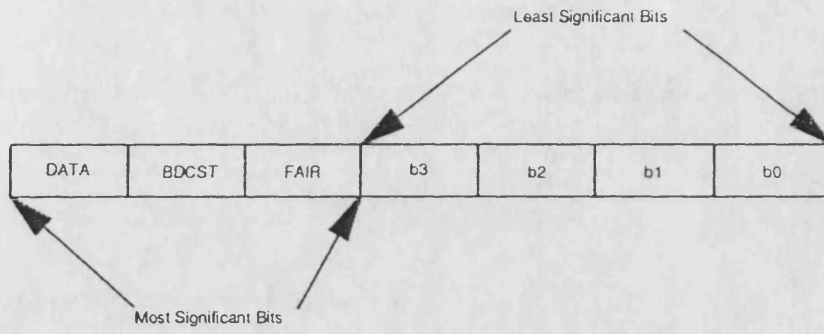


Figure 3.10 The arbitrator control bits

Figure 3.11 A two rack system (The BUTPC)



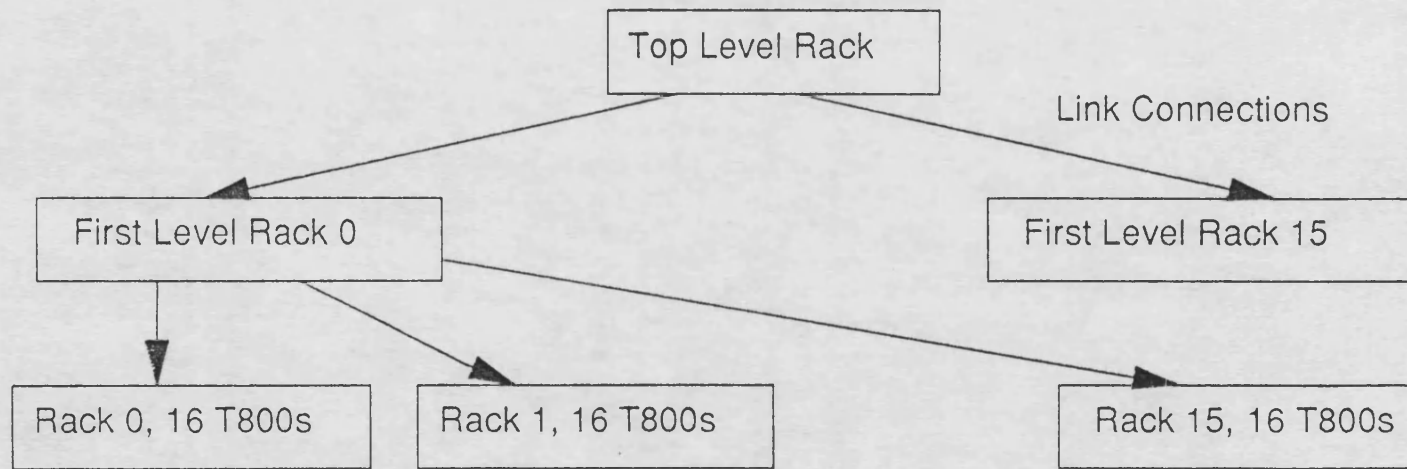


Figure 3.12 Expanded rack based parallel computer system

# Chapter 4

# The System Software

## 4.1 Introduction

Every computer hardware system requires an operating system to coordinate the resources offered by it. An operating system is a set of programmes that controls the operation of the hardware resources, utilises them efficiently and manages the data flow within the computer system.

At Bath University, a generation of parallel computers have been built around the Motorola MC68000 [40] and the MC68020 [41]. The operating system TRIPOS [42, 43, 44] was ported to these machines. TRIPOS is a single-user, multitasking operating system for small computers, originated in the University of Cambridge. It was modified by Dale to be used as a multiprocessing, multiplier system. TRIPOS is written in BCPL and supports a BCPL compiler. This gives the facility of high level programming and portability within the TRIPOS systems.

While the Bath University Transputer based Parallel Computer (BUTPC) was being built, it was decided to port an operating system to the BUTPC that:

- o would allow a high level language programming environment
- o would enable the user to write portable programmes
- o would be cost effective
- o would be independent of the processor type used
- o would allow other useful programmes to be ported on the system easily

Since the transputer was developed with OCCAM in mind [45], the obvious choice at that time would have been the Transputer Development System (TDS) supported by



INMOS. This would have made OCCAM as the programming language. But using OCCAM had its disadvantages as far as the BUTPC and its envisaged use was concerned. OCCAM did not support dynamic memory allocation [46]. It was not very much popular as a language to be ported on a large number of other processors, making its long term future uncertain. This could mean rewriting the whole existing programmes, and hence losing portability. Secondly using OCCAM would have prevented porting useful programmes and tools available on other systems such as Unix based machines.

There was an operating system called Helios being developed at Perihelion Software Limited, Shepton Mallet near Bath. Porting Helios to the BUTPC promised the following advantages:

- o The team developing Helios had been involved in the development of TRIPOS as well, which was already in use at Bath University.
- o There was a possibility of having full collaboration and support from Perihelion, making the porting and running of Helios to the BUTPC cost effective.
- o Helios is written in C, a high level language becoming popular as a system language. Writing programmes in C meant that programmes developed under Helios could be ported easily across other systems and vice versa.
- o Helios provides a Unix like shell environment.
- o Another advantage promised by Helios was that it was being developed keeping in mind the pace of technology, so that it could easily be tailored to the needs of other new sophisticated processors such as Intel i860 [3], hence providing no loss of time and effort in transferring from the existing system to the new system.

These were the reasons that decided the implementation of Helios on the BUTPC [31].

## 4.2 Helios: The Operating System

Helios is a multiuser, multiprocessor distributed operating system and is designed to run on a wide range of multiprocessor architectures [47]. For a transputer based system, it often supports an environment in which the transputers simply act as accelerators for the existing machines, such as the IBM PC, the SUN. In the case of the BUTPC, the host is a Philips 68070 based computer discussed in the last chapter. The host processor retains many of the IO functions of the standalone machine.

A standard IO server program that runs on the host machine acts as an interface to the transputer back end and allows the user to access all the standard features of the host environment.

Helios controls all the resources available within a network which is a collection of processors. It provides a consistent mechanism for accessing the resources and hides the distributed nature of the architecture from the user.

The processors in the network can be of different kinds, such as the Motorola MC68000 or the Intel 80286 plus one or more Inmos transputers. A network can be subdivided into a number of smaller networks and can be booted with any configuration topology that is physically allowed by the hardware.

Helios consists of the following main components.

- o Nucleus
- o Servers
- o Posix library

- o User interface.

## **4.2.1 The Nucleus**

The nucleus is the minimum system that must be present on every processor in the network. Its prime purpose is to control the resources of a single processor and to integrate it into the global system. It provides message passing facilities at the lowest level to all other high level system services such as file servers, resource manager, etc., which are added onto the nucleus. The nucleus consists of:

- o Kernel
- o system library
- o Loader
- o Processor manager
- o IO controller (IOC)
- o Name server

### **4.2.1.1 Kernel**

The kernel is directly responsible for managing the hardware resources of the individual processors in the network. It provides support for local and networked message passing described later in this chapter. It controls both the external and on chip memory, and provides event management and semaphore handling as well.

### **4.2.1.2 System library**

The system library is a resident sharable library which provides a general interface to the operating system services. It keeps track of the system resources allocated to a task and releases them to the system pool when the task finishes. The libraries are

loaded only on demand so that they do not impose an overhead when not required on a particular processor.

#### **4.2.1.3 Loader**

The loader is responsible for loading object code and libraries into a processor and unloading them, when they are not needed. It supports standard server interface.

#### **4.2.1.4 Processor manager**

The processor manager is an autonomous process, responsible for creating tasks, managing them when they are running and dismantling them when they terminate. It is also responsible for trapping all signals sent to a given task. When an exception such as stack overflow or arithmetic overflow occurs, a signal message is sent to the processor manager that is controlling the relevant task. The processor manager then reports the exception by running an exception routine provided either by the runtime system or by the program .

#### **4.2.1.5 IO Controller**

The IO controller (IOC) is another autonomous process that is responsible for handling IO requests of another task. Each task has its own IOC, and an IOC is also created for each link on the processor. The IOCs act as IO agents for tasks in a remote processor and manage the in-coming requests on the four links. When a nucleus wishes to gain access to an object in a remote processor and it does not have a destination port, described later, requests can be directed to the link IOC on the remote processor, which may then access that object directly.

#### 4.2.1.6 The Name Server

Every item in a Helios network is referred to as an object. This object model provides a consistent interface for accessing any item within the network; for example the same mechanism is used to locate a processor as that used to locate a file within a directory of file system located on a given processor.

To ensure that an object is unique, Helios implements a unified naming scheme, which is based on the naming table maintained by the nucleus in each processor.

Each local name table is organised as an hierarchical representation of the network as it is perceived from that processor. To access an object or a service, a task sends a general server protocol message to its IOC which looks up, in the name table, the name identified by the current string offset in the message. The IOC then passes the request on to the server port found in the name table. If the name is not present, the IOC asks the neighbouring processors. These may, in turn, need to ask their neighbours, thus implementing a distributed search of the entire network. Whenever such a search succeeds, the server, machine and cluster entries are added to the local name server, thus expanding the processor's knowledge of the network and making subsequent accesses to this server more efficient.

In the last chapter, an example of a hierarchical network was given and a block diagram is shown in figure 3.12. If the processing node 0 in the rack 0 of level 3 subnetwork has file called 'procman', then the full network address of this object will be:

`'/net/level2rack0/level3rack0/processor0/procman'.`

## 4.2.2 The Servers

Helios is based on the client-server model in which a client wanting some service sends a message to a server which performs the operation and returns a reply message.

The nucleus is a server that provides a virtual machine interface and allows the user a transparent access to all the physical resources of a processor in a transputer network. However, there are some resources which are not available on every processor in the network, for example serial console, disc drivers or printers. To control these devices, additional servers are provided so that a consistent interface to all physical resources can be maintained.

A general server protocol (GSP) is supported by all system server. All requests within this protocol have a control vector as shown in figure 4.1, where Context and Name are offsets into the data vector of null terminated pathname strings. If the offset is -1, the string is not present. The context string is equivalent to a current directory, relative to which a pathname is evaluated. The Next field is the offset of the name element to be used next. The capability encodes the client's access rights to the context object. Extra request specific parameters may be added at the end of this structure.

Each server consists of a single task. A single port supplies GSP requests to the server, which always come from an IOC. A single dispatcher process is dedicated to receiving these requests and queuing them for attention by a second process, which can either be a static process, a dynamically created process or a pool of worker processes to handle each request as it arrives.

Multiple servers may coexist so long as they support the general server protocol.

A number of standard system servers are provided with the Helios operating system [48]. Two of these are discussed below.

#### **4.2.2.1 Network server**

The network server is responsible for the initial creation and then subsequent control of the Helios nucleus through a network. It uses a resource map to boot all the processors within a defined network. A resource map defines the topology of the network, the individual processor types, the location of specific physical devices such as SCSI controllers etc. and the hierarchical subdivision of the processing units within the network.

The network server is a distributed service, with a separate component of the network server being responsible for each subnetwork defined in the network.

#### **4.2.2.2 Task Force Manager**

The collection of tasks is referred to as a task force and the task force manager (TFM) is a distributed server that controls these tasks. It consists of a number of identical servers distributed throughout a network, each controlling a different area of the network.

The TFM processes all client level program execution requests. It analysis the current state of the network and distributes the component tasks of the task force to the most suitable processing element.

Once a task force has been scheduled, then the TFM monitors each of the component tasks and informs the client when all components have terminated.

The TFM also provides the facility to abort a task force. It is thus responsible for load balancing of Helios networks, ensuring that all tasks are optimally scheduled.

### **4.2.3 The Posix Library**

The posix library is provided in Helios to make it compatible with Unix. This facility aids portability of existing programs and tools from the Unix environment to the Helios [49].

### **4.2.4 The User Interface**

A task called Shell is provided in Helios that acts as a command line interface to the operating system. the Helios Shell interface enables the user to create and control jobs interactively, both in the background and the foreground. It includes various shorthand methods to save typing and personalisation methods to enable the users to use their own commands. Several Shells can be used simultaneously.

The Shell includes support for a subset of the component distribution language (CDL). The CDL enables the user to declare the resources requirements of the component tasks of a task force. A CDL script is compiled to generate a binary object that is submitted direct to the TFM as an executable object.

## **4.3 Communication Methods In Helios**

Helios provides four different levels of communications. These are:

- o Language level IO.
- o Posix Level IO.
- o System Level IO.



- o Kernel Level IO.

### **4.3.1 The language level IO**

This is the highest level communication level. The calling convention depends on the type of language used. For example, in the 'C' implementation under the Helios operating system, these routines are represented as shown in figure 4.2. The 'fread' and 'fwrite' are used for direct input and output purposes respectively. These functions use a file stream to receive or send a given number of items of a given size and return the number of items received or sent respectively. These functions have high error recovery, therefore they tend to overload the data communication between two Helios objects, which can be a combination of tasks or files or both.

### **4.3.2 The Posix level IO**

In this second level, the IO Posix routines, shown in figure 4.3, are used for data communication between two Helios objects. If data is being read from or being written into a file, a file descriptor associated to that file is used to transfer data to or from that file. If the data is being read from or written into a task, then there is a pipe or a FIFO to which the file descriptor is associated, using which data transfer is completed. The actions of these 'read' and 'write' routines depend on the state of the pipe or FIFO they are using for data communication. An error recovery mechanism is also built in to these routines that allows them to return an error message in case of failure [49].

### **4.3.3 The system level IO**

The third level communication routines are provided by the system library. These are called 'Read' and 'Write' and are shown in figure 4.4. These routines use the Helios

streams for data communication and transfer data of a given size into or out of a buffer. They have builtin error recovery and checking which, although presenting a communication overhead, does improve the portability and safety. These routines are provided with a timeout facility as well. A timeout value of -1 gives an infinite timeout.

#### 4.3.4 The kernel level IO

The prime means of communication under Helios is message passing which is the lowest level of communication and is implemented by the kernel. The kernel routines that constitute the message passing mechanism use Helios ports for data communication and have minimum error recovery and checking associated with them.

A message port is a data structure that is implemented by the Helios kernel to make message passing transparent to the user. A message port can be a local port for message transfers between processes on the same processor or it can be a surrogate port which is an entry in the port table referring to a port on a neighbouring processor. Figure 4.5 shows the Helios port formats.

Index represents the port's offset into the port table. Cycle is incremented each time a table slot is reused. The uses field is there to garbage collect ports that are unused because a program has crashed or failed to tidyup. The link field identifies the link through which the message must be sent.

A message is divided into three parts: a header, a control vector and a data vector as shown in figure 4.6. The message consists of the sizes of the control and data vectors and the destination and reply port descriptors through which the message has to pass. The ContSize is limited to 256 words whereas the DataSize is limited to 64 kbytes.

The kernel routines that implement message passing are:

- o `NewPort();`
- o `PutMsg(mcb);`
- o `GetMsg(mcb);`

`NewPort` allocates an unused slot in the port table, initialises it and returns the port descriptor for it. `PutMsg` and `GetMsg` expect a preinitialised message control block (MCB) as argument. An MCB is shown in figure 4.7.

`PutMsg` transmits the message header on the port channel, followed by the contents of the control vector, and then the data from the data vector. If the message is not delivered before the expiry of the timeout, the whole transfer fails. A timeout of -1 gives an infinite waiting time.

`GetMsg` checks the port descriptor given in the `MsgHdr.Dest` and waits for a message header from the channel. It receives appropriate data according to `MsgHdr.ContSize` and `MsgHdr.DataSize` into the control and data vectors. It can timeout if data does not arrive within `MgHdr.Timeout` period.

`PutMsg` and `GetMsg` are point to point message passing routines. `PutMsg` cannot return until its message has been accepted or it has been timed out. Similarly `GetMsg` cannot return until it has received a message or it has timed out. In order to allow the main task to carry on doing other jobs while data communication is being done, a subprocess has to be forked to do `PutMsg` and `GetMsg`. Separate processes have also to be forked if a port is to be shared by several tasks as Helios allows only one process to transmit or receive a message on a port at one time.

If a message does not belong to a local process, it passes through to the link to a neighbouring processor and, if this is still not the final destination, it is passed through another link, and so on until it reaches the processor containing its destination port. This routing of the message is done by a high priority kernel process called a Link Guardian (LG). Each link in the system has a LG attached to it. When PutMsg finds the type of the port non-local, it sends a protocol header byte followed by the MsgHdr and then control and data vectors if they are present. This protocol header byte is interrupted by the LG which receives the MsgHdr, replaces the reply port by the descriptor of a newly allocated surrogate port, looks up the destination port in the port table and examines its type. If it is a local port and there is a receiver waiting for it, the LG sends the header to the receiver and waits to allow the receiver to take the control vector and data directly from the link. If no receiver is present, the message is received into a buffer and queued for later delivery. If the destination port is a surrogate and the next link is available, then the LG transmits the protocol byte and the message header. The control vector and data are received in parallel and are transmitted down the link when ready. If the link is busy, the message is received into a buffer and queued for later transmission and, if no buffer is available, the message is thrown away.

When a message is thrown away as a result of congestion, an exception message is generated and sent back to the reply port. The Link Guardians make a little more effort to deliver the messages but congestion may still cause them to be lost [48].

Table 4.1 shows communication speeds using different communication methods provided in the Helios operating system when the transmitter and the receiver are on adjacent processors [50]. As the simulation time available for a 1 degree step for a diesel engine running at 2000 revolutions per minute is 83 microseconds, the communication times given in table 4.1 show that using conventional Helios

communication mechanisms would not give a simulation speed up. Hence special hardware features provided in the BUTPC were used for the data communication and synchronization in the diesel engine simulator.

Table 4.1: Communication performance using Helios primitives

Message size (bytes)	Posix level read/write (micro second)	System level Read/Write (micro second)	Kernel level GetMsg/PutMsg (micro second)
4	1110	1100	125

## 4.4 New Communication Routines for the PDESIM

As mentioned earlier, the Helios communication mechanisms that use links, were not suitable for the communication requirements of the diesel engine simulation. Therefore new faster communication primitives were developed for data communication and task synchronization. The following different methods were used for these routine.

- o Backplane data communication and synchronization routines using TAS flags.
- o Packet switching using shared memory flags.
- o Packet queues using TAS flags.
- o Broadcast signals.

## 4.4.1 The backplane routines

These software routines were written to make the shared memory system appear as much as possible like faster transputer links [33]. Using these routines a direct hardware link between any two processors could be simulated. These include:

- o BPSemaphoreInit
- o BPSignal
- o BPWait
- o BPInitPort
- o BPPutData
- o BPGetData

To use these routines, a backplane event handler is installed that reads data off the EVENTREG when an event occurs and sends this data through the Helios channel to a task which is either waiting for it or is going to wait.

The BPSemaphoreInit is used to initialize a backplane semaphore structure, shown in figure 4.8, which is used to simulate the internal semaphore structure of the Helios using shared memory.

The BPSignal works by locking the processor to be signalled by reading the TAS flag. It then checks if a task on the remote or local processor has already executed a BPPWait corresponding to this BPSignal, if not, then it sets the semstate flag to 1, and releases the processor. If the remote task has done a BPWait, then, if the task is on the local processor, the BPSignal sends a byte at its local channel to inform the waiting processor, otherwise it sends an event to the remote processor and releases the processor.

The BPWait also locks the processor using the TAS flag. If the semstate of the BPSemaphore has already been set to one, then it means that a BPSignal has already been done by another task corresponding to this BPWait, so the BPWait clears the semstate flag to 0, and then releases the processor. Otherwise, the BPWait announces that it is going to wait for a BPSignal by setting the semstate flag of the BPSemaphore to one and then waits on an input channel. The BPWait is only allowed at a semaphore which is local to the processor on which the task, that executes a BPWait, is running.

The BPPutData and BPGetData use a backplane port structure shown in figure 4.9 for data communication. This port structure is initialized using BPInitPort. Each communicating task must have an initialized BPPort on its side. The BPPutData copies the data directly into the data buffer pointed to by the remote port, whereas the BPGetData just waits for the data to arrive.

Table 4.2 shows timings for these backplane communication routines. These timings still indicate that using these routines will not achieve a greater speedup for the diesel engine simulation. Hence other direct communication methods were investigated.

Table 4.2: Communication performance using backplane routines	
Message size (bytes)	BPGetData/BPPutData (micro second)
4	75

## **4.4.2 Packets**

Previously the diesel engine simulations were carried out under the Tripos operating system [21, 21]. The main communication mechanism under the Tripos is packet switching. A Tripos packet structure is shown in figure 4.10.

Tripos maintains a linked list of packet queues. The link field in the packet structure points to the next packet in the linked list queue. The id field is the identity number of the task to which this packet is being sent or if it has been received from somewhere then it indicates the id of the sender. The type field is the type of action for which this packet is intended. The res1 and res2 fields are used normally by Tripos but can be utilised to send data between two tasks. The argument fields at the end of the packet are variable and can carry long word data as permitted by Tripos.

### **4.4.2.1 Packets switching using flags**

While designing new communication primitives for the diesel engine simulation, the first approach was to simulate the Tripos packet mechanism using the BUTPC shared memory system. For this purpose, a fixed size packet structure, shown in figure 4.11, was selected. The packet structure was kept constant in order to keep a consistent packet structure throughout.

Two different packet switching methods were tried. In the first method, a shared memory flag mechanism was implemented. In this mechanism, each type of packet on each task was identified by a unique global flag. The bottom 4 Kbytes of the T800's RAM was used to allocate memory to these flags as this area of memory was unknown to Helios and could be safely used directly.



Figure 4.12 shows the flag structure. Here the `pktaddr` field gives the address of the packet for which the flag is used whereas the `SHRD_COMM_STRUC` is a table of the flag structures corresponding to each type of the flag in the PDESIM.

Whenever a packet is to be queued, its address is placed at its corresponding location in the remote processor and the flag is set to -1, indicating a new packet has arrived. The waiting task keeps on polling its local `SHRD_COMM_STRUC` table and when it finds a flag set to -1, it reads off the corresponding packet address and clears the flag to 0. Figure 4.13 shows a packet being queued from processor 0 to processor 1.

This system provided a few drawbacks. For a task that was waiting it was very time consuming to poll through all the flags corresponding to each packet type as the number of packet actions increased. Similarly increasing the number of tasks also increased the polling time as each task had its own set of packets as well. Hence a second more general approach was considered.

#### **4.4.2.2 Packet switching using queues**

In this implementation of communication routines, each task was allocated a fixed size circular buffer to simulate a packet queue. The size of this buffer was selected large enough to avoid any congestion on the queue.

All those tasks which want to send a packet to a particular task wait at the queue entry and place the packet address on the queue when it is available. The receiving task picks up the packet address from the queue and advances the tail of the queue.

A task table is copied onto all the tasks of the PDESIM. A task table is an array of task identifiers as shown in figure 4.14. Each task identifier structure contains a pointer to the TAS location of the processor on which that task is located, the head

and tail pointers to the addresses of the head and tail of the task's packet queue and the pointers to the start and end of the queue. The headptrptr and tailptrptr provide the critical section of the queue and are manipulated by only one task at a time. The qstartptr and qendptr are used to circulate the buffer queuing on the packet buffer.

The communication primitives that simulate this mechanism are:

- o Qpkt
- o Taskwait

### **Qpkt**

This routine is used to put the address of a packet on the local queue of a task which may be a local or a remote task. A pointer to the identifier of the task is fetched from the local task table using the task's ID given in the packet structure. Then the ID field in the packet is changed to that of the sender so that this packet could be identified by the receiver and sent back if required.

Now an attempt is made to access the queue of the remote task by reading its TAS location. If it is less than zero, it means the MSB of the TAS is set to 1 and someone else is using the queue. The sender delays itself and checks the TAS flag again. This process is repeated until the TAS flag is no longer less than zero.

Once the queue is locked, the current head pointer is read and the next address on the queue is calculated. If the next head value is equal to the address of the tail then the queue is full. This comparison is carried on until the receiving task takes a packet off the queue. Then the sender places the packet address at the head of the queue, advances the head pointer and releases the queue so that it could be used by other tasks in the system. Figure 4.15 shows a flow diagram of the Qpkt routine.

## Taskwait

This routine is used to wait for any packet to arrive at the task's queue. It does so by comparing the head and tail pointers. If these pointers are same then the queue is empty, and a special busy wait is carried out that allows the transputer to deschedule if necessary. When the queue is not empty the packet address is taken from the tail of the queue and the tail pointer is advanced.

Since there is only one task that updates the tail pointer when it executes a Taskwait, there is no need to lock the queue while advancing the tail pointer.

The Taskwait routine returns the packet address which was taken from the queue. A flow diagram of the Taskwait routine is shown in figure 4.16.

Communication performance for the Qpkt and the Taskwait routines is given in table 4.3. It is clear from the table that these routines provide a considerable improvement over the Helios and the backplane primitives.

Message size (bytes)	Taskwait/Qpkt (micro second)
4	22

### 4.4.2.3 Broadcasting

The broadcast feature provided in the BUTPC provided a means to signal all the processing nodes at once. In the PDESIM there exist occasions when a single task wants to synchronise with most of the other tasks at once. For example, while trying to find out system stability all of the tasks are supposed to reach a particular point where they wait for a signal from the stability task. Under such situations, it would be quicker to use broadcast and signal all these waiting tasks in one go rather than sending them individual signals. Hence the last method tried was a combination of Qpkt/Taskwait and Broadcast.

## 4.5 Data Communication and Synchronization

The data communication and synchronization using above mentioned routines may be done in a number of different ways. Some of these which are used to implement the DESIM on the BUTPC, described in the next chapter, are:

- o The data that is to be sent is written in the argument fields of the packet and then the packet is queued into the queue of the remote task. The remote task picks up the packet and reads the data directly from its arguments fields.
- o If some data is required from a remote task, then an empty packet may be sent to that task, which on receiving that packet will fill the argument fields with the required data, and send the packet back.
- o Instead of sending data directly via a packet, a pointer to a buffer is sent to the remote task that reads this address and then uses this address to read or write data into that buffer.

- o A two way synchronization can be achieved by sending a particular packet to a remote task and then waiting for it to come back.

## 4.6 Summary

In this chapter, the operating system Helios is described, and different communication methods available are discussed. New communication primitives, developed using backplane features of the BUTPC, are described. It is found that these new communication routines are faster than the Helios functions, and may provide significant speedup when used for the Diesel engine simulation.

In the next chapter, different parallel numerical algorithms and their implementation on the BUTPC, using a single cylinder model, is described.

```
typedef struct IOCCCommon {  
    Offset Context;    /* offset of context string */  
    Offset Name;      /* offset of object name string */  
    Offset Next;      /* offset of next element in path */  
    Capability Access; /* capability of context object */  
} IOCCCommon;
```

Figure 4.1 The General Server Protocol structure

```
size_t fread ( void *ptr, size_t size, size_t nobj, FILE *stream)
size_t fwrite (const void *ptr, size_t size, size_t nobj, FILE *stream)
```

Figure 4.2 The Language Level IO functions

```
int read (int fildes, char *buf, unsigned nbyte)
int write (int fildes, char *buf, unsigned nbyte)
```

Figure 4.3 The Posix Level IO functions

```
WORD Read (Stream *stream, BYTE *buffer, WORD size, WORD timeout)
WORD Write (Stream *stream, BYTE *buffer, WORD size, WORD timeout)
```

Figure 4.4 The System Level IO functions

```

typedef struct PORT {
    INT16 Index;      /* index into port table */
    BYTE Cycle;      /* port slot cycle number */
    BYTE Flags;      /* flag byte */
} PORT;

```

Figure 4.5a The Port Descriptor.

```

typedef struct Port {
    BYTE Type;      /* = T_local */
    BYTE Cycle;     /* current cycle value */
    bits TxState : 2; /* transmit protocol state */
    bits RxState : 2; /* receive protocol state */
    bits Flags : 4;
    BYTE Uses;     /* garbage collection field */
    Channel Chan;  /* data transfer channel */
    WORD *TxId;    /* pointer to transmitter */
    WORD *RxId;    /* pointer to receiver */
} Port;

```

Figure 4.5b The Port Table Entry.



```

typedef struct Surrogate {
    BYTE    Type;          /* = T_surrogate      */
    BYTE    Cycle;        /* current cycle value */
    BYTE    Flags;        /* flags + number of link */
    BYTE    Uses;         /* garbage collection field */
    Port    Port;         /* remote port descriptor */
    WORD    *TxId;        /* pointer to transmitter */
    WORD    Unused;
} Surrogate;

```

Figure 4.5c The Surrogate Port Table Entry.

```

typedef struct Message {
    struct MsgHdr MsgHdr;
    WORD          Control[...];
    BYTE          Data[...];
} Message;

```

Figure 4.6a The message structure

```

typedef struct MsgHdr {
    unsigned      DataSize : 8;
    unsigned      Contsize : 8;
    unsigned      Flags : 8;
    Port          Dest;
    Port          Reply;
    WORD          FnRc;
} MsgHdr;

```

Figure 4.6b The message header structure

```

typedef struct MCB {
    struct    MsgHdr;    /* message header buffer    */
    WORD     Timeout;   /* transfer timeout         */
    WORD     *Control;  /* pointer to control vec   */
    WORD     *Data;     /* pointer to data vector   */
} MCB;

```

Figure 4.7 The message control block structure

```

typedef struct BPSemaphore {
    Channel  lock;
    int      semstate;
    void     *userdata;
} BPSemaphore;

```

Figure 4.8 The BPSemaphore structure

```
typedef struct BPPort {  
    BPSemaphore sem;  
    struct BPPort *brother;  
    void        buf;  
    long        len;  
} BPPort;
```

Figure 4.9 The BPPort structure

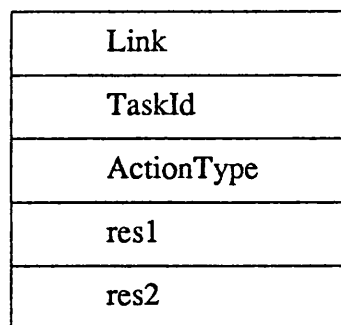


Figure 4.10 The Tripos packet structure

```

typedef struct PACKET {
    int      id;
    ACTION   type;
    ARGDATA  res1;
    ARGDATA  res2;
    ARGDATA  arg1;
    ARGDATA  arg2;
    ARGDATA  arg3;
    ARGDATA  arg4;
    ARGDATA  arg5;
    ARGDATA  arg6;
    ARGDATA  arg7;
    ARGDATA  arg8;
    ARGDATA  arg9;
    ARGDATA  arg10;
} PACKET;

```

```

typedef union ARGDATA {
    char  dummy[8];
    char  c, *cp;
    int   i, *ip;
    long  l, *lp;
    float f, *fp;
} ARGDATA;

```

Figure 4.11 The simulated packet structure

```
typedef struct FLAGS {  
    int    f;  
    long   pkt;  
} FLAGS;  
  
typedef struct SHRD_COMM_STRUC {  
    FLAGS flag[LAST_ACTION];  
} SHRD_COMM_STRUC;
```

Figure 4.12 The Flags structure

Processor 0					
	a	b	.	.	.
P0					
P1					
.					
.					
.					

Processor 1					
	a	b	.	.	.
P0		f p			
P1					
.					
.					
.					

f = flag corresponding to pkt type b on P0  
p = address of packet 'p' of type 'b'

Figure 4.13 The packet switching using flags from P0 to P1

```

typedef struct TASKID {
    volatile int    *semflag;
    volatile long  **headptrptr;
    volatile long  **tailptrptr;
    volatile long  *Qstartptr;
    volatile long  *Qendptr;
} TASKID;

```

Figure 4.14 The Task Identifier structure

```

void Qpkt ( PACKET * pkt)
{
    change pkt id from that of receiver to the sender();
    get access to the remote queue();
    get current head value and calculate new head value();
    if next head value is same as the tail value then wait();
    place pkt on the queue();
    update the head value();
    release the queue();
}

```

Figure 4.15 Flow chart for Qpkt



```
PACKET *Taskwait ( void )  
{  
    if head is same as tail then queue is empty, so wait();  
    read packet address from tail();  
    calculate and update new tail value();  
    return packet address();  
}
```

Figure 4.16 Flow chart for Taskwait

# Chapter 5

# Parallel Numerical Integration

## 5.1 Introduction

The state equations describing the Diesel engine model are ordinary differential equations (ODEs) and give rates of change of the state variables, mass fuel to air ratio and temperature. In order to get the values of these state variables over the engine cycle these state equations have to be integrated. Since the initial state of these engine variables may be estimated or taken from a knowledge of the engine, the problem is reduced to solving ordinary differential equations with initial conditions.

A number of serial methods exist to solve this initial value problem. Two different approaches exist to implement a solution using parallel processing; the geometric parallelism approach and the algorithmic parallelism approach. In this chapter, four different numerical integration methods are described. A single cylinder Diesel engine is simulated using three of these methods.

## 5.2 Geometric Parallelism

In this type of parallelism, the set of  $n$  ODEs is partitioned and allocated to each of the parallel processors. Then a standard integration algorithm is used with each processor responsible for calculating only those equations allocated to it.

The simplest method of integration is Euler's method [51]. If the differential equation given in equation 5.1 is to be solved, provided  $y=y_i$  is known at  $x=x_i$ , then a new value of  $y_{i+1}$  at  $x_{i+1}$  at a distance  $h$ , can be predicted using the slope of the function at the initial point  $x_i$ .

$$\frac{dy}{dx} = f(x,y) \quad 5.1$$

Figure 5.1 represents Euler's method graphically. The extrapolated value is given by

$$y_{i+1}^P = y_i + f(x_i, y_i) h \quad 5.2$$

A fundamental source of error in Euler's method is that the derivative at the beginning of the integration step  $h$  is assumed to apply across the entire interval.

One method to improve the estimate of the slope involves the determination of the derivative twice for the interval, once at the initial point and once at the end point. The derivatives are averaged to obtain an improved estimate of the slope for the entire interval, as shown in figure 5.2 and given by equation 5.3.

$$\frac{dy}{dx} = f(x,y) \quad 5.3a$$

$$f_i = f(x_i, y_i) \quad 5.3b$$

$$y_{i+1}^P = y_i + f_i h \quad 5.3c$$

$$f_{i+1} = f(x_{i+1}, y_{i+1}^P) \quad 5.3d$$

$$y_{i+1}^C = y_i + \frac{f_i + f_{i+1}}{2} h \quad 5.3e$$

Equation 5.3c predicts the first approximation and equation 5.3e corrects it at the end point. Hence this gives the name Modified Euler Predictor Corrector Method (MEPCM).

Since  $y_{i+1}$  appears on both sides of the corrector formula, it can be applied iteratively to improve the predicted solution.

Euler's methods are based on a Taylor series and use only first order terms of the series which increases round off errors. Those methods which use higher order terms of a Taylor series as well, introduce lesser error but at a computation cost. An example of such a method is the Runge-Kutta (RK) method.

A general RK method may be described by equation 5.4 [51].

$$y_{i+1} = y_i + \varphi(x_i, y_i, h) h \quad 5.4$$

where  $\varphi(x_i, y_i, h)$  is called an increment function which can be interpreted as a representative slope over the integration interval. It can be written in general form as shown in equation 5.5.

$$\varphi = a_1 k_1 + a_2 k_2 + \dots + a_n k_n \quad 5.5$$

where the  $a$  terms are constants and the  $k$  terms are

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h)$$

$$k_n = f(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \dots + q_{n-1,n-1} k_{n-1} h)$$

If  $n=1$  then the general RK formula reduces to the Euler method. The values of  $p$  and  $q$  are evaluated by equating the terms in equation 5.4 to the terms in a Taylor series expression. Equation 5.6 gives a classical fourth order RK formula [51].

$$y_{i+1} = y_i + [k_1 + 2k_2 + 2k_3 + k_4] h / 6 \quad 5.6$$

where

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i+0.5h, y_i+0.5hk_1)$$

$$k_3 = f(x_i+0.5h, y_i+0.5hk_2)$$

$$k_4 = f(x_i+h, y_i+hk_3)$$

## 5.3 Algorithmic Parallelism

In this method, the parallelism is achieved by partitioning across the algorithm rather than across the system of equations.

Franklin [52] discusses parallel predictor corrector and RK methods. The numerical integration schemes of the RK type proceed a step at a time while performing several function evaluations within the step. These internal computations may be interpreted as approximations to the solutions at points lying within the step. If acceptable values of the discrete solution at equally spaced intervals within a step or block are obtained, then the method that does so is called a block method. In a  $k$ -point block method, each pass through the algorithm simultaneously produces  $k$  new equally spaced solution values.

Block implicit methods may be applied in a one-step mode or in a predictor corrector mode [53]. In one-step mode, only the last point in the block is used to compute the first approximations to the  $k$  values of the next block. Then, implicit formulas are applied iteratively until convergence is achieved to the maximum order of accuracy obtainable. An example of a parallel four point block implicit one-step method (BIOSM) due to Worland [54] is given in equation 5.7 and is graphically represented in figure 5.3.

Predictor:

$$y_{i+r,0} = y_i + r h f_i \quad r=1,2,3,4 \quad 5.7a$$

Corrector:

$$\begin{pmatrix} y_{i+1,s+1} \\ y_{i+2,s+1} \\ y_{i+3,s+1} \\ y_{i+4,s+1} \end{pmatrix} = y_i + h \begin{pmatrix} 251/720 & 646/720 & -264/720 & 106/720 & -19/720 \\ 29/90 & 124/90 & 24/90 & 4/90 & -1/90 \\ 27/80 & 102/80 & 72/80 & 42/80 & -3/80 \\ 14/45 & 64/45 & 24/45 & 64/45 & 14/45 \end{pmatrix} \begin{pmatrix} f_i \\ f_{i+1,s} \\ f_{i+2,s} \\ f_{i+3,s} \\ f_{i+4,s} \end{pmatrix}$$

5.7b

When the block implicit method is adopted to a predictor-corrector mode, then all the solution values of a block may be used to predict a solution at each node of the next block.

Birta and Abou-Rabia [55] developed a formula for the block implicit predictor-corrector method (BIPCM). Equation 5.8 gives predictor and corrector formulas.

Predictor:

$$y_i^p = y_0 + h \sum_{j=0}^k B_{ij} f_{-j} \quad 5.8a$$

$$f_i^p = f(t_i, y_i^p) \quad 5.8b$$

Corrector:

$$y_i^c = y_0 + h \sum_{j=0}^k C_{ij} f_j^p \quad 5.8c$$

$$f_i^c = f(t_i, y_i^c) \quad 5.8d$$

where

$$i = 1, 2, \dots, k$$

$$f_{-j} = f(t_{-j}, y_{-j}), \quad f_j = f(t_j, y_j)$$

$$t_{-j} = t_0 - jh, \quad \text{and} \quad t_j = t_0 + jh$$

Here  $(t_0, y_0)$ ,  $(t_{-1}, y_{-1})$ , ...  $(t_{-k}, y_{-k})$  are known values within the current block and the base point,  $t_0$ , separates the known data from the set of new values. The coefficients  $B_{ij}$  and  $C_{ij}$  are constants that depend upon the value of  $k$ . Equation 5.9 gives a four point BIPCM due to Abou-Rabia [56].

Predictor:

$$\begin{array}{c|c|c|c|c|c|c} y_{n+1,1}^p & & & & & & f_n \\ y_{n+2,1}^p & & & & & & f_{n+1,1} \\ & = y_n + h & 1901/720 & -27774/720 & 2666/720 & -1274/720 & 251/720 \\ & & 1079/90 & -2396/90 & 2594/90 & -1316/90 & 269/90 \\ y_{n+3,1}^p & & 2877/80 & -7638/80 & 8712/80 & -4698/80 & 987/80 \\ y_{n+4,1}^p & & 3914/45 & -11456/45 & 13704/45 & -7616/45 & 1634/45 \\ & & & & & & f_{n+2,1} \\ & & & & & & f_{n+3,1} \\ & & & & & & f_{n+4,1} \end{array}$$

5.9a

Corrector:

$$\begin{array}{c|c|c|c|c|c|c} y_{n+1,1}^c & & & & & & f_n \\ y_{n+2,1}^c & & & & & & f_{n+1,s} \\ & = y_n + h & 251/720 & 646/720 & -264/720 & 106/720 & -19/720 \\ & & 29/90 & 124/90 & 24/90 & 4/90 & -1/90 \\ y_{n+3,1}^c & & 27/80 & 102/80 & 72/80 & 42/80 & -3/80 \\ y_{n+4,1}^c & & 14/45 & 64/45 & 24/45 & 64/45 & 14/45 \\ & & & & & & f_{n+2,s} \\ & & & & & & f_{n+3,s} \\ & & & & & & f_{n+4,s} \end{array}$$

5.9b



## 5.4 Power Series Expansions

Another example of 'parallelising' a serial method is the use of the technique of power series expansions using symbolic differentiation. The recursive calculation of the power series coefficients and in particular the calculation of the total derivatives follows a problem-dependent algorithm which may be automated.

In the first step of the procedure, the right hand members of the differential system are broken into simple expressions of two operands, such as  $a+b$ , and elementary functions, such as  $\sin(x)$ , while introducing auxiliary variables for the intermediate results. Then, recursive formulas are used to compute the derivatives. These recursive routines are independent and may be executed concurrently on a parallel computer of the multiple instruction multiple data type.

Equation 5.10 gives a Taylor series expansion for  $dy/dx = f(x,y)$ .

$$y_{i+1} = y_i + hy^{(1)} + \frac{h^2}{2!} y^{(2)} + \dots + \frac{h^k}{k!} y^{(k)} \quad 5.10$$

where

$$y^{(k)} = \frac{d^k y}{dx^k}$$

and  $k$  is the order of the Taylor series. When  $k=1$  the Taylor series reduces to the Euler method.

Barton et al [57] evaluate Taylor series methods and compare them with predictor-corrector and RK methods. They describe it as a highly stable, flexible, and fast variable step method.

Halin [58] and Halin et al. [59] describe a number of recursive formulas to calculate the terms of the Taylor series iteratively. Equation 5.11 gives a recursive formula for the Taylor series.

$$y(t) = \sum y_i^{(v)}(t_0) \frac{h^{(v)}}{v!} \quad 5.11$$

where  $h = t - t_0$ ,  $v =$  number of terms in the Taylor series and  $y_i^{(v)}(t_0)$  is the  $v$ th derivative of  $y_i$  at  $t_0$ . Some of the recursive formulas for addition, multiplication and division are given in equation 5.12.

$$\begin{aligned} r &= p + q \\ r_u &= p^{(v)} + q^{(v)} \quad v \geq 0 \end{aligned} \quad 5.12a$$

$$\begin{aligned} r &= p q \\ r^{(v)} &= \sum_{s=0}^v \binom{v}{s} p^{(s)} q^{(v-s)} \quad v \geq 0 \end{aligned} \quad 5.12b$$

$$\begin{aligned} r &= p / q \\ r^{(v)} &= p_0 / q_0 \quad v = 0 \\ &= \left( p^{(v)} - \sum_{s=1}^v \binom{v}{s} r^{(s)} q^{(v-s)} \right) / q_0 \quad v > 0 \end{aligned} \quad 5.12c$$

The power series method was tried on a simple set of ODEs. Appendix A gives a procedure to solve a set of equations. This approach was discontinued because of the fine grain parallelism involved in solving the ordinary differential state equations describing the Diesel engine.

## 5.5 Single Cylinder Implementation on Transputers

Three different methods were investigated to implement a single cylinder model on the BUTPC. These are:

- o Modified Euler Predictor-Corrector Method (MEPCM).

- o Block Implicit One-Step Method (BIOSM).
- o Block Implicit Predictor-Corrector Method (BIPCM).

A simple cylinder model was taken for these experiments. Apart from having no manifolds, a fixed amount of fuel injection was allowed and the speed of the crank shaft was kept constant at 1500 rpm. The poppet valves were directly connected to the atmosphere. A Watson heat release model was used to calculate heat release due to combustion and the Hohenberg heat transfer model was used to calculate the heat loss from the cylinder gases to the cylinder walls.

The following sections discuss the above three implementations of the single cylinder model on the BUTPC and describe different approaches utilised to solve data transfer and synchronisation problem.

### **5.5.1 Single Cylinder Implementation using the MEPCM**

Since the MEPCM does not introduce any algorithmic parallelism, the system must be separated across physical boundaries to make it a parallel problem. As a single cylinder itself may be taken as a single control volume system, with mass flow across the valves and energy flow across the walls while producing power at the input of the crankshaft, the solution of system of a single cylinder may be implemented as a single task. Figure 5.4 gives the solution algorithm for the MEPCM to solve a single cylinder model.

All the cylinder processes are divided into six main sections depending upon the physical processes and the valve states; namely: scavenge, induction, and exhaust, when both inlet and exhaust valves are open, when only inlet valve or only exhaust valve is open, and compression, combustion, and power, when all the valves are closed.

In order to speed up the calculation the following set of lookup tables are used.

- o cylinder volume
- o cylinder surface area
- o cylinder rate of change of volume
- o inlet valve effective flow area
- o exhaust valve effective flow area
- o heat transfer
- o mass flow

The stability criteria used is

$$1 - \varepsilon < \frac{y^c}{y^p} < 1 + \varepsilon \quad 5.13$$

where  $\varepsilon$  is the maximum error allowed between two consecutive points. An auto step reduction scheme was implemented which reduced the step size by one half if the stability criteria did not meet after a predefined number of iterations.

Figure 5.5a shows the effect of step size on the over all computation time. It is clear from the graph that as the step length increases the computation time reduces but after a certain steplength the computation time starts increasing again. This is because, as step size increases the system becomes unstable and the number of substeps per integration step increases, which in turn, increases the amount of calculation, specially in the scavenge or the combustion sections where the system may exhibit stiffness due to valve overlapping or combustion processes. Hence the advantage of having a higher stepsize is lost in order to keep the system stable. Figure 5.5b shows the same effects for a similar single cylinder engine but using only valve effective area tables. The average calculation time is considerably less due to a greater number of computations involved in junction flow and cylinder volume calculations. Figure

5.5 also shows that if the tolerance on the allowable error is increased, the calculation time per cycle is reduced.

Figure 5.6 shows one cycle of the state parameters pressure, temperature, fuel to air ratio and mass within the cylinder of a single cylinder engine as simulated by the MEPCM.

### **5.5.2 Single Cylinder Implementation using the BIOSM**

Unlike the MEPCM, the BIOSM presents algorithmic parallelism. As it is a block method, more than one point can be estimated at one time and the solution progressed, theoretically, at a speed equal to the number of points in the block times the speed of the serial equivalent. In practice, the speed is not a linear function of the number of points in the block due to the overheads caused by inter processor communications and synchronisations. Haysom [21] has mentioned a figure of two for a four point block one-step method.

While implementing a single cylinder model using the BIOSM, the following points were considered.

- o A fixed step size approach was used and the step sizes used were taken from the experience of the MEPCM. These are given in table 5.1.
- o A four point block was selected, which, along with giving an accuracy of the order of six, provides a reasonable number of processors to see the effects of interprocessor communication..
- o No stability checks were done on the results. Hence a fixed stepsize BIOSM was implemented.

Figure 5.7 gives the solution algorithm for the implementation of the single cylinder BIOSM. There are two synchronisation points in this algorithm. The first synchronisation is just before the correction phase. This is to wait until all the processors in the block have predicted their corresponding points as these are needed to calculate the corrected values.

Table 5.1 Fixed step sizes for the BIOSM single cylinder implementation

Section	Stepsize (deg)	
Scavenge	0.25	
Induction	0.50	$\theta \leq 80^\circ$
	1.00	$\theta > 80^\circ$
Compression	5.00	$\theta \leq q_{inj} - 20^\circ$
	1.00	$\theta > q_{inj} - 20^\circ$
Igdel	0.50	
Combustion	0.25	$\theta \leq 400^\circ$
	1.00	$\theta > 400^\circ$
Power	5.00	
Exhaust	1.00	$\theta \leq 660^\circ$
	0.50	$\theta > 660^\circ$

The second synchronisation is just before advancing the block calculations to the next point. This wait is done in order to keep all the processors together and have same base value throughout the block at all times. The synchronising routines are discussed in section 5.6.

The BIOSM may be improved if a stability criteria is introduced as in the MEPCM, and then the step reduction is allowed through the block. The calculations are repeated if the stability does not reach within a certain number of tries [60].

Figure 5.8 shows pressure, temperature, fuel to air ratio and mass plots over one cycle for the simulated single cylinder BIOSM. The calculation time for this simulation is 0.225 sec per cycle which compares with that of the MEPCM with an accuracy of 0.5% and a step size of  $2^0$ .

### 5.5.3 Single Cylinder Implementation using the BIPCM

This method has been studied by Birta et al [55] and Abou-Rabia [56]. They also describe a number of error functions to test the stability of the system [56]. While implementing a single cylinder Diesel engine model on the BUTPC using the BIPCM, following points were considered.

- o A four point block was taken.
- o A variable step scheme was adopted.
- o A stability criteria due to Abou-Rabia and Birta [60] was used to decide the step reduction.

The integration stepsize adjustment method of Abou-Rabia and Birta is based on an analysis of the error characteristics for the previous block. The maximum error for all the points in the block is taken as the error for the whole block. The error for each point in the block is a normalised error derived from the difference of the predicted and corrected values and a permissible error value. The value of the error is given by equation 5.14.

$$R = \frac{|y_i - y_i^p|}{|\epsilon|} \quad 5.14$$

The block maximum error,  $R_m$ , is used as a basis for a steplength adjustment scheme. The size of the steplength adjustment is given by equation 5.15, and is dependent on

the point in the block from which the maximum error was derived and the number of points in the block.

$$h_{\text{new}} = \sigma h_{\text{old}} \quad 5.15a$$

$$\sigma = R^{-\gamma} \quad 5.15b$$

$$\gamma = (\rho + 1)^{-1} \quad 5.15c$$

$\rho$  is the order of the solution value corresponding to  $R$ . The  $R_m$  determines the progress of the solution. If it is less than unity, then the solution may proceed, otherwise the step is repeated. The new step length is used in both the cases.

Figure 5.9 gives the solution algorithm for the implementation of the single cylinder BIPCM Diesel engine on the BUTPC. The Bmatrix is the coefficient matrix for the predictor formula, and has to be recalculated as it depends upon the maximum error in the whole block [55].

Here again synchronisation between all the processors of the block is needed just before the corrector is applied, so that only the latest predicted values are used. Similarly synchronisation is required just before a block maximum error value is determined so as to decide the system stability later on. The synchronising routines are discussed in the next section.

Figure 5.10 shows the state variables pressure, temperature, fuel to air ratio, and mass within the cylinder as simulated by the BIPCM. The calculation time for this simulation is 4.185 seconds. Such a high value of calculation time is due to a check on stability that reduces the block step size to a smaller value, especially when valves are about to open or close and also during early stages of combustion.



## **5.6 Synchronisation Mechanism for The Block Methods**

Synchronisation in the BIOSM and in the BIPCM requires all the processors in the block to reach a fixed point before any of them commence further calculations, such as starting the corrector phase. Two different mechanisms were implemented to achieve this; one using shared memory flags [21], and the other using backplane routines BPSignal and BPWait [33]. Since both of the block algorithms were implemented for four points, the routines were written to synchronise four processors.

### **5.6.1 Four Point Synchronisation Using Flags**

In order to synchronise four processors, four flags are needed to represent each processor in the block. Similarly each processor keeps a separate flag in its local memory for each of the block processors. Hence there are sixteen flags altogether.

On every synchronisation, each processor sets its respective flag on all the other processors, and then waits for all its local flags to be set by others. After all the flags are found set, calculation is proceeded by clearing all the local flags for the next time use.

On the T800, these flags are implemented as 4 bytes of a long word. Each byte represents one of the processors in the block, which is set by the corresponding processor. The local processor waits by checking the whole long word. It is cleared to zero and the synchronisation process is finished.

If only one set of flags is used for synchronisation, then it is possible for a processor to continue and reach the next synchronisation point and set its remote flags for a second time prior to another processor clearing its local flag for the first synchronisation. A processor 'deadlock' will occur as the second setting of the flag

will be lost when the second processor clears its local flags for the first synchronisation point. This loss of signal can be avoided by having two sets of flags, one for the first synchronisation, for example before the correction phase, and the other for the second synchronisation after the correction phase. In this way double buffering is provided between clearing and setting of the flags.

### **5.6.2 Four Point Synchronisation Using BPSignal and BPWait**

This method uses the back plane routines implemented on the BUTPC [33]. Each processor in the block has a set of backplane semaphores corresponding to the remote processors. For a four point block, each processor has three backplane semaphores, on which it is going to wait and it also has a pointer to the backplane semaphores on the other three processors which belong to it.

All the semaphores are initialised with zero count so that any processor which does a BPWait waits until a BPSignal is executed by a remote processor. Six processes are forked on each processor, three for BPSignalling the remote processors and three for BPWaiting at the local BPSemaphores. The synchronising routine triggers first the signalling processes and then the waiting processes. It exits only when all the local BPSemaphores have been signalled.

As the BPSignal and BPWait do not implement actual signal and wait procedures, but instead simulate these processes in such a way that the BPSemaphore may have only one of the two values, 0 or 1, the actual signalling and waiting mechanism becomes like setting and clearing flags. Hence a protective mechanism is needed, just like the one described for the flag mode synchronisation, in order to avoid any 'deadlock' due to a loss of any BPSignal. Thus two sets of BPSemaphores are maintained, each for alternative synchronisation.

## 5.7 Summary

In this chapter a single cylinder Diesel engine is simulated on the BUTPC using three different numerical algorithms. It is found that a compromise exists between step size, simulation speed and accuracy of the solution. The BIOSM, with no step reduction and stability test, provides a high speed up ratio compared with the MEPCM, but using a stability testing mechanism, the gain in speed is lost due to a reduction in actual integration step size, as in the case of the BIPCM. In the next chapter a full implementation of a six cylinder turbocharged Diesel engine using MEPCM is described.

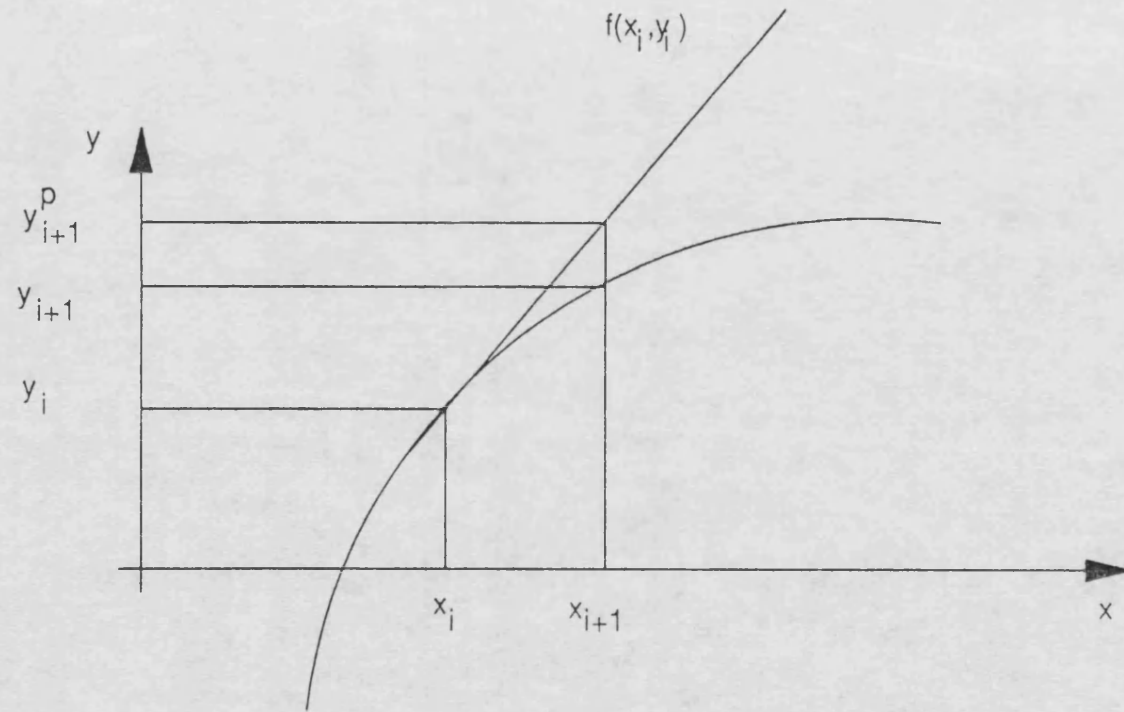


Figure 5.1 Graphical representation of Euler method

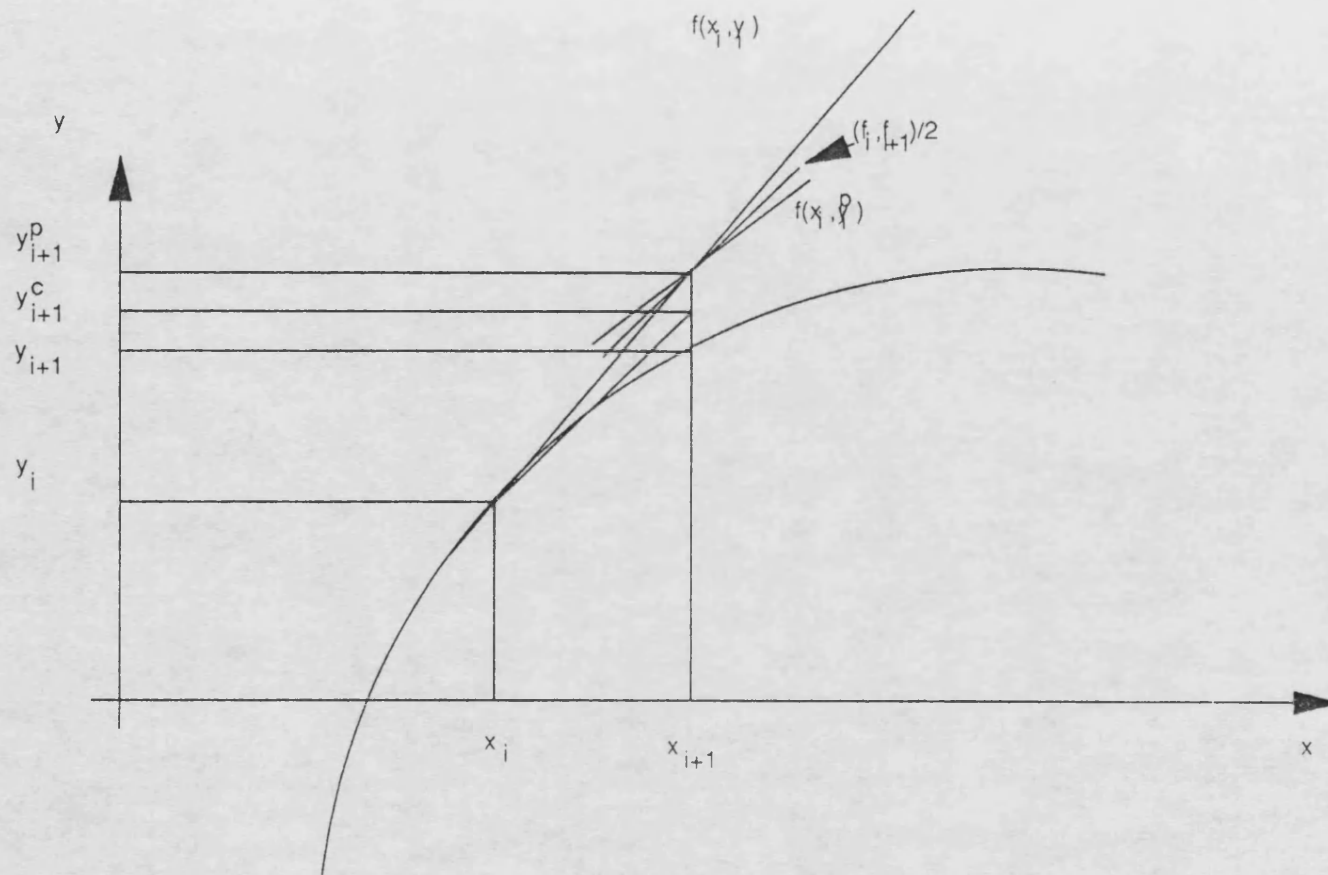


Figure 5.2 Graphical representation of Modified Euler Predictor Corrector method

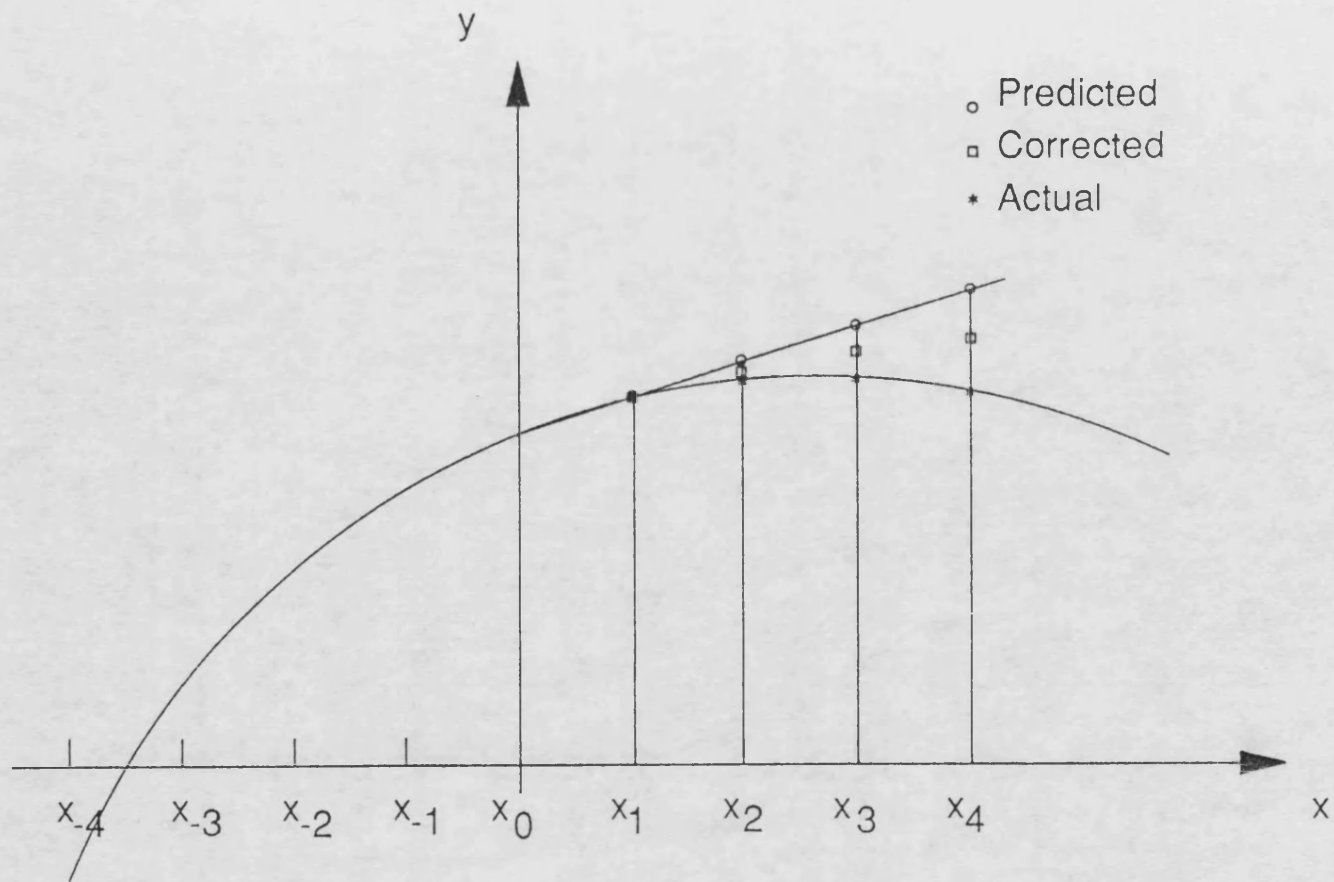


Figure 5.3 Graphical representation of Block Implicit One Step method

```

main()
{
    initialise();
    for (cycle=0; cycle<max_cycles; cycle++)
    {
        for (ca=CaCycleStart;ca<CaCycleEnd;ca=ca+CaCycleStep);
        ca1 = ca;
        ca2 = ca + CaCycleStep;
        castep = CaCycleStep;
        castep5 = castep/2;
        castepend = ca2;
        while(ca1 < castepend)
        {
            predict();
            calculate_cylinder_volume();
            for (loop=0;loop<max_loops;loop++)
            {
                calculate_gas_properties();
                calculate_junction_flows();
                correct();
                find_stability()
                if(stable) break loop;
            }
            if (stable)
            {
                ca1 = ca2;
                ca2 = ca1 + castep;
                advance_point();
            }
            else (unstable)
            {
                castep = castep5;
                castep5 = castep/2;
                ca2 = ca1 + castep;
            }
        }
        accumulate_cycle_results();
    }
    output_results();
}

```

Figure 5.4 The single cylinder MEPCM algorithm

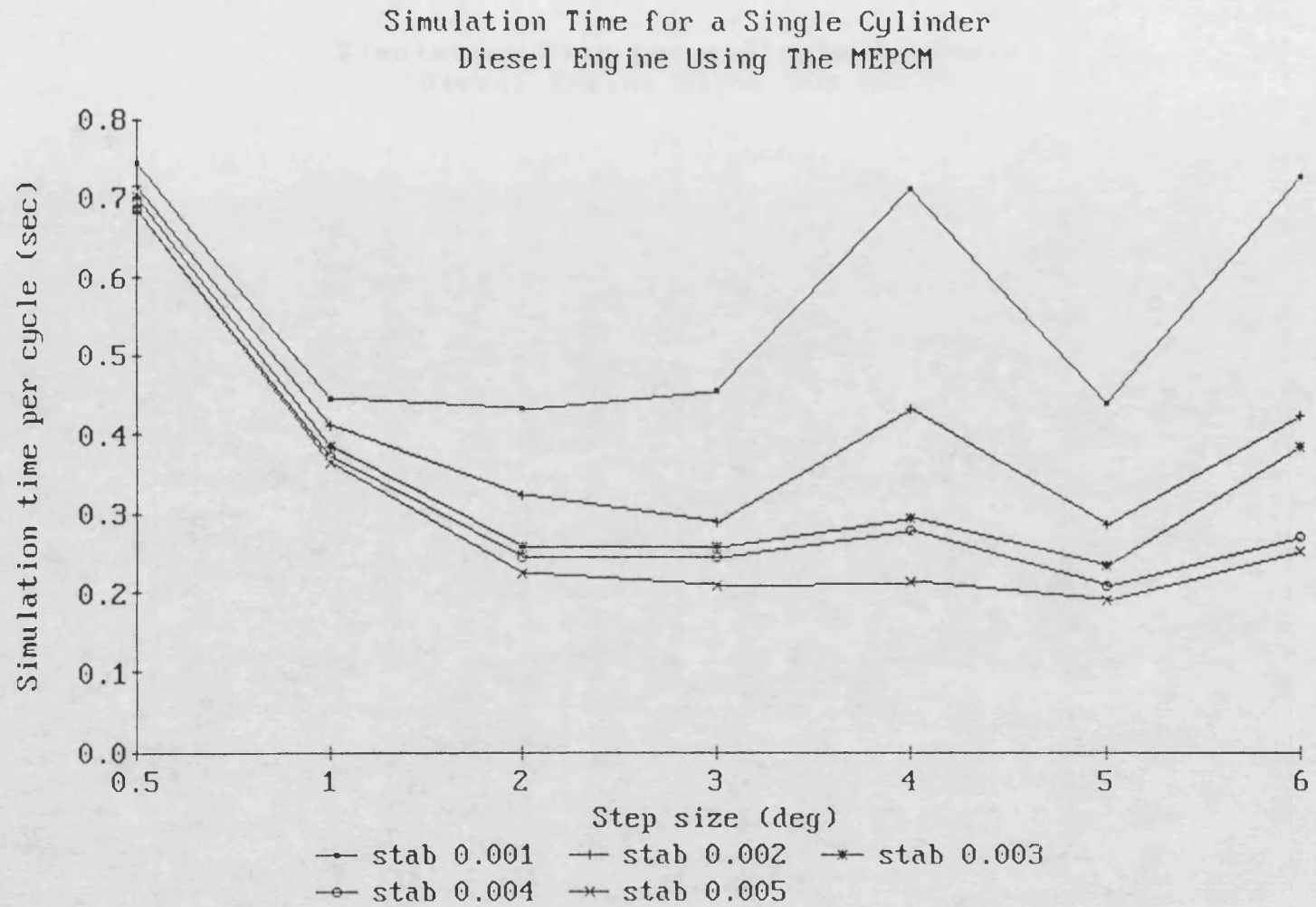


Figure 5.5a Simulation time as a function of stepsize for different stability values  
(The MEPCM with tables)



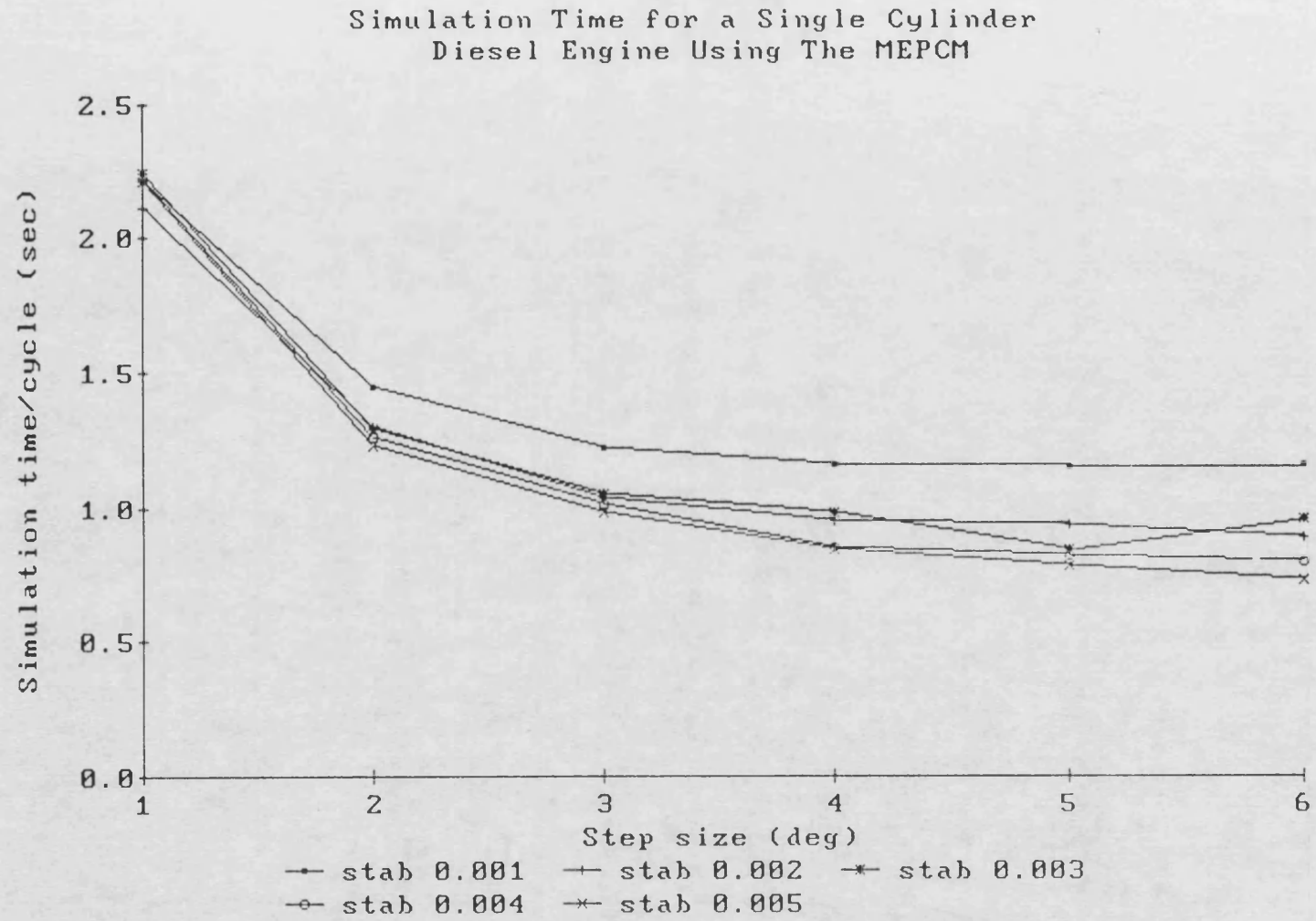


Figure 5.5b Simulation time as a function of stepsize for different stability values

(The MEPCM with no tables)

The MEPCM  
Pressure Plot

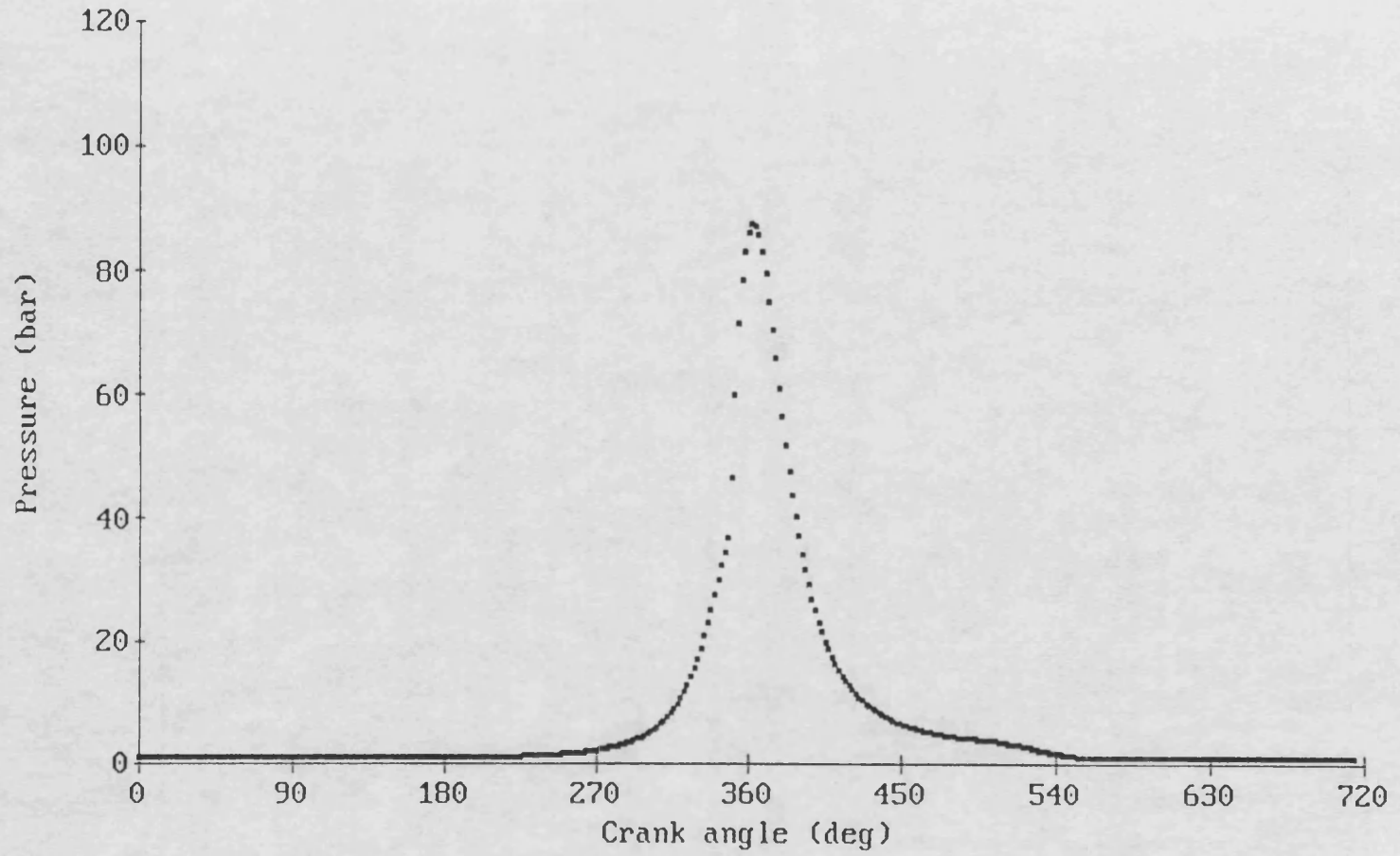


Figure 5.6a Pressure plot for the MEPCM

The MEPCM  
Temperature Plot

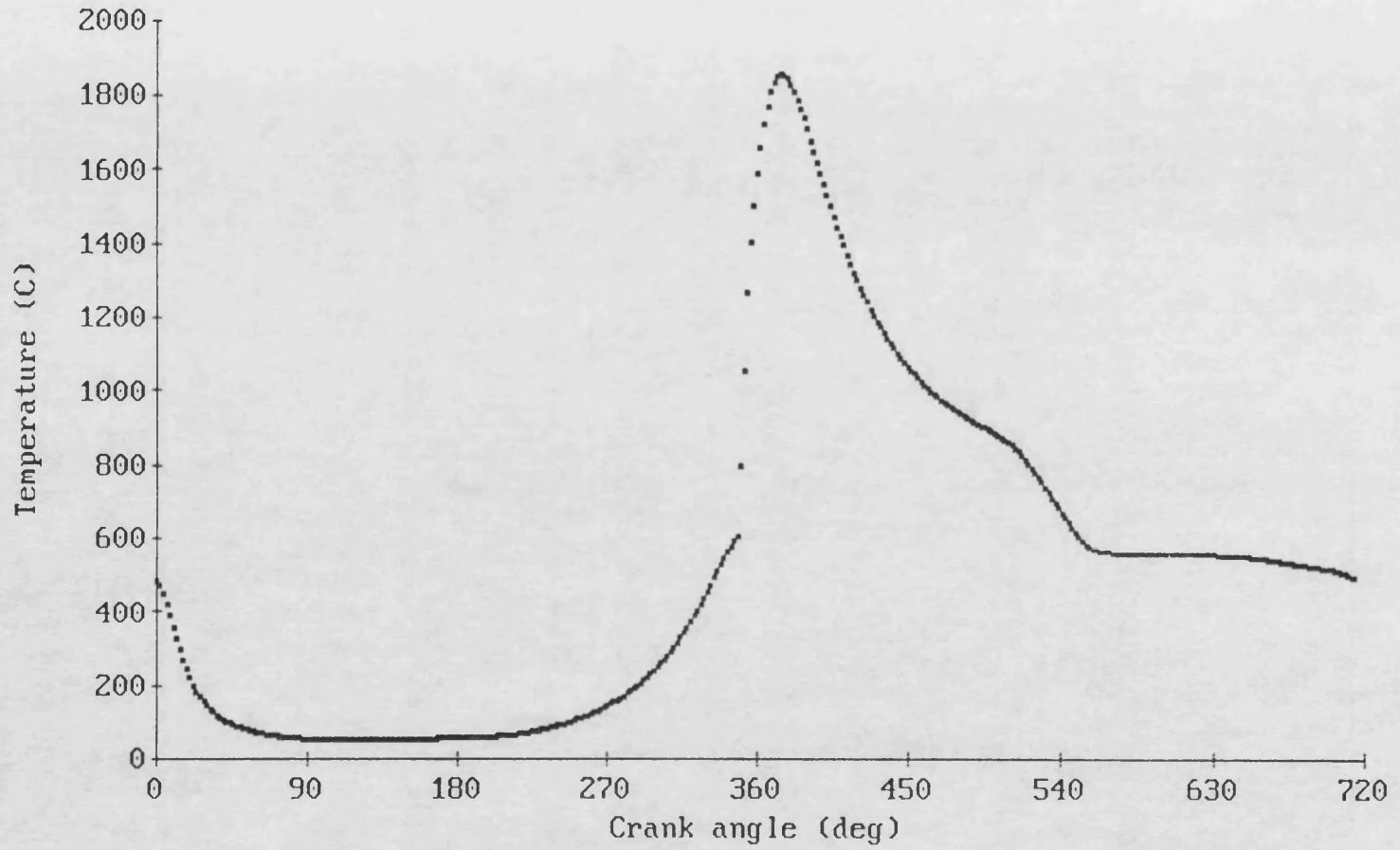


Figure 5.6b Temperature plot for the MEPCM

The MEPCM  
Fuel To Air Ratio Plot

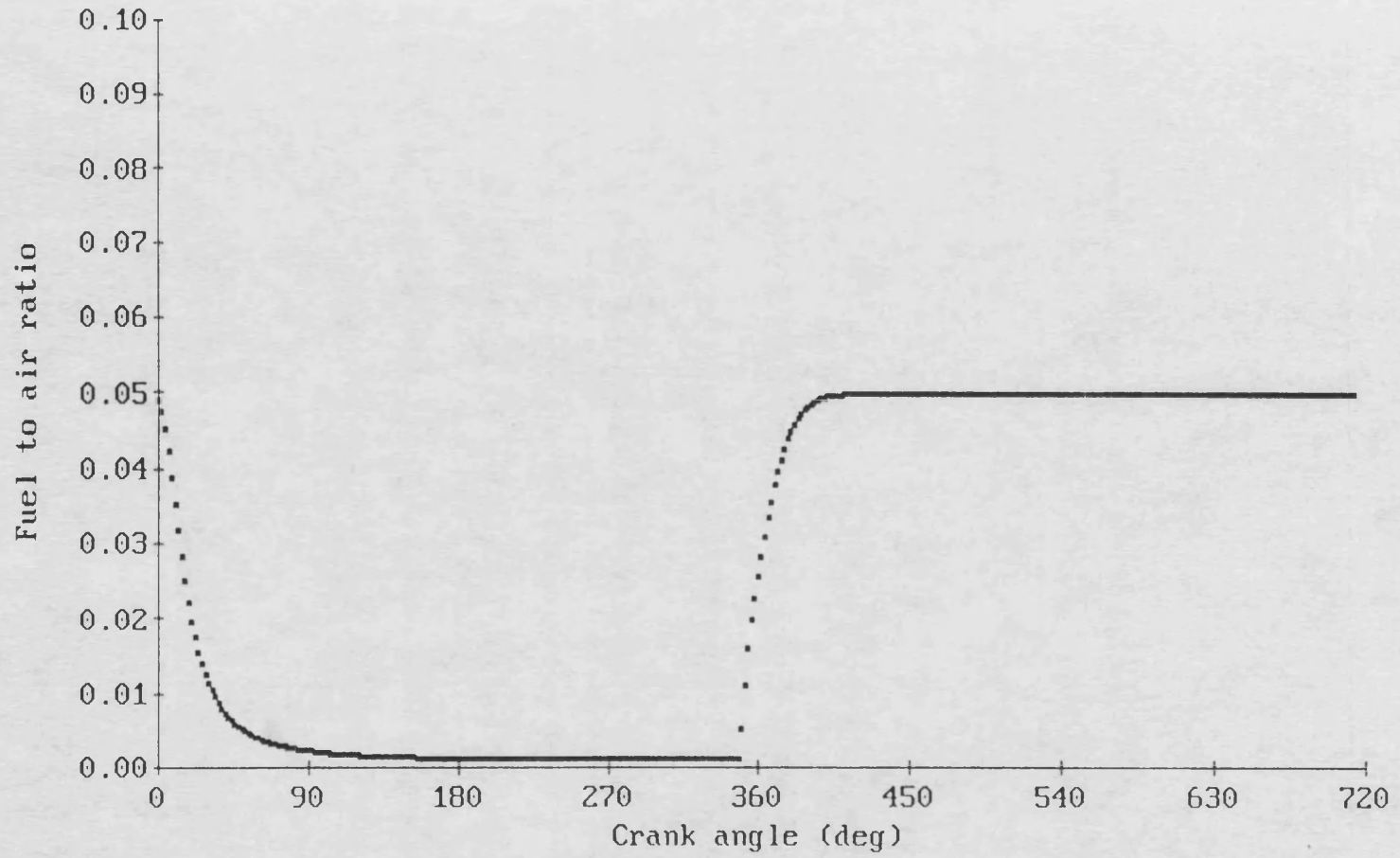


Figure 5.6c Fuel to air ratio plot for the MEPCM

The MEPCM  
Mass Plot

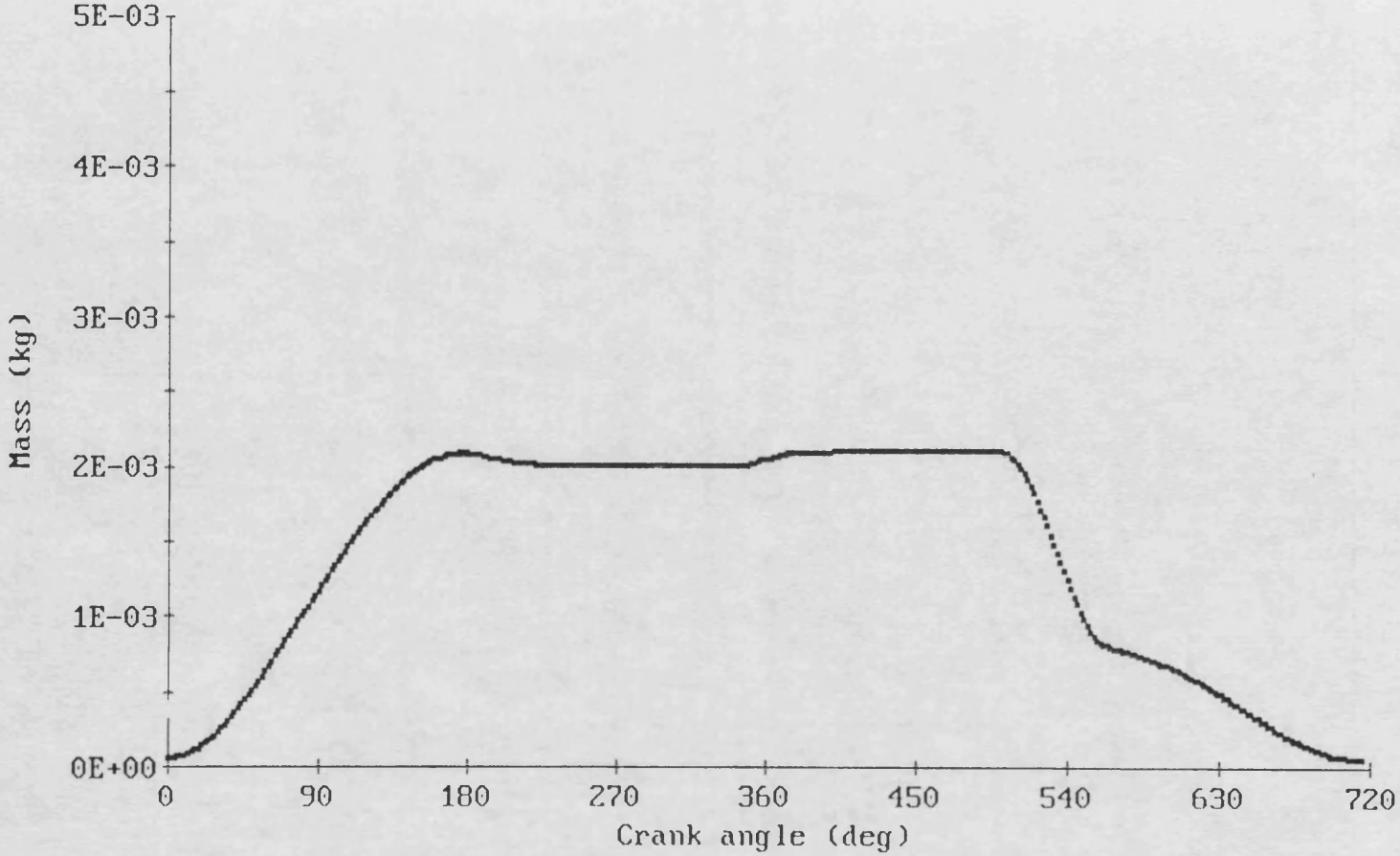


Figure 5.6d Mass plot for the MEPCM

```

biosm
{
  get step size for the current cylinder section();
  while (base angle of block <= section stop angle)
  {
    get  $y_0$ ();
    calculate t (&ca) for the current processor();
    predict();
    find predicted slopes();
    synchronise();
    correct();
    find corrected slopes();
    synchronise();
    advance base t (&ca) for the current processor();
    handle any outstanding packets();
  }
}

```

Figure 5.7 Single cylinder BIOSM algorithm

The BIOSM  
Pressure Plot

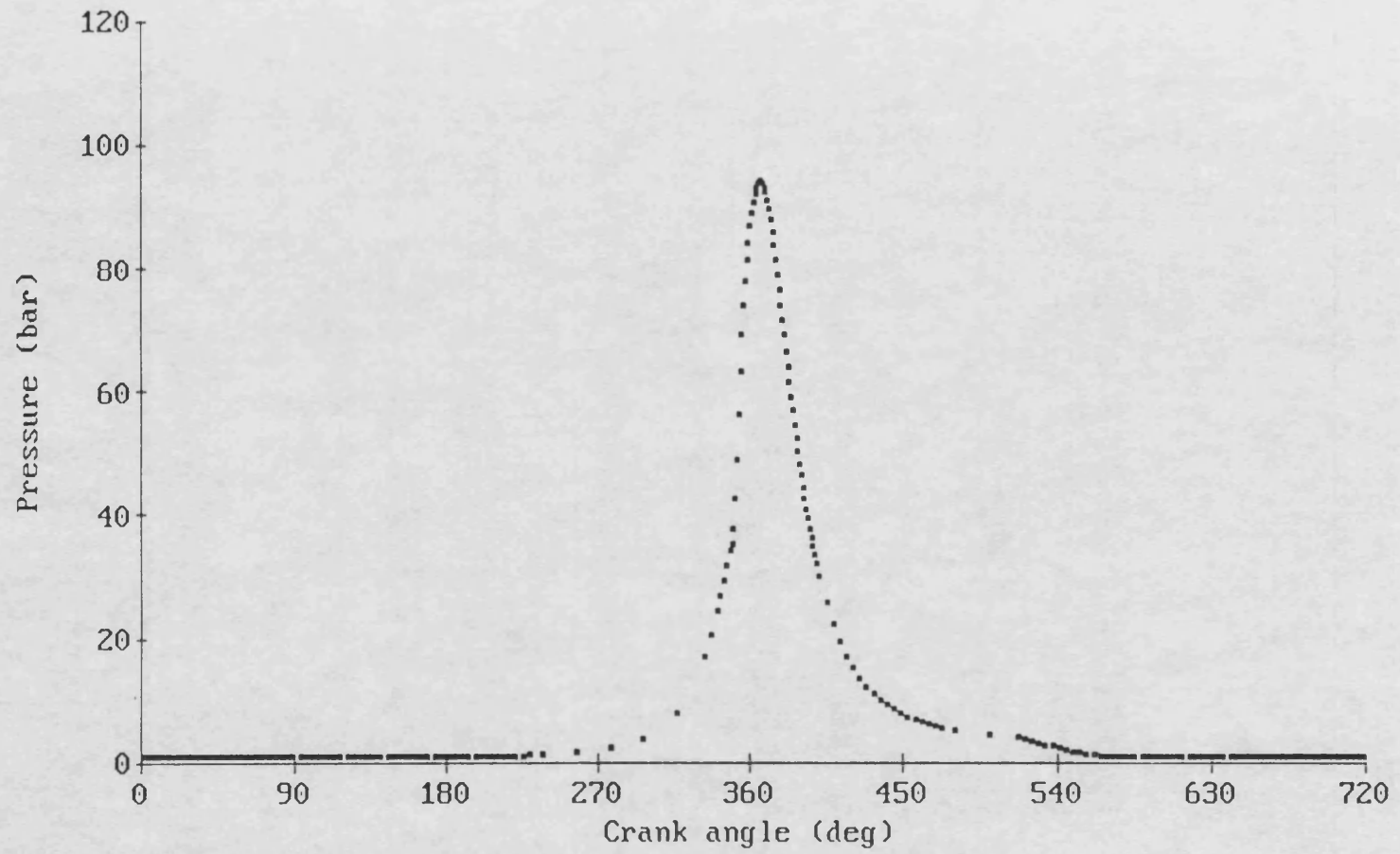


Figure 5.8a Pressure plot for the BIOSM

The BIOSM  
Temperature Plot

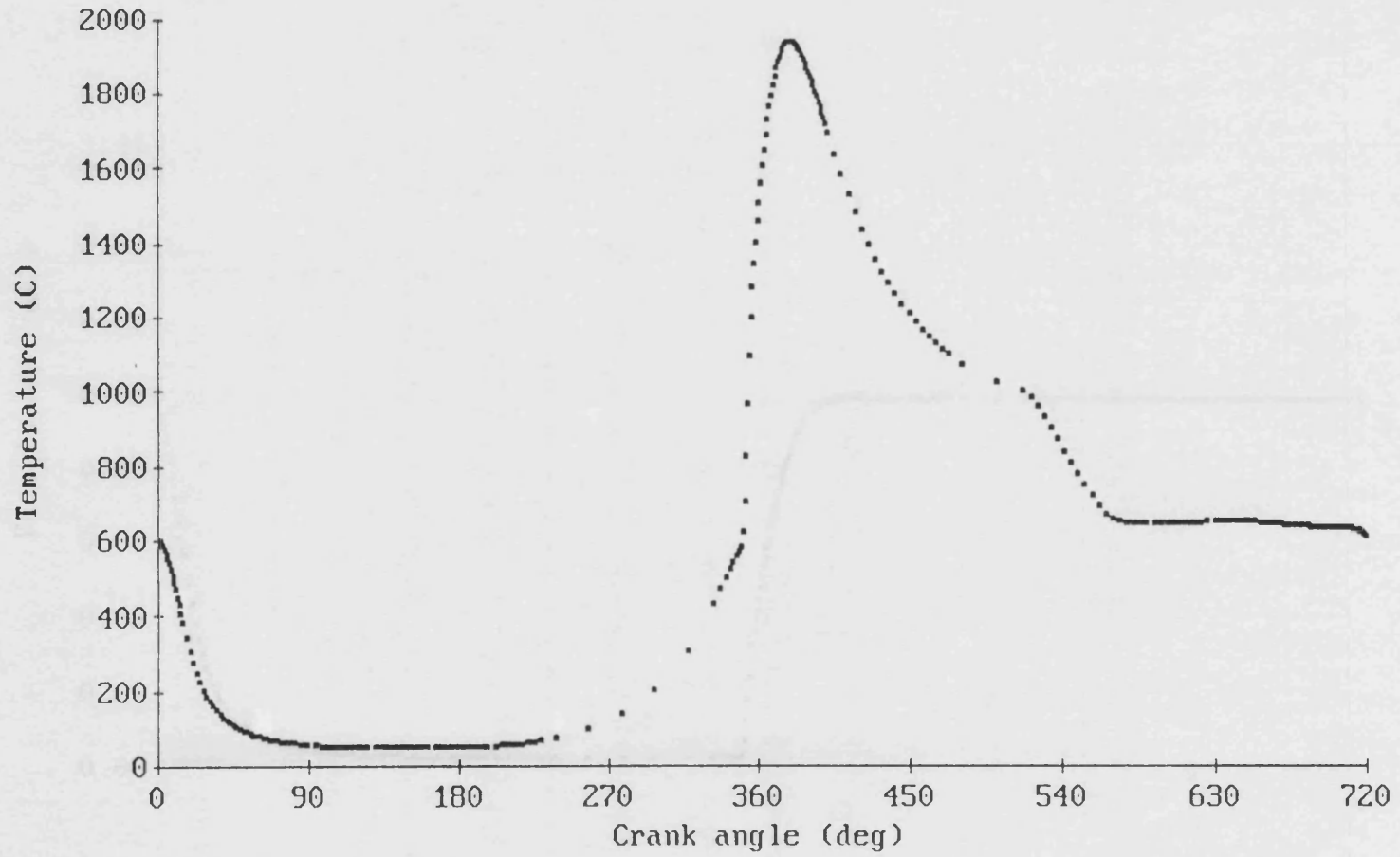


Figure 5.8b Temperature plot for the BIOSM



The BIOSM  
Fuel To Air Ratio Plot

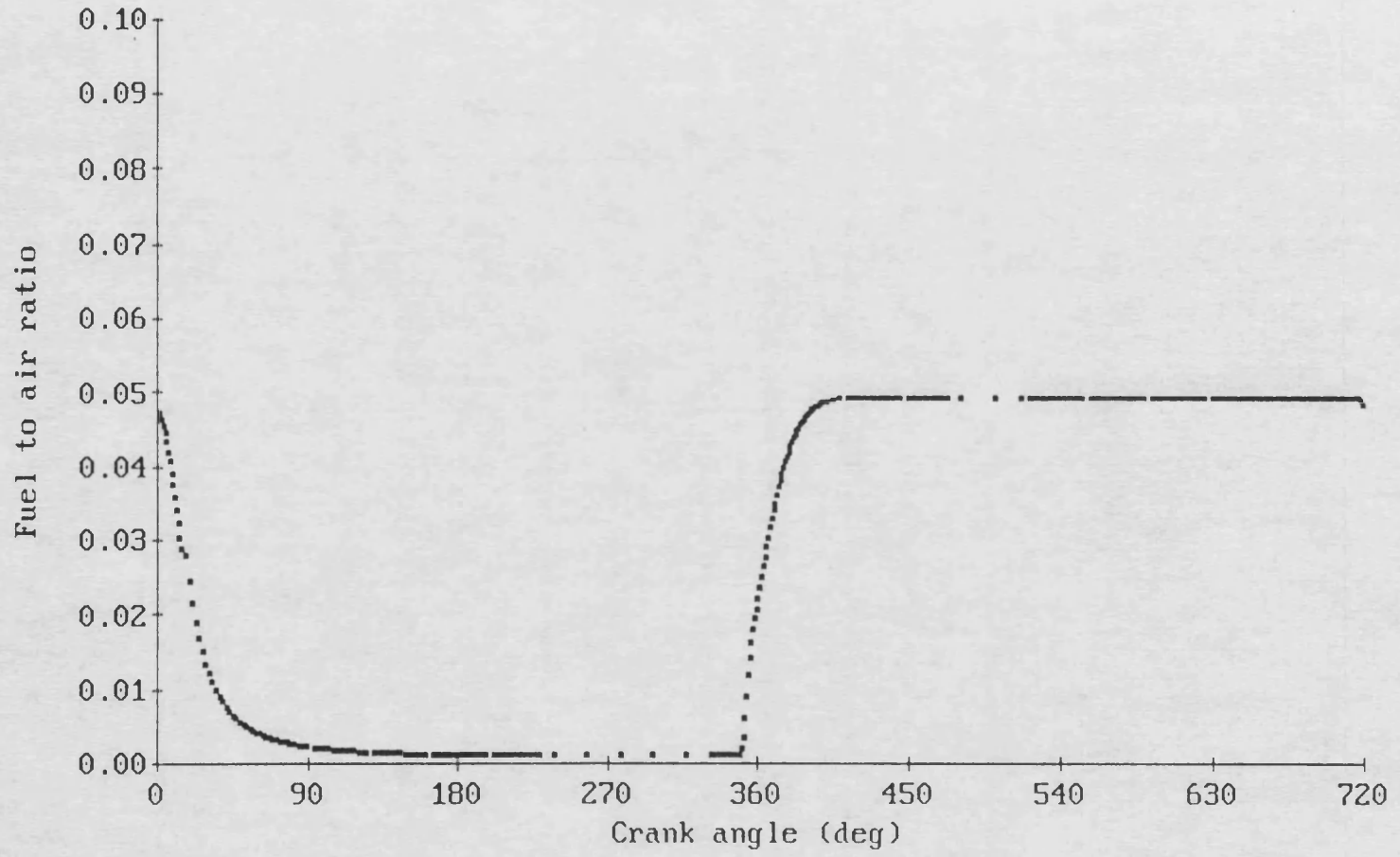


Figure 5.8c Fuel to air ratio plot for the BIOSM

The BIOSM  
Mass Plot

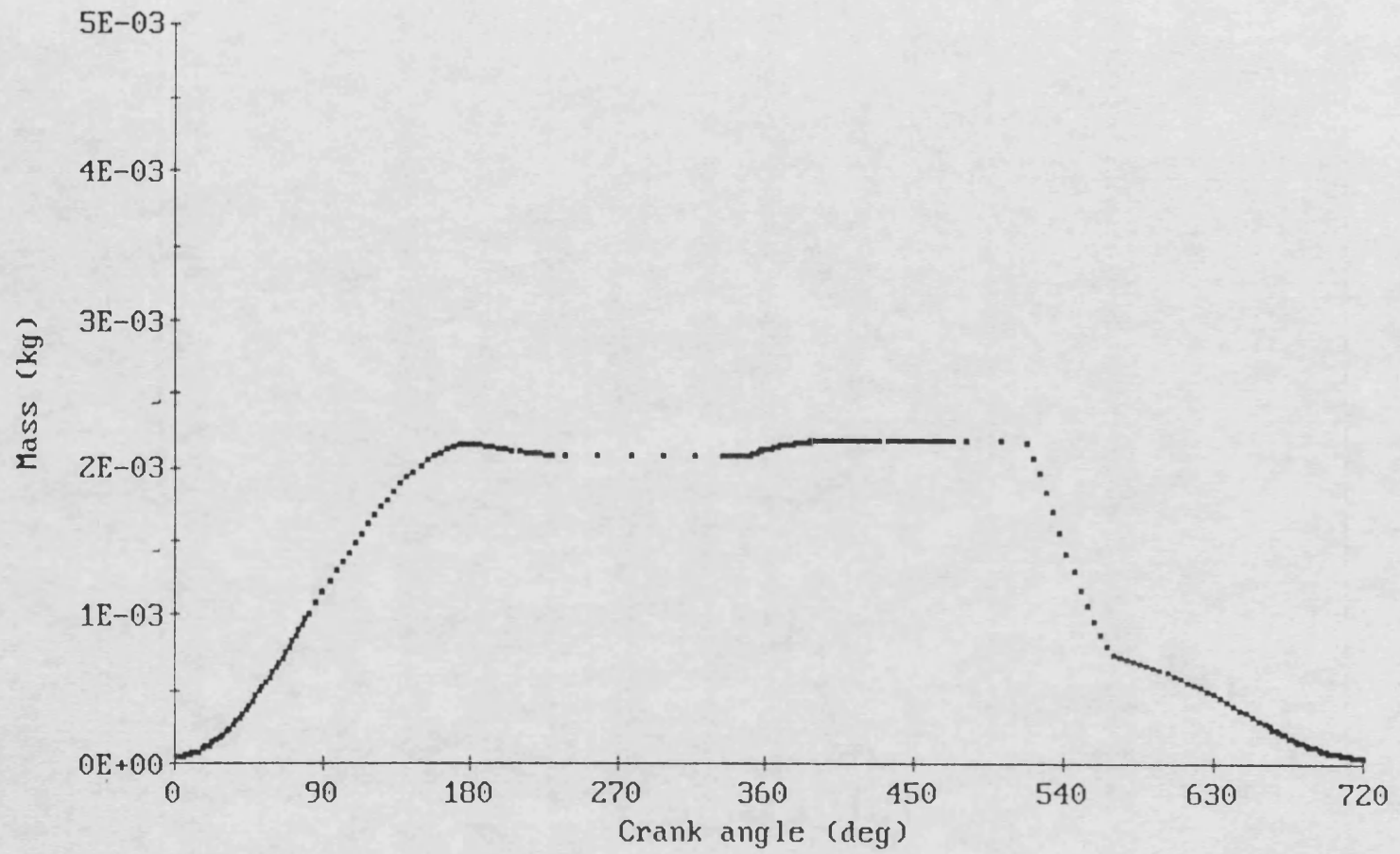


Figure 5.8d Mass plot for the BIOSM

```

bipcm
{
  get step size for the current cylinder section();
  while (base angle of block <= section stop angle)
  {
    get current slopes();
    get  $y_0$ ();
    while (unstable)
    {
      calculate t (&ca) for the current processor();
      predict();
      find predicted slopes();
      synchronise();
      correct();
      find accuracy and  $R_{max}$ ();
      synchronise();
      set variables for stability test();
      calculate new step length();
      calculate new Bmatrix();
    }
    find corrected slopes();
    advance base t (&ca) for the current processor();
    handle any outstanding packets();
  }
}

```

Figure 5.9 Single cylinder BIPCM algorithm

The BIPCM  
Pressure plot

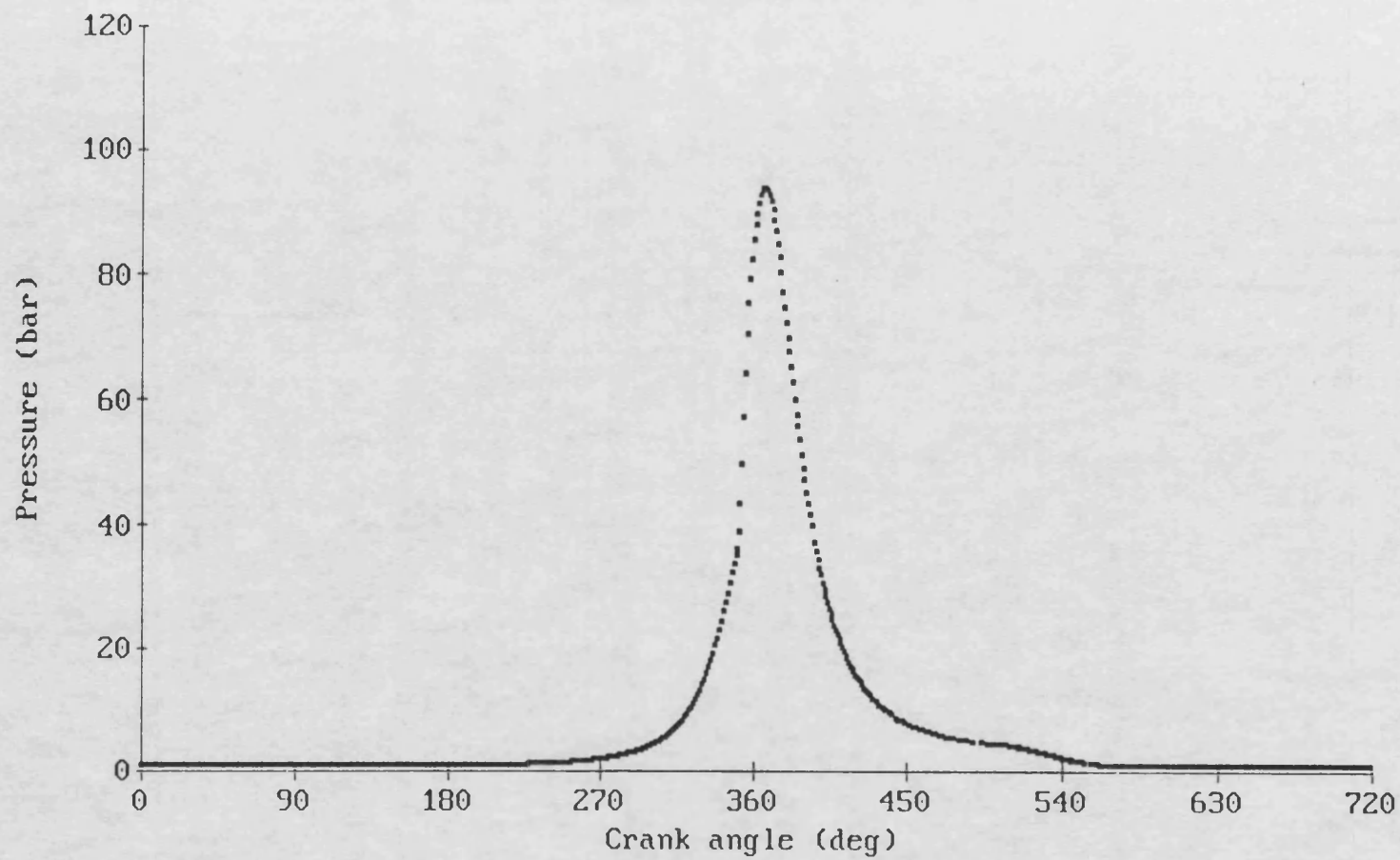
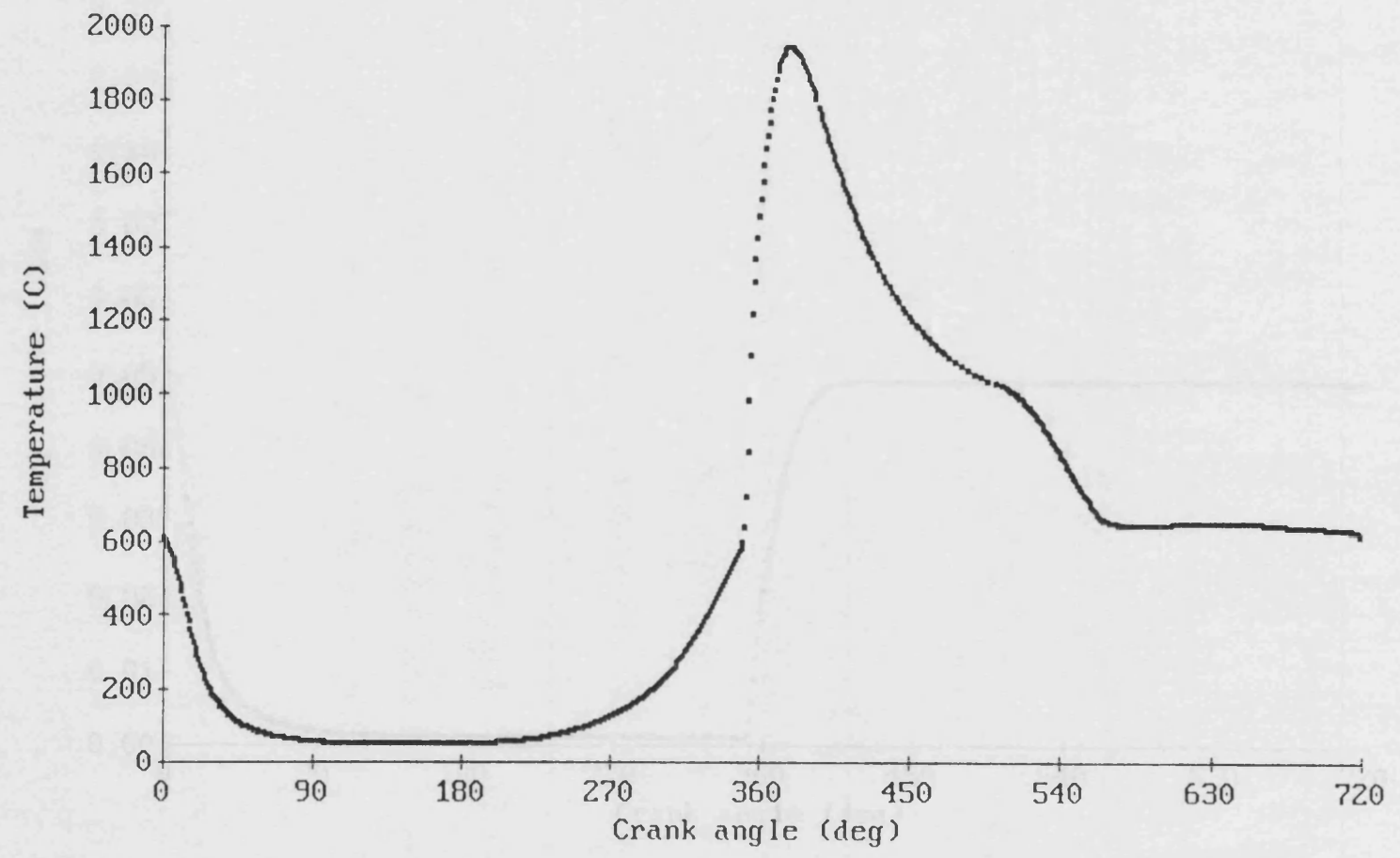


Figure 5.10a Pressure plot for the BIPCM

The BIPCM  
Temperature Plot



140

Figure 5.10b Temperature plot for the BIPCM

The BIPCM  
Fuel To Air Ratio Plot

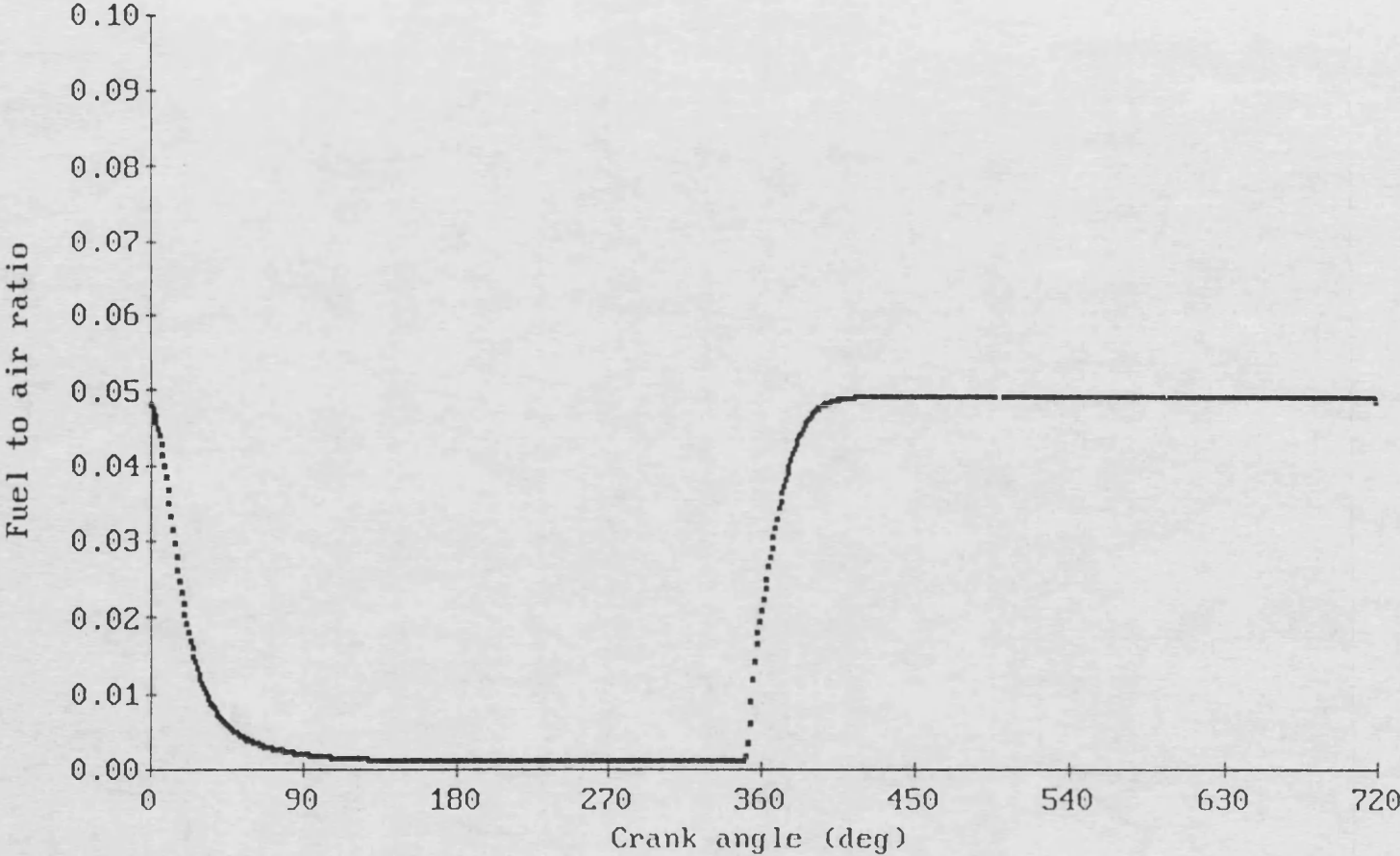


Figure 5.10c Fuel to air ratio plot for the BIPCM

The BIPCM  
Mass Plot

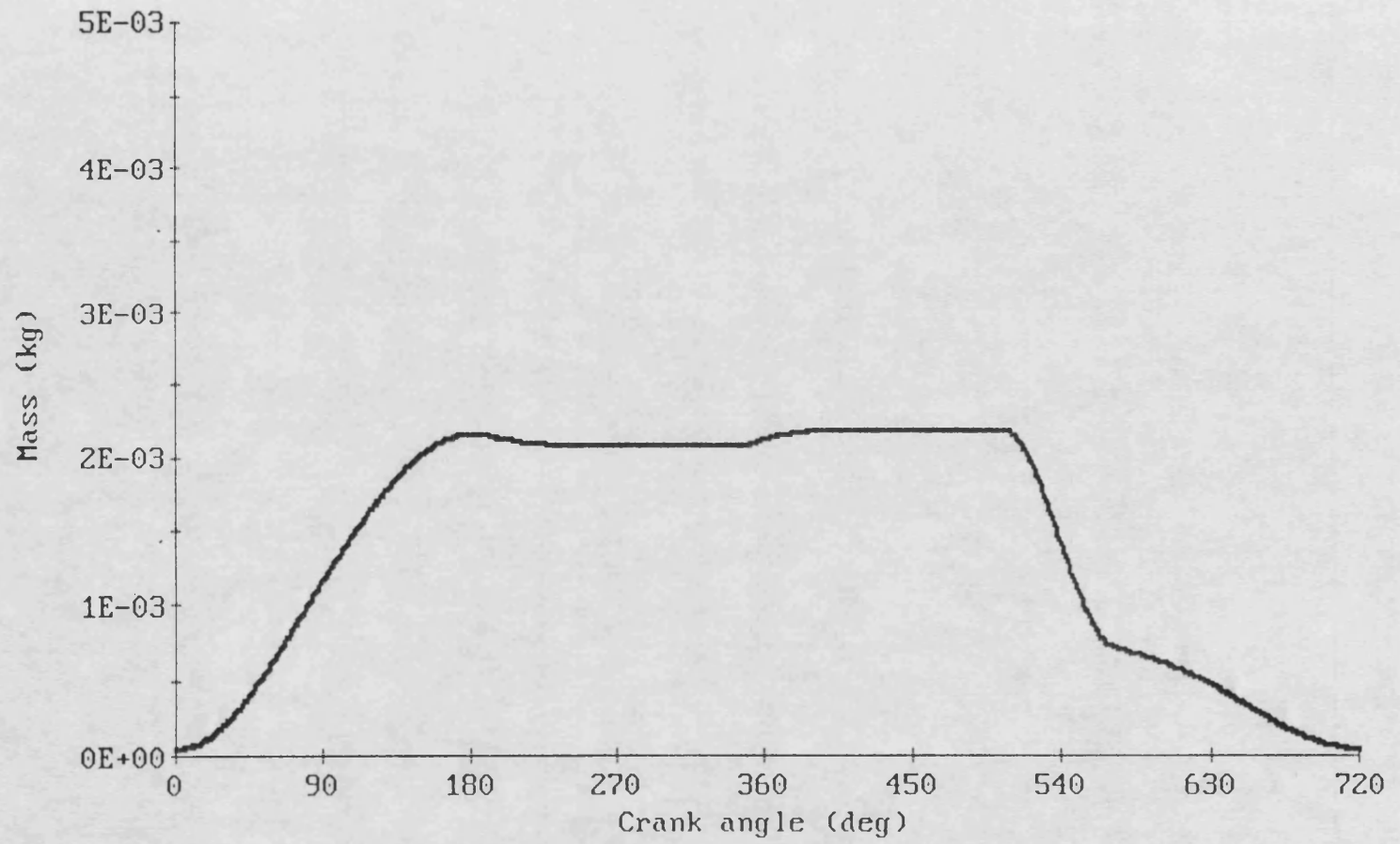


Figure 5.10d Mass plot for the BIPCM

# Chapter 6



# A Multi Cylinder Diesel Engine Simulation

## 6.1 Introduction

In this chapter the simulation of a six cylinder turbocharged Diesel engine on the BUTPC is described. The integration algorithm used is the MEPCM and a communication scheme called distributive iteration step communication (DISC) due to Haysom [21] is employed for data synchronisation within the simulator. Different tasks that constitute the parallel Diesel engine simulator (PDESIM) are also described. At the end of the chapter, the performance of the simulation is discussed.

## 6.2 The DISC

The DISC algorithm under the MEPCM views a Diesel engine as a control system consisting of cylinder and manifold control volumes which are interconnected via valves and which transfer power via shafts. This provides an inherent 'geometrical' parallelism in the system, divided into cylinders, manifolds and shafts. The PDESIM consists of six cylinder tasks, three manifold tasks, one dynamics task, one actuator task and one stability task.

The cylinder task is responsible for the calculation of cylinder processes, such as compression, combustion, etc. It also calculates the flows through valves connected to it. The manifold task calculates the states of the gases inside the manifold and is also responsible for the flow calculations of any compressor or turbine connected to it. This division of junction calculations between cylinders and manifolds provides overlapped calculation and hence a parallel path. There are some dynamic processes inside a cylinder, such as fuel injection timing and the amount of mass of fuel injected

into the cylinder, which are separately calculated by a task called the actuator task. The dynamics task is responsible for the calculation of shaft speeds and also provides an estimate of the cumulative torque produced by the engine shafts. The last task, the stability task, does not contribute to any system calculations but it tracks the system stability and keeps all the other tasks, which require system stability for their progress, informed about the latest stability situation in the system.

The main features of the DISC implementation are:

- o It has no centralised task for data communication between different tasks. All communication is distributed throughout the system and is point to point. For example, if a cylinder task wants some data from a manifold, it puts the request directly to that manifold and reads from its memory when the data is ready.
- o Data communication occurs only between those tasks which are physically connected at that point, for example, when a junction valve is open and gas is flowing from the inlet manifold to a cylinder. If the valve is closed, then both the cylinder and the manifold do not require any data from each other, so no communication occurs during a valve close period.
- o Lookup tables are used for volume, valve flow and heat transfer calculations. Table 6.1 gives a list of the look up tables used in the simulator.
- o The MEPCM is iteratively applied until stability is reached. Step reduction is allowed only if the system does not stabilise

within a given number of iterations. In the case of the PDESIM, this iteration count is fixed at three. Communication between the control volumes occurs at each iteration, for each predictor and corrector, until all the control volumes which are connected together are stable.

Table 6.1 List of Lookup Tables

Cylinder volume
Cylinder surface area
Cylinder rate of change of volume
Cylinder inlet valve effective flow area
Cylinder exhaust valve effective flow area
$(C1HO * \text{cylinder pressure})^{0.8}$
$(\text{Average piston speed} * C2HO)^{0.8}$
$(\text{Cylinder volume})^{0.6}$
$(\text{Cylinder temperature})^{0.4}$

Figure 6.1 shows a connection diagram of the PDESIM. The following section describes how PDESIM is implemented on the BUTPC, which is a transputer based parallel computer running the operating system Helios.

### 6.3 The PDESIM on the BUTPC

The PDESIM is implemented on the BUTPC in a master-slave mode. A master or supervisor task called 'engine' is responsible for the initial creation of all the cylinder, manifold, actuator, dynamics and stability tasks.

The engine task reads a data file which provides initial information about the tasks it is going to create. Then it creates all the remote tasks and writes the initialisation data required by each task to setup its local variables. Each remote task then sends back information about itself which is copied to all other tasks in the form of a table. Then information about shared variables is exchanged and the simulation is started once all the remote tasks are ready. Figure 6.2 shows a distribution of shared data across different tasks.

When initialisation is complete, the master task enters an interactive routine which helps the user to extract useful information from the PDESIM.

### **6.3.1 Task Creation and Initialisation**

In the PDESIM, each remote task is loaded onto a separate processor. The processor name string is provided in a data file called 'proc-alloc'; an example processor allocation format is given appendix B. The file starts with a comment and has three columns of information. The first column is ignored as it is there only to help identify a task by a name. The next column gives the actual name of the task and the last column provides the processor name string on which the task has to be loaded. The engine task is loaded on any processor other than those given in this list.

A special purpose 'createtask' routine is used to create, load and start a task onto a remote processor. The remote task, once started, is sent initial data by copying the data across via Helios ports, which are created between the remote task and the engine task especially for this purpose. Once initialised, all data communication is done using shared memory primitives, Qpkt and Taskwait, described in chapter 4.

### **6.3.1.1 'createtask', 'reply' and 'gettxport'**

The 'createtask' routine first locates the processor manager of the processor at which the task is to be loaded. Then it locates the task itself and starts it on the remote processor. After starting the task, information about the input and output environment is sent to the task via Helios streams.

Once a remote task is started, it first prepares the processor to get access to the backplane by clearing the 'eventretry' register to zero, so that a maximum of two hundred and fifty six tries may be possible. Then it allocates a local Helios port and sends it back to the parent task using the routine 'reply'. The parent task receives the information about the remote port using the routine 'gettxport' and stores it in a table so that data could be sent to the remote task via this port later on. Figure 6.3 gives flow diagrams for these routines.

### **6.3.1.2 'getinfo' and 'putinfo'**

These two routines are used to exchange data between two tasks using Helios ports. The 'getinfo' routine waits on a port corresponding to the remote task and reads data directly into the data area, a pointer to which is provided as an argument to the routine. The 'putinfo' routine copies data directly to the remote task via the transmitter port. The flow diagrams for these routines are given in figure 6.4.

## **6.3.2 Data file Format**

All previous attempts to simulate a parallel Diesel engine by Jones [20] and Haysom [21] were oriented towards a six cylinder diesel engine simulation with embedded data for a Leyland TL11 turbocharged Diesel engine. In the present research, a separate data file format was adopted in order to facilitate data entry and also to avoid

recompilation of part or all of the code when a minor change in the engine specification was needed. For this purpose, a data format similar to that of SPICE [15] was adopted in order to have a similar feel as that of SPICE. Appendix C describes how a data file is prepared for the PDESIM, and appendix D provides an example data file for the Leyland TL11 turbocharged six cylinder Diesel engine. Table 6.2 gives those parameters which may be modified.

Another feature of the data file is the ease in the selection of different routines for heat release and heat transfer. As described in chapter 2, heat release due to combustion in the cylinder can be calculated in different ways. In the case of PDESIM either of the two methods, the Watson heat release method or the Wiebe heat release method, may be used just by specifying its name at the start of heat release data. Similarly, heat transfer between the cylinder gases and the cylinder walls may be estimated either by the Hohenberg heat transfer model or by the Woschni heat transfer model.

### **6.3.3 Data Communication and Synchronisation**

A two way communication mechanism is used to transfer data between tasks. A communication packet is allocated to each of the two way inter-task transfers, which controls the reading and writing of data between the tasks. As soon as data is ready, a packet is sent to a task which may need this data now or later in the process. Return of that packet from the remote task means that the data has been read by that remote task and it may now be overwritten by the sender.

If a task needs some data from a remote task, it sends a packet to that task and waits for the packet to be returned. It then reads the required data from the memory of the remote task directly.

Table 6.2 Engine parameters that may be modified

Cylinder bore
Cylinder stroke
Cylinder compression ratio
Cylinder connecting rod length
Cylinder offset angle
Cylinder friction mean effective pressure
Manifold volume
Manifold surface area
Manifold heat transfer coefficient
Inlet valve data
Exhaust valve data
Load inertia
Load torque
Shaft speed
Initial
pressure, temperature, and fuel to air ratio
of
cylinders, manifolds, and atmosphere
Integration step size
Tolerance for stability test
Section angles corresponding to
valve open, valve close, and
static fuel injection timing
Combustion duration

In order to make all communication asynchronous, so that overlapped communication and calculation is possible, a flag is maintained for each packet on both the communicating tasks. This flag is set by a packet handler on the arrival of the corresponding packet. The waiting task is only forced to wait if the communication flag is not set.

The cylinder control volume task communicates with all other tasks. It needs information about pressure, temperature, fuel to air ratio and general gas properties of both the inlet and the exhaust manifolds it is connected to. It sends valve flow parameters to the connected inlet and exhaust manifolds. From the actuator task, it reads static fuel injection timing and the amount of fuel injected, and from the dynamics it obtains the speed of the crank shaft. The dynamics task, on the other hand, needs torque produced by the cylinders in order to calculate the engine speed, and turbine torque from the exhaust manifold and compressor torque from the inlet manifold to calculate turbo speed. The actuator task needs only engine speed from the dynamics task. The stability task reads the local stabilities of cylinders and manifolds and returns them system stability.

There are two different step sizes of the system; one is an overall calculation step size which is kept constant and is used to accumulate cycle data at constant intervals, and the other is an intermediate iteration step size which happens whenever system stability fails and a decision is made to reduce the step. Those control volumes which are connected to each other keep on exchanging data during the reduced step. Thus communication between the cylinder and the manifold control volume tasks occurs for every iteration step of the MEPCM, so that only the latest values of the state variables may be used at each point.

The communication between the cylinder and the dynamics tasks occurs at each calculation step. This is because the shaft dynamics have a high integration time



constant, and therefore, are solved by a simple integration formula, given in equation 6.1.

$$y_{n+1} = y_n + h f_n \quad 6.1$$

The actuator equations have also a high time constant and are solved independently. The synchronisation of the actuator task is provided at each calculation step when it reads engine speed from the dynamics task.

The stability task works in a different way. As mentioned earlier, it does not take part in any calculations of the system equations. It communicates only with those cylinder and manifold control volumes which are connected to one another via an open valve. If both the inlet and the exhaust valves of a cylinder are closed, then the cylinder is not connected to any of the manifolds. Therefore it does not communicate with any manifold or with the stability task. Transition between a closed section and an open section occurs when a valve starts to open or close and is described in the next section. Figures 6.5a and 6.5b show flow diagrams of a cylinder control volume when it is in the power stroke and the exhaust stroke respectively. Similarly figure 6.5c shows the flow diagram of an exhaust manifold.

### 6.3.4 Open and Closed Loop Connections

Communication between different tasks of the PDESIM is divided into two main sections depending upon the opening and closing of valves. These are the open loop section when at least one of the valves is open and the closed loop section when both valves are closed.

The point at which cylinder valves open or close is determined by the engine crankshaft angle. The manifold control volume tasks have no absolute knowledge of

the crankshaft angle, therefore, the detection of opening and closing of valves is done by the cylinder control volume tasks.

A valve is presumed to open at the start of the integration step containing the valve opening angle and is presumed to close at the end of the integration step containing the valve closure angle. This is to allow to overlap between the section of the cylinder with the higher number of terms in the state equations, such as scavenge, onto the section of the cylinder with the lesser number of terms in the state equations, such as exhaust or induction.

A cylinder control volume task uses three different packet types to inform a manifold about opening or closing a valve and valve data transfer. These packets are:

- o A communication packet
- o A valve open packet
- o A valve close packet

Each manifold keeps a table for the storage of the cylinder communication packets. When a cylinder has to open a communication channel, it sends a valve open packet a step in advance to the manifold which stores this packet in its communication packet table and increments a counter to indicate that the number of currently open channels has been increased. Communication is established at the start of the next iteration. When a channel has to be closed, a valve close packet sent by a cylinder to a manifold is returned by the manifold after updating a local counter that indicates the number of channels currently closed. So at the start of the next step, no packet is sent to that cylinder whose valve was closed, and the communication channel is closed.

The opening and closing of the communication channels between the cylinder and the manifold control volume tasks is synchronised by sending the valve open or close

packet to the manifold before communicating with the stability task. In this way the manifold task is ensured to receive all the information before it decides to move forward, to loop around or to reduce step and iterate, depending upon the system stability.

## 6.4 Speed Performance of the PDESIM

Three different variables are used to observe the speed performance of the PDESIM. These parameters are given in table 6.3.

Table 6.3:	
Engine speed	1200 rpm - 2000 rpm
Integration step length	0.5°, 1° - 5°
Integration stability criterion	0.1% - 0.5%

Figure 6.6a shows the simulation speed as a function of the integration step length for different stability criteria. The engine speed is 2000 rpm. It gives a peak simulation speed of 140 rpm for a stability criterion of 0.5% and at 2° step length. As the integration step length increases, the simulation speed decreases after reaching a peak value. This is due to a need to reduce intermediate stepsize in the region of greater instability.

In figure 6.6b, a graph of simulation speed as a function of engine speed is plotted for different integration step lengths. The stability criterion is 0.5%. It shows that an optimum step length for an engine speed below 1600 rpm is 1° and that for an engine speed at and above 1600 rpm is 2°.

## 6.5 Summary

A six cylinder turbocharged Diesel engine is simulated on a transputer based parallel computer. The simulation utilizes the distributive memory features of the BUTPC. All data transfer and synchronisation is decentralised and is asynchronous in nature. Data entry to the simulator is made flexible by the use of a data file, which allows the user to see the effects of changes in different parameters on the engine performance without changing actual code. A peak performance simulation speed of 140 rpm is achieved which is 14.3 times 'real time'.

In the next chapter, simulation results from the PDESIM and the FJH model are compared.

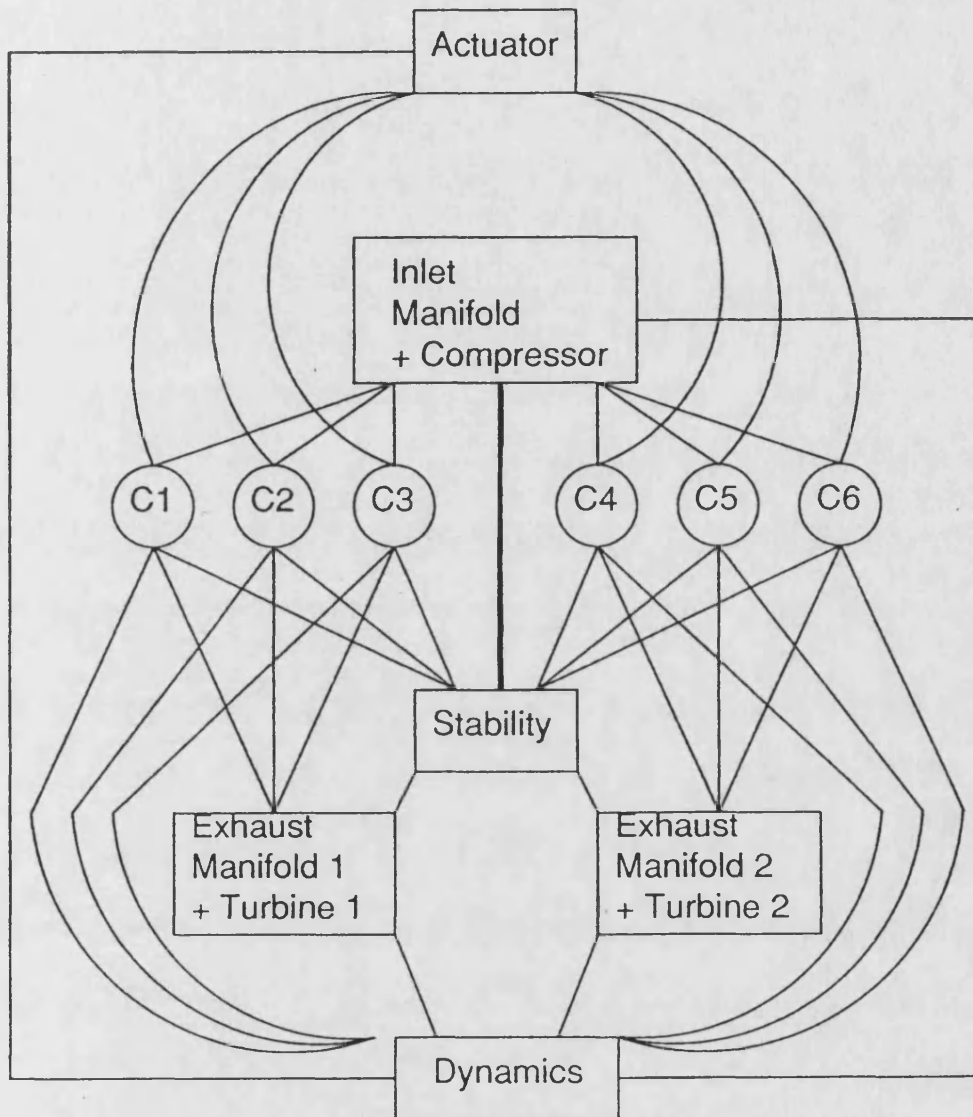


Figure 6.1 The six cylinder PDESIM communication scheme

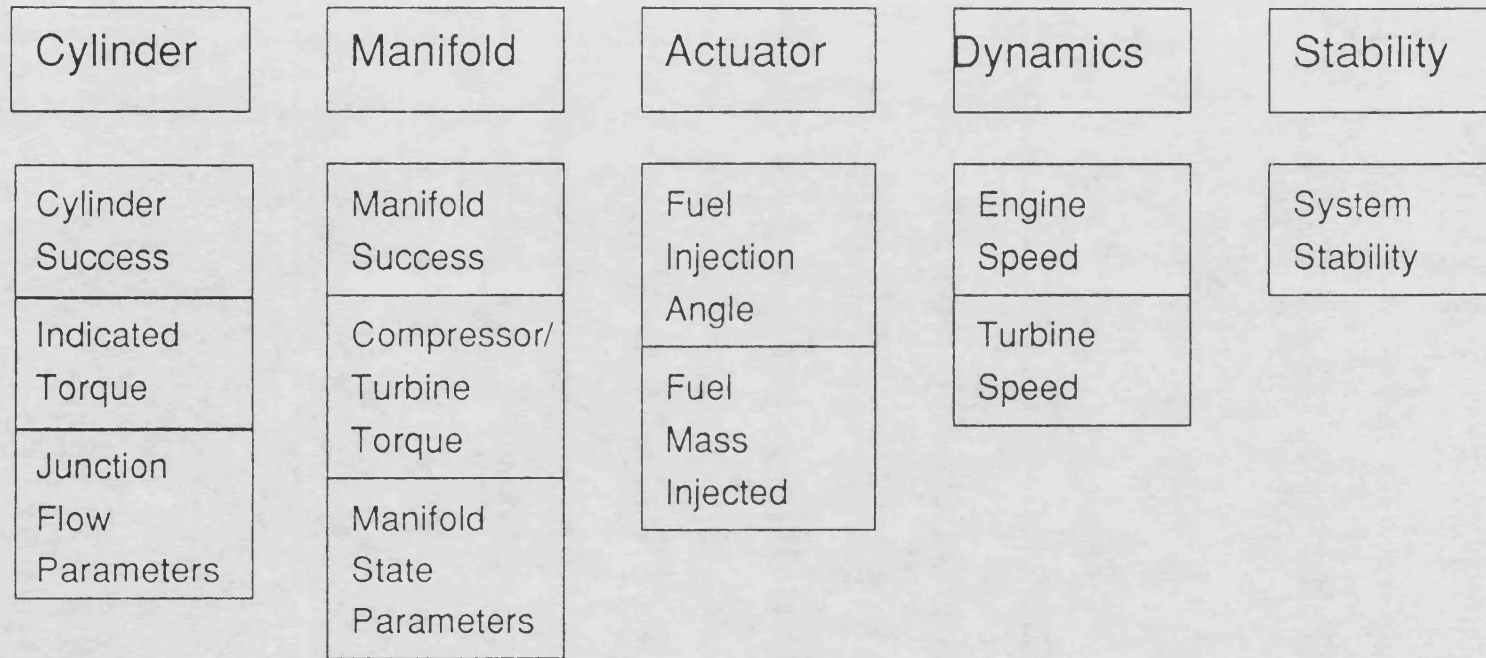


Figure 6.2 Distribution of shared data across the system

```
Object createtask()
{
    construct the name of the processor manager();
    locate the processor manager();
    locate the program to be loaded();
    start the program up();
    construct suitable environment for the new program();
    send environment to the new program();
    return program object();
}
```

Figure 6.3a: 'createtask'

```
reply()
{
    send dummy message to the parent task();
}
```

Figure 6.3b: 'reply'

```
Port gettxport()
{
    get dummy message from the remote task();
    return reply port();
}
```

Figure 6.3c: 'gettxport'



```
getinfo()
{
    get message data from remote task at the given address();
}
```

Figure 6.4a: 'getinfo'

```
putinfo()
{
    send data from the given address to the remote task();
}
```

Figure 6.4b: 'putinfo'

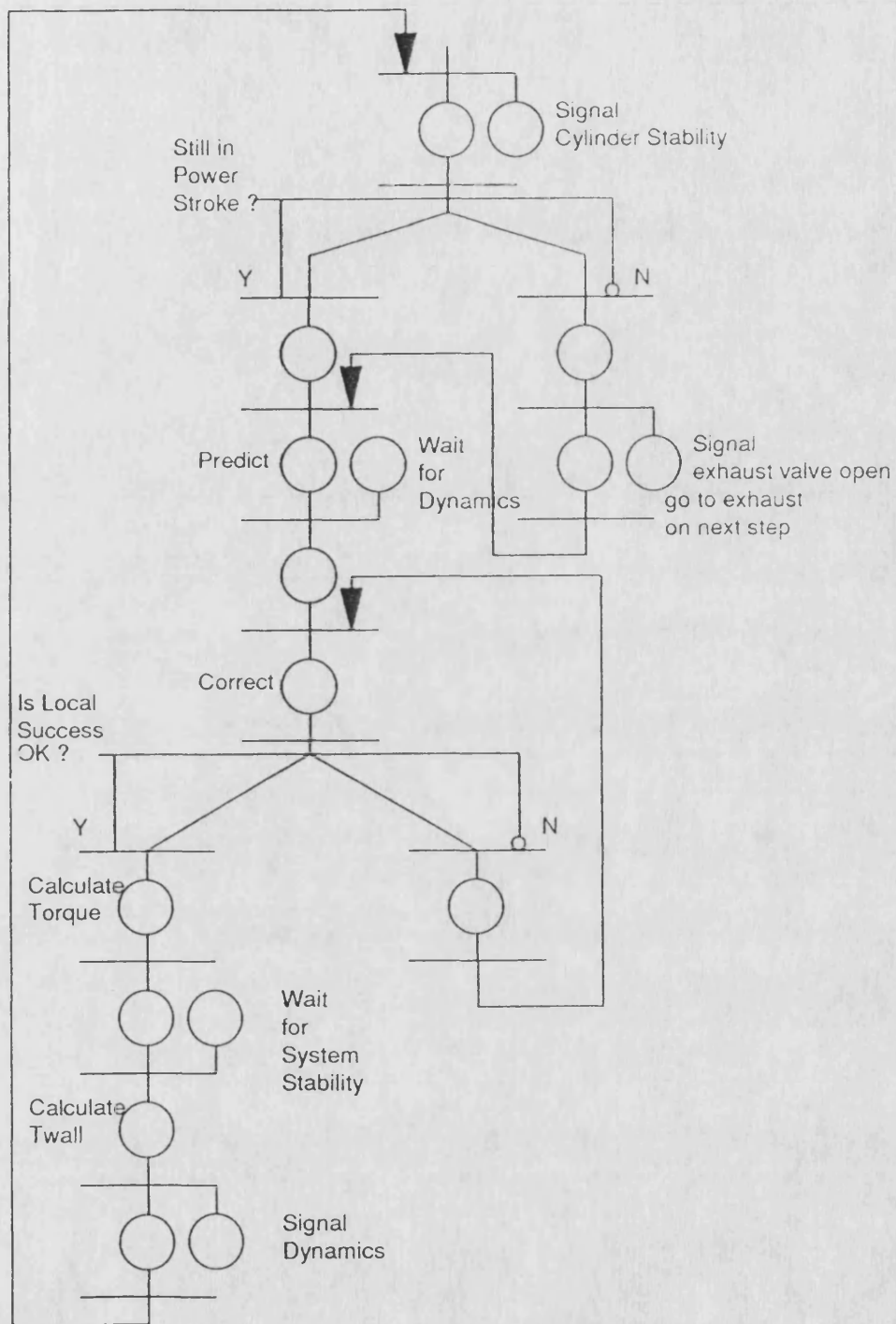


Figure 6.5a Cylinder flow diagram when in power stroke

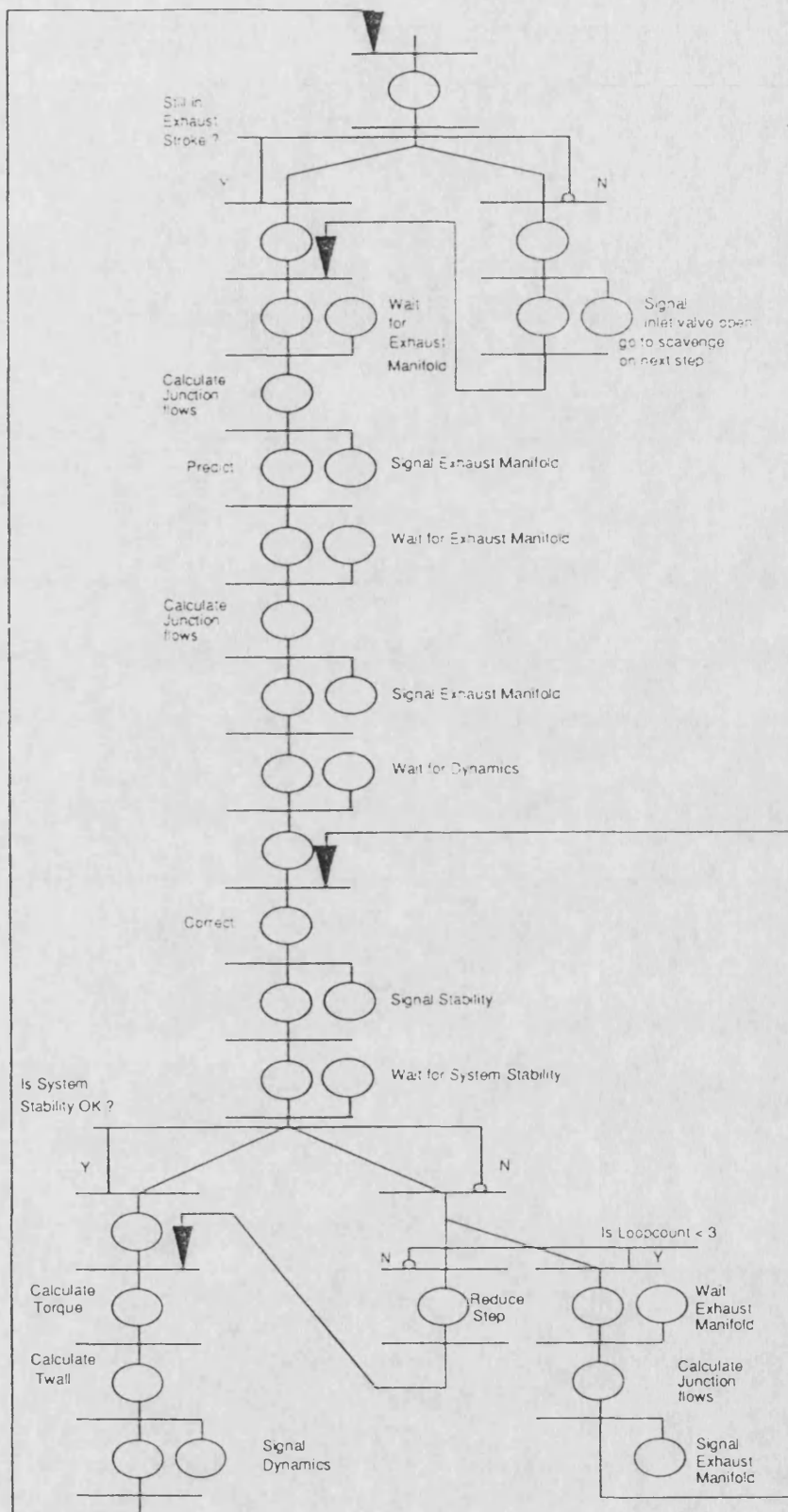


Figure 6.5b Cylinder flow diagram when in exhaust stroke

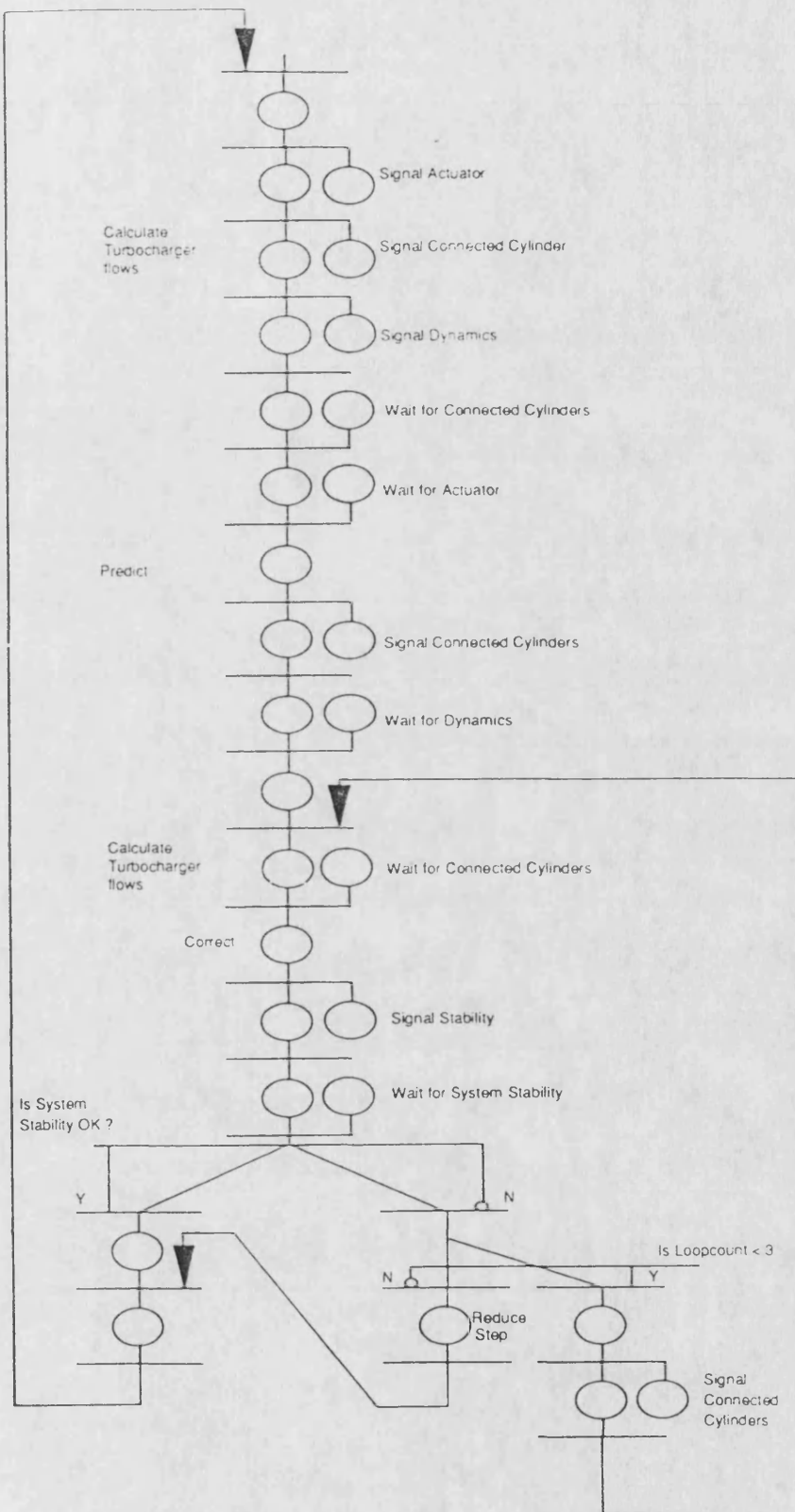


Figure 6.5c Exhaust manifold flow diagram

Parallel Diesel Engine Simulation  
Engine Speed 2000 rpm

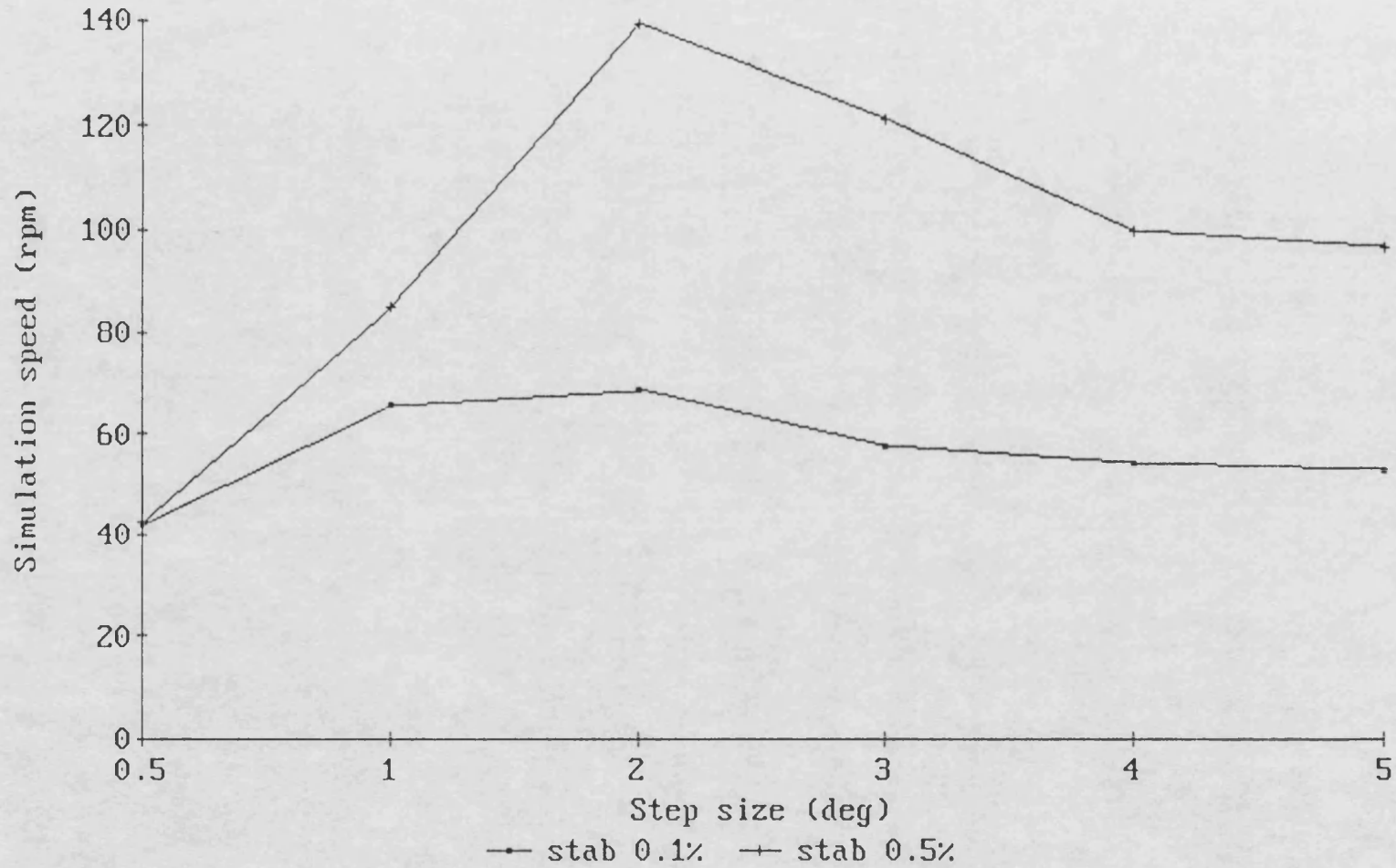


Figure 6.6a The effect of the integration steplength on the simulation speed for stability criteria of 0.1% and 0.5%

Parallel Diesel Engine Simulation  
Stability Criterion 0.5%

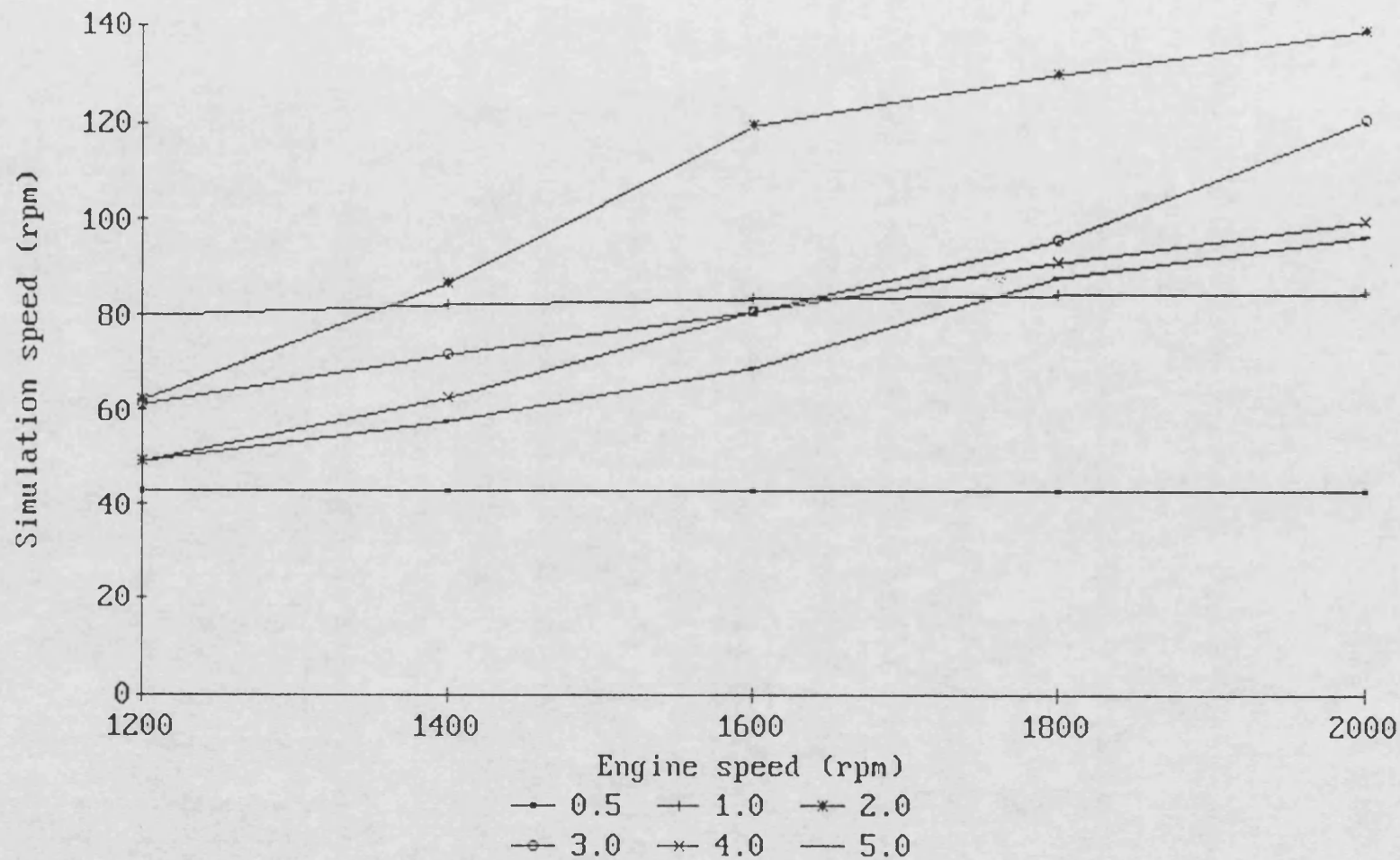


Figure 6.6b The effect of the simulated engine speed on the simulation speed for different initial integration steplengths

# Chapter 7

# **Engine Simulation Results and Their Validation**

## **7.1 Introduction**

This chapter describes different engine output parameters and their calculation over a cycle in the PDESIM. A six cylinder Diesel engine is simulated and the results are compared with those obtained from a similar Diesel engine model due to Haysom [21]. At the end of the chapter, a brief comparison is given between the output results of an experimental Leyland TL11 turbocharged Diesel engine and the simulated results obtained from the PDESIM.

## **7.2 The Engine Cycle Parameters**

An internal combustion engine is characterised by a number of parameters. These parameters describe the overall performance of the engine. In general, it is of interest to know about the power generated by a particular engine and its efficiency. To get this information, a number of engine variables are collected. These variables are of two main types; actual engine variables which define the instantaneous engine state such as pressure, and temperature, together with overall engine parameters which define the performance of the engine such as power and fuel consumption. The overall performance parameters are derived from the engine instantaneous variables. The following sections describe these parameters.



## 7.2.1 The Engine Cycle Variables

The engine cycle variables are spread across the engine system. Some of these belong to the cylinders and some belong to the manifolds and shafts. These variables are of three different types, average, maximum and accumulative over the engine cycle.

In the PDESIM, each cylinder control volume task collects the following cycle information.

- o Open cycle work in joules

This is the amount of indicated work done by a piston during the gas exchange strokes.

$$W_{oc} = \sum_{BDC_{exh}}^{BDC_{ind}} (P_{cyl}) \frac{dV_{cyl}}{d\theta} \Delta\theta \quad 7.1$$

- o Closed cycle work in joules

This is the amount of indicated work done by a piston during compression and expansion strokes.

$$W_{cc} = \sum_{BDC_{ind}}^{BDC_{exh}} (P_{cyl}) \frac{dV_{cyl}}{d\theta} \Delta\theta \quad 7.2$$

- o Mass flowing through inlet valve in kg

This is the total mass of gas entering the cylinder through the inlet valve during a cycle.

$$m_{iv} = \sum_{\theta_0}^{\theta_0 + 4\pi} \frac{dm_{iv}}{d\theta} \Delta\theta \quad 7.3$$

- o Mass flowing through exhaust valve in kg

This is the total mass of gas that passed into the exhaust through the exhaust valve during a cycle.

$$m_{ev} = \sum_{\theta_0}^{\theta_4\pi} \frac{dm_{ev}}{d\theta} \Delta\theta \quad 7.4$$

- o Enthalpy through inlet valve in joules

This is the total amount of enthalpy associated with the mass that came into the cylinder.

$$H_{iv} = \sum_{\theta_0}^{\theta_4\pi} h_{iv} \frac{dm_{iv}}{d\theta} \Delta\theta \quad 7.5$$

- o Enthalpy through exhaust valve in joules

It is the total amount of enthalpy associated with the mass that went out to the exhaust.

$$H_{ev} = \sum_{\theta_0}^{\theta_4\pi} h_{ev} \frac{dm_{ev}}{d\theta} \Delta\theta \quad 7.6$$

- o Maximum cylinder pressure (Pmax) in bar

The highest pressure reached during the entire cycle.

- o Wall heat transfer in joules

The amount of heat transferred to the wall of the cylinder.

$$Q_{wall} = \sum_{\theta_0}^{\theta_4\pi} \frac{dq_{wall}}{d\theta} \Delta\theta \quad 7.7$$

o Volumetric efficiency

This is defined as the ratio between the mass of air trapped inside the cylinder to the ideal mass trapped inside the cylinder at mean inlet manifold conditions based on the swept volume of the cylinder.

$$\eta_{vol} = \frac{m_{a,trapped}}{\rho_{im} V_{cyl,swept}} \quad 7.8$$

where

$$m_{a,trapped} = \frac{m_{cyl}}{1 + \lambda_{ivc}} \quad 7.9$$

and

$$\rho_{im} = \frac{P_{im,av}}{T_{im,av} R_{im,av}} \quad 7.10$$

o Friction mean effective pressure (FMEP) in bar

The work to overcome friction per cycle divided by the swept volume.

o Average engine speed ( $\omega_e$ ) in revolutions per minute

o Fuel mass injected ( $m_f$ ) in kg

The following cycle variables are collected by each manifold control volume task.

o Average manifold pressure (P) in bar

o Average manifold temperature (T) in Kelvin

o Average turbocharger speed ( $\omega_{tc}$ ) in revolutions per minute

- o Mass flow through the compressor or turbine per cycle in kg

$$m_{tc} = \sum_{t_0}^{t_{4\pi}} \frac{dm_{tc}}{dt} \Delta t \quad 7.11$$

- o Mass flow to or from the cylinders per cycle in kg

$$m_{cyl} = \sum_{t_0}^{t_{4\pi}} \frac{dm_{cyl}}{dt} \Delta t \quad 7.12$$

- o Enthalpy flow out of, or into, the manifold through the turbocharger compressor or turbine in joules.

$$H_{tc} = \sum_{t_0}^{t_{4\pi}} h_{tc} \frac{dm_{tc}}{dt} \Delta t \quad 7.13$$

- o Enthalpy flow to or from the manifold through poppet valves in joules

$$H_{cyl} = \sum_{t_0}^{t_{4\pi}} h_{c,m} \frac{dm_{cyl}}{dt} \Delta t \quad 7.14$$

where  $h_{c,m}$  is the enthalpy associated with the mass flows across the junctions to or from the cylinders.

- o Manifold wall heat transfer in joules

$$Q_{wall} = \sum_{t_0}^{t_{4\pi}} \frac{dq_{wall}}{dt} \Delta t \quad 7.15$$

- o Average power generated by the turbine or required to run the compressor in Watts

$$P_{tc} = \sum_{t_0}^{t_{4\pi}} \tau_{tc} \omega_{tc} \Delta t / (t_{4\pi} - t_0) \quad 7.16$$

- o Average torque required to move the turbocharger shaft ( $\tau_{tc}$ ) in Nm
- o Average turbocharger efficiency ( $\eta_{tc}$ )

## 7.2.2 The Derived Engine Parameters

The engine cycle parameters described above, are used to derive the following engine performance characteristics.

- o Indicated mean effective pressure (IMEP) in bar  
This is the uniform pressure which, acting through the power stroke only, would do the same amount of indicated work during one cycle as is done by the varying pressure inside the cylinder [61].

$$\text{IMEP} = \frac{W_{cc} + W_{oc}}{V_{cyl,swept}} \quad 7.17$$

- o Pumping mean effective pressure (PMEP) in bar

$$\text{PMEP} = \frac{W_{oc}}{V_{cyl,swept}} \quad 7.18$$

- o Brake mean effective pressure (BMEP) in bar

This is the uniform pressure on the cylinder piston which will produce actual output at the crankshaft after overcoming the friction.

$$\text{BMEP} = \text{IMEP} - \text{FMPEP} \quad 7.19$$

- o Mechanical efficiency

This is a measure of total mechanical losses and is calculated as

$$\eta_{\text{mech}} = \text{BMEP} / \text{IMEP} \quad 7.20$$

- o Indicated power (IP) in Watts

This is the power developed inside the engine cylinders.

$$\text{IP} = \frac{(W_{\text{cc}} + W_{\text{oc}}) \omega_e}{4 \pi} \quad 7.21$$

- o Brake power (BP) in Watts

This is the power produces by the engine after overcoming fictional losses.

$$\text{BP} = \text{IP} \eta_{\text{mech}} \quad 7.22$$

- o Indicated specific fuel consumption (ISFC) in kg/kW hr

It is the fuel flow rate necessary to produce unit indicated power from a given engine.

$$\text{ISFC} = \frac{mf}{W_{\text{cc}} + W_{\text{oc}}} * 3.6 \cdot 10^6 \quad 7.23$$

- o Brake specific fuel consumption (BSFC) in kg/kWhr

$$\text{BSFC} = \text{ISFC} / \eta_{\text{mech}} \quad 7.24$$

- o Indicated efficiency

$$\eta_{\text{ind}} = \frac{(W_{\text{cc}} + W_{\text{oc}})}{mf \text{ calval}} \quad 7.25$$

where the calorific value 'calval' is in J/kg.

- o Brake thermal efficiency

$$\eta_{\text{bk}} = \eta_{\text{ind}} \eta_{\text{mech}} \quad 7.26$$

- o Total work output by engine in joules

$$W_{\text{out}} = (W_{\text{cc}} + W_{\text{oc}}) \eta_{\text{mech}} \quad 7.27$$

- o Work done against friction in joules

$$W_{\text{fric}} = (W_{\text{cc}} + W_{\text{oc}}) - W_{\text{out}} \quad 7.28$$

Under steady state operation, that is constant speed and load, the mass and the energy balance for each control volume can be used to indicate the correctness of the sub-models and their implementation and to the numerical integrator. For any control volume, the mass balance MB is given by

$$MB = \frac{\text{total mass entering per cycle}}{\text{total mass leaving per cycle}} \quad 7.29$$

and the energy balance EB is given by

$$EB = \frac{\text{total energy entering per cycle}}{\text{total energy leaving per cycle}} \quad 7.30$$

For a cylinder

$$\begin{aligned} \text{total mass entering} &= m_{iV} + m_f \\ \text{total mass leaving} &= m_{eV} \\ \text{total energy entering} &= m_f \text{ calval} \\ \text{total energy leaving} &= H_{ex} - H_{in} + Q_{wall} + W_{out} + W_{fric} \end{aligned} \quad 7.31$$

For a manifold

$$\begin{aligned} \text{total mass entering} &= \dot{m}_{tc} \\ \text{total mass leaving} &= m_{cyl} \\ \text{total energy entering} &= H_{tc} \text{ (inlet), } H_{cyl} \text{ (exhaust)} \\ \text{total energy leaving} &= H_{tc} \text{ (exhaust), } H_{cyl} \text{ (inlet)} \end{aligned} \quad 7.32$$

## 7.3 Collection and Presentation of Engine Cycle Parameters

Data from the Diesel engine simulation is collected at two different stages. Firstly, results that are recorded at each integration step, and secondly, results that are calculated and recorded on completion of a cycle. Table 7.1 gives those parameters which are recorded at each integration step and table 7.2 gives those parameters which are recorded for every cycle.



To collect and manipulate engine variables, an interactive task is setup which acts as a link between the user and the PDESIM. When information at each integration step is required, the interactive task sends a packet to the corresponding task. Each simulation task has a logging area for recording step information. On receiving a log packet, the simulation task starts logging information in that area. It returns the log packet to the interactive task when the logging is finished. The user interface task can either display these logged variables immediately or can store the results.

Table 7.1 Variables recorded at each step

Cylinder	Manifold
angle	relative angle
pressure	pressure
temperature	temperature
mass	fuel to air ratio
volume	mass
flow through inlet valve	flow through turbocharger
flow through exhaust valve	flow through poppet valves
heat transferred to wall	turbocharger torque
heat released into cylinder	turbine speed
engine torque	heat transferred to wall

The cycle results are collected in a different way. Each cylinder and manifold task keeps a circular buffer for recording cycle data. It updates this buffer every cycle, and advances a pointer to the currently valid cycle data area. When a request for cycle data logging is received, the pointer to the latest valid cycle data area is returned. The user interface task then reads in data directly from this data location and does all the

required post-processing to calculate engine performance parameters. The results may then be displayed on the monitor or they may be stored so that a hard copy may be obtained.

Figure 7.1 gives the structure of the Diesel engine simulation user interface menus. The 'record time history' command records data collected every cycle and the 'log cycle data' command records data collected at each step. The 'modify parameters' command allows the user to change some of the engine parameters. If a value of -1 is sent for the rack position, the injection angle, or the amount of fuel injected, then dynamically calculated values of these parameters are used. Otherwise the given value is used in the simulation.

Table 7.2 Variables recorded over a cycle

Cylinder	Manifold
open cycle work	mass flow through valves
closed cycle work	mass flow through turbocharger
mass flow through inlet valve	energy entering manifold
mass flow through exhaust valve	energy leaving manifold
energy flow through inlet valve	mean pressure
energy flow through exhaust valve	mean temperature
maximum pressure	mean turbocharger speed
heat transferred to wall	heat transferred to wall
volumetric efficiency	mean turbocharger power
FMEP	mean turbocharger torque
engine speed	mean turbocharger efficiency
mass of fuel injected	
swept volume	

The 'display CV data' command displays either the cylinder or the manifold data on the screen. The 'get map data' automatically records engine performance data over a range of engine speed and fuelling.

## **7.4 Validation of Results**

In order to validate the simulation results, a Leyland TL11 Diesel engine is simulated and the performance characteristics are compared with those obtained from a similar engine using FJH Model [21].

### **7.4.1 The Leyland Tl11 Diesel Engine**

The Leyland TL11 Diesel engine is a high speed truck engine. It is being used as an experimental engine in the School of Mechanical Engineering, University of Bath. It has six cylinders 'in-line'. It is a four stroke, 11.1 litre turbocharged Diesel engine capable of producing 190 KW of power at a maximum speed of 2100 rpm.

The TL11 has a direct fuel injection system. The fuel pump is driven at half engine speed from the engine crankshaft and is controlled in two ways; by the rack control and by the timing control. The rack control defines the amount of fuel injected into the engine and the timing control defines the angle at which the fuel enters the cylinders.

Earlier, the TL11 was fitted with a variable geometry Holset turbocharger, which has been replaced with a fixed geometry Garrett turbocharger. The engine has an air-to-water aftercooler which cools the charge air before it enters the inlet manifold.

The power from the TL11 engine is absorbed by a hydrostatic dynamometer in three different modes; constant speed, constant torque, and with torque proportional to the square of the engine speed. A complete description of the Leyland TL11 engine and its behaviour is given by Roberts [62].

The TL11 has been extensively instrumented to give as full a picture of the operation of the engine as possible. A description of this may be found in the Leyland TL11 instrumentation manuals [63]. To measure the cylinder pressure, two Kistler 6121 piezo-electric pressure transducers and their associated amplifiers are fitted at the top of cylinders three and six. These pressure transducers are removable to allow the deposits from fuel combustion to be removed.

Temperatures in the engine system are measured using Chromel-Alumel (K type) thermocouples. These thermocouples are buffered using conditioning cards which allow the user to present temperature readings to a display on the engine instrumentation panel or to read using a computer.

The dynamic timing of the injection of fuel into cylinders is indicated by needle lift transducers which are fitted to cylinders three and six.

To obtain the speed of the engine, the crankshaft position is monitored using an AVL crankshaft optical encoder type 360C/600. It has two rings of dark and bright bands, one provides a crankshaft resolution of 0.6 degrees and the other outputs a pulse when cylinder one is at TDC.

A number of other transducers are also fitted to the engine to measure turbocharger speed, manifold pressures, fuel flow rates, engine torque etc. A description of these is given by Haysom [21] and Scaife [64].

To monitor, record and present results from the TL11 engine, a computer data acquisition system has been designed and built by Haysom [21]. This system is based on a Motorola MC68000 based single board computer system. It logs analogue and digital data at the rate of 25 kHz. The system consists of a master board, slave boards, a two megabyte backplane memory board, a multilink local area network board and an EFCIS colour graphics board. A complete description of the data acquisition system may be found in the theses of Haysom and Scaife.

## **7.4.2 The FJH Model**

The simulation model due to Haysom has data built into the program. It uses a Holset turbocharger in the six cylinder TL11 Diesel engine simulation and has been verified against the real engine. To allow a comparison between the FJH Model and the PDESIM, the Holset turbocharger was modelled. The Watson heat release model [25] was used for heat release calculations and the Hohenberg model [28] was used for heat transfer calculations. These models were selected because the FJH Model uses only these two functions for simulation. The same poppet valve effective area tables and initial conditions were fed to the PDESIM as those used in the FJH model.

To compare the steady state performance of the two models, the simulations were setup to achieve a constant engine speed by selecting a high load inertia. The amount of fuel injected into the system was adjusted so that the same amount of fuel could be used in both cases. For each point, the simulations were allowed to reach steady state before any results were taken. The simulations were run with an initial integration step of 2 degrees and a stability criterion of 0.5 percent.

A comparison of the results from the PDESIM and the FJH Model is shown in figures 7.2 and 7.3. In figure 7.2, the BMEP and the brake power results from the FJH Model

and the PDESIM are plotted against fuel flow rate. A Willan's line [61] is superimposed on these graphs. The Willan's line gives an approximation to the FMEP and the friction power of compression ignition engine. In these graphs, the point at which the line crosses the x-axis provides the rate of fuel flow needed to overcome friction. It gives the value of the FMEP and the friction power required to overcome it when it touches the y-axis, where zero fuelling occurs at a particular speed.

In figure 7.3 engine maps showing brake power, maximum cylinder pressure, the BSFC, the brake efficiency, the boost pressure, the turbocharger speed, the compressor power, and the fuel flow are given. These maps are drawn against the BMEP as function of the engine speed and show similar trends for both the models.

### **7.4.3 The TL11 engine**

A new turbocharger has been fitted onto the TL11, and is being tested in the School of Mechanical Engineering [65]. In order to have a test comparison, the PDESIM was run using data for the new Garrett turbocharger and two of the results compared with those available from the engine. These results are shown in figure 7.4, and the agreement between theoretical and experimental results is extremely promising.

In figure 7.4a and 7.4b, a Willan's' line is superimposed on the BMEP plots for engine speeds of 1200 rpm and 1600 rpm respectively. It gives a FMEP value of 1.57 bar and 1.71 bar at 1200 rpm and 1600 rpm respectively. Similarly, the friction power required to overcome this friction is 17.4 and 25 KW for the respective speeds.

## **7.5 Summary:**

The results of the parallel engine simulated on the BUTPC are compared with an equivalent simulation due to Haysom which has already been verified against the data of a real engine. The results from the present PDESIM show good correspondence with results from the FJH Model. BMEP and brake power curves for two different engine speeds are also compared with the real experimental results of the TL11 Diesel engine.

Figure 7.1 The user interface menu

c	display CV data	s	store cycle & history
t	record time history	m	modify parameters
l	log cycle history	r	rpm over one minute
g	get map data	q	quit to Helios

Display CV data
-----------------

c	cylinder
m	manifold
r	raw data
q menu	quit to previous

Modify parameters
-------------------

r	rack position	s	engine speed
i	injection angle	f	fuel injected
		q	quit to previous menu



Parallel Diesel Engine Simulation  
1200 rpm , 338 degree injection

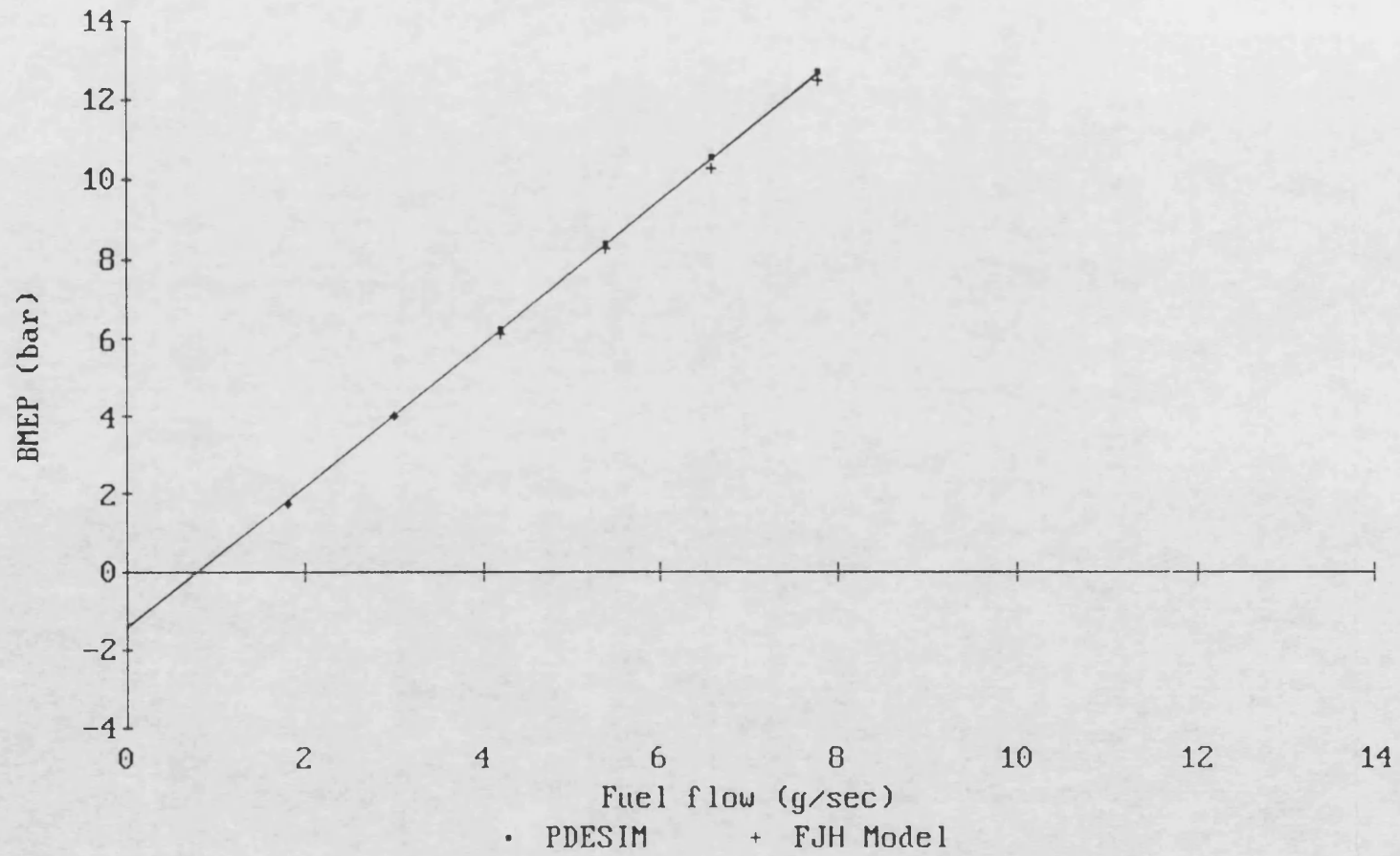


Figure 7.2a A Willan's line superimposed on the BMEP results from the PDESIM and the FJH Model at 1200 rpm

Parallel Diesel Engine Simulation  
1400 rpm, 338 degree injection

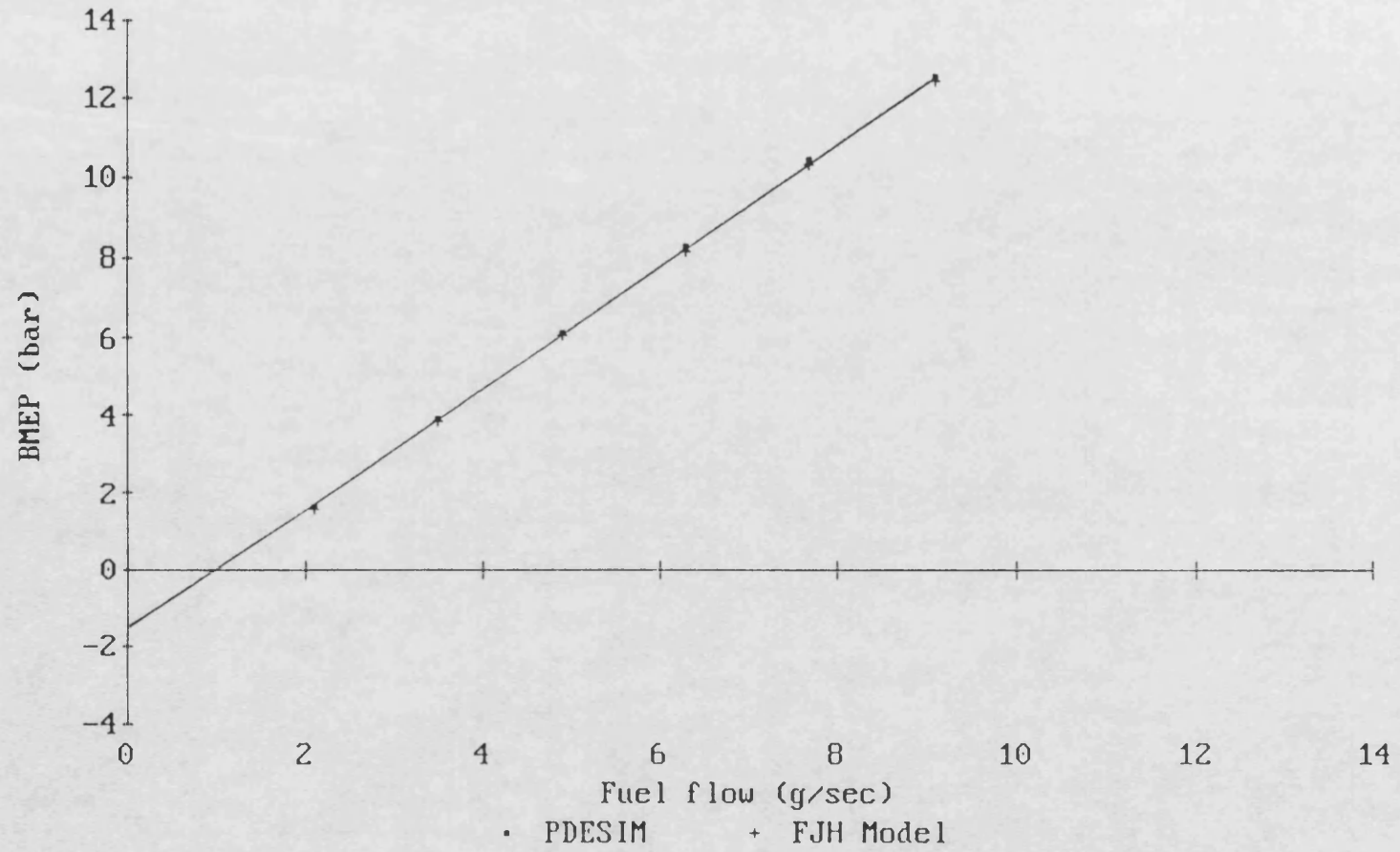


Figure 7.2b A Willan's line superimposed on the BMEP results from the PDESIM and the FJH Model at 1400 rpm

Parallel Diesel Engine Simulation  
1600 rpm, 338 degree injection

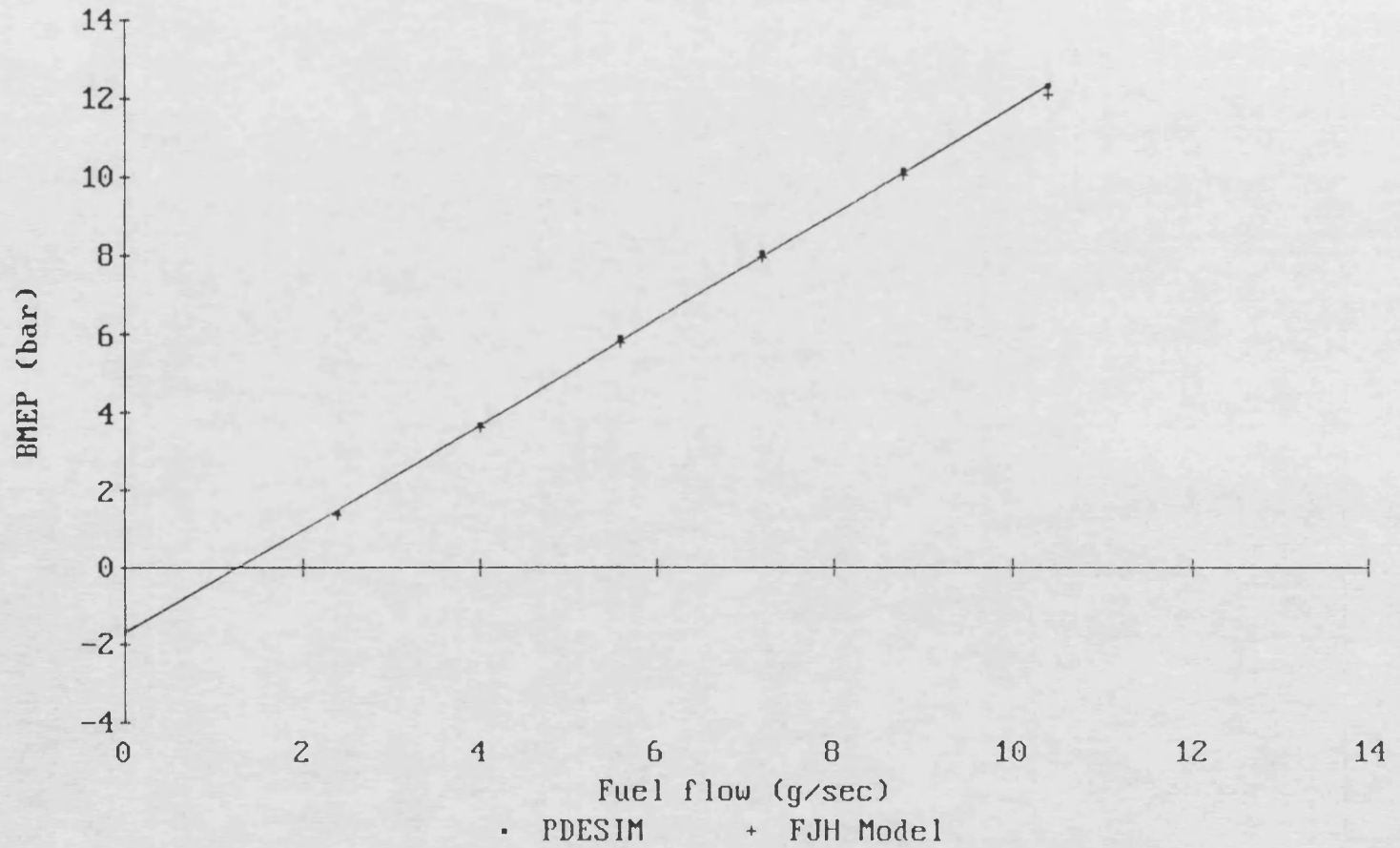


Figure 7.2c A Willan's line superimposed on the BMEP results from the PDESIM and the FJH Model at 1600 rpm

Parallel Diesel Engine Simulation  
1800 rpm, 338 degree injection

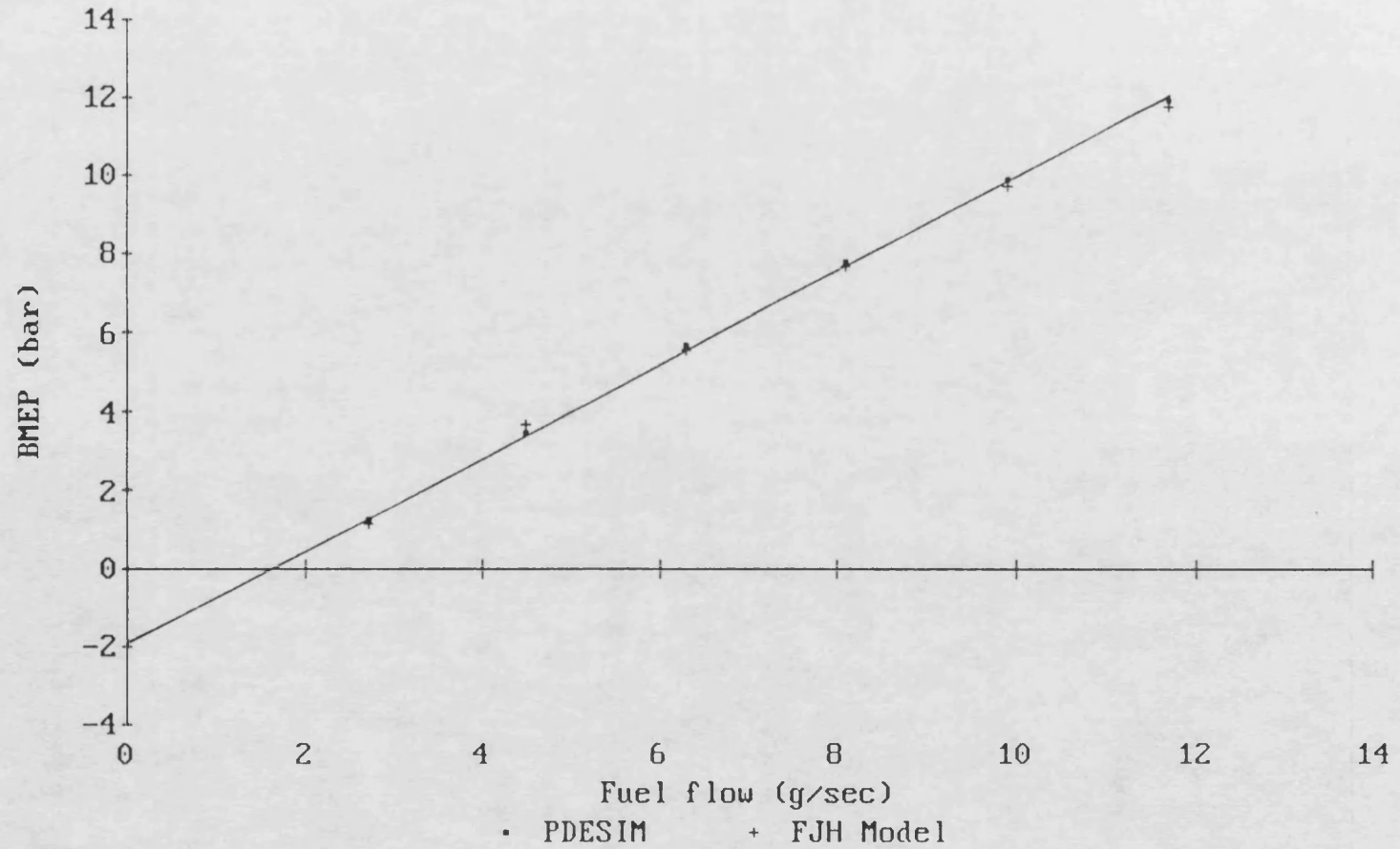


Figure 7.2d A Willan's line superimposed on the BMEP results from the PDESIM and the FJH Model at 1800 rpm

Parallel Diesel Engine Simulation  
2000 rpm, 338 degree injection

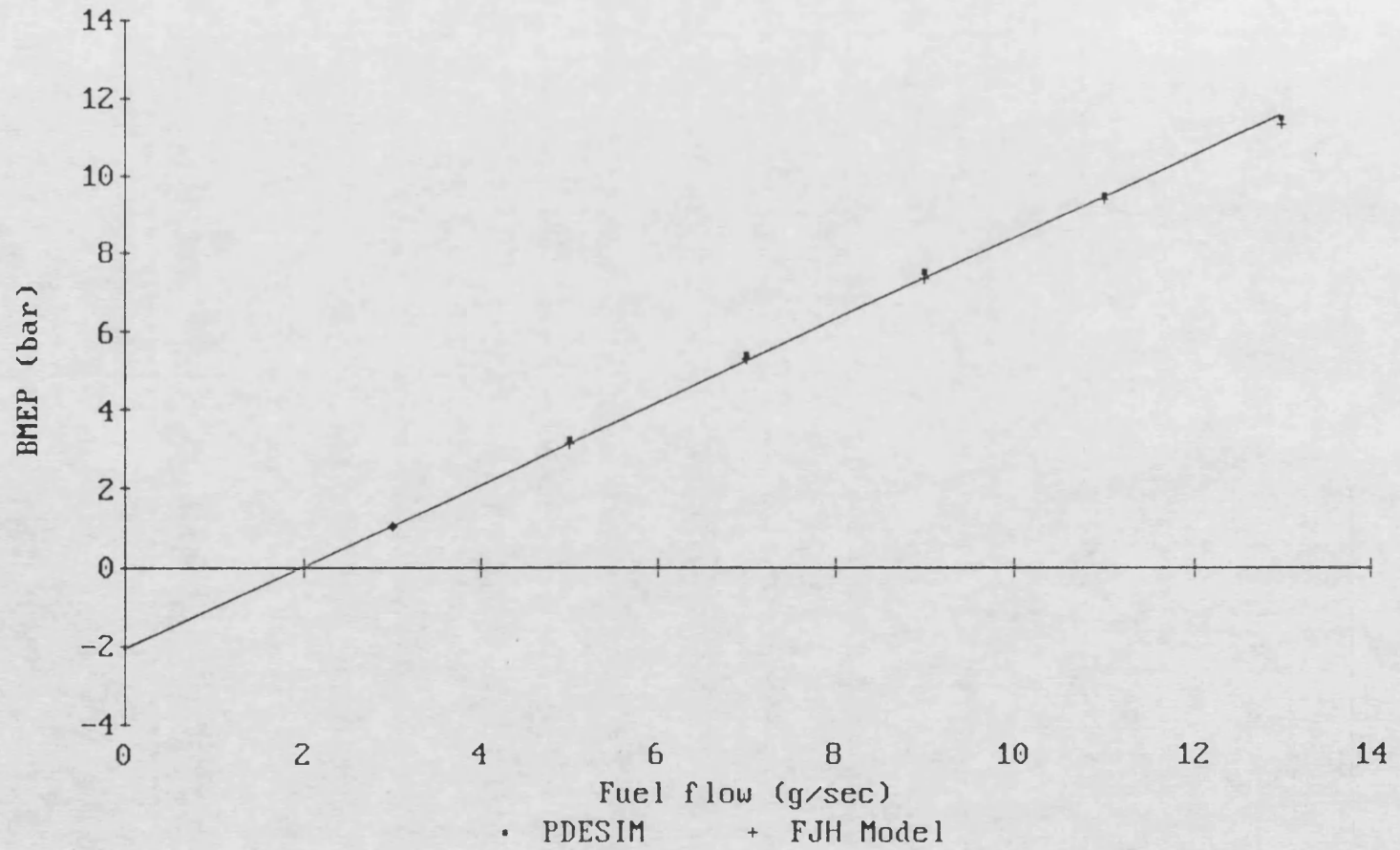


Figure 7.2e A Willan's line superimposed on the BMEP results from the PDESIM and the FJH Model at 2000 rpm

Parallel Diesel Engine Simulation  
1200 rpm , 338 degree injection

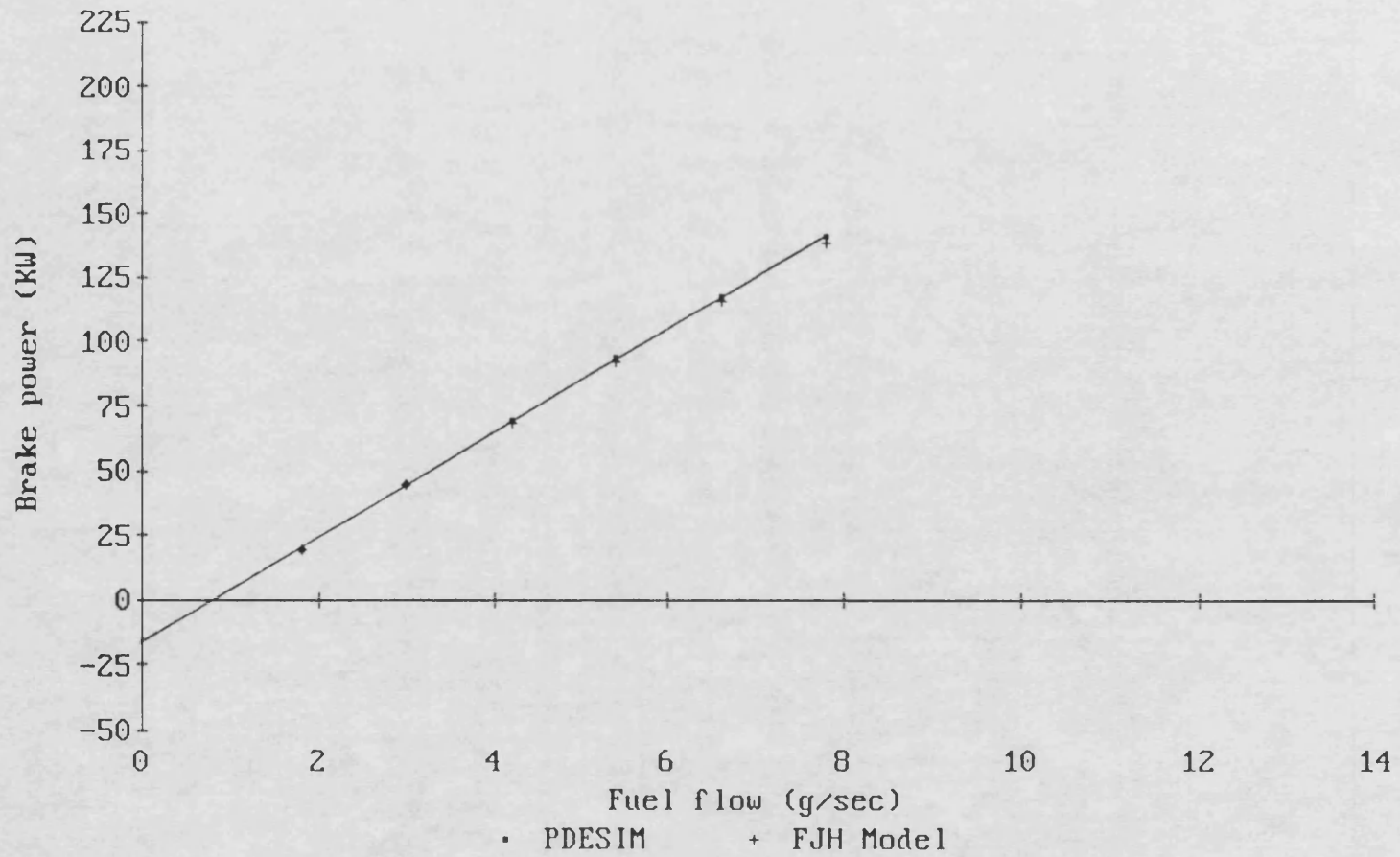


Figure 7.2f A Willan's line superimposed on the brake power results from the PDESIM and the FJH Model at 1200 rpm

Parallel Diesel Engine Simulation  
1400 rpm, 338 degree injection

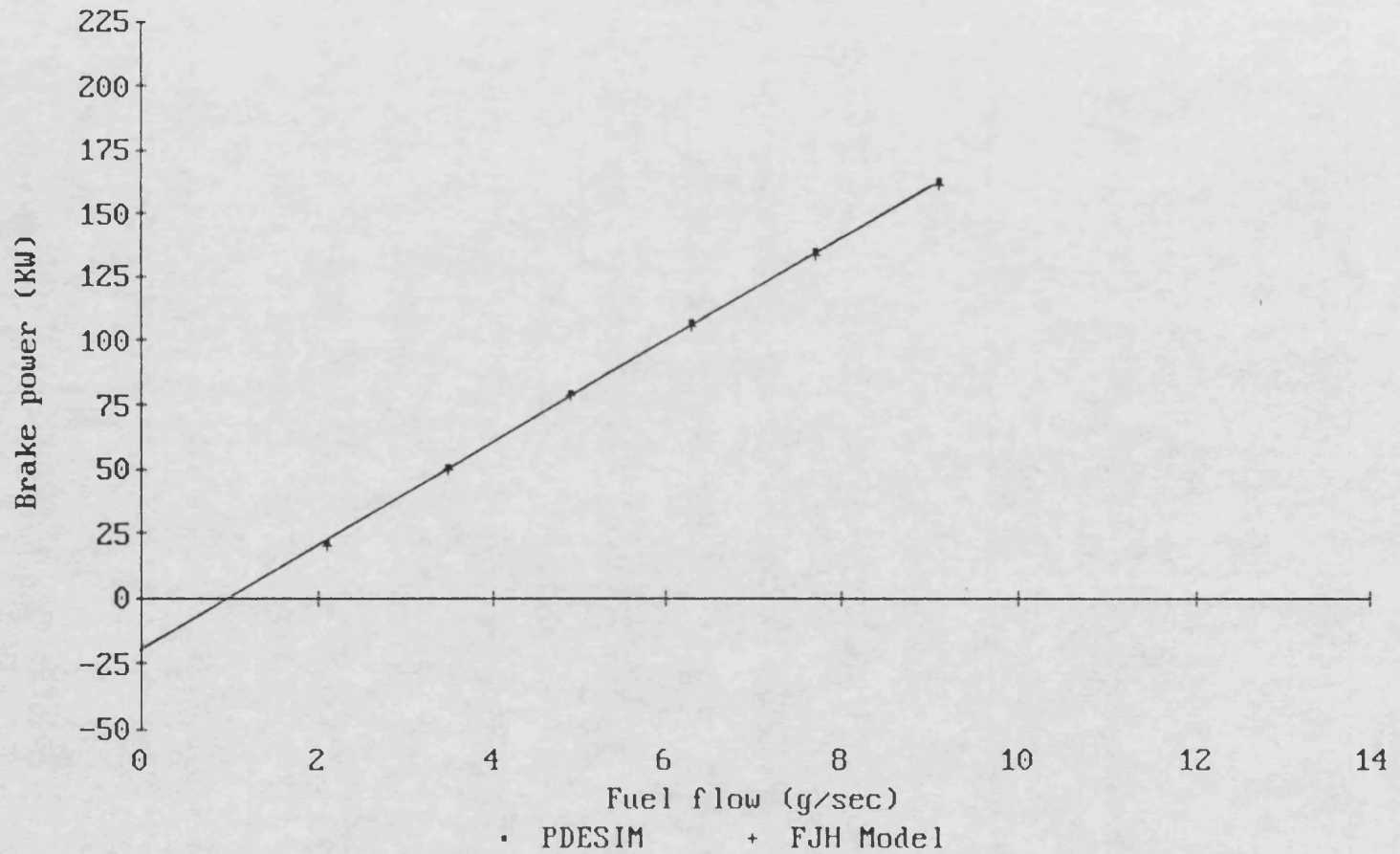


Figure 7.2g A Willan's line superimposed on the brake power results from the PDESIM and the FJH Model at 1400 rpm

Parallel Diesel Engine Simulation  
1600 rpm, 338 degree injection

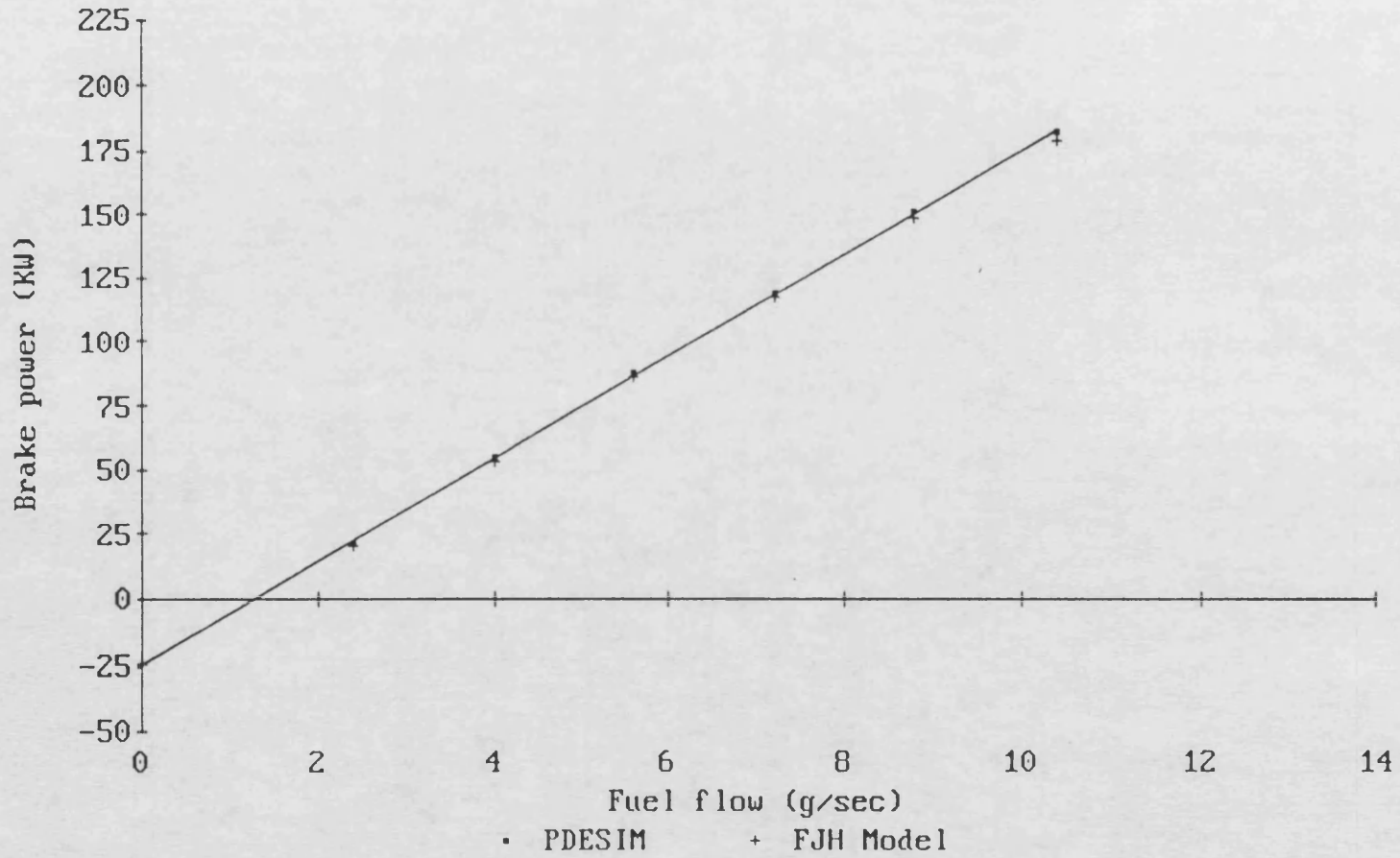


Figure 7.2h A Willan's line superimposed on the brake power results from the PDESIM and the FJH Model at 1600 rpm



Parallel Diesel Engine Simulation  
1800 rpm, 338 degree injection

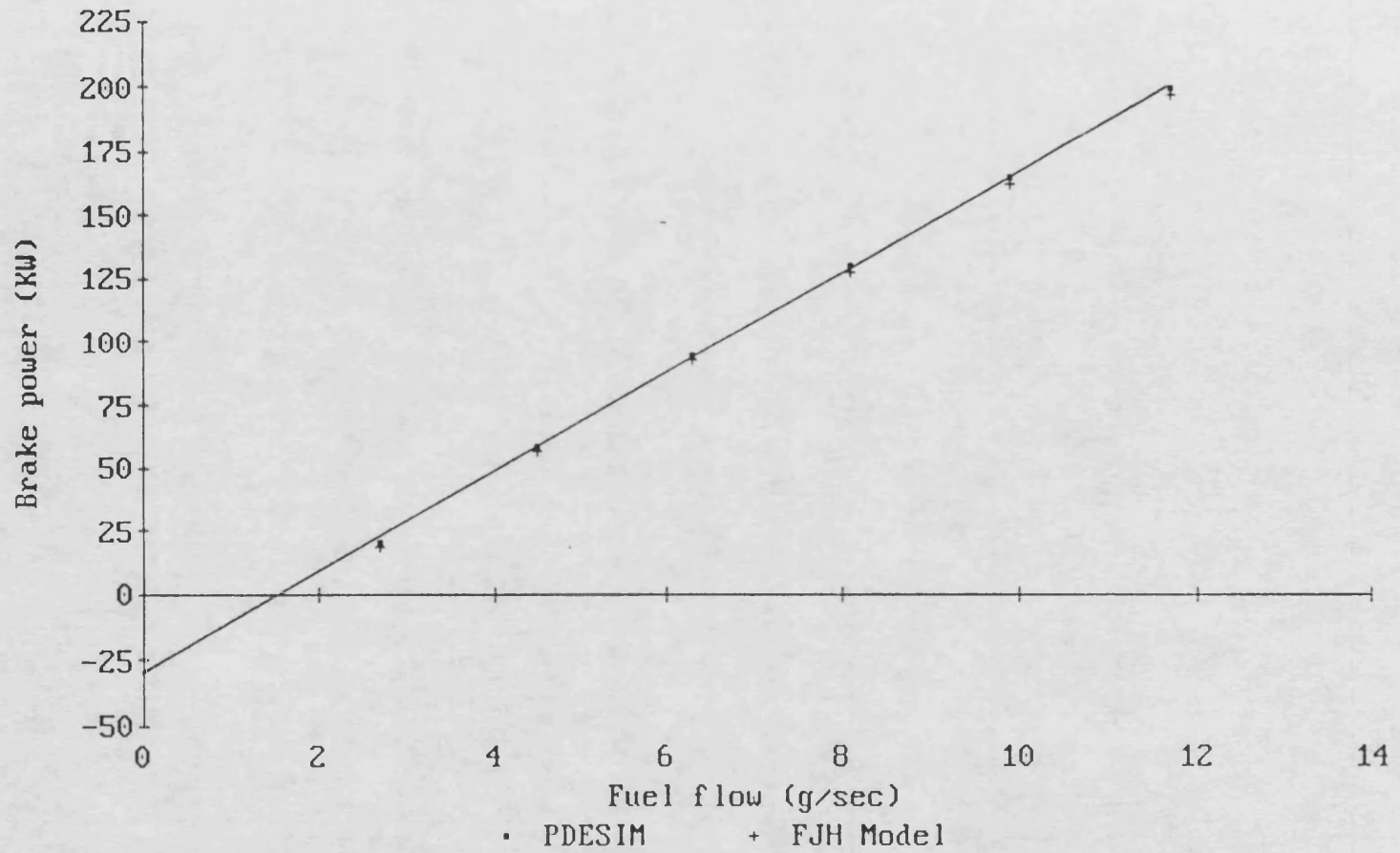


Figure 7.2i A Willan's line superimposed on the brake power results from the PDESIM and the FJH Model at 1800 rpm

Parallel Diesel Engine Simulation  
2000 rpm, 338 degree injection

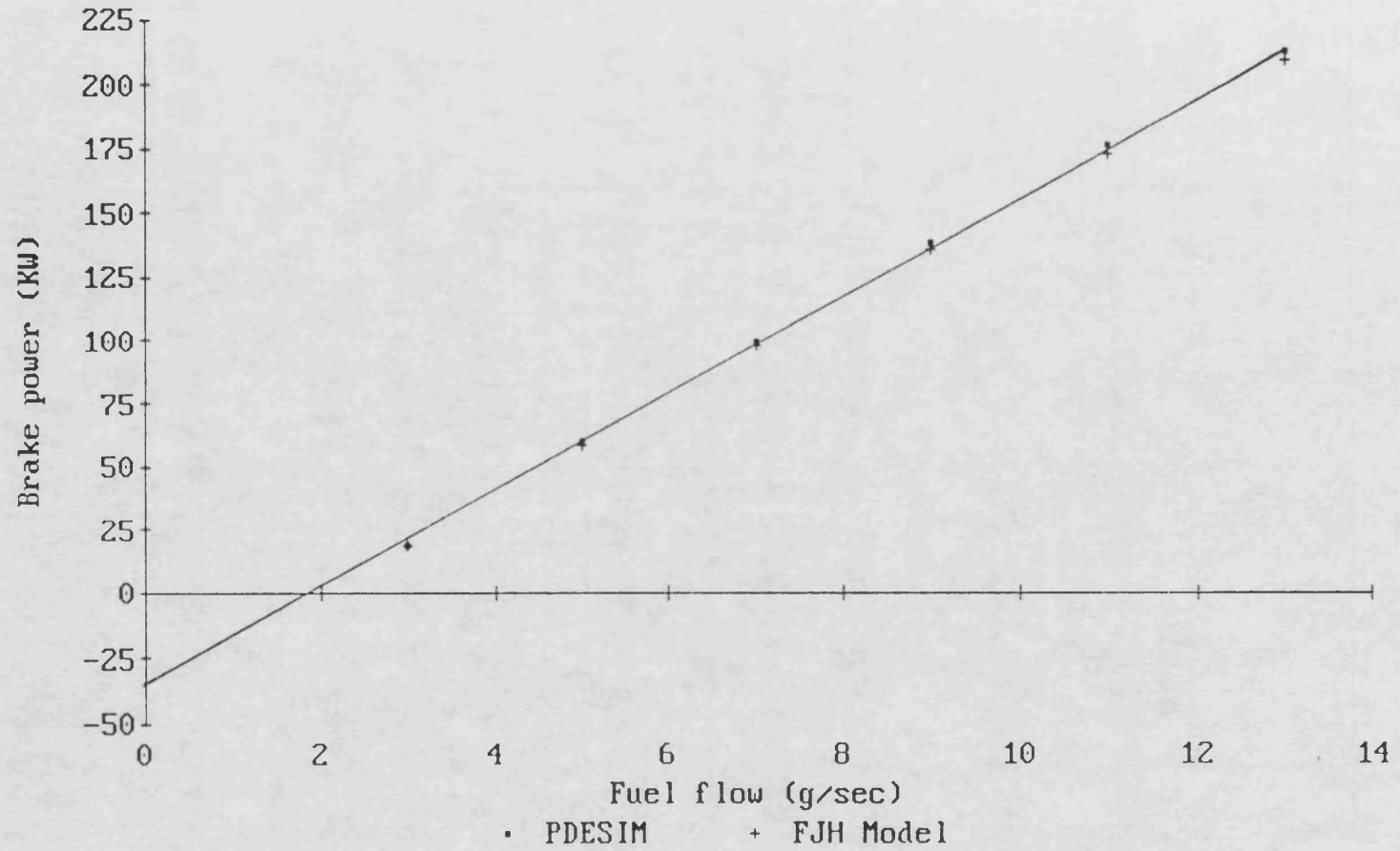
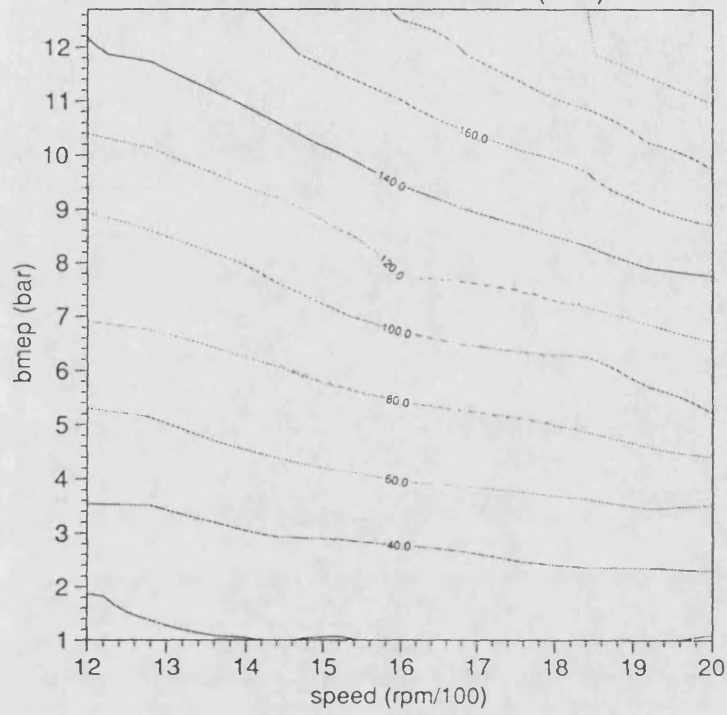


Figure 7.2j A Willan's line superimposed on the brake power results from the PDESIM and the FJH Model at 2000 rpm

Figure 7.3a Engine contour maps showing brake power

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Brake Power (KW)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Brake Power (KW)

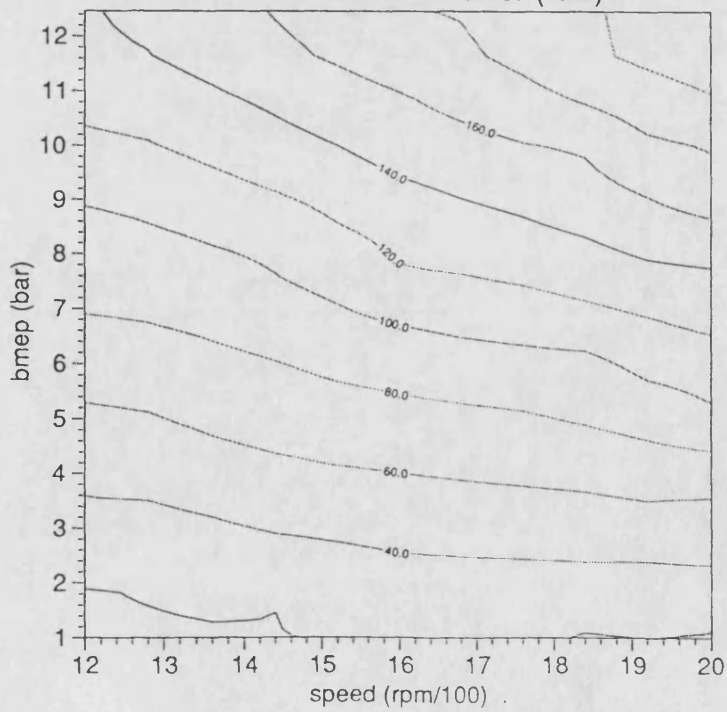
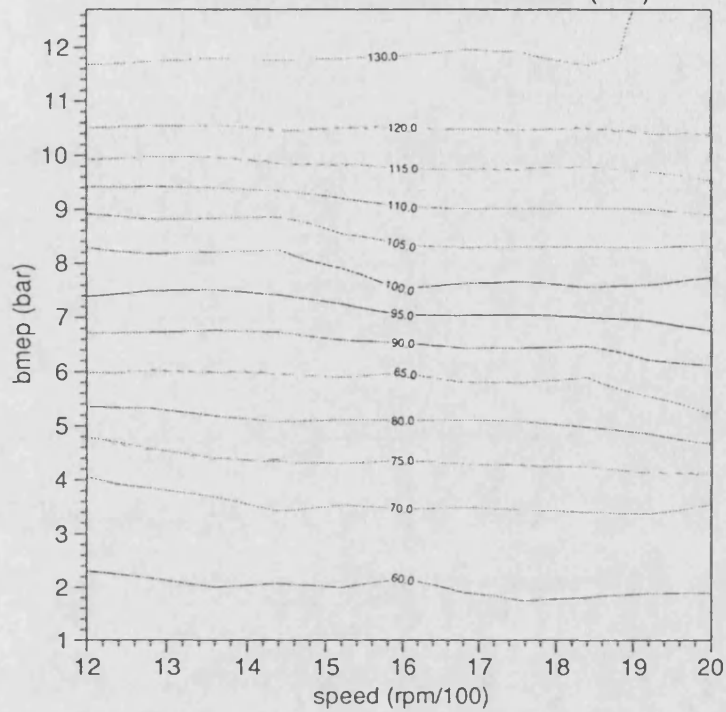


Figure 7.3b Engine contour maps showing maximum cylinder pressure

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Maximum Pressure (bar)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Maximum Pressure (bar)

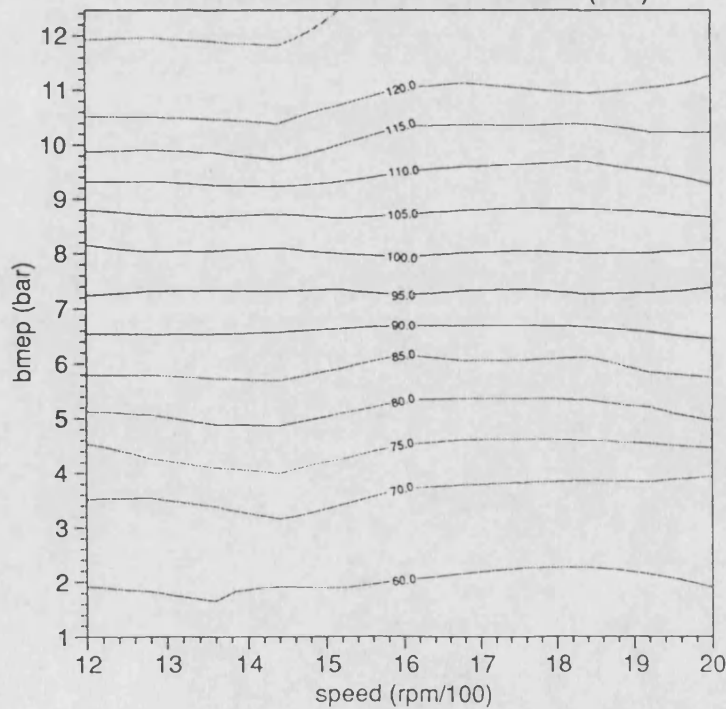
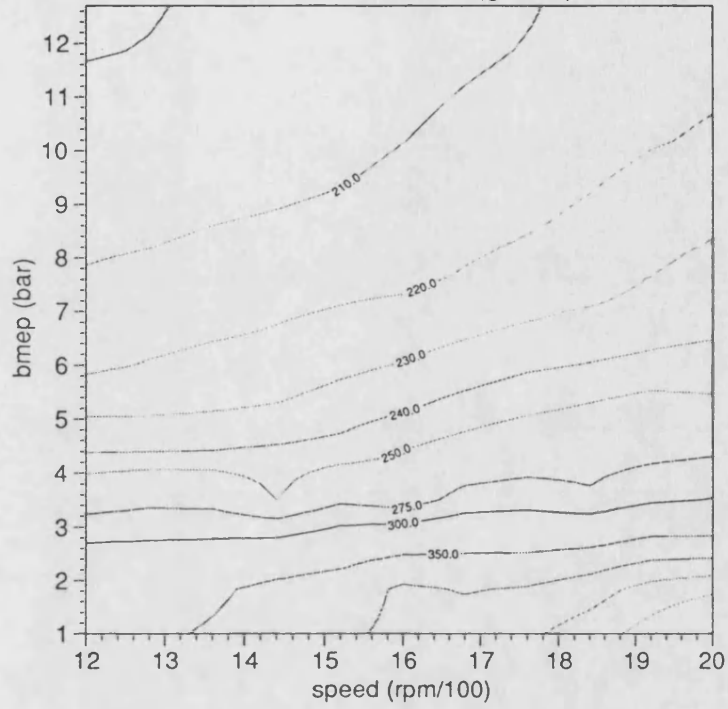


Figure 7.3c Engine contour maps showing the BSFC

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of BSFC (g/kwh)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of BSFC (g/kwh)

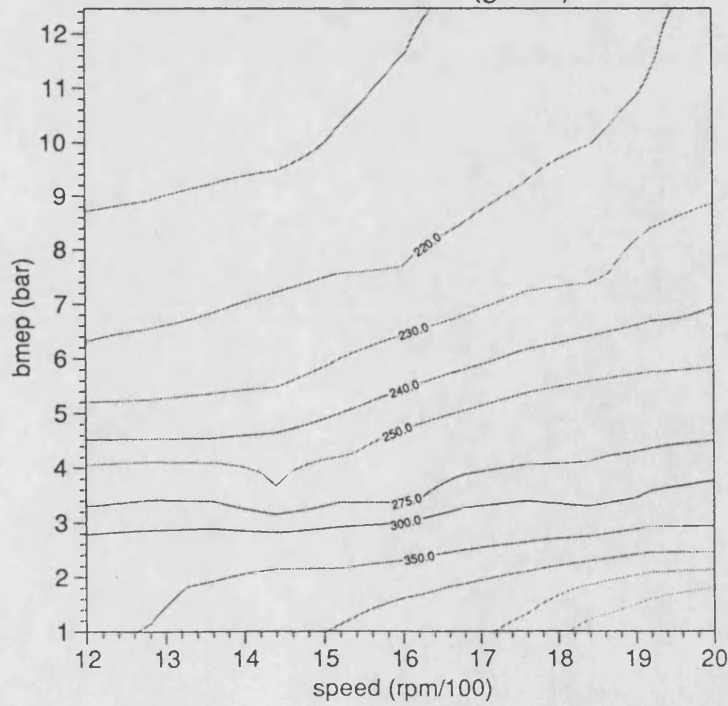
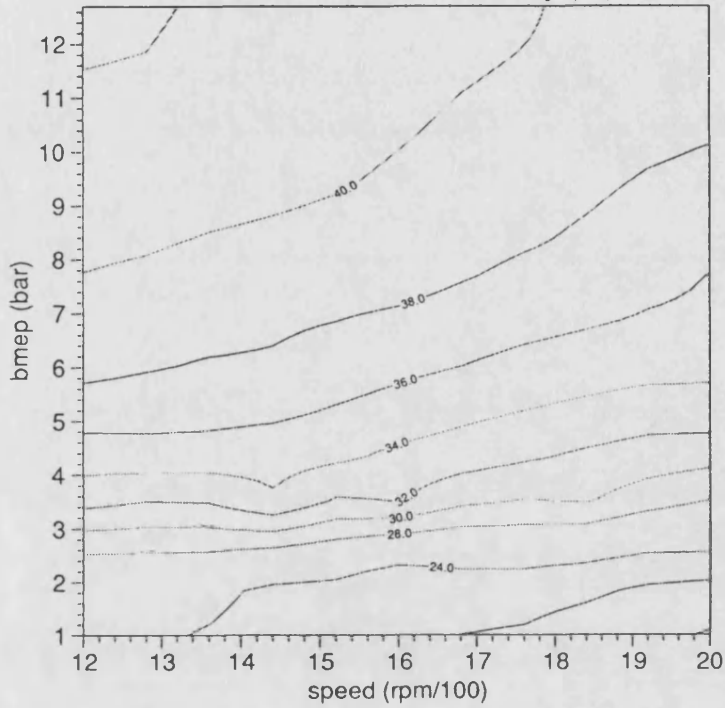


Figure 7.3d Engine contour maps showing brake efficiency

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Brake Efficiency (%)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Brake Efficiency (%)

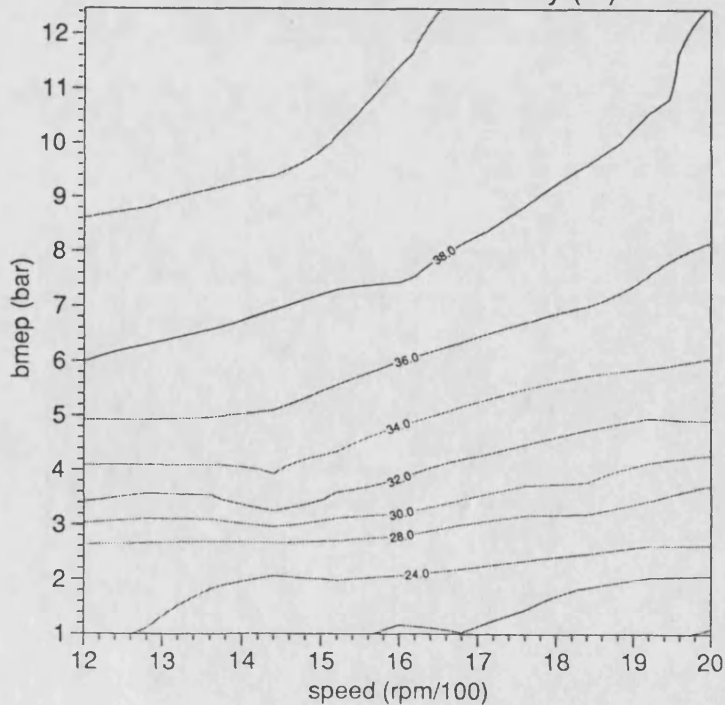
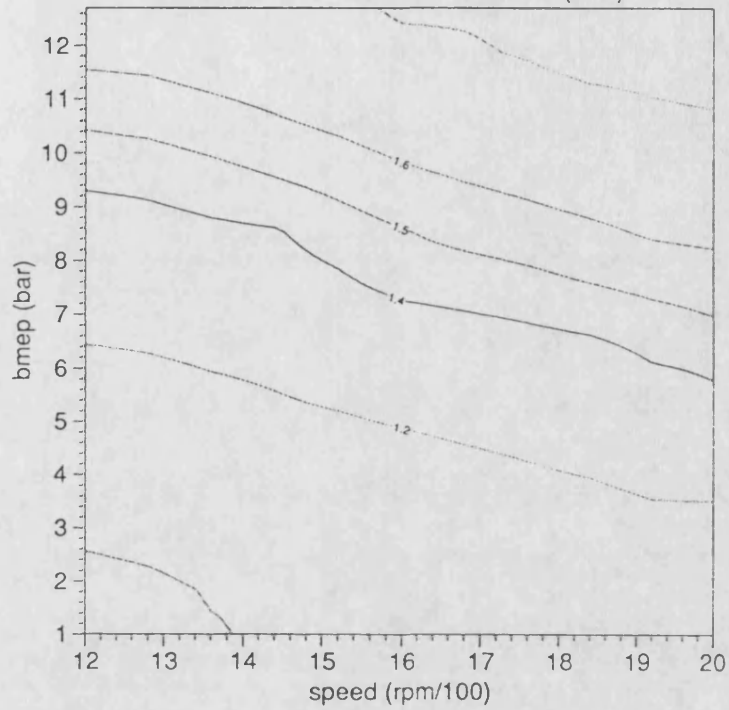


Figure 7.3e Engine contour maps showing boost pressure

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Boost Pressure (bar)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Boost Pressure (bar)

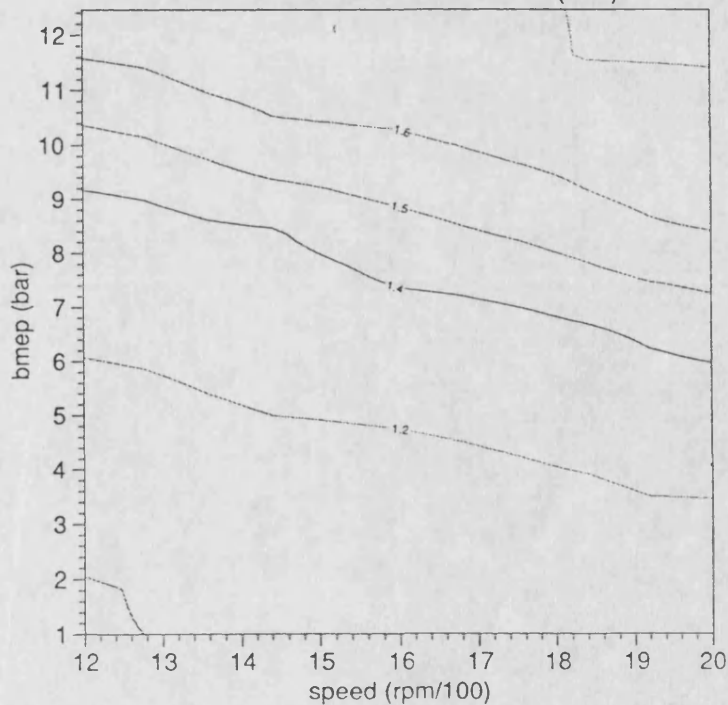
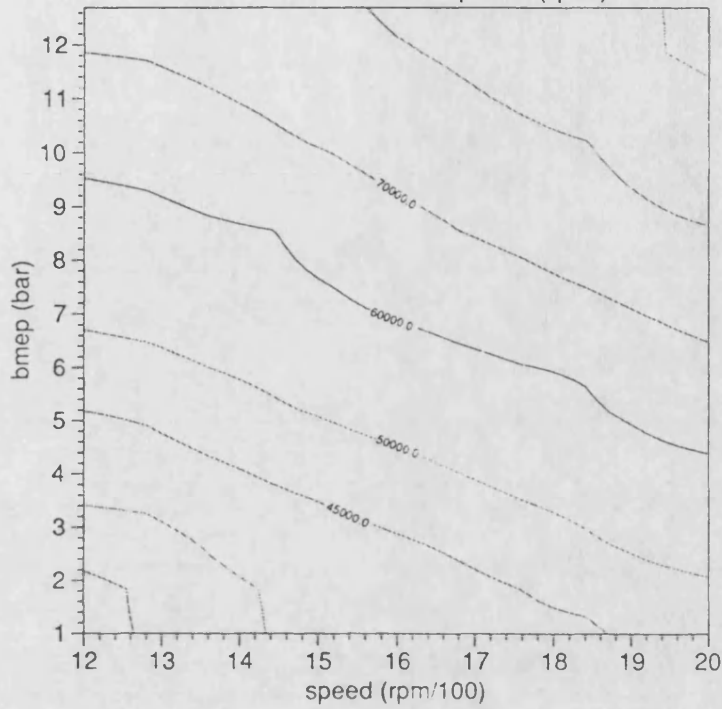


Figure 7.3f Engine contour maps showing turbocharger speed

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Turbo Speed (rpm)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Turbo Speed (rpm)

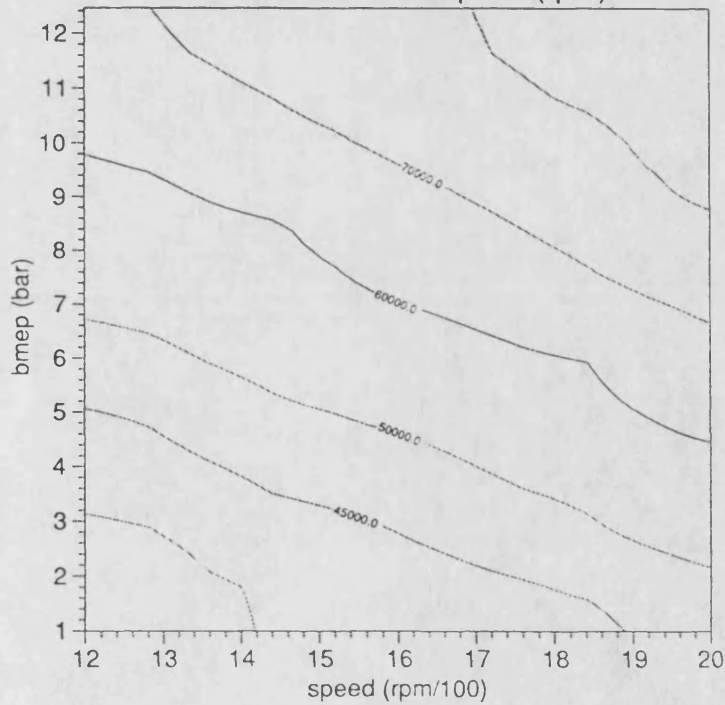
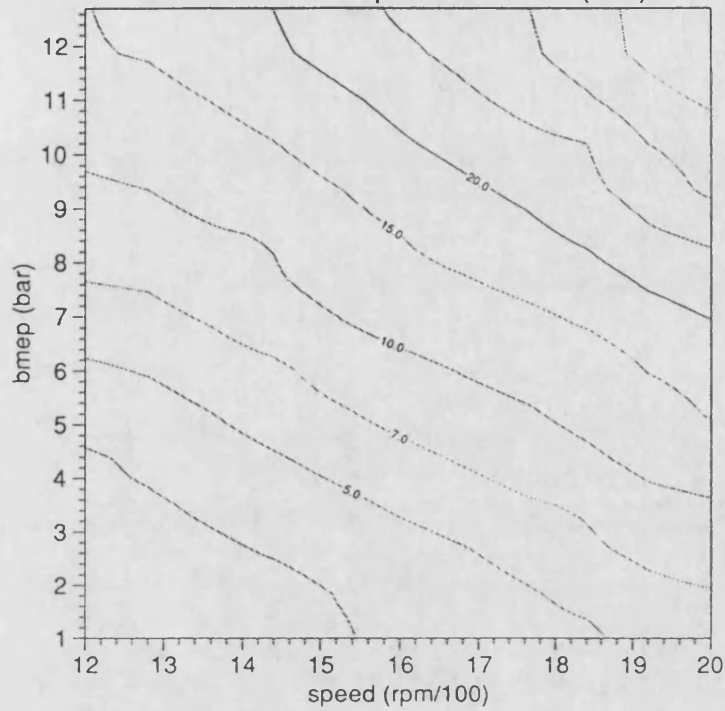




Figure 7.3g Engine contour maps showing compressor power

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Compressor Power (KW)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Compressor Power (KW)

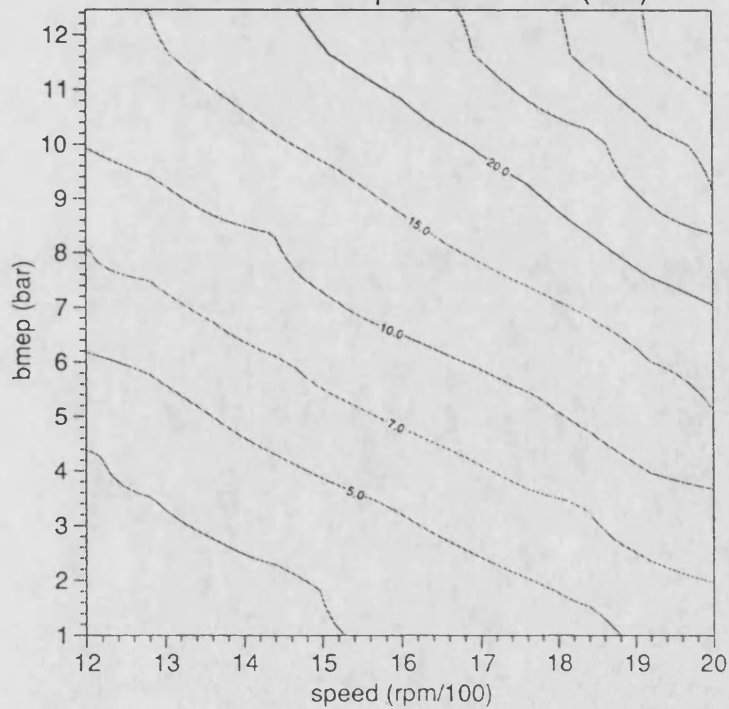
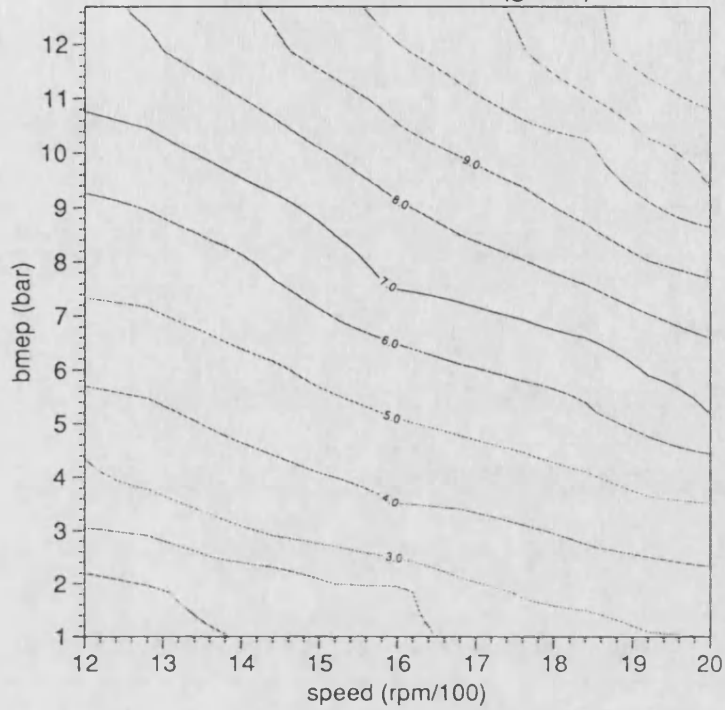
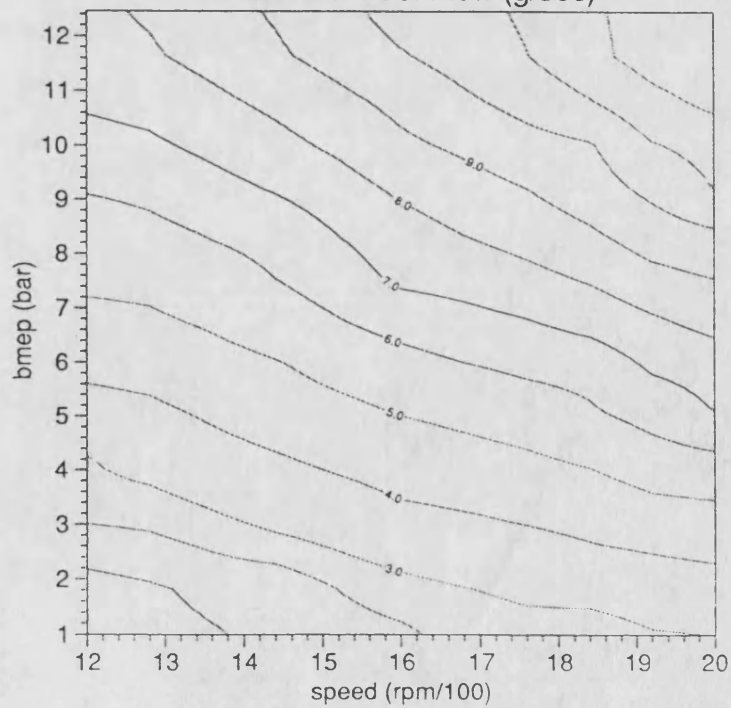


Figure 7.3h Engine contour maps showing fuel flow

The BUTPC Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Fuel Flow (g/sec)



The FJH Parallel Diesel Engine Model With Holset Turbocharger  
Contours of Fuel Flow (g/sec)



Parallel Diesel Engine Simulation  
1200 rpm (Garrett Turbocharger)

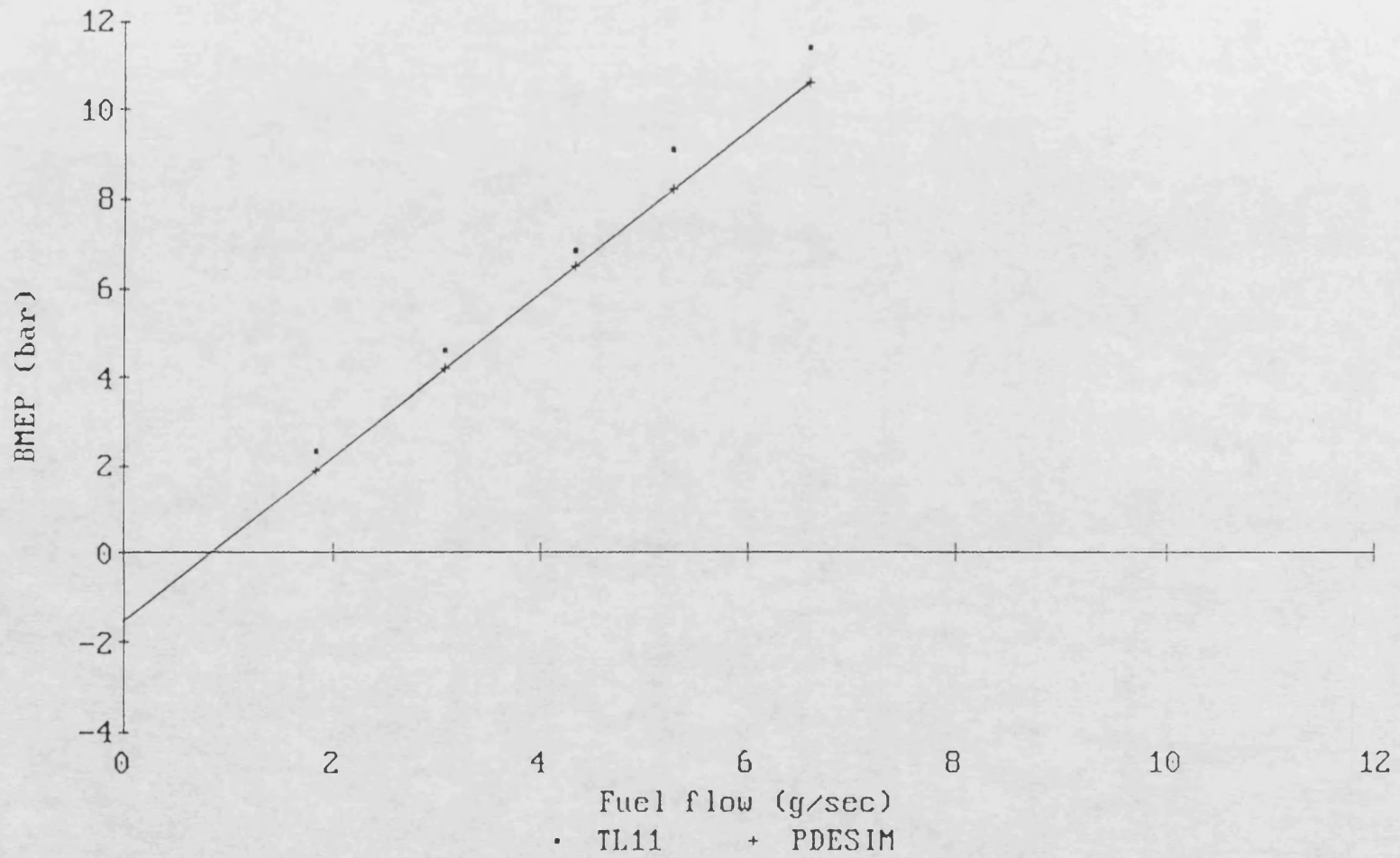


Figure 7.4a A Willan's line superimposed on the BMEP results from the PDESIM and the TL11 Diesel engine at 1200 rpm

Parallel Diesel Engine Simulation  
1600 rpm (Garrett Turbocharger)

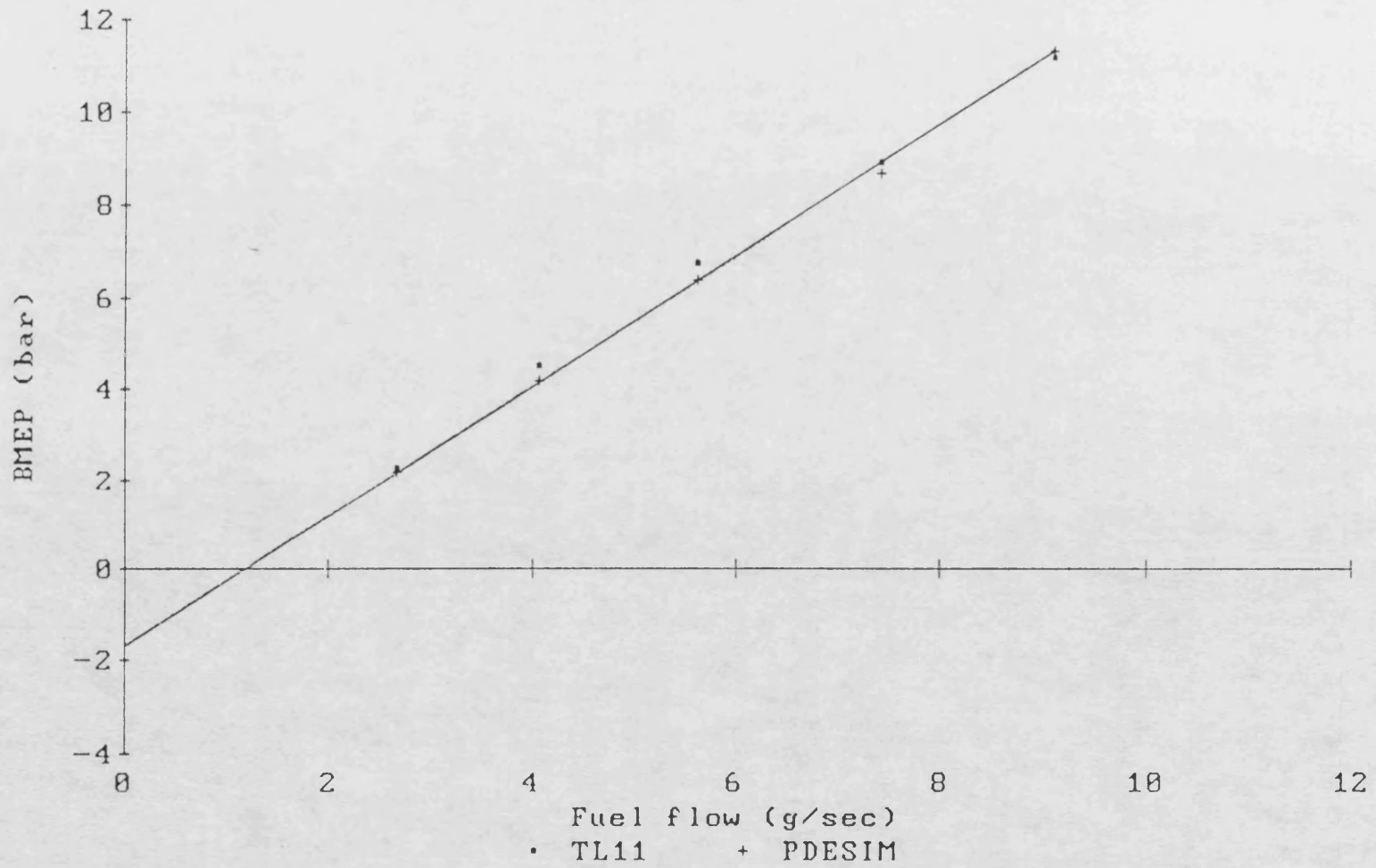


Figure 7.4b A Willan's line superimposed on the BMEP results from the PDESIM and the TL11 Diesel engine at 1600 rpm

Parallel Diesel Engine Simulation  
1200 rpm (Garrett Turbocharger)

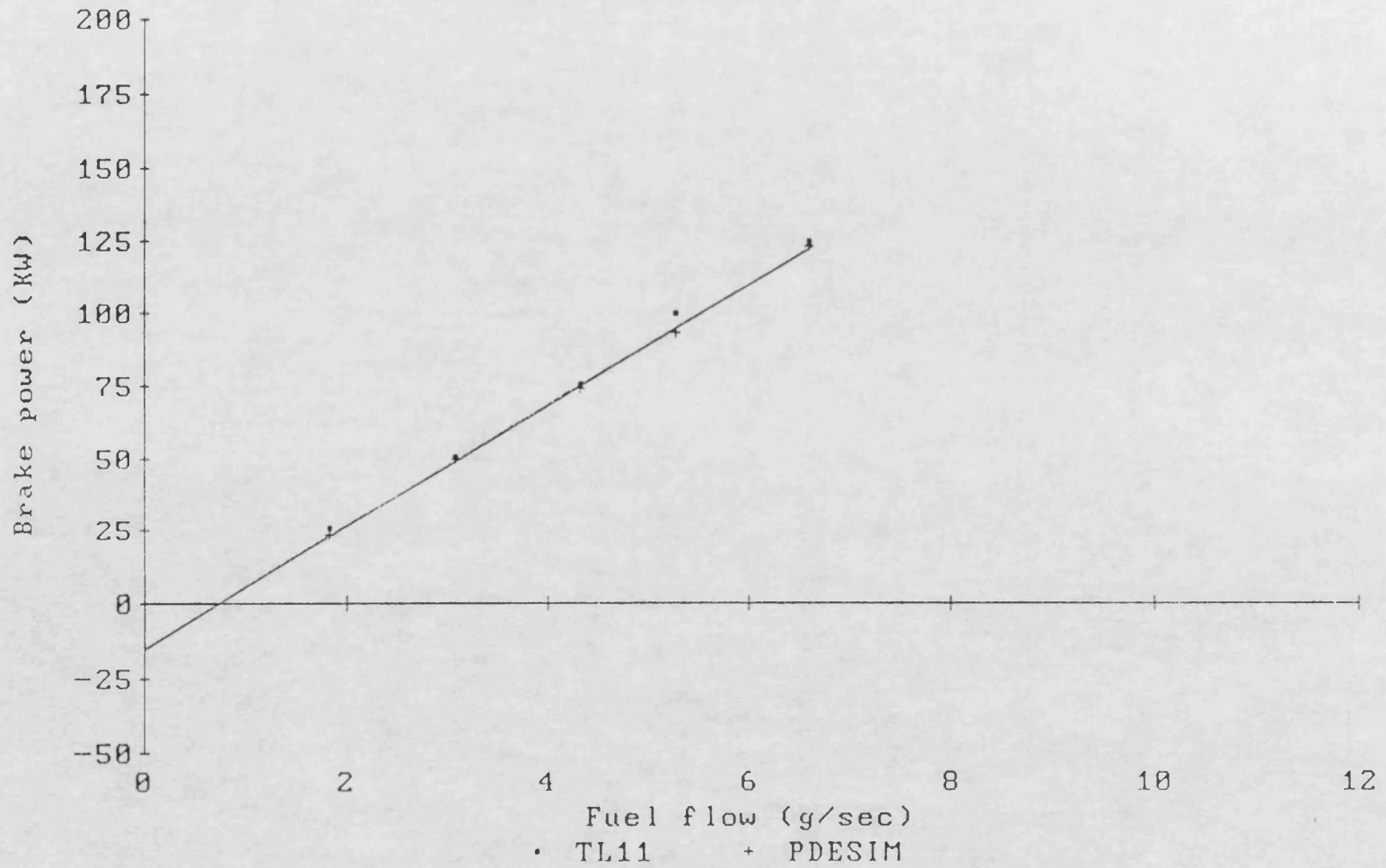


Figure 7.4: A Willan's line superimposed on the brake power results from the PDESIM and the TL11 Diesel engine at 1200 rpm

Parallel Diesel Engine Simulation  
1600 rpm (Garrett Turbocharger)

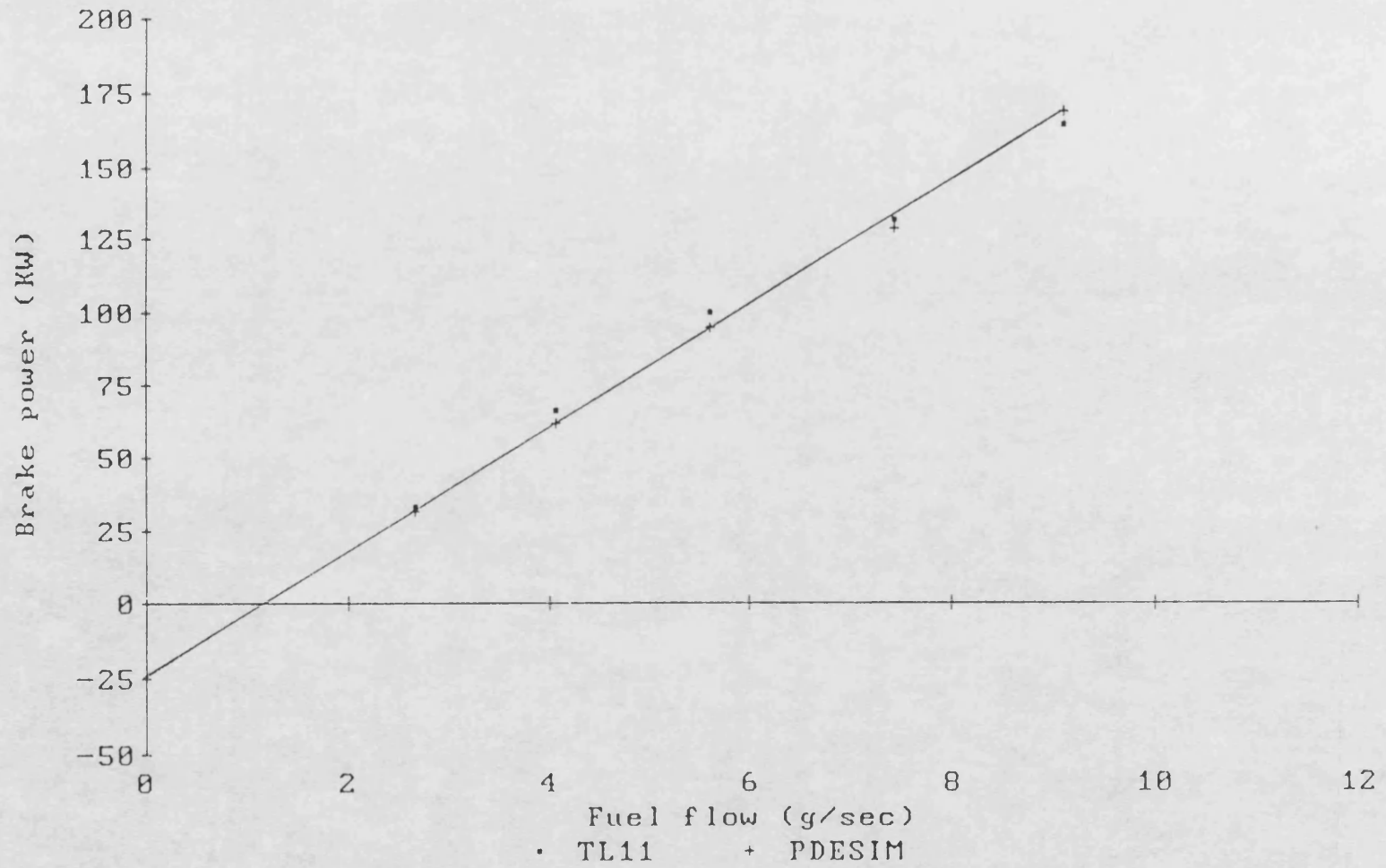


Figure 7.4d A Willan's line superimposed on the brake power results from the PDESIM and the TL11 Diesel engine at 1600 rpm

# Chapter 8

# **Application of the Engine Simulation**

## **8.1 Introduction**

Engine simulation has always been used in a number of applications, but its main use has been in the areas of engine design and performance comparisons where the relative response of the engine simulation is not important. With the introduction of high speed computers and the application of parallel processing, the simulation time has been considerably reduced. This has opened new potential applications areas where the time of simulation is relatively important when compared with the actual engine response. The following sections describe some of the possible applications of the Diesel engine simulation.

## **8.2 Simulation as a Design Tool**

Existing engine simulation using complex models is slow and can limit the use of the engine simulation as a design tool. With a fast simulation, the response time to an action is very short. An interactive control of a simulated engine can be given to the designer, who can change different engine parameters interactively and see their effects on different performance parameters before going into detailed prototype design of the engine. This can lead to experiments with novel designs which may be expensive to try using real hardware. Simulation can help the understanding of combustion and an efficient engine design with less combustion products released to exhaust may easily be designed. The parallel Diesel engine simulation developed in this thesis has the ability to allow some of the parameters to be modified and their effects recorded.



## 8.3 Condition Monitoring and Fault Diagnosis

A number of maintenance techniques are employed to keep an engine running. One way is to replace or repair when failure occurs. This is not always feasible, particularly in cases where continuity of operation is required, such as in case of ship engines. Another method is to do maintenance at planned intervals and replace components on 'running hour' bases. This can lead to replacement of those components which are still healthy and hence can contribute an overall cost increase and produce inefficiency of the system. A third type of maintenance method is based on condition of the engine, and is called condition monitoring [66]. Condition monitoring is considered the most reliable, cost effective and efficient technique for maintaining an engine. It improves safety and gives more running time and less maintenance time.

Condition monitoring relies on objective assessment and quantitative measurements of the engine by taking regular measurements and analysing them. This is achieved by comparing different engine parameters with already established known values of these parameters for a healthy engine.

The requirements to compare the engine results with a reference provides some difficulties. Uitermarkt [67] notes that a reference database of an engine test-bed results for a large ship engine would probably be incomplete due to the huge variations in operating and environmental conditions that such an engine would meet. Introducing a complete set of operating conditions will increase the size of the database to a considerable extent. A fast engine simulation system would help to overcome this.

Another aspect of condition monitoring is fault diagnosis. Faults can be simulated using a complex engine model and the results stored in a database file. A complete

condition monitoring system could then compare these simulated faulty results with those from an engine and could predict a possible fault in a particular part of the engine.

It is envisaged that an on-board condition monitoring and fault diagnostic system would be used in conjunction with an expert system. Katsoulakos et al [68] describe such a system. Simple engine condition monitoring using test-bed results has already been used commercially. A model that is based on test-bed results and makes adjustments to these to take account of operating conditions has been implemented by the company Mak [69].

## **8.4 Engine Control**

Testing new control strategies using real engines is costly because of high amount of instrumentation involved. By using a high speed simulation, new management and control systems can be investigated before applying these techniques to the real engine. This can provide a reduction in cost and effort just by reducing the number of trials on a real engine before perfecting the method.

## **8.5 Summary**

The application of high speed simulation to Diesel engine design and subsequent control and maintenance offers a great deal of economic and environmental advantages. A combination of condition monitoring and control applied in conjunction with expert systems may develop into a semi-auto control system that will be applied to monitor big engines both in off-shore applications such as ship and in inland applications such as power generation.

# Chapter 9

# Conclusions

A Diesel engine simulation has been implemented using the distributed memory features of the Bath University transputer based parallel computer. A filling and emptying model is used to simulate a Diesel engine, which treats a Diesel engine as a thermodynamic system consisting of cylinder and manifold control volumes, junctions that interconnect these control volumes, and shafts that transfer power generated by the engine. An advantage of this method is that a direct correlation exists between the real engine and the simulation, and using cylinders and manifolds as basic control volume modules almost any type of engine configuration can be studied.

The filling and emptying model also provides lesser communication between cylinder and manifold tasks as these control volumes are relatively independent and share only those variables which affect the flow between them.

A new flexible data entry system has been developed that helps the user to define the topology of an engine and initialise different variables of the engine simulation. This system also provide the facility to change different parameters of the engine without changing the actual code. It is also possible for the user to change some control parameters interactively and see their effect on the engine performance.

Control and synchronisation within the simulation is distributed across all of its constituent tasks. Thus, once the simulation is initialised, synchronisation and data transfer occurs as and when required by a particular task.

Different communication functions are provided with the operating system Helios, but these are slow and do not satisfy the fast data communication requirements for the Diesel engine simulation. New communication primitives have been developed using

the backplane features of the new computer which are faster than the Helios routines. These new routines have been used for data transfer and synchronisation within the Diesel engine simulation.

The new parallel Diesel engine simulator has been compared with a previous but slower, simulator. Results from the two simulators were taken for varying fuelling over a range of speeds. Engine maps were plotted and the results were found in agreement.

The results from the new simulation have also been compared with results from a real engine. An extremely promising correlation has been shown to exist between theoretical and experimental results for different speed and fuelling conditions.

# Chapter 10

## Further Work

The new parallel Diesel engine simulation implemented using transputers has demonstrated a number of areas where further investigation may be carried out. One of these is the addition of a database type input to the simulation that defines the topology of the engine and initialises different operating and environmental variables. This can be extended further by combining the simulation with an expert system to provide a better and faster engine management system for condition monitoring and fault diagnosis of large industrial and ship engines.

At present, a turbocharged engine can be modelled with a variable number of control volumes. An extension to this facility would be to be able to select any number of turbochargers.

In the present simulation, only one crankshaft and one turbocharger shaft are allowed. The addition of more shafts will add more flexibility in defining the overall topology of the engine and it could be possible to simulate any thermodynamic control system beside a complete Diesel engine.

The empirical models used in the simulation have values for the constants which are suggested by the authors of these models. To simulate a particular engine, more appropriate constants should be found that may improve the actual model as well.

A natural enhancement to the simulation will be the addition of a user friendly man-machine interface for plotting different parameters of the simulation.

Technology is advancing at a great pace and new and fast processors are appearing in the market. An obvious enhancement to the present work would be to adopt it accordingly so that the latest computing systems available may be utilised. For

example, further work could be carried out to port the simulation developed in this thesis to an Intel's i860 based system which would promise a greater speed of simulation.



# References

# References

1. Asimov, I., *The Book of Facts*, Coronet Books, 1979.
2. de Cogan, D., *Design and technology of integrated circuits.*, John Wiley, 1990, pp 1-7.
3. Margulis, N., "i860 microprocessor internal architecture", *Microprocessors and microsystems*, Vol. 14, No. 2, March 1990, pp 89-96.
4. Atherton, W. A., "Pioneers", *Electronics & Wireless World*, 1987, pp 1213-1214.
5. Morrison, P., Morrison, E., *Charles Babbage and his calculating engines.*, N.Y., 1961.
6. Burks, A. W., "The ENIAC: First general-purpose electronic computer", *Ann. Hist. Comput.*, Vol. 3, No. 4, pp 310-399.
7. Wilkes, M. V., Renwick, W., "The EDSAC, an electronic calculating machine", *J. Sci. Instrum.*, Vol. 26, 1949, pp 385-391
8. Treleaven, P. C., "Future parallel computers", *Lecture notes in computer science - 237, CONPAR-86*, pp 41-47.
9. Wilkinson, J. H., "The pilot ACE", *Computer structures: Readings and examples*, Chapter 11, Ed. Bell, C. G. and Newell, A., N.Y., McGraw-Hill, 1953, pp 193-199.
10. Helay, A. C. D., "DEUCE: a high speed general purpose computer", *Proc. I.E.E.*, Vol. 103, Part B, Suppl. 2, 1956, pp 165-173.
11. Smith Associates, *Intl. Conf. on Parallel Processing*, The Royal Society, London, 9th-10th Dec., 1987.
12. Payne, M., "Multiple transputers will enhance Marconi radars", *Electronics Weekly*, June 27, 1990, p 12.
13. Parsytec, "British Aerospace goes parallel with Parsytec", *S.E.R.C./D.T.I. transputer initiative*, Mailshot, June 1990, pp 30-31.

14. Watson, N., "Computers in Diesel engine turbocharging system design", *I.Mech.E.*, C05/87, 1987, pp 269-280.
15. Charlton, S. J., *SPICE: Simulation program for internal combustion engines.*, Bath University, U.K., 1986.
16. Katsoulakos, P. S., Hornsby, P. P. W., Zanconato, R., "DEEDS: The Diesel engine expert diagnostic system", *Maritime communication and control*, 26-28 Oct., 1988, Paper 22.
17. Tomisawa, N., Toki, S, "Trends in electronic engine control and development of optimum microcomputers", S.A.E. Paper No. 880136.
18. Clark, C. A., "Electronics applied to systems using internal combustion engines; from lawnmowers to low speed marine engines", *I.Mech.E. Seminar: The benefits of electronic control systems for internal combustion engines.*, Jan., 1989.
19. Gerrett, K., "Microprocessor control for Diesel engines", *Automotive Engineer*, Vol.15, No. 4, Aug./Sep. 1980, pp21-25.
20. Jones, A. D., *The application of parallel processing to diesel engine modelling.*, Ph.D. Thesis, University of Bath, 1987.
21. Haysom, F. J., *Enhanced performance simulation of diesel engines.*, Ph.D. Thesis, University of Bath, 1989.
22. Ramos, J. I., "Mathematical models of Diesel engines", *Computer simulation for fluid flow, heat and mass transfer, and combustion in reciprocating engines*, Ed. Markatos, N. C., N.Y., 1989, pp 67-130.
23. Henein, N. A., Bolt, J. A., "Ignition delay in Diesel engines", S.A.E. Paper No. 670007.
24. Wolfer, H. H., "Ignition lag in Diesel engines", Translated by Mullins, M. F., Royal Aircraft Establishment Library No. 358.
25. Watson, N., "Combustion and gas properties", Report E17, Imperial college of science and technology, London, Sep., 1979.

26. Watson, N, Pilley, A. D., Marzouk, M., "A combustion correlation for Diesel engine simulation", S.A.E. Paper No. 800029.
27. Watson, N., Janota, M. S., "Modelling", Chapter 15, Turbocharging the internal combustion engine, MacMillan, 1982, p 537.
28. Hohenberg, G. F., "Advanced approaches for heat transfer calculations", SAE/SP-79/449, 1979.
29. Woschni, G., "A universally applicable equation for the instantaneous heat transfer coefficient in the internal combustion engine", S.A.E. Paper No. 670931.
30. Chen, S. K., Flynn, P. F., "Development of a single cylinder compression ignition research engine", S.A.E. Paper No. 650733.
31. Berry, T., Personal communications.
32. Data Sheet, "IMS T800 transputer", Inmos, 1987.
33. Dunn, R. W., Daniels, A. R., Gott, V. S., Selwyn, C. G., "A new architecture of high performance parallel computer for use in condition monitoring of large diesel engines", I.E.E. Conf. Publication 309, Sep. 1989.
34. Daniels, A. R., Dunn, R. W., Gott, V. S., Selwyn, C. G., "Real time simulation of diesel engines using the T800 transputer", S.E.R.C./D.T.I. Transputer Initiative workshop on Transputer development environment, 1987.
35. Meiko Scientific, 650 Aztec West, Almondsbury, Bristol, U.K.
36. The Transputer Family, Inmos, 1986
37. Helios User's Manual, Perihelion Software, 1988.
38. Daley, M. B., A link topology controller for a sixteen transputer multiprocessor system., B. Sc. (Elect) Thesis, University of Bath, 1988.
39. Hafeez, M., An expandable input/output and graphics system for distributed memory parallel computers., Ph.D. Thesis, University of Bath, 1990.
40. Dale, L. A., Real time modelling of multi-machine power systems., Ph.D. thesis, University of Bath, 1986.

41. Berry, T., Real time modelling of complex power systems using parallel processing., Ph.D. Thesis, University of Bath, 1989.
42. Introduction to Tripos, Metacomco, Issue May 1986.
43. Tripos programmer's reference Manual, Metacomco, Issue 1986.
44. Sargent, P., "Tripos", *Personal Computer World*, June 1986, pp 140-147.
45. May, D., "Occam", *SIGPLAN Notices*, Vol. 18, No. 4, April 1983, pp 69-79.
46. Pountain, D., "Occam II", *Byte*, Oct. 1989, pp 279-284.
47. Grimsdale, C. H. R., "Distributed operating system for transputers", *Microprocessors and microsystems*, Vol. 13, No. 2, March 1989.
48. Helios Technical Manual, Perihelion Software, 1988.
49. Helios Developer's Manual, Perihelion Software, 1988.
50. Powell, J., Garnett, N., "Helios performance measurement", Technical report No. 22, Perihelion Software, Feb. 1990.
51. Chapra, S. C., Canale, R. D., Numerical methods for engineers, Second edition, McGraw-Hill, N.Y., 1989, pp 565-705.
52. Franklin, M. A., "Parallel solution of ordinary differential equations", *I.E.E.E. Transactions on Computers*, Vol. C-27, No. 5, May 1978, pp 413-420.
53. Kerckhoffs, E. J. H., "Parallel algorithms for ordinary differential equations: An introductory review", *Proceedings 1986 Summer Computer Simulation Conference*, Reno, NV, U.S.A., 28-30 July 1986, Eds: Crosbie, R., Luker, P., pp 947-952.
54. Worland, P. B., "Parallel methods for the numerical solution of ordinary differential equations", *I.E.E.E. Transactions on Computers*, Oct. 1976, pp 1045-1048.
55. Birta, L. G., Abou-Rabia, O., "Parallel block predictor-corrector methods for ode's", *I.E.E.E. Transactions on Computers*, Vol. C-36, No. 3, March 1987, pp 299-311.
56. Abou-Rabia, O., Multiprocessing in continuous system simulation., Ph.D. Thesis, University of Ottawa, Canada, 1985.

57. Barton, P., Willers, I. M., Zahar, R. V. M., "Taylor series methods for ordinary differential equations - An evaluation", *Mathematical software*, Ed. Rice, J. R., Academic Press, 1971, pp 369-390.
58. Halin, J. H., "Integration across discontinuities in ordinary differential equations using power series", *Simulation*, Feb. 1979, pp 33-45.
59. Halin, J. H., Buhner, R., Halg, W., Benz, H., Bron, B., Brundiers, H. J., Isacson, A., Tadian, M., "The ETH multiprocessor project: Parallel simulation of continuous systems", *Simulation*, Oct. 1980, pp 109-123.
60. Abou-Rabia, O., Birta, L. G., "Some variations on the BPC parallel integration method", *Proceedings 1987 summer computer simulation conference*, Montreal, Que., Canada, July 1987, Ed. Chou, J. Q. B., pp 37-42.
61. Greene, A. B., Lucas, G. G., *The testing of internal combustion engines.*, The English Universities Press Ltd., London, 1969.
62. Roberts, E. W., *Variable geometry turboengine optimisation and control.*, Ph.D. Thesis, University of Bath, 1984.
63. *Leyland TL11 Instrumentation Manuals*, Wolfson Laboratory, School of Mechanical Engineering, University of Bath.
64. Scaife, M. W., *An experimental facility for the development of intelligent engine diagnostics.*, M.Phil. Thesis, University of Bath, 1990.
65. Scaife, M. W., *Personal communications*, 1990.
66. Haddad, S. D., "Condition monitoring and fault diagnosis in Diesel engines", in *'Principles and performance in Diesel engineering'*, Eds: Haddad, S. D., Watson, N., Ellis Harwood Limited, Chichester, U.K., 1984, pp 246-277.
67. Uitermakt, R. W. P., "Engine fault diagnosis", *Marine engine review*, July 1984, pp 8-10.
68. Katsoulakos, P. S., Newland, J., Stansfield, J. T., Ruston, T., "Monitoring databases and expert systems in the development of engine fault diagnostics", *Journal of non-destructive testing*, 1988, Vol. 30, No. 4, pp 263-273.

69. Anonymous, "Engine history at Dicare's heart", *The motor ship*, July 1987, p 26.

# Appendices



# Appendix A

## Solution of Ordinary Differential Equations Using Power Series

**A.1** For  $y = f(t)$  Taylor series is defined as

$$f(t_i + \Delta t) = f(t_i) + \Delta t f'(t_i) + \frac{(\Delta t)^2}{2!} f''(t_i) + \frac{(\Delta t)^3}{3!} f'''(t_i) + \dots$$

where  $\Delta t = t - t_i$  or  $t = t_i + \Delta t$

Rewriting the above equation

$$f(t) = f^{(0)}(t_i) + \Delta t f^{(1)}(t_i) + \frac{(\Delta t)^2}{2!} f^{(2)}(t_i) + \frac{(\Delta t)^3}{3!} f^{(3)}(t_i) + \dots$$

where  $f^{(k)}(t)$  is the  $k$ th derivative of  $f(t)$ .

In general

$$f(t) = \sum_{k=0}^{k=\infty} f^{(k)}(t_i) \cdot \frac{(\Delta t)^k}{k!}$$

Substituting  $y_k = f^{(k)}(t_i) \cdot (\Delta t)^k / k!$ , the general solution  $y = f(t)$  can be written as

$$y = y_0 + y_1 + y_2 + y_3 + \dots$$

**A.2** Let the example set of ordinary differential equations be

$$y_1^{(1)} = y_2 \quad \text{A.1a}$$

$$y_2^{(1)} = \frac{-y_1}{(y_1^2 + y_2^2)^{3/2}} \quad \text{A.1b}$$

where both  $y_1$  and  $y_2$  are functions of  $t$ .

The initial conditions are

$$y_1(0) = 1$$

$$y_2(0) = 0$$

The solution of the above equations is

$$y_1 = \cos(t) \quad \text{A.2a}$$

$$y_2 = \sin(t) \quad \text{A.2b}$$

**A.3** To integrate these initial value first order ordinary differential equations using Taylor series, these equations are required to be broken down into set of equations involving two variables only. To do so new auxiliary variables are introduced. These sub equations are

$$y_1^{(1)} = y_2 \quad \text{A.3a}$$

$$a_0 = -y_1 \quad \text{A.3b}$$

$$a_1 = y_1^2 \quad \text{A.3c}$$

$$a_2 = y_2^2 \quad \text{A.3d}$$

$$a_3 = a_1 + a_2 \quad \text{A.3e}$$

$$a_4 = (a_3)^{3/2} \quad \text{A.3f}$$

$$a_5 = a_0 / a_4 \quad \text{A.3g}$$

$$y_2^{(1)} = a_5 \quad \text{A.3h}$$

In the following sections, it is assumed that  $y_{10}$  represents 0th derivative of  $y_1$ ,  $y_{20}$  represents 0th derivative of  $y_2$ ,  $a_{10}$  represents 0th derivative of  $a_1$ , and so on.

Selecting the number of terms  $k=4$  and the interval  $\Delta t=h$  for the Taylor series, the solution will be:

#### A.4 Formula:

$$\text{For } r^{(1)} = p \quad \text{A.4a}$$

the Taylor series terms are

$$r_{k+1} = (h/(k+1)) p_k \quad \text{A.4b}$$

Applying this formula to equation A.3a gives

$$y_{10} = 1 \quad \text{A.5a}$$

$$k=0 \quad y_{11} = h y_{20} \quad \text{A.5b}$$

$$k=1 \quad y_{12} = (h/2) y_{21} \quad \text{A.5c}$$

$$k=2 \quad y_{13} = (h/3) y_{22} \quad \text{A.5d}$$

$$k=3 \quad y_{14} = (h/4) y_{23} \quad \text{A.5e}$$

$$k=4 \quad y_{15} = (h/5) y_{24} \quad \text{A.5f}$$

#### A.5 Formula:

$$r = \text{constant} \cdot p \quad \text{A.6a}$$

the Taylor series terms are

$$r_k = \text{constant} \cdot p_k \quad \text{A.6b}$$

Applying this formula to equation A.3b gives

$$k=0 \quad a_{00} = \dots y_{10} \quad \text{A.7a}$$

$$k=1 \quad a_{01} = \dots y_{11} \quad \text{A.7b}$$

$$k=2 \quad a_{02} = \dots y_{12} \quad \text{A.7c}$$

$$k=3 \quad a_{03} = \dots y_{13} \quad \text{A.7d}$$

$$k=4 \quad a_{04} = \dots y_{14} \quad \text{A.7e}$$

## A.6 Formula:

$$r = p^2 \quad \text{A.8a}$$

the Taylor series terms are

$$r_k = p_0 \cdot p_0 \quad (k = 0)$$

$$r_k = 2 \sum_{s=0}^{(k-1)/2} p_s p_{k-s} \quad (k=\text{odd and } k>0)$$

$$r_k = 2 \sum_{s=0}^{(k-2)/2} p_s p_{k-s} \quad (k=\text{even and } k>0)$$

A.8b

Applying this formula to equation A.3c gives

$$k=0 \quad a_{10} = y_{10} y_{10} \quad \text{A.9a}$$

$$k=1 \quad a_{11} = 2 (y_{10} y_{11}) \quad \text{A.9b}$$

$$k=2 \quad a_{12} = 2 (y_{10} y_{12}) + y_{11} y_{11} \quad \text{A.9c}$$

$$k=3 \quad a_{13} = 2 (y_{10} y_{13} + y_{11} y_{12}) \quad \text{A.9d}$$

$$k=4 \quad a_{14} = 2 (y_{10} y_{14} + y_{11} y_{13}) + y_{12} y_{12} \quad \text{A.9e}$$

**A.7** Applying the formula A.8 to equation A.3d gives

$$k=0 \quad a_{20} = y_{20} y_{20} \quad \text{A.9a}$$

$$k=1 \quad a_{21} = 2 (y_{20} y_{21}) \quad \text{A.9b}$$

$$k=2 \quad a_{22} = 2 (y_{20} y_{22}) + y_{21} y_{21} \quad \text{A.9c}$$

$$k=3 \quad a_{23} = 2 (y_{20} y_{23} + y_{21} y_{22}) \quad \text{A.9d}$$

$$k=4 \quad a_{24} = 2 (y_{20} y_{24} + y_{21} y_{23}) + y_{22} y_{22} \quad \text{A.9e}$$

**A.8** Formula:

$$r = p + q \quad \text{A.10a}$$

the Taylor series terms are

$$r_k = p_k + q_k \quad \text{A.10b}$$

Applying this formula to the equation A.3e gives

$$k=0 \quad a_{30} = a_{10} + a_{20} \quad \text{A.11a}$$

$$k=1 \quad a_{31} = a_{11} + a_{21} \quad \text{A.11b}$$

$$k=2 \quad a_{32} = a_{12} + a_{22} \quad \text{A.11c}$$

$$k=3 \quad a_{33} = a_{13} + a_{23} \quad \text{A.11d}$$

$$k=4 \quad a_{34} = a_{14} + a_{24} \quad \text{A.11e}$$

**A.9** Formula:

$$r_k = p^{\text{constant}} \quad \text{A.1}$$

2a

the Taylor series terms are

$$r_k = p^{\text{constant}} \quad (k=0)$$

$$r_k = \frac{1}{k p_0} \left[ \sum_{s=1}^k (\text{constant} \cdot k - (\text{constant}+1) \cdot s) \right] \quad (k>0)$$

A.12b

Applying this formula to equation A.3f gives

$$k=0 \quad a_{40} = a_3^{1.5} \quad \text{A.13a}$$

$$k=1 \quad a_{41} = 1.5 a_3^{0.5} a_{31} \quad \text{A.13b}$$

$$k=2 \quad a_{42} = 3 y_{12} + 1.5 (y_{21})^2 \quad \text{A.13c}$$

$$k=3 \quad a_{43} = 1.5 a_{33} \quad \text{A.13d}$$

$$k=4 \quad a_{44} = 1.5 a_{34} + 0.75 y_{12} a_{32} + 0.375 (y_{21})^2 a_{32} \quad \text{A.13e}$$

### A.10 Formula:

$$r_k = p/q \quad \text{A.14a}$$

the Taylor series terms are

$$r_k = p_0 / q_0 \quad (k=0)$$

$$r_k = \frac{1}{q_0} \left[ p_k - \sum_{s=1}^k (q_s \cdot r_{k-s}) \right] \quad (k>0)$$

A.14b

Applying this formula to equation A.3g gives

$$k=0 \quad a_{50} = a_{00} / a_{40} \quad \text{A.15a}$$

$$k=1 \quad a_{51} = (a_{01} - a_{41} a_{00}/a_{40}) / a_{40} \quad \text{A.15b}$$

$$k=2 \quad a_{52} = 2 y_{12} + 1.5 (y_{21})^2 \quad \text{A.15c}$$

$$k=3 \quad a_{53} = 2 y_{13} + 3 y_{21} y_{22} \quad \text{A.15d}$$

$$k=4 \quad a_{54} = -y_{14} - (3 y_{12} + 1.5 (y_{21})^2) \cdot (2 y_{12} + 1.5 (y_{21})^2) + a_{44} \quad \text{A.15e}$$

**A.11** Applying formula A.4 to equation A.3h gives

$$y_{20} = 0 \quad \text{A.16a}$$

$$k=0 \quad y_{21} = h a_{50} \quad \text{A.16b}$$

$$k=1 \quad y_{22} = h a_{51} / 2 \quad \text{A.16c}$$

$$k=2 \quad y_{23} = h a_{52} / 3 \quad \text{A.16d}$$

$$k=3 \quad y_{24} = h a_{53} / 4 \quad \text{A.16e}$$

$$k=4 \quad y_{25} = h a_{54} / 5 \quad \text{A.16f}$$

**A.12** Solving the above simultaneous equations and substituting in the Taylor series gives

$$\begin{aligned} y_1 &= y_{10} + y_{11} + y_{12} + y_{13} + y_{14} + \dots \\ &= 1 - h^2/2 + h^4/24 + \dots \end{aligned}$$

and

$$\begin{aligned} y_2 &= y_{20} + y_{21} + y_{22} + y_{23} + y_{24} + \dots \\ &= -h + h^3/6 + \dots \end{aligned}$$

Hence if  $h=0.1$  then

$$y_1 = 0.99500416$$

and

$$y_2 = -0.09983333$$

The actual values are

$$y_1 = \cos(0.1) = 0.99500416$$

and

$$y_2 = -\sin(0.1) = -0.09983341$$

## A.13 References:

Barton, P., Willers, I. M., Zahar, R. V. M., "Taylor series methods for ordinary differential equations - An evaluation", Mathematical software, Ed. Rice, J. R., Academic Press, 1971, pp 369-390.

Halin, J. H., "Integration across discontinuities in ordinary differential equations using power series", Simulation, Feb.. 1979, pp 33-45.

Halin, J. H., Buhner, R., Halg, W., Benz, H., Bron, B., Brundiers, H., J., Isacson, A., Tadian, M., "The ETH multiprocessor project: Parallel simulation of continuous systems", Simulation, Oct. 1980, pp 109-123.



# Appendix B

## An Example Data File For Processor Allocation

```
/*
*-----*
*   First column will be ignored.           *
*   Second column gives the name of the task to be loaded. *
*   Third column is the processor name string. *
*   The engine task should be loaded remotely on a processor *
*   other than those given in this list. *
*-----*
*/
cyl1  cylinder    04
cyl2  cylinder    05
cyl3  cylinder    06
cyl4  cylinder    07
cyl5  cylinder    08
cyl6  cylinder    09
man1  manifold    10
man2  manifold    11
man3  manifold    12
act   actuator    13
dyn   dynamics    14
stab  stability    15
disp  display     03
```

# Appendix C

## Data Preparation For the PDESIM

There are three different data files for the PDESIM. The first data file describes the Diesel engine itself. The second file gives intercooler and compressor data and the third file provides turbine data to the PDESIM.

### C.1 Main Data File:

The main data file for the PDESIM is divided into a number of sections. Each section of data must be preceded by a comment terminated by '<space>\*/'. The comment may cover any number of lines. The maximum length of any string in a comment line is limited to 80 characters.

Any variable preceded by the abbreviation 'int' must be written as an integer. The numbers may be delimited by any number of blanks or new lines unless otherwise stated.

The volume numbering must start from 1 and manifolds should be preceded by cylinders. Numbering of all other data sets should start from zero.

Each section of the data file is explained below individually.

#### C.1.1 Title and Control Data:

Title string may be of any length and includes the title string of the control data as well. The control data has four arguments. All of these arguments are zero as dynamic data logging is provided in the PDESIM.

- o arg 1 (int) number of cycles for simulation
- o arg 2 (int) number of first cycle to be printed out
- o arg 3 (int) number of last cycle to be printed out
- o arg 4 (int) interval between printout cycles

### C.1.2 System Data:

This section provides information about the system components. The maximum number of control volumes is limited. At present only one compressor and turbine may be connected to the system, and since the governor and load data is not being used, no data is provided for these sections.

- o arg 1 (int) number of cylinders (1 to 10)
- o arg 2 (int) number of manifolds (1 to 5)
- o arg 3 (int) number of junctions (1 to 20)
- o arg 4 (int) number of compressors (1)
- o arg 5 (int) number of turbines (1)
- o arg 6 (int) number of poppet valve sets (8)
- o arg 7 (int) number of timing sets (1)
- o arg 8 (int) number of heat release sets (10)
- o arg 9 (int) number of cylinder heat transfer sets (2)
- o arg 10 (int) number of manifold heat transfer sets (3)
- o arg 11 (int) number of shaft sets (2)
- o arg 12 (int) number of governor sets (0)
- o arg 13 (int) number of load sets (0)

### C.1.3 Cylinder Data:

Data for each cylinder is provided in a line and each line consists of nineteen arguments. At present it is assumed that all cylinders have only one inlet and one exhaust valve. The offset angle of cylinder 1 is taken as zero and all other cylinders are referenced from it.

- o line 1 data related to cylinder 1
- o line 2 data related to cylinder 2
- o .
- o .
- o .
- o line n data related to cylinder n
  
- o arg 1 (int) volume number (1 to n)
- o arg 2 (int) volume type (1=cylinder, 2,3=manifold)
- o arg 3 (int) task type (2=cylinder, 3=inlet manifold, 4=exhaust manifold)
  
- o arg 4 (int) cylinder fault (0, No fault at present)
- o arg 5 (---) cylinder bore (m)
- o arg 6 (---) cylinder stroke (m)
- o arg 7 (---) compression ratio
- o arg 8 (---) offset angle with respect to cylinder 1 (deg)
- o arg 9 (---) connecting rod length (m)
- o arg 10 (---) piston mass (kg) (Not used)
- o arg 11 (---) friction mean effective pressure (bar, or < 0.0 if to be calculated by the program)
  
- o arg 12 (int) heat release set
- o arg 13 (int) heat transfer set

- o arg 14 (int) shaft set associated with cylinder
- o arg 15 (int) timing set associated with cylinder
- o arg 16 (int) number of inlet junction (1)
- o arg 17 (int) number of exhaust junction (1)
- o arg 18 (int) array of inlet junction numbers
- o arg 19 (int) array of exhaust junction numbers

### C.1.4 Manifold Data:

Data for each manifold is provided in a line and each line consists of eight arguments and two arrays.

- o line 1 data related to manifold 1
- o line 2 data related to manifold 2
- o .
- o .
- o .
- o line m data related to manifold m
- o arg 1 (int) volume number (n+1 to n+m)
- o arg 2 (int) volume type (1=cylinder, 2,3=manifold)
- o arg 3 (int) task type (2=cylinder, 3=inlet manifold, 4=exhaust manifold)
- o arg 4 (int) associated turbocharger number (0)
- o arg 5 (---) heat transfer set associated with manifold (-1 for no heat transfer)
- o arg 6 (---) volume ( $m^3$ )
- o arg 7 (int) number of junctions where flow is normally in

- o arg 8 (int) number of junctions where flow is normally out
- o arg 9 (int) array of in junction numbers
- o arg 10 (int) array of out junction numbers

### C.1.5 Junction Data:

Data for each junction is provided in a line and each line consists of seven arguments.

At present a twin entry turbine is modelled for the PDESIM.

- o line 1 data for junction 0
- o line 2 data for junction 1
- o .
- o .
- o .
- o line j data for junction j-1
  
- o arg 1 (int) junction number (0 to 20)
- o arg 2 (int) junction type (1=poppet, 2=orifice, 3=compressor, 4=turbine)
- o arg 3 (---) effective area ( $m^2$ ) (Not used)
- o arg 4 (---) coefficient of discharge (Not used)
- o arg 5 (int) poppet valve set if poppet valve  
(int) 0 if compressor  
(int) first turbine entry junction number if second turbine for a twin entry turbine
- o arg 6 (int) entry volume number assuming normal flow direction or highest volume number plus one if the entry volume is atmosphere.
- o arg 7 (int) exit volume number assuming normal flow direction or highest volume number plus two if the exit volume is atmosphere.

### C.1.6 Heat Release Data:

Heat release data is provided in lines. Each line gives data for the corresponding heat release set. A cylinder may have any one of the data set for the calculation of its heat release.

- o line 1 data relating to heat release set 0
- o line 2 data relating to heat release set 1
- .
- .
- .
- o line r data relating to heat release set r-1
  
- o arg 1 (string) name of the heat release model (Watson or Wiebe)
- o arg 2 (-----) calorific value of fuel (J/kg K)
- o arg 3 (-----) duration of combustion in degrees
- o arg 4 (-----) fuel pipe length in meters
- o arg 5 (-----) Wiebe combustion coefficient C1 (0.0 for Watson)
- o arg 6 (-----) Wiebe combustion coefficient C2 (0.0 for Watson)

### C.1.7 Cylinder Heat Transfer Data:

This data is also divided into lines, each line providing data for the corresponding cylinder heat transfer set. For the PDESIM, a uniform temperature distribution is assumed over the entire surface of the cylinder and an average value of the wall temperature is used to calculate heat transfer through the cylinder walls. This wall temperature is dynamically calculated by the program and is initialised by the value given by arg 4.

- o line 1 data relating to cylinder heat transfer set 0
- o line 2 data relating to cylinder heat transfer set 1
- .
- .
- .
- o line t data relating to cylinder heat transfer set t-1
  
- o arg 1 (string) name of the cylinder heat transfer model (Hohen or  
Woschni)
- o arg 2 (int ) number of heat transfer areas (1)
- o arg 3 (-----) area array (0.0)
- o arg 4 (-----) temperature array corresponding to above areas,  
(initial wall temperature in Kelvin)

### C.1.8 Manifold Heat Transfer Data:

This data is also divided into lines, each line providing data for the corresponding manifold heat transfer set. In the PDESIM, a uniform temperature distribution is assumed over the entire surface of the manifold and an average value of the wall temperature is used to calculate heat transfer through the manifold walls. This wall temperature is dynamically calculated by the program provided it is selected in the manifold data. The wall temperature is initialised by the value given by arg 6.

- o line 1 data relating to manifold heat transfer set 0
- o line 2 data relating to manifold heat transfer set 1
- .
- .
- .
- o line u data relating to manifold heat transfer set u-1



- o arg 1 (string) name of the manifold heat transfer model (ht\_simple)
- o arg 2 (-----) surface area of the manifold in  $m^2$
- o arg 3 (-----) heat transfer coefficient of exhaust gas
- o arg 4 (-----) thermal resistance of manifold wall
- o arg 5 (-----) thermal capacitance of manifold wall
- o arg 6 (-----) initial temperature of manifold wall (K)

### C.1.9 Poppet Valve Data:

Poppet valve data is divided into section of sets. Each section consists of four arguments followed by two arrays, one for crank angle and the other for corresponding cross sectional area of the valve.

- o arg 1 (int) number of values in each crank angle and area array (max. 50)
- o arg 2 (---) shift factor in degrees to shift the camshaft diagram (0 for no shift)
- o arg 3 (---) stretch factor to scale the duration of the valve opening(1.0 for no scaling)
- o arg 4 (---) area scale factor to scale the valve area (1.0 for no scaling)
- o arg 5 (---) array of crank angles in degrees
- o arg 6 (---) array of corresponding valve areas in  $m^2$

### C.1.10 Shaft Data:

Each line in this section provides data for the corresponding shaft and has seven arguments and an array which gives the corresponding connected control volumes or junctions.

- o line 1 data for shaft 0
- o line 2 data for shaft 1
- o .
- o .
- o .
- o line s data for shaft s-1
  
- o arg 1 (int) shaft number
- o arg 2 (int) shaft type (1=crank, 2=turbo)
- o arg 3 (int) speed dynamics (0) (Not used)
- o arg 4 (---) shaft inertia ( $\text{kg/m}^2$ )
- o arg 5 (---) initial shaft speed(revolutions per minute)
- o arg 6 (---) crank case pressure in  $\text{N/m}^2$  for crank shaft, 0.0 for other cases
- o arg 7 (---) mechanical efficiency if turbo shaft, 0.0 for all other cases
- o arg 8 (int) number of cylinders connected to crank shaft or number of junctions connected to turbo shaft
- o arg 9 (int) array of cylinder numbers if crank shaft, array of junction numbers if turbo shaft

### **C.1.11 Initial Volume Conditions:**

- o line 1 initial conditions for volume 1
- o line 2 initial conditions for volume 2
- o .
- o .
- o .
- o line i initial conditions for volume i

- o arg 1 volume pressure in  $\text{N/m}^2$
- o arg 2 volume temperature in K
- o arg 3 volume fuel to air ratio

### C.1.12 Initial Ambient Conditions:

There are three lines for ambient data, one for the ambient conditions at the inlet and two for the ambient conditions at the exhaust. Each line has three arguments.

- o arg 1 ambient pressure in  $\text{N/m}^2$
- o arg 2 ambient temperature in K
- o arg 3 ambient fuel to air ratio

### C.1.13 Integration Step Information:

This data structure is a single line with six arguments that provide information to control the integration algorithm.

- o arg 1 (---) step size in seconds (Not used)
- o arg 2 (---) step size in degrees
- o arg 3 (---) starting crank angle for cylinder 1 (Not used)
- o arg 4 (int) number of correction loops before step length is reduced  
(Not used)
- o arg 5 (---) convergence tolerance of predictor-corrector method for mass, fuel to air ratio and temperature, fraction of change at each correction.
- o arg 6 (---) convergence tolerance (Not used)

## C.1.14 Timing Information:

This section provides angles in degrees at which valves open or close and static fuel injection starts with reference to cylinder 1.

- o line 1 data for timing set 0
- o line 2 data for timing set 1
- o .
- o .
- o .
- o line n data for timing set n-1
  
- o arg 1 reference angle at which exhaust valve closes
- o arg 2 reference angle at which inlet valve closes
- o arg 3 reference angle at which static fuel injection starts
- o arg 4 estimated reference angle at which combustion finishes
- o arg 5 reference angle at which exhaust opens
- o arg 6 reference angle at which inlet opens

## C.2 Compressor and Intercooler Data File:

The intercooler is included as part of the compressor data file. The compressor is simulated by using a digitised map of pressure ratio, mass flow parameter and efficiency.

### C.2.1 Control data:

This file starts with a comment, which is followed by a line that describes different variables required to set up compressor map. This line has following six arguments.

- o arg 1 number of speed curves (N) (max 11)
- o arg 2 number of lines for pressure ratio, mass flow parameter and efficiency (M) (max 11)
- o arg 3 pressure ratio scaling factor
- o arg 4 speed curve scale factor
- o arg 5 efficiency scale factor (this will convert percentage to fraction of unity)
- o arg 6 mass flow scale factor

### C.2.2 Intercooler data:

This data follows the compressor setup line. It must have a comment before it. It consists of the following three arguments.

- o arg 1 intercooler pressure loss ( $N/m^2$ )
- o arg 2 intercooler coolant inlet temperature (K)
- o arg 3 intercooler effectiveness

### C.2.3 Compressor data:

This data section consists of N block of digitised compressor map curves. Each section consists of one line for the non dimensionlised compressor speed ( $w/K^{1/2}$ ), where w is in radians per second, and M lines, each consisting of the following three arguments.

- o arg 1 pressure ratio
- o arg 2 mass flow parameter ( $\text{kg K}^{1/2} \text{ m}^2/\text{Ns}$ )
- o arg 3 isentropic efficiency (0 to 1)

Each block is delimited by an asterisk on its own line including the last line of the file.

## C.3 Turbine Data File:

The turbine is also simulated by using a digitised map of pressure ratio, mass flow parameter and efficiency, and is fed as a separate file to the simulator.

### C.3.1 Turbine data:

This file starts with a comment, which is followed by a line that describes different variables required to set up turbine map. This line has following seven arguments.

- o arg 1 number of speed curves (N) (max 11)
- o arg 2 number of lines for pressure ratio, mass flow parameter and efficiency (M) (max 11)
- o arg 3 number of turbine inlets
- o arg 4 pressure ratio scaling factor
  
- o arg 5 speed curve scale factor
- o arg 6 efficiency scale factor (this will convert percentage to fraction of unity)
- o arg 7 mass flow scale factor

### C.3.2 Turbine data:

This data section consists of N block of digitised turbine map curves. Each section consists of one line for the non dimensionlised turbine speed ( $w/K^{1/2}$ ), where w is in radians per second, and M lines, each consisting of the following three arguments.

- o arg 1 pressure ratio
- o arg 2 mass flow parameter (kg K<sup>1/2</sup> m<sup>2</sup>/Ns)
- o arg 3 isentropic efficiency (0 to 1)

Each block is delimited by an asterisk on its own line including the last line of the file.



# Appendix D

## An Example Engine Data File

```
/*
*-----*
*   engine.dat
*
*   Maximum string length in comments - 80 characters      *
*   Variable preceded by int need to be integer and not floating point number *
*-----*

```

Leyland t111 engine (six cylinder engine with Garrett turbocharger), 338 injection timing

### D.1.1 Control data for number of cycles and printout of results:

```
(int) number of cycles, (int) first cycle for printout, (int) last cycle
for printout, (int) interval between printouts
*/
0000

```

### D.1.2 Number of engine components:

```
(int) no_cylinders, (int) no_manifolds, (int) no_junctions,
(int) no_compressors, (int) no_turbines, (int) no_valve_sets,
(int) no_timing_sets, (int) no_heat_rel_sets,

```

```

(int) no_ht_sets, (int) no_ht_man_sets,
(int) no_shaft_sets, (int) no_guv_sets, (int) no_load_sets,
output files, turbo files in junction order
*/
6 3 15
1 1 2
1 1
1 2
2 0 0
engine.res engine.plt
compressor.dat turbine.dat

```

```
/*
```

### D.1.3 Cylinder data:

```

(int) vol.number,(int) vol.type (1 - cylinder, 2, 3- manifold, 4 - atmosphere,),
(int) task_type (2=cyl,3=im,4=em)
(int) cylinder fault ( 0 - no fault, non_zero - faulty)
bore(m), stroke(m), comp_ratio, ca offset (deg w.r.t. cyl 0),
con_rod_length(m), piston_mass (including reciprocating mass of con rod),
fmep (N/m2, <= 0.0f for calculation by program),
(int) heat_rel_set,(int) ht_set, (int) shaft_set, (int) timing_set,
(int) no_in_junct, (int) no_ex_juncts,
(int) in_junct_nos, (int) ex_junct_nos (one of each only at present)

```

```
*/
```

```

1 1 2 1 0.12708 0.14605 15.75 000.0 0.2667 1.0 -1 0 0 0 0 1 1 0 1
2 1 2 1 0.12708 0.14605 15.75 240.0 0.2667 1.0 -1 0 0 0 0 1 1 2 3
3 1 2 1 0.12708 0.14605 15.75 480.0 0.2667 1.0 -1 0 0 0 0 1 1 4 5
4 1 2 1 0.12708 0.14605 15.75 120.0 0.2667 1.0 -1 0 0 0 0 1 1 6 7

```

```

5 1 2 1 0.12708 0.14605 15.75 600.0 0.2667 1.0 -1 0 0 0 0 1 1 8 9
6 1 2 1 0.12708 0.14605 15.75 360.0 0.2667 1.0 -1 0 0 0 0 1 1 10 11

```

```
/*
```

### D.1.4 Manifold data:

```

(int) vol_no, (int) vol_type (2 - no ht, 3 - with ht),
(int) task_type (2=cyl,3=im,4=em), (int) tc_no (which turb/comp to use)
ht_set (-1 for no ht),
volume (m3), (int) no_in_junct, (int) no_ex_juncts, (int) in_junct_nos, (int)
ex_junct_nos

```

```
*/
```

```

7 2 3 0 -1 0.00488 1 6 12 0 2 4 6 8 10
8 3 4 0 0 0.00138 3 1 1 3 5 13
9 3 4 0 1 0.00120 3 1 7 9 11 14

```

```
/*
```

### D.1.5 Junction data:

```

(int) junc_no,
(int) junction type (2 - oriface, 1 - poppet, 3 - comp. 4 - turb),
effective area (m2), cd, (int) poppet valve_set, shaft set (for crank) and number of
first turbine entry (other 0 including first turbine entry),
(int) entry volume, (int) exit volume

```

```
*/
```

```

0 1 0.0 0.0 0 7 1
1 1 0.0 0.0 1 1 8
2 1 0.0 0.0 0 7 2

```

3 1 0.0 0.0 1 2 8  
4 1 0.0 0.0 0 7 3  
5 1 0.0 0.0 1 3 8  
6 1 0.0 0.0 0 7 4  
7 1 0.0 0.0 1 4 9  
8 1 0.0 0.0 0 7 5  
9 1 0.0 0.0 1 5 9  
10 1 0.0 0.0 0 7 6  
11 1 0.0 0.0 1 6 9  
12 3 0.0 0.0 0 10 7  
13 4 0.0 0.0 0 8 11  
14 4 0.0 0.0 13 9 11

/\*

### D.1.6 Heat release data:

(int) hr\_set\_no, cal\_value (J/kg K), combustion duration (deg),  
fuel\_pipe\_length (m) (same for all the models), wiebe\_c1, wiebe\_c2

\*/

watson 42800000.0 125.0 1.1 0.0 0.0

/\*

### D.1.7 Cylinder heat transfer data:

(string) heat transfer model (on own line),

woschni--(int) number of surface areas

array of areas (0.0)

array of temperatures (K)

hohen----1 surface area

0 area

initial wall temp (K)

woschni 1

0.0

423

\*/

hohen 1

0.0

423.0

/\*

### **D.1.8 Manifold heat transfer data:**

(string) model type (ht\_simple)

surface\_area, heat transfer coeff (exhaust gas),

thermal resistance, thermal capacitance,

estimated wall temp.

\*/

ht\_simple 0.15 175. 0.225 450.0 303

ht\_simple 0.15 175. 0.225 450.0 303

/\*

### **D.1.9 Valve data:** repeat for each valve:

(int) number of points, shift factor, stretch factor, area factor

crank angle array (deg), corresponding valve area array (m<sup>2</sup>)

\*/

32 0.0 1.0 1.0

000.0 005.0 015.0 025.0 045.0

050.0 055.0 065.0 075.0 085.0

090.0 095.0 100.0 105.0 115.0

125.0 135.0 145.0 150.0 155.0

160.0 165.0 175.0 185.0 190.0

200.0 210.0 220.0 230.0 710.0

715.0 720.0

1.175e-4 2.020e-4 5.570e-4 9.630e-4 1.572e-3

1.633e-3 1.674e-3 1.675e-3 1.634e-3 1.602e-3

1.589e-3 1.581e-3 1.580e-3 1.579e-3 1.580e-3

1.588e-3 1.615e-3 1.655e-3 1.673e-3 1.683e-3

1.674e-3 1.641e-3 1.485e-3 1.167e-3 9.630e-4

5.250e-4 2.310e-4 9.500e-5 000000.0 000000.0

3.300e-5 1.175e-4

32 0.0 1.0 1.0

000.0 004.0 014.0

494.0 504.0 514.0 534.0 539.0

544.0 549.0 554.0 559.0 564.0

574.0 584.0 594.0 604.0 614.0

624.0 634.0 644.0 654.0 664.0

669.0 674.0 679.0 684.0 689.0

694.0 704.0 714.0 720.0

1.500e-4 8.200e-5 000000.0

000000.0 8.200e-5 3.060e-4 9.680e-4 1.104e-3  
1.159e-3 1.189e-3 1.201e-3 1.175e-3 1.163e-3  
1.092e-3 1.084e-3 1.089e-3 1.091e-3 1.091e-3  
1.090e-3 1.088e-3 1.083e-3 1.119e-3 1.175e-3  
1.199e-3 1.189e-3 1.160e-3 1.095e-3 9.630e-4  
8.010e-4 4.450e-4 1.980e-4 1.500e-4

/\*

### D.1.10 Shaft details:

(int) shaft no., (int) shaft\_type (1 - crank, 2 - turbocharger),  
(int) dynamic, (0 - constant shaft speed; 1 - dynamic shaft speed),  
load\_inertia, load\_torque,  
shaft\_speed, crankcase pressure, mechanical efficiency (turbocharger)  
(int) no\_of\_cyls or junctions connected to shaft,  
if shaft\_type = 1:- (int) cylinder numbers, else leave blank  
if shaft\_type = 2:- (int) junction (turbine or compressor) numbers, else blank  
\*/

0 1 0 1e21 300.0 1500.0 1e5 0.0 6 0 1 2 3 4 5  
1 2 1 0.0002 0.0001 68829.60 0.0 0.95 3 12 13 14

/\*

### D.1.11 Initial conditions:

In control volume order.  
Pressure (N/m<sup>2</sup>), Temperature (K), FAR  
\*/

117246.78 652.69 2.8482158e-002

197465.72	387.44	6.6022854e-004
583237.44	1108.38	3.5283495e-002
148741.17	334.12	7.9941779e-004
162893.17	793.59	3.5289638e-002
9415444.00	1426.87	1.0372365e-002
152357.84	300.94	4.0311828e-005
107916.05	653.61	3.5221875e-002
155900.39	785.62	3.5240222e-002

/\*

### D.1.12 Ambient conditions:

Pressure (N/m<sup>2</sup>), temperature (k) and fuel/air ratio at inlet and exhaust

\*/

1.0022e5	290.842	0.0000
1.0022e5	290.842	0.0000
1.0022e5	290.842	0.0000

/\*

### D.1.13 Integration step information:

stepsize in seconds or degrees (set time to zero for degree option),

starting crank angle for cylinder 0

(int) loopcount, tolerance (for LoStab, HiStab)

\*/

0.0 2.0 0.0 4 0.005 0.9

/\*



## D.1.14 Crankangles at transition points:

In degrees w.r.t. cylinder 0

evc, ivc, static timing(same for all the sets),

estimated\_pow\_angle, evo, ivo

\*/

14.0 230.0 338.0 463.0 494.0 710.0

/\*-----\*/

## D.2 An Example Compressor Data File

```
/*
*-----*
* Compressor data file *
* *
* UNITS - All values are multiplied by the appropriate scaling factor. *
* When this is has been done, the units should be : *
* *
* Speed : Actual turbine speed in rads/s. *
* Pressure Ratio : This is a simple ratio hence is dimensionless. *
* Efficiency : Dimensionless ( 0 < eff < 1 ). *
* Mass Flow Rate : Normalised mass flow rate in kg/s. The program scales *
* these values by P(ambient)/sqrt(T ambient). *
* *
* The values for ambient temperature and pressure are in SI units, ie *
* temperature ( K ) *
* pressure ( N/m^2 ) *
*-----*

* D.2.1 control data: No.speed curves, No.points, Scaling factors:
pr,np,ef,mp *
*/
11 11 1.0 0.1047 1.2e-3 1.37091e-6
```

/\*

### D.2.2 Intercooler data:

press loss( $n/m^2$ ),inlet temp(k),effectiveness

\*/

2622 289.6672 0.872592

/\*

### D.2.3 Tabulated map:

Pressure ratio, Mass flow parameter, efficiency

\*/

46300

1.300 00 000

1.250 20 725

1.220 25 650

1.150 30 520

1.080 35 425

1.000 40 350

1.000 45 275

1.000 50 225

1.000 55 200

1.000 60 175

1.000 65 150

\*

56300

1.575 00 000

1.550 20 725

1.545 25 750

1.513 30 740

1.450 35 690

1.375 40 575

1.250 45 475

1.100 50 375

1.000 55 300

1.000 60 250

1.000 65 200

\*

67900

1.900 00 000

1.875 20 670

1.860 25 710

1.850 30 750

1.840 35 760

1.790 40 740

1.700 45 700

1.540 50 585

1.225 55 450

1.000 60 375

1.000 65 325

\*

77900

2.275 00 000

2.255 20 550

2.250 25 630

2.245 30 700

2.235 35 745

2.225 40 765

2.175 45 755

2.075 50 735

1.900 55 660

1.450 60 450

1.100 65 325

\*

87100

2.675 00 000

2.650 20 450

2.645 25 525

2.640 30 600

2.635 35 675

2.625 40 725

2.600 45 750

2.588 50 755

2.475 55 735

2.275 60 675

1.250 65 000

\*

95100

3.140 00 000

3.063 20 520

3.056 25 580

3.050 30 630

3.044 35 670

3.038 40 700

3.031 45 725

3.025 50 740

2.975 55 745

2.825 60 720

2.375 65 600

\*

102700

3.500 00 000

3.488 20 490

3.481 25 565

3.475 30 615

3.469 35 650

3.463 40 675

3.456 45 700

3.450 50 710

3.438 55 720

3.375 60 720

2.950 65 650

\*

## D.3 An Example Turbine Data File

```
/*
*-----*
* Turbine data file *
* *
* UNITS - All values are multiplied by the appropriate scaling factor. *
* When this has been done, the units should be : *
* *
* Speed: Normalised turbine speed in rads/s. These speeds are *
* multiplied by sqrt(turb inlet temp) within the model. *
* Pressure Ratio: This is a simple ratio hence is dimensionless. *
* Efficiency: Again dimensionless ( 0 < eff < 1 ) *
* Mass Flow: Normalised mass flow in kg/s. The program multiplies *
* these values by P(e_man)/sqrt(turb inlet temp). *
* *
* The de-normalising parameters have standard SI units, ie *
* temperature ( K ) *
* pressure ( N/m^2 ) *
* *
*-----*
```

### D.3.1 Control data:

No. speed curves, No. points, No. inlets,

Scaling Factors: pr, np, ef, mp

\*/

1 11 2 1.0 6.0171e-3 1.0e-3 1.26984e-6

/\*

### D.3.2 Tabulated turbine map:

Pressure ratio, Mass flow parameter, Efficiency

\*/

000

1.0 00.0 500

1.1 22.0 640

1.2 30.0 675

1.4 37.6 700

1.6 41.8 705

1.8 44.2 710

2.0 45.5 710

2.2 46.0 700

3.0 44.0 650

4.0 42.0 600

5.0 42.0 600

\*



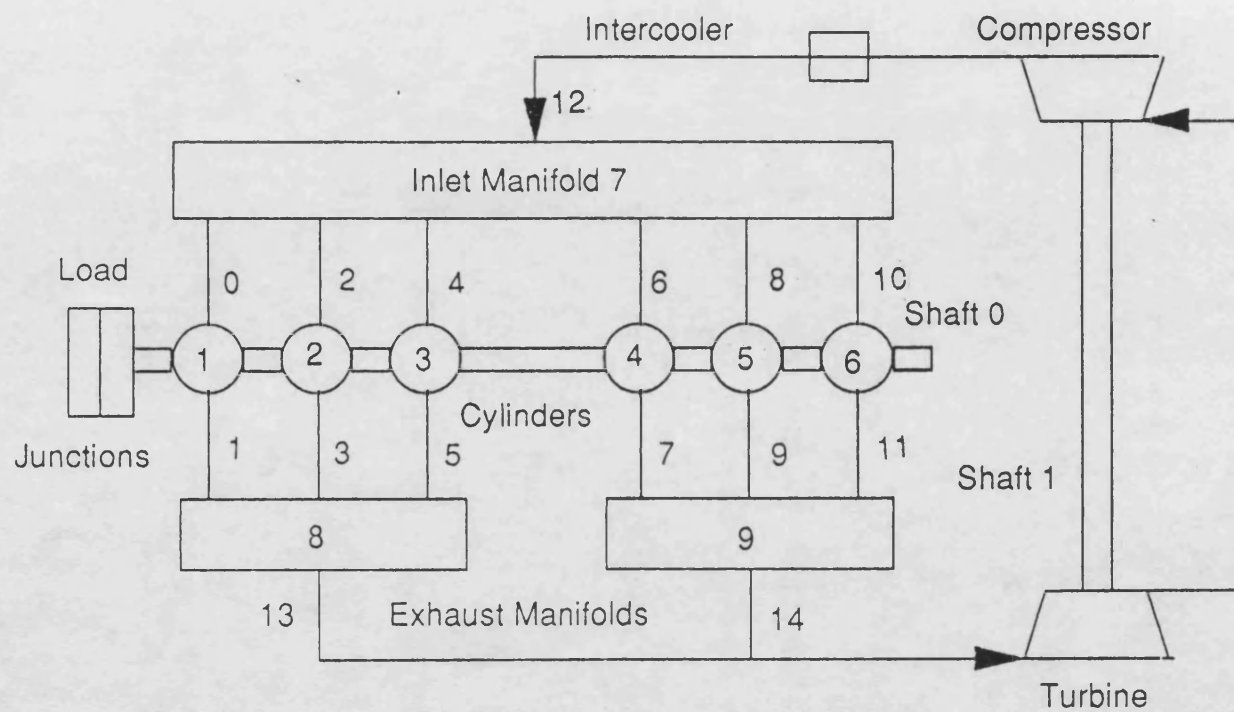


Figure D.1 Schematic drawing of a six cylinder turbocharged Diesel engine