NASA Contractor Report 189621

Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems (DAHPHRS) Phase 1

C. Scheper
R. Baker
G. Frank
Research Triangle Institute
Research Triangle Park, NC 27709

S. Yalamanchili
Honeywell, Inc.
Systems and Research Center
Minneapolis, MN 55418

G. Gray
Virginia Polytechnic Institute and State University
Bradley Department of Electrical Engineering
Blacksburg, VA 24061

Contract NAS1-17964
May 1992

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

Contents

List of Figures

ACKNOWLEDGEMENTS

This report was prepared for the Rome Air Development Center (RADC) and the National Aeronautics and Space Administration's Langley Research Center (NASA-LaRC) under Contract NAS1-17964. The work was performed in the Center for Digital Systems Research (CDSR) of Research Triangle Institute (RTI) by C. Scheper (Project Leader), R. Baker, G. Frank, and H. Waters; in the Systems and Research Center of Honeywell, Inc. by S. Yalamanchili and T. Carpenter; and in the Bradley Department of Electrical Engineering of Virginia Polytechnic Institute (VPI) by G. Gray, L. DeBrunner, and T. White.

We wish to express our appreciation for the guidance and support provided by the project technical monitors, Capt. G. Paeper of RADC; and W. Bryant, C. Elks, and S. Johnson of NASA-LaRC.

Technical support in preparing the DAHPHRS documentation and this report was provided by G. Crim, J. Muller, P. Noell, J. Sapp, K. Simmons, B. Shay, and B. Taylor.

1. Introduction

The Strategic Defense Initiative (SDI) concepts require the development of systems for complex space applications subject to demanding real time computational resource requirements and very high reliability requirements. Complex space applications are characterized by large amounts of numerical processing, large data bases, and iterative approximations to optimal solutions. The algorithms used in the signal and image processing that are necessary for target detection, classification, tracking, and trajectory estimation are computationally intensive and must meet real-time deadlines to match the incoming data rates of wideband sensors. However, these algorithms can usually be decomposed into highly regular computational structures. On the other hand, mission planning functions employ programming techniques in which the computational requirements vary with dynamic changes in the incoming data. The missions that these applications address require extremely long system operating life. They are also divided into phases composed of long periods of moderate activity followed by very short periods of high activity and characterized by vastly different reliability and performance requirements. In addition, the system operates under demanding system weight and power requirements in an environment subject to radiation and thermal and mechanical stress.

These system characteristics lead to complex non-uniform architectures comprising a large number of processors with mechanisms to extend operational life and to support changes in mission phase or the attrition of system resources. Specifically, the applications require architectures that combine elements of massive parallel computing techniques and of fault-tolerant system design. Parallel computing is a rapidly emerging field of research and many of the complex practical problems related to realizing effective architectures have not been solved. This, coupled with the extra complexity brought on by the fault tolerance requirements, presents the system engineer evaluating the performance and reliability for such systems with the extremely difficult problem of conducting performance and reliability trade-offs over large design spaces and verifying performance and reliability over a wide range of operating conditions.

The methods and tools to fully and effectively deal with these problems do not exist. This is due in part to the sheer size of the systems involved. However, the very nature of these systems dictate evaluation criteria that differ in many respects from those used to evaluate more traditional computing systems. Consequently, tools which can meet the challenge presented by the high level of complexity and the expanded evaluation criteria for such systems are required. Further, the combining of parallel computing with fault tolerance requirements dictates the need for integrating

the performance and reliability evaluation tools in order to facilitate trade-offs between performance and reliability. Even if tools which meet the demands of present applications did exist, the advances in this rapidly emerging area could soon render them inadequate. Thus, the goal of this program is to develop an integrated set of performance and reliability tools capable of managing the complexity of such designs and robust enough to adapt to the inevitable technological advances.

In order to determine what tools are needed and how those tools should interact, this first phase of the toolset development was focused on relating tools to a methodology framework through the development and analysis of a paradigm of the design process for an SDI-like system. The paradigm was used to determine what system models are needed, how the models interact, and what experiments and analyses are needed in an effective methodology for system design for performance and reliability. It was also used to illustrate how tools support such a methodology and what the tool features and capabilities should be. Based on the identified interactions between the reliability and performance analyses, the paradigm will be expanded in a subsequent phase to allow the examination of fault tolerance mechanisms and the study of performance/reliability trade-offs. Also, existing tools will be identified and additional tools and interfaces specified and built in a subsequent phase.

This report documents the activity and findings during the first phase of this contract. Section 2 discusses the issues related to tools and methodology. Section 3 discusses the development of the paradigm and the algorithms and architectures selected for inclusion therein. Sections 4 and 5 discuss the performance and reliability analyses performed for the paradigm.

2. Tools/Methodology

The dual requirements of high reliability and high performance for systems that will operate nearly autonomously in mission- and life-critical applications dictate that those systems be validated to a high level of confidence. Accordingly, it is expected that a rigid development process be utilized to assure that design errors are eliminated before the system is delivered and to assure that the system will meet reliability and performance objectives. A design for reliability methodology framework has been set forth in the working document of the SDIO BM/C^3 Processor and Algorithm Working Group [13]. This methodology specifies eight steps for system design:

1. Identify classes of expected faults over the lifetime of the system.
2. Specify goals for the dependability of system performance.
3. Partition the system into subsystems for implementation, taking into account both performance and fault tolerance.
4. Select error detection and fault diagnosis algorithms for every subsystem.
5. Devise state recovery and fault removal techniques for every subsystem.
6. Integrate subsystem fault tolerance on system scale.
7. Evaluate the effectiveness of fault tolerance and its relationship with performance
8. Refine the design by iteration on steps three through seven.

The methodology also specifies five phases of design and establishes milestones for each phase and deliverables for the design reviews that occur at the end of each phase. The specified design reviews are as follows:

1. System requirements review to specify a computational model, requirements for performance and fault tolerance, applicable architectural approaches, and a development plan.
2. System design review to evaluate architectural trade-offs and to select an architectural approach and fault tolerance strategy.

3. Preliminary design review to specify preliminary hardware and software design and to provide performance and fault tolerance evaluation.

4. Critical design review to provide a completed hardware and software design, refined analysis of performance and fault tolerance attributes, and a plan for a feasibility demonstration.

5. Demonstration, evaluation, and test review to include a demonstration of brassboard components and operational software, and an experimental evaluation of performance and fault tolerance features.

The working document also specifies the use of tools in each phase and describes the characteristics of those tools, as summarized below.

During the system requirements phase, tools are needed to evaluate very high-level designs without detailed hardware and architectural information. These tools must interface with the tools that analyze more detailed designs in later phases. The outputs of these tools should be usable as inputs to more detailed tools and/or compared with more detailed evaluation results to verify that the high-level requirements are met by the actual design. It should be possible to share data files by tools at all levels in the design process. The need for integration is paramount. Current practice in this area results in *ad hoc* methods being reinvented by each contractor and results in tools that do not readily interface with tools at other levels of design. A generally accepted standard is sorely needed.

At the system design review, architecture alternatives are evaluated. Tools are needed that will allow meaningful comparisons of the performance and fault tolerance attributes of each alternative system. The tools must model high-level architecture features and must incorporate a high-level fault model. In the expert report, it is also recommended that error propagation effects and the effects of corruption of system state due to faults be modeled at this level. An accurate testability analysis is also required. Some tools are available that do part of this job (PMS and ISP simulators, for example). Unfortunately, these simulators do not consider the effects of faults. An integrated tool that evaluates both performance and fault tolerance attributes using a common data base and model is needed.

At the preliminary design review, more details of the selected design are available. It should be possible to refine the models created for the system design review to reflect the newly available detail. More accurate estimations of performance and fault tolerance parameters should then be possible. The tools should now be able to provide accurate estimates of coverage of the error detection mechanism and to

evaluate the quality of the error containment and error recovery procedures. The tools must have a clean interface to the more detailed tools that will come later and to the more general tools used earlier. Accurate reliability modeling tools are also a necessity at this stage.

At the critical design review, details of both hardware and software designs are completed. The evaluation tools should allow further refinement of the models to reflect the additional detail. Detailed hardware and software simulations should now be performed. Since existing tools cannot handle the complexities of modern designs at this stage, some form of hierarchical simulation will most likely be needed. A small part of the system will be modeled in great detail and interfaced with higher-level simulations of the remainder of the system. This will require a clean interface between the higher-level and the lower-level simulation tools. Reliability models will also need to be refined to reflect the more detailed information. Simulation results such as coverage factors, recovery times, etc., need to be easily transferred from the simulation program to the reliability analysis program. Again, integration is needed.

At the test, implementation, and validation review, the results of experiments performed on the prototype should be presented. The reliability modeling tool should have predicted a behavior of the system that can be verified by actual injection experiments. The modeling and simulation tools should be interfaced with the testbed so that the required inputs to the testbed can be generated automatically and the outputs compared with those predicted by the simulations.

Given this methodology framework and its reliance on tools to support design and produce deliverables for the design reviews, a tools/methodology task was included in the DAHPHRS program. The objectives of this task were to show how the use of performance and reliability tools within such a methodology framework can support the design process and to identify the interactions between the performance and reliability analyses. The tools/methodology task benefited from actual evaluations of architectures and algorithms using currently available performance and reliability tools. These evaluations produced a number of observations concerning the capabilities of tools for the performance and reliability analysis of multiprocessor architectures. In particular, by addressing the question of what information (level/resolution) is required to ensure that the fidelity of the analysis is adequate and that desirable system configurations are not precluded from consideration, it has become evident that distinct computing paradigms require distinct simulation paradigms. For example, tightly coupled architectures require different modeling techniques from those that are suitable for loosely coupled multiprocessor architectures. It also became clear in the course of this effort that a number of tools are necessary for managing

the complexity of the simulation of the designs. The need to establish the accuracy of the simulation results also arose along with a recommended approach for doing so. Validation of results is especially important in critical applications.

2.1. Performance Models and Tools

The role of performance models and tools in support of the methodology was investigated through performance analyses of the paradigm algorithms and architectures. These analyses illustrate the roles of measurement, functional simulation, and stochastic methods in support of modeling. Two fundamental issues are the validation of models and the selection of an appropriate level (fidelity) of modeling. The appropriate level depends both on the information required to build the model versus the information available and on the amount of data that can be derived from the model versus its reliability and usefulness. The analyses also identified additional tool needs.

System performance should be modeled by multiple models of increasing detail and complexity consistent with the amount of information available at particular design stages. Also, as the performance analyses conducted by Honeywell for the Encore Multimax and the JPL Hypercube demonstrate, the resolution required in a performance model also depends on the architecture. Consequently, the methodology requires a decision as to what level of model is needed to support the required granularity of simulation for a given architecture at a given design phase. At the highest level, the system is modeled by an analytical model using primitive information such as processing and communication workloads for the algorithms and processors and IO and memory bandwidths for the architecture. As the hardware and software of the system are defined in more detail, performance modeling by simulation and engineering models can be initiated. The performance modeling process, as illustrated in Figure 2.1, starts with a description of the architectures and algorithms that make up the system. From this description, the functional, performance, and engineering models can be developed.

The performance model is the essential system model, consisting of a data/control flow model of the algorithmic processes which has been mapped, or constrained, to a structural model of the architecture. A functional model can be built which describes the behavior of either the components of the architecture or the algorithms that will be executed by the system. Finally, the engineering model consists of system prototypes for the algorithms and/or architectures. All three models produce performance predictions, the functional and performance models through simulation and the engi-

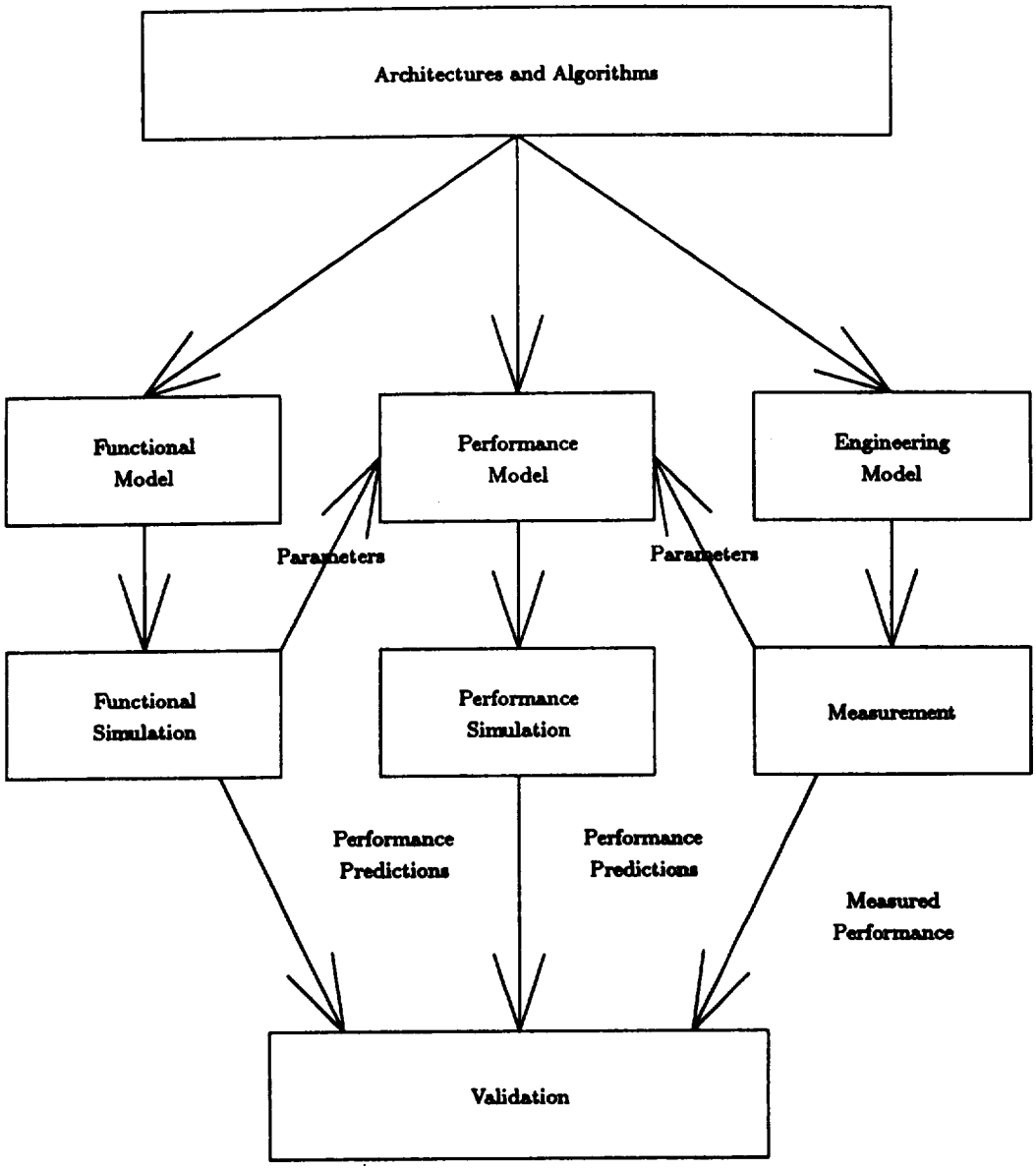


Figure 2.1. Performance Modeling Process

neering model through measurement. The three models interact through parameter values computed by the functional and engineering models for use by the performance model. Each of the models can be built to whatever degree of detail is reasonable for the stage of design under consideration, and the cross-validation that occurs among the models as they produce consistent performance assessments allows the models to be used and built upon with confidence at subsequent levels. Thus, performance models that have been built from and validated by measurements from engineering models can be used in later analyses or subsequent system designs without the need for implementing a full engineering model.

The performance and functional models of the paradigm architectures and algorithms were created using the Architecture Design and Assessment System (ADAS). ADAS is a tool developed by RTI for the hierarchical description and assessment of system designs. In ADAS, the system performance model is created from a structural model of the architecture and a data/control flow model of the processes. The structural model, or ADAS hardware graph, is a directed graph comprising nodes to describe the architectural components and arcs to describe the connectivity among components. Attached to each node and arc of the graph are attributes which become constraints in the performance model based on the construction of a mapping from the structural model to the algorithm model. The algorithm model, or ADAS software graph, is a directed graph describing the data/control flows (arcs) of the software processes (nodes) in the system. Its attributes define the required computing and communication resources and control the assignment of software components to hardware components. The constrained software graph created by the construction of a mapping and containing processing times defined by the attributes of both graphs becomes the ADAS performance model. This performance model can be simulated by the ADAS tool GIPSIM to predict the performance of the software processes on the architecture for that mapping and thereby predict whether the system will attain its throughput requirements. In ADAS, a CSIM functional simulation model can also be constructed to simulate the function of software processes (such as the application algorithms and, as was done by VPI, reconfiguration algorithms) or the detailed operations and interactions of the system components (as was done by Honeywell for the Multimax model).

Recent modifications to ADAS will make it easier to explore the large experiment space inherent in the design of parallel and fault-tolerant systems. An Attribute Definition Language (ADL) and ADL Evaluator (ADLEVAL) have been developed for the creation of parameterized performance models. ADL expressions describe the performance model attributes in terms of system parameters. These expressions are associated with ADAS nodes, arcs, and graphs, and are translated into ADAS

attributes by ADLEVAL. Also, the ADL expressions can be inherited and synthesized throughout the hierarchy of graphs that describe the system so, for example, processor instruction processing speeds, memory and interconnect bandwidths, and mission parameters can be included in an ADL file at the root graph and be accessed at all subsequent graph levels. A prototype ADL and ADLEVAL were used in this study and, as a result, modifications to the ADL specifications and their implementation were identified that provide more flexible parameterization and facilitate modeling more complex systems.

The Graph Transformation System (GTS) is a rule-driven tool that aids the user in customizing software to fit system constraints, including the capabilities of available hardware resources and the processing requirements of other algorithms that are part of the system model. The transformation rule base is a set of ADAS graphs, each of which describes the patterns and transforms of a coexistent set of rules designed to transform an algorithm to improve its fit with the system constraints, without changing its function. Transformations can be used to increase or decrease parallelism, to insert fault-tolerant features into an algorithm, to represent the cost of communications delays, or to eliminate unnecessarily redundant operations from an algorithm's description.

In addition to the ADAS performance and functional models, an engineering model of the WAUCTION_ASSIGNMENT algorithm was constructed from available engineering code. This engineering model was used to measure expected system performance as well as to provide parameters for the performance model. It is especially important that measurements and statistical data be produced to derive the parameters of simulation models of mission planning algorithms, since the control flow and iterative structure of such algorithms is highly data and mission parameter dependent. It is also necessary for these algorithms that stochastic model attributes can be specified and utilized in performance simulation.

The complexity and size of the systems under consideration are certainly at the stress limits for the existing tools for both model creation and simulation. A considerable amount of effort was required to create the algorithm models. This effort highlights the need for automatic generation of models from system and requirements descriptions, such as those developed by CASE tools. It also highlights the need for a library of models of functions common to a wide class of algorithms, such as sorting and linear programming constructs. Since parameterized function models can be built to a high level of detail independent of any architecture, they can contain a higher degree of information than those that were built for this study. Thus, when a library of function models exists, the individual models can be used to build an application-specific

model of greater resolution than would be feasible by constructing the model from scratch.

2.2. Reliability Models and Tools

The role of reliability models and tools in support of the methodology was investigated through reliability analyses of the paradigm architectures. Since the major emphasis on fault tolerance will occur in Phase II, a selection of preliminary reliability analyses were undertaken in this phase. These preliminary analyses highlighted the reductions of large models, particularly the creation of approximate models to bound the exact model of a complex system; the use of tools to document and provide verification of the models; and additional tool needs.

The high system reliability requirements that exist for an application critical to a long space mission can only be met by a system designed to be fault-tolerant. Such a system has to be carefully evaluated to determine whether or not the reliability requirements are met. However, the existence of fault tolerance mechanisms makes that evaluation more difficult by increasing the number and complexity of significant factors affecting system reliability. Thus, the ideal reliability modeling tools will have to handle very large, complex systems; analyze complex processor intercommunication networks; handle permanent, transient, and intermittent faults; accommodate time-variable failure rates; allow system starting states other than zero failures with unity probability; model complex fault recovery processes; allow for cool spares with reduced failure rates until activated; handle time sequence dependencies between certain faults; and allow for multiple near-coincident faults.

As with system performance, system reliability should be assessed by multiple models of increasing detail and complexity within an overall framework. The reliability modeling process, as illustrated in Figure 2.2, starts with a specification of the fault environment within which the system will operate and of the goals for the dependable performance of the system. System models are then created from a description of the architectures and algorithms that comprise the system. The reliability model describes the behavior of the system in response to the occurrence of faults and is used to predict the probability that the system will be operating correctly at a given time. The fault and recovery models describe the mechanisms by which the system detects faults and isolates and recovers from them. These models are used to predict the effectiveness of these mechanisms and to derive measures of parameters used in the reliability model. Each of the system models can be built to whatever degree of detail

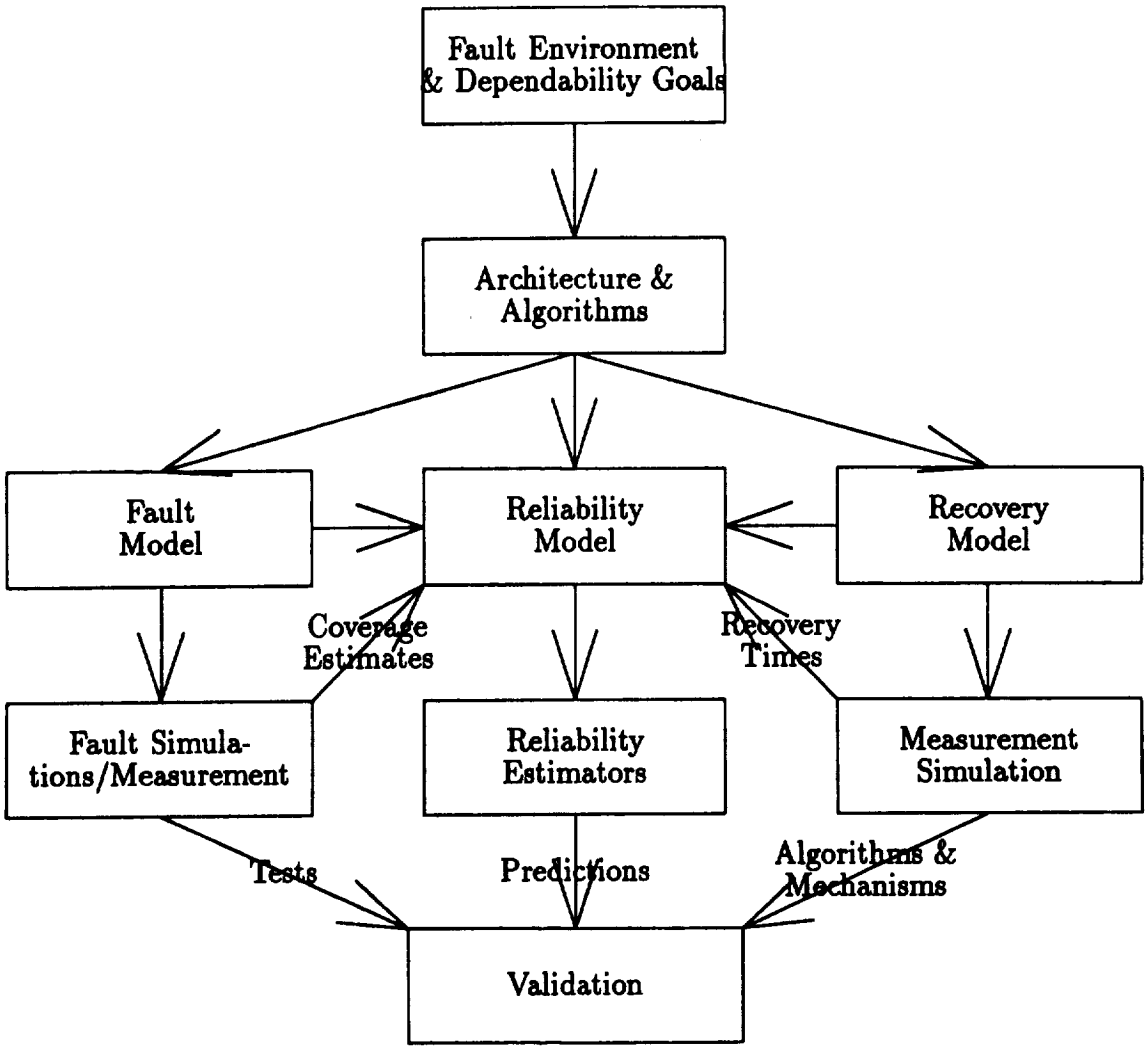


Figure 2.2. Reliability Modeling Process

is reasonable for the stage of design under consideration. However, as the fault and recovery models become more detailed, the parameters of the reliability model can be specified more accurately. When engineering prototypes of the system have been developed, measurements of the fault detection, isolation, and recovery parameters can be made and used to validate the models of the mechanisms for future use.

Two common techniques for reliability analysis are fault tree analysis and Markov modeling. Although the creation of fault tree models is straightforward and tools to solve them exist, such as the NASA-LaRC Fault Tree Compiler [11], they can only be used to describe fault-occurrence behavior. However, for a system to be able to attain the ultrareliability required for mission-and life-critical applications, it must have some fault handling capability and would probably incorporate dynamic reconfiguration. In fact, the fault handling mechanisms can become the most important factor in system reliability for short missions or for particular mission phases. Therefore, tools such as ARIES [12] have been developed based on finite-state, continuous-parameter Markov processes. Since in a Markov model system behavior is described by transitions among operational and failure states, reconfiguration is easily modeled.

The fault-handling recovery mechanisms that are usually represented by a transition rate parameter in Markov models can be explicitly modeled with the semi-Markov extensions of tools such as CARE III [4]. These extensions allow the inclusion of multiple recovery transitions even though they may occur at rates much faster than the fault occurrence transitions.

The solution of Markov and semi-Markov models is computationally difficult and leads to tools being built around fixed, parameterized fault-handling models with simplifying assumptions that permit a stable and efficient solution [1]. Since a rigid model precludes a tool's use on systems not satisfying all of its assumptions and requires that the user develop an in-depth understanding of the model and its assumptions, attempts to generalize or include multiple models have resulted in tools such as HARP [3].

In addition to handling built-in models, most Markov tools assume fixed, usually exponential, distributions for the transitions. However, the Semi-Markov Unreliability Range Estimator (SURE) program from NASA-LaRC [5], can solve models with slow, exponential fault arrival transitions and arbitrary recovery transitions, as long as the fast transitions are several orders of magnitude faster than the slow ones. SURE is fast and accurate for these models because it computes upper and lower bounds on system death state probabilities based on algebraic properties of the means and variances of recovery times, rather than directly solving the differential/integral equations of the

model.

Markov (including semi-Markov) modeling provides a flexible way of describing fault-tolerant systems, and tools exist to compute solutions of the models. However, since the Markov state descriptions increase rapidly with system size and complexity and the solutions are computationally difficult, tools that are based on Markov models have inherent limitations. In general, since highly-reliable, high performance systems result in large, sequence-dependent models, no current reliability analysis tool can address all of the complex interactions of such systems. Even if the underlying model is flexible enough to describe the system, the complete model would be too large to be solved. Furthermore, no tools exist to assess the reliability of the complex multiprocessor networks found in these systems.

Independent of any tool's ability to compute a solution for a particular model, the creation and validation of a correct reliability model is difficult. Thus it is important to have a tool such as the Abstract Semi-Markov Specification Interface to the SURE Tool (ASSIST) [8] that can create a parameterized model from a program language description of the system's fault occurrence and handling behavior. Such a tool not only facilitates the creation and subsequent modification of large models, but also provides documentation for the model and a means of communicating a system's fault tolerance description among different staff and phases of the development process.

Any methodology for reliability analysis has to include facilities for tracking the system requirements through some conceptual reliability model to actual and analyzable approximate models. This suggests exploring methods for the systematic creation of a complete reliability model from the system description to which reduction techniques can be applied to produce analyzable models. A technique for generating a model based on the system description could start with the definition of system failure in terms of software module failure. Module failure would in turn be defined in terms of processor failure, based on the software to hardware mapping. Ultimately, processor failure is defined in terms of hardware component failure, such as the CPU, memory, and interconnections.

As an example of the construction of such a model, consider a system where system failure is defined as the failure of 4 processes. Each process consists of 3 redundant modules and is considered to have failed if any 2 of its 3 modules fail. A module fails if the Processing Element (PE) that it is mapped to fails. A PE fails if either its CPU or its Network Element (NE) connection fails. If an NE fails, then all PEs connected to it fail. An NE fails if either its CPU or any of its cluster connections fail. Figure 2.3 shows the mapping of such a system to the FTPP cluster. In this

figure, the processors that comprise each of the four TMR's are labeled as P1, P2, P3, or P4. Each of the four processes are mapped to a distinct TMR. For example, process module M1 and its redundant modules M4 and M7 are mapped to TMR P1. The ASSIST code to describe the system and generate the Markov description of the system is listed in Figure 2.4.

Since the systematic creation of reliability models of complex systems will result in large models, reduction techniques that can be applied to produce analyzable models are needed. One technique is to bound the reliability of the complete system by that of an approximate system. An optimistic bounding model can be built by relaxing constraints on reconfiguration and eliminating some of the degrading transitions. A pessimistic, or conservative, model can be built by limiting reconfiguration possibilities and by creating some degrading transitions. For example, a mapping of the complete model of the above example to a reduced, conservative model can be constructed by aggregating all states where the number of failed modules per process is the same and combining redundant transitions. A Markov description of the complete model contains 260 states, 3312 transitions and 2434 death states (aggregated into 4 death states); the reduced model, 17 states, 128 transitions, and 98 death states (aggregated into 1 death state). While the complete model of this example is not necessarily too large to solve, it illustrates how model size can be significantly reduced.

2.3. Integrated Performance and Reliability Tools

The performance and reliability of a system that must be both fault tolerant and capable of high throughput cannot be accurately gauged independently. A system's fault tolerance mechanisms must be included in the system performance analysis as well as in the reliability analysis since they not only require significant processing resources, but have to be executed within strict timing constraints. Furthermore, the definition of what constitutes an operational versus a failed system state for reliability has to be derived from an assessment of the ability of the system in that state to achieve the required performance levels. Another level of interaction is required when embedding applications into a parallel system since the fault tolerance mechanisms affect the partitioning of tasks. For example, the task granularity that gives the best computational performance may not be optimum for systems where rollback is used for error recovery.

The goal of integrated performance and reliability analysis tools is to provide consis-

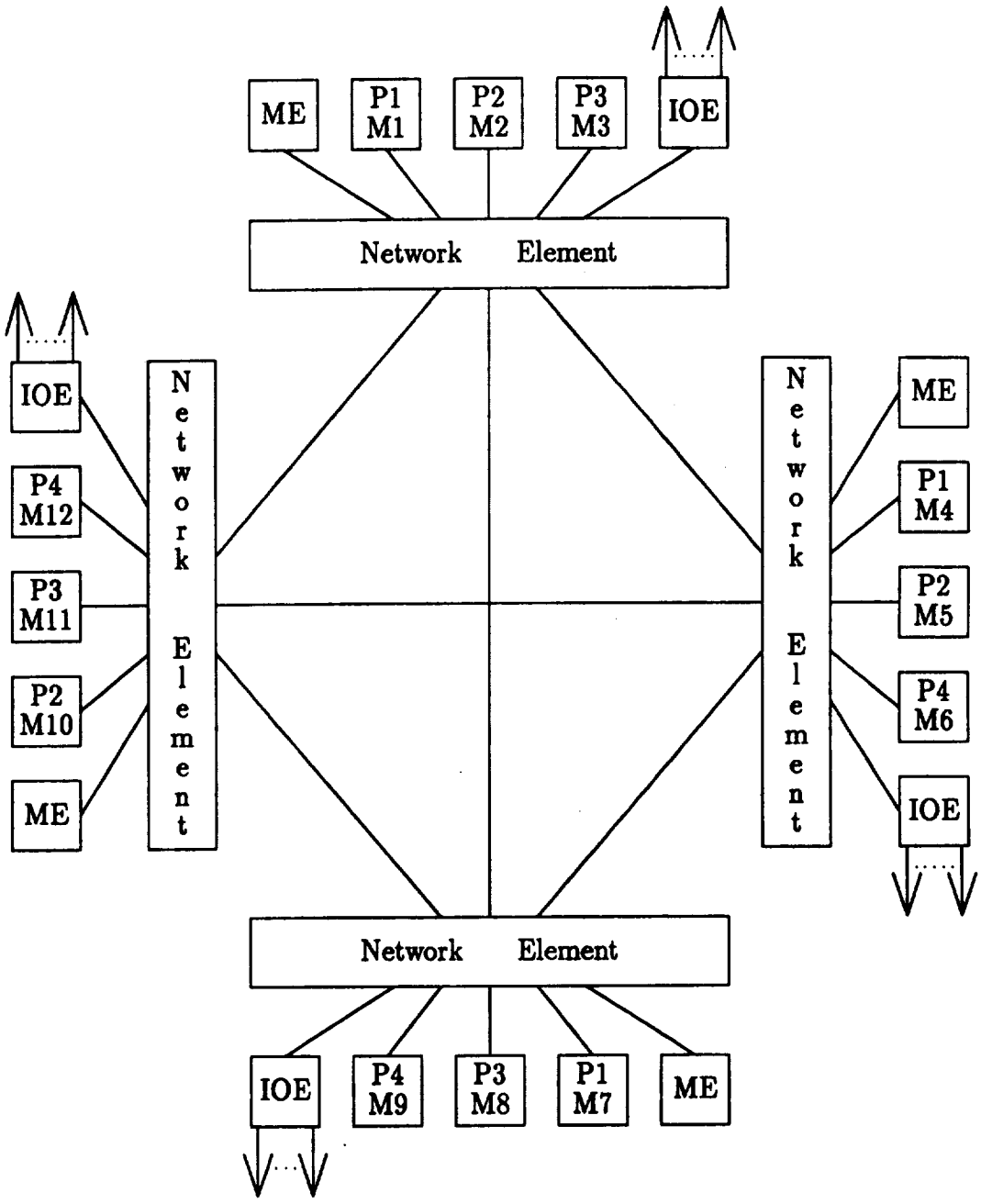


Figure 2.3. Process to FTPP Cluster Mapping

```
n_nes = 4;     (* Number of Network Elements (NEs) *)
n_processes = 4;(* total number of processes in the system *)
redundancy = 3;        (* number of redundant modules per process *)
n_modules = n_processes * redundancy;
l_con = 1e-7;  (* failure rate of interconnections *)
l_vote = 1e-6; (* failure rate of voters *)
l_cpu = 1e-5; (* failure rate of processors *)
l_pe = l_cpu + l_con;
l_ne = l_vote + (n_nes-1)*l_con;
space = (m: array[1..n_modules] of 0..1);
start = (n_modules of 1);

               (* transitions due to processor failure *)
               (* each processor services exactly one module *)

for j=1,n_modules
   if m[j] > 0 tranto m[j] = 0 by l_pe;
endfor;

          (* transitions due to fault containment region failures *)

if m[1] + m[2] + m[3]  > 0
        tranto m[1] = 0, m[2] = 0, m[3] = 0 by l_ne;
if m[4] + m[5] + m[6]  > 0
        tranto m[4] = 0, m[5] = 0, m[6] = 0 by l_ne;
if m[7] + m[8] + m[9]  > 0
        tranto m[7] = 0, m[8] = 0, m[9] = 0 by l_ne;
if m[10] + m[11] + m[12]  > 0
        tranto m[10] = 0, m[11] = 0, m[12] = 0 by l_ne;

               (* death states are defined *)
        (* by the mapping of algorithm modules to processors *)
        (* and by the grouping of modules into processes *)

deathif m[ 1] + m[ 4] + m[ 7] < 2     (* process 1 failure *)
deathif m[10] + m[ 2] + m[ 5] < 2     (* process 2 failure *)
deathif m[ 8] + m[11] + m[ 3] < 2     (* process 3 failure *)
deathif m[ 6] + m[ 9] + m[12] < 2     (* process 4 failure *)
```

Figure 2.4. ASSIST Code

tent performance and reliability models and to facilitate the transfer of data from one model to the other so that through an iterative design process a reasonable trade-off between performance and reliability can be attained. The flow of data among the models is illustrated in Figure 2.5. This data includes the architectural and performance measures that affect reliability model parameters (such as the number and failure rates of components and the system configuration), the network topology, and models of the fault tolerance mechanisms that can be used to assess their impact on performance.

The high-level interactions between the performance and reliability models that need to be studied are those that determine both the minimum number of components, active and spare, and the system configuration required to achieve both the performance and reliability requirements for the system. At a more detailed level, the fault detection, isolation, and reconfiguration (FDIR) parameters such as time and resources required to attain adequate reliability can be measured, and the software-implemented fault tolerance mechanisms such as system reconfiguration algorithms can be evaluated. For example, the reliability analysis may include a coverage model parameter based on the time for execution of an FDIR algorithm. If a performance model of the algorithm exists, its execution time could be obtained from a performance analysis of that model mapped to the target architecture. On the other hand, if a system performance requirement is specified on the basis of an application parameter, such as the number of targets a system should be able to prosecute at a given time during a mission, the reliability analysis could be used to determine the expected number of processors available at that time and a performance analysis could then determine if the required target capacity could be met. Finally, performance/reliability trade-offs can be performed to assess the efficacy of fault-tolerant mechanisms and their effect on system performance, such as a comparison of triplex redundancy versus duplex redundancy with checkpoint/restart with respect to system reliability and performance cost.

Techniques for closer interaction between reliability analysis tools and performance analysis tools will depend upon further work on reliability analysis. In the second phase of this effort, when the fault tolerance mechanisms and algorithms for the paradigm systems will be modeled, the appropriate performance/reliability trade-offs will be conducted. However, in this first phase some examples of preliminary types of analyses were conducted, such as the high-level assessments of the communication costs associated with the FTPP's strategy to achieve Byzantine Resilience (Section 4.3) and a group of analyses to evaluate the performance of alternative reconfiguration procedures (Section 4.7).

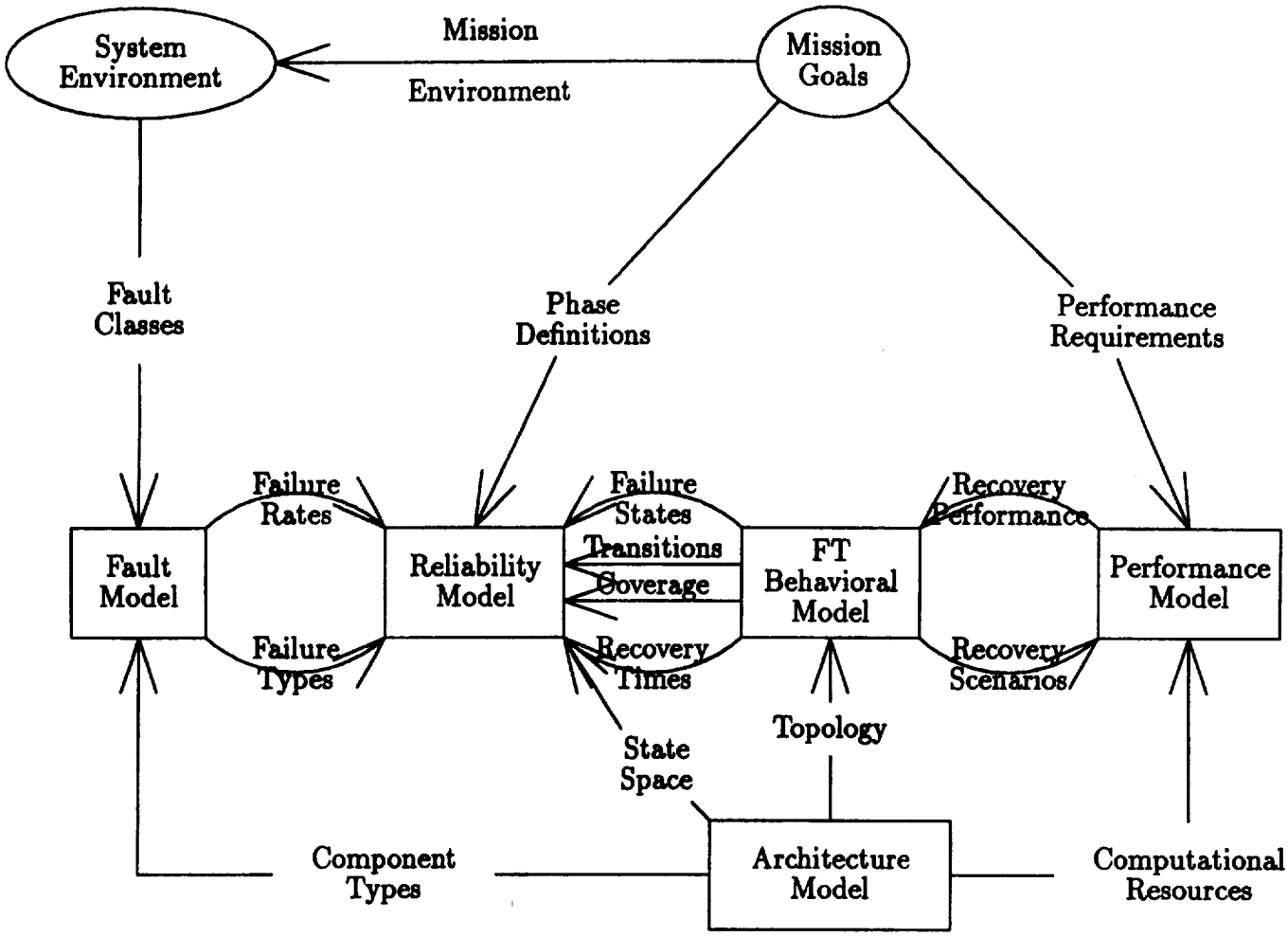


Figure 2.5. Data Flow Among Performance and Reliability Models

In addition to enhanced capabilities for performance and reliability analysis, integrated tools would support a design-for-reliability methodology in the areas of experiment planning and documentation for validation. Experiment planning is essential due to the large size of the design space, the use of tools at the limits of their capacity, and the need for regression testing of models. Additional research in experiment planning is needed in the areas of consistency checking and configuration management. Consistency checking has to address ways of assessing that the level of resolution is consistent across all system models and especially that there is consistency of resolution between different input parameters, between simulation time and input parameters, and between input and output. Configuration management policies have to address what consistency checks need to be applied and when they are required, as well as how to support regression testing. Tools for running experiments should allow the user to define the search space, to define the search strategies, and manage the files generated by and for the experiments. They should also support the validation of the models and aid the comparison of results from different runs.

A proposed structure for integrating performance and reliability analysis tools is illustrated in Figure 2.6. This structure would provide a shared data base to facilitate interaction between the models and an evaluation controller to implement the experiment planning and execution functions. In addition to the data required to build the performance and reliability model, the data base would include a library of models of primitive functions that could be used to build algorithm models; rules for the transformation of architecture models to meet requirements of specific applications, to reflect alternate configurations, or to achieve parallelization goals; and rules for mapping algorithms to architectures.

The evaluation controller would control the design and analysis space, maintain consistency among the models, and support model regression testing. To be able to control the design and analysis space, it should support application of appropriate tools at each design and analysis phase, searches of the design space, and pruning of unnecessary analyses. It should provide consistency maintenance through built-in configuration management and consistency verification. Also, since each design change requires some validation that previous constraints have not been violated, the evaluation controller should provide support for regression testing through back annotation and generation of consistent model configurations.

A configuration can be defined as the collection of files which describe and support the analysis of a particular model or of all the models of a system. Many versions of models and the files associated with them result from an iterative and hierarchical approach to design. Thus, it is important that configuration management provide

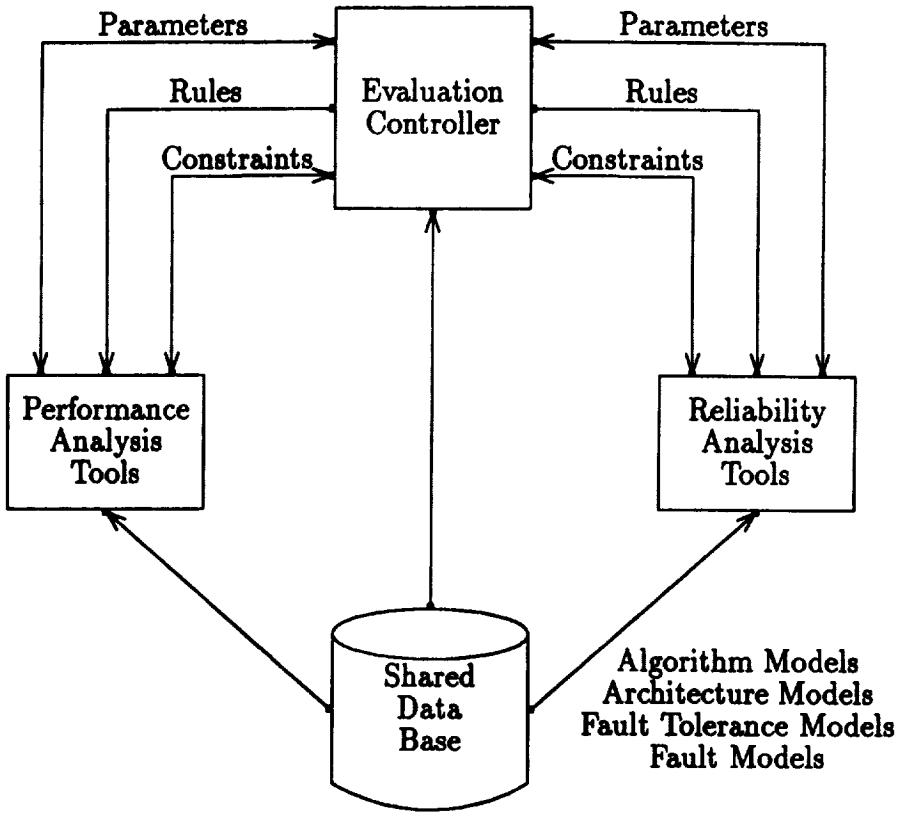


Figure 2.6. Integrated Performance and Reliability Analysis Tools

access to consistent files for use by the tools. Preliminary requirements for tools to support configuration management are that each file must contain a file version number, each tool must be able to verify that it is processing files for a consistent configuration, and each tool must include configuration information in its outputs. A tool can verify that it is using files from a consistent configuration by building its own configuration, by using a configuration built by the evaluation controller, or by accessing the configuration information in the data base. The configuration data base can be updated by either the evaluation controller or the tool, but consistency checking must be done whenever configurations are defined or modified.

The main issue in the design of the shared data base is the management of the heterogeneous collection of files that results from the many models required to describe and analyze the performance and reliability attributes of a system. Other issues are how to provide common storage for shared data to support both the generation of tool-specific input files and the extraction of parameters from output files and how to incorporate the configuration management facilities of the evaluation controller.

3. Paradigm

Based on the framework provided by the design for reliability methodology of [13], a paradigm for performance/reliability modeling in support of system development was created, as illustrated in Figure 3.1. In this paradigm, the system development phases from system concept to implementation and test are carried out in the appropriate system engineering domains under the guidelines of the methodology. As architectures and algorithms are developed in this process, performance and reliability tools assist the individual designers in evaluating and changing the designs and in maintaining the consistency of the designs with the overall system requirements and specifications. Selected results from the modeling effort are used to satisfy portions of the requirements for the various development milestones such as design reviews.

Ideally, it would be desirable to exercise the paradigm through the analysis of a complete "SDI-like" system to determine what system models are needed, how the models interact, what experiments and analyses are needed, how tools support the methodology, and what the tool features and capabilities should be. Lacking both the specification of such a system and the funding, it was not possible to exercise and assess all aspects of performance and reliability modeling support for the system development process. Consequently, effort was focused on those facets of the paradigm likely to reveal weaknesses in the existing methods and tools or likely to yield payoffs in the form of refinements to large portions of the methodology. To this end, areas where the characteristics of the complex space mission are distinguished from more ordinary applications were considered of special interest, and two algorithms and three architectures were selected for analysis.

A typical SDI application could have requirements to detect and track potential targets and to allocate weapons necessary to destroy targets. The signal/image processing algorithms that would be employed to provide target detection, classification, tracking, and trajectory estimation are computationally intensive. However, most often they can be decomposed into highly regular computational structures that can be effectively handled by vector and pipeline processing techniques. The optimal allocation of weapon resources to targets requires the use of algorithms that differ significantly from signal/image processing algorithms. These mission planning functions employ linear, integer, nonlinear, or dynamic programming techniques which have computational requirements that are dependent upon the incoming target data and that vary with the number of targets. These algorithms are more difficult to decompose and embed in a parallel computing architecture, and were therefore judged to be of particular interest for this effort. Two mission planning algorithms were

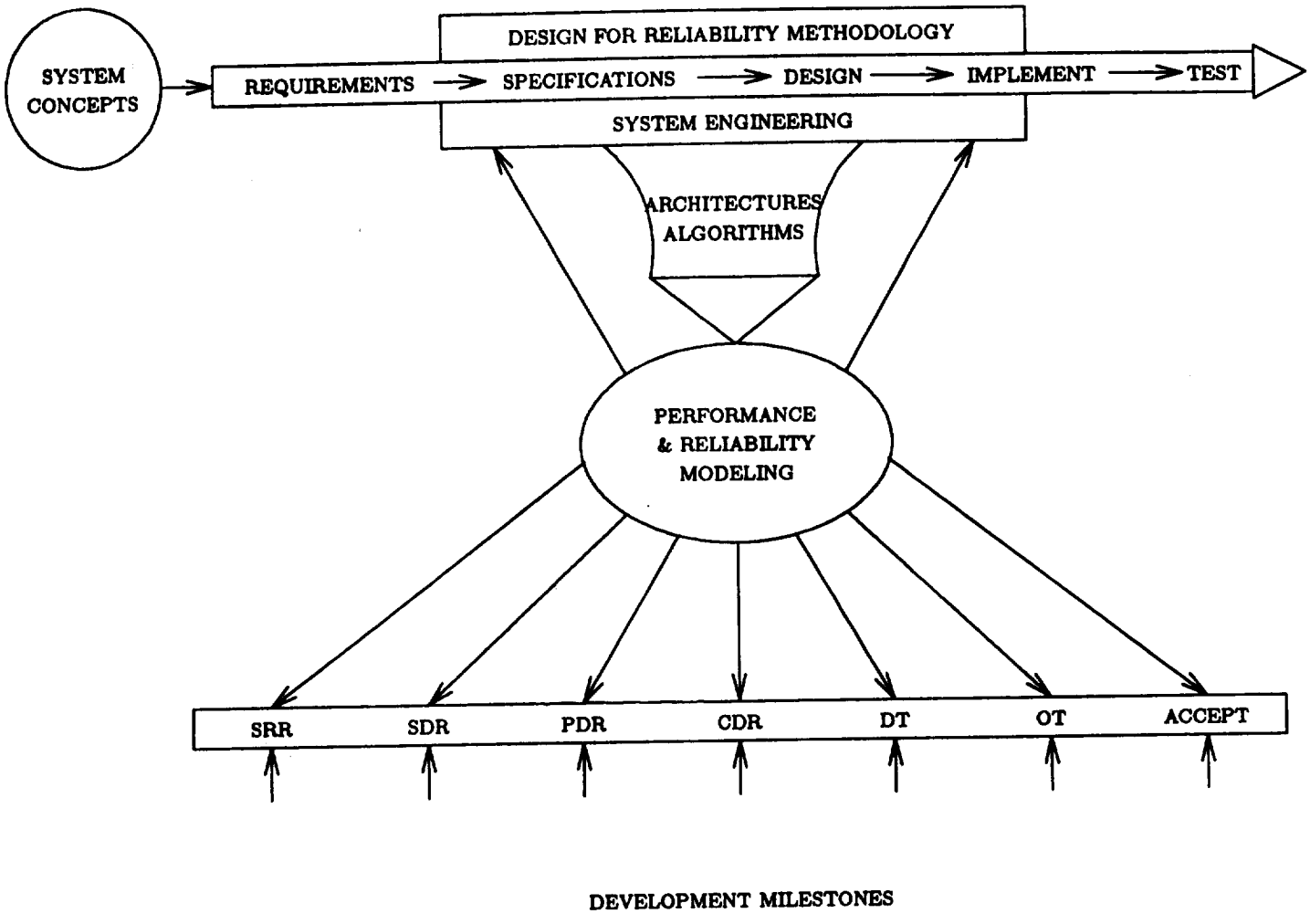


Figure 3.1. Paradigm for Performance/Reliability Modeling in Support of System Development

selected and used in this study. These algorithms are discussed in Section 3.1.

Meeting the demanding throughput requirements of such applications will require advanced architectures consisting of a large number of interconnected processors or computers. Of the various parallel computing architectures that have been proposed, three were selected for use in this study. The JPL Hypercube, an MIMD distributed memory architecture, was selected primarily because the hypercube is one of the most extensively investigated parallel computing architectures. The Encore Multimax, an MIMD shared memory architecture, was selected to provide contrast to the hypercube particularly in the area of interprocessor communications. Finally, the Charles Stark Draper Labs Fault-Tolerant Parallel Processor (FTPP) was selected because it is the only parallel processing architecture which has the advanced fault-tolerant features necessary to attain very high reliability. As such, modeling for the FTPP is distinguished from ordinary parallel processor modeling and should be expected to provide insight into weaknesses in the methods and tools as they pertain to high reliability applications. These architectures are discussed in Section 3.2.

An important aspect of the paradigm is the information required to carry out the modeling for various stages of the design process. As part of this study, an audit of the information used to construct the various models was conducted. This information is discussed in Section 3.1, Algorithms, and Section 3.2, Architectures.

3.1. Algorithms

The algorithms selected for the paradigm are two mission planning algorithms developed by Alphatech, Inc. The first algorithm solves the weapon-target assignment and target sequencing decision problem by breaking it into a four-level optimization problem [16]; the second, by conducting an auction among the targets for the available weapons [16].

3.1.1. Algorithm 1: Weapon to Target Assignment and Target Sequencing (WTA/TS)

This algorithm is directed toward the optimal assignment of space-based Directed Energy Weapons to multiple hostile boosters [16]. It solves this optimization problem by partitioning into four subproblems that can be solved iteratively. WTA/TS

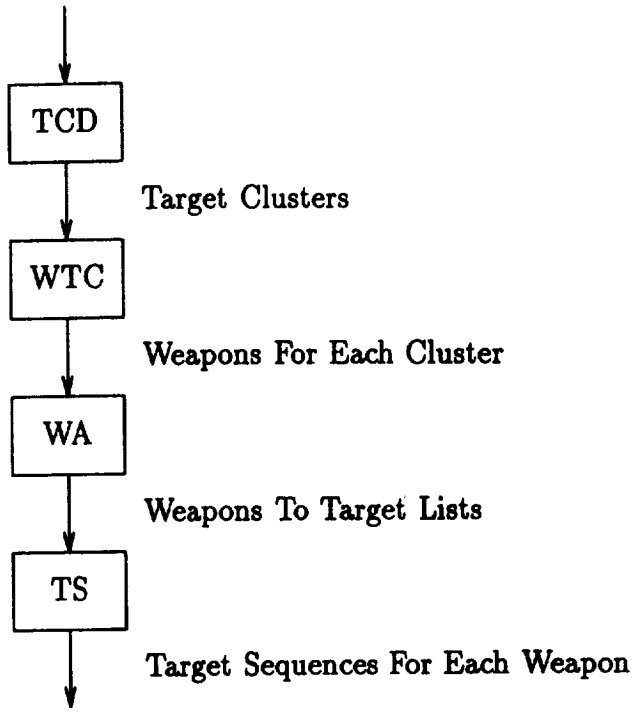


Figure 3.2. WTA/TS Data Flow Diagram

consists of four component functions corresponding to the four subproblems: target cluster definition (TCD), weapon-target cluster allocation (WTC), weapon to target assignment (WA), and target sequencing (TS). Figure 3.2 shows the major data flows among the WTA/TS functions.

The input parameters that control the size of the problem are the number N of targets, the number W of weapons, the number C of clusters to be formed, and the degree R of redundancy for weapon assignment. Other external inputs include the $N \times N$ target interdistance matrix, the $N \times W$ kill probability matrix, the $N \times W$ weapon slew time matrix, the $N \times W$ utilization matrix, the $N \times W$ target processing time matrix, the vector of N target due dates, the vector of N target values, the vector of N target processing times, the time start for the planning frame, and the time start for a new launch. Each of the WTA/TS functions is summarized below.

TCD: The target cluster definition function assigns targets to clusters based on intertarget distance. It seeks to optimize $Q = F[intertargetdistance]$, where each target is in one and only one cluster. The optimization problem is an integer (0/1) programming problem solved using LaGrange multipliers and a subgradient iterative

algorithm.

WTC: The weapon-target cluster allocation function balances the cluster allocation load across weapons, assigning multiple weapons to each cluster of targets. It seeks to optimize

$$Q = F\,[kill\ prob, target\ value, weapon\ use, reassign\ cost]$$

where a cluster is assigned to at least R_k weapons. The optimization problem is solved by breaking it into subproblems. The first subproblem is defined to be

$$Q_1 = \sum_{cluster} MIN\ F\,[killprob, targetvalue, reassigncost, use, LaGrange]$$

and solved as multiple integer programming problems. The general integer programming solution is illustrated in Figure 3.3.

The second subproblem is defined to be

$$Q_2 = F\,[Use, LaGrange]$$

and solved as simple scalar.

The two subproblems combine as $Q_1 + Q_2$ with

$$Max\ [Q_1[M] + Q_2[M]]$$

being solved via a subgradient method.

WA: The weapon-to-target assignment function assigns weapons to targets within a cluster based on kill probability, time required to switch from target to target, and the value of the target. It seeks to optimize

$$Q = F\,[killprob, switchtime, targetvalue, weapon]$$

for each cluster, where at least one weapon is assigned to each target. This problem is solved similar to the weapon to target cluster allocation problem.

TS: The target sequencing function establishes an optimum firing sequence for the list of targets assigned to each weapon. It seeks to optimize

$$Q = F\,[targetvalue, duedate, proctime, switchtime]$$

Seven suboptimal algorithms are under consideration for this function; these algorithms use dynamic programming concepts and two use heuristics.

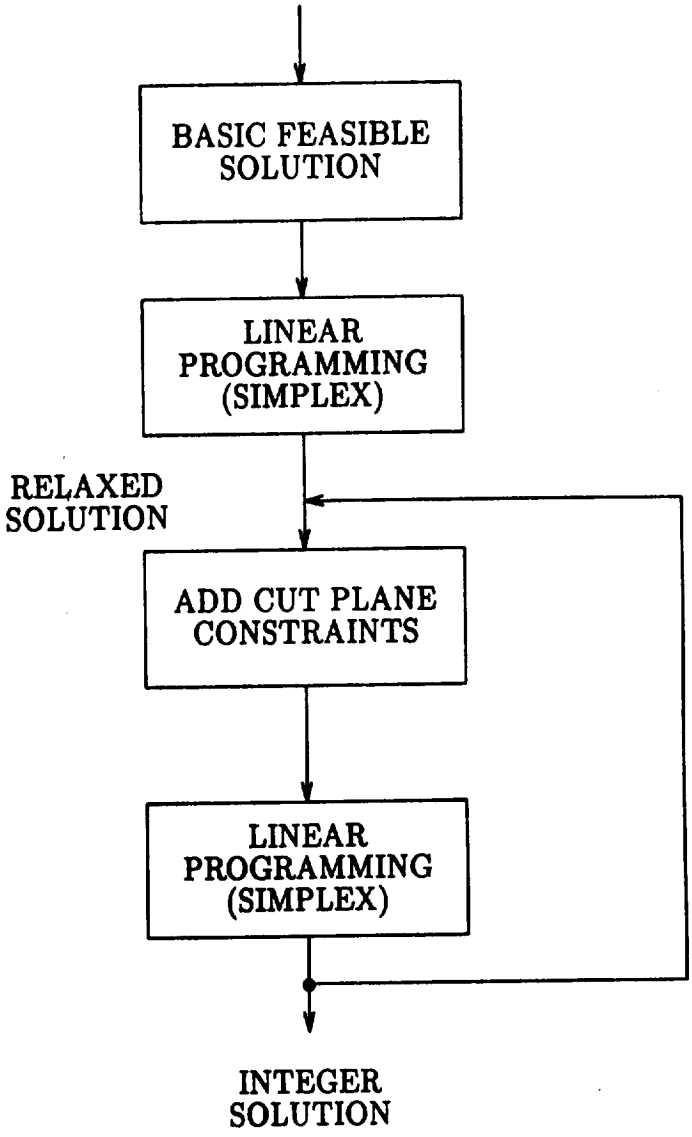


Figure 3.3. General Integer Programming Solution

3.1.2. WAUCTION_ASSIGNMENT

The WAUCTION_ASSIGNMENT algorithm is a target-oriented weapon-to-target assignment algorithm with the objective of minimizing the expected value of surviving targets [16]. ALPHATECH's solution to this non-linear, NP-complete problem is Iterative Linear Network programming (ILINE), which approximates the non-linear problem iteratively with a linear one [16]. Their formulation of ILINE is as follows:

Step 0: Initialize for all targets and weapons:

$$x^{p_{i,j}} \leftarrow 0$$

(the number of interceptors permanently assigned from weapon j to target i)

Step 1: Solve the linear programming problem:

$$\underset{x_{i,j}}{\text{MAX}} \sum_i \sum_j v_i p_{i,j} x_{i,j}$$

$$\sum_i x_{i,j} \leq m_j \text{ for every weapon } j$$

$$\sum_j x_{i,j} \leq 1 \text{ for every target } i$$

Step 2: Locate the weapon-target pair j^*, i^* yielding the largest value of $(v_i \cdot p_{i,j} \cdot x_{i,j})$ from the LP solution.

Step 3: Update the permanent assignments, weapon capacity, and target value; check for termination:

$$x^{p_{i^*,j^*}} \leftarrow x^{p_{i^*,j^*}} + 1$$

$$n_{j^*} \leftarrow n_{j^*} - 1$$

$$v_{i^*} \leftarrow v_{i^*}(1 - p_{i^*,j^*})$$

Stop if all interceptors from all weapons have been allocated.

Otherwise, return to Step 1.

WAUCTION_ASSIGNMENT implements ILINE using a modified auction strategy to produce the linear programming solution in Step 1. This auction strategy is based on the auction algorithm developed by Bertsekas [16]. The weapons, targets, and the feasible matchings between them form a network. Initial values are assigned to each target and a minimum bid increment is specified. Cycles of bidding in which targets bid for weapons alternate with assignment phases. The initial value of a target is updated to an expected value after the target is assigned to a weapon. These expected values are computed as the product of target value and the kill probability of the weapon-target pair. The bidding phases along with the assignment of expected values and the specified minimum bid increments allow alternative weapon-target assignments to be considered, thus increasing the optimality of the solution. The bidding-assignment phases are conducted iteratively within an outer, non-linear iteration until all interceptors from all weapons have been allocated to targets and the targets of greatest value have been assigned. The inputs to the WAUCTION_ASSIGNMENT algorithm are the number of weapons, number of targets, number of assignments to be made per iteration, the bid scaling constant, the kill probability matrix, the initial target values, the number of interceptors per weapon, and the minimum kill value threshold. The outputs from the algorithm are the assignment matrix of number of shots per weapon i assigned to target j and the expected surviving value of each target. A data flow graph of WAUCTION_ASSIGNMENT is shown in Figure 3.4.

3.2. Architectures

The primary goal of this research is to determine the nature and requirements of system engineering tools for the performance analysis of algorithm/architecture combinations. Since the specific algorithms are "representative", their absolute performance is not by itself of much value. Of greater interest is the structure of the algorithms and the techniques and associated tools used to analyze the target architectures.

In order to increase the coverage of tool and methodology areas of interest, three systems representative of distinct classes of architectures were analyzed. The JPL Mark III Hypercube represents a class of distributed-memory, loosely coupled architectures. A major issue in the design of efficient parallel algorithms for such machines is the minimization of interprocessor communication. Solutions to related problems such as algorithm partitioning, mapping, and scheduling, attempt to achieve this goal. The Encore Multimax represents the class of shared-memory, tightly coupled architectures. One of the principal determinants of machine performance is conflicts in accessing of shared-memory. These conflicts occur because of contention for the

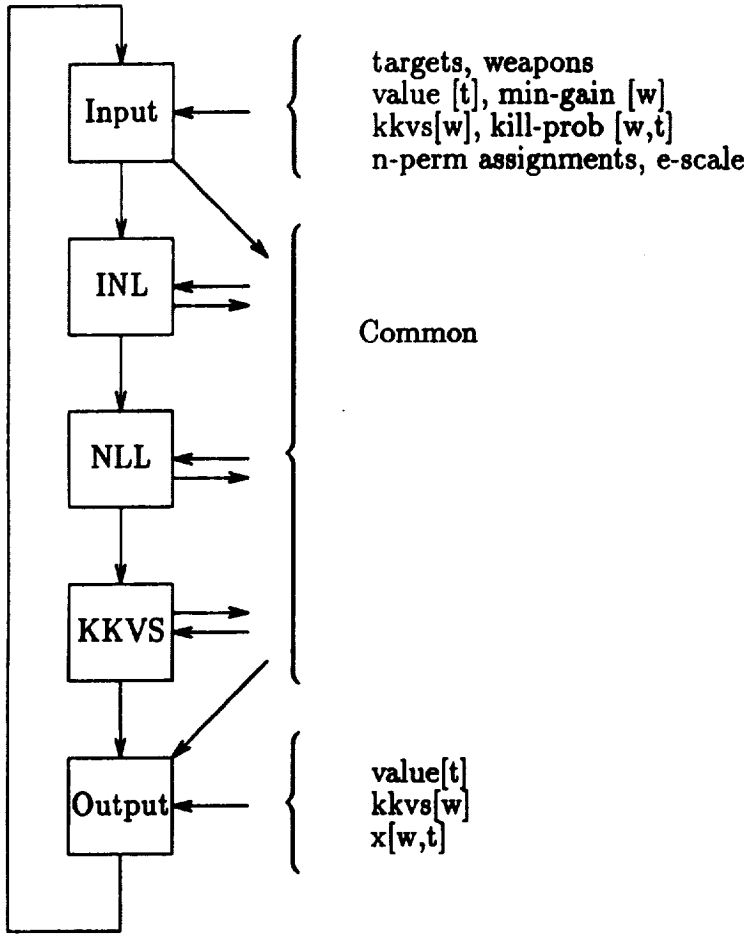


Figure 3.4. WAUCTION_ASSIGNMENT Data Flow Diagram

processor-memory bus, or in access to shared variables. Again, solutions to related problems attempt to minimize such access conflicts. The Charles Stark Draper Laboratory Fault-Tolerant Parallel Processor represents the class of architectures designed for fault tolerance. It is a Byzantine Resilient Multiple Instruction stream Multiple Data stream computer designed to produce orders of magnitude more processing power than current highly reliable systems. The fault tolerance mechanisms of this architecture are of particular interest, especially the impact of increased communications requirements on performance.

One of the advantages of choosing fundamentally different types of architectures is the differing demands they will place on the system analysis tools. Let us consider the algorithm representations described in the previous section. Two nodes in this graph that are ready to fire cannot do so if they both require the same resource. In distributed memory architectures, tasks executing on distinct processors utilize separate processor-memory paths. Only tasks (communication nodes) mapped to the same processing element (link or intermediate routing node) are so constrained. Thus, for a given mapping algorithm, the resolution of the simulation can be that of the firing delay of the smallest task. The accuracy of the firing delays themselves can be computed based upon detailed knowledge of the operation counts of the tasks and the architecture of the individual node. Based on these characteristics, the GIPSIM utility of ADAS is suitable to analyze the performance of the Mark III Hypercube.

By definition, processors in bus-based shared-memory architectures require access to a shared resource on potentially every instruction execution cycle. The execution of every instruction competes with each other since they all require the use of the global bus for memory access. Multiple processors can also conflict on every cycle in the access to shared memory. Thus, these architectures are very dependent upon the mapping of (shared) data structures into the memory modules. Furthermore, memory modules tend to be heavily interleaved to match the processor/bus bandwidth. The normal approach to achieving high simulation resolution is to map tasks to processors and all communication nodes to the bus, essentially serializing all communication. However, most architectures use some form of a cache to alleviate the processor/bus/memory bottleneck. The performance depends very heavily on the use of the cache. In the case of the Encore Multimax, modeling is further complicated by the fact that pairs of processors share access to a single cache. Cache coherency control adds to the memory traffic in a non-trivial way and must be a part of the analysis. The interactions between the processor, cache, bus and memory are at the level of the fastest device (typically the cache or bus), and this dictates the level of resolution of the simulation. Since phenomenon such as cache hit rates and bus arbitration are generally non-deterministic, an approach to simulation must enable behavioral

modeling while explicitly preserving the structure of the architecture. Therefore, the CSIM facility of ADAS is required to attain the necessary simulation resolution.

To summarize, the simulation resolution that can be achieved depends upon the granularity of parallelism being modeled. High-level resolution simulations are possible for loosely coupled systems, and a relatively simpler, purely structural simulation, is possible (GIPSIM). For modeling tightly coupled, fine-grained architectures, one needs low-level simulation resolution. Furthermore, the introduction of a non-deterministic phenomena such as cache operation requires the capability for modeling stochastic behavior (CSIM).

3.2.1. JPL Mark III Hypercube

A binary k-cube of P = 2^k processing elements (PE) is arranged as a p = log_2P dimensional hypercube. Each processing element is represented by a binary address $p_{n-1}p_{n-2},...,p_1p_0$, and is connected to all processing elements whose address differs from it in exactly one bit, i.e., $p_{n-1}p_{n-2},...,\bar{p}_i,...,p_1p_0$, for all i. The topology of a sixteen-node hypercube is shown in Figure 3.5. The architecture of each hypercube node or PE is illustrated in Figure 3.6. The node architecture is comprised of three basic blocks: the communication processor, the data processor, and the floating point array processor.

The communication processor (CP) is a 32 bit MC68020 and is dedicated to handling interprocessor communication and node I/O. Each node is interconnected to its neighbors by channels. Associated with each channel is a 64-byte buffer. Interprocessor communication is achieved by reading/writing the channel buffers as 64-byte packets. Thus, if processing element 1 in Figure 3.5 is to send a packet of data to processing element 0, a packet of data is transferred from the memory of PE 1 to the input buffer of PE 0 on the appropriate channel. This sequence is repeated several times for multipacket transfers. When transferring data from local memory to a channel buffer (or vice-versa) the CP has higher priority access to local memory than the data processor, and the latter stalls. The CP static RAM holds the node operating system which is responsible for interprocessor communication and synchronization. On receiving a packet in one of its buffers, the CP checks the destination. If it is local, it is transferred to local memory. If it is not, it is transferred to a buffer on the appropriate channel as defined in its destination or routing tag. Thus the three modes of data (or equivalently, packet) motion the CP is responsible for are local-memory-to-channel-buffer, channel-buffer-to-channel-buffer, and channel-buffer-to-local-memory. These are the modeling parameters of interest.

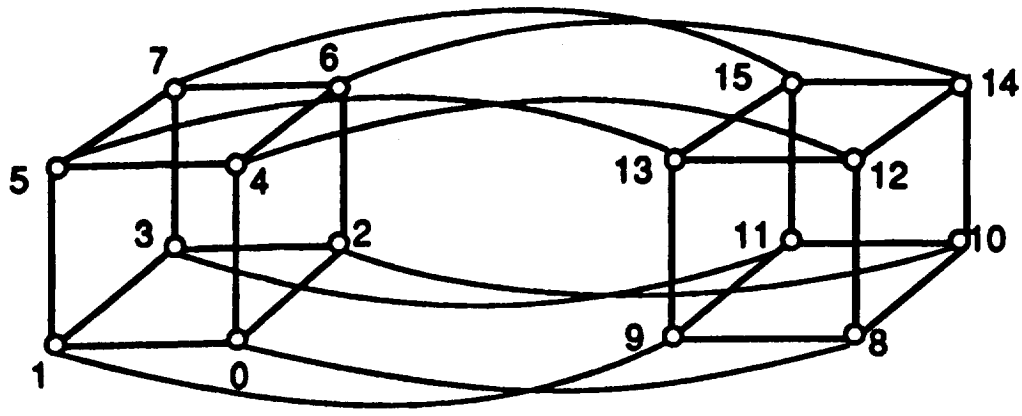


Figure 3.5. 16 Node Binary Hypercube

The data processor is also a MC68020 and operates with the MC68851 conventional memory management unit and a MC68881/2 floating point co-processor. In addition, there is an array processing unit built around a 30 Mflop Weitek chip set. This processor runs as a slave to the data processor. The modeling parameters of interest are the interrupt latency to respond to the presence of a message, the speed of the processor, the speed of the array processor, and the memory access latency.

3.2.2. Encore Multimax

The Encore Multimax is a shared memory tightly coupled multiprocessor architecture. The overall organization is illustrated in Figure 3.7. The principal components are the Dual Processor Card (DPC), the Shared Memory Card (SMC), and the System Control Card (SCC). The remaining components are not of direct interest from the point of view of computing performance. All of the components are configured around the 100 Mbytes/sec nanobus. Current configurations provide up to twenty processors.

The DPC is comprised of two processors which share access to a 32K instruction/data cache. The cache reduces the traffic on the nanobus, and features a write-through update policy with snooping cache controllers to enforce cache coherency when multiple processors update shared data. The processors are from the NS32X32 family of microprocessors and currently can be augmented with special-purpose floating-point

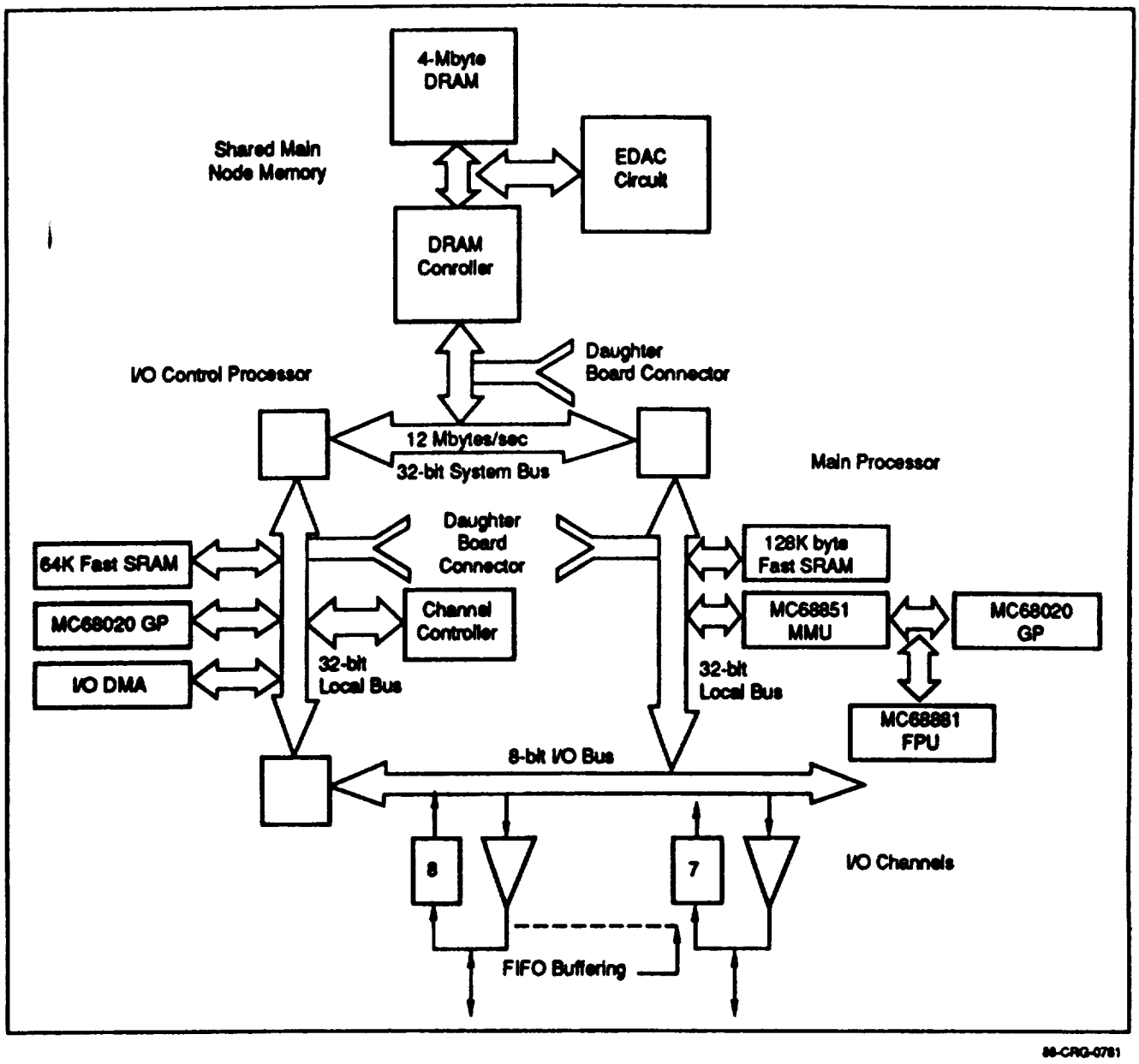


Figure 3.6. Mark III Node Architecture

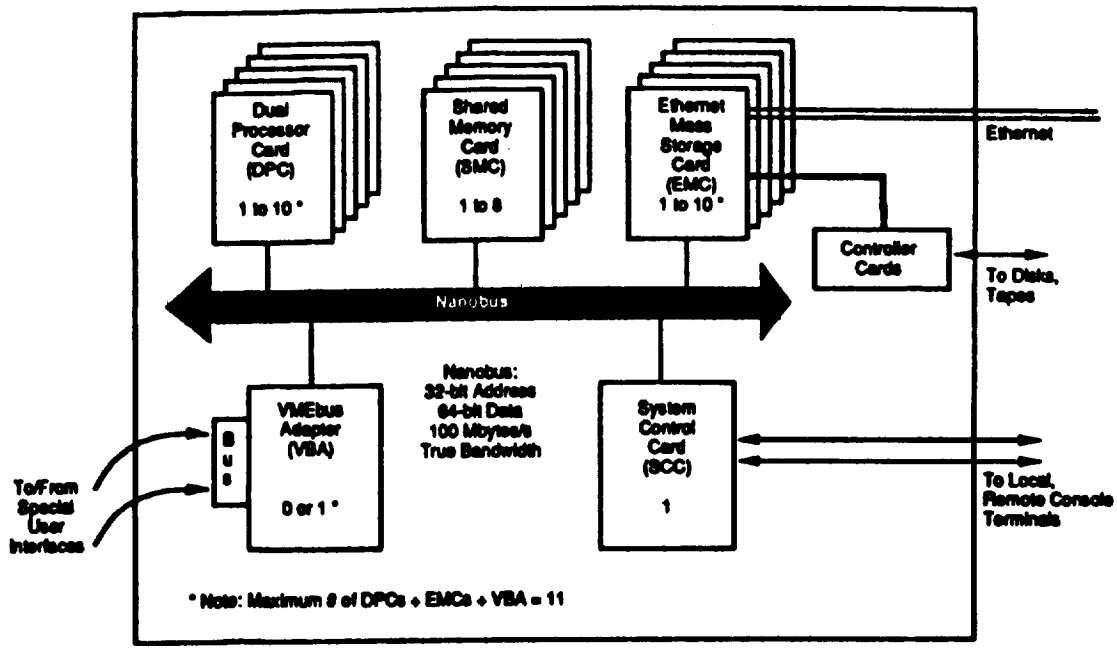


Figure 3.7. Encore Multimax

hardware. Each SMC provides up to 16 Mbytes of RAM in two independent banks. In addition, 4-way interleaving among SMCs is permitted, providing 8-way interleaving for structured accesses. The SCC functions as the bus arbitrator and diagnostic center for the Multimax. The bus allocation strategy is a "rotating" daisy chain — the current bus master becomes the lowest priority requester in the next arbitration cycle.

The Multimax is programmed by using shared memory locations as locks to coordinate multiple processes. Code is stored in a single memory module while data is distributed across memory modules. In particular, access to matrices can take advantage of the interleaving in the memory system design.

3.2.3. FTPP

The FTPP architecture was proposed in an effort to combine the disciplines of fault-tolerant computing and parallel processing to yield theoretically sound design principles for a high throughput, highly reliable computer [7]. In particular it was designed to provide greater performance than that achievable by current fault-tolerant systems, especially the fully connected cluster-based architectures such as the Software Implemented Fault Tolerance (SIFT) and Multicomputer Architecture for Fault-Tolerance (MAFT) computers, while maintaining their high level of reliability. The architecture consists of a partially connected network of clusters of multiple primary fault containment regions (PFCR) consisting of an input/output element and multiple processing elements connected to a network element. The network elements are specialized hardware components that execute the synchronization, voting, and consistent ordering protocols required to achieve Byzantine Resilience. This allocation of protocol tasks to the network elements prevents the attached processors from being diverted from application tasks. The FTPP architecture components are illustrated in Figure 3.8, Figure 3.9, and Figure 3.10 from [7].

The configuration of processing elements into computational sites is constrained by the necessity of allocating each channel of a redundant processing (fault masking) group to a different network element so that each channel will belong to separate PFCRs. Figure 3.11 [7] illustrates a possible cluster configuration of 16 processors into one quadruplex (Q1), one triad (T1), and nine simplexes (S1-S9) distributed over four network elements. Note that the quadruplex channels are distributed over each of the four network elements, the triad over three, and the simplex over all four. To support the distribution of the computational sites, the network elements that form a cluster are fully connected by point to point communications links.

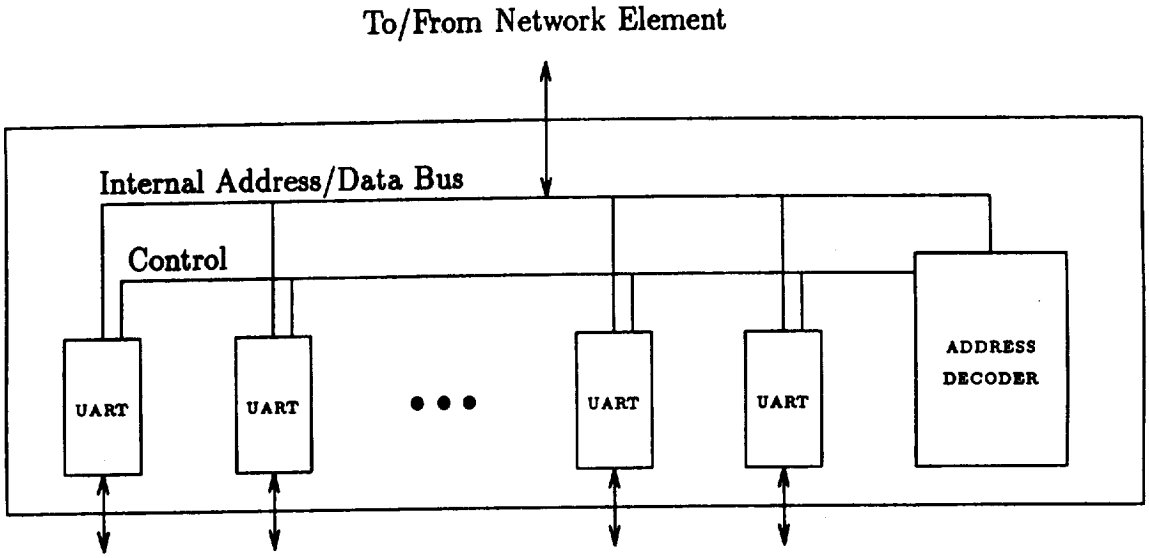


Figure 3.8. FTPP Input/Output Element

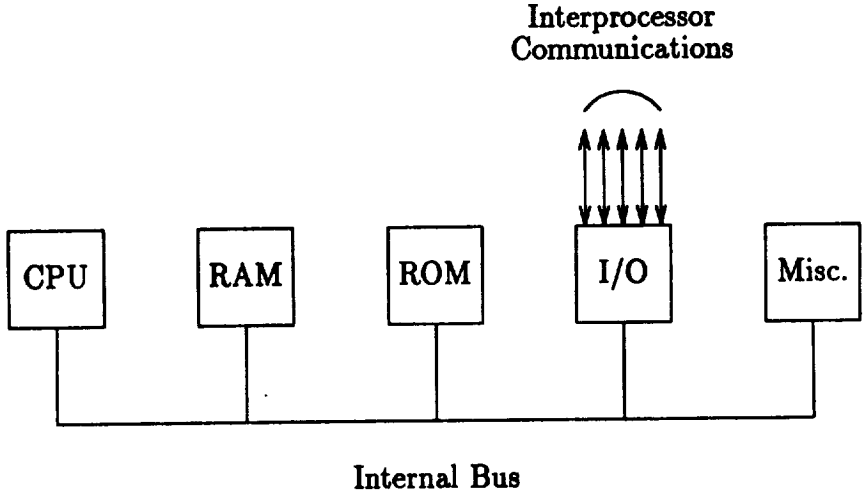


Figure 3.9. FTPP Processing Element

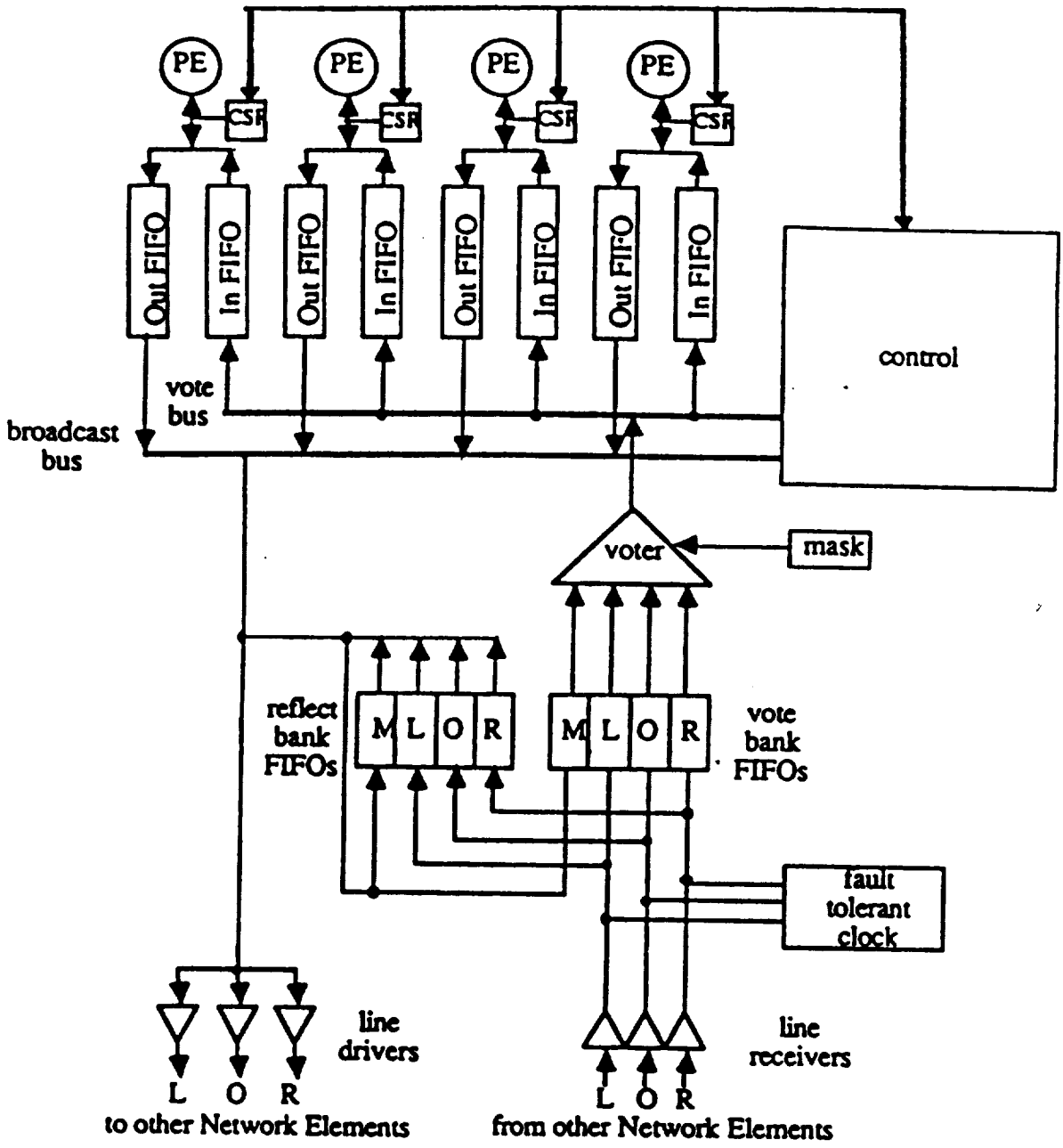
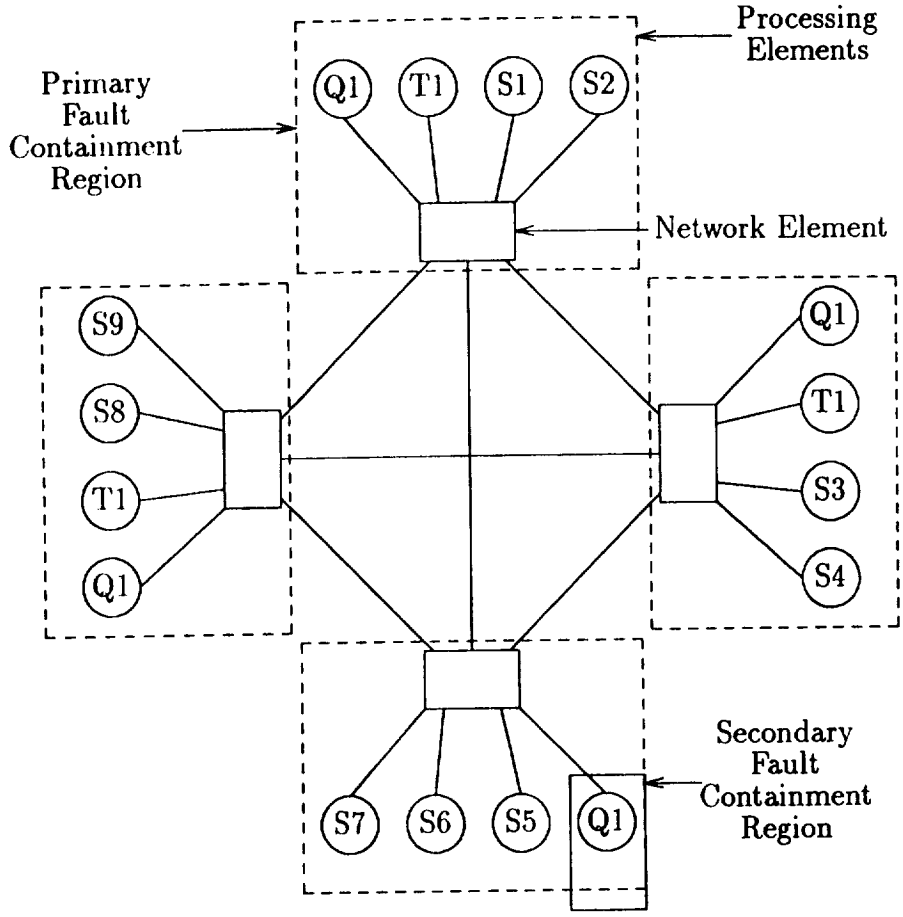


Figure 3.10. FTPP Network Element



Q1 Quad 1
T1 Triad 1
S1-S9 Simplexes 1-9

Figure 3.11. Possible 16-Processor Cluster Configuration

Since the execution speed and communications link bandwidth of the network element are not sufficient to support the large numbers of processors that many applications require, multiple interconnected clusters can be built. The special input/output elements that are attached to each of the network elements in a cluster are used to implement the connection. Figure 3.12 [7] illustrates a possible multicluster topology, but the number of clusters and the connection topology, as well as the number of network elements per cluster and the number of processing elements per network, are determined by the performance and reliability requirements of the application.

The fault tolerance mechanisms of the FTPP are based on the Byzantine Resilient Virtual Circuit (BRVC) Abstraction, which guarantees that messages sent by one FMG to another are delivered uncorrupted and in the order sent, that all non-faulty members of a recipient FMG receive messages in identical order, and that the absolute times of arrival of corresponding messages at the members of a recipient FMG differ by a known upper bound (skew). In the FTPP there are sixteen message exchange classes of two basic types. Messages that emanate from a source sufficiently redundant that a vote will guarantee receipt of identical information at all destinations are Class 1 exchanges. Those from a single source are Class 2 exchanges. A Class 1 exchange thus originates from a FMG, consists of a broadcast among all network elements associated with the source and destination FMGs and a vote on the received message, and can be completed in one round. A Class 2 message exchange is originated by a single processor; it consists of a broadcast among NEs, a reflection of the received message, and a vote on the reflected message, and thus requires a two-round interactive consistency protocol.

Therefore, the redundancy management overhead is dependent upon the mix of exchange classes utilized in the application.

The FTPP network element provides the message exchange services for the cluster. It operates in cycles consisting of frames to recognize and classify exchange patterns, to decide which messages to transmit, and, finally, to transmit, vote and receive messages. For inter-cluster communication, messages are exchanged among connected IO elements attached to the network elements. A voted message from the relevant network elements is sent to the designated IO element group, where it is validated by message exchanges occurring at a specified rate. When a group receives a message for another cluster, all group members transmit that message to the IO group in the destination cluster. The source IO group also computes routing information when a message must pass through intermediate clusters to reach its destination.

The prototype FTPP architecture uses 12.5 MHz 68020 processors, 68881 floating

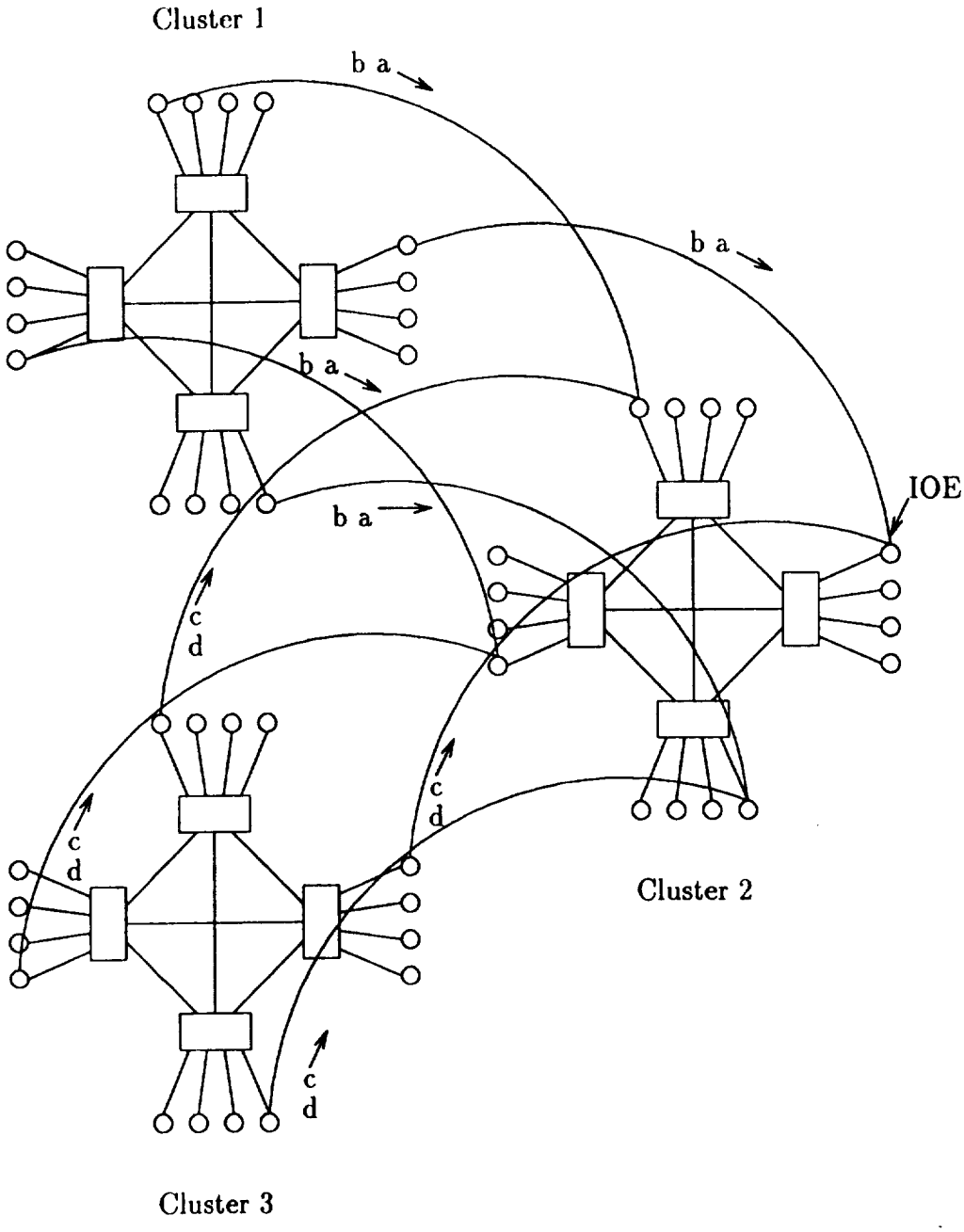


Figure 3.12. Possible Multicluster Topology

point coprocessors, 1 MB RAM, 128 KB EPROM, and 64 Mbps buses (8MHz $\times$ 8 bits).

3.3. Paradigm Framework for Analyses

Since the aim of the paradigm was to relate tools to the methodology, a framework was constructed from which analyses could be selected that would illustrate the role of particular tools and analyses in the design process. This framework specifies the characteristics and interactions of system models in the early design stages. It also specifies the information required for the analyses and the decisions that can be made from the analysis results. From this framework, process graphs were constructed to show the information flow and the decision points of particular modeling phases.

Figure 3.13 illustrates a high-level process that can be used to determine the required processing resources for a system based on mission requirements and to express those resources in terms of mission parameters. Figure 3.14 illustrates the process of making architectural trade-offs by building on the high-level process, refining the model descriptions, and providing more detailed information about the system.

Based on these process graphs, particular performance and reliability analyses of the algorithms and architectures described in the preceding sections were selected to illustrate the types of models that are needed, how the models interact, and the types of experiments and analyses that can be conducted.

The models that were constructed and the performance analyses that were carried out are described in Section 4; the reliability models and analyses are described in Section 5.

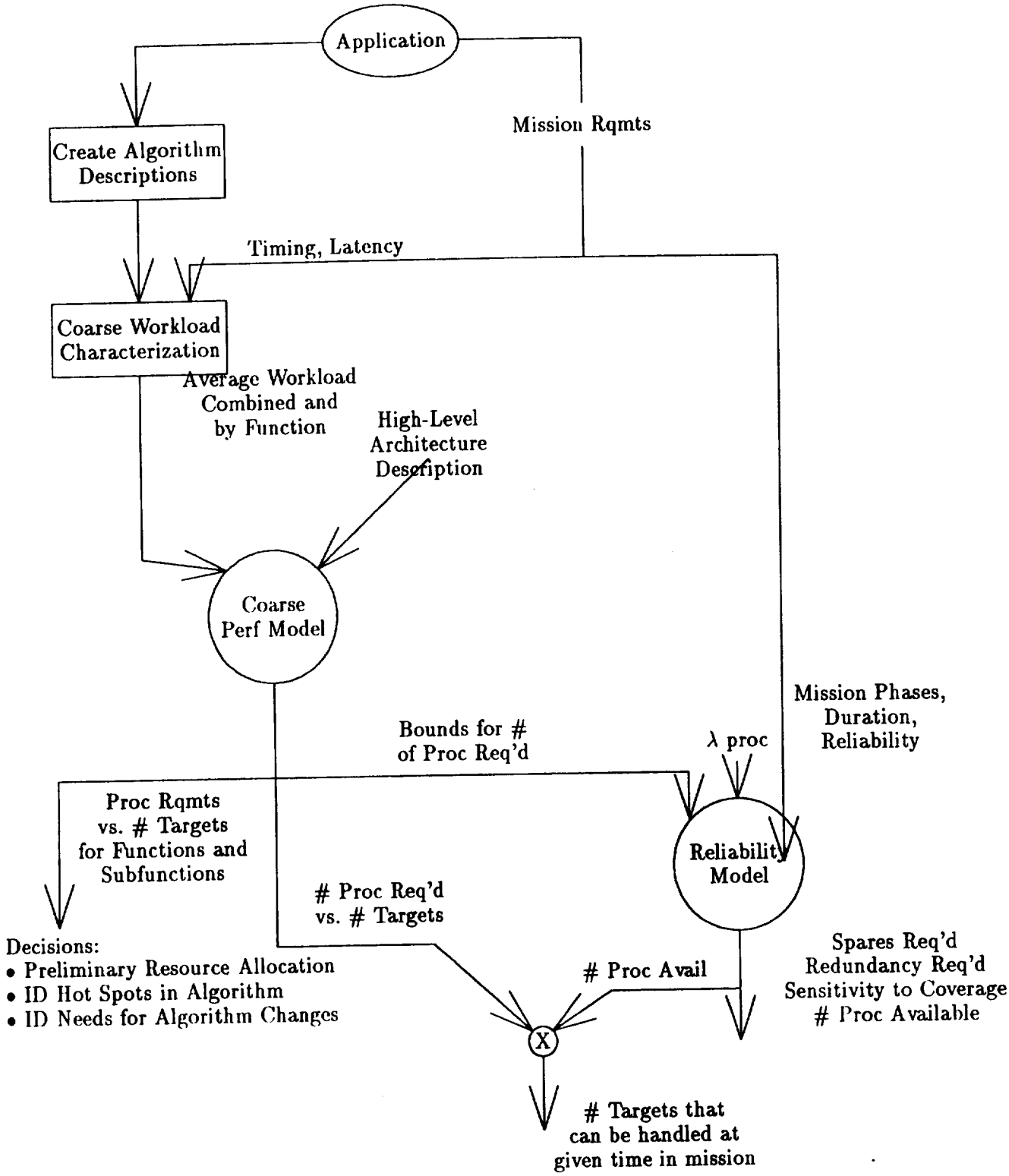


Figure 3.13. High-Level Process Graph

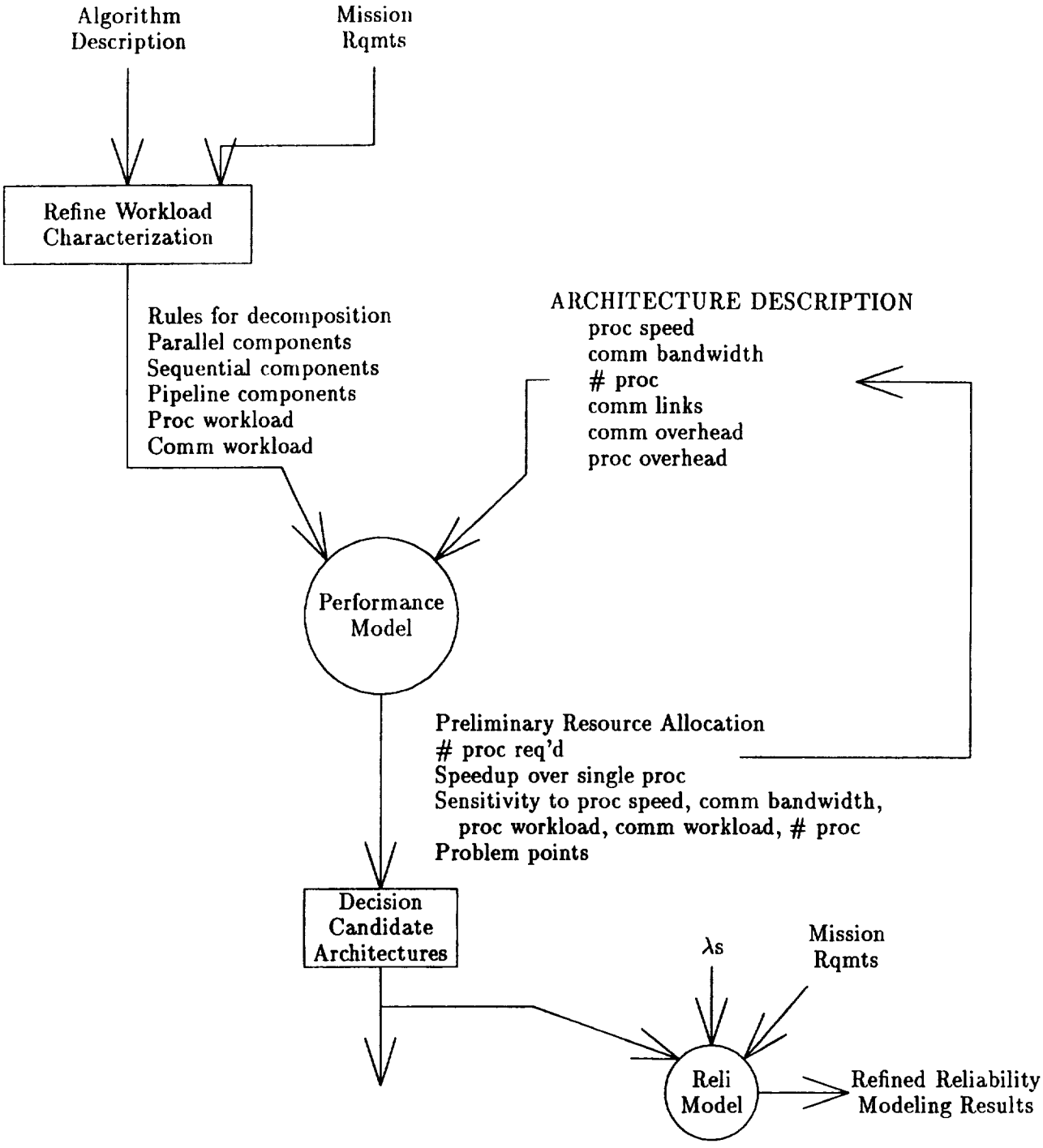


Figure 3.14. Refined Process Graph

4. Performance Analyses

This section describes the performance modeling and analyses that were carried out for highly parallel, highly reliable systems within the framework described in Section 3. The cases for study were selected so that issues related to the design of distributed fault-tolerant systems could be assessed. In particular, they were selected so that the adequacy of existing methods and tools could be tested. In this Phase I of the study, emphasis was placed on the impact of parallel processing issues on the methods and tools; even though some aspects of fault tolerance were drawn into the performance modeling, it was planned that the fault tolerance issues would be emphasized in the second phase of this program.

There are many opportunities for performance modeling in support of parallel architecture design, particularly in the area of embedding applications in an architecture so that the processing resources are well utilized. In this area, performance modeling can support evaluating the relative effectiveness of various granularity levels and algorithm decompositions, and can aid the analysis of load balancing, resource scheduling and contention, and deadlock prevention; studies of interprocessor communications; data flow analyses to identify data/program dependencies; and the analysis of resource utilization and memory sizing. The specific issues investigated in this effort include the impact of network communications on performance and the impact of fault tolerance on communications, the speedup and effectiveness achievable through varying levels of parallelism as a function of processor/communication speed and the existence of sequential and non-parallelizable tasks, the embedding of algorithms in architectures, and the distribution of workload by function and by processing resource.

The performance modeling activities conducted to examine these issues included creating algorithm descriptions, building models of algorithms and architectures, selecting and developing performance assessments, conducting the assessments with the chosen simulation tools, analyzing the simulation results, and assessing the use and limitations of the methods and tools.

Figure 4.1 lists the areas of performance modeling identified in the paradigm framework that are investigated in this research effort and identifies the particular case studies for each area.

AREA	CASES	
High-level workload and workload distribution	WTA/TS	WAUCTION
High-level decomposition and parametric studies	Generic Model for parallel decomposition	WTA/TS on FTPP
Interprocessor communications	WTA/TS — high level	detailed level
Functional simulation	MULTIMAX cache	reconfiguration algorithms
Measurement to support model development and validation	WAUCTION	
Detailed mapping, scheduling communications and resource contention	WTA/TS on Hypercube	WTA/TS on MULTIMAX
Fault Tolerance/performance	WTA/TS on FTPP	reconfiguration algorithms

Figure 4.1. Summary of Performance Studies

4.1. Development of Algorithm Descriptions

The primary concern of the algorithm description effort was the identification of the information, with respect to both type and resolution, that would be required to ensure adequate fidelity of the performance modeling at the various stages in the development process. It was also important to assure that in the early stages of the development process the algorithm descriptions were not so detailed as to preclude potentially desirable system configurations.

Considerable effort was necessary to develop the algorithm descriptions used for the development of performance models. These descriptions were developed from information that was less complete than would normally be available to the system engineering process. This was due in part to the status of the algorithm development and to the fact that the algorithm developers were not directly involved in this effort. Assumptions were made regarding algorithm behavior and implementation that in a normal development process would be known to the system engineers. However, the format, content, and detail is representative of that available early in the system engineering and design phases of a development program. As such, the role that these descriptions play in the modeling process, the fidelity of the resulting model, and the need for incorporating additional information in them can be assessed for the purpose of improving methods and tools.

The algorithm descriptions include a diagram of the major subfunctions within the algorithm. For each subfunction, the inputs, processing, and outputs were described. Where appropriate, subfunctions were further decomposed and inputs, processing, and outputs were identified and described for each resulting component. Data inputs and outputs were characterized by their source or destination, by their size as functions of system parameters such as numbers targets or weapons, and by their type such as numerical or logic variables. Subfunction processing steps were described wherever possible. To the extent possible, data dependencies, data addressing patterns such as linear or random and opportunities for parallel decomposition were included in the descriptions. Diagrams, called N-square diagrams, indicating all inputs and outputs for each subfunction and their relationship to other subfunctions were included in the descriptions.

Real time processing constraints were specified for each algorithm. Finally, estimated operation counts and memory access counts were included for each subfunction. These operation counts constituted the primitive information used to characterize workloads for the performance models.

The format followed for these descriptions is similar to that used for computer program performance specifications used for DoD system development prior to the use of DoD-STD-2167 documentation standards.

A description of a weapons-to-target assignment and target sequencing algorithm (WTA/TS) is given in [2]. This description was developed from a document [16] which described a proposed strategy for obtaining optimal weapons-to-target assignment and target sequencing for a constellation of directed energy weapons. For each step in the proposed strategy, cost functions, constraints and solution techniques were suggested. This information was not adequate for high fidelity performance modeling. To create an adequate algorithm description, assumptions regarding the type and characteristics of the required computations were made. The primary assumptions involved the characteristics of linear integer programming methods to carry out the constrained optimizations indicated in the proposed strategy. Further assumptions involved the average number of iterations required to obtain optimal solutions. Another assumption made to simplify the algorithm description and subsequent performance modeling was that the number of targets assigned to a target cluster was equal to the total number of targets divided by the number of clusters formed. While these assumptions impact the specifics of the performance modeling, the resulting descriptions were adequate to serve the primary purpose of the paradigm which was to exercise tools and methods with "SDI-like" examples.

The algorithm description for the WAUCTION algorithm [14] was derived from a software implementation of the weapons-to-target assignment technique. In contrast to WTA/TS, the description for WAUCTION did not require assumptions to be made regarding the computational methods used to implement the algorithm. Further, the dependence of workload on parameters such as numbers of targets and numbers of weapons could be determined by executing the code. Consequently, the algorithm description for WAUCTION contains more detail and higher fidelity estimates of computational workload.

The effort required to develop the algorithm descriptions could be reduced by including additional features in the modeling tools. Use of computer-aided software engineering (CASE) tools to specify and create the algorithm descriptions would be a more effective way to handle the development of algorithm descriptions.

An additional benefit would be that performance models could be automatically constructed from these algorithm descriptions which in turn could be transferred to the performance modeling tool via an appropriate CASE tool/performance modeling tool interface. Effort could be further reduced by incorporating a library of models for a wide range of common processing algorithms such as sort algorithms and linear programming algorithms.

4.2. WTA/TS High-Level Workload Analysis

Typically, the design of a system progresses from high-level, low-fidelity knowledge of the design to detailed low-level, high-resolution design details. In the early stages, the high-level information can be used to identify critical design problems, to select potential candidate architectures, to determine certain system characteristics and to eliminate system configurations which are not feasible. High-level performance modeling results can support this phase of the design process by providing quantitative measures of the computational workload.

One of the first steps in selecting the architectures to be considered is to obtain a coarse estimate of the workload requirements. This estimate can be used to establish the approximate number of processors required and the speed of the processors. To illustrate how this first step is carried out and how the results can be used to guide the design process, the WTA/TS algorithm was considered for a single processor system. ADAS software graphs were developed for the algorithm. The graphs shown in Figures 4.2–4.6 represent a two-level breakdown of the algorithm.

Figure 4.7 shows an example of the ADL files that were created for each graph to establish the parameters to be used in the ADAS simulations. The parameters such as number of targets and weapons are used to determine the number of operations required for various functions in the algorithm. These operation counts in turn determine the associated firing delays in the ADAS simulations. An implementation expansion factor or multiplier was used to convert the primitive operation counts to implementation operation counts. A factor of five was selected for this analysis. In an actual design study, this factor may be determined either by experience or by additional analysis. In addition to the implementation expansion factor for operation counts, assumptions regarding the average number of iterations required for the WTA/TS integer programming algorithms to converge to a solution were made. In a more precise analysis these parameters would have to be better characterized.

The total execution times required as a function of number of targets and weapons for a single processor operating at an average of 1×10^6 operations per second are given in Figure 4.8. These results can be used by the system designer in various ways. For example, if 5 seconds has been budgeted for the WTA/TS algorithm when 100 targets are being processed, a single 1×10^6 operations/second processor cannot meet the requirements. A single processor of 25×10^6 operations/second capability could meet the requirement. The system designer may wish to consider a multiple processor system. Under the best conditions, twenty-five 1×10^6 operations/second

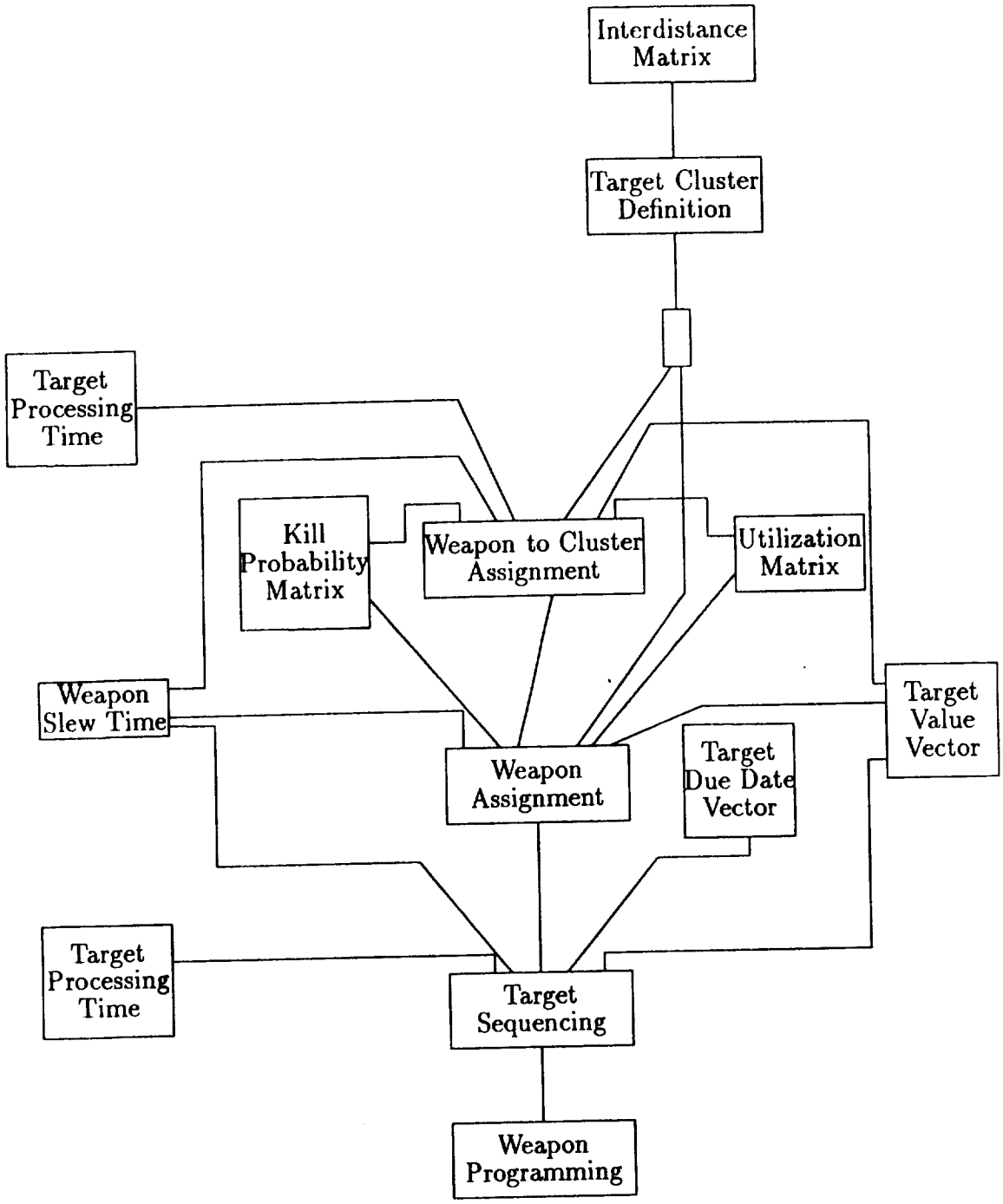


Figure 4.2. Top-Level WTA/TS ADAS Graph

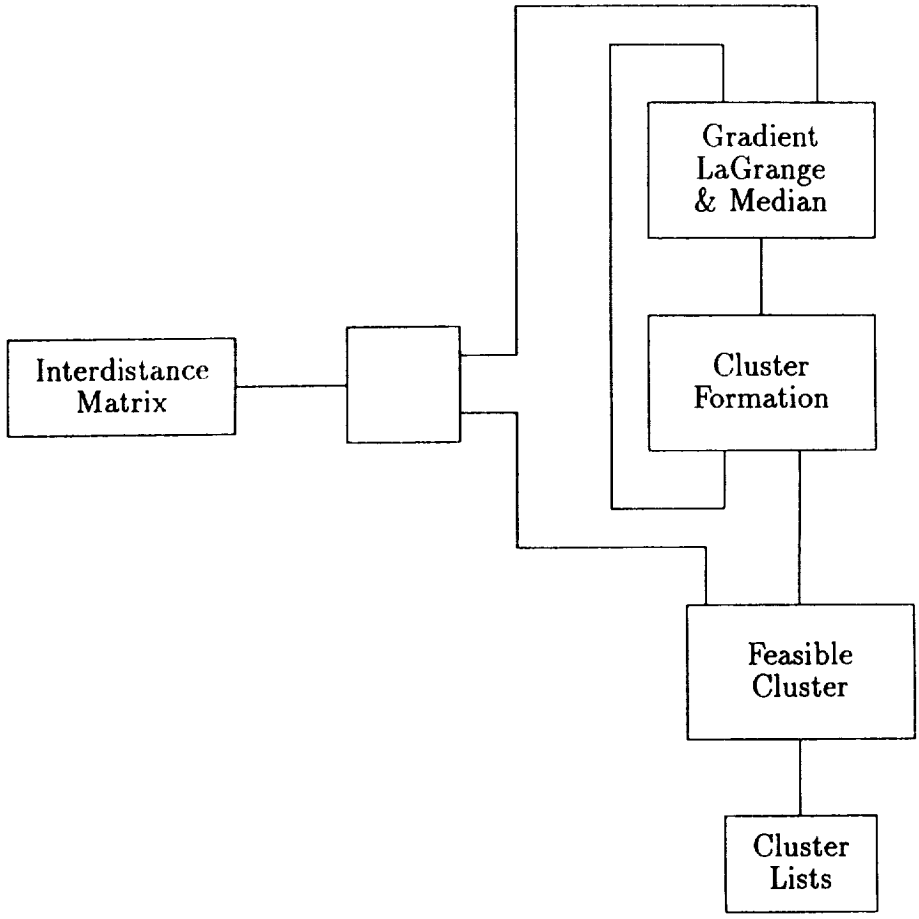


Figure 4.3. WTA/TS Target Cluster Definition ADAS Graph — Level 2

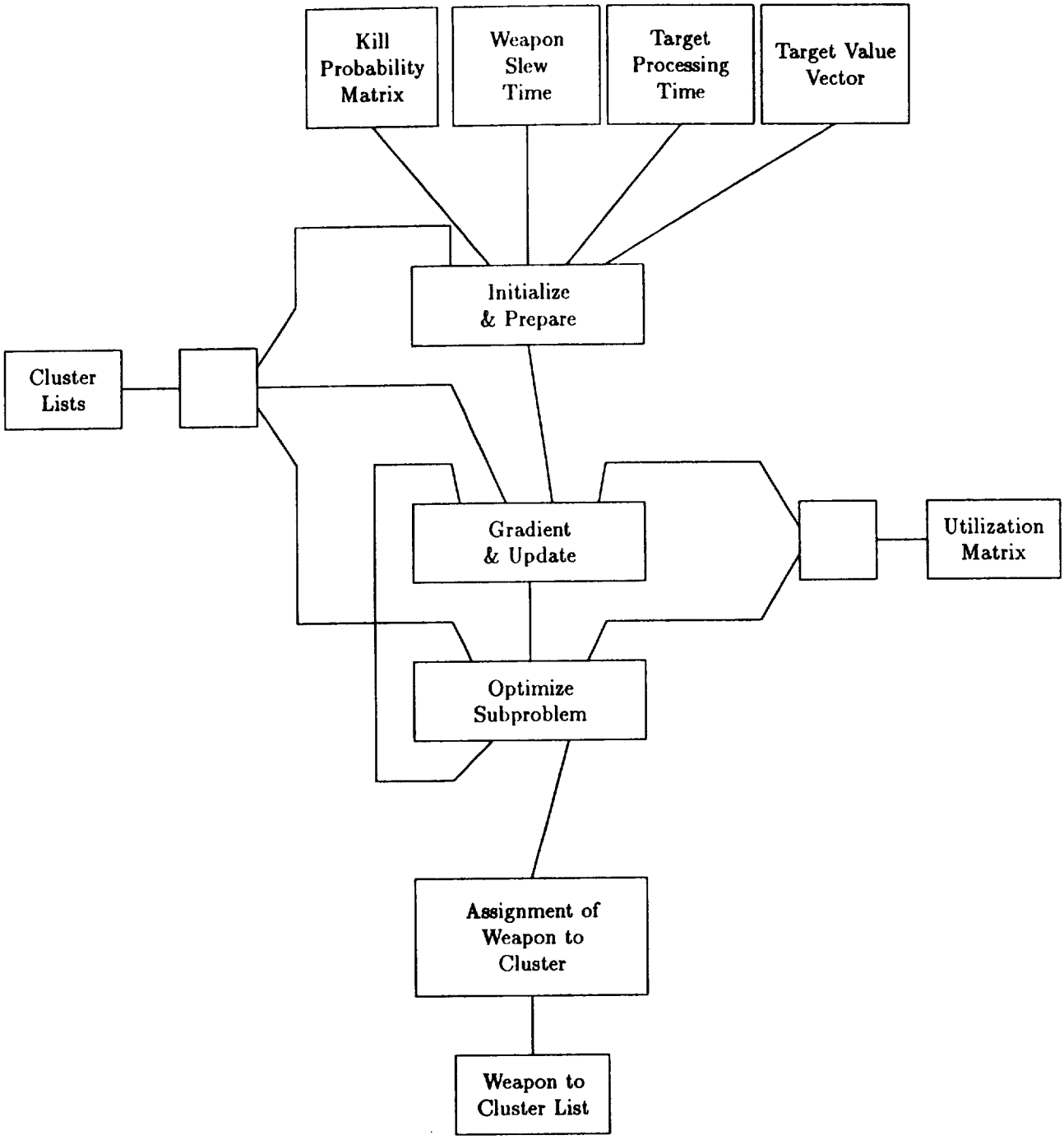


Figure 4.4. WTA/TS Weapon to Cluster Assignment ADAS Graph — Level 2

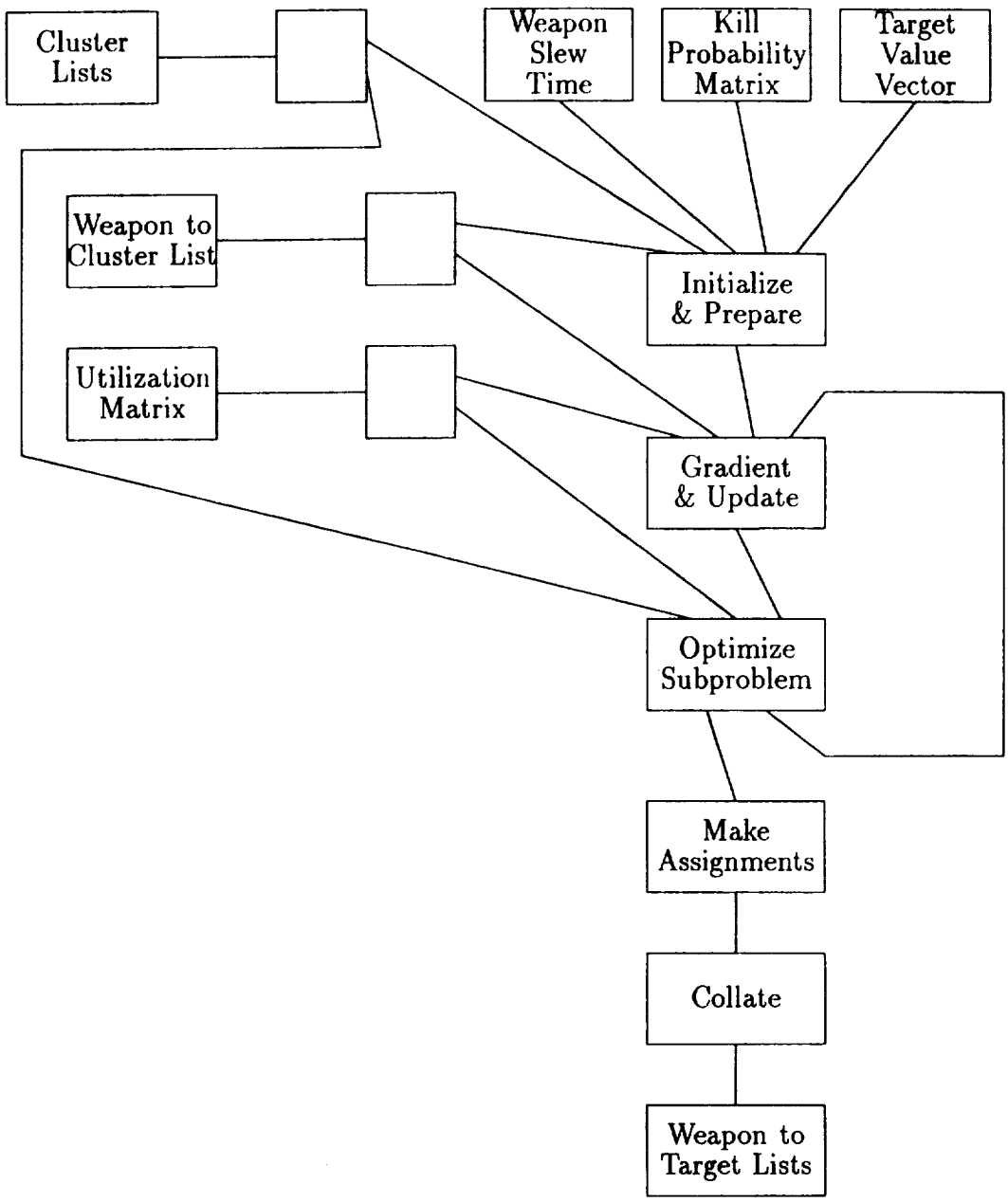


Figure 4.5. WTA/TS Weapon Assignment ADAS Graph — Level 2

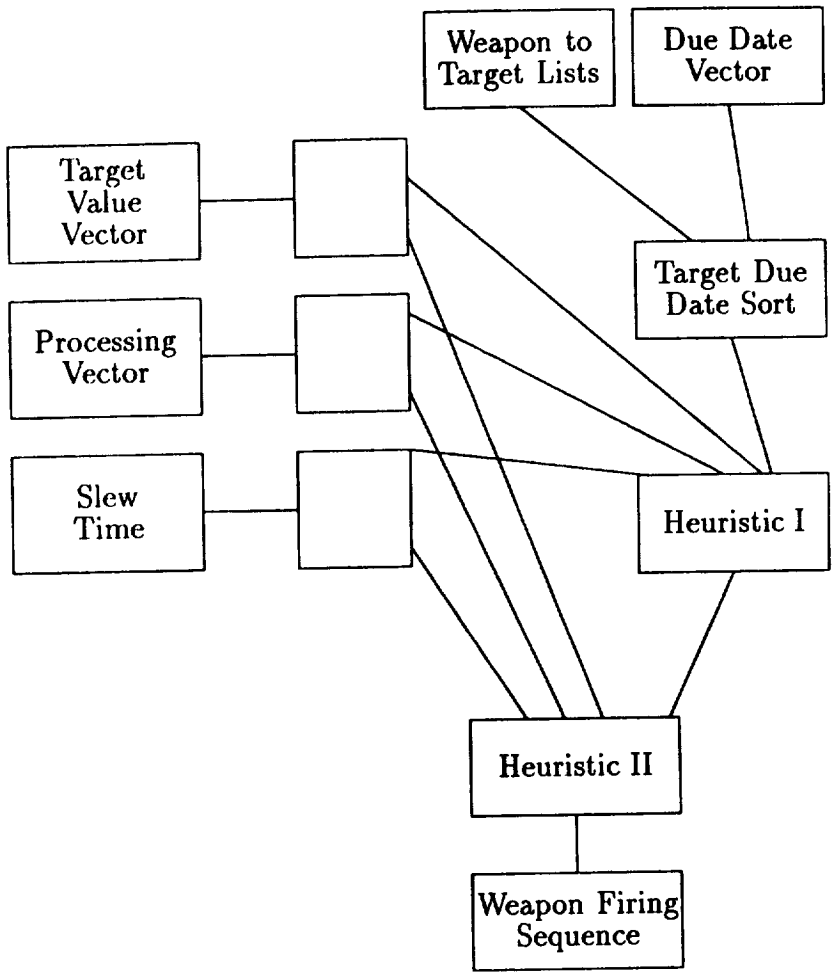


Figure 4.6. WTA/TS Target Sequencing ADAS Graph — Level 2

```
/*****************************************************************
*                                                                *
*  This is the Attribute Definition Language file for            *
*                                                                *
*  Node: GLM in graph tcd.swg                                    *
*                                                                *
*  File: tcd_glm.adl                                             *
*                                                                *
*  Description:  This ADL defines the firing delay, produce,     *
*   consume, threshold, and initial attributes for               *
*   the Gradient, LaGrange, and Median processing node           *
*   in the Target Cluster Definition graph.  This node           *
*   is a leaf node.                                              *
*                                                                *
*  Top level graph:  wtats.swg                                   *
*  Author:  C. Scheper (Revised G. Frank)                        *
*  Date:  03/10/88 (03/18/88)                                    *
*                                                                *
*****************************************************************/

graph:    -- Key Input Parameters
          nbr_targets,          -- The number of targets (N).
                                -- N needs to take on values of
                                -- 25, 50, 100, and 200.
          LIc,                  -- LaGrange Iteration Count
          Cim_size,             -- Size of Cim data structure
          dim_size,             -- Size of dim data structure
          xim_size,             -- Size of xim data structure

-- Hardware Parameters
       fpt_mult,        -- Time required for a floating point multiply
       read_fpt,        -- Time required to read a floating pt nbr
       mips;            -- Instruction processing rate

int:   op_count,        -- Count of the number of operations
       instr_count,     -- Count of the number of instructions read
       mpy_count,       -- Count of the number of floating pt mpys
       io_ops;          -- Count of the number of I/O operations.

       op_count = 8 * nbr_targets * nbr_targets;
       instr_count = 5 * op_count;
       mpy_count = nbr_targets;
       io_ops = xim_size + dim_size + Cim_size;
       firing_delay = ( instr_count/mips ) +
                      ( mpy_count / fpt_mult ) +
                      ( io_ops / read_fpt );

       -- Consumes an entire xim matrix each iteration
       token_consume_rate(in0) = xim_size;
       firing_threshold(in0) = token_consume_rate(in0);
       -- An initial xim matrix starts the loop
       initial_token_count(in0) = token_consume_rate(in0);
       -- Consumes an entire dim matrix each iteration
       token_consume_rate(in1) = dim_size / LIc;
       firing_threshold(in1) = token_consume_rate(in1);
       -- Produces an entire Cim matrix each iteration
       token_produce_rate(out0) = Cim_size;
```


Figure 4.7. Example ADL File for WTA/TS ADAS Graph

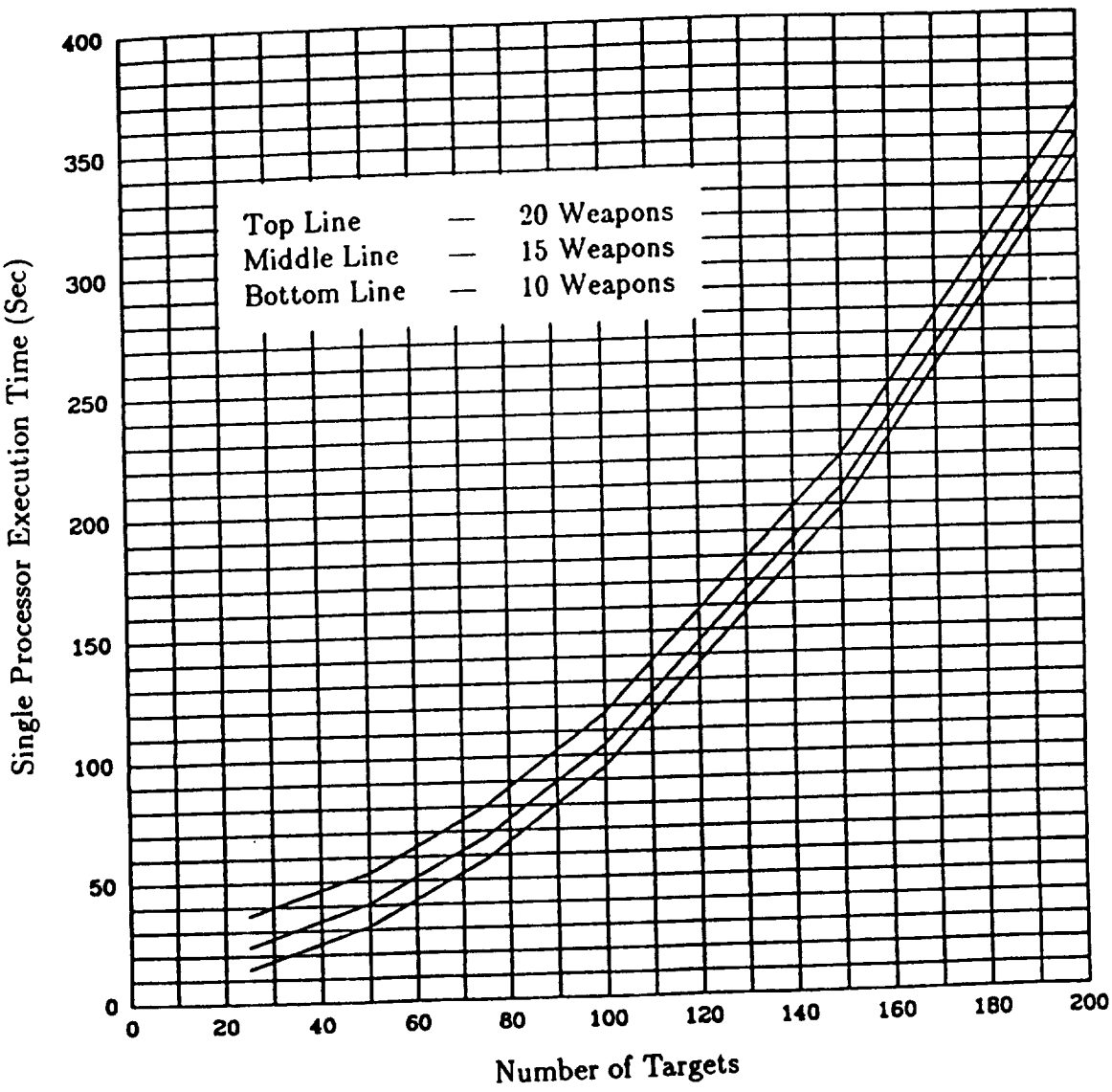


Figure 4.8. Single Processor Execution Times for WTA/TS

processors would be required. It is likely that substantially more processors would be required and the system designer may wish to set about determining a better estimate of the number of such processors required. In any case, these results based on coarse high-level information can serve as a starting point for the system design.

The next step in the WTA/TS performance modeling was to determine the distribution of workload among the various WTA/TS subfunctions. Figure 4.9 shows execution time requirements for each major function within WTA/TS as a function of number of targets and assuming a single processor. Figure 4.10 shows the percentage of the total execution time required by each major function. These results, derived from relatively high level performance modeling, are sufficient to draw the system designer's attention to potential problem areas and to help identify system characteristics.

Observe in Figure 4.10 that as the number of targets increase, the function that has the largest workload shifts from the Weapon-to-Target Cluster Assignment (WTC) function to the Target Cluster Definition (TCD) function. If this algorithm had to be decomposed and distributed among several processors, it is probable that a workload decomposition that is effective for a small number of targets would not be effective for a larger number of targets. If efficient use of resources is necessary for all numbers of targets, a designer could conclude from these results that the system would require some form of dynamic workload decomposition to cope with different numbers of targets. Alternatively, a decision to design an effective decomposition for the maximum number of targets could be supported if the resource utilization was not significant and a less complex design was desired.

Since TCD workload is dominant except for smaller numbers of targets, the distribution of workload within TCD was examined. Figure 4.11 shows the percentage distribution of workload among subfunctions within TCD. This distribution does not change with the number of targets. Further, it can be seen that Cluster Formation is the dominant subfunction. In the description of WTA/TS in [2], it can be seen that while the Gradient, LaGrange, Median subfunctions can be easily decomposed for parallel processing, the dominant subfunction, Cluster Formation, cannot be easily decomposed. Furthermore, it is embedded in a loop such that it appears as a sequential component that cannot be overlapped or pipelined with other functions. Based on this coarse design information, a potential limitation or problem area has been identified. At this point in an actual system development, it could be necessary to restructure the algorithm or to develop a special system architecture to avoid this potential problem.

Several valuable roles that high-level performance modeling can play in the early stages of system design have been demonstrated by this example.

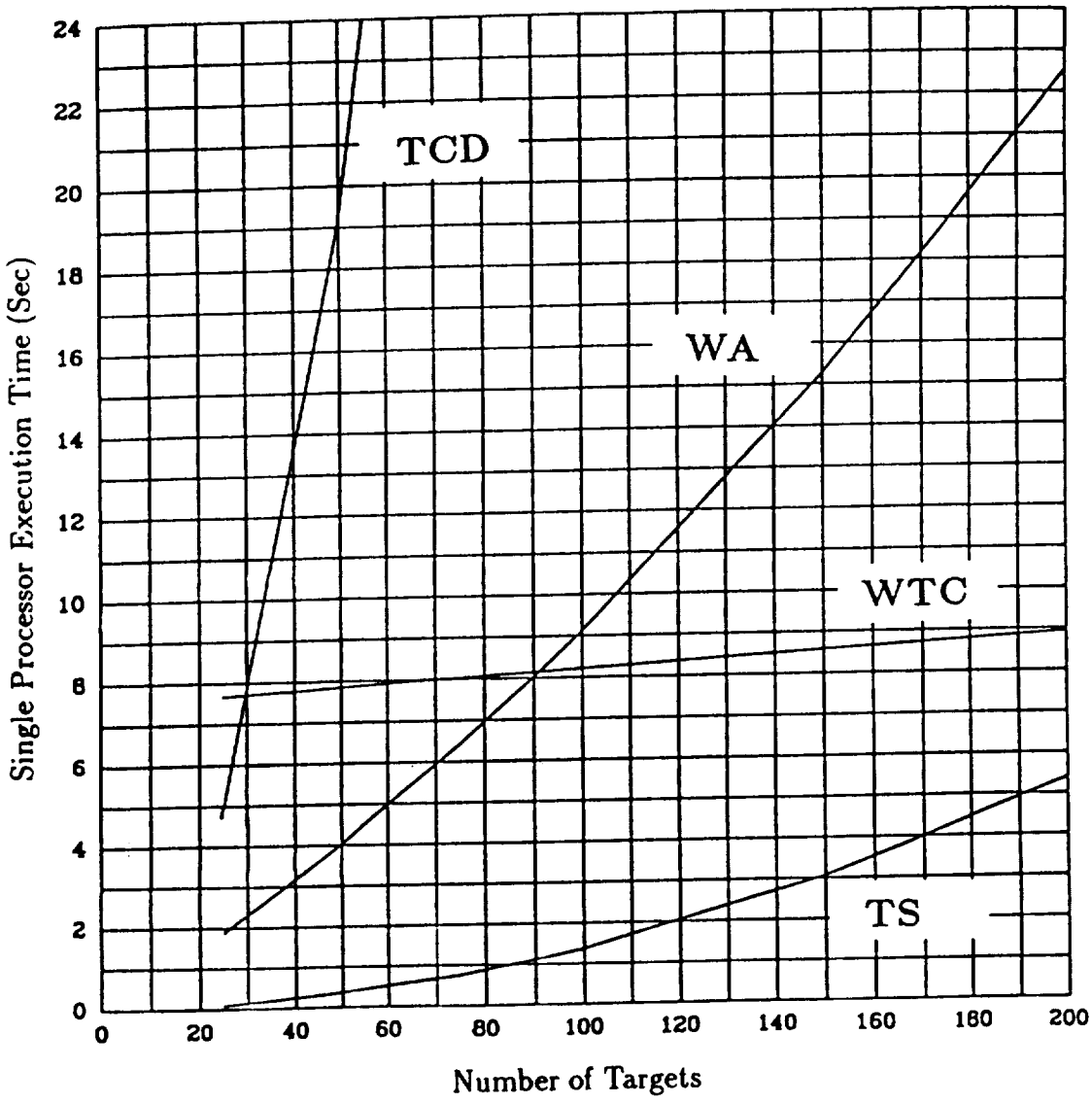


Figure 4.9. Execution Time Requirements of WTA/TS Functions

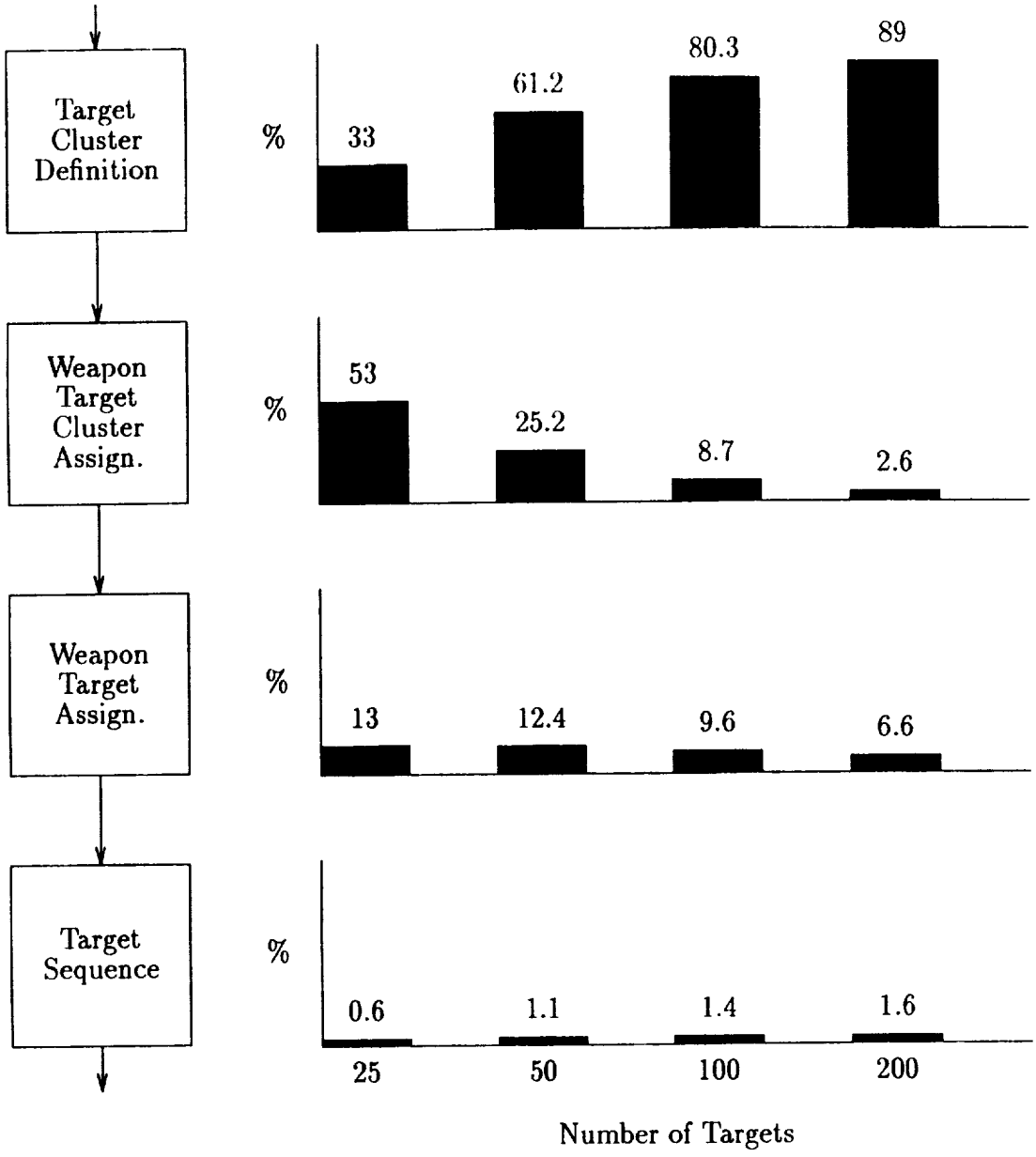


Figure 4.10. Execution Time Percentages by WTA/TS Function

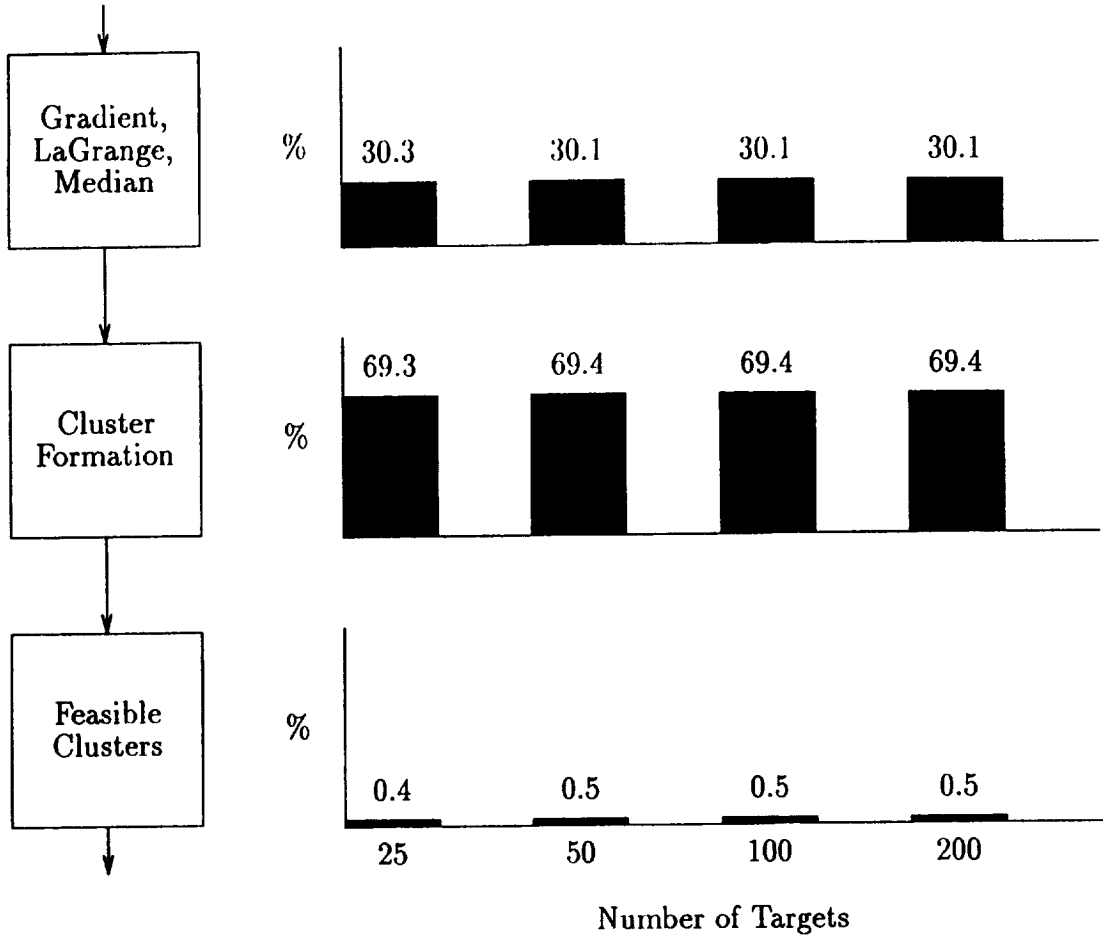


Figure 4.11. Percentage Workload Distributions for TCD Subfunctions

4.3. High-Level Decomposition of an Algorithm for FTPP

An important factor in the effectiveness of particular architectures for parallel processing is interprocessor communication. Often interprocessor communication is the dominant performance limiting factor. For topologies other than the fully connected network, interprocessor communication generally is influenced not only by communications bandwidth but also by the "distance" that communicating processor pairs are from each other. Evaluation of interprocessor communications in the decomposition of algorithms for parallel processing architectures is, therefore, a necessary component of performance modeling. A parallel decomposition of the Gradient, LaGrange and Median (GLM) portion of the WTA/TS algorithm was used to conduct interprocessor communications performance analyses for several configurations of an FTPP.

The ADAS software model is shown in Figure 4.12. In this model, the Gradient Lagrange Median function (GLM) is decomposed into four parallel parts which can be assigned to four different processors. A diagram of an FTPP configuration is shown in Figure 4.13. In this graph, four FTPP clusters are linearly connected by dual links between clusters. Figure 4.14 shows the steps involved in FTPP data transfers between processors four clusters apart in the linearly connected network. The vote and reflect steps provide fault tolerance. Each of the four parallel parts of GLM were assigned to a processor within the FTPP. In one case, the assigned processors were within a single cluster. In another, two processors were used from each of two clusters. Finally, a single processor was used from each of the four clusters. Figure 4.15 shows the execution time for GLM for each of the three cases. The portion of the communications delay due to fault tolerance was removed and the results are also shown in Figure 4.15. Figure 4.16 shows the percentage of processing time due to fault tolerance for each of the three cases.

The processor speed was varied for each of the cases discussed above. Figure 4.17 shows the ratio of execution time for four processors compared to a single processor of the same speed. As can be seen, the single processor execution time in this case is less than the multiple processor execution time for processor speeds above 1 MIP. These results show the impact of communication overhead and its increase as the distance between processors increases.

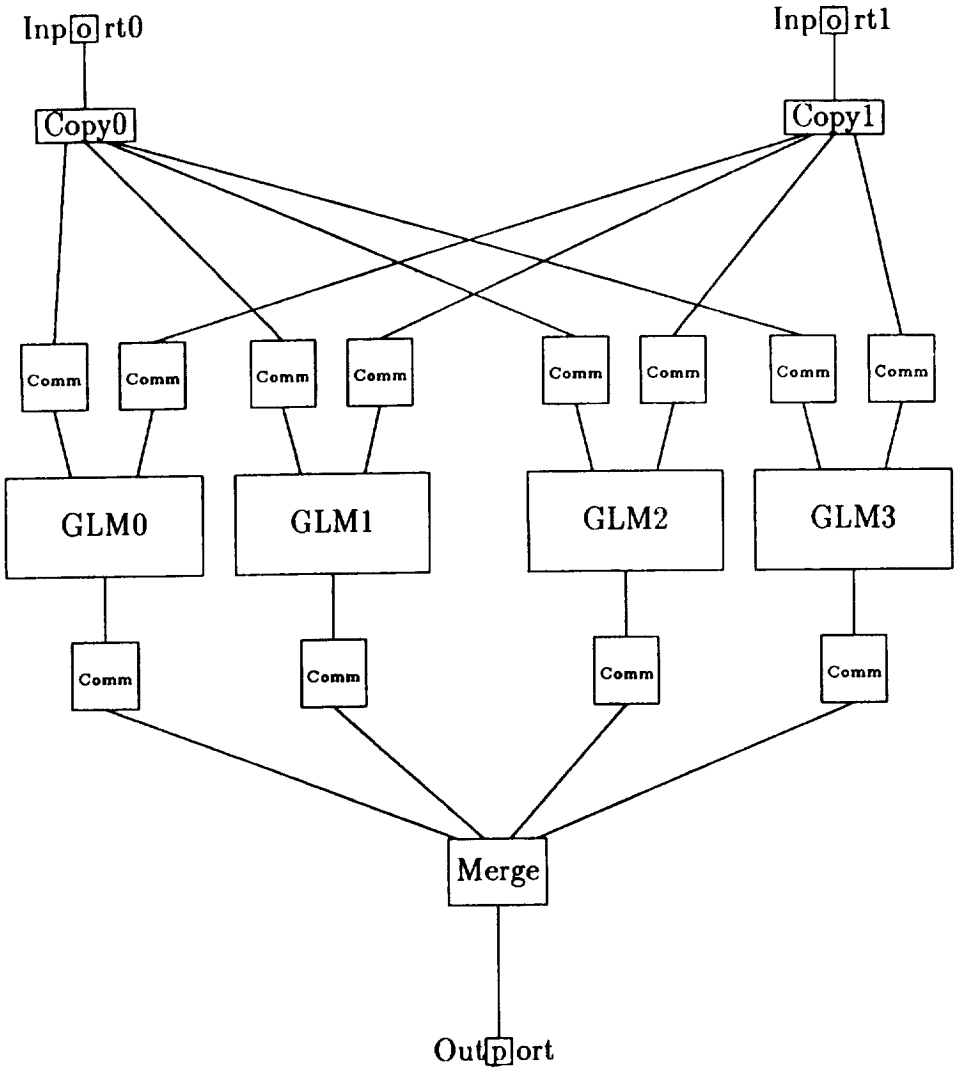


Figure 4.12. ADAS Model of Parallel Gradient Lagrange Median Portion of WTA/TS

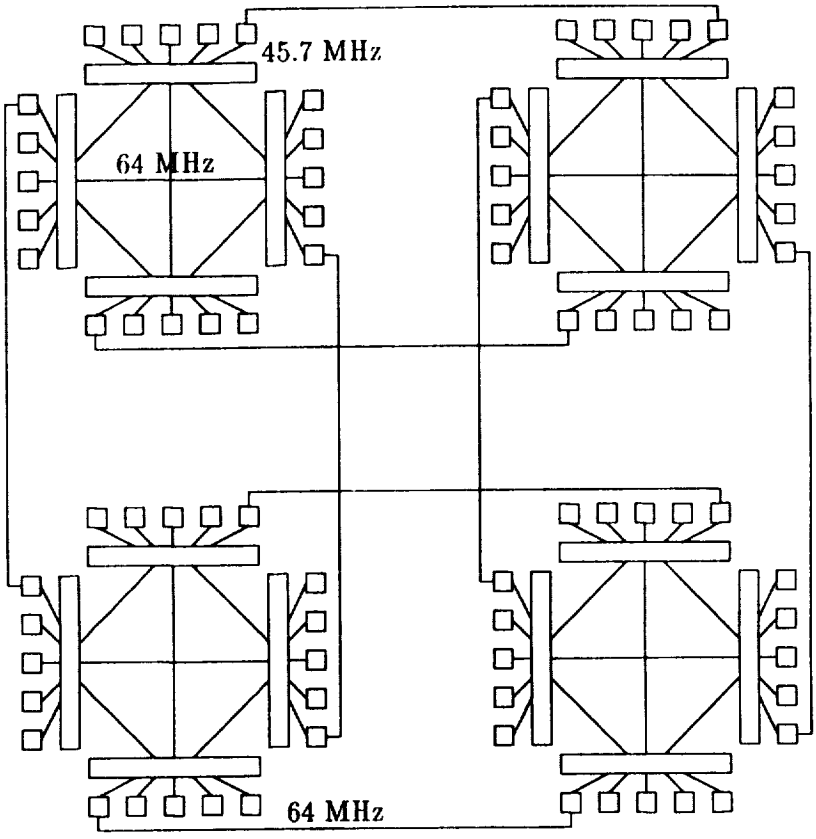


Figure 4.13. Diagram of a Linearly Connected FTPP Configuration

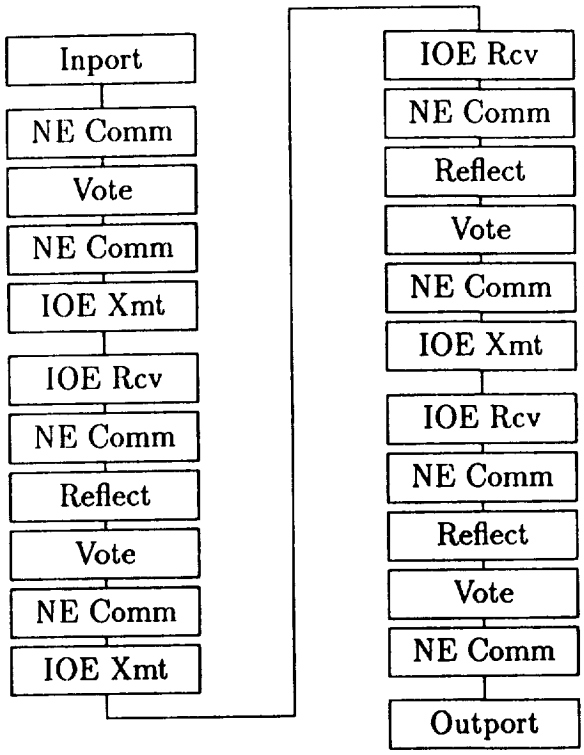


Figure 4.14. FTPP Data Transfer Steps for Four Cluster Distant Processor Pairs

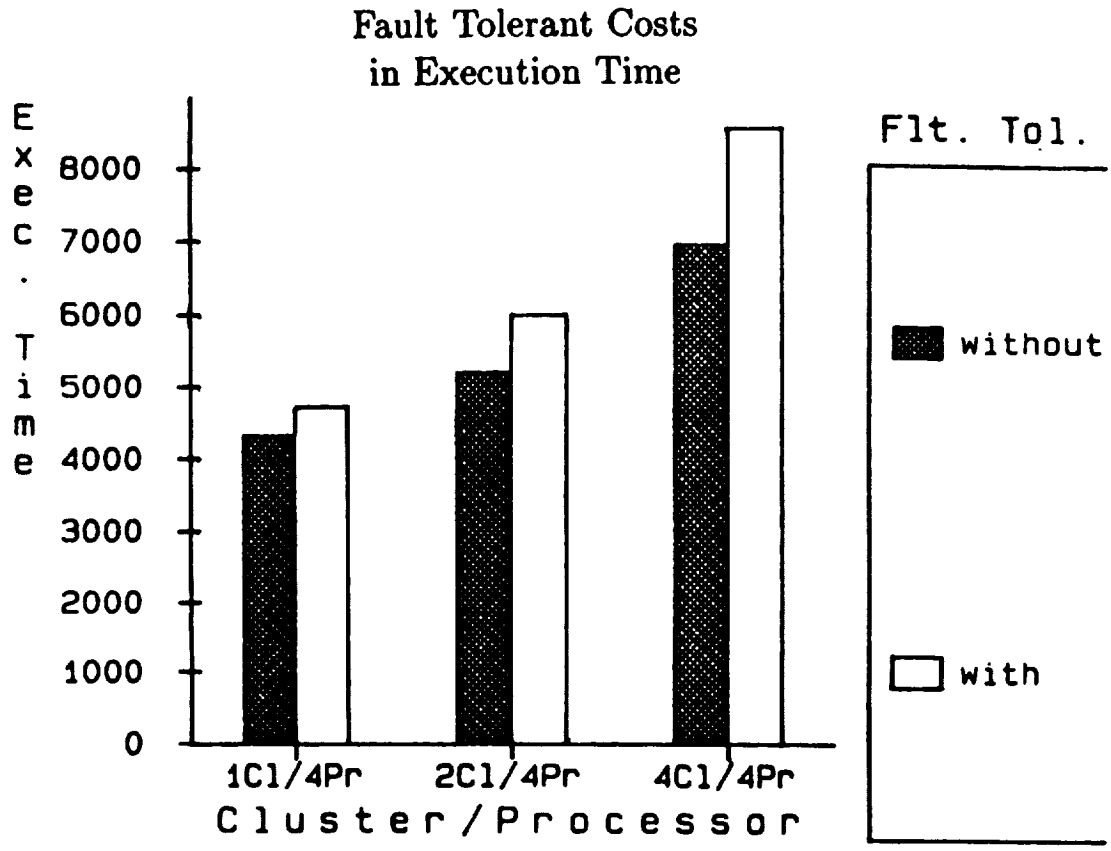


Figure 4.15. Parallel Gradient Lagrange Median Execution Times

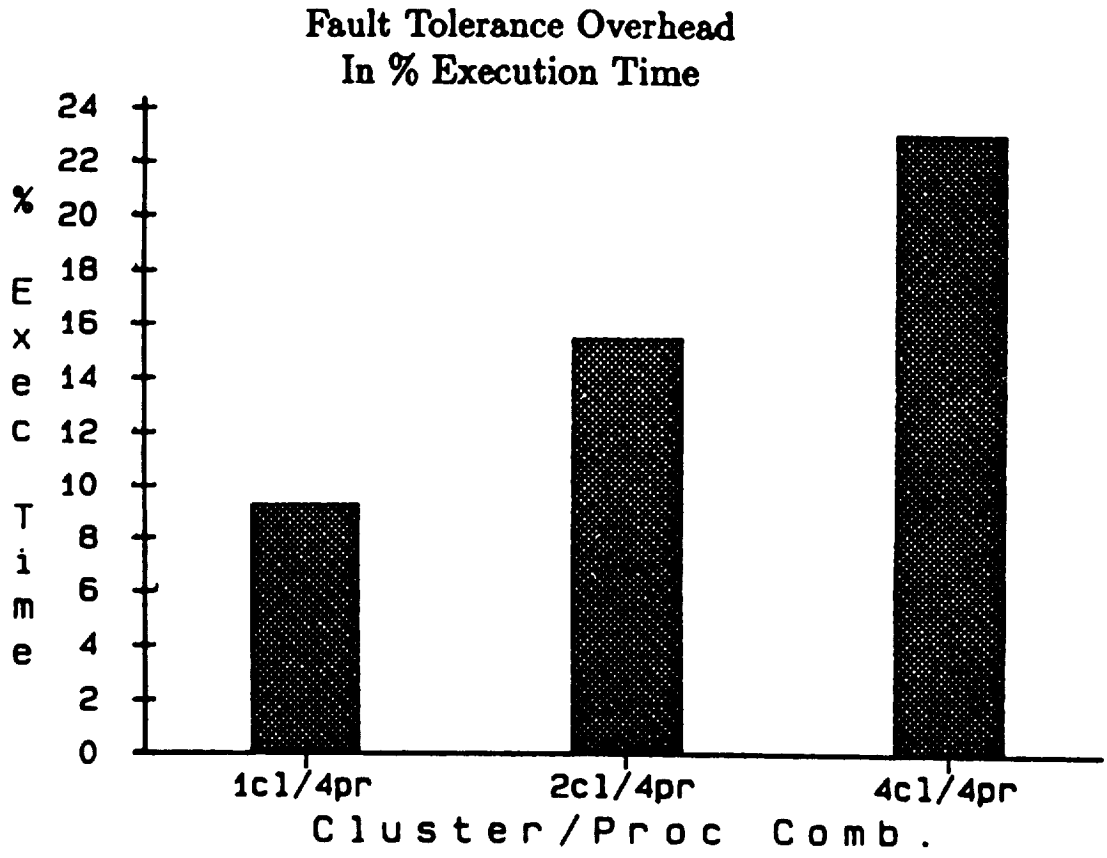


Figure 4.16. Fault-Tolerant Communications Time Percentage of Processing Time

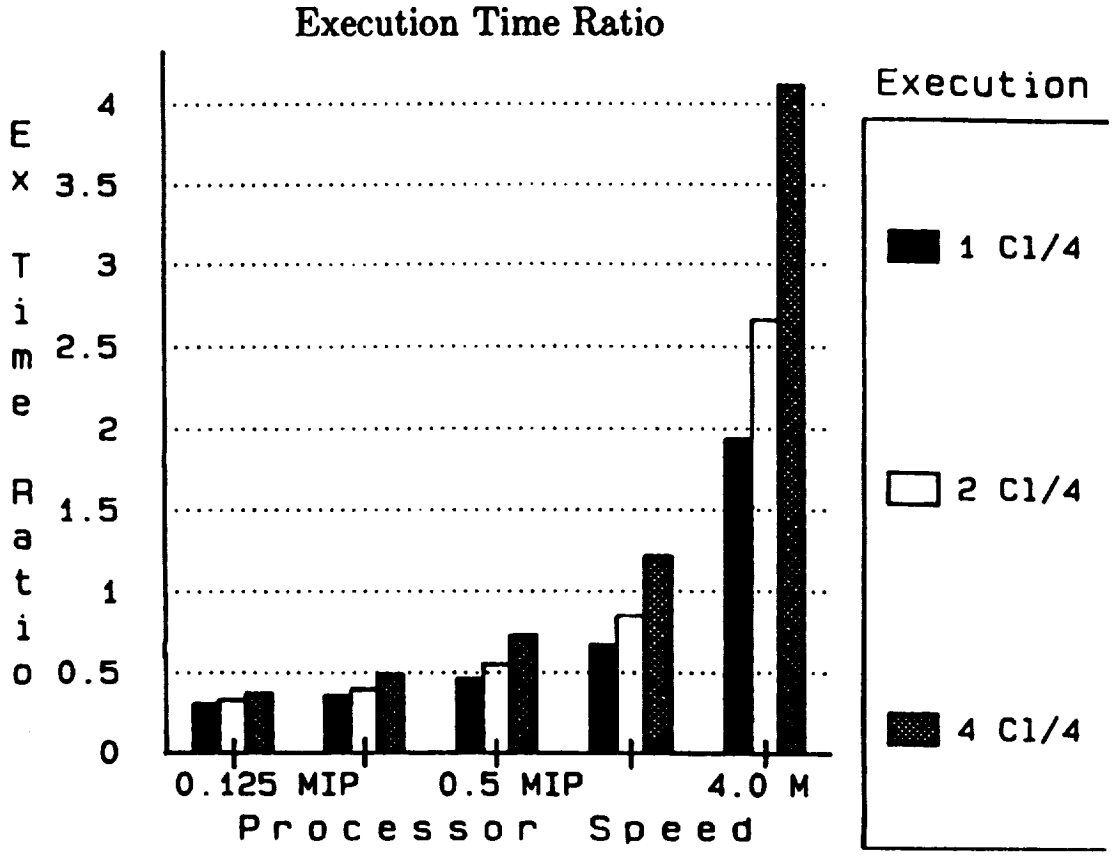


Figure 4.17. Execution Time Ratios for Parallel versus Single Processor as Function of Processor Speed

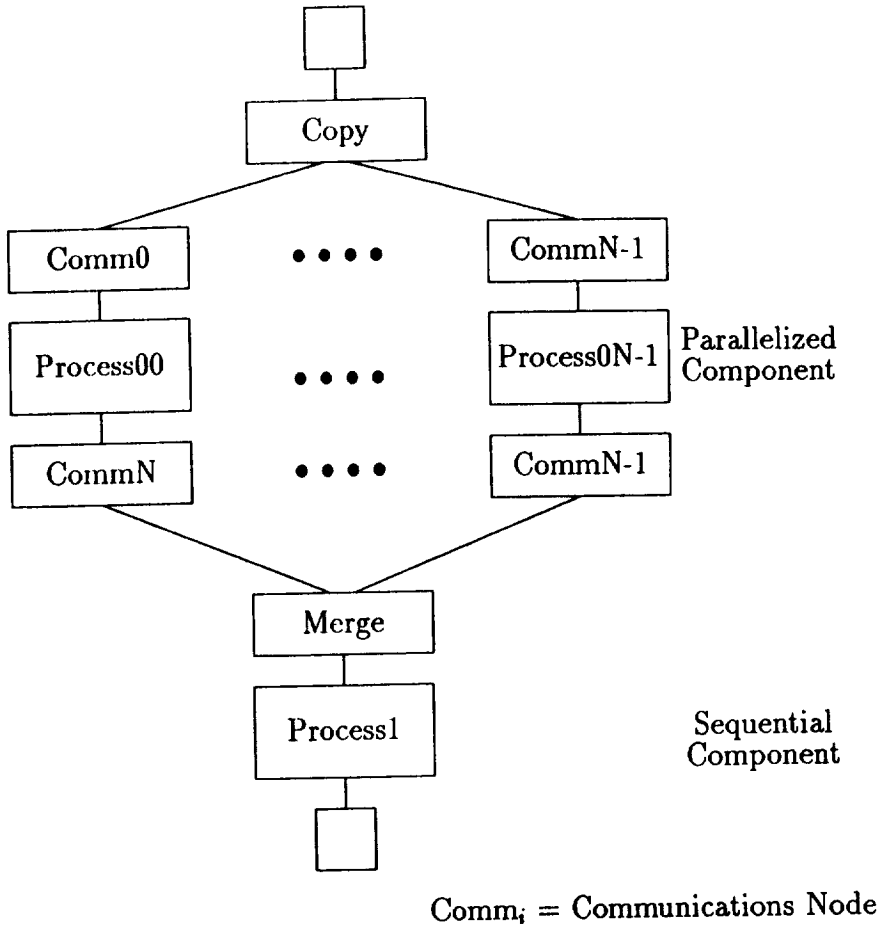


Figure 4.18. Generic Process Structure

4.4. High-Level Generic Parallel Performance Models

Based on the results of the performance modeling described in the previous sections, it was decided to develop a high-level model to assess the effectiveness of algorithm decomposition and embedding strategies for a given architecture. The process structure of interest, whose occurrence was observed frequently in the modeling described above, is that given in Figure 4.18. In this ADAS graph, a process that can be decomposed into parallel tasks is followed by a task that cannot be decomposed.

The process or algorithm structure was characterized by the processor workload, com-

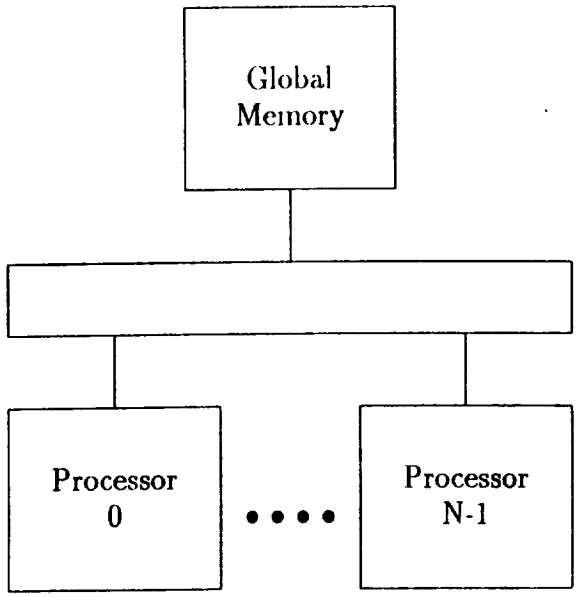
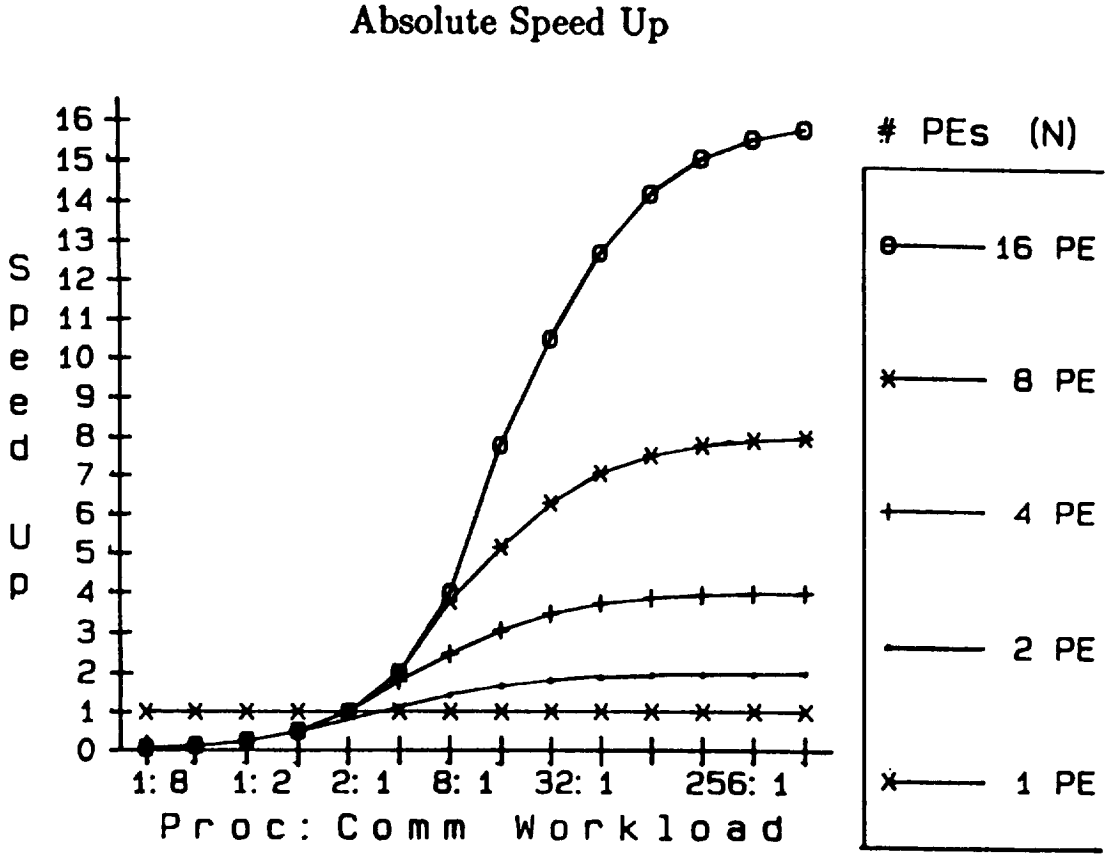


Figure 4.19. Architecture for Generic Parallel Performance Model

munications workload, rules for distributing workloads when the process is decomposed for parallel processing, and the workload distribution between decomposable and sequential tasks. The architecture shown in Figure 4.19 could be characterized at the higher level by the number of processors, processor speed and overhead, communications speed and overhead, memory speed, and communication distance between processors.

Experiments were conducted for the two categories of sequential and no sequential component in the algorithm. For the no sequential component case, simulations were performed to measure the effects of increased processor workload and increased communication workload. By varying the ratio of processor workload to communications workload for a given number of processors, the effectiveness of decomposition for different algorithms can be examined for a given architecture. By varying the ratio of processor speed to communications speed for a given number of processors, the effectiveness of different architectures can be examined for a given algorithm. Figure 4.20 shows the speedup for different numbers of processors as a function of the processor to communications workload ratios.

Figure 4.21 shows the same information normalized by the number of processors being used in the configuration.



R is ratio of sequential workload to parallel workload

Figure 4.20. Speedup as Function of Processing to Communication Workload Ratios

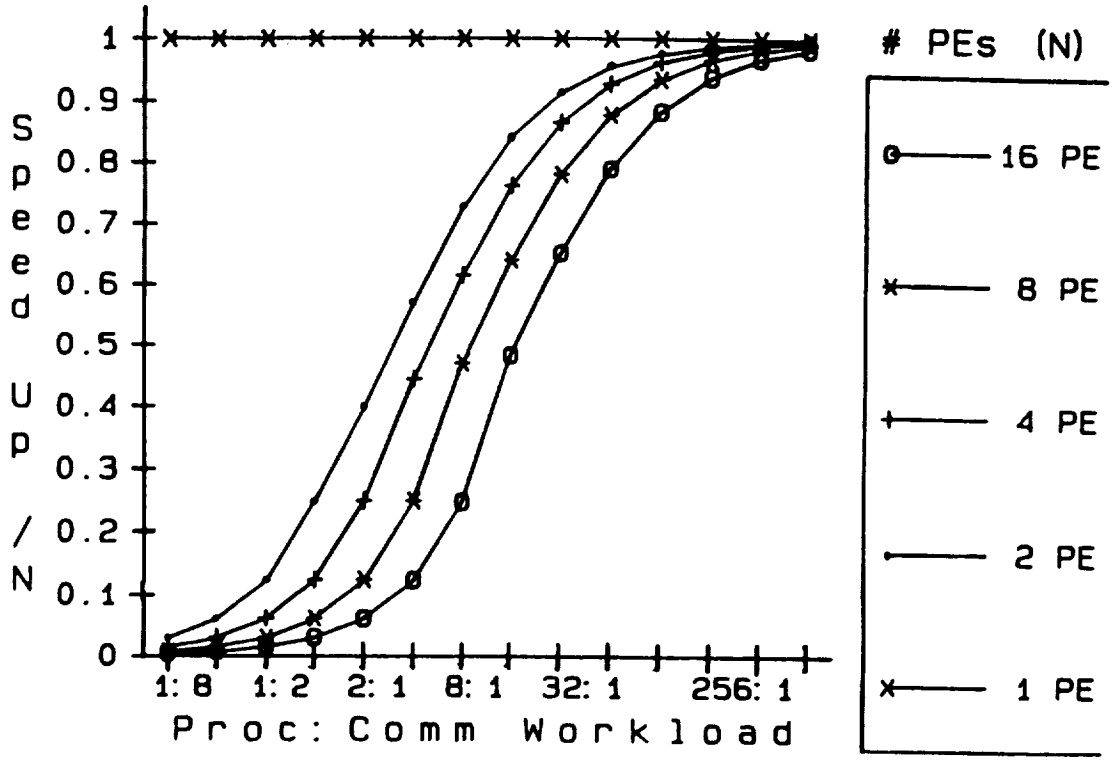


Figure 4.21. Normalized Speedup

This information would provide guidance as to the effectiveness of a given architecture for different algorithms. Conclusions could be drawn regarding which algorithm could be executed more rapidly on a given architecture. Note that for this case processes whose communications workload is at least one half of the processing workload result in no speedup over a single processor.

The required processor-to-communication speed ratio to attain a given speedup for a particular algorithm can be determined from the model.

For the sequential component case, simulations were performed for increased processor workload-to-communication workload ratio and increased sequential component to parallel component ratio. Given that a process consists of two tasks that require T_1 and T_2 execution times, respectively, the total process time for the sequential execution of the two tasks is $T = T_1 + T_2$. If task_1 can be parallelized across N processors, $T = T_1/N + T_2$. If task_2 is non-parallelizable (sequential)

$$\begin{aligned} \text{T(N)} &= T_1/N + T_2 \\ &= T_1 \cdot (1/N + T_2/T_1) \\ &= T_1 \cdot (1/N + R) \end{aligned}$$

where $R = T_2/T_1$ is the sequential-to-parallel ratio of the process. Then $T(1) = T_1 \cdot (1 + R)$ and

$$\begin{aligned} \text{Speedup} &= \frac{T(1)}{T(N)} \\ &= \frac{T_1 \cdot (1+R)}{T_1 \cdot (1/N+R)} \\ &= \frac{N \cdot (R+1)}{R \cdot N+1} \end{aligned}$$

This relationship is known as Amdahl's Law. Figure 4.22 shows the maximum speedup that can be obtained as a function of the ratio of sequential workload to parallel workload. Figures 4.23 and 4.24 show the speedup obtained as the algorithms workload ratios are varied when the sequential component workload is two times the parallel workload.

It is believed that an enhanced model of the type described above would be a valuable tool for the initial high-level design phases in the development of parallel computing applications.

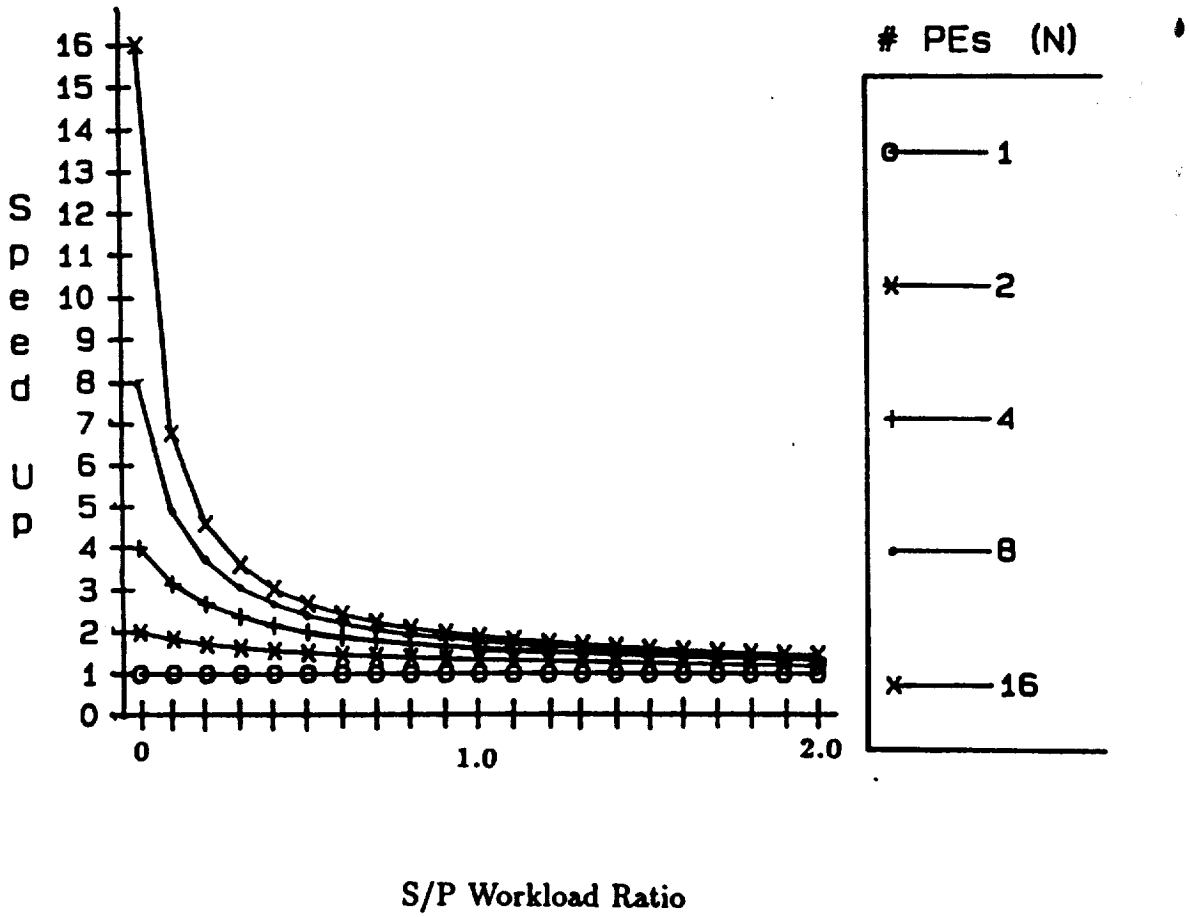
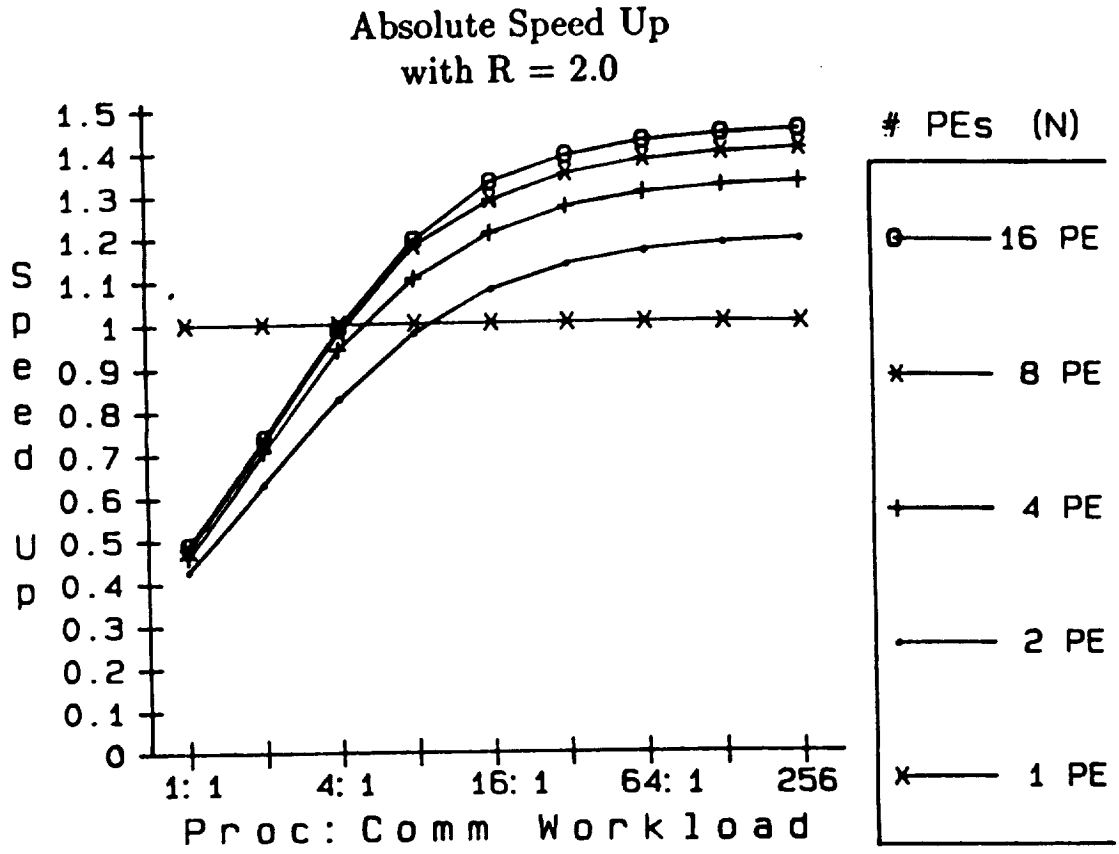
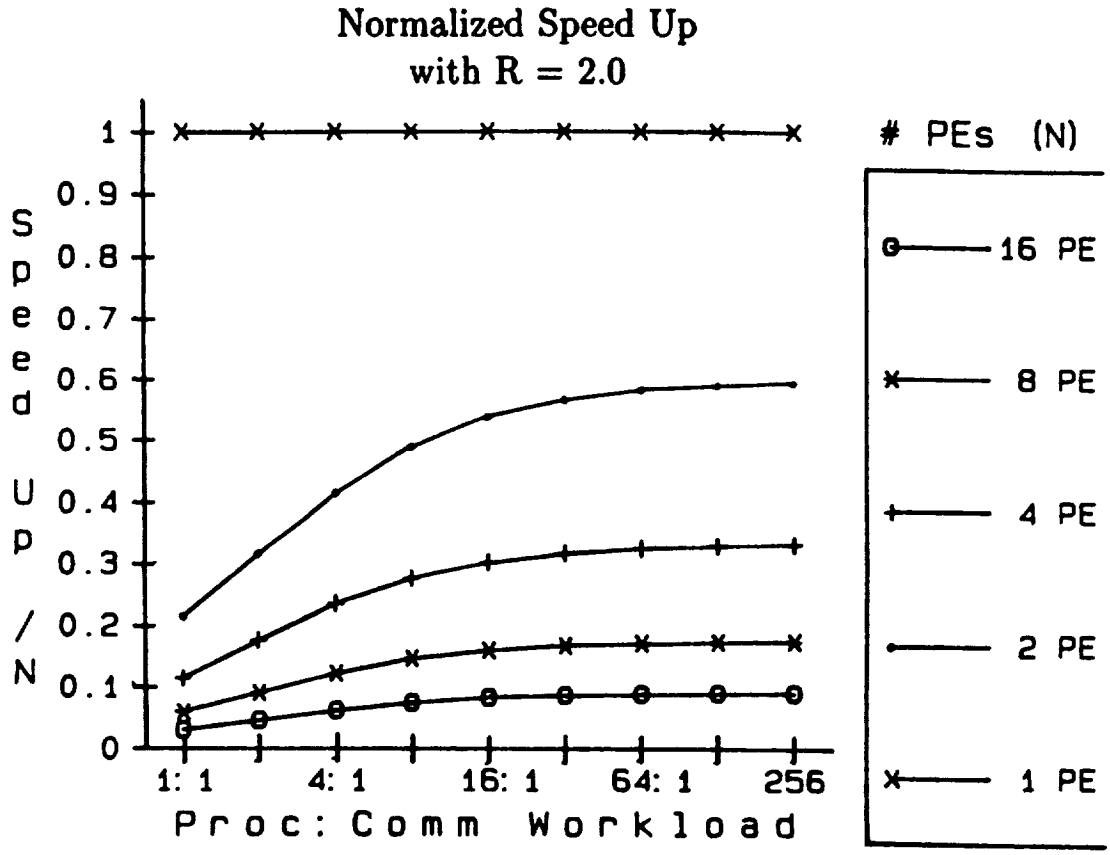


Figure 4.22. Maximum Speedup as Function of Sequential-to-Parallel Workload Ratio



R is ratio of sequential workload to parallel workload

Figure 4.23. Speedup as a Function of Processing-to-Communication Workload Ratio for Sequential-to-Parallel Workload Ratio = 2



R is ratio of sequential workload to parallel workload

Figure 4.24. Normalized Speedup as Workload Varies

4.5. WAUCTION_ASSIGNMENT High-Level Workload Assessment

The WAUCTION_ASSIGNMENT algorithm is an example of a mission planning algorithm used for determining optimal resource allocation. In the performance analysis conducted for WAUCTION_ASSIGNMENT, workload and workload distribution was assessed. The methods and modeling process used were similar to those described for WTA/TS. However, the modeling effort differed from that used for WTA/TS in several ways.

As discussed in Section 3.1, the algorithm description was developed using an engineering software implementation of the algorithm. The operation counts which were used in the performance model were derived in much the same manner as was used for WTA/TS, with the exception that the WAUCTION_ASSIGNMENT algorithm was known in detail, whereas assumptions had to be made regarding the techniques used to solve the integer programming problems handled by the WTA/TS algorithm. In this respect, a much higher fidelity description of the algorithm was produced.

For the WTA/TS performance modeling, assumptions were made regarding the number of iterations required to attain optimal assignments. For WAUCTION_ASSIGNMENT, the number of iterations for all data dependent loops in the implementation were measured by instrumenting and executing the engineering software. As a result, the models derived for WAUCTION_ASSIGNMENT should be more accurate. Use of the engineering software to attain parameters for the performance modeling introduces a role for measurement in the performance modeling methodology.

Figures 4.25 – 4.27 show the ADAS graphs that were used to determine workload for WAUCTION_ASSIGNMENT. The WAUCTION_ASSIGNMENT model contained thirteen parameters whose values represent the number of iterations required for thirteen iterative processes. As indicated previously, these parameters were measured by executing the engineering software for a range of targets and weapons. Average values were used for most of the model parameters. Linear dependence upon the number of targets or the number of weapons was used where appropriate.

Figure 4.28 shows the predicted workload as a function of the number of targets for different numbers of weapons. Figures 4.29 – 4.34 show the distribution of workload for each WAUCTION_ASSIGNMENT function as the number of targets is varied. These results indicate that Make Permanent Assignment (MPA) and Linear Loop/Insert Bid dominate the workload and increase with the number of weapons. Figure 4.35

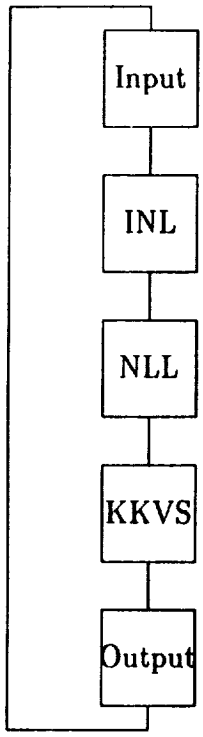


Figure 4.25. Top Level WAUCTION_ASSIGNMENT ADAS Graph

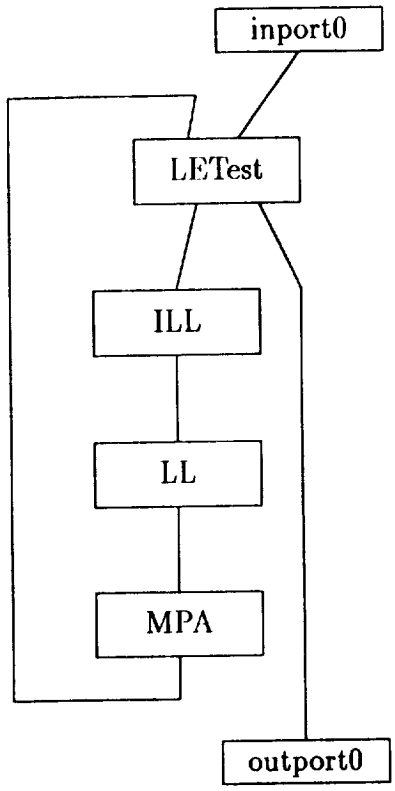


Figure 4.26. WAUCTION_ASSIGNMENT Non-Linear Loop ADAS Graph — Level 2

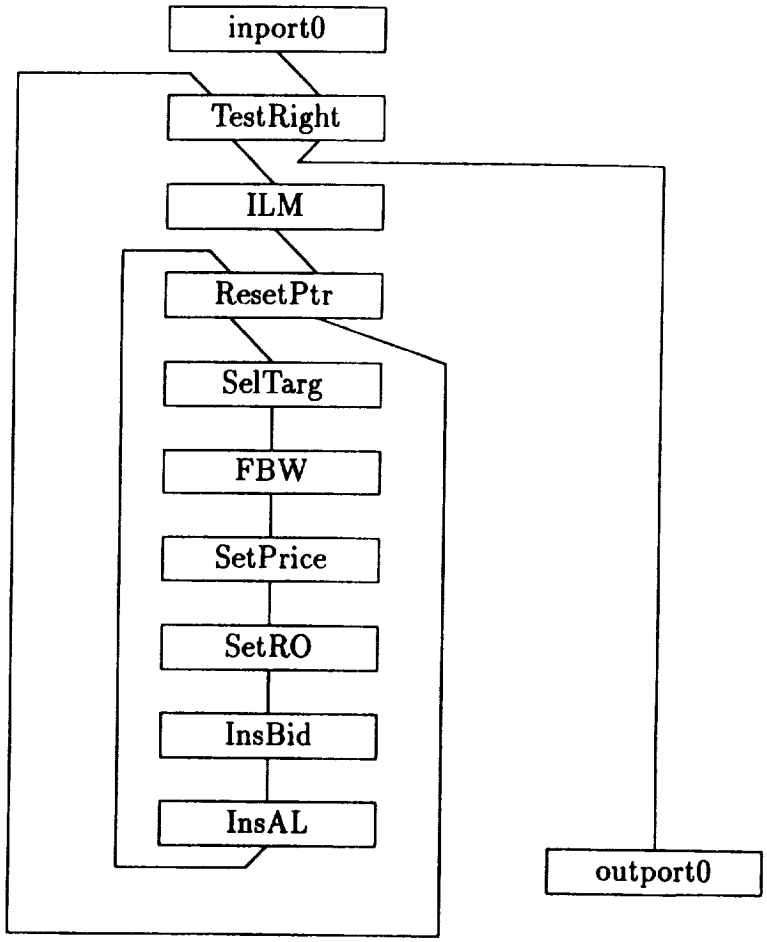


Figure 4.27. WAUCTION_ASSIGNMENT Linear Loop ADAS Graph — Level 2

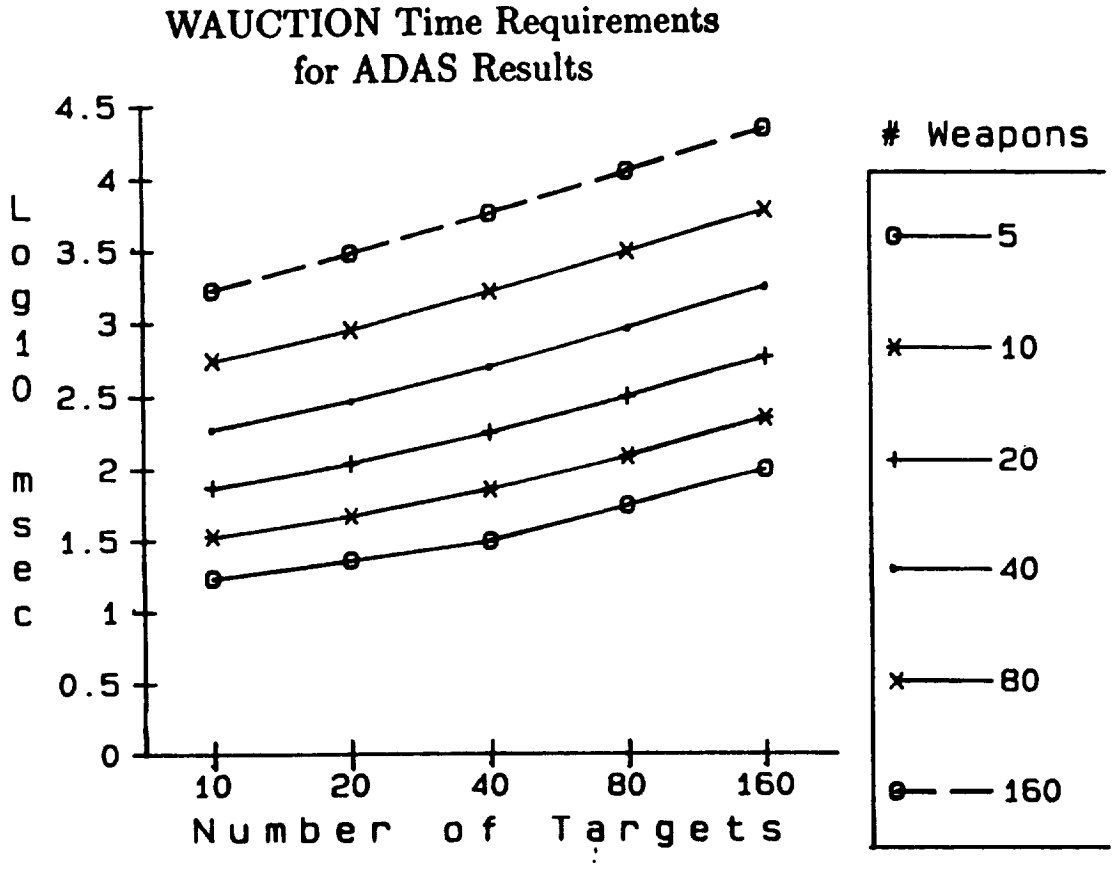


Figure 4.28. Predicted Workload for WAUCTION_ASSIGNMENT as Function of Targets and Weapons

shows the first difference in the workload as the number of targets is varied. Such results are useful to determine the sensitivity of the workload to changes in algorithm parameters. All workload/execution times for these results are based strictly upon the operation counts with no expansion factor included for implementation.

Since the engineering software was available, the CPU time required for the program to execute was obtained for various combinations of targets and weapons. Figure 4.36 shows the measured CPU time for WAUCTION_ASSIGNMENT under varied parameters. Note that these curves are not as smooth as those shown in Figure 4.28 which were produced by the performance model. However, they increase in a similar manner. The departure in shape can be explained by examining the actual behavior of the loop iteration counts measured using the engineering software. The loop iteration counts for the three major nested loops are shown in Figures 4.37 and 4.39. Note that the iterations for the target/weapon pairing loop, the inner most nested loop, is not ordered by number of weapons. Note also that for weapons = 20 and targets = 160 the count is extremely high. This large jump is reflected in the measured execution time shown in Figure 4.36. Note that the measured linear loop iteration count has a maximum, or peak, at the point where the number of targets is equal to five times the number of weapons. Since the number of shots allowed for each weapon is five for this example, the peak occurs at the point where the number of targets is equal to the total number of shots available. Recall that only average values and simple linear functions were used to model these iteration counts.

The main observation to be made regarding these measured results is that unlike algorithms typically used in signal/image processing, the workload of the WAUCTION_ASSIGNMENT algorithm cannot be characterized by simple functions of complexity parameters such as N, N^2, or logN. The behavior is much more complex. To characterize this behavior, measurements have to be made and the results incorporated into the model to provide more accurate results. Consequently, the role for measurement in performance modeling can be seen to be important for this algorithm. Two roles are apparent: that of determining parameters to be used in the model and that of validating overall modeling results.

In comparing the measured results to the performance modeling results given in Figures 4.28 and 4.36, note that no mention of the absolute scale for the two families of curves was made. Since the model used primitive operation counts to determine results, the overall difference in the measured versus the model results can be attributed to an implementation expansion factor. To get the two families of curves to overlap as shown in Figure 4.40, an implementation factor of 10 was used. It is not claimed that the factor of 10 should be used for all modeling. Rather, the significance of this result

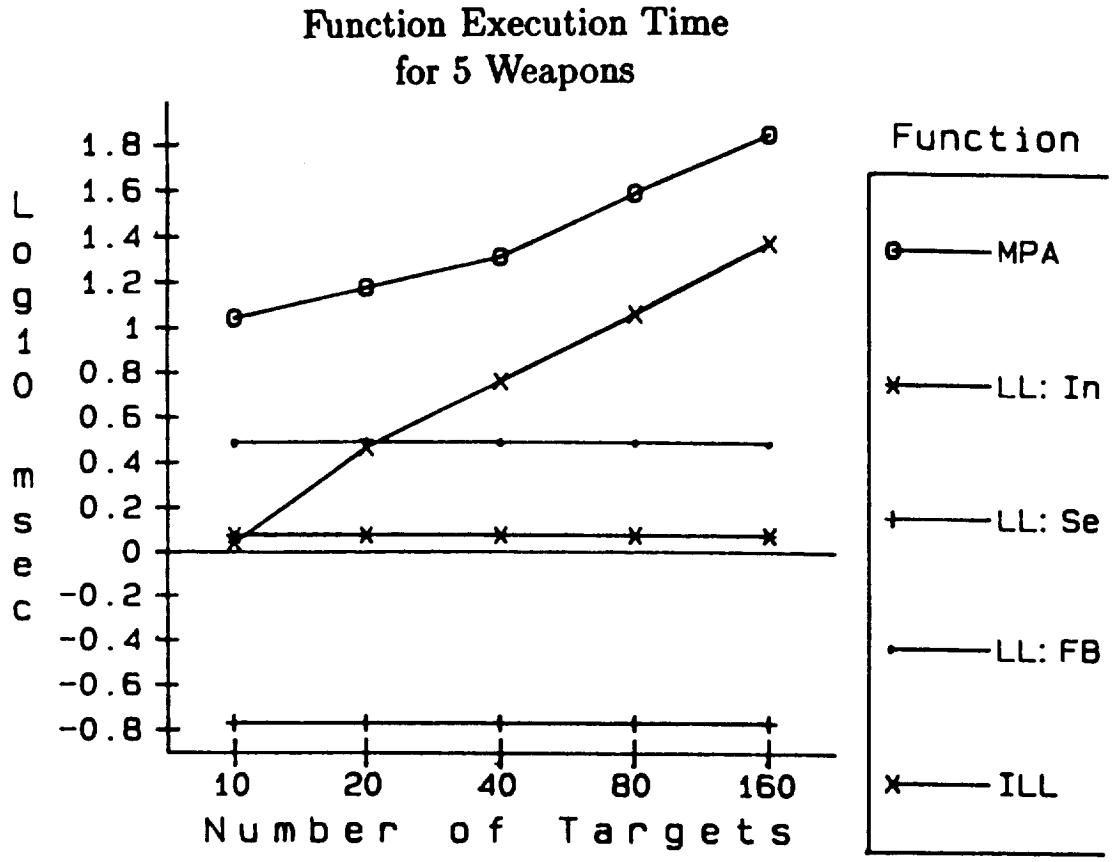


Figure 4.29. WAUCTION_ASSIGNMENT Workload Distribution by Function for Weapons = 5

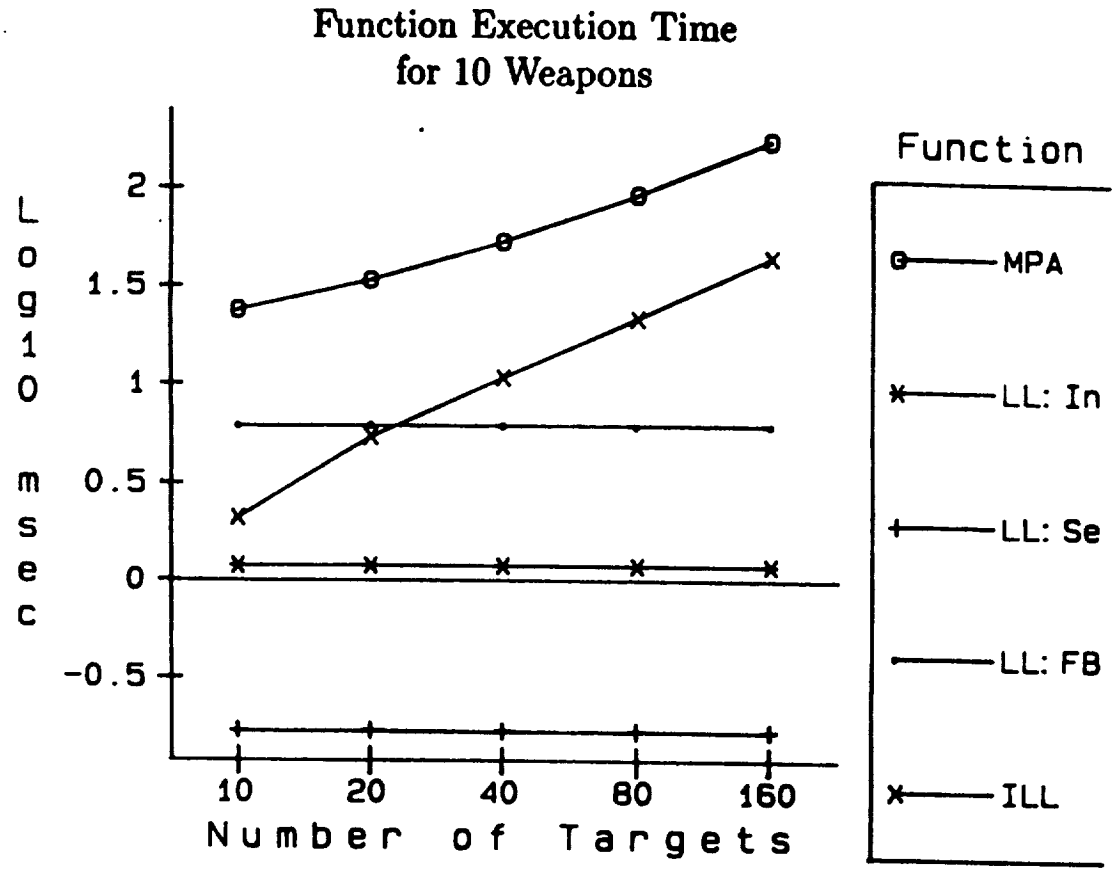


Figure 4.30. WAUCTION_ASSIGNMENT Workload Distribution by Function for Weapons = 10

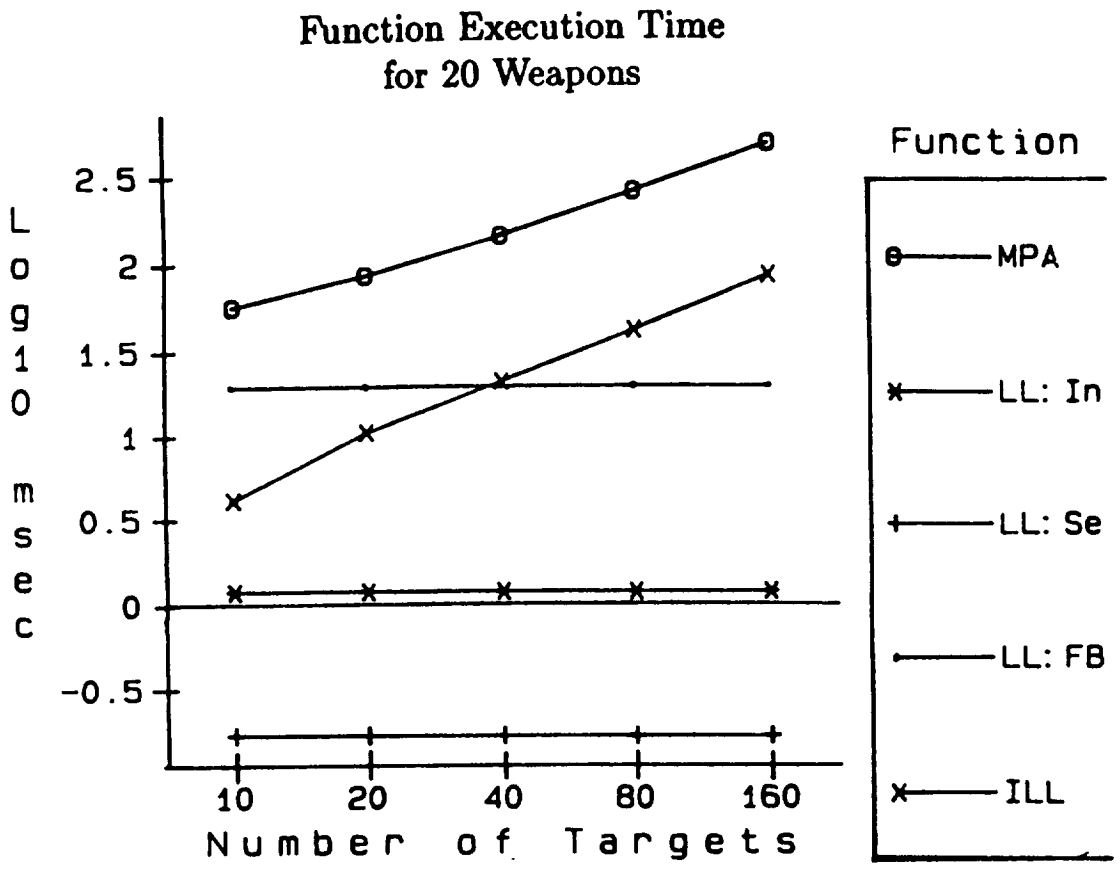


Figure 4.31. WAUCTION_ASSIGNMENT Workload Distribution by Function for Weapons = 20

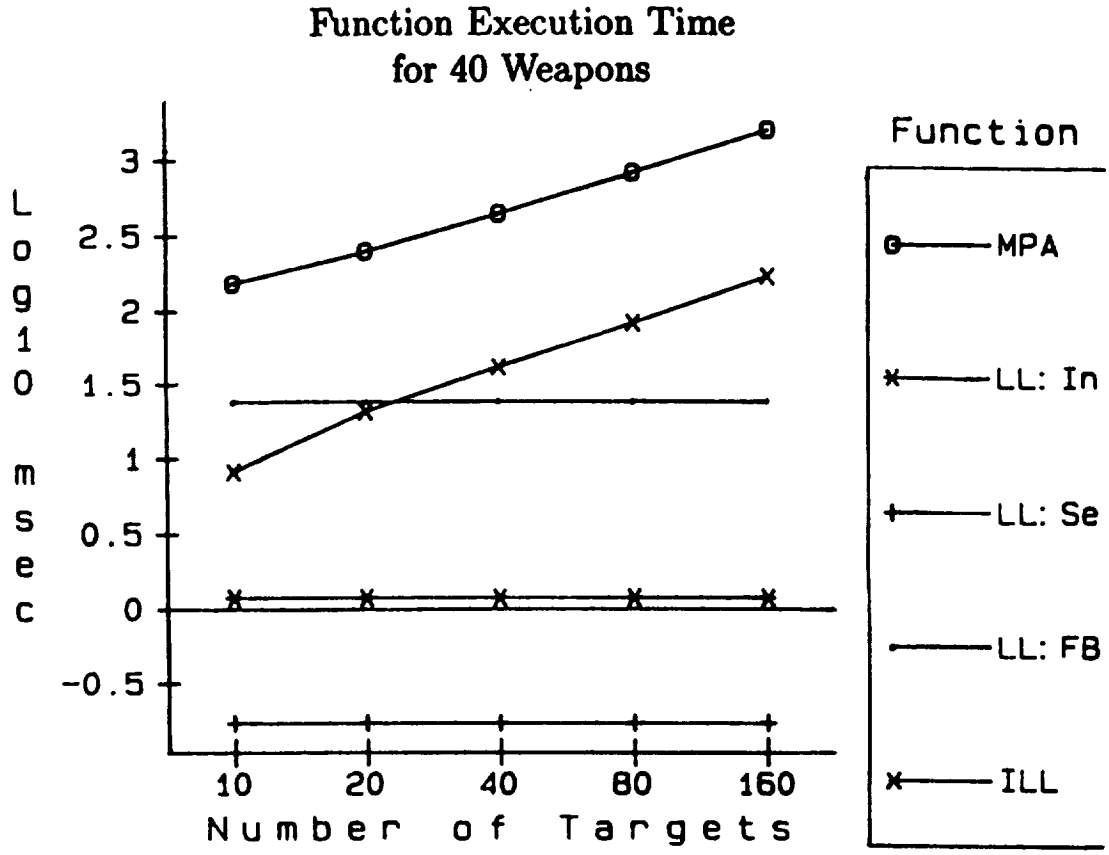


Figure 4.32. WAUCTION_ASSIGNMENT Workload Distribution by Function for Weapons = 40

C-2

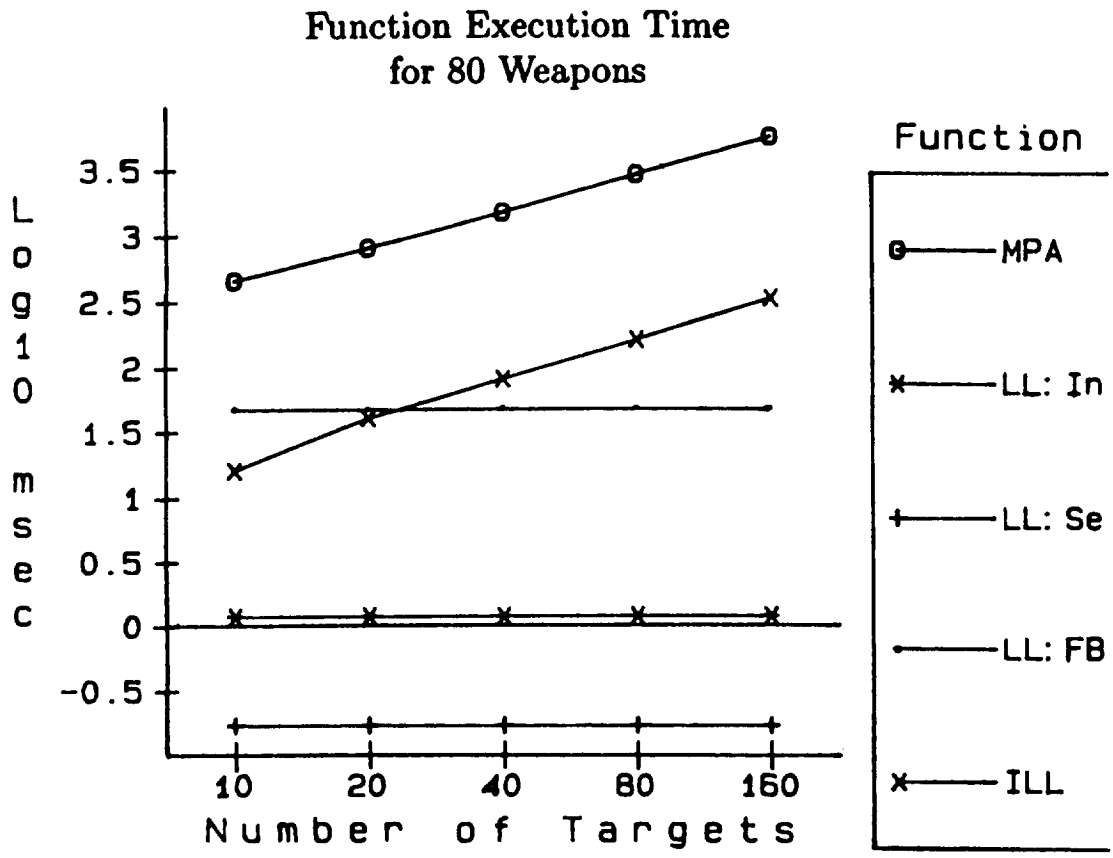


Figure 4.33. WAUCTION_ASSIGNMENT Workload Distribution by Function for Weapons = 80

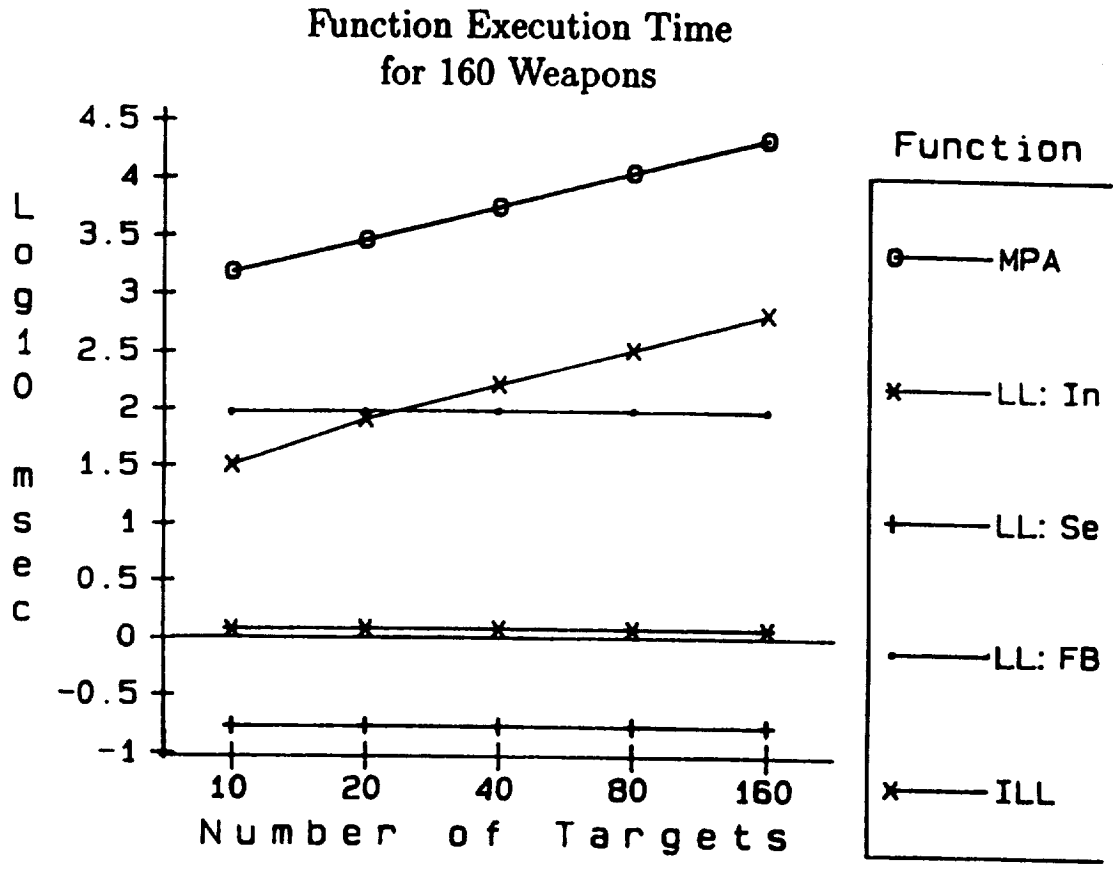


Figure 4.34. WAUCTION_ASSIGNMENT Workload Distribution by Function for Weapons = 160

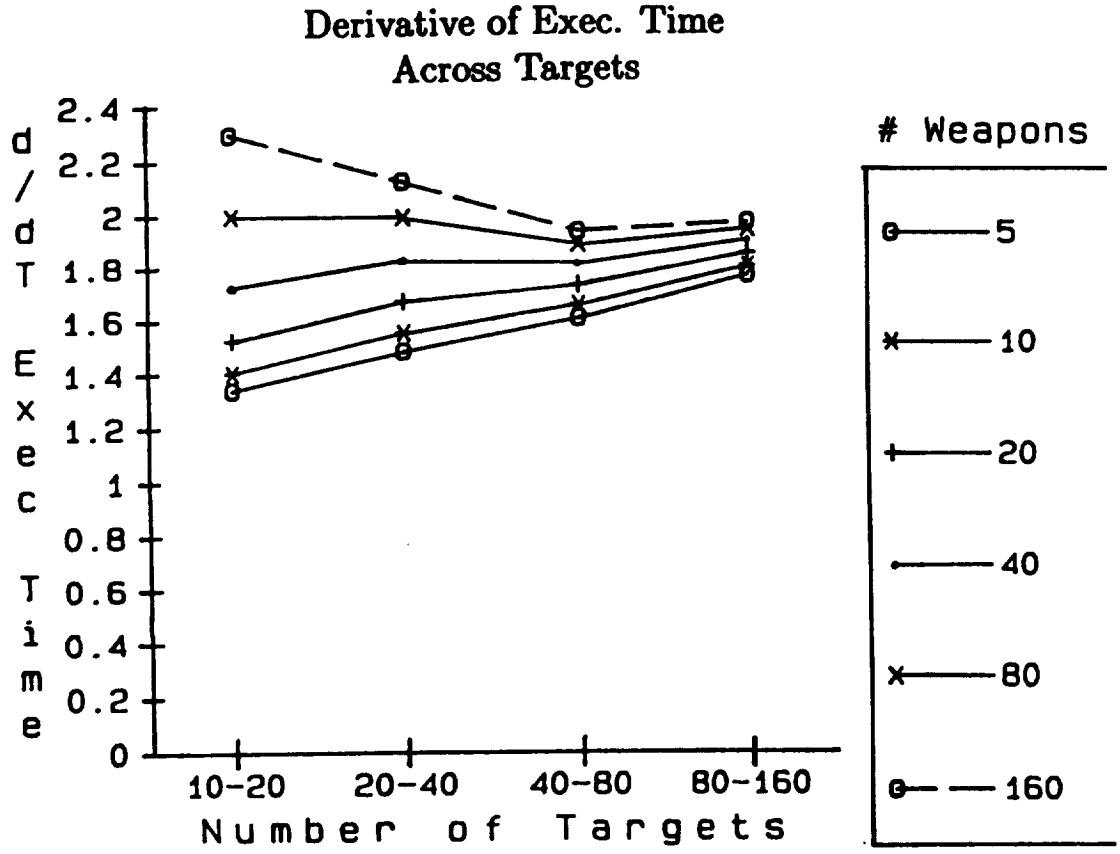


Figure 4.35. First Difference of WAUCTION_ASSIGNMENT Workload with Respect to Number of Targets

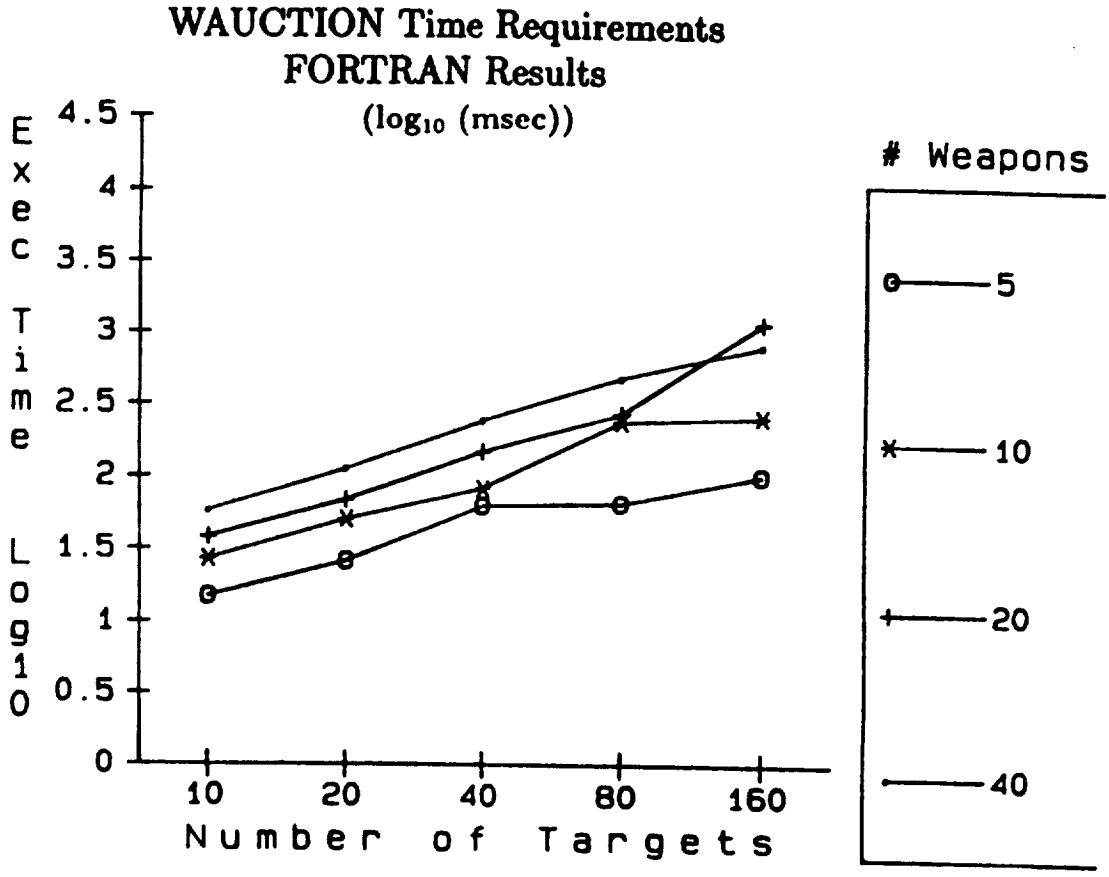


Figure 4.36. Measured CPU Time for WAUCTION_ASSIGNMENT

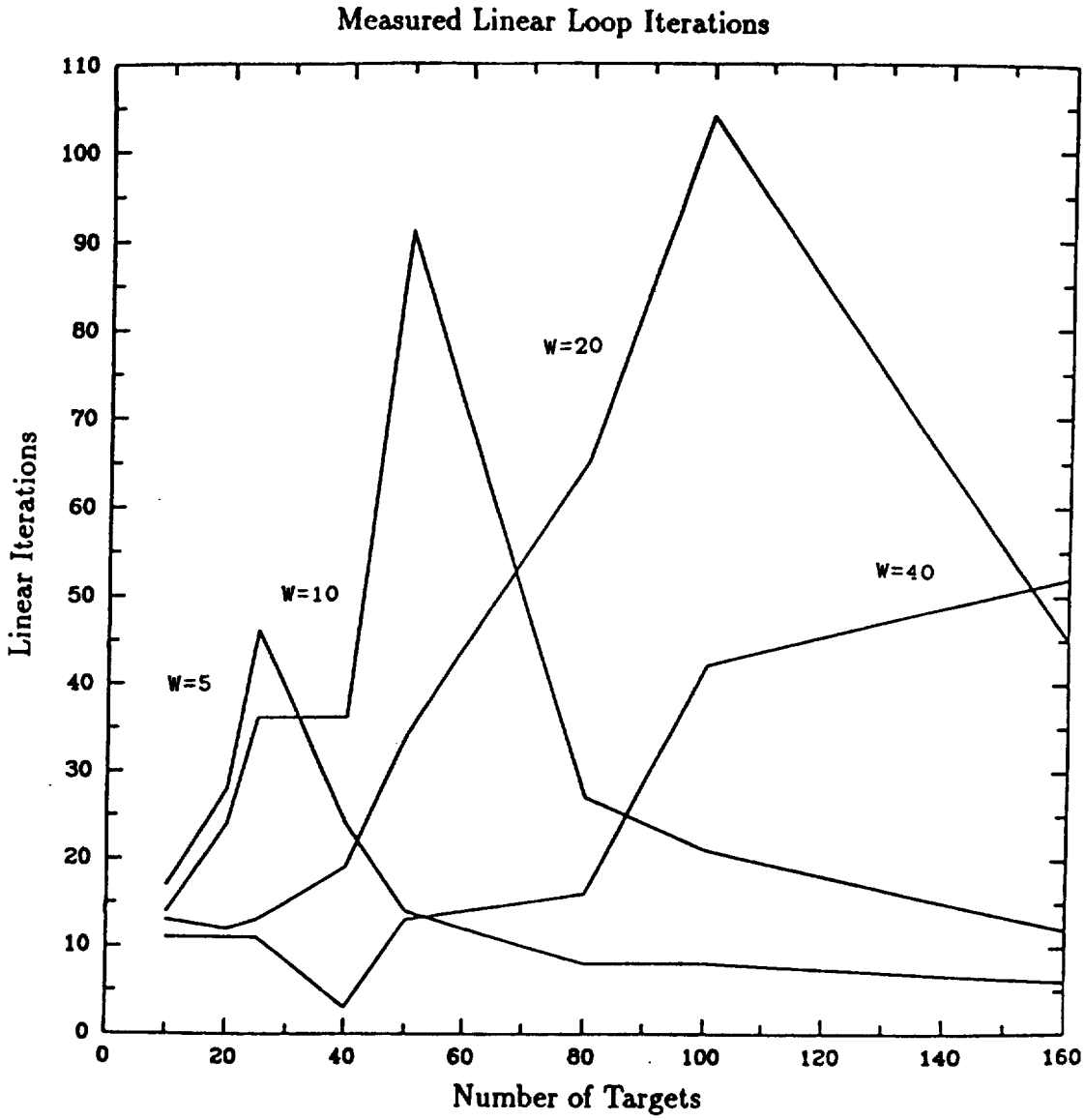


Figure 4.37. Loop Iteration Counts for WAUCTION_ASSIGNMENT Linear Loop

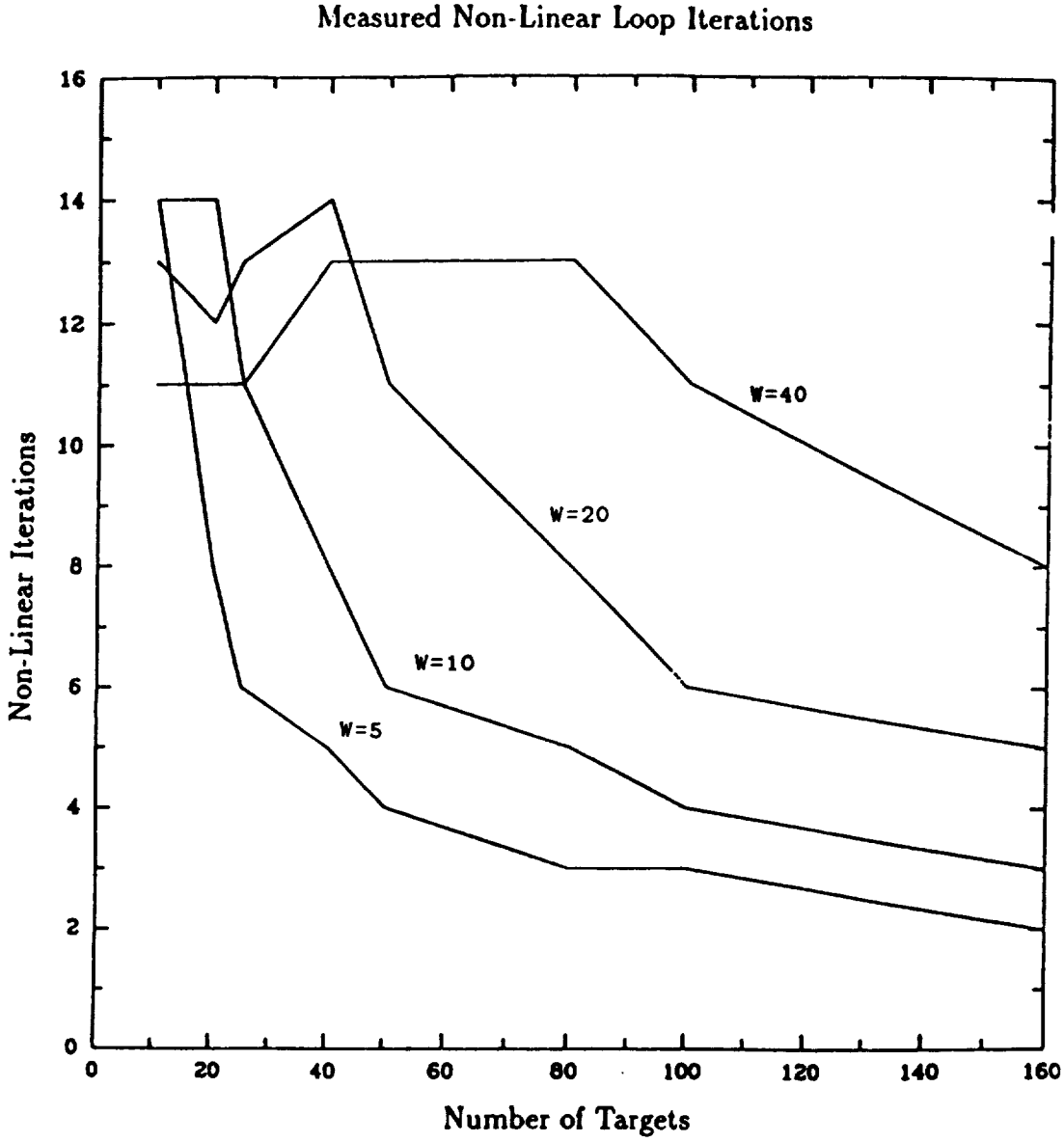


Figure 4.38. Loop Iteration Counts for WAUCTION_ASSIGNMENT Non-Linear Loop

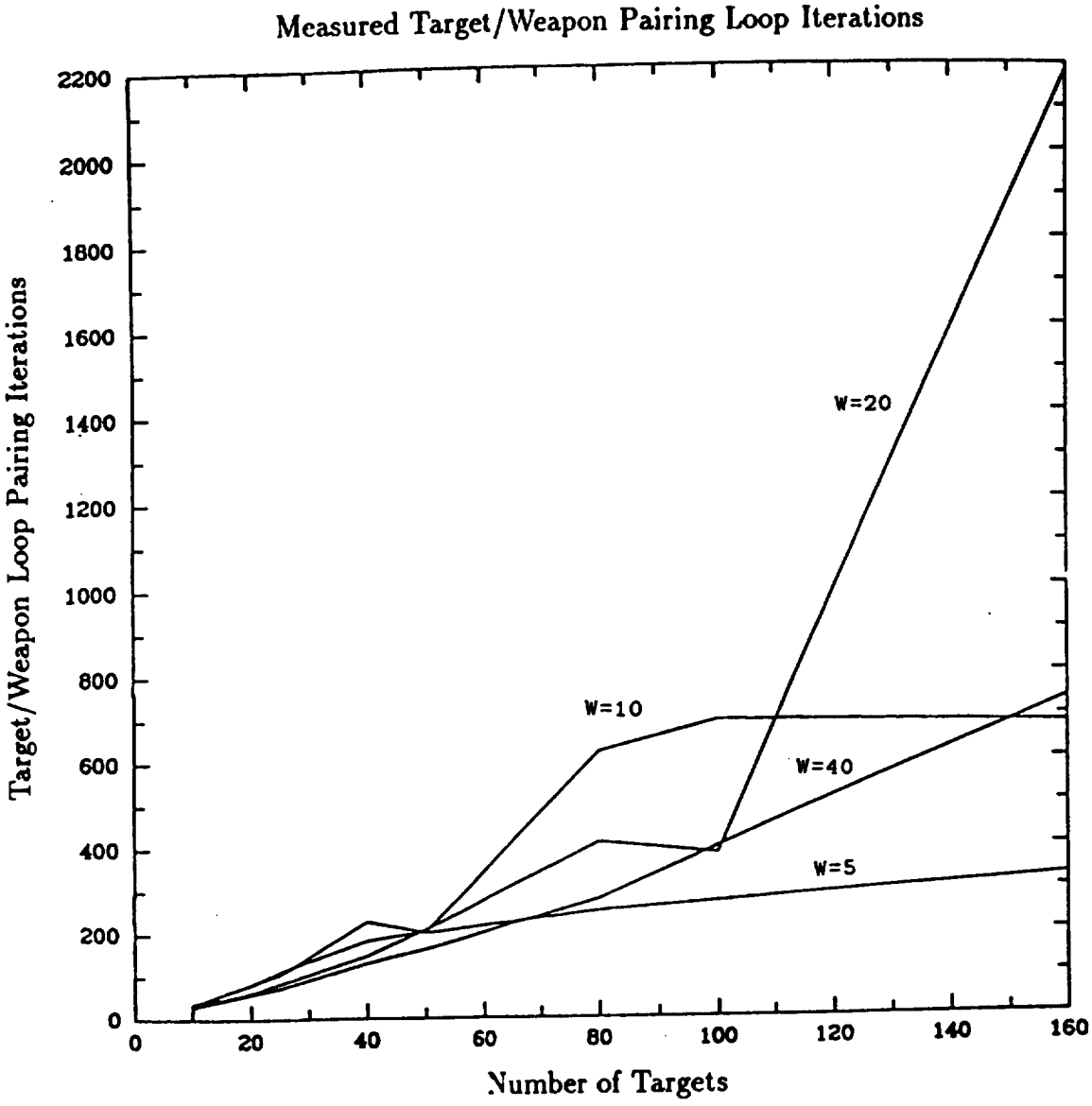


Figure 4.39. Loop Iteration Counts for WAUCTION_ASSIGNMENT Target Weapon Pairing Loop

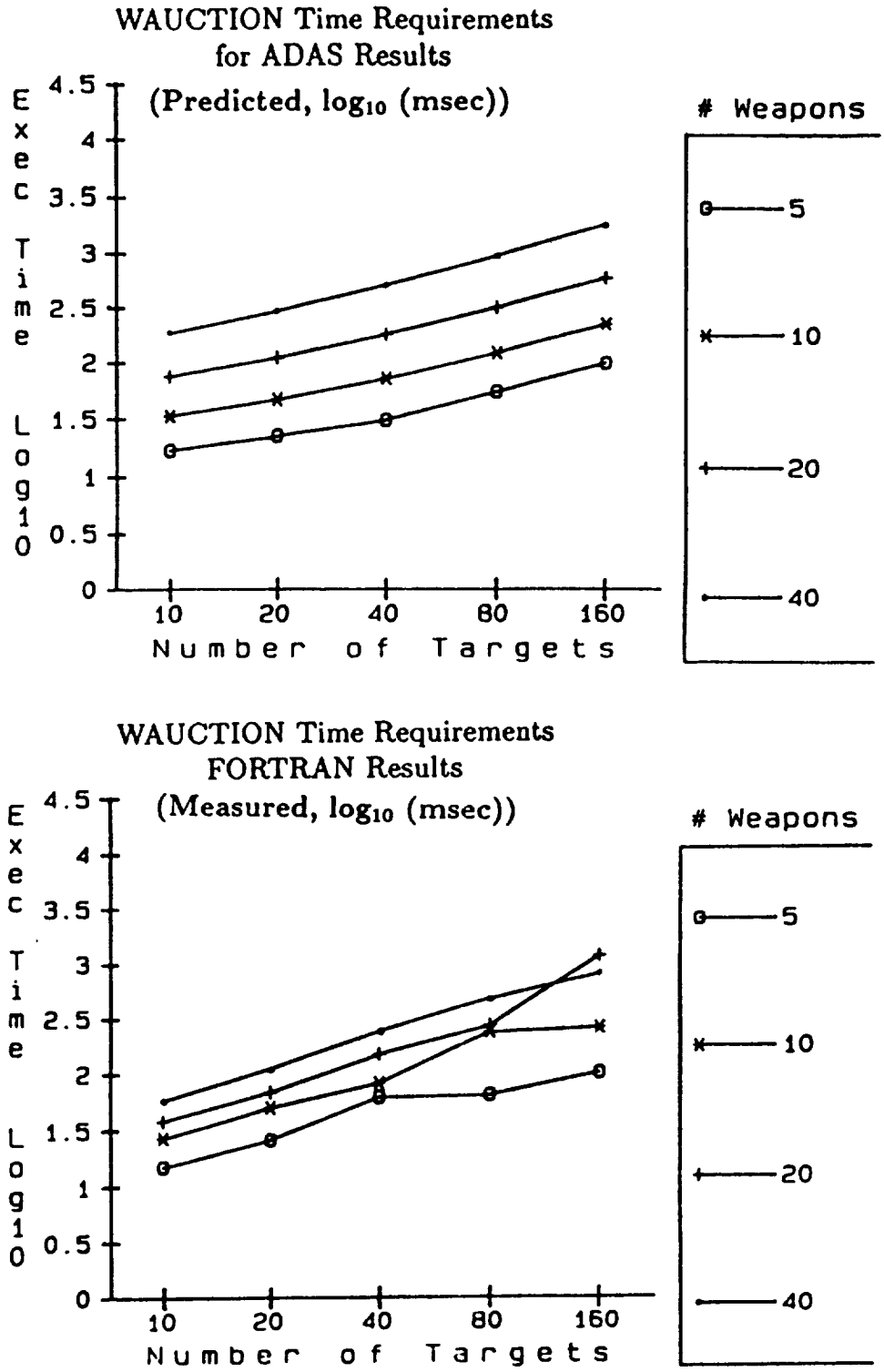


Figure 4.40. Measured versus Predicted for WAUCTION_ASSIGNMENT

lies in the impact that such a factor can have on modeling results. Multiplication of the results which were based on the primitive information used in the model by such a large factor requires that the factor be known with some degree of accuracy. Consequently, analyses or measurements must be made to establish the implementation expansion factor. Alternately, the components contributing to the factor, such as software operating system characteristics and compiler expansion characteristics, must be incorporated into the performance modeling process.

4.6. WTA/TS Using the Hypercube and the Multimax

Two distinct parallel processing architectures were investigated using the WTA/TS algorithm as the target application workload. Honeywell Systems Research Center conducted the investigation under a subcontract from RTI.

The performance modeling carried out in this effort is representative of that which would occur after the high-level, early design phases. As such, the models require the use of more detailed algorithms and architecture design information. These tasks resulted in the most extensive use of the performance modeling tools during this research effort. As anticipated, these more demanding tasks taxed the methods and tools and resulted in the identification of weaknesses and limitations.

The SDI algorithm analyzed in this study embodies the functions of target cluster detection (TCD), weapon-to-target-cluster assignment (WTC), weapon-to-target assignment (WA), and target sequencing for fire control (TS). These major functions occur in a fixed serial order. However, opportunities exist for parallelism within each of these functions. Parallelism can be realized as a function the number of targets being tracked, the number of target clusters, or the the number of weapons platforms. For the purposes of this study, the number of clusters and weapons platforms have been fixed at ten each. The number of targets is variable, and the performance for 100, 200, 400 and 600 targets was examined. The definition of the computational requirements of the algorithm utilized in this study have been documented by Research Triangle Institute and served as the starting point for this analysis. The computational requirements of each task are described by three parameters: operation count, number of reads/writes, and number of multiplications. Partitioning of the major functions and subfunctions and coalescing of all tasks that must execute serially produces an algorithm structured as a macropipeline as shown in Figure 4.41. The individual ADAS graphs for each function and subfunction are documented in Appendix A of [15].

In Figure 4.41, each set of tasks that can execute in parallel will be referred to as a *stage* of the pipeline. The tasks in a stage will be referred to as *parallel tasks* and all other tasks will be referred to as *serial tasks*. Each stage of the pipeline can be represented as an ADAS graph, as illustrated in Figure 4.42. Each task can be represented by a node in the ADAS graph. This node will be assigned a *firing delay* to model the execution of this task. The firing delay is computed from the knowledge of the operation counts and the architectural parameters such as the processor speed, memory access latency, and I/O speed.

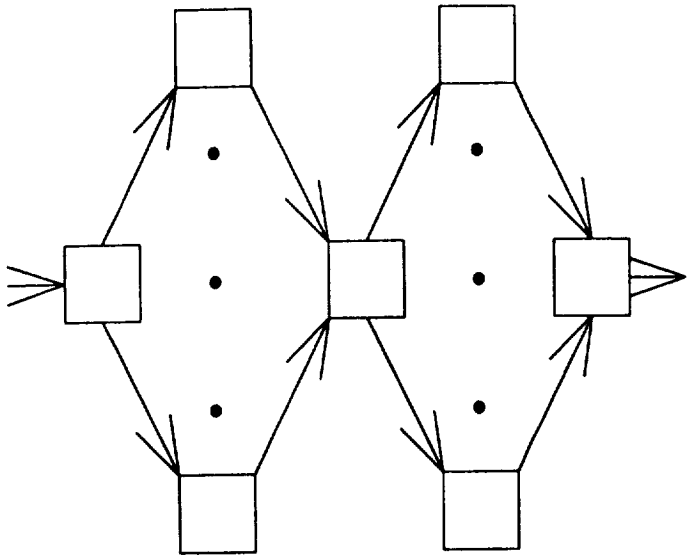


Figure 4.41. Algorithm Structure

In the ADAS graph it is necessary to explicitly represent events such as the transfer of information between tasks. In the algorithm structured as it is in Figure 4.41, there are two communication patterns that must be supported. At the beginning of a pipeline stage, information is transferred from a serial task to all of the parallel tasks in the stage. Each parallel task receives distinct partitions of data, e.g., a matrix. This pattern of providing data to all of the tasks within a stage is a *broadcast tree.* Analogously, at the conclusion of a stage, each of the parallel tasks provides results which will be utilized by the serial task preceding the next stage. Such a pattern of communication is an *accumulation tree.* Communication nodes representing the broadcast and accumulation of data are explicitly represented as shown in Figure 4.42.

The algorithm model studied does not allow very much parallelism. This is evident from Figure 4.43 which illustrates the percentages of serial operations (those cannot be parallelized) as a function of increased workload (represented by the number of targets). Based on this mix of serial operations, the maximum speedup for a parallel processor over a uniprocessor would be bounded by a factor of about 3 for 100 targets. However, the primary goal of this program is not the absolute performance of the algorithm, but the development of insight into the requirements for tools necessary for the design and performance evaluation of similar systems.

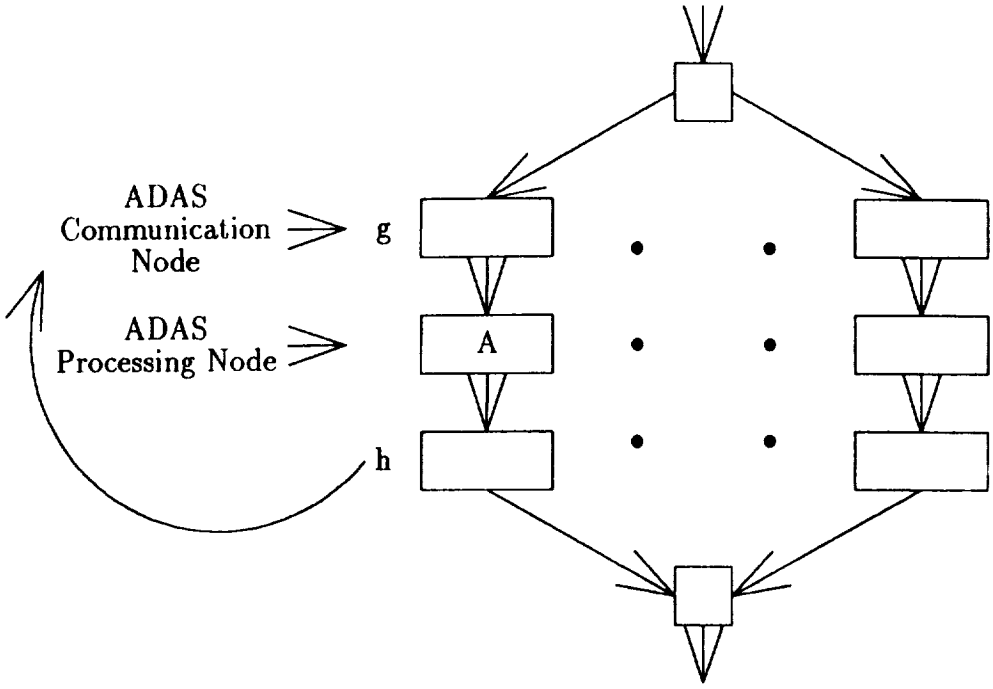


Figure 4.42. ADAS Representation of a Pipeline Stage

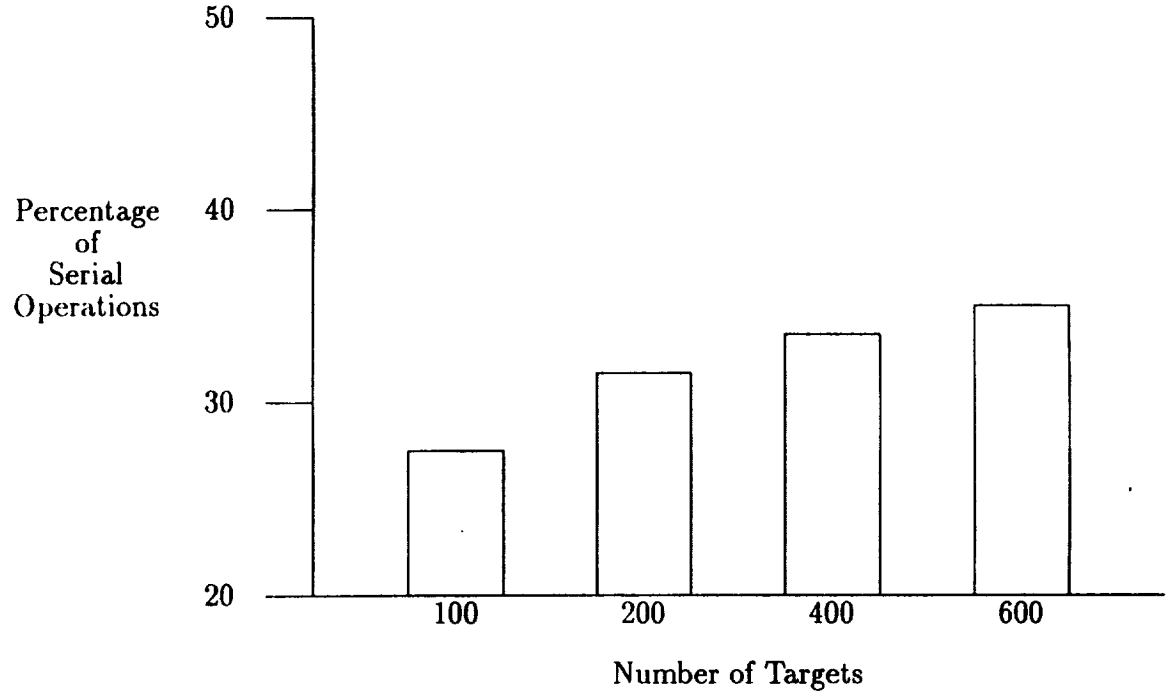


Figure 4.43. Algorithm Characteristics

As will become evident in the following section, this description of the algorithm is utilized in two different ways in the analysis of the Mark III Hypercube and the Encore Multimax.

4.6.1. JPL Mark III Hypercube

Consider the algorithm representations in Figures 4.41 and 4.42. Two nodes in the graph that are ready to fire cannot do so if they both require the same hardware resource. In distributed memory architectures, tasks executing on distinct processors utilize separate processor-memory paths. Only tasks (communication nodes) mapped to the same processing element (link or intermediate routing node) are so constrained. Thus, for a given mapping algorithm, the resolution of the simulation can be that of the firing delay of the smallest task. The accuracy of the firing delays themselves can be computed based upon detailed knowledge of the operation counts of the tasks and the architecture of the individual node. Based on these characteristics, the GIPSIM utility of ADAS was used to analyze the performance of the Mark III hypercube.

The modeling parameters of interest for the communication processor (CP) are the three modes of data (or equivalently packet) motion the CP is responsible for: local-memory-to-channel-buffer, channel-buffer-to-channel-buffer, and channel-buffer-to-local-memory.

The modeling parameters of interest for the data processor are the interrupt latency to respond to the presence of a message, the speed of the processor and the speed of its associated array processor.

The analysis of the Mark III Hypercube involves two steps: task mapping, and the scheduling of intertask communication. Our approach to both of these problems is described in the following subsection.

4.6.1.1. Mapping

Given the macropipeline structure of the algorithms, the mapping is relatively straightforward. All of the serial sections can be mapped to one PE, e.g., PE 0. All of the tasks in the same stage of the pipeline can be mapped to any set of distinct PEs and can therefore execute in parallel. Tasks in different stages are independent, are never active at the same time, can be mapped independently, and can co-exist on the same

PE. The specific choice of PEs for a stage is dictated by the need to minimize communication, i.e., the time required to realize the broadcast trees at the start of each stage, and the accumulation trees at the end. This is achieved by placing all tasks in a stage on PEs that are as close as possible to the PE executing the serial section. In our analysis we chose PE 0 to execute the serial section. Given the symmetric nature of the hypercube topology, any other PE would have produced equivalent results. Figure 4.44 illustrates the hypercube of Figure 3.5 redrawn to show PEs at a given distance from PE 0.

Ideally, it would be desirable for all of the tasks in a pipeline stage to be mapped onto PEs adjacent to PE 0. However, due to the fixed number of channels per PE, this is not always possible. A straightforward greedy algorithm for performing the mapping is as follows. With reference to Figure 4.44, mapping for a stage is performed left to right, level by level, starting at the level of PEs distance 1 from PE 0. One possible mapping for a stage of six tasks is $(0 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 4, 3 \rightarrow 8, 4 \rightarrow 5, 5 \rightarrow 3)$, where $\rightarrow$ signifies a mapping from a task to a PE. Since the communication pattern is that of a broadcast tree, it does not matter how individual tasks are assigned to PEs within a level.

The remaining issue concerns the mapping of the communication nodes corresponding to the broadcast trees and accumulation trees. Since some pairs of tasks can be at least distance two from each other, it is not feasible to construct a mapping of communication nodes to links or intermediate nodes. Our approach is to assign delays that reflect intermediate node routing, and conflicts in the use of channels. This requires a complete static scheduling of intertask communication of the hypercube for a given mapping. The regular structure of the hypercube and the algorithms under consideration makes this possible. This interprocessor communication schedule for hypercubes and macropipelines is described in the following subsection.

4.6.1.2. Scheduling of Intertask Communication

Initially we obtain an expression for the communication delay for a sequence of packets traversing an arbitrary number of links. This expression is in terms of the parameters of interest and assumes the absence of any routing conflicts. All communication takes place at the beginning and end of an algorithm pipeline stage. The communication delays are then adjusted to include routing conflicts between each independent sequence of packets. All scheduling of intertask communication is performed statically. This is possible because of the structure of the hypercube and the simplified structure of the algorithms.

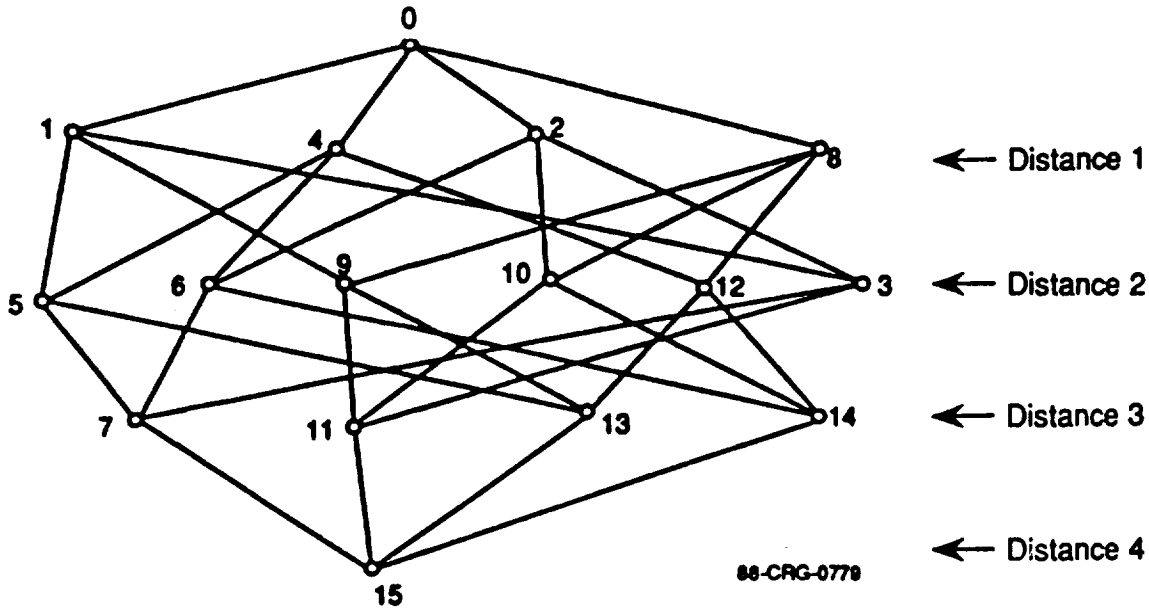


Figure 4.44. Interprocessor Distances in a 16 Node Binary Hypercube

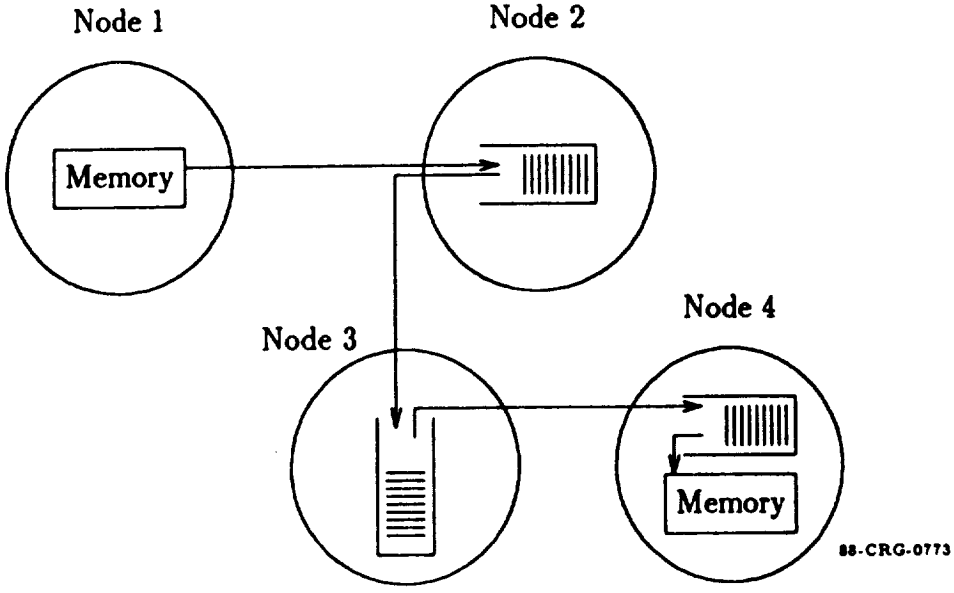


Figure 4.45. Interprocessor Communication in the Mark III

Figure 4.45 illustrates the functional nature of interprocessor communication in the Mark III. The source node must transfer data from local memory to the buffer of an adjacent node. Intermediate nodes must perform transfers between channel buffers. Destination nodes must perform transfers from channel buffers to local memory. Each packet in the Mark III is 64 bytes long. Of this, 56 bytes are for data and the remainder are required for destination routing and control. Packet transmission can be pipelined across these links.

When the first packet reaches the destination, it interrupts the processor. This interrupt latency is typically large relative to the interprocessor packet transfer time. It is also a function of the processor design and the operating system. Subsequent packets incur a smaller interrupt latency. With the Mark III running the Crystalline operating system, these latencies are 100μsecs and 25μsecs respectively. As depicted in Figure 4.45, multipacket transfers can be treated as a pipeline with the following expression for the delay:

$$\text{Delay} = t_m + (h-1)t_{rout} + t_{int} + t_m + (p-1)(t_{rout} + max(t_{rout}, (t_{pint} + t_m))),$$

where

t_m	=	memory to channel buffer packet transfer time,
h	=	number of links traversed,
t_{rout}	=	packet routing delay at a node,
t_{int}	=	interrupt response time for the first packet,
p	=	number of packets, and
t_{pint}	=	interrupt response time for successive packets

Consider Figure 4.44 and the beginning of a stage of computation involving 10 tasks. These 10 tasks will be mapped to the processors in the first two rows of Figure 4.44. Each of these processors must receive data from processor 0 to begin their computations. Data communication between processor 0 and the remaining processors is scheduled in the following manner. Data is successively transferred to processors 1,4,2 and 8. Data is then transferred to processor 5, being pipelined across two links ($0 \rightarrow 1$ and $1 \rightarrow 5$). After the last packet is transferred to processor 1, communication of data can begin to processor 6 via two links ($0 \rightarrow 4$) and ($4 \rightarrow 6$). In a similar fashion, the initiation of communication to processor 9 can be overlapped with the last few packets being transmitted to processor 6. The richness of the topology of the hypercube enables this distribution of data to the processors to take place without having to wait for a busy link to become available.

The firing delays of each of these communication nodes can now be fixed according to the communication schedule described above. For example, in Figure 4.44, the delay for an ADAS node representing communication delay of data to a task mapped onto processor 6 would be computed as follows:

$$t_6 = t_1 + t_4 + t_2 + t_8 + t_5 - F$$

where t_i = time to transfer data to processor i, and F = time to transfer the last few packets to processor 5. These are the packets that overlap with transfers to processor 6.

Thus, all scheduling is done statically, and there are no conflicts. The accumulation of data at processor 0 from all of the processors is performed in a similar manner. Figure 4.46 illustrates a schedule for sixteen processors. Each vertical column represents the unit of time required to transfer the required data between a pair of processors. Specific data transfers are indicated within each column, i.e., $5 \rightarrow 7$ represents a transfer from processor 5 to processor 7. Multiple transfers in the same column imply that they take place simultaneously. Figure 4.47 provides a similar schedule for ten processors. All of the data is being accumulated in one processor (processor 0). Relative to this processor, both of these schedules enable data to be transferred into it in the minimum amount of time; i.e., processor 0 is never waiting for data. Therefore, this schedule represents an efficient schedule. The actual values of the communication delays to be associated with the ADAS communication nodes at the end of a stage are derived from the first line in the schedules shown in Figures 4.46 and 4.47. For example, if the data from a specific node arrives at processor 0 in the ninth time unit, then the delay of the corresponding communication node is computed as (9*the duration of the time unit). The duration of the time unit is the amount of time to transfer the required number of packets between adjacent processors.

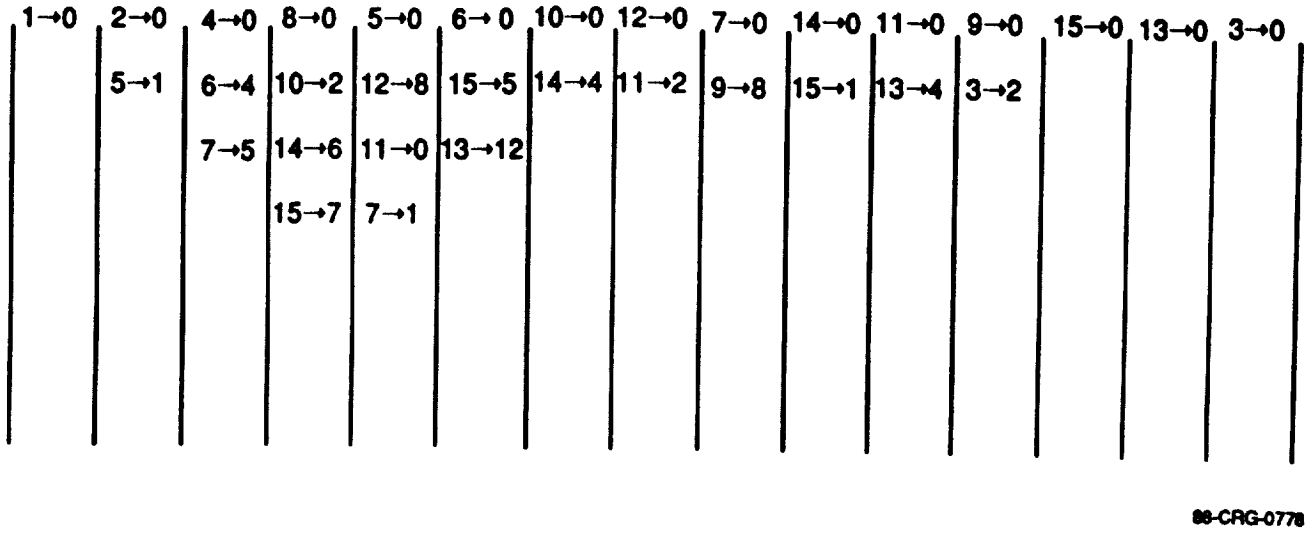


Figure 4.46. A Communication Schedule for 16 PEs

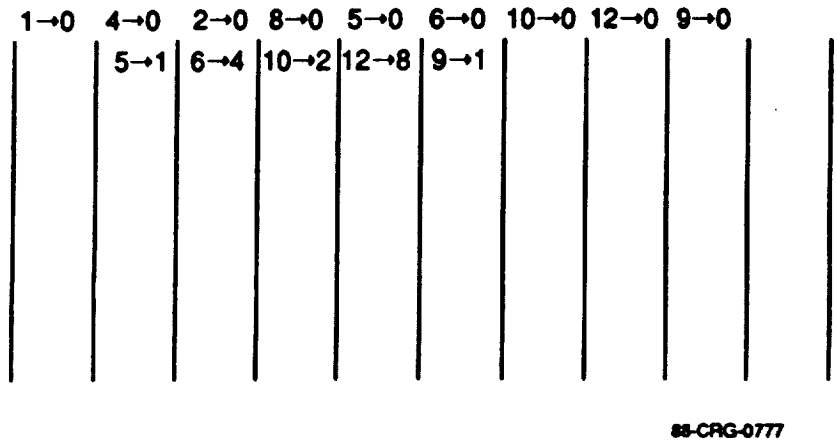


Figure 4.47. A Communication Schedule for 10 PEs

The algorithms were mapped, scheduled, and the firing delays for the communication nodes fixed according to the schedules described above. The results of the ADAS simulations are described in the following section.

4.6.1.3. Simulation Results

The architectural parameters of interest are the the processor speed, the routing delay, and the memory speed (this includes transfer into and out of the channel buffers). The algorithm parameter of interest is the number of targets, which dominates the complexity of the algorithm. The performance parameters of interest are the latencies for the algorithm execution and the processor utilizations.

Our approach has been to analyze the performance of the "base" case, and then the effect of varying the architectural and algorithmic parameters of interest. In this manner it would be possible to discern how much improvement can be obtained by improving a particular parameter. For example, if 40% improvement in latency is required, it may be possible to observe that 20% can be obtained by improvements in processor speed, 15% from improvements in routing delay, and 5% from feasible improvements in memory speed. The simulation experiments have all examined the effect of a single parameter on the performance of the machine relative to the base case. The base case is characterized by a processor speed of 2 MIPs, memory access time of 260 ns, routing delay of 40 ns per node, and 100 targets. The range of parameter values considered were:

- 100, 200, 400, and 600 targets
- 2, 5, 10, 15, and 20 MIPs
- 260, 150, 100, and 50 ns memory access time
- 40, 20, and 1 nsec routing delay

The effect of the architecture/algorithm parameters on the latency and utilization is shown in Figures 4.48 through 4.52.

From Figures 4.48 through 4.50 it is evident that the TCD is the dominant function. Processor 0 hosts the serial fraction. Furthermore, due to the sizable serial fraction and the compute-bound nature of the application, the speed of interprocessor communication and memory access has a negligible effect. This is also evident

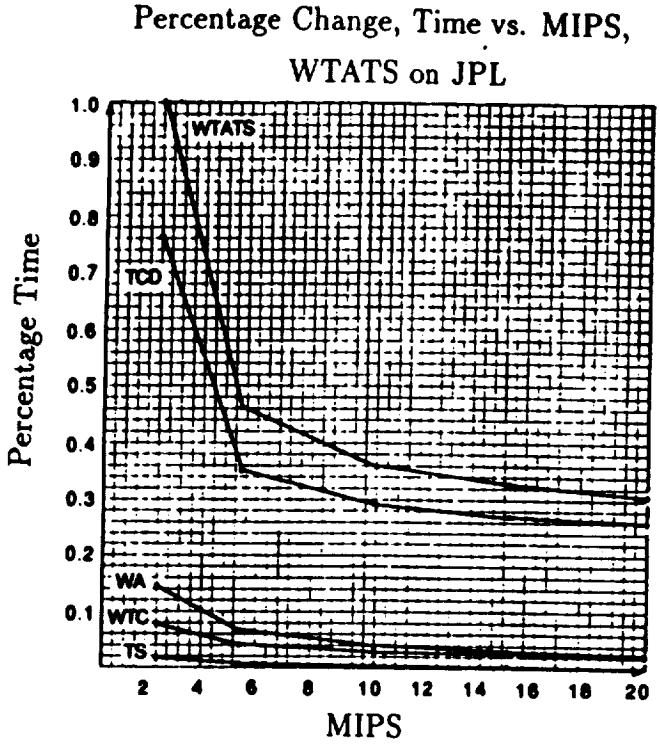


Figure 4.48. Latency vs. Processor Speed

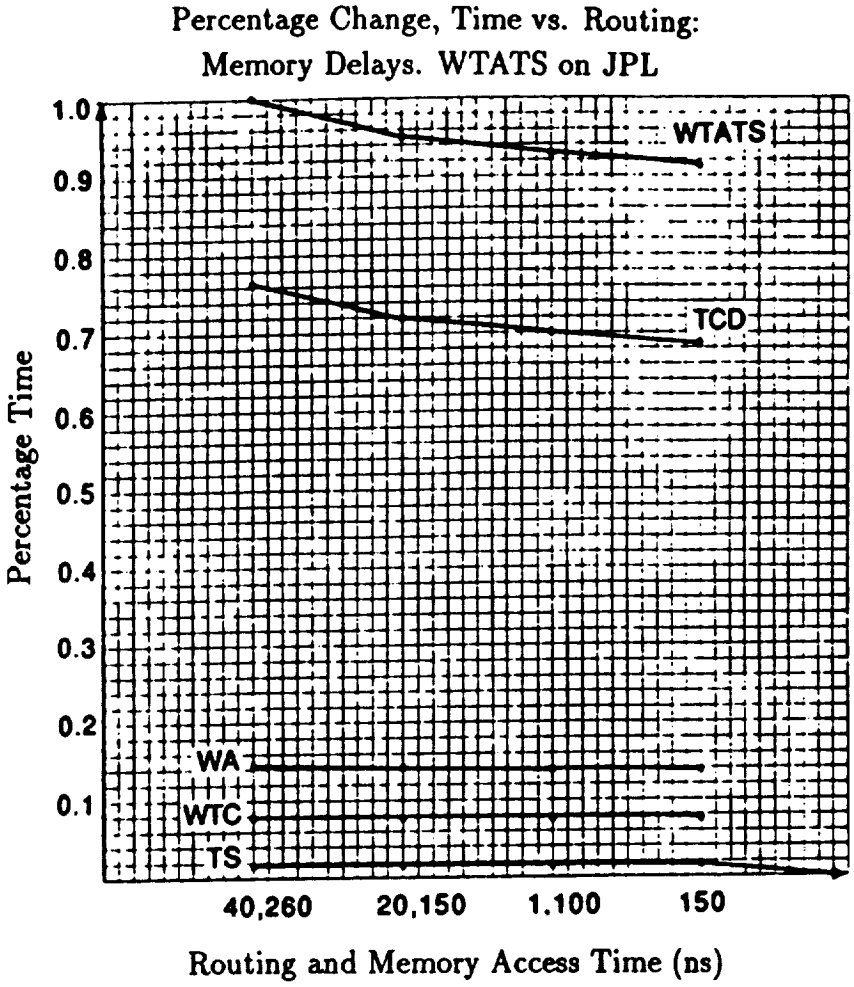


Figure 4.49. Latency vs. Routing Delay and Memory Access Time

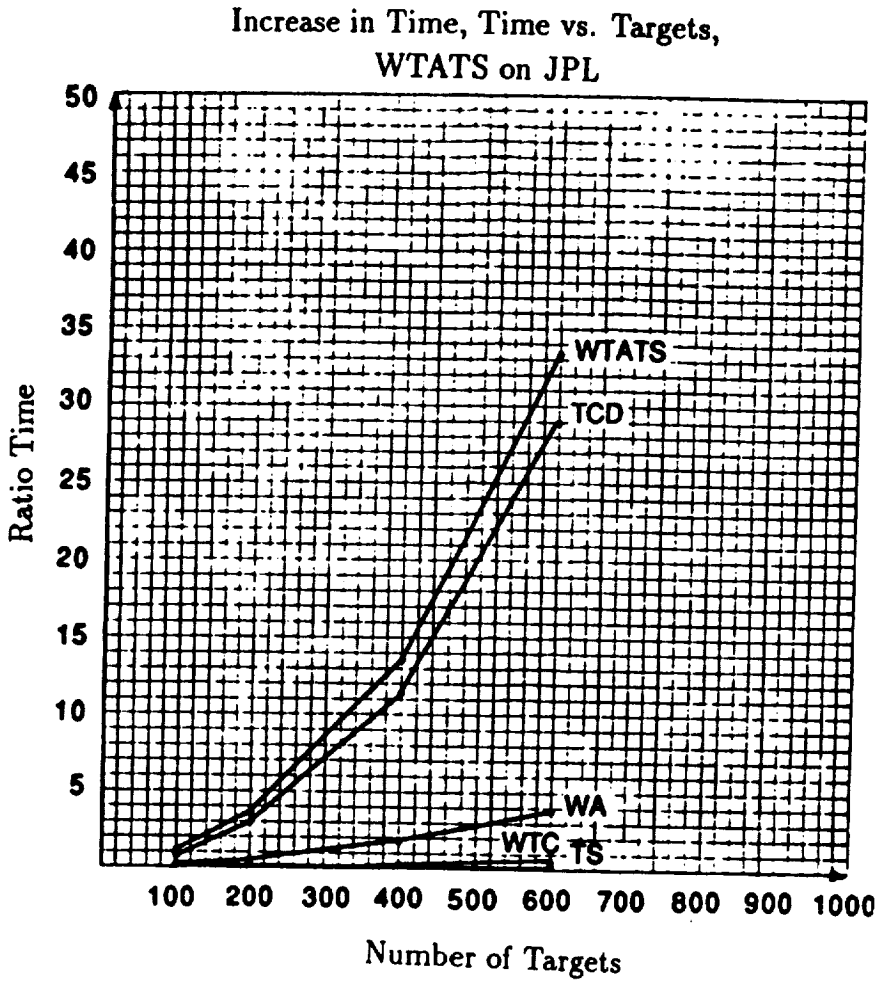


Figure 4.50. Latency vs. Number of Targets

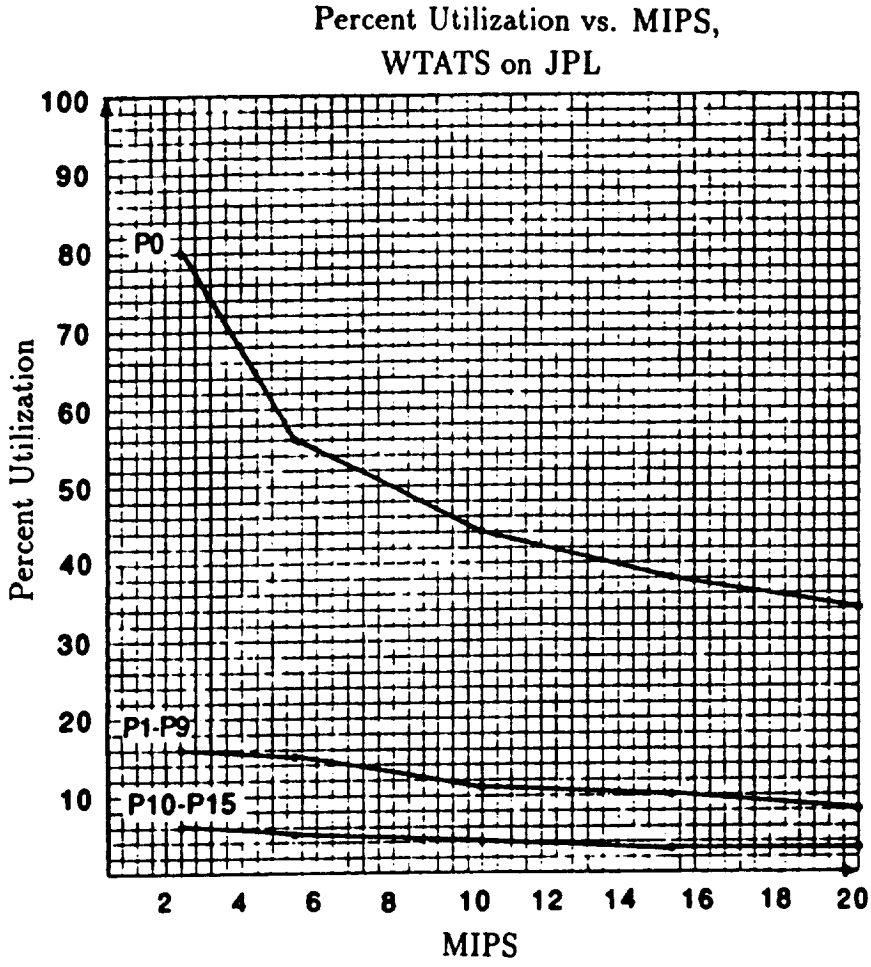


Figure 4.51. Utilization vs. Processor Speed

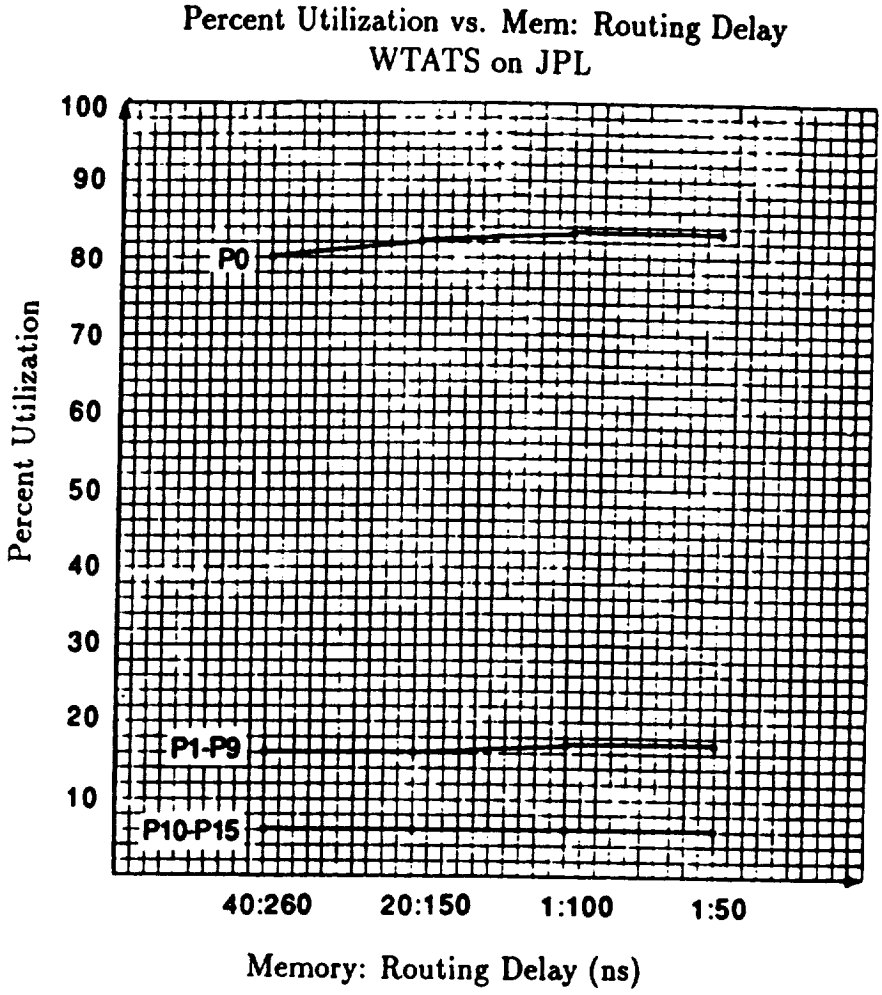


Figure 4.52. Utilization vs. Routing Delay and Memory Access Time

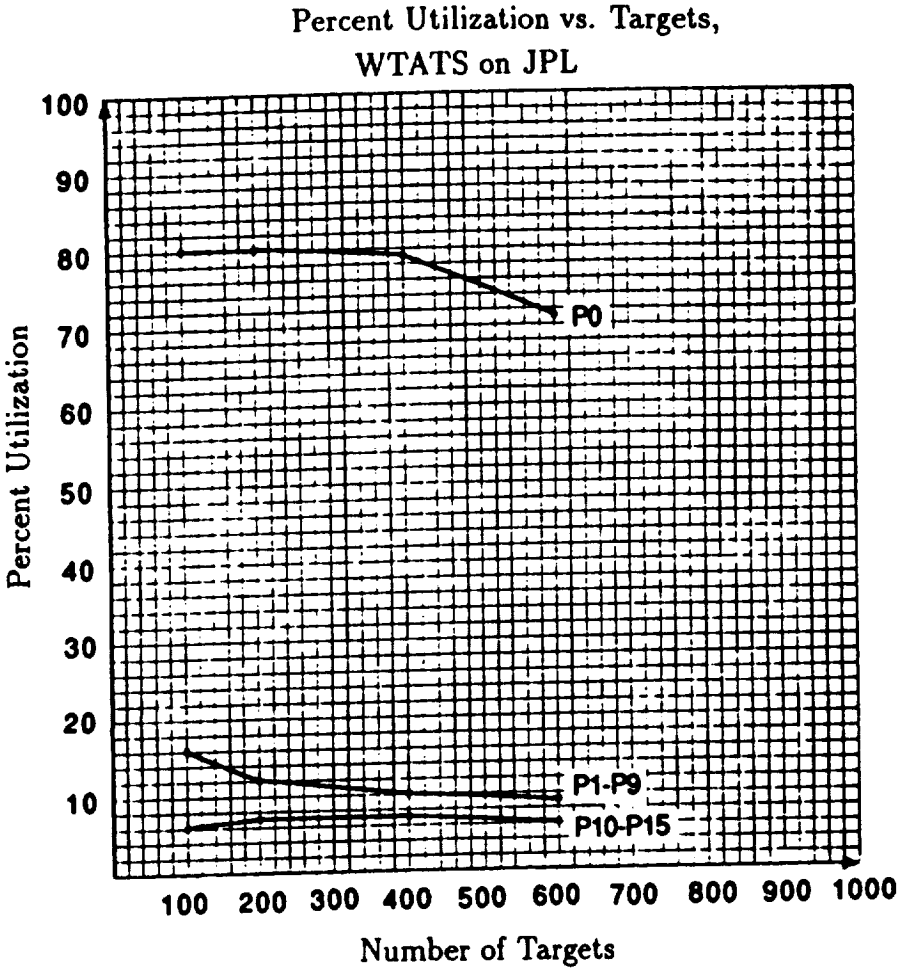


Figure 4.53. Utilization vs. Number of Targets

from Figures 4.51 through 4.53. This information is reproduced in tabular form in Figure 4.54. This table also indicates the percentage performance improvement that can be expected from improvements in the individual architectural parameters.

4.6.2. Encore Multimax

By definition, processors in bus-based shared memory architectures require access to a shared resource on potentially every instruction execution cycle. The execution of every instruction potentially competes with each other since they all require the use of the global bus for memory access. Multiple processors can also potentially conflict on every cycle in the access to shared memory. Thus, these architectures are very dependent upon the mapping of (shared) data structures into the memory modules. Furthermore, memory modules tend to be heavily interleaved to match the processor/bus bandwidth.

The normal approach to achieving high simulation resolution is to map tasks to processors and all communication nodes to the bus, essentially serializing all communication. However, most architectures use some form of a cache to alleviate the processor/bus/memory bottleneck. The performance depends very heavily on the use of the cache. In the case of the Encore Multimax, modeling is further complicated by the fact that pairs of processors share access to a single cache. Cache coherency control adds to the memory traffic in a non-trivial way and must be a part of the analysis. The interactions between the processor, cache, bus and memory are at the level of the fastest device (typically the cache or bus), and this dictates the level of resolution of the simulation. Since phenomenon such as cache hit rates and bus arbitration are generally non-deterministic, an approach to simulation must enable behavioral modeling while explicitly preserving the structure of the architecture. For this we used the CSIM facility of ADAS.

The behavior of each modeled component of the Multimax was described by a program written in the C programming language. The basic time unit of the model was the speed of the fastest component in the system — the nanobus. The nanobus is capable of transferring data between processors at the rate of two 32-bit words every eighty nanoseconds. The time for bus arbitration and control is modeled as a function of the bus interface units. The behavior of the nanobus itself required reading processor and memory inputs in a round robin fashion and routing them serially to the appropriate processor cache or memory module. Access to the memory modules was modeled as a two-stage pipeline where the delay of each stage was an integral number of bus cycles. In addition, the memory would process the requests responding to read requests while

Module Execution Time in Seconds

MIPS	2	5	10	15	20
TCD	8.277	3.797	3.137	2.917	2.807
WTC	0.840	0.429	0.292	0.276	0.223
WA	1.567	0.704	0.417	0.321	0.273
TS	0.170	0.076	0.045	0.035	0.030
WTATS	10.854	5.005	3.890	3.548	3.333
Mem:Bus Delay	40:260	20:150	1:100	1:50	
TCD		7.805	7.591	7.449	
WTC		0.820	0.810	0.802	
WA		1.524	1.503	1.485	
TS		0.167	0.164	0.163	
WTATS		10.316	10.068	9.899	
Targets	100	200	400	600	
TCD		31.291	123.608	313.453	
WTC		1.112	1.117	1.129	
WA		5.251	19.074	41.468	
TS		0.636	2.461	5.476	
WTATS		38.291	146.259	361.527	

Percent Deviation of Execution Time from Baseline

MIPS	2	5	10	15	20
TCD		-54.1	-62.1	-64.8	-66.1
WTC		-48.9	-65.2	-67.1	-73.4
WA		-55.1	-73.4	-79.5	-82.6
TS		-55.3	-73.5	-79.4	-82.3
WTATS	100%	-53.9	-64.2	-67.3	-69.3
Mem:Bus Delay	40:260	20:150	1:100	1:50	
TCD		-5.7	-8.3	-10.0	
WTC		-2.4	-3.6	-4.5	
WA		-2.7	-4.1	-5.2	
TS		-1.8	-3.5	-4.1	
WTATS		-4.9	-7.2	-8.8	
Targets	100	200	400	600	
TCD		278.0	1393.4	3687.0	
WTC		32.4	33.0	34.4	
WA		235.1	1117.2	2546.3	
TS		274.1	1347.6	3121.2	
WTATS		252.8	1247.6	3230.9	

Figure 4.54. Performance of the Mark III Hypercube

acting as a sink for write requests. All instruction accesses were directed to a single memory module that stored a copy of the code. Instruction reads prompt the memory to return a block of instructions, thereby mimicking block loads that typically occur on an instruction cache miss. Only the first instruction reference to a block causes a miss and the transfer of the block into the cache. Thereafter, remaining instructions in that block can be referenced at cache speeds. No explicit cache fetch policy, replacement policy, or mapping policy was implemented within the model. All of these policies affect the hit rate, and it is only the variances in the cache hit rate that is modeled.

The DPC consists of two processors, the cache and the nanobus interface. The nanobus interface acts as a simple delay for accessing the nanobus. The cache is the sink for all processor requests and memory reads. It embodies a user specifiable cache hit rate. This determines the number of requests that are actually transmitted over the nanobus. The processor generates all of the instruction/data read and write requests. On a read miss to the cache, the processor waits until the read is satisfied by main memory. On a write miss to the cache, the write to memory is initiated and the processor can continue execution from the cache. This model executes a specific algorithm by having the processors generate instruction reads and data read/write requests in accordance with the specific tasks mapped onto them. The means to do this must also enforce sequencing constraints between tasks. Our approach to achieving this is best understood after an explanation of the mapping of tasks to processors.

4.6.2.1. Mapping

Since each stage can be mapped independently of other stages, mapping of macropipelines to bus-based multiprocessors is trivial. The serial tasks are mapped to one processor (processor 0). All of the parallel tasks in a stage are mapped to distinct processors. There is no data distribution for each stage since this is a shared memory machine. All data is in shared memory. Each of the tasks on a processor belong to different pipeline stages, and therefore they can be executed in a fixed order. The mapping of tasks to a processor is captured in a table such as that illustrated in Figure 4.55. This shows a table for any of the processors not executing the serial tasks. The symbols DR, IR, and DW stand for data read, instruction read, and data write respectively. The numbers reflect the actual number of corresponding operations for the related task. During simulation, the processor model reads these values from the table and proceeds to generate the exact number of memory references and operate (fire) at a user-specified rate, e.g., 2 MIPS. When the task execution is completed, the processor signals processor 0 and waits. When processor 0 has received completion

signals from all processors in a pipeline stage, the next stage is initiated. When a processor model is invoked again during simulation, the next set of values for DR, IR, and DW are used. This corresponds to the task of the next stage of the pipeline that is mapped on that processor. This table is identical for all of the processors executing parallel tasks. Since processor 0 also executes the serial section, its table is different. The table for processor 0 contains information to distinguish between blocks of code that correspond to the serial tasks, and blocks of code that correspond to the parallel tasks (and therefore control has to be broadcast to other processors to initiate their tasks).

The values of DR, DW and IR were derived from expressions supplied by RTI. These expressions were in terms of the algorithm parameters, e.g., number of targets, redundancy, number of weapon platforms and number of clusters. The expressions themselves were derived from an analysis of the algorithms. The input to Honeywell's effort was a description of the parallel segments of the BM/C^3 algorithms, and expressions for DR, DW and IR for each of the segments.

The CSIM model is independent of the algorithm. All of the algorithm-specific functionality is embodied in the table that is created for each processor. It is possible to envisage compilers that produce these (or similar) tables for a CSIM architecture model from a specification of the computational requirements of the algorithm.

4.6.2.2. Scheduling of Intertask Communication

Intertask communication takes place over the nanobus. While the individual DPC caches reduce the nanobus traffic, it is a shared resource for which access conflicts can occur. These conflicts are a function of the processor behavior, memory access patterns, and the resulting cache hit rate. In the CSIM behavioral model, these conflicts occur naturally in the course of the execution of the simulation. Therefore, no static scheduling of communication need take place.

4.6.2.3. Simulation Results

In the Multimax, interference between parallel tasks can take place at the granularity of an instruction fetch or data store. In order to accurately model the performance of the architecture, simulation must be at the resolution of these events — the 80ns bus cycle time. The cache cycle time is the same as that of the bus, as is the nanobus

		Number of Targets			
Parallel		100	200	400	600
tcd_glm	DR	46875	187500	750000	1687500
	IR	1250079	50000156	20000312	45000468
	DW	46875	187500	750000	1687500
	Broadcast	0	0	0	0
wtc_g	DR	800	800	800	800
	IR	20200	20200	20200	20200
	DW	800	800	800	800
	Broadcast	0	0	0	0
wtc_o	DR	30000	30000	30000	30000
	IR	410000	410000	410000	410000
	DW	30000	30000	30000	30000
	Broadcast	0	0	0	0
wa_wta	DR	51400	102800	205600	308400
	IR	702004	1404004	2808004	4212004
	DW	57400	102800	205600	308400
	Broadcast	0	0	0	0
ts	DR	9100	35800	142000	318600
	IR	312400	1201600	4710400	10526400
	DW	9100	35800	142000	318600

Figure 4.55. Operation Counts for Processor Executing Parallel Tasks

interface cycle time. The memory (non-interleaved) access time is four bus cycles and the processor speeds range between two and twenty MIPS. At this level of granularity and running on a SUN 3/360 workstation, each simulation required approximately 1000 CPU hours. This was clearly infeasible, especially since we are interested in exploring a range of possibilities. As a result, the computational requirements were scaled to reflect the "reasonable" simulation times and the available scope of this effort. Scaling factors used varied from 1000 to 100000. Since it was infeasible to run the simulation at full resolution, it is difficult to assert the effects of scaling effects. It does appear that if the program achieves a form of steady state behavior with respect to its pattern of memory references, and if it achieves this behavior even after after scaling back the computations, then the effects of scaling on performance are insignificant. For example, consider a module of 1000000 instructions, that we scaled back to 1000 instructions. If steady state behavior is achieved with 500 instructions, executing to 1000000 instructions would not significantly affect the performance characteristics. It is difficult to determine within the scope of this effort if indeed this was the case.

The architectural parameters of interest are the processor speed, the bus speed, and the memory speed. The algorithm parameter of interest is the number of targets, which dominates the complexity of the algorithm. The performance parameters of interest are the latencies for the algorithm execution and the processor utilizations. The effect of the architecture/algorithm parameters on the latency and utilization is depicted in Figures 4.56 through 4.63. As in the analysis of the Mark III, performance was examined with respect to a base case-processor speed of 2 MIPs, a bus speed of 80 ns, memory access time of 320 ns and 100 targets. The range of parameters examined were

- 100, 200, 400, and 600 targets
- 2, 5, 10, 15, and 20 MIPs
- 320, 150, 100, and 50 ns memory access time
- 80 ns, 40 ns, 20 ns, and 1 ns bus cycle time

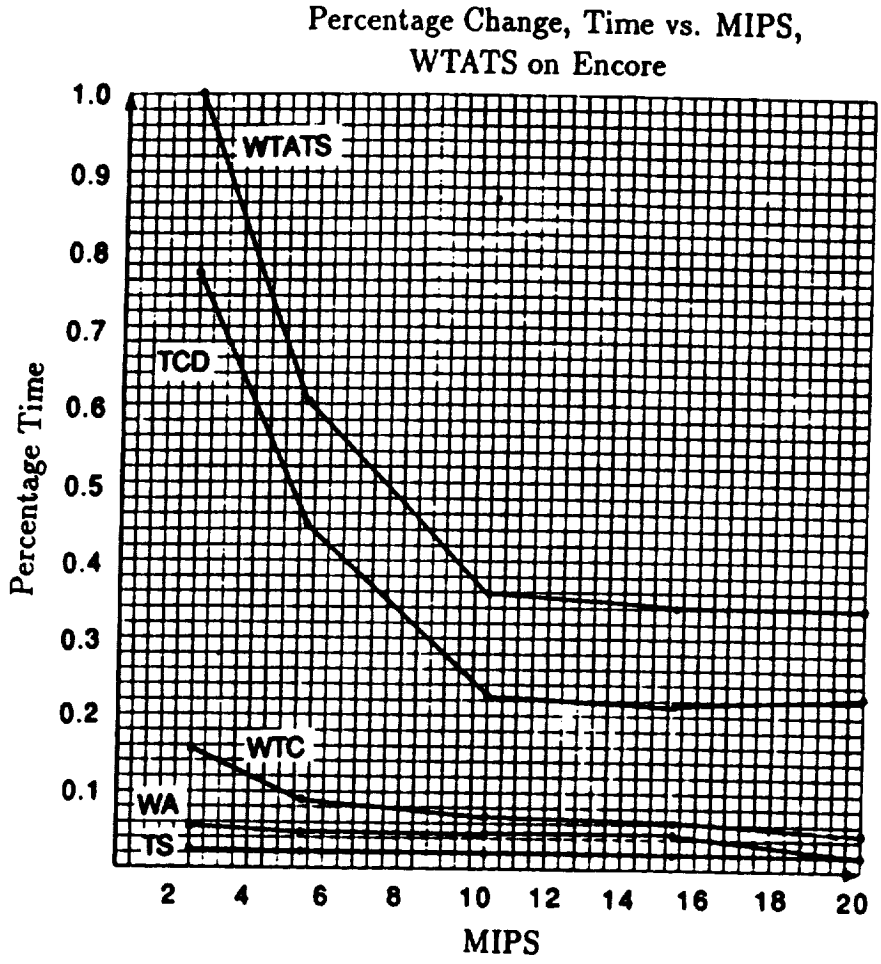


Figure 4.56. Latency vs. Processor Speed

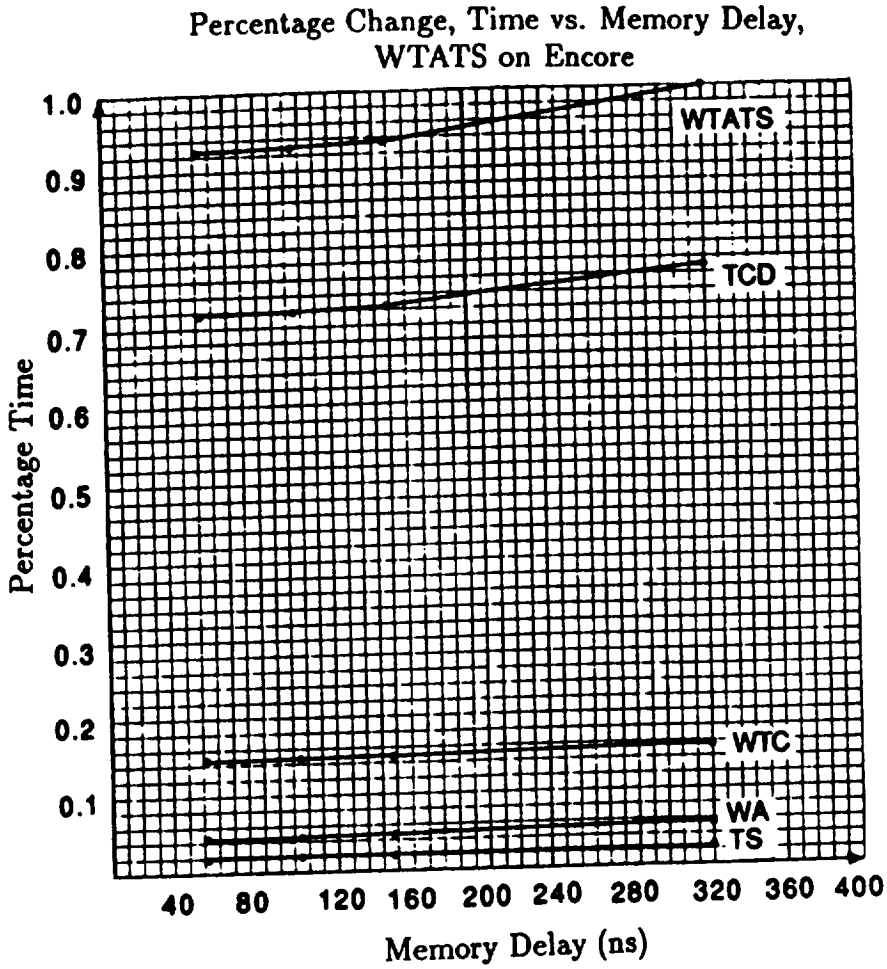


Figure 4.57. Latency vs. Memory Access Time

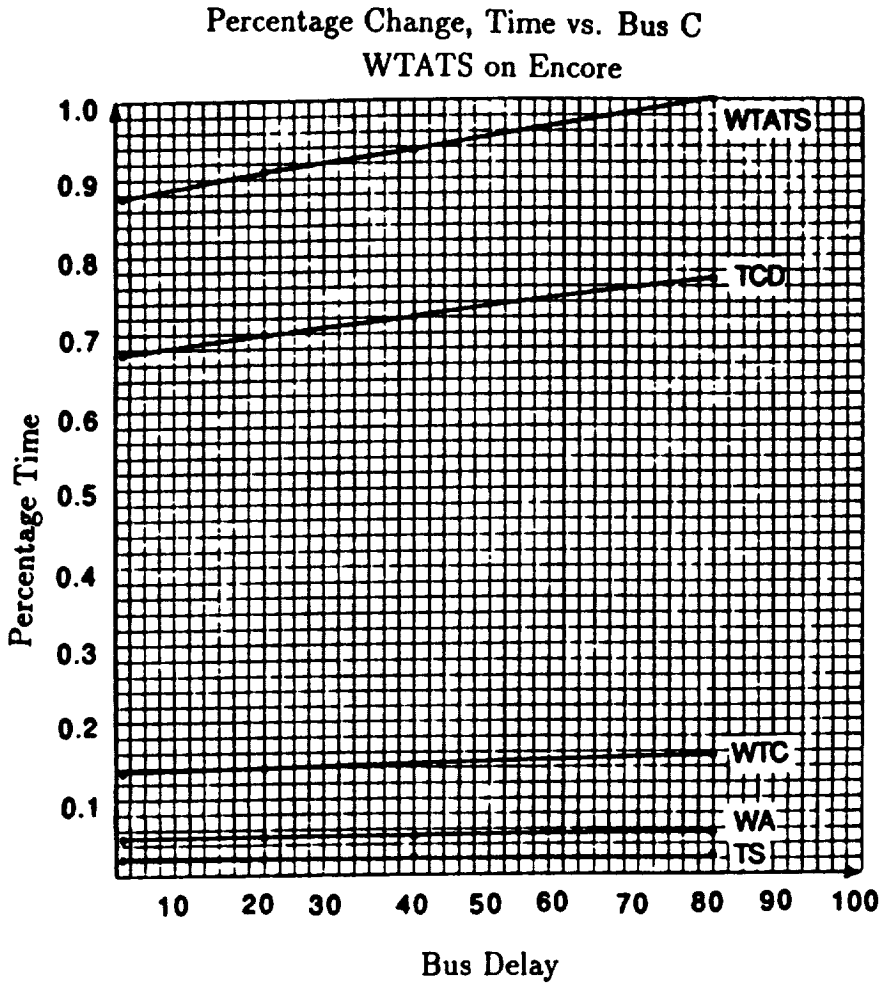


Figure 4.58. Latency vs. Bus Speed

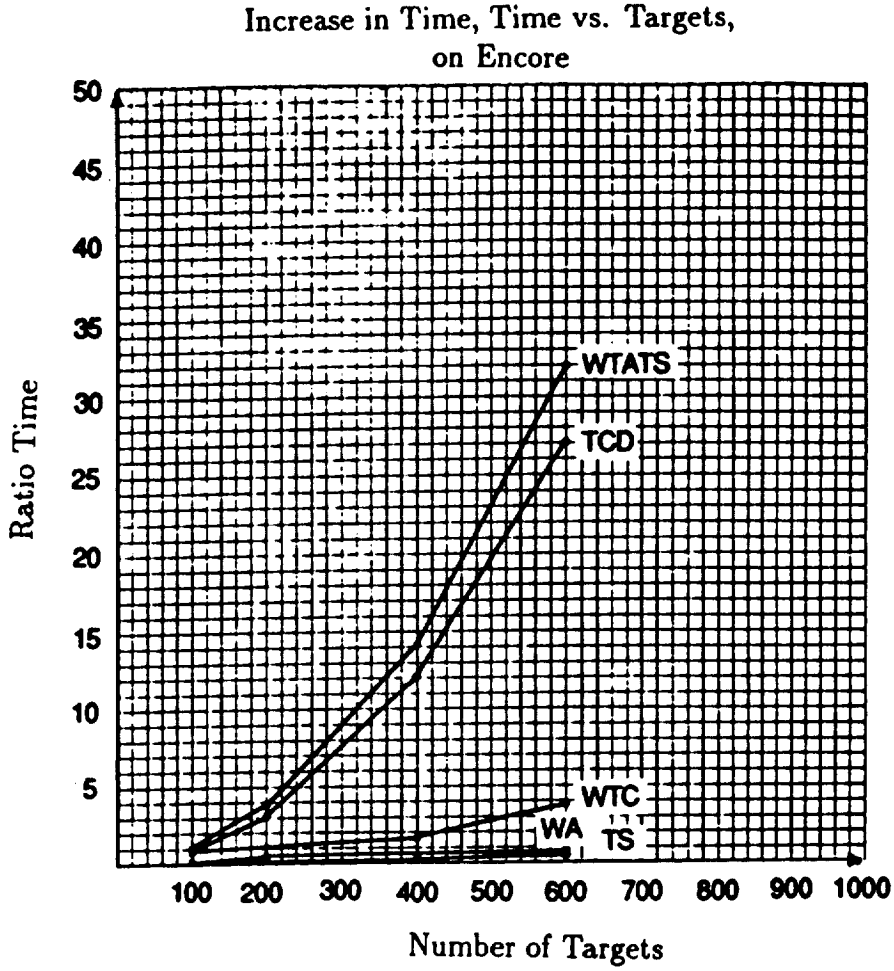


Figure 4.59. Latency vs. Number of Targets

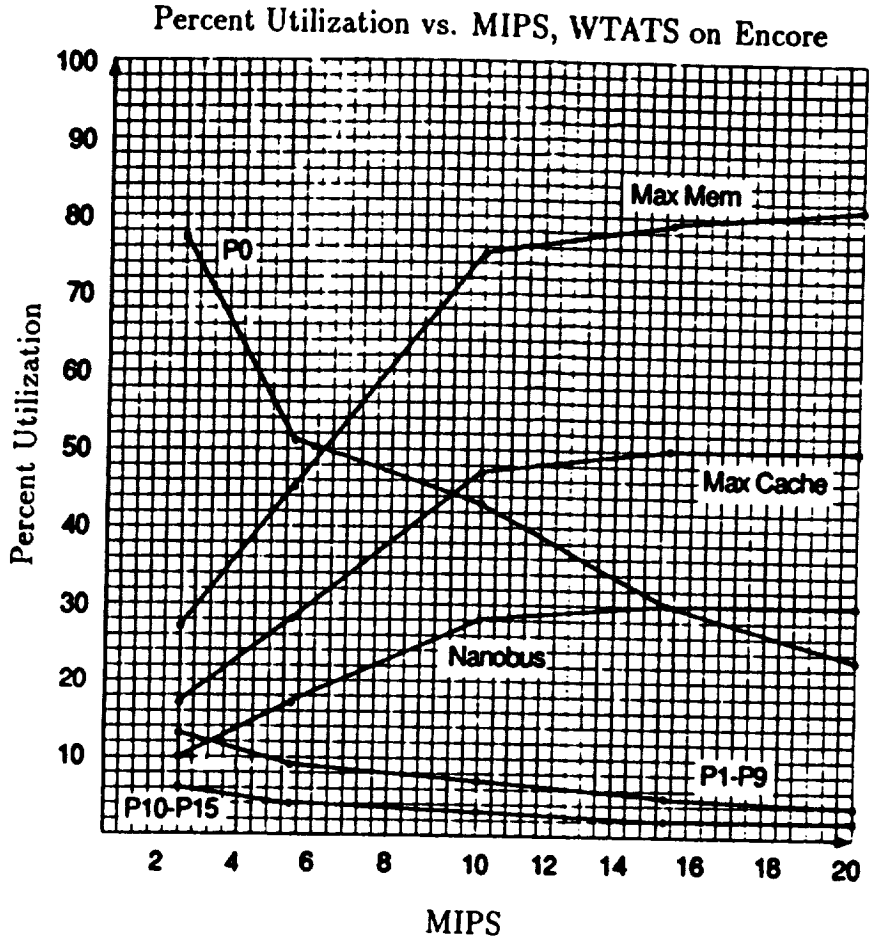


Figure 4.60. Utilization vs. Processor Speed

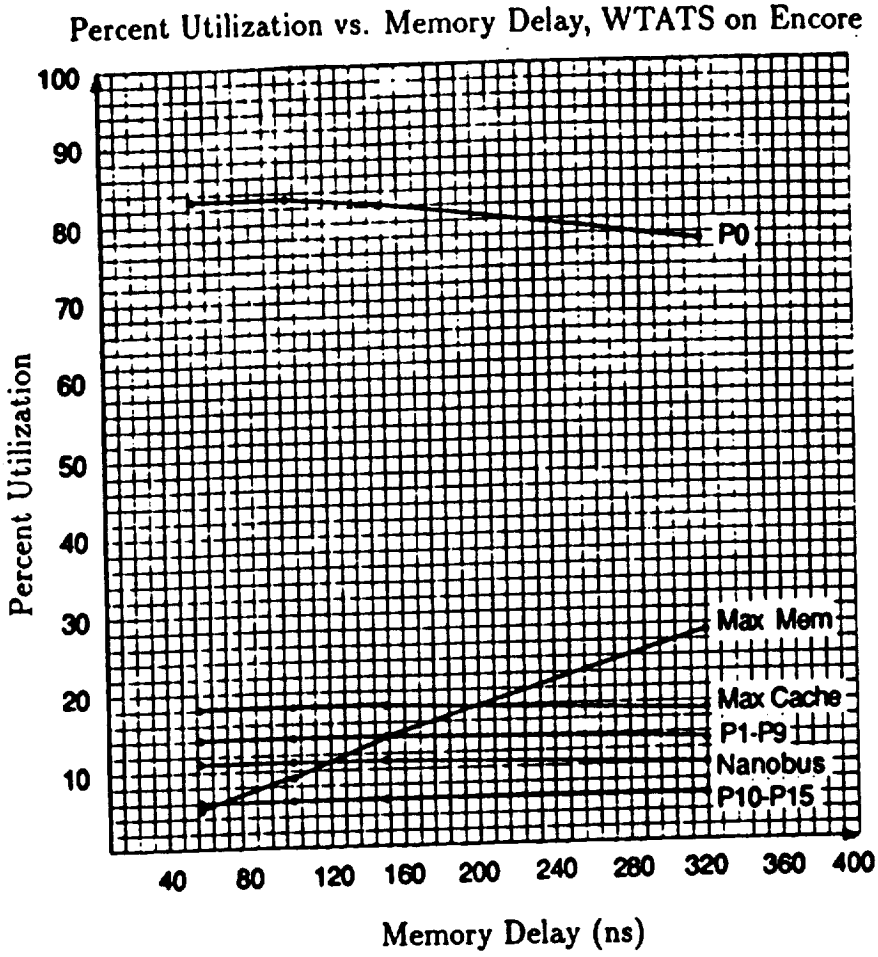


Figure 4.61. Utilization vs. Memory Access Time

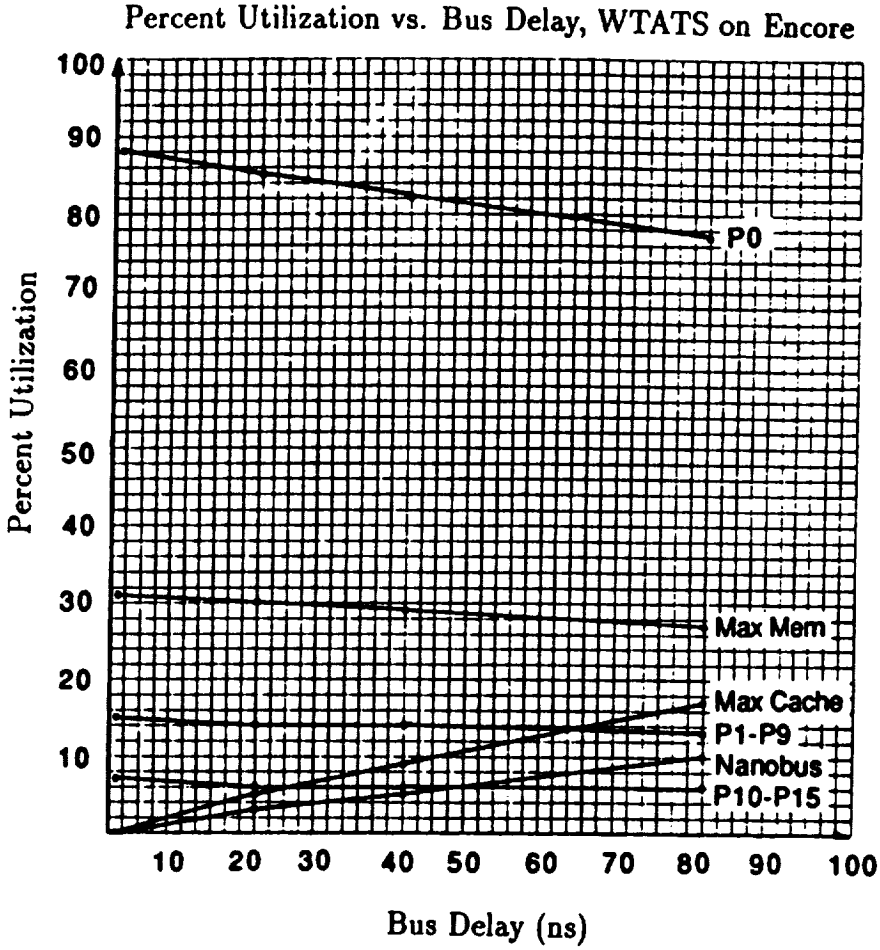


Figure 4.62. Utilization vs. Bus Speed for Processors P0 through P15

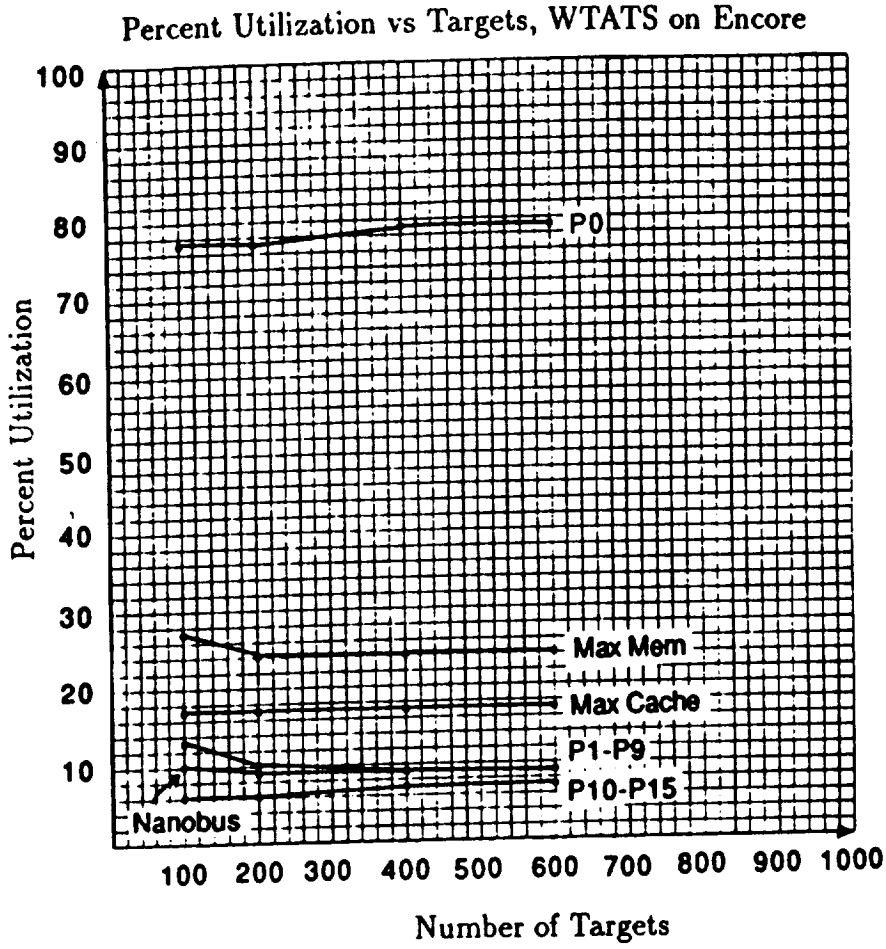


Figure 4.63. Utilization vs. Number of Targets

Figures 4.56 through 4.59 echo the trends of the Mark III performance data. In Figure 4.60, Max Mem and Max Cache refer to the maximum utilizations of the memory modules and caches respectively. These values level out at a processor speed of about 10 Mips, indicating faster processors saturate the cache. The remaining figures essentially indicate trends where slowing some component of the architecture increases its utilization. The data in Figures 4.56 through 4.63 are reproduced in the tables in Figure 4.64 with the percentage improvement afforded by each architectural feature. The first table shows the execution time in seconds for the baseline case of 2 MIPS, 100 targets, 320ns memory delay, and 80ns bus delay. The second table shows the percentage change in time t_i from the baseline time t_0, where

$$\Delta\% \equiv \frac{t_0 - t_i}{t_0}$$

It should be noted that each of the numbers in the tables reflect the effect of *one* parameter on the baseline. These entries do not reflect cumulative effects or the simultaneous effect of a number of different parameters. Thus, if one needed to achieve a 40% improvement in execution speed over the baseline configuration, it would be possible to select entries from the table that totaled 40%. These entries in turn would dictate the required values of architectural parameters.

Task and Module Execution Time in Seconds

MIPS	2	5	10	15	20
TCD	8.882	5.157	2.597	2.462	2.600
WTC	1.777	1.028	0.780	0.709	0.573
WA	0.613	0.537	0.526	0.528	0.538
TS	0.278	0.240	0.229	0.223	0.218
WTATS	11.550	6.962	4.132	3.922	3.928
Memory Delay	320ns	150ns	100ns	50ns	
TCD		8.349	8.295	8.264	
WTC		1.697	1.687	1.683	
WA		0.526	0.523	0.521	
TS		0.221	0.215	0.211	
WTATS		10.793	10.720	10.678	
Bus Delay	80ns	40ns	20ns	1ns	
TCD		8.311	8.058	7.768	
WTC		1.648	1.604	1.547	
WA		0.580	0.580	0.577	
TS		0.262	0.241	0.240	
WTATS		10.800	10.483	10.112	
Targets	100	200	400	600	
TCD		35.12	138.33	311.6	
WTC		6.05	18.62	43.4	
WA		1.19	2.40	4.5	
TS		0.98	3.78	8.7	
WTATS		43.33	163.12	368.2	

Percent Change in Execution Time from Baseline

MIPS	2	5	10	15	20
TCD	100%	-41.9	-70.8	-72.3	-70.7
WTC	100%	-42.1	-56.1	-60.1	-67.8
WA	100%	-12.4	-14.2	-13.9	-12.2
TS	100%	-13.7	-17.6	-19.8	-21.6
WTATS	100%	-39.7	-64.2	-66.0	-66.0
Memory Delay	320ns	150ns	100ns	50ns	
TCD		-6.0	-6.6	-7.0	
WTC		-4.5	-5.1	-5.3	
WA		-14.2	-14.7	-15.0	
TS		-20.5	-22.7	-24.1	
WTATS		-6.6	-7.2	-7.5	
Bus Delay	80ns	40ns	20ns	1ns	
TCD		-6.4	-9.3	-12.5	
WTC		-7.3	-9.7	-12.9	
WA		-5.4	-5.4	-9.1	
TS		-5.8	-13.3	-13.7	
WTATS		-6.5	-9.2	-12.5	
Targets	100	200	400	600	
TCD		295.4	1457.4	3408.7	
WTC		240.5	947.8	2342.3	
WA		94.1	291.5	634.1	
TS		252.5	1259.7	3029.5	
WTATS		275.2	1312.3	3087.9	

Figure 4.64. Performance of the Encore Multimax

in detail in [10]; however, this algorithm is not of interest in the present context.

Given the above system, two alternative algorithms for recovery from the occurrence of faulty cells were investigated. Since the control plane controls the system configuration, it is the area of interest. The control plane cells must reconfigure the system to perform correctly in the presence of the faulty cells. Given that a fault has been detected, by whatever means, by neighboring cells, no evaluation of the fault detection mechanism is made at this time.

4.7.2. Description of ADAS Model

The evaluation of the recovery algorithms was conducted using an ADAS model of the architecture. The array model (Figure 4.69) used in the simulation is a seven-by-seven matrix representing an execution array of forty-nine cells. This is the main body of the array. In addition, there is a clock node which provides timing tokens to initiate each of the operational cycles of the array. These timing tokens are input to seven intermediate nodes, called split nodes, which provide tokens to each of the cell nodes in the execution array. The clock node produces one token each time increment, which is transmitted to the split nodes. The split nodes have firing delays set at zero, so they provide no time delay to the model. All other nodes have firing delays equal to one.

The clock node has seven output arcs, one to each of the split nodes. Each of the split nodes has seven output arcs, each of which is connected to one node in the execution array in the same row as the split nodes.

There are forty-nine cells in the array of the simulation model. They are numbered from zero to forty-eight, beginning with the upper left corner and increasing along the rows to cell number 48 in the lower left corner. Rows begin with cell numbers 0, 7, 14, 21, 28, and 35.

The nodes in the simulation array each have eleven input ports, or inports, numbered zero through ten and ten output ports, or outports, labeled zero through nine. The arcs delivering the clock tokens from the split nodes are connected to inport ten, with the other ten inports reserved for input from neighboring cells. All cell outports are connected to neighboring cells. The port assignment for cell arcs is

0.	Cell to the farwest	(CELL 0)
1.	Cell to the west	(CELL 1)
2.	Cell to the northwest	(CELL 2)
3.	Cell to the north	(CELL 3)
4.	Cell to the northeast	(CELL 4)
5.	Cell to the fareast	(CELL 5)
6.	Cell to the east	(CELL 6)
7.	Cell to the southeast	(CELL 7)
8.	Cell to the south	(CELL 8)
9.	Cell to the southwest	(CELL 9)
10.	Clock input	

Since the port number three is assigned to be both the inport and the outport communicating with the cell to the north, the following pairing results between inports and outports. Inport 0 receives the output of outport 5 of the proper cell. Inport 1 is connected to an outport 6, inport 2 to outport 7, inport 3 to outport 8, inport 4 to outport 9, inport 5 to outport 0, inport 6 to outport 1, inport 7 to outport 2, inport 8 to outport 3, and inport 9 to outport 4.

Arcs that make the above connections are routed to allow the graphical representation on the display to resemble as nearly as possible a drawing of the active connections during simulation. Arcs have the default color of orange when the graph is initially drawn on the screen, but the color attribute is modified by the CSIM programs when simulation begins. During simulation, active arcs will have a color of magenta, and inactive arcs will be gray. This gives the appearance almost identical to a hand-drawn representation of the array and makes debugging of the model very easy.

Simulation is begun when a token from the split nodes reaches a cell inport. The node-user-text attribute of each cell is set to "any," which means any token received at any inport will start the execution of the cell. All tokens are consumed when received. Firing delays, the time allotted for cell execution, are set to one time increment.

At first glance, the graph may appear strange. To achieve the desired appearance, all execution cells are three-by-three grids in size. This enables routing arcs into a cell boundary without connecting to a port. Using this technique, all arcs either enter the center of one of the four edges of the nodes and go to a grid crossing before being connected to a port, or they enter the corner of a cell to a grid point and then go to a port. This allows the graph to have a very symmetrical appearance when only the activated arcs are enabled.

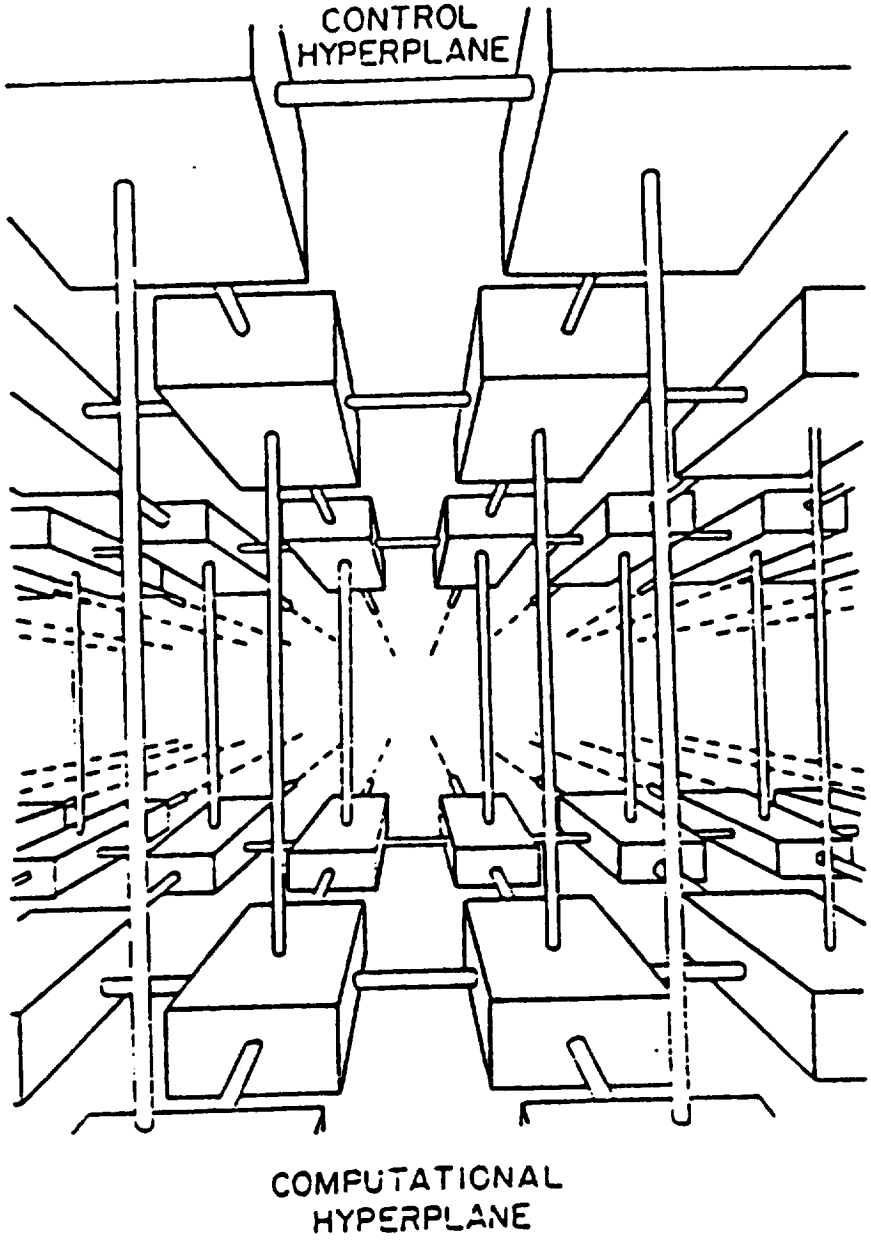


Figure 4.65. Fault-Tolerant Cellular Array [16]

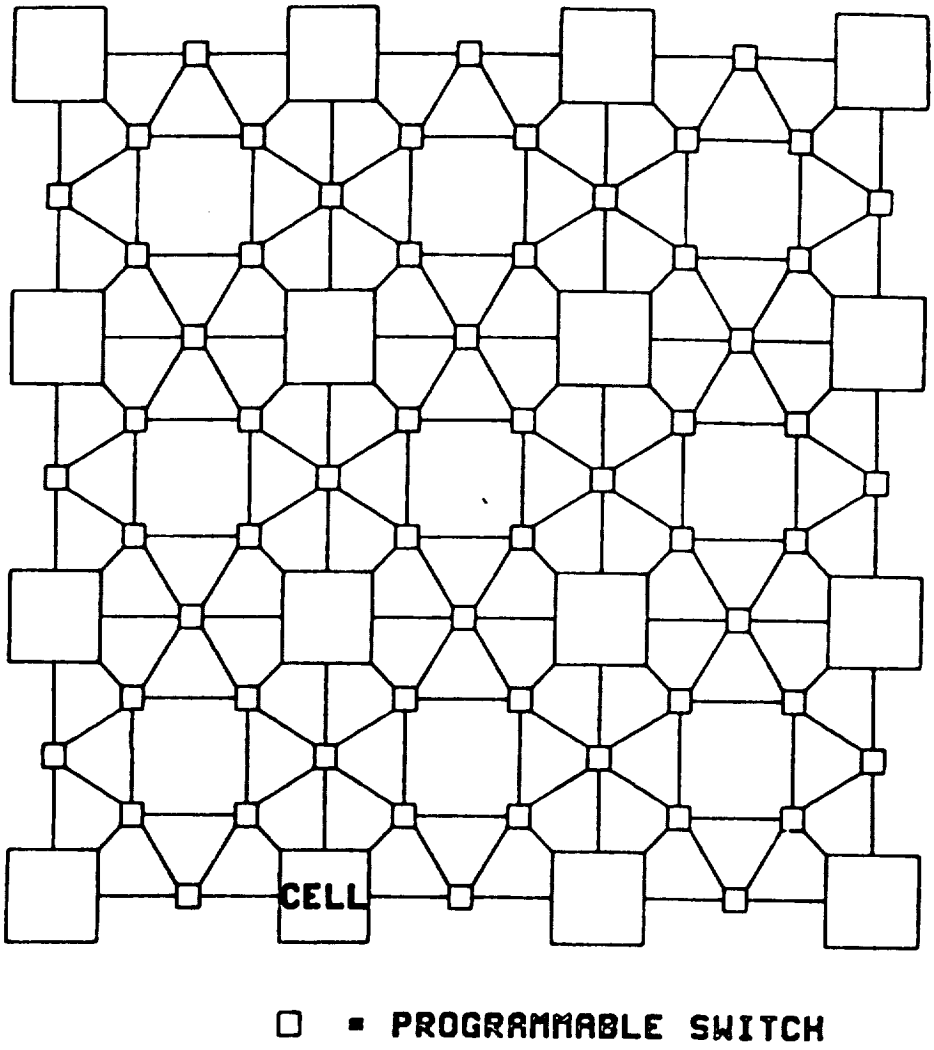


Figure 4.66. Computational Plane [16]

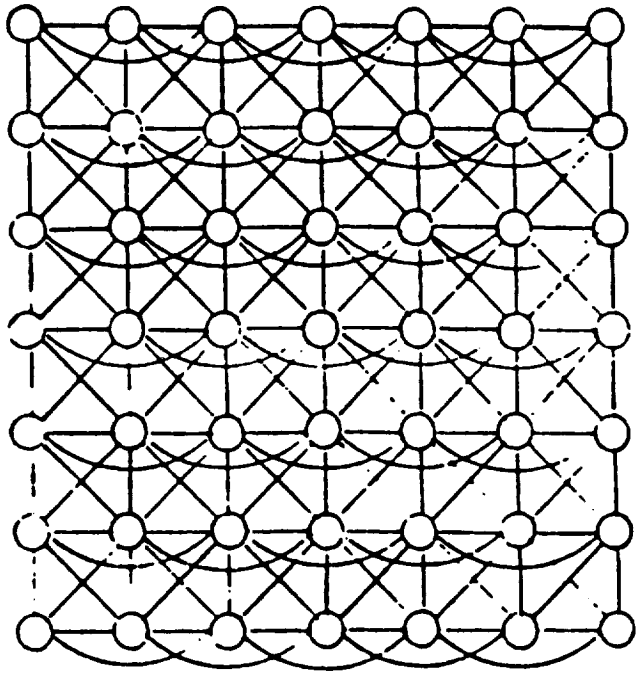


Figure 4.67. Control Plane

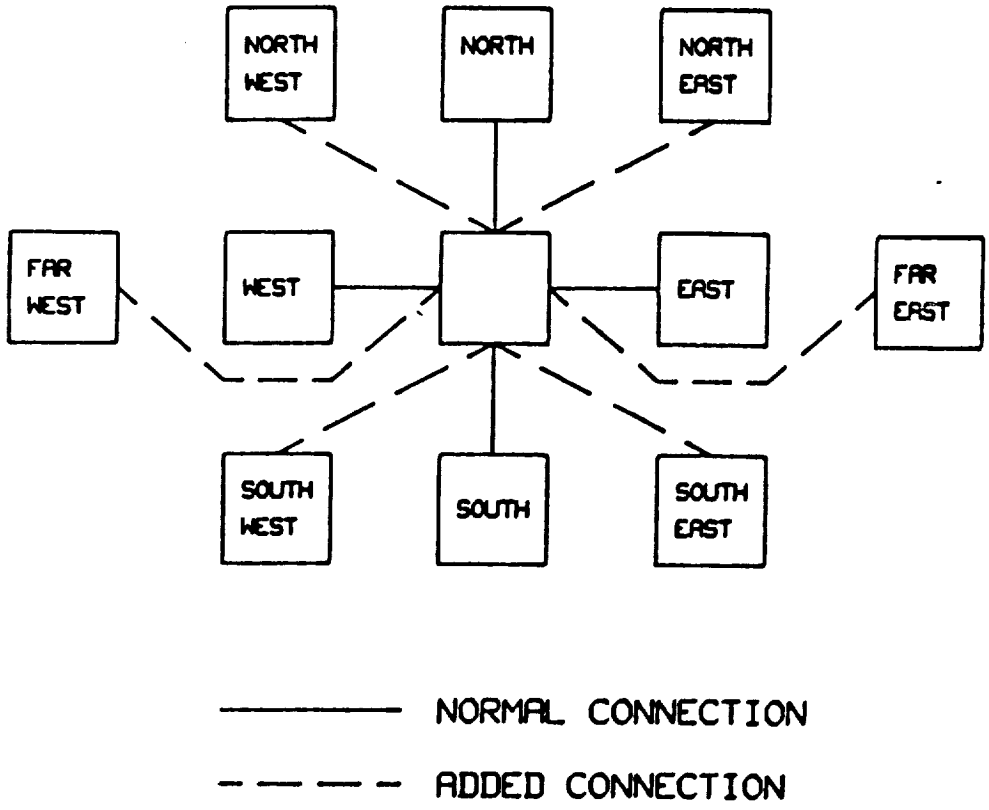


Figure 4.68. Connections to a Cell in the Control Plane [16]

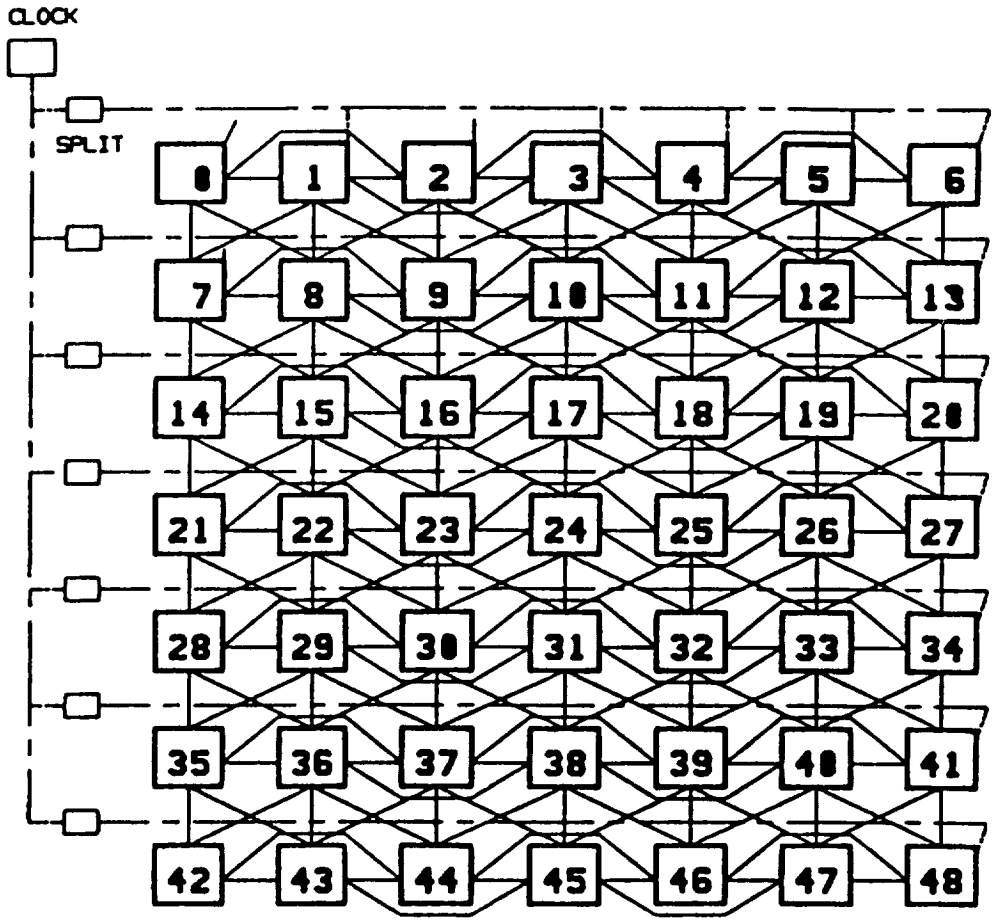


Figure 4.69. ADAS Simulation Model [16]

There are no bi-directional arcs to simulate the method of communications in the array, so two arcs must be used. The two arcs are routed in the same space outside the cell boundary. This practice was found very useful for checking configurations and debugging the algorithm. When the simulator is renewing the screen graphics, if an arc from cell A to cell B is enabled, but not from B to A, the results will be obvious. The screen is redrawn one cell and one arc at a time; by watching the graphics it is possible to tell when one cell has activated a port, and the other has not. Either the arc from A to B will initially be colored red when cell A is drawn and be changed to black when B is drawn, or the colors will change from black to red when the second cell is drawn.

A routine must be written in the C programming language for each node type. For this simulation, only three are required: clock, split, and cell.

The clock model does no functional operations. Therefore, the C routine for this node simply sets outputs and is the same for both algorithms.

The C routine for the split nodes also performs no functional operations; this program is used to generate necessary output to debug, verify, and collect data. The output generated is specific to each algorithm and is sufficient to determine any information needed to compare the two algorithms. This node is used differently in the two simulations. In the White-Gray model, the output is activated by setting the node-user-integer of one of the split nodes to a non-zero value. Usually split0 is used for this purpose. The split nodes are used to generate the output because they execute only when all cells have completed processing. Having the cells produce the output would require changing the reporting cell each time the model size is changed. The output is controlled by setting the user-integer in split0 to any number between one and seven. The output produced for each value of the integer can be selected by the user. In the Yanney-Hayes simulation, each cell node produces its own output; therefore, the split nodes are only used to produce tokens to drive the simulation.

The C routine for each cell defines the reconfiguration algorithm and is different for each algorithm being investigated. The C routine for the Yanney-Hayes algorithm is documented in Appendix B of [15]. The C routine for the White-Gray algorithm is documented in [17].

4.7.3. Yanney-Hayes Algorithm

The Yanney-Hayes algorithm works as follows:

- Each active node can detect all errors from a specified set occurring in its neighborhood. The detection process is assumed to be complete and not subject to failure.

- In response to detecting an error in node j, node i executes the following recovery strategy which is assumed to be fault free:

  1. If there is a spare node in the neighborhood of node i that has the correct connections to replace node j, then node i changes the state of the spare node to that of node j and the recovery is complete.
  2. If there are no spares in the neighborhood of node i that can assume the role of node j, then node i assumes the state of node j if it has the correct connections. This creates another error (the absence of state i) which will be detected by some other node. This other node will then execute the recovery strategy.
  3. If neither of the first two conditions holds, then the recovery attempt fails, and node i leaves the system state unchanged.

- An error at node j results in the removal, due to failure or a recovery step, of the state s_j from the system. It is the absence of s_j that is detected by the neighbor assigned to process errors in node j.

- The error sets detected by each node are disjoint. The union of all error sets is the set of all possible error conditions.

- The failure modes and the recovery strategy are constrained so that no more than one error at a time can be present in the system.

- When configuring its neighborhood in response to the absence of state s_j, a node either assigns state s_j to a spare node or changes its own state to s_j.

- The neighborhood of a node is restricted to adjacent nodes in the system graph.

This algorithm is described in more detail in [18]. To evaluate this algorithm, a C routine was written so that each node in the ADAS array performs the same procedure. See Appendix B of [15] for the complete program.

4.7.4. White-Gray Algorithm

The White-Gray Algorithm was developed in a thesis at Virginia Tech [17]. The ADAS simulation was done as part of that thesis and only the results are used here to provide a comparison with the Yanney-Hayes algorithm. This algorithm has two modes of operation, local and global. The local mode is activated when only a single fault occurs, or a multiple fault occurs in which there is no more than one faulty node in each row. The global mode is activated whenever more than one fault occurs in the same row of the active array.

In the local mode of operation, each cell passes the contents of a ten-bit register to each of its four immediate pattern neighbors (two in the same row and two in the same column of the pattern). These neighbors may not be in the same physical row or column if a prior reconfiguration has occurred. Based on the information received, each node updates its ten-bit register and passes the new value. This continues until equilibrium is reached.

At any time that a node discovers two or more faults in a row, the global mode is initiated. In the global mode, the array is cleared, a new seed is planted, a space detection algorithm is executed, seed migration occurs, and a completely new pattern is grown in a different part of the array. This is the same algorithm developed by [10]. The details will not be given here.

For comparison purposes, the assumptions underlying the algorithm are given below:

- A faulty cell can not be relied upon to produce any valid output signals. For this reason, methods must be found to isolate the faulty cell without utilizing fault-free cells solely for this purpose. A switch-controlled communications scheme can serve this purpose.
- Wrap-around communications will not be used within the array. It is felt that this adds communications paths to the array that severely limit array timing, especially one of wafer-scale size. Therefore wrap-around communications is not considered.

- The array implementing this algorithm will have at least one column of spare cells on the far eastern boundary.
- All cell faults are permanent. No provision is made for intermittent faults, although a slight modification to the algorithm would cover this possibility. Retesting a faulty cell and re-enabling the communications network should be investigated.
- All cell failures are independent. No single cell failure will cause other cells to fail.
- Each cell tests all cells with which it has active communications.
- Testing will be sufficient to determine whether a fault is internal to a cell, or in the communications network. In the event of a communications network failure, the testing algorithm must determine which cell to isolate.
- Communications between two cells within the array will be under the complete control of the two cells involved. No control signals will be generated, and no communications paths will be added, outside cell boundaries.
- All four cells communicating with a faulty cell will detect the fault during the same clock period.
- The cells in the control plane are used only to determine operations of execution cells. Programmable switches in the execution plane are either controlled by the execution cells, or are controlled by control cells in another plane.

4.7.5. Experimental Results

A three-by-three square array with four nearest neighbor connections was selected as the base graph to be embedded. The base graph, the global architecture with spare nodes and spare links, and the base graph embedded in the global architecture is shown in Figure 4.70.

To implement the Yanney-Hayes algorithm, an assignment of error detection responsibilities must be made. The algorithm itself does not specify how this should be done; the selected assignment is shown in Figure 4.71. This has not been verified to be an optimum assignment, but it performed better than any other assignment.

The two algorithms sometimes reconfigure in the same way, and sometimes differently. Figure 4.72 shows how each algorithm responds to a fault in cell 8. In this case, the final configuration is different. Figure 4.73 shows how both algorithms respond to a sequential double fault in cells 9 and 6 (a single fault in cell 6 occurs only after the system has correctly reconfigured in response to a fault in cell 9). In this case, both algorithms produce the same final configuration.

ADAS simulations were run for all single faulty cells and for all possible sequences of sequential double faults. A sequential double fault is defined as two single faults separated in time so that reconfiguration in response to the first fault has completed before the second fault occurs. For a description of the failure sequences and final configurations for all single faults and for selected sequential double faults using the Yanney-Hayes algorithm, see Appendices C and D of [15]. Results for the White-Gray algorithm are given in [17].

The first comparison criterion to be considered is coverage. Both algorithms cover all single faulty cells. The local mode of the White-Gray algorithm covers all coincident or near-coincident double faults that do not both occur in the same row. Coincident faults occur during the same clock period. Near-coincident faults occur close enough together so that the system is responding to the first fault when the second occurs. The Yanney-Hayes algorithm does not cover coincident or near-coincident double faults. Recall that one of the assumptions is that only one fault is active in the system at a time. Some sequential double faults are covered by the Yanney-Hayes algorithm and some are not. The local mode of the White-Gray algorithm will cover any sequential double fault in which both faults are not in the same row. The global mode of the White-Gray algorithm will cover all double faults, regardless of their position. A comparison of the local mode of the White-Gray algorithm with the Yanney-Hayes algorithm is made because the global algorithm is much more complex

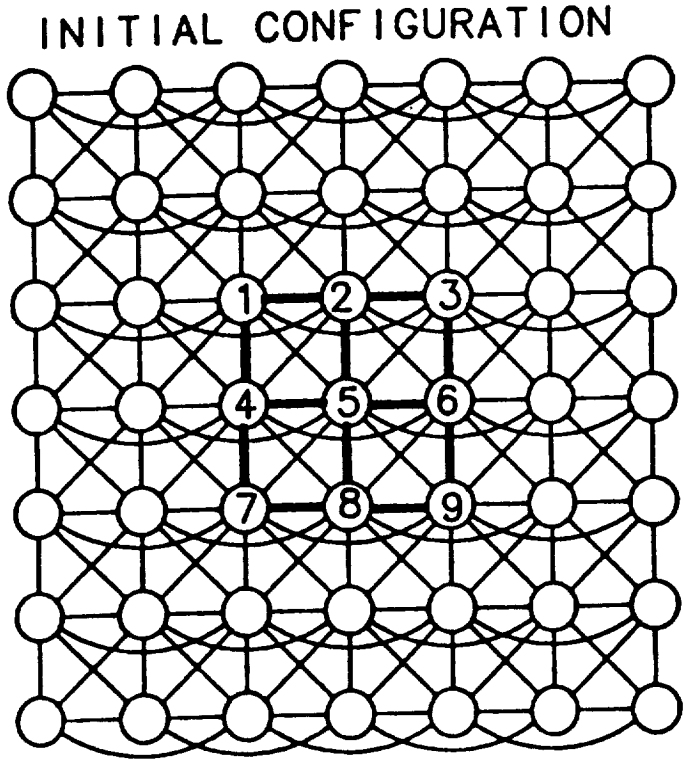


Figure 4.70. Base Graph Embedded in Global Architecture

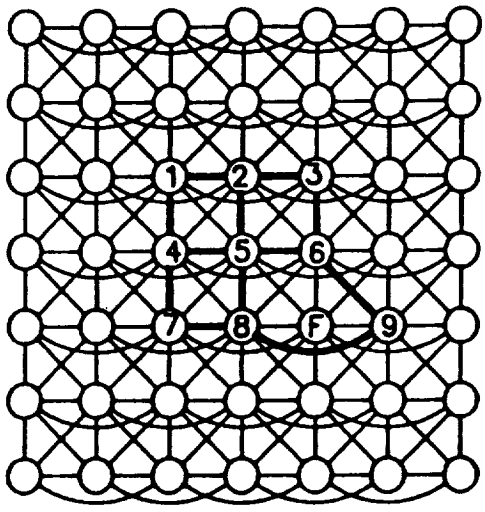
and time consuming than the local mode and it could be added as a second mode to any local algorithm (including Yanney-Hayes algorithm). Figure 4.74 contains a summary of the results. From the table, we conclude that the best algorithm relative to coverage depends on the relative probability of coincident and sequential double faults. White-Gray in local mode is much better for coincident double faults, but not as good for sequential double faults. In addition, there is one particularly troubling fault mode in the Yanney-Hayes algorithm for the sequential double fault E(9)E(9), where the algorithm goes into an infinite loop. Such faults would have to be identified in general and compensated for with some type of timer or counter.

Recovery time is another important attribute. A comparison of recovery times compiled from ADAS simulations is shown for single faults in Figure 4.75 and for double faults in Figure 4.76. The time unit of two-phase steps was used in both cases. In terms of two-phase steps, Yanney-Hayes is better than Gray-White in about 50% of the cases of sequential double faults and in about 33% of the single faults.

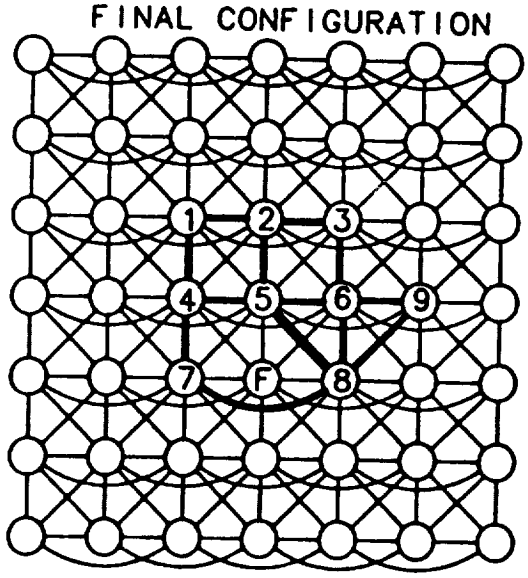
Although two-phase steps were convenient for simulation purposes, they have different interpretations for the two algorithms. In the White-Gray algorithm, a two-phase step consists of a serial exchange of ten bits of information in the first phase followed by a decoding of these bits and conditional setting of control signals in the second phase. The second phase is very short and takes only one PLA combinational logic circuit delay. On the other hand, for the Yanney-Hayes algorithm, the first phase consists of an exchange of state information (similar to that for White-Gray), but the second phase requires the execution of a complex decision algorithm using a significant amount of stored data that varies with time. We believe that the implementation of phase two will require a micro-controller of some type. Therefore, phase two will require a significantly longer time to complete than phase two of White-Gray. The conclusion is, therefore, that the Yanney-Hayes algorithm will require longer to execute in all cases than White-Gray.

CELL STATE	ERRORS CHECKED
1	—
2	E(1)
3	E(2)
4	—
5	E(4)
6	E(3), E(5)
7	—
8	E(7), E(9)
9	E(6), E(8)

Figure 4.71. Error Detection Assignment for Yanney-Hayes Algorithm



White-Gray



Yanney-Hayes

Figure 4.72. Reconfiguration for a Single Fault in Cell 8

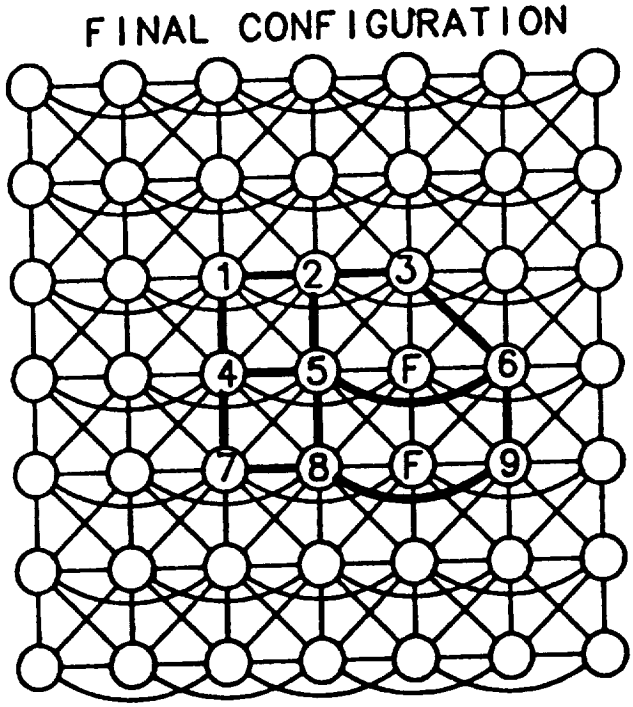


Figure 4.73. Reconfiguration for a Sequential Double Fault in Cells 9 and 6

Fault Class	YH	WG	
	Local	Global	
Single Faults	100%	100%	—
Coincident Double Faults	0%	75%	25%
Near-Coincident Double Faults	0%	75%	25%
Sequential Double Faults	83%	67%	33%

Figure 4.74. Summary of Coverage from ADAS Simulations

Error Condition	Faulty Cell	Two-Phase YH Steps	Two-Phase WG Steps
E(1)	16	1	3
E(2)	17	2	2
E(3)	18	1	1
E(4)	23	1	3
E(5)	24	2	2
E(6)	25	1	1
E(7)	30	1	3
E(8)	31	2	2
E(9)	32	1	1

Figure 4.75. Time Analysis of Single Faults from ADAS Simulations

Fault Sequence	Faulty Cells (YH)	Coincident or Near-Coincident Faults		Double Sequential Faults	
		Two-Phase YH Steps	Two-Phase WG Steps	Two-Phase YH Steps	Two-Phase WG Steps
E(1)E(1)	16,15	—	—	4	Fails
E(1)E(2)	16,17	Fails	Fails	Fails	Fails
E(1)E(3)	16,18	Fails	Fails	2	Fails
E(1)E(4)	16,23	Fails	3	2	6
E(1)E(5)	16,24	Fails	3	3	5
E(1)E(6)	16,24	Fails	3	2	4
E(1)E(7)	16,30	Fails	3	2	6
E(1)E(8)	16,31	Fails	3	3	5
E(1)E(9)	16,32	Fails	3	2	4
E(2)E(1)	17,16	Fails	Fails	Fails	Fails
E(2)E(2)	17,18	—	—	Fails	Fails
E(2)E(3)	17,19	Fails	Fails	3	Fails
E(2)E(4)	17,23	Fails	3	3	5
E(2)E(5)	17,24	Fails	2	4	4
E(2)E(6)	17,25	Fails	2	3	3
E(2)E(7)	17,30	Fails	3	3	5
E(2)E(8)	17,30	Fails	2	4	4
E(2)E(9)	17,32	Fails	2	3	3
E(3)E(1)	18,16	Fails	Fails	2	Fails
E(3)E(2)	18,17	Fails	Fails	Fails	Fails
E(3)E(3)	18,19	—	—	3	Fails
E(3)E(4)	18,23	Fails	3	2	3
E(3)E(5)	18,24	Fails	2	3	3
E(3)E(6)	18,25	Fails	1	2	2
E(3)E(7)	18,30	Fails	3	2	4
E(3)E(8)	18,31	Fails	2	3	3
E(3)E(9)	18,32	Fails	1	2	2
E(4)E(1)	23,16	Fails	3	2	6
E(4)E(2)	23,17	Fails	3	3	5
E(4)E(3)	23,18	Fails	3	2	4
E(4)E(4)	23,22	—	—	4	Fails
E(4)E(5)	23,24	Fails	Fails	Fails	Fails
E(4)E(6)	23,25	Fails	Fails	2	Fails
E(4)E(7)	23,30	Fails	3	2	6
E(4)E(8)	23,31	Fails	3	3	5
E(4)E(9)	23,32	Fails	3	2	4
E(5)E(1)	24,16	Fails	3	3	5
E(5)E(2)	24,17	Fails	2	4	4
E(5)E(3)	24,18	Fails	2	3	3
E(5)E(4)	24,23	Fails	Fails	Fails	Fails
E(5)E(5)	24,25	—	—	Fails	Fails
E(5)E(6)	24,26	Fails	Fails	Fails	Fails
E(5)E(7)	24,30	Fails	3	3	5
E(5)E(8)	24,31	Fails	2	4	4
E(5)E(9)	24,32	Fails	2	3	3

Figure 4.76. Time Analysis of Double Faults from ADAS Simulations

Fault Sequence	Faulty Cells (YH)	Coincident or Near-Coincident Faults		Double Sequential Faults	
		Two-Phase YH Steps	Two-Phase WG Steps	Two-Phase YH Steps	Two-Phase WG Steps
E(6)E(1)	25,16	Fails	3	2	4
E(6)E(2)	25,17	Fails	2	3	3
E(6)E(3)	25,18	Fails	1	2	2
E(6)E(4)	25,23	Fails	Fails	2	Fails
E(6)E(5)	25,24	Fails	Fails	Fails	Fails
E(6)E(6)	25,26	—	—	Fails	Fails
E(6)E(7)	25,30	Fails	3	2	4
E(6)E(8)	25,31	Fails	2	3	3
E(6)E(9)	25,32	Fails	1	2	2
E(7)E(1)	30,16	Fails	3	2	6
E(7)E(2)	30,17	Fails	3	3	5
E(7)E(3)	30,18	Fails	3	2	4
E(7)E(4)	30,23	Fails	3	2	6
E(7)E(5)	30,24	Fails	3	3	5
E(7)E(6)	30,25	Fails	3	2	4
E(7)E(7)	30,29	—	—	4	Fails
E(7)E(8)	30,31	Fails	Fails	Fails	Fails
E(7)E(9)	30,32	Fails	Fails	2	Fails
E(8)E(1)	31,16	Fails	3	3	5
E(8)E(2)	31,17	Fails	2	4	4
E(8)E(3)	31,18	Fails	2	3	3
E(8)E(4)	31,23	Fails	3	3	5
E(8)E(5)	31,24	Fails	2	4	4
E(8)E(6)	31,25	Fails	2	4	3
E(8)E(7)	31,30	Fails	Fails	Fails	Fails
E(8)E(8)	31,32	—	—	Fails	Fails
E(8)E(9)	31,26	Fails	Fails	3	Fails
E(9)E(1)	32,16	Fails	3	2	4
E(9)E(2)	32,17	Fails	2	3	3
E(9)E(3)	32,18	Fails	1	2	2
E(9)E(4)	32,23	Fails	3	2	4
E(9)E(5)	32,24	Fails	2	3	3
E(9)E(6)	32,25	Fails	1	2	2
E(9)E(7)	32,30	Fails	Fails	2	Fails
E(9)E(8)	32,31	Fails	Fails	Fails	Fails
E(9)E(9)	32,33	—	—	Loop	Fails

Figure 4.76. Time Analysis of Double Faults from ADAS Simulations (continued)

5. Reliability Analysis

The application characteristics of complex space missions that impact reliability modeling requirements are mission criticality, long operating life, distinct mission phases with diverse activity levels and reliability requirements, very high throughput requirements, a harsh operating environment, and strict constraints on volume, weight, and power.

The high system reliability requirements that exist for a mission critical application can only be met by a system that is both fault-tolerant and, in the event of system failure, fail-safe. Such systems have to be designed for fault tolerance and carefully evaluated to determine whether or not the requirements are met. However, the existence of fault tolerance mechanisms makes that evaluation more difficult by increasing the number and complexity of significant factors affecting system reliability. Likewise, increased complexity of system design and evaluation also result from the other application characteristics.

Since the system will have a long operating life, the MTBF's of major components will be less than the mission duration, and attrition of components via failures will be a dominating factor for system failure due to permanent faults. Consequently, techniques that extend system life, such as a pool of cool spares that have reduced failure rates and that can be activated upon failures, will be required and nonuniform architectures will result. Further, the extended periods without maintenance implies system complexity will be increased to permit self repair and redundancy management.

The multiple mission phases characterizing these applications require performance of different functions, are of widely varying duration, and are subject to different reliability requirements. The multiphase nature of the mission may also require mechanisms for dynamic resource management. During the low activity phases, adequate system resources should be available for self test and some components could be powered down to reduce power consumption and failure rates. The attainment of the desired reliability for the brief engagement phase is conditional upon the successful attainment of the lower reliability requirements of the long pre-engagement phase. During the short, but very active, engagement phase, fault coverage may be the dominant factor contributing to system failure and fault masking mechanisms will have to be employed.

The need for very high throughput will result in a large number of interconnected processors. The reliability analysis of such systems is made more difficult by the

number of components and the complexity of the interactions between them. It is also made more difficult by the need to evaluate complex networks, which requires identifying the very large number of combinations of link failures that lead to system failure.

The occurrence of faults in the system is accelerated by the harsh environment in which it will operate. Faults may be induced in the early phases by the stress of launch and thermal differentials. Natural and hostile radiation dosing will lead to transient faults throughout the mission and accelerated failure rates and intermittent faults at the end of the operating life.

Any component redundancy provided for reliability will have to be carefully justified since weight and power are at a premium. This will lead to more complex designs than one would encounter with simple redundancy techniques. For example, error correction hardware may be used on program memories and system state memories but limited error correction hardware may be provided for data memories. This leads to a more complex reliability model both in terms of the number of unique major components and in terms of the diversity of error rates.

For the purposes of defining the tools that are needed to evaluate highly parallel, high performance systems for these complex space missions, the large system problem, the impact of sequence dependencies and model complexity resulting from system configuration and management strategies to attain high reliability goals, and the interactions between the performance and reliability analyses are issues that need to be investigated. During this phase of the project, a selection of preliminary reliability analyses was devised to address these issues using models from the paradigm architectures. These preliminary analyses address two of the design phases specified in [13].

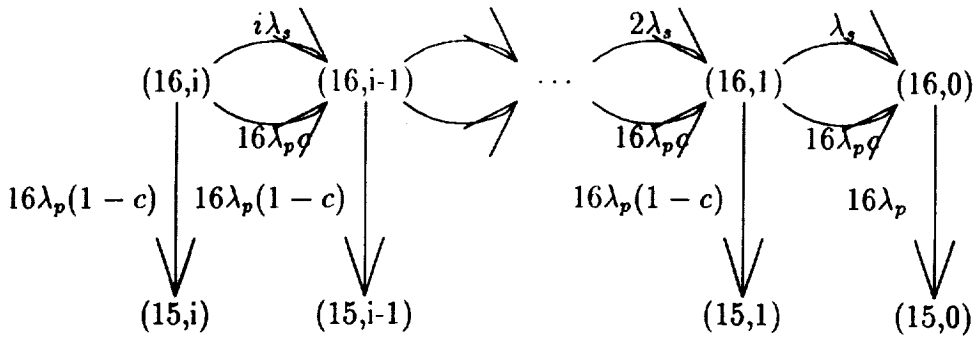
From the areas of reliability analysis that were identified in the paradigm framework (Section 3.3), a high-level analysis appropriate to the system requirements phase to determine sparing levels required to keep a system operational for a long mission; an analysis appropriate to the system design review of an FTPP cluster; and a network analysis were selected. These analyses are described below and were performed using reliability estimation tools provided by NASA Langley Research Center, including ASSIST, SURE, and the Scaled Taylor Exponential Matrix (STEM) program. SURE computes an upper and lower bound for the probability of entering a death state of a semi-Markov model containing arbitrary recovery transitions and STEM computes death state probabilities of Markov models based on the assumption that the distributions are exponential. ASSIST was used to create models in the input form required by SURE and STEM.

In the high-level analyses, the large number of components and spares required to maintain the system through the five-year mission resulted in a large model with long paths to the death states, even though the complexities of the recovery processes were not modeled. Also, the mission length was sufficient for exercising those paths. The number and length of the paths made the use of SURE infeasible due to excessive memory requirements and execution times. However, STEM, which has less modeling flexibility but is more computationally efficient for this type of problem, was able to input the same ASSIST-generated model and compute the system failure probability.

Since a detailed model of a large complex system such as the FTPP is unsolvable, model reduction techniques were of particular interest in the FTPP cluster study. One approach is to truncate the generation or solution of certain paths through the model based on reduced contribution to system failure. Aggregation of states based on complex definitions of state vectors can also be accomplished. The liability of these approaches results from the likely creation of what is only an approximate model under the guise of creating an exact model. Also, the model will very likely be difficult to verify or understand. Because of these difficulties with exact models, the approach used in the reliability analysis of the FTPP cluster was to create approximate, bounding models based on the underlying triad structure of the configuration chosen for the paradigm. The triad models were then combined and embellished to produce lower and upper bounds on the system failure probability of the FTPP.

5.1. High-Level Analyses

High-level analyses were performed of the type used to determine how many spares would be required to attain a mission goal of five years and how long mission life could be extended by accepting degraded performance. For the spares analysis, a 16 processor system was assumed with varying numbers of spares. Figure 5.1 shows the Markov model used for these cases. The input to STEM was created by the ASSIST file shown in Figure 5.2.



i = 16,25,32,36,40,50,64,100
34 states, 49 transitions to 202 states, 301 transitions

Figure 5.1. Spare Analysis State Model

For each case the probability of system failure ($P(SF)$) was computed for ($1 - c$) = 10^{-4} to 10^{-6}, where c is the probability that a failure is detected and the failed processor is successfully replaced by a spare. The models ranged in size from 34 states and 49 transitions to 202 states and 301 transitions. The results of the analyses are shown in Figure 5.3.

For the degraded performance analysis, a system of 16 processors and 16 spares was assumed. The $P(SF)$ of that system at five years was computed and then successive system models were built allowing the system to isolate the faulty processor(s) and continue, down to ten fault-free processors. Each degradable system was analyzed to determine how many operational years it could attain with the same $P(SF)$ as that of the non-degradable system at five years. The largest model contained 136 states and 359 transitions. The results of this analysis are shown in Figure 5.4.

The large number of components in parallel systems and the large number of spares required to maintain a system for a long mission result in large models with long paths through the system states. The long mission times and the long paths stress the reliability tools with large memory requirements and excessive execution times. The creation of the generic parameterized models via ASSIST made it much easier to vary the system parameters and thus to perform analyses for a range of those parameters. The analysis of extended mission time versus degraded performance illustrates a point of interaction between reliability and performance analyses.

```
INPUT nspares;                  (* Number of Spares *)
INPUT nprocs;                   (* Number of Processors *)
INPUT min_conf;                 (* Minimum Number of Processors Required *)
INPUT cov;                      (* Probability of Detecting and Reconfiguring *)
lambda_p = 2.5e-5;              (* Active Processor Failure Rate *)
lambda_s = 1.25e-6;             (* Spare Processor Failure Rate *)
"prune = 1.0e-20;"

SPACE = (np: 0..nprocs,             (* Number of Active Processors *)
         ns: 0..nspares);           (* Number of Available Spares *)

START = (nprocs,nspares);

DEATHIF (np < min_conf);

IF (np >= min_conf AND ns>0) THEN

   (* Spare Failure *)
   TRANTO (np, ns-1) BY ns*lambda_s;

   (* Failed Active Processor Replaced by Spare *)
   TRANTO (np, ns-1) BY np*lambda_p*cov;

   (* Transition to Degraded State Due to Coverage Failure *)
   TRANTO (np-1,ns) BY np*lambda_p*(1-cov);
ENDIF;

IF (np>=min_conf AND ns=0)

   (* Transition to Degraded State Due to Exhaustion of Spares *)
   TRANTO (np-1,ns) BY np*lambda_p;
```

Figure 5.2. Spare Analysis ASSIST File

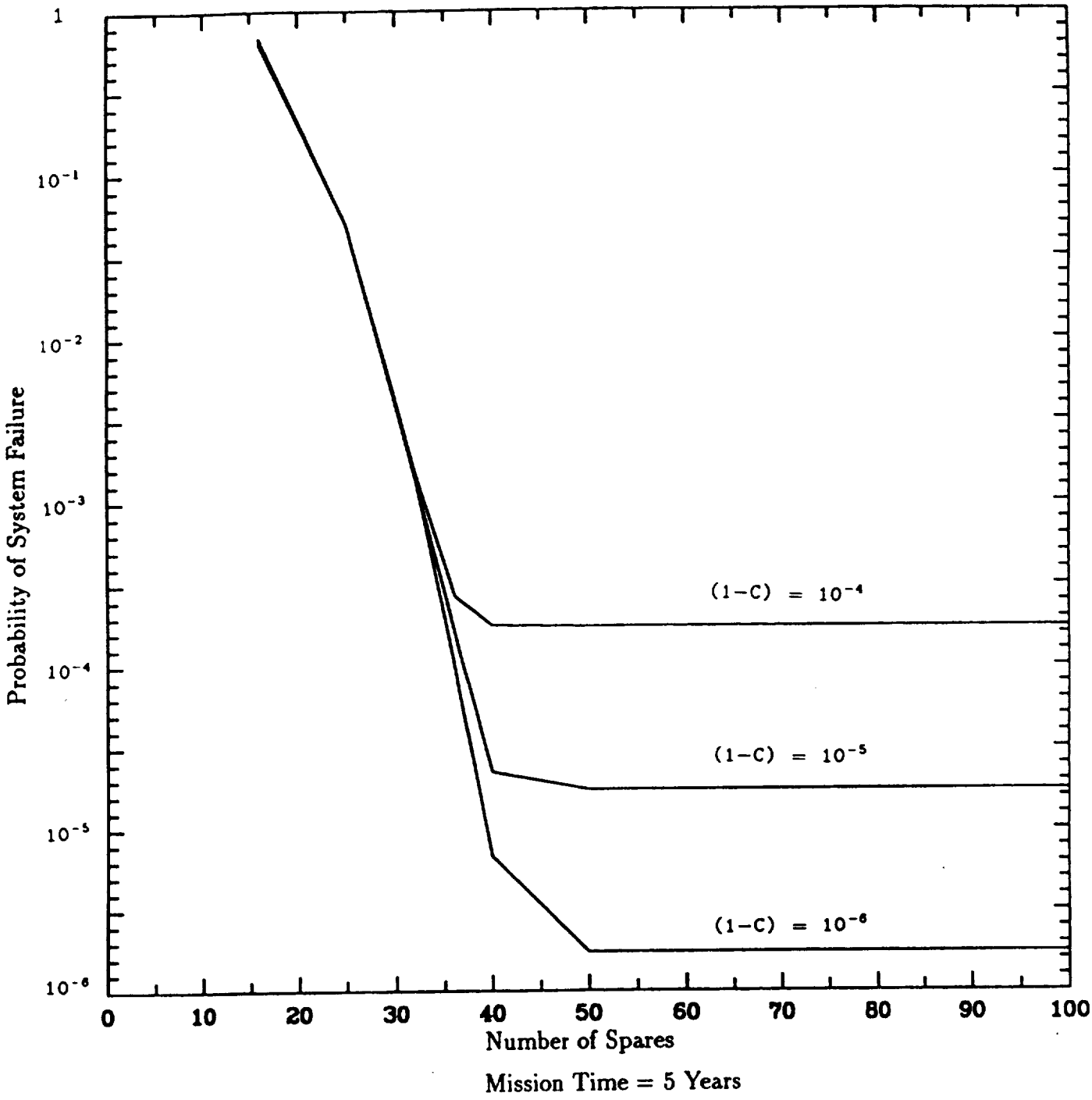


Figure 5.3. Spare Analysis Results

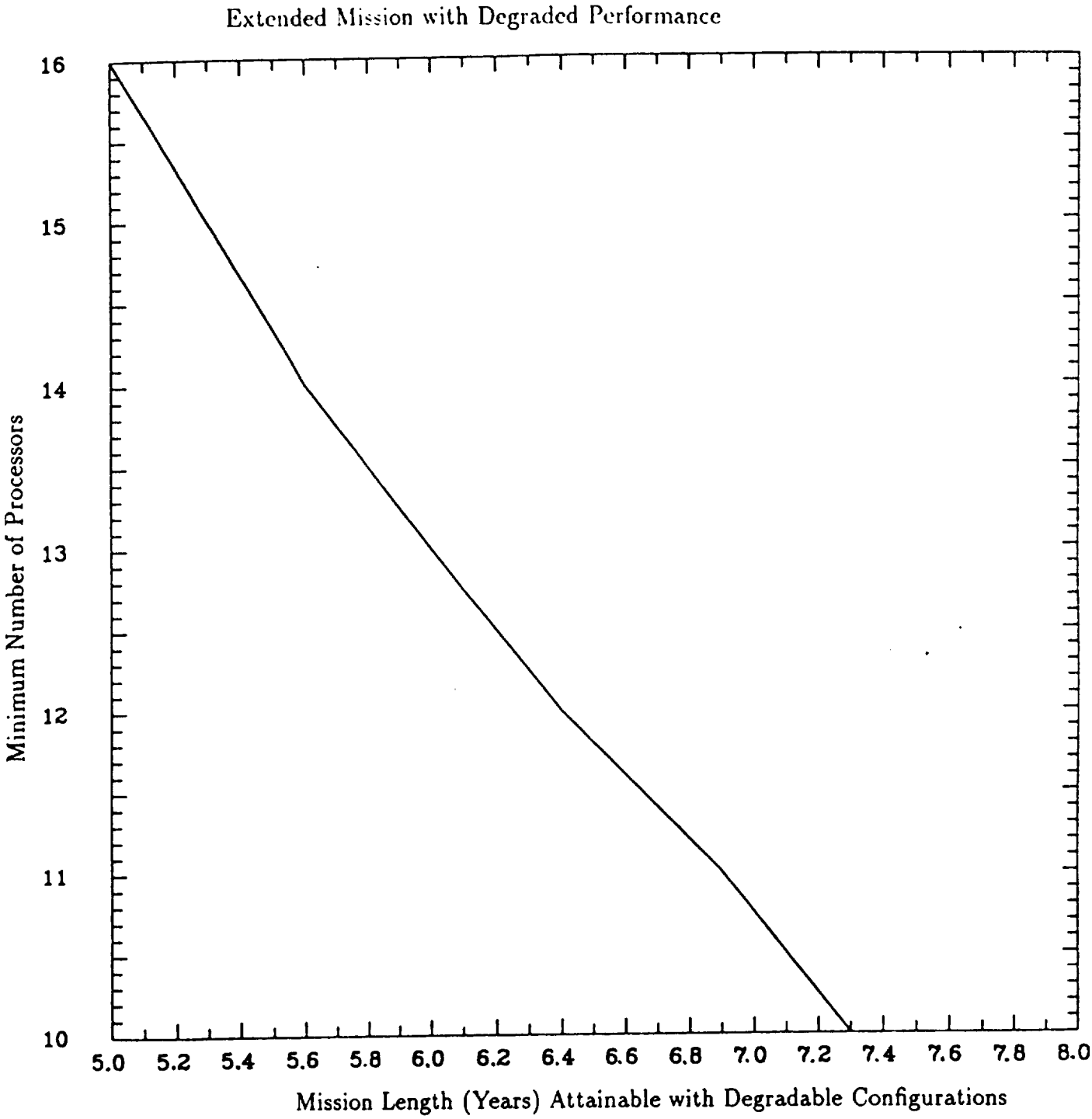


Figure 5.4. Extended Mission Analysis Results

5.2. FTPP Cluster Analysis

The FTPP was designed to achieve high performance in a system designed to meet stringent fault tolerance requirements. It has been described and evaluated in [7] and [9]. The primary emphasis of the Phase I FTPP analysis was to illustrate issues in constructing reliability models, not to assess the reliability of the FTPP. It was performed on the system configuration used for the performance analysis of WTA/TS to facilitate the future investigation of the interactions between performance and reliability analyses. The selected configuration consists of four interconnected clusters, where each cluster consists of four processing and four network elements. The processing elements in each cluster are configured in triads, with the three processors in each triad being distributed over three network elements. The network elements within a cluster are fully connected, and each cluster is connected to each of the other three clusters through special purpose input/output processing elements (IOEs). The cluster structure is illustrated in Figure 5.5 and the network of clusters in Figure 5.6.

In this analysis, which is an example of a preliminary analysis, no sparing or reconfiguration was modeled. Thus, for a cluster, system failure is the loss of any triad. The processors in a triad perform redundant processes and the results are voted to isolate a faulty processor. Thus, triad failure is the failure of any two of the component processors. For the analysis, two processor failure rates were considered: 1E-5 and 5E-5 + 1E-6. The latter failure rate includes the failure rate of the link from the processor to the network element. Likewise, two network element failure rates were considered: 1E-6 and 1E-5 + 1E-6. A mission time of one-half hour was assumed. The reliability analysis was partitioned into an analysis of a cluster and an analysis of the network of clusters.

A complete Markov model of a cluster consisting of 4 network elements and 12 processing elements contains more than $2^4 \cdot 2^{12}$ states. The distribution of the triad components over the network elements introduces sequence dependencies between NE and PE failures, which further complicate the model. The size and complexity of the system thus make it infeasible to create and attempt to solve a complete model, so reduction techniques have to be considered. Depending on the tools available for construction and analysis, either the construction of the model can be simplified or the solution of the model can be truncated.

One common technique to create smaller, simpler models is to decompose a system, if possible, into independent subsystems that can be solved separately and the results combined to assess the complete system. Another frequently used method to reduce the number of states in a model is to aggregate states based on a common failure

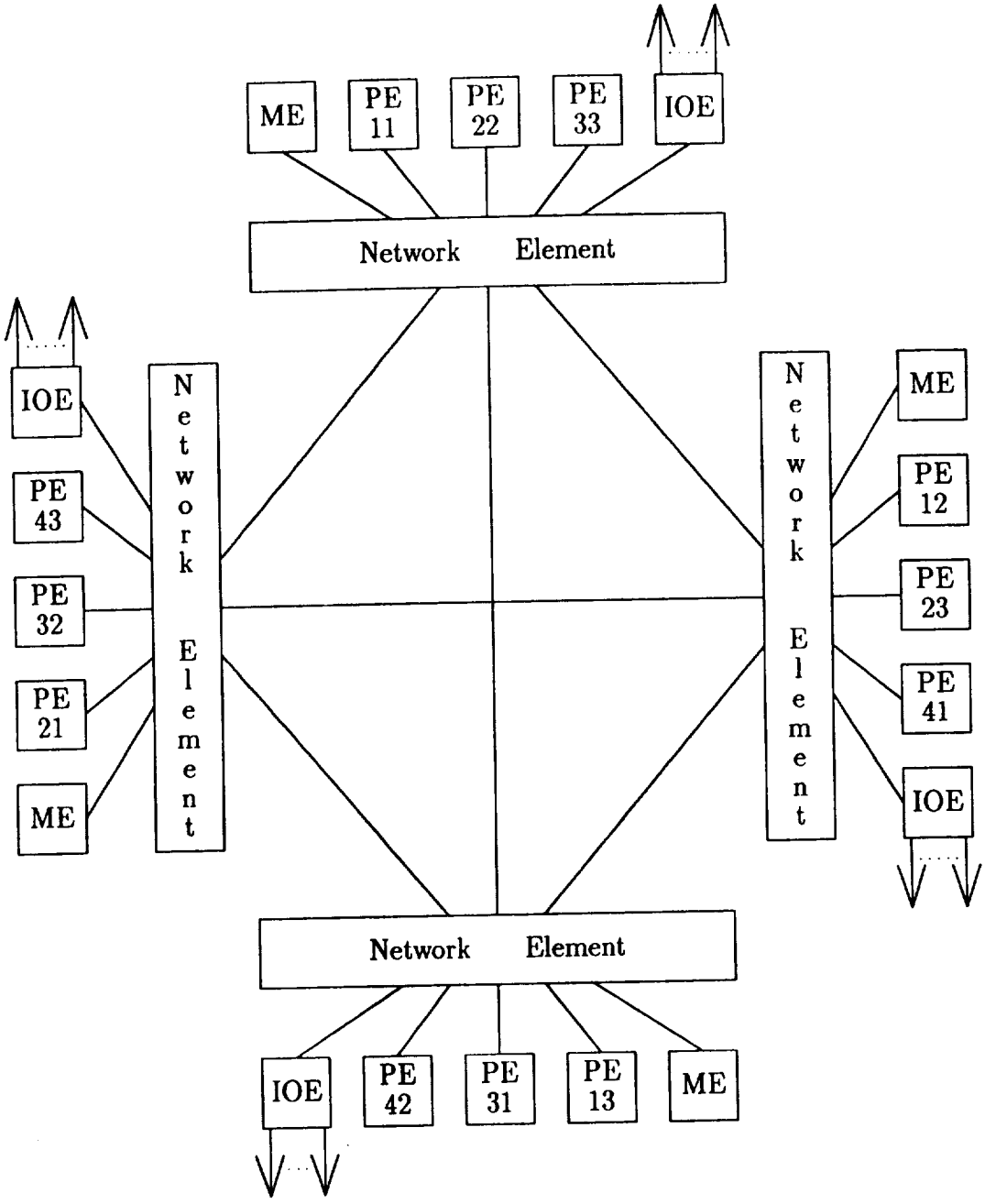


Figure 5.5. Cluster Analysis System Configuration

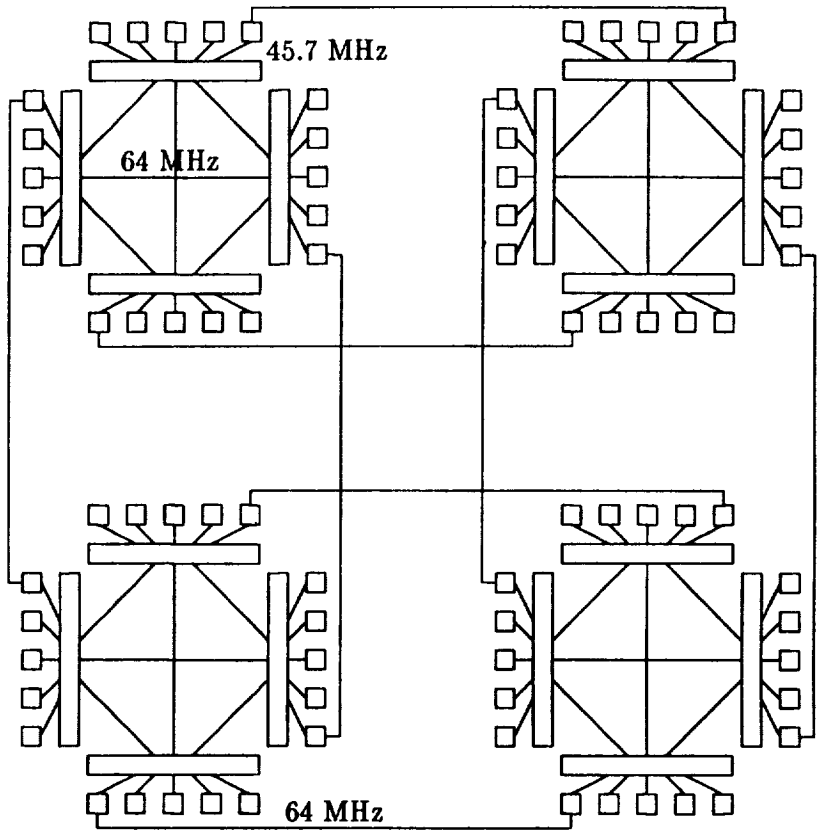


Figure 5.6. Cluster Network Topology

aspect, such as total number of faults existing in the system or the number of active processes. That is, states that have a common feature relevant to the operational state of the system but are reached by different transitions from possibly different states can be combined. The rates of the transitions associated with the combined states can in turn be combined and new transitions created.

Some truncation mechanisms are usually included in the tools to allow states and transitions in a model to be ignored if their probability of occurrence falls below a specified floor, and thus do not significantly impact system reliability. ASSIST also allows other user-specified model truncations to be effected by death state definitions.

Finally, bounding models can be created to approximate system reliability. That is, a tractable model of a system is built which can be shown to have greater or less system reliability than the actual system under consideration. If optimistic and pessimistic models within a sufficiently close range of reliability can be built and solved, then the actual system reliability can be known within acceptable bounds. An optimistic bounding model can be built, for example, by relaxing constraints on reconfiguration and eliminating some of the transitions to degraded, operational states. Likewise, a pessimistic model can be built by limiting reconfiguration possibilities and by creating some degrading transitions.

There are various approximate models that can be used to bound the $P(SF)$ of the FTPP. The most optimistic approximation is that of four independent triads. Adding four independent network elements produces another, less optimistic, model. On the other hand, a conservative approximation can be made by a model that captures interdependencies between the triads and the network elements, but does not distinguish all of the failure sequences that lead to continued operation rather than failure.

The probability of failure, $P(F_{T_1})$ of a triad for time = 10 hours is computed by SURE to be between 7.49994E-11 and 7.50000E-11. Since the four triads are assumed to be independent, the probability of system failure is

$$P(SF) \approx P(F_{T_1}) + P(F_{T_2}) + P(F_{T_3}) + P(F_{T_4})$$

Thus, 2.99997E-10 $< P(SF) <$ 3.00000E-10. However, for the purposes of considering the methods of reducing models, it is instructive to consider a full model of a system of four independent triads.

The Markov model shown in Figure 5.7 consists of states that represent the occurrence of faults on specific triads. States 3 through 6 represent the occurrence of one fault on one triad; states 7 through 16 represent the occurrence of two faults, either two on one

triad (failure states denoted by squares) or on two triads (active states denoted by circles); and so on to states 46 through 49 (five failures). For this model, the probability of system failure was computed to be 2.99997E-10 $< P(SF) <$ 3.00009E-10.

This model can be reduced by aggregating states according to the number of failures that have occurred and the distribution of those failures across triads, as illustrated by the first diagram in Figure 5.8.

This aggregated model can be further reduced to that illustrated in the second diagram by aggregating the transitions. For this reduced model, system failure was computed to be 2.99997E-10 $< P(SF) <$ 3.00009E-10. The addition of four independent network elements to the system results in a computed system failure of 3.01497E-10 $< P(SF) <$ 3.01509E-10.

An approximate model was built that captured interdependencies between the triads and the network elements; however, this model overestimated the loss of processors by aggregating certain combinations of network element and processor failures in worst-case states. This model contained 130 states and 184 transitions; the probability of system failure was computed to be 3.76495E-10 $< P(SF) <$ 3.76510E-10. Thus, since this is a conservative estimate and that of the independent triads and network elements is optimistic, the true lower bound on $P(SF)$ is between 3.01497E-10 and 3.76495E-10, and the true upper bound is between 3.01509E-10 and 3.76510E-10.

Figure 5.9 shows the approximate model reduced to 6 states and 10 transitions. In this reduced model, the states represent the number of degraded triads and the number of full triads, and the transitions are due to combinations of processor and network element failures. The probability of system failure was computed to be 3.76825E-10 $< P(SF) <$ 3.76840E-10.

The $P(SF)$ of each model as computed by SURE for a half-hour mission, assuming a processor failure rate of 1E-5 and a network element failure rate of 1E-6 and assuming a processor failure rate of 5E-5 + 1E-6 and a network element failure rate of 1E-5 + 1E-6, is summarized in Tables 5.1 and 5.2 respectively. Note that the higher failure rates also include the failure rates of the associated links.

5.3. Network Analysis

The FTPP network configuration used in the performance analyses is illustrated in Figure 5.6. It consists of four clusters interconnected so that each network element

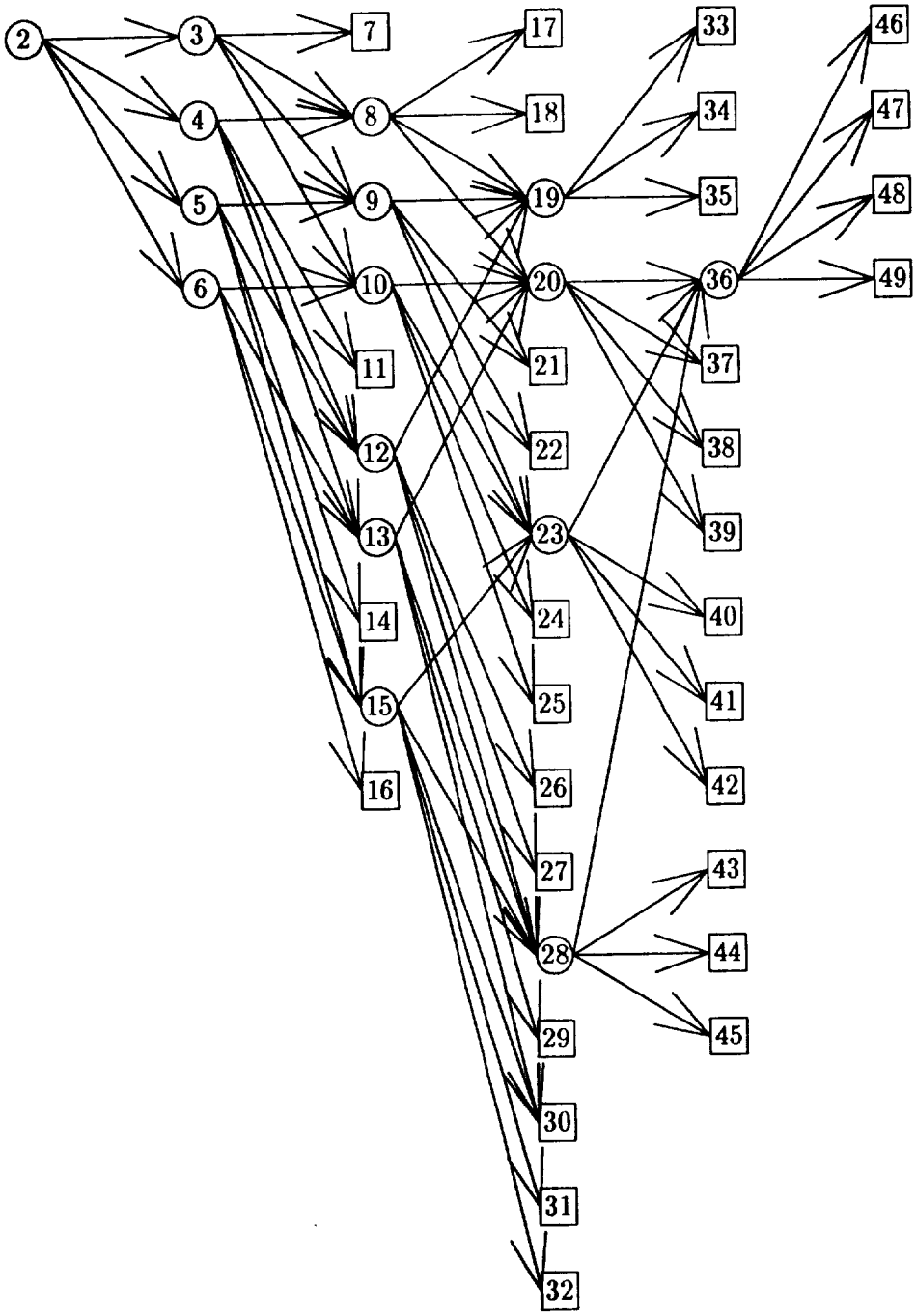
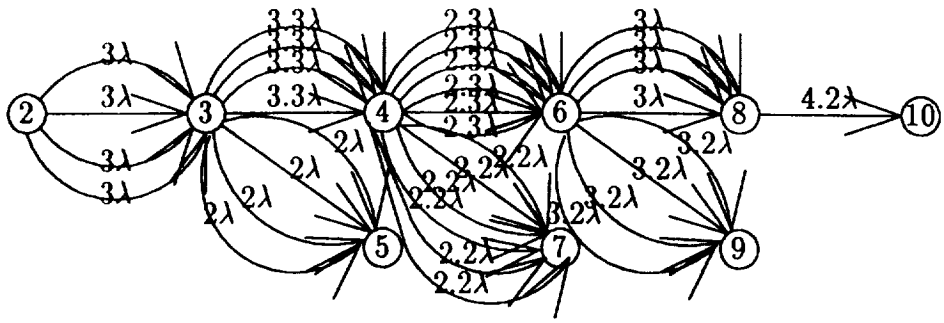
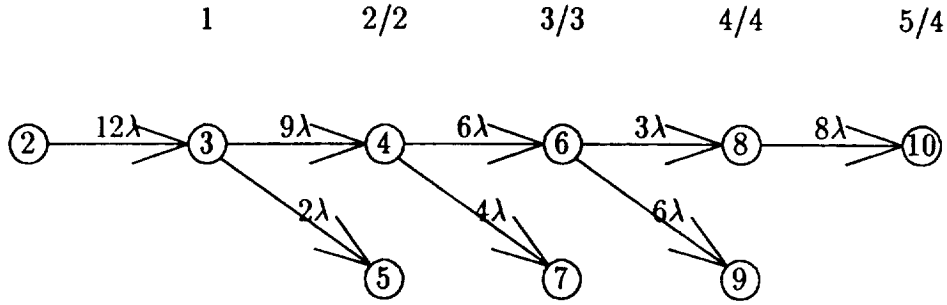


Figure 5.7. Full Model of Independent Triads



Aggregated States

9 States
33 Transitions



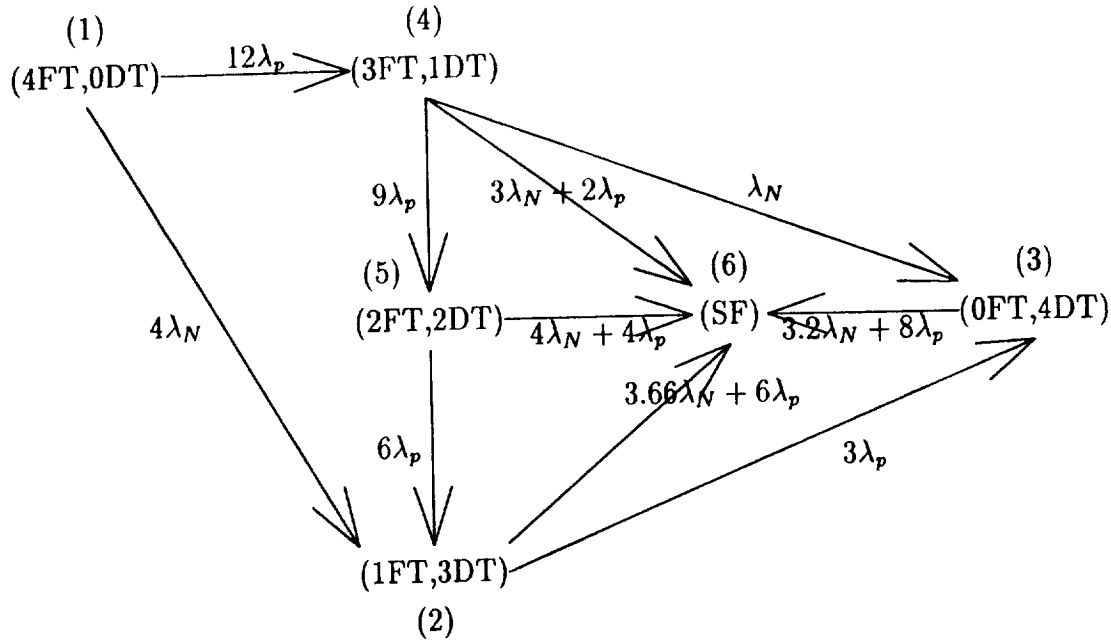
Aggregated Transitions

9 States
8 Transitions

Figure 5.8. Reduced Model of Independent Triads

Model	Lower Bound	Upper Bound
Full Triad	2.99997E-10	3.00009E-10
Reduced Triad	2.99997E-10	3.00009E-10
Independent Triads & Network Elements	3.01497E-10	3.01509E-10
Full Interdependent Triads & Network Elements	3.76495E-10	3.76510E-10
Reduced Interdependent Triads & Network Elements	3.76825E-10	3.76840E-10

Table 5.1. $P(SF)$ with λ_p = 1E-5, λ_n = 1E-6, t = .5



(iFT,jDT)≡(i Full Triads, j Degraded Triads)

6 States, 10 Transitions

Figure 5.9. Reduced Model of Interdependent Triads and Network Elements

Model	Lower Bound	Upper Bound
Full Triad	7.80267E-09	7.80419E-09
Reduced Triad	7.80267E-09	7.80419E-09
Independent Triads & Network Elements	7.98417E-09	7.98585E-09
Full Interdependent Triads & Network Elements	1.21910E-08	1.21936E-08
Reduced Interdependent Triads & Network Elements	1.22310E-08	1.22335E-08

Table 5.2. $P(SF)$ with λ_p = 5E-5 + 1E-6, λ_n = 1E-5 + 1E-6, t = .5

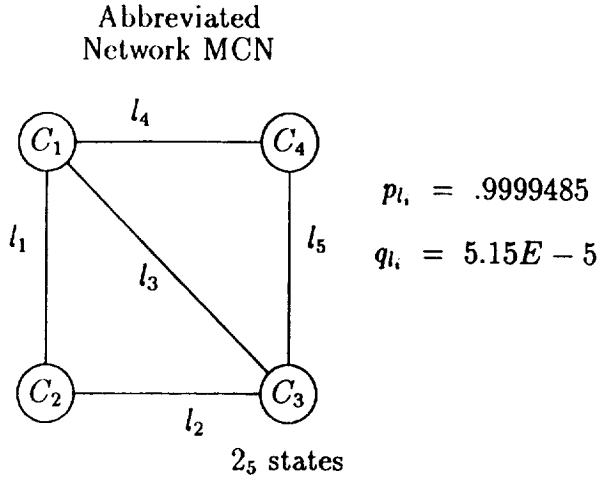


Figure 5.10. Example Network Model

in each cluster is connected to a network element in another cluster through their associated PEs. This connection scheme gives each cluster two connections to each of two clusters and allows messages to be passed between any two clusters. To completely model all interactions among all components would require more than $2^8 \cdot 2^{16} \cdot 2^{16} \cdot 2^{48}$ states.

Since tools to compute network reliability are not widely available and those that are being developed are generally for the two-terminal case, an example of a disjoint products technique to assess all-terminal reliability that is described in [6] is included here. This example is illustrated for the simple network topology illustrated in Figure 5.10.

For an all-terminal, undirected problem, the reliability of the network is defined to be

$$Rel_A \equiv \text{the probability that for every pair } v_1v_2 \text{ of nodes, } \exists \text{ a path from } v_1 \text{ to } v_2.$$

Assuming statistical independence of failures and the perfect operation of nodes, an approximate model can be created by only considering network link failures. However, since this is an all-terminal case, perfect node operation does not have to be assumed: system reliability can be computed as the product of perfect-node Rel_A and the individual node reliabilities.

The technique described in [6] begins with an enumeration of the minimal sets of operational states, or minpaths, for a network, N. For each minpath P_i, let E_i denote

the event that all edges in P_i are operational. Then the reliability of N is the sum over all minpaths P_i of the probability of each disjoint product event D_i formed from the events $E_1, \ldots, E_i$. Each disjoint product event is written as a Boolean expression and hence a Boolean expression is derived for $Rel(N)$. This Boolean expression can then be simplified and evaluated.

The minpaths for an all-terminal network are spanning trees, and [6] has modified an algorithm by Ball and Nemhauser to generate a Boolean expression for $Rel(N)$ while generating the spanning trees. The Boolean expression generated by this algorithm for the all-terminal reliability of the network in Figure 5.10 is

$$\begin{aligned} Rel_A(N) \;=\; & x_1x_2x_4 + x_1x_2\bar{x}_4x_5 + x_1\bar{x}_2x_3x_4 + x_1\bar{x}_2x_3\bar{x}_4x_5 + \\ & x_1\bar{x}_2\bar{x}_3x_4x_5 + \bar{x}_1x_2x_3\bar{x}_4x_5 + \bar{x}_1x_2\bar{x}_3x_4x_5 \end{aligned}$$

Assuming that the probability, p_{l_i}, that a link i is operational is .9999485, the probability, q_{l_i}, that it is not is 5.15E-5. Substituting p_{l_i} for x_i and q_{l_i} for $\bar{x}_i$, $Rel_A(N)$ is computed to be .99999. Assuming a node failure probability of 1.22335E-08 (see Table 5.2), Rel_A *with node failures* = .99998.

6. Summary

6.1. Goals

The goal of the DAHPHRS effort is to develop an integrated set of tools to assist the system architect in the design of high-performance, highly reliable systems. A program consisting of three phases was designed to achieve this goal. In Phase I, tools would be related to a methodology framework and baseline system descriptions would be developed and analyzed. In Phase II, tools would be specified and the baseline system descriptions enhanced for fault tolerance. Finally, in Phase III, tools and interfaces would be built.

Phase I has been completed by developing a paradigm of the systems design process using a representative example system subject to representative requirements and methods. For this paradigm, the JPL Hypercube, the Encore Multimax, and the CSDL FTPP architectures and the ALPHATECH WTA/TS and WAUCTION_-ASSIGNMENT algorithms were modeled. The issues investigated through the performance analysis of these systems include the impact of communications in networks, the cost of fault tolerance, the speedup and effectiveness achievable through varying levels of parallelism as a function of processor/communication speed and the extent of sequential tasks, mapping algorithms to architectures, and workload distribution. The creation of the paradigm models and the selection of the performance analyses were directed toward the tool/methodology issues of 1) selecting an appropriate model level, or fidelity, based on the information required to build the model versus the information available and the amount and type of data that would result from the analysis; 2) the roles of measurement, functional simulation, and stochastic methods in support of modeling; and 3) the validation of models.

The issues investigated in the reliability analysis of parallel, highly reliable systems include the large system problem, the impact of system configuration and management on the size and complexity of reliability models, and the interaction between the performance and reliability analyses. The reliability analysis tools/methodology issues considered are the creation of approximate models to bound an exact and difficult to analyze model and the relationship of the performance model to the reliability model.

6.2. Simulation Paradigms

System simulation is essentially an approximation of the system behavior. The granularity of a simulation is the amount of simulated time that can be realized in a unit of real time. It is always desirable to realize maximum simulation granularity, consistent with a minimum acceptable level of accuracy. An important outcome of this project has been the identification of a relationship between the nature of the system being simulated and the appropriate simulation paradigm. Loosely coupled systems can typically be modeled with high simulation granularity, whereas tightly coupled systems typically realize lower simulation granularity. Higher simulation granularity is desirable since such models require less real time to execute.

Coupling is the extent to which the execution of one processor (process) can interfere with the execution of another processor (process) and defines the granularity of process. Coupling can be due to resource sharing (e.g., memory) or control (e.g., lock step execution vs. message based synchronization). Fine grained architectures are tightly coupled while coarse grain machines are loosely coupled. In modeling loosely coupled systems, e.g., the Mark III, each task (an ADAS node) physically exercises a separate processor-memory data path. The execution time for that task can be computed based on processor specific characteristics, and is *independent of the execution times for other tasks.* All of the information a task requires for execution is locally available. The only reason a task cannot fire when ready is due to externally specified constraints (e.g., resource constraints due to mapping). The delay of an ADAS node can represent a task as large as the granularity of the computation being represented. Thus, for loosely coupled, medium/coarse grain computations, a GIPSIM-type modeling paradigm is efficient and desirable.

In tightly coupled architectures such as the Multimax, processors can potentially interfere on every memory access since all accesses must be across the bus. In this case, we must model events to that level of granularity, i.e., a bus transaction. The size of the ADAS graph increases drastically since each one must now represent a memory access, in some cases beyond several hundred million nodes. Even if one were to assume that mapping and scheduling such a graph is feasible, we notice that cause and effect is not local any more. The behavior of a memory access depends on whether it is a write or read access. The behavior of the bus depends on whether the processor or memory enabled the bus. Further, the operation of the cache and the bus arbitration policy are (or maybe) non-deterministic. In such cases it is necessary to model data dependent computation — *it is no longer enough to satisfy only sequencing constraints between events.* Modeling in such cases requires CSIM-like behavioral

modeling tools.

In summary, the nature of the architectures being modeled place demands on the features of appropriate modeling tools. Loosely coupled architectures modeling medium-coarse grained computation can achieve high simulation granularity with tools that represent sequencing constraints between events. Tightly coupled systems modeling fine grained computation realize low simulation granularity and often require the ability to model data dependent computation. For either class of machines, modeling non-deterministic phenomena requires the ability to model data dependent computation.

6.3. Operational Tools

Experiments require exercising the simulation models over a range of parameters — both algorithmic and architectural. Modeling changes in these parameters requires making changes to some attribute of the simulation model. This can be both time consuming and error prone, and generally involves straightforward, repetitive calculations. The architectures being modeled are becoming more complex, and the search space of possible solutions is growing exponentially. Tools for managing and performing these repetitive simulations are necessary.

In particular, when dealing with complex algorithms, the number of nodes required can be from several hundred (this program) to several thousand. When a single algorithmic parameter is changed, several thousand to tens of thousands of attributes need to be recomputed. Further, when these tools are hierarchically organized, it is desirable that intermediate nodes inherit values from nodes higher in the hierarchy, or be able to synthesize values from nodes lower in the hierarchy. The capability for *model inheritance* enables one to reuse basic blocks such as processors and networks in several different models. It is essential for rapid prototyping, configuration management and efficiency. The individual node descriptions that support such descriptions and the flow of information between them also serve as documentation for the behavior of that node. There are no conceptual barriers to the development of such a capability, and the existence of a prototype embodying some of these concepts is the Attribute Definition Language (ADL). We found the use of ADL, though not yet fully developed, to be invaluable. Such a tool is mandatory for any large-scale simulation toolset.

The systematic search of the architecture search space is the realm of a Simulation

Management Facility, such as that described in Appendix A of [15]. We again reiterate the value of such a facility. This program demonstrated the need for a similar tool for algorithmic parameters (as discussed above).

6.4. Mapping and Scheduling

It has become widely established that communication and synchronization overhead are often the principal determinants of performance in parallel architectures. However, adequate tools for enabling system designers to examine these issues are really lacking. As an outcome of this program, we have been able to disconcern two possible approaches — one for each simulation paradigm.

In the case of loosely coupled architectures (the GIPSIM-like world) the ADAS software graphs consist of nodes representing tasks interspersed with nodes representing intertask communication. A mapping algorithm may assign two communicating tasks to processors that are "far" apart, i.e., communication between them must cross several links. The ADAS communication node between these two tasks can be replaced by a series of nodes. The number of nodes represents the number of intermediate processors that communication must be routed through. This information can be derived from a description of the target architecture. Communication nodes mapped to the same processor node cannot fire simultaneously. Whether communication nodes and task nodes mapped to the same processor can fire simultaneously or not depends on the architecture of the node itself. Either case is easily modeled by appropriately setting the hardware attribute value in the individual nodes. This approach requires structural modification of the ADAS graphs. This can be either fully automated or performed with interactive user approval.

For tightly coupled architectures (the CSIM-like world), all of the algorithm specific information is embodied in the tables that control the behavior of each of the components. It is possible that some description of an algorithm (such as an ADL description) can be compiled to produce these tables. Placement of the tables is dictated by a mapping algorithm, and scheduling is handled naturally within the model. One can think of a library of CSIM architectural models being available, and algorithm descriptions being compiled to produce the tables. These tables would form the input to a mapping algorithm, and would drive the simulation.

6.5. Validation

In the course of constructing and executing simulation models one is required to make a number of assumptions about the architectures and algorithms. In addition, it is necessary to make decisions about what facets of the architecture, algorithm, or architecture/algorithm combination are important enough to be modeled. This naturally leads to the issue of the accuracy of the simulations themselves. We feel the ultimate test is the design, construction, and operation of the architecture/algorithm being modeled. However, the purpose of reliable simulation is to be able to make design decisions without having to learn from experience.

A more practical alternative is the construction of benchmarks that exercise different aspects of the architecture. These benchmarks may be evaluated via simulation as well as executed on various test machines. This approach is subject to the same criticisms as benchmarks in general, but also realizes many advantages. The important distinguishing feature from conventional benchmarks is that *these benchmarks are for the purpose of evaluating the accuracy of the modeling and simulation tools, not to evaluate the performance of the architectures.*

6.6. Miscellaneous

The remaining issues are not as significant as those discussed above. They are desirable more from the point of view of ease of use of tools themselves than any increased modeling or decision power that they would realize.

1. Simulation results should be exported and displayed in real time. This makes it easy to identify trends and may preclude the execution of unnecessarily long simulations.
2. Checkpointing and incremental evaluation is necessary to recover from unnatural termination of long simulations.
3. Incremental compilation.
4. ADAS graphs should be created automatically from standard descriptions, e.g., programs or other representations internal to software development tools.
5. Perhaps the largest benefit will be derived from the availability of tools for parallelizing applications. This is not central to modeling and simulation, but

will have a great impact — both from the point of view of ease in deriving ADAS representations of the computations and the validity of the results of the computations.

What we foresee is generation of tools that are integrated to form complete System Development Environments. Such environments will include tools that manage descriptions and requirements, tools that perform modeling and simulations, and finally, tools that enable validation and system development. This integrated environment is necessary to ensure the synergistic combination of tools that deal with the diverse aspects of large complex parallel/distributed systems.

6.7. Conclusions

As a result of the Phase I paradigm construction and analysis, a performance modeling process based on simulation of functional and performance models and measurement of engineering models of algorithms and architectures was proposed. The interactions of the three threads of this approach address the problems of fidelity of model to information and maintaining model consistency throughout the design process and result in validated, reusable models.

The information found to be necessary to create models for the assessment of architectures includes

- the structure and processing speed of the functional elements,
- the structure, size, and bandwidth of the memory elements,
- the structure and bandwidth of the interconnection network,
- the message formatting and error checking overhead required by the interconnection network,
- the structure and bandwidth of the input/output subsystem,
- the message formatting and error checking overhead required by the input/output subsystem,
- the fault-tolerant features of the architecture (e.g., fault containment regions, reconfiguration algorithms, redundancy management techniques), and

- the functions and overhead associated with the operating system (e.g., distributed executives, task allocation overhead).

The necessary information for the assessment of algorithms includes

- the potential for parallel implementation,
- the size and organization of major data structures,
- the system data and control flow, preferably as hierarchical data flow diagrams or input/output matrices,
- the use of standard mathematical and signal processing functions (e.g., matrix multiply, linear programming, and Fourier transforms),
- memory addressing patterns for each function and each data structure used by the function (e.g., linear, random but local, random and global),
- parallel implementation approaches (for some basic functions, the parallel processing capabilities may be known, e.g., matrix multiply),
- the number of instructions executed for each module of the system as a function of parameters which characterize an operational scenario, and
- information about data and control flow dependencies.

Tools can provide valuable insights into design decisions at the early stages in the design process; however, the current set of tools is unable to handle the size or complexity of SDI algorithms and architectures effectively. There do exist methods for reducing the complexity of the models to the point where they can be solved by existing tools, but the construction and validation of the reduced models is not well understood. Further, the current set of tools is not well integrated.

A preliminary environment for the integration of performance and reliability analysis, consisting of an evaluation controller, performance analysis tools, reliability analysis tools, and a shared data base, was proposed. The specification of the requirements for the tools and their integration into this environment which are scheduled for Phase II will derive from the following needs identified in Phase I:

- Tools need to be sensitive to parameter changes in the algorithm.

- Sophisticated mapping and scheduling tools are needed.
- A proper match of model resolution to tool capabilities must be maintained.
- Run-time modification of hardware attributes to model dynamic resource management is required.
- The development of requirements via CASE tools and the automatic creation of performance models from the CASE descriptions are needed.
- The effects of the software operating system need to be brought into the modeling process.
- Measurement is necessary to support and validate modeling.
- The systematic generation of reliability models and tools for model reduction needs to be explored.
- Tools for network reliability analysis need to be explored.
- Mechanisms for bringing fault tolerance mechanisms into performance models are needed.
- Tools to support experiment planning are needed.
- A shared data base consisting of the following elements is required:
  - the data required for performance and reliability models,
  - a library of primitive function models,
  - architecture transformation rules,
  - parallelization transformation rules, and
  - mapping rules.

6.8. Further Work

The Phase I effort results in three areas of further work for Phase II: methodology, the paradigm, and tool/data base specification and development. In the area of methodology, methods need to be developed for modeling operating system features, dynamic parallelism and fault tolerance features; for model validation; and for experiment planning. Further work on the paradigm is needed to study modeling

fault tolerance mechanisms, operating systems, and other types of applications, as well as more modeling of the WAUCTION_ASSIGNMENT algorithm as a vehicle for investigating dynamic performance models.

References

[1] R. Baker and C. Scheper. Evaluation of Reliability Modeling Tools for Advanced Fault Tolerant Systems. Contractor Report 178067, NASA, October 1986.

[2] R. L. Baker. Algorithm I Description. Technical report, Research Triangle Institute, Research Triangle Park NC, May 1988.

[3] S. J. Bavuso and A. L. Martensen. A Fourth Generation Reliability Predictor. In *1988 Proceedings Annual Reliability and Maintainability Symposium*, Los Angeles, CA, January 26-28, 1988.

[4] S. J. Bavuso, P. L. Petersen, and D. M. Rose. CARE III Model Overview and User's Guide. Technical Memorandum 85810, NASA - Langley Research Center, Hampton, VA, June 1984.

[5] R. W. Butler and A. L. White. SURE Reliability Analysis: Program and Mathematics. Technical Paper 2764, NASA, March 1988.

[6] C. J. Colbourn. *The Combinatorics of Network Reliability.* International Series of Monographs on Computer Science. Oxford University Press, Inc., NY, 1987.

[7] R. E. Harper. Critical Issues in Ultra-Reliable Parallel Processing. Technical Report CSDL-T-944, Charles Stark Draper Laboratory, Inc., Cambridge, MA, June 1987.

[8] S. C. Johnson. ASSIST User's Manual. Technical Memorandum 87735, NASA, August 1986.

[9] S. C. Johnson. Evaluation of Fault-Tolerant Parallel-Processor Architectures over Long Space Missions. Technical Memorandum 4123, NASA, 1989.

[10] R. Kumar. *A Fault Tolerant Cellular Architecture.* PhD thesis, Virginia Polytechnic Institute Institute and State University, 1984.

[11] A. L. Martensen and R. W. Butler. The Fault-Tree Compiler. Technical Memorandum 89098, NASA, January 1987.

[12] Y. W. Ng and A. Avizienis. A Unified Reliability Model for Fault-Tolerant Computers. *IEEE: Transactions on Computers*, C-29(11), November 1980.

[13] Rome Air Development Center, SDIO BM/C^3 Processor and Algorithm Working Group. *Application of Fault Tolerance Technology; Volume I: Design of Fault-Tolerant Systems; Volume II: Management Issues: Contractor Milestones and Evaluation; Volume III: Tools for Design and Evaluation of Fault-Tolerant Systems; Volume IV: System Security and its Relationship to Fault Tolerance*, October 1987.

[14] C. O. Scheper. Algorithm II Description. Technical report, Research Triangle Institute, Research Triangle Park NC, December 1988.

[15] C. O. Scheper, R. L. Baker, G. A. Frank, S. Yalamanchili, and F. G. Gray. Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems (DAHPHRS). Interim Report, RADC-TR-90-81, RADC, May 1990.

[16] J. J. Shaw et al. Battle Management Structures, Volume I: Directed Energy Weapon, Weapon Target Assignment Algorithms and Volume II: Kinetic Energy Weapon, Weapon Target Assignment Algorithms. Technical report, RADC-TR-89-84, RADC, January 1988.

[17] T. S. White. A Distributed Control Reconfiguration Algorithm for 2-Dimensional Mesh Architectures which Tolerates Single Faults Per Row. Master's thesis, Virginia Polytechnic Institute and State University, 1988.

[18] R. M. Yanney and J. P. Hayes. Distributed Recovery in Fault-Tolerant Multiprocessor Networks. *IEEE Transactions on Computers*, C-35(10):871–879, October 1986.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1992	Contractor Report

4. TITLE AND SUBTITLE Integration of Tools for the Design and Assessment of High-Performance, Highly Reliable Computing Systems (DAHPHRS) Phase I

5. FUNDING NUMBERS
C NAS1-17964
WU 505-64-54-01

6. AUTHOR(S) C. Scheper, R. Baker, G. Frank, S. Yalamanchili, G. Gray

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Research Triangle Institute
P.O. Box 12194
Research Triangle Park, NC 27709-2194

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23665-5225

Rome Laboratory
Griffiss Air Force Base,
NY 13441-5700

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
NASA CR-189621

11. SUPPLEMENTARY NOTES
Langley Technical Monitors: Sally C. Johnson and Carl R. Elks

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Unclassified - Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)* Systems for Space Defense Initiative (SDI) space applications typically require both high performance and very high reliability. These requirements present the system engineer evaluating such systems with the extremely difficult problem of conducting performance and reliability trade-offs over large design spaces. A controlled development process supported by appropriate automated tools must be used to assure that the system will meet design objectives. This report describes an investigation of methods, tools and techniques necessary to support performance and reliability modeling for SDI systems development. Models of the JPL Hypercube, the Encore Multimax, and the C. S. Draper Lab Fault-Tolerant Parallel Processor (FTPP) parallel-computing architectures using candidate SDI weapons-to-target assignment algorithms as workloads were built and analyzed as a means of identifying the necessary system models, how the models interact, and what experiments and analyses should be performed. As a result of this effort, weaknesses in the existing methods and tools were revealed and capabilities that will be required for both individual tools and an integrated toolset were identified.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Fault-tolerant computing; Performance evaluation; Multiprocessors, Reliability Analysis; Parallel processing, CAE tool integration	187
	16. PRICE CODE: A09

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		