

AD-A259 511



2

RESEARCH TRIANGLE INSTITUTE

ARTIFICIAL INTELLIGENCE FOR
VHSIC SYSTEMS DESIGN (AIVD)
FINAL REPORT

DTIC
ELECTE
DEC 30 1992
S A D

December 1988

Prepared for:

Department of the Army
U.S. Army Electronics Research and Development Command
Fort Monmouth, New Jersey 07703-5000
Contract No. DAAL01-86-C-0040

Prepared by:

Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Octy, Inc.
10920 Oxford Court
Fairfax Station, Virginia 22039

92-32882
N807

This document has been approved
for public release and sale; its
distribution is unlimited.

ARTIFICIAL INTELLIGENCE FOR VHSIC SYSTEMS DESIGN (AIVD) FINAL REPORT

December 1988

Department of the Army
U.S. Army Electronics Research and Development Command
Fort Monmouth, New Jersey 07703-5000
Contract No. DAAL01-86-C-0040

Prepared by:
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Octy, Inc.
10920 Oxford Court
Fairfax Station, Virginia 22039

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC QUALITY INSPECTED -

Contents

LIST OF FIGURES	iii
ACKNOWLEDGEMENTS	iv
1 INTRODUCTION	1
1.1 AIVD Project Goals	1
1.2 Background	1
1.2.1 Real-Time System Design Characteristics	1
1.2.2 The ADAS View of Software/Hardware Codesign	2
1.3 AIVD Requirements	4
1.4 The AIVD Design Methodology	5
2 AIVD PROJECT DELIVERABLES	9
2.1 The AIVD System	9
2.2 The AIVD Tools	9
2.2.1 The Intelligent User Interface	9
2.2.2 The Graph Transformation System	9
2.2.3 The Attribute Definition Language Evaluator	9
2.2.4 The ADAS Simulators	11
2.2.5 The Allocator of Software to Hardware (ASH)	11
2.3 The AIVD Data Bases	11
2.3.1 The Application Domain Hierarchy	11
2.3.2 The Hardware Domain Hierarchy	11
2.3.3 The Transformation Rule Base	12
2.3.4 Attribute Definition Language Programs	12
2.3.5 ADAS Software Data Base	13
2.3.6 ADAS Hardware Data Base	13
2.4 The AIVD Workstation	13
2.5 AIVD Documents	14
2.6 The AIVD Demonstration	14
2.6.1 The Demonstration Scenario	14

	ii
2.6.2 Demonstration Data Bases	15
2.6.2.1 The Application Domain Hierarchy	15
2.6.2.2 The Hardware Domain Hierarchy	17
2.6.2.3 The Transformation Rule Base	21
2.7 AIVD Reviews	22
3 THE AIVD DEVELOPMENT APPROACH	24
3.1 Overview of Approach	24
3.2 An Application of AIVD	25
3.2.1 Description of Application	25
3.2.2 The High Level Model	26
3.2.3 The Detailed Model	30
3.2.4 Software Partitioning and Parallelism	33
3.2.5 Communication and Fault Tolerance	35
4 FURTHER WORK	38
4.1 Further Application of AIVD Tools	38
4.2 Further Enhancement of AIVD Tools	38
4.2.1 Develop a System Experiment Environment	38
4.2.2 Enhance the Graph Transformation System	39
4.2.3 Enhance the Software to Hardware Mapping Program	40
4.3 Converting AIVD to Product Quality	40
5 CONCLUSIONS	41

List of Figures

1.1	The AIVD Design Process	8
2.1	AIVD System Structure	10
2.2	Application Domain Hierarchy	16
2.3	ADAS Data Flow Graph for SAR Algorithm	18
2.4	SAR Graph Adapted to FPHP Architecture	19
2.5	Hardware Domain Hierarchy	20
3.1	Top Level ADAS Graph of WTA/TS Algorithm	28
3.2	WTA/TS Graph ADL Program	29
3.3	WTA/TS Process Utilization	31
3.4	ADAS Subgraph of Target Cluster Definition Algorithm	32
3.5	GLM Node ADL Program	33
3.6	Target Clustering Process Utilization	34
3.7	Performance Costs of Fault Tolerance	37

ACKNOWLEDGEMENTS

This report was prepared for the U.S. Army Electronics Research and Development Command under Contract N. DAAL01-86-C-0040. The work was performed in the Center for Digital Systems Research (CDSR) of Research Triangle Institute (RTI) by H. Waters (Project Leader), L. Dent, S. Duncan, G. Frank, W. Ingogly, C. Scheper, W. K. Neighbors, W. Ransdell, and consultant, D. Van Den Bout, and at Octy, Inc. by J. Cuadrado, B. Glazer, and G. Linsenmayer.

We wish to express our appreciation for the guidance and support provided by the project technical monitors, S. Waldman and M. Coy of Army LABCOM, Ft. Monmouth, New Jersey.

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

1. INTRODUCTION

1.1 AIVD Project Goals

The Artificial Intelligence for VHSIC Systems Design (AIVD) project was undertaken in order to enhance the capabilities of the Architecture Design and Assessment System (ADAS),¹ and to provide additional support for the ADAS software/hardware codesign methodology. AIVD supports software/hardware codesign in several ways:

- AIVD assists the user in building complex software system models from libraries of generic algorithms.
- AIVD assists the user in transforming generic algorithm descriptions to meet hardware resource constraints and interacting software subsystem resource requirements.
- AIVD assists the user in defining software performance attributes in terms of mission parameters and architecture parameters.
- AIVD assists the user in building experiments to explore trade-offs across large design spaces.

AIVD uses artificial intelligence techniques to implement these capabilities:

- Rule-based programs for software to hardware allocation and for graph transformations.
- Transformation rules (in the form of context-sensitive graph grammar productions) for modifying the structure and attributes of graphs.
- Attribute grammars to define performance measures in terms of mission and hardware parameters and to back-annotate models with performance results.

1.2 Background

1.2.1 Real-Time System Design Characteristics

The types of system design problems that ADAS was developed to support share several common characteristics:

¹ADAS is a registered trademark of the Research Triangle Institute.

Iterative Processing System design is an iterative process. There are several types of cycles that can occur in the design process. A system design tool should support the process of working through all of those cycles efficiently. The most important iterative cycle to support efficiently is the innermost iteration, i.e., the cycle which is repeated most often. AIVD focuses on supporting the incremental modifications to the model based on different system parameters that are inherent in performing trade-off studies. This gives the system architect more time to focus on designing different variations of the software and the hardware, rather than on building the models required to perform trade-off analyses.

Quantitative Trade-offs ADAS was designed to assist the system architect by providing quantitative performance information. AIVD extends that support to include the many performance analyses required to make trade-offs. Thus critical issues for a system architect using AIVD are defining the set of trade-off experiments to be performed, defining the parameters for those experiments, and deciding how to interpret the results of the simulations.

Software and Hardware Interactions A basic tenet of the ADAS methodology is that system performance is based on the interaction between the software and the hardware. ADAS models this interaction in terms of the mapping of software to hardware and the contention of software processes for shared hardware resources. AIVD enhances the ADAS capabilities for automatically mapping software to hardware.

Multidimensional Design Space The need to perform trade-off studies across many different design options leads to a multidimensional design space. Typical trade-off studies must consider a bewildering number of different design options and system parameters. Each option and parameter defines a different dimension of the design space. Each simulation evaluates one point in that design space. Thus the system architect must use the modeling resources wisely if a large number of options and parameter values is to be considered. AIVD is designed to allow the system architecture to explore a wider range of options.

1.2.2 The ADAS View of Software/Hardware Codesign

The first basic assumption that ADAS makes is that timing is a critical design issue. ADAS focuses on the design and assessment of real-time computer systems. A *real-time system* is a system in which there are critical requirements upon the time at which events occur. These requirements may take several forms:

- The rate at which an input data stream is consumed and processed without losing data may be critical. For example, a sensor must be sampled at a particular

rate.

- The rate at which an output data stream is produced may be critical. For example, an image must be outputted 30 times a second in order to be flicker-free.
- The sequence of events may be critical. For example, a parachute must be deployed before the landing gear is extended.
- The time between events may be critical. For example, the time between sighting a target and launching a weapon must be less than 5 seconds.

The second basic assumption that ADAS makes is that the system software can be modeled with hierarchical data flow graphs. This assumption is based on extensive work on the use of data flow graphs for structured systems analysis and structured system design, pioneered by Tom DeMarco.² ADAS extends this model by providing attributes for graphs, nodes, and arcs which describe the performance characteristics of the system. These attributes lead to a capability to simulate the performance of the system using a form of Petri Nets called *marked graphs*.

The third basic assumption that ADAS makes is that the system design must take into account the interactions between software and hardware. ADAS has modified the marked graph models in order to account for the contention for hardware resources that are experienced when independent software processes share hardware resources.

The ADAS methodology is based on performing a series of steps to build a model of the system and then evaluate its performance:

- Building hierarchical data flow diagrams which describe the system software.
- Defining the performance attributes of the system software.
- Building hierarchical block diagrams which describe the structure of the system hardware.
- Mapping software to hardware.
- Simulating performance at the marked graph level.
- Evaluating the performance results produced by the simulation.

ADAS provides mapping and simulation tools and a graphical user interface to support the construction of models. AIVD focuses on assisting the user in iteratively making incremental changes in the models in order to evaluate many trade-off options.

²Tom DeMarco. *Structured Analysis and System Specification*. Englewood Cliffs: Yourdon Press, 1979.

1.3 AIVD Requirements

The AIVD System Requirements Document describes the requirements for the AIVD system in terms of:

- A set of five requirements for ADAS which AIVD must support.
- A set of four requirements for AIVD which enhance the capabilities of ADAS.

The ADAS system has five major functions, all of which are supported by AIVD:

Construct System Data and Control Flow This function is currently provided in ADAS by the graph editor, EDIGRAF.

Verify System Data and Control Flow Consistency This function is currently provided in ADAS by the consistency checker, CONCH, and by the data flow analyzer, FLOWBAL.

Map System Data Flow Onto Hardware Resources This function is currently provided in ADAS by the software to hardware mapper, ASH.

Evaluate System Performance This function is currently provided in ADAS by the performance simulator, GIPSIM.

Verify System Function This function is currently provided in ADAS on software graphs by the functional simulators, CSIM and ADASIM, and on hardware graphs by the HDL interfaces, VHDLGEN, HELIXGEN, and ISPGEN.

AIVD enhances the Construct System Data and Control Flow function with three additional subfunctions:

Select Algorithms The Intelligent User Interface (IUI) allows the user to build the system data flow by selecting algorithms from the Application Domain Hierarchy (ADH) and connecting them to implement the system functions. The ADH provides a hierarchical structure for a collection of domain-specific algorithm data bases. The IUI allows the user to select algorithms rapidly by moving around the ADH and allows the user to read help files about the algorithms in the current application domain.

Transform Algorithms The Graph Transformation System (GTS) supports both interactive and automatic transformation of generic algorithm descriptions to fit the software environment and hardware resources.

Instantiate Algorithms The Attribute Definition Language (ADL) allows the algorithm descriptions in the ADH to be parameterized. The Attribute Definition Language Evaluator (ADLEVAL) program automatically instantiates a parameterized algorithm description and provides the specific attribute values required for the other ADAS tools, such as CONCH, FLOWBAL, GIPSIM, CSIM, and ADASIM.

Map System Data Flow Onto Hardware Resources AIVD enhances the ADAS Allocator of Software to Hardware (ASH) to increase compatibility with marked graph simulator GIPSIM, to allow interactive display and control of the mapping process, and allow more general optimization rules and mapping constraints.

1.4 The AIVD Design Methodology

The AIVD design process is shown in Figure 1.1.

The process includes several types:

Select Architecture from Hardware Domain Hierarchy. The architecture is described by:

- A set of component models (e.g., CPUs, IOPs, Memories, Buses).
- A set of basic models for the interconnection patterns (e.g., a hypercube would have a 2 node hypercube as a basic component.).
- A set of rewrite rules which would allow particular instances of the architecture to be built from the basic models (e.g., a hypercube would have a rewrite rule for building a larger hypercube from a smaller one by copying the existing hypercube and then connecting all the corresponding nodes in the two copies).
- A set of attribute definitions which would be used to configure the algorithms.

Transform Architecture In order to simulate a system, a specific instance of the architecture must be constructed. This is done by configuring an architecture using the proper number of each type of component and connecting them according to the connection patterns of the architecture. With AIVD, the system architect uses the graph transformation system to connect the components according to the specification for the architecture as encapsulated in the graph transformation rules for the architecture. A design trade-off will often experiment with different numbers of components. Figure 1.1 shows the change in

these numbers, *Network Size Parameters* being the middle loop for the design process.

Modify Architecture Attributes Typical architecture attributes are processor speed, memory bandwidth, interconnect bandwidth, and memory size. These attributes will effect the mapping process and the algorithm node processing times. A design trade-off will often require experiments with different values for these attributes. Figure 1.1 shows the change in these attributes, *Module Performance Parameters* being the inner loop for the design process.

Select Algorithm The user selects appropriate algorithms for each of the system functions from Application Domain Hierarchy. The ADH is organized hierarchically by function in order to make it easier for the user to determine which algorithms in the library are appropriate for the particular function and application. Furthermore, the ADH may have several variations on a particular algorithm, where different variations are associated with different hardware architectures. Thus the user may use information about the architecture, as provided by attribute definitions, to select appropriate variations on the algorithms suited to the functions of the application.

Transform Algorithm The next step is to customize each of the algorithms in the system to fit the available architecture resources. This is done in a global context, so that trade-offs for resources can be made between algorithms for different functions of the system. When the algorithms are selected from the ADH, they come equipped with different transformations which are needed to adapt the algorithm to different instances of architectures. Once the architecture has been defined, its characteristics can be used to select the appropriate transformations for the algorithms. The architect uses the graph transformation system to customize the algorithm. Typical reasons for making architecture-specific transformations are:

- Increase the parallelism of an algorithm.
- Decrease the parallelism of an algorithm.
- Model the communication costs of an algorithm as distributed across the architecture.
- Provide the data or processing redundancy needed to support system fault tolerance.

Modify Algorithm Attributes Once the algorithm has been transformed, the algorithm attributes such as data structure size and operation counts are combined with the architecture attributes such as processor speed, memory bandwidth, and interconnect bandwidth to define the produce and consume rates

and firing delays of the performance model. This process is performed by ADL-EVAL, using the attribute definition programs attached to the algorithm nodes, arcs, and graphs, and the ADL include files provided by the architecture model.

Map Software to Hardware The fully instantiated algorithm is mapped to the fully instantiated architecture using the Allocator of Software to Hardware.

Simulate System In AIVD, the user can evaluate performance by simulating the system at the marked graph level. Alternatively, the user can build a functional model using C or Ada³ code segments for each leaf node, and then perform a combined functional and performance simulation.

Evaluate Simulation Results This critical step is performed manually by the AIVD user, who must decide whether or not to continue the search through the design space and which point in the design space should be evaluated next. The user implements the decision by setting new parameter values in ADL files, and by invoking the appropriate tool to start the next cycle through the process.

³Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

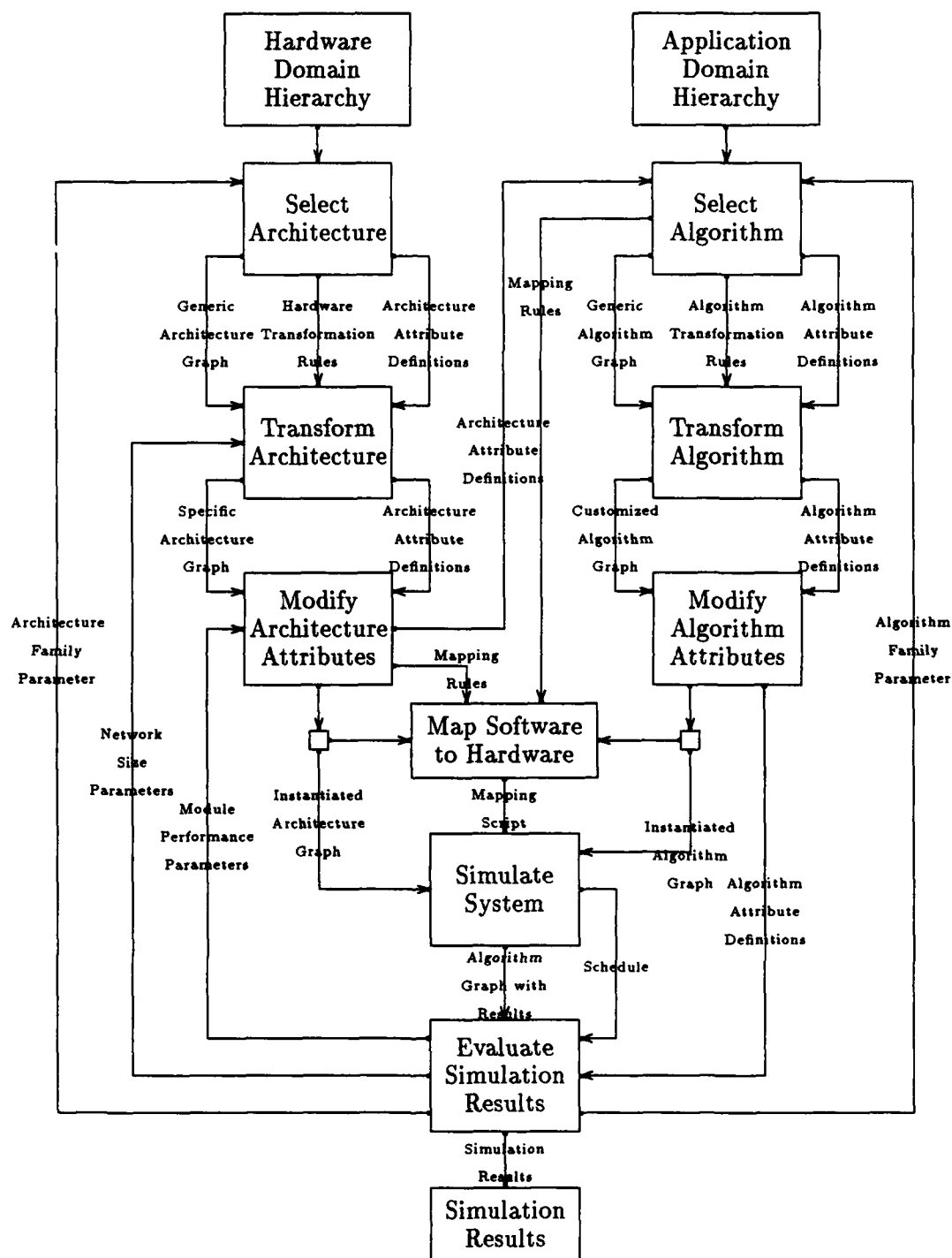


Figure 1.1: The AIVD Design Process

2. AIVD PROJECT DELIVERABLES

2.1 The AIVD System

The AIVD system structure is shown in Figure 2.1. The system consists of four major tools interacting with six major data bases and the existing commercial ADAS tool set. The tools and data bases are described in the following sections.

2.2 The AIVD Tools

2.2.1 The Intelligent User Interface

The Intelligent User Interface (IUI) guides the user in selecting appropriate algorithms from the Application Domain Hierarchy. The IUI provides a browsing capability for exploring hierarchical libraries of generic algorithms. This includes viewing the graph and template hierarchies associated with a library and reading the help files associated with the library. It also includes an object-oriented editing capability for hierarchical models, so that the user can point to an object and edit text files associated with the object, such as an ADL program, Ada or C language source code files, and help files.

2.2.2 The Graph Transformation System

The Graph Transformation System (GTS) 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.

Transformations define how to change 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.

2.2.3 The Attribute Definition Language Evaluator

ADLEVAL translates ADL programs into script files which can be read by the other AIVD tools, including the ADAS editor and simulator. The script files set the performance attributes of the ADAS models, including node *firing_delay* and *module_class*, input *token_consume_rate* and *firing_threshold*, and output *token_produce_rate*.

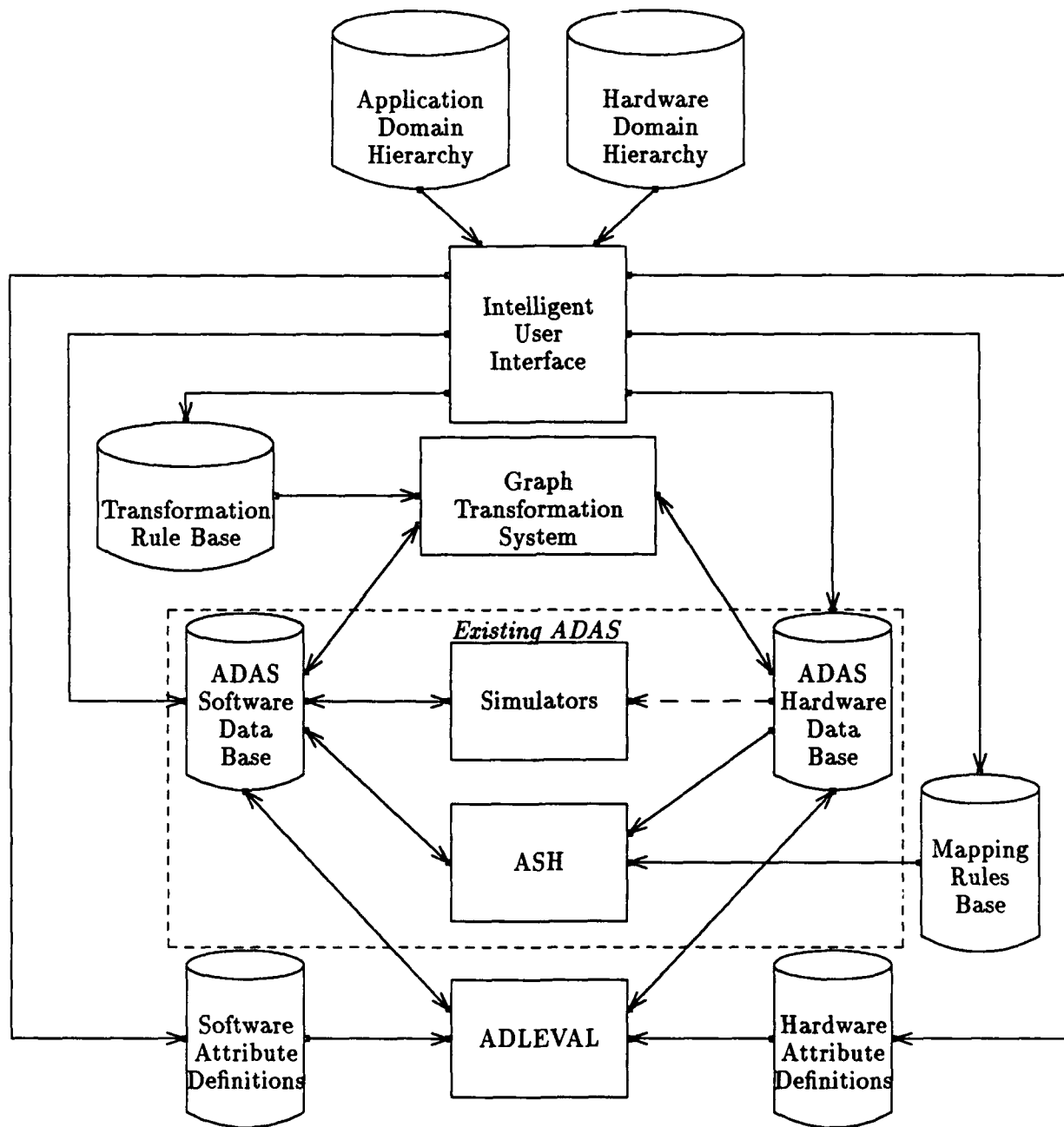


Figure 2.1: AIVD System Structure

2.2.4 The ADAS Simulators

The AIVD tool set is compatible with the three ADAS simulators: GIPSIM, CSIM, and ADASIM. GIPSIM is a directed graph simulator, while CSIM and ADASIM are functional simulators using C and Ada, respectively. The AIVD tool set creates a loosely-coupled interface with the ADAS simulators through the generation of ADAS script files.

2.2.5 The Allocator of Software to Hardware (ASH)

The Allocator of Software to Hardware (ASH) has been enhanced in the AIVD program to employ simulated annealing methodology in assigning software functions to hardware resources. These functions and resources are represented in ADAS software graphs and hardware graphs, respectively. The allocations are made in an effort to minimize the maximum utilizations of given hardware resources.

2.3 The AIVD Data Bases

2.3.1 The Application Domain Hierarchy

The Application Domain Hierarchy (ADH) organizes the descriptions of generic algorithms by application. Each level of the hierarchy has a template file associated with it containing the templates for all the lower level algorithms. With each level may be associated transformation rules that can be applied to any of the graphs in the lower levels of the ADH. At the lowest level, there are several types of information:

- A set of node and arc templates which refer to common libraries of basic algorithms, such as sort algorithms or Fourier transform algorithms.
- The ADL programs for each of the algorithms describing the performance characteristics of the algorithm, typically in terms of the number of instructions executed or in terms of the number of operations required.
- Ada program fragments for the primitive algorithms which are developed to the level needed to support functional simulation of the algorithm.

2.3.2 The Hardware Domain Hierarchy

The Hardware Domain Hierarchy (HDH) contains the descriptions of potential architectures. For each architecture, there are several types of information:

- A set of node and arc templates which refer to common libraries of components, such as 1750A or 32020 processors, PI-Bus or TMDE interconnections, and different memory architectures.
- A set of transformation rules for building an instance of the architecture from the components.
- A set of transformation rules for adapting a software algorithm to the architecture.
- A set of ADL descriptions of the performance characteristics of the components of the architecture and information about the operating system for the architecture, such as compilation rates for estimating machine code size from source code.

2.3.3 The Transformation Rule Base

The transformation rule base is a set of ADAS graphs, each of which describes the patterns and transforms of a consistent set of rules designed to transform a software data flow graph in order to achieve a specific purpose. Transformation rule bases may be developed for a specific application by the user or may be constructed by selecting and then merging rules that are distributed throughout the ADH and the HDH.

2.3.4 Attribute Definition Language Programs

ADL programs describe the ADAS node, graph, and arc attributes in terms of formulae which use system parameters defined by the user. These parameters typically consist of:

- Mission parameters.
- Hardware performance parameters.
- Parameters used for back annotation.

The ADL supports two types of communication: *inheritance* and *synthesis*. Inheritance allows parameters to be defined at a global level (such as the root graph of the system software hierarchy) and inherited into all nodes and arcs in the subgraph below the graph where they are defined. Inheritance is appropriate for distributing mission parameters to all the software nodes. Synthesis allows parameters to be computed by aggregating the values of ADAS attributes in a subgraph to define the value of an attribute in the parent node or graph. Synthesis is appropriate for back-annotating either software or hardware graph hierarchies.

2.3.5 ADAS Software Data Base

The ADAS software data base is a hierarchy of data flow graphs. It serves as the central working data base during an AIVD session, which typically involves multiple trade-off experiments. At the top levels of the hierarchy, it describes the functions of the system. At the middle levels of the hierarchy, it describes the algorithms and data structures that provide the functions required of the system. At the lowest levels of the hierarchy, it describes the partitioning of the algorithms and data structures to fit onto the system hardware architecture.

The attributes of the ADAS software graph hierarchy describe the performance characteristics of the system, including the amount of time it takes to perform each atomic action of each algorithm and how much input and output data is required for each atomic action.

2.3.6 ADAS Hardware Data Base

The ADAS hardware data base is a hierarchy of graphs which describe the structure and components of the system hardware architecture.

The ADAS hardware graph hierarchy constrains the possible mappings performed by ASH. Each component of the architecture belongs to a *module_class* and each atomic action of each algorithm must be mapped to a hardware component with the same *module_class*. Each arc of the software data flow graph must map onto a node or arc of the hardware graph. During simulation, atomic actions of algorithms must contend for the shared resources of the hardware components.

The attributes of the ADAS hardware graph hierarchy describe the performance characteristics of the components of the architecture, such as sensors, processors, memories, and interconnections.

2.4 The AIVD Workstation

As part of the AIVD project, RTI installed an AIVD workstation at the Army LAB-COM, Electronic Technology and Devices Laboratory, Ft. Monmouth, NJ. This workstation consisted of a VaxStation II/GPX with 19 inch color monitor, 4 million bytes of fast storage, and 70 million bytes of disk storage. The workstation was equipped with the VMS Version 4.5 operating system. Languages supported by the workstation included DEC Ada, DEC C, and Quintus Prolog Version 2.0. Application tools provided with the workstation included RTI's ADAS Version 2.5 and the AIVD tools.

2.5 AIVD Documents

In parallel with the sequence of reviews for this project, RTI and Octy, Inc. produced and delivered a series of documents describing the system as it went through its development cycle. The documents included:

The AIVD System Requirements This document describes the system requirements, the relationship between ADAS and AIVD, and describes some scenarios for the use of the AIVD tools.

The AIVD User Reference Manual RTI and Octy developed two different versions of the reference manual. The draft version was used as an aid in developing the specifications of the tools that were implemented. The final version describes the tools which were actually delivered.

The AIVD Tool Specifications RTI and Octy developed two different versions of the specification document. The draft version describes the Intelligent User Interface commands, the Attribute Definition Language, and the Transformation Rule Base format. The final version describes the IUI commands, a revised definition of the Attribute Definition Language, the Graph Transformation System and the Transformation Rule Base, and the Allocator of Software to Hardware.

The AIVD Demonstration Plan The AIVD Demonstration Plan describes the example that was used for demonstrating the power of the tools and the integration of the tools. The example is based on the Synthetic Aperture Radar (SAR) example described in Bowen and Brown, "Systems Design," Volume II of *VLSI Systems Design for Digital Signal Processing*.

2.6 The AIVD Demonstration

2.6.1 The Demonstration Scenario

The AIVD demonstration has the following steps:

Select a Design Domain Use the IUI to access the Application Domain Hierarchy. Demonstrate the capability to review the various application subdomains. Select the SAR subdomain to develop the demonstration application design.

Create a Generic Software Graph Demonstrate the use of the IUI to create a top-level application SAR design as shown in Figure 2.3. Use ADLEVAL to instantiate all graph, node, and arc attribute values which are needed by the GTS to select rules, evaluate matches, and choose transformations.

Transform the Generic Software Graph Use GTS to complete the design by applying appropriate graph transformations and instantiating the attribute values dependent upon hardware operating characteristics. Demonstrate the control mode capabilities of GTS.

Map the Transformed Software Graph to the Hardware Use ASH to map the resulting designs to a hardware configuration.

Compute the Performance Attribute Values Use ADLEVAL to instantiate attribute values dependent upon hardware operating characteristics.

Simulate and Evaluate the Designs Use GIPSIM to simulate the performance of the designs as mapped to the hardware configurations.

2.6.2 Demonstration Data Bases

2.6.2.1 The Application Domain Hierarchy

The Application Domain Hierarchy (ADH) developed for the demonstration consists of a four level taxonomy of application-specific functions. The structure of the ADH is not necessarily a tree, but rather a DAG, since several functions may be shared across different domains or subdomains (such as Fourier transforms). This hierarchy is implemented by parallel ADAS graph hierarchies and directory structures. The first level of the hierarchy defines three major application domains:

- Radar processing.
- Image processing.
- Sonar processing.

The radar processing domain is divided into two specific project areas: Synthetic Aperture Radar (SAR) and AWACS. Subdomains of the SAR domain may be shared among other radar processing domains. Two such potential subdomains would be a doppler processing domain and a range processing domain.

Each ADH domain may contain software graph files, template files, Attribute Definition Language (ADL) files, transformation rule bases, and help files pertinent to that domain. Figure 2.2 depicts a portion of the ADH tree structure created for the demonstration.

Figure 2.3 shows the ADAS data flow graph of a synthetic aperture radar algorithm. The purpose of a SAR is to obtain high resolution two-dimensional ground images from air-to-ground or satellite-to-ground radar data. The input range data

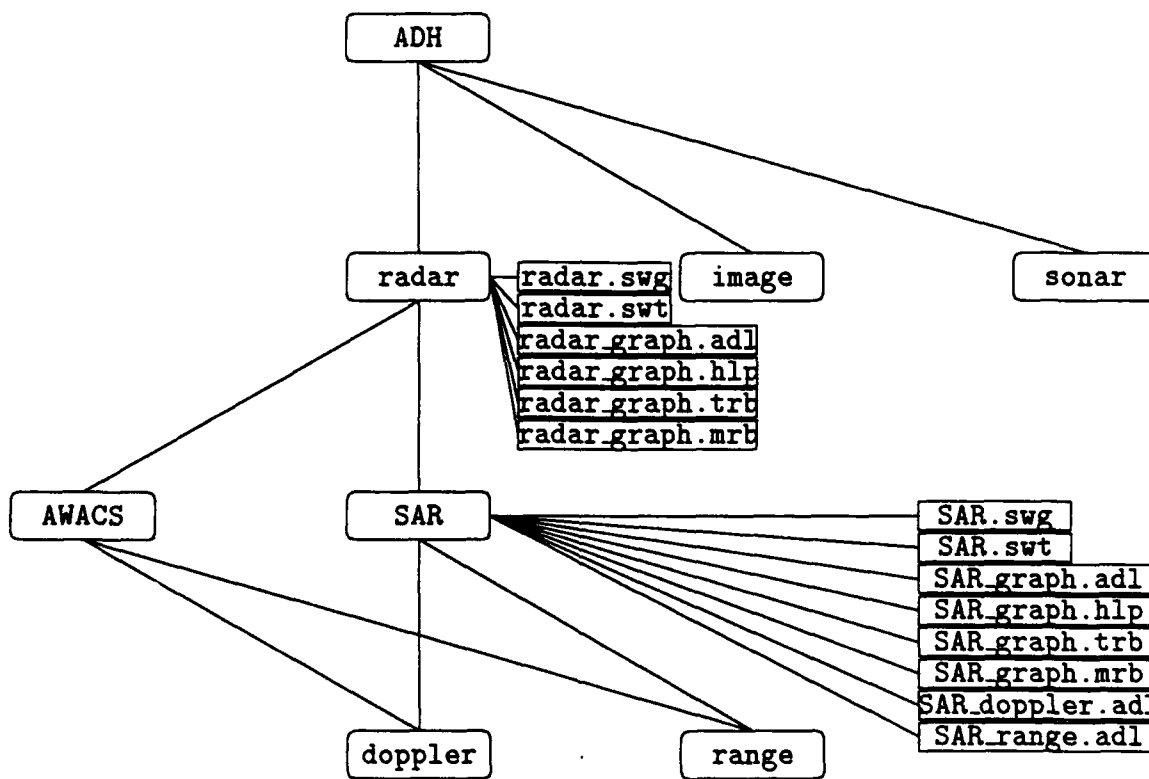


Figure 2.2: Application Domain Hierarchy

consists of vectors of complex data values; each vector contains 9978 data points and the frequency of generation of data vectors is 1418 Hz, for a total input data rate of about 14 million complex values per second.

The basic processing algorithm is a two-dimensional filtering operation on a data array of 2048 range vectors to achieve high radar resolution in both range and azimuth. The filter is implemented as two one-dimensional filters, one in range ("range compression") and one in azimuth ("azimuth compression").

The first filter operation is done on individual range vectors. The output range vectors are stored in a "corner turn" memory to form azimuth frames, which consist of a total of 2048 range vectors. (This process actually stores only 1732 new vectors each frame; a 316 vector overlap completes the 2048 vector frame.) Azimuth vectors are read out in a sequence of 9978 vectors, each a length of 2048 complex numbers. The azimuth vectors are filtered to achieve an azimuth compression and high azimuth output resolution.

Figure 2.4 shows the transformed version of the SAR data flow graph after it has been adapted to the FTTP architecture. Each of the processing and memory nodes has a subgraph representing a Triple Modular Redundant (TMR) implementation for fault tolerance. Each arc in the original data flow graph now has an associated communications node. Each communications node has a subgraph showing the various hardware resources required to transmit the message from its source to its sink.

2.6.2.2 The Hardware Domain Hierarchy

The demonstration data bases include a Hardware Domain Hierarchy (HDH) which serves the function of collecting the information on the architecture models. This is parallel to the function that the ADH serves in collecting information about the algorithm models. In the HDH delivered as part of the demonstration system, two candidate architectures are included: the Common Signal Processor (CSP), and the Fault-Tolerant Parallel Processor (FTPP). Each of the two candidate architectures, CSP and FTTP, are stored in separate subdomains. Both of these have subdomains for each of their components as described below.

Within each HDH domain lie hardware graph files, ADL files, and help files pertinent to that domain. The top level domain for each architecture also contains the hardware template file for that architecture. Figure 2.5 depicts a portion of the HDH tree structure created for this demonstration.

The candidate hardware designs are the CSP of IBM and the FTTP of Charles Stark Draper Laboratories. Both architectures are modular in design, and both emphasize high performance processing. The latter design includes high-reliability processing features.

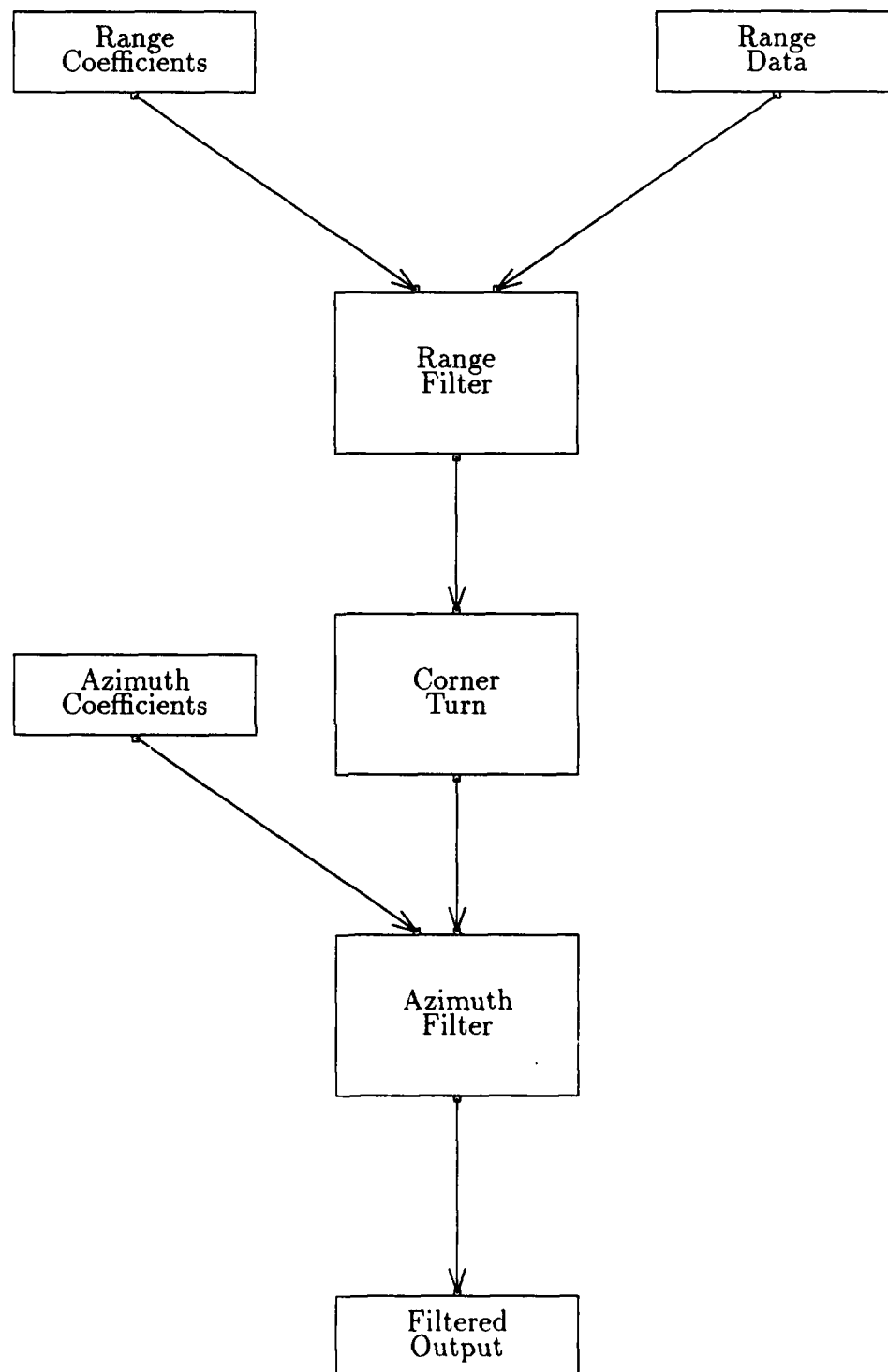


Figure 2.3: ADAS Data Flow Graph for SAR Algorithm

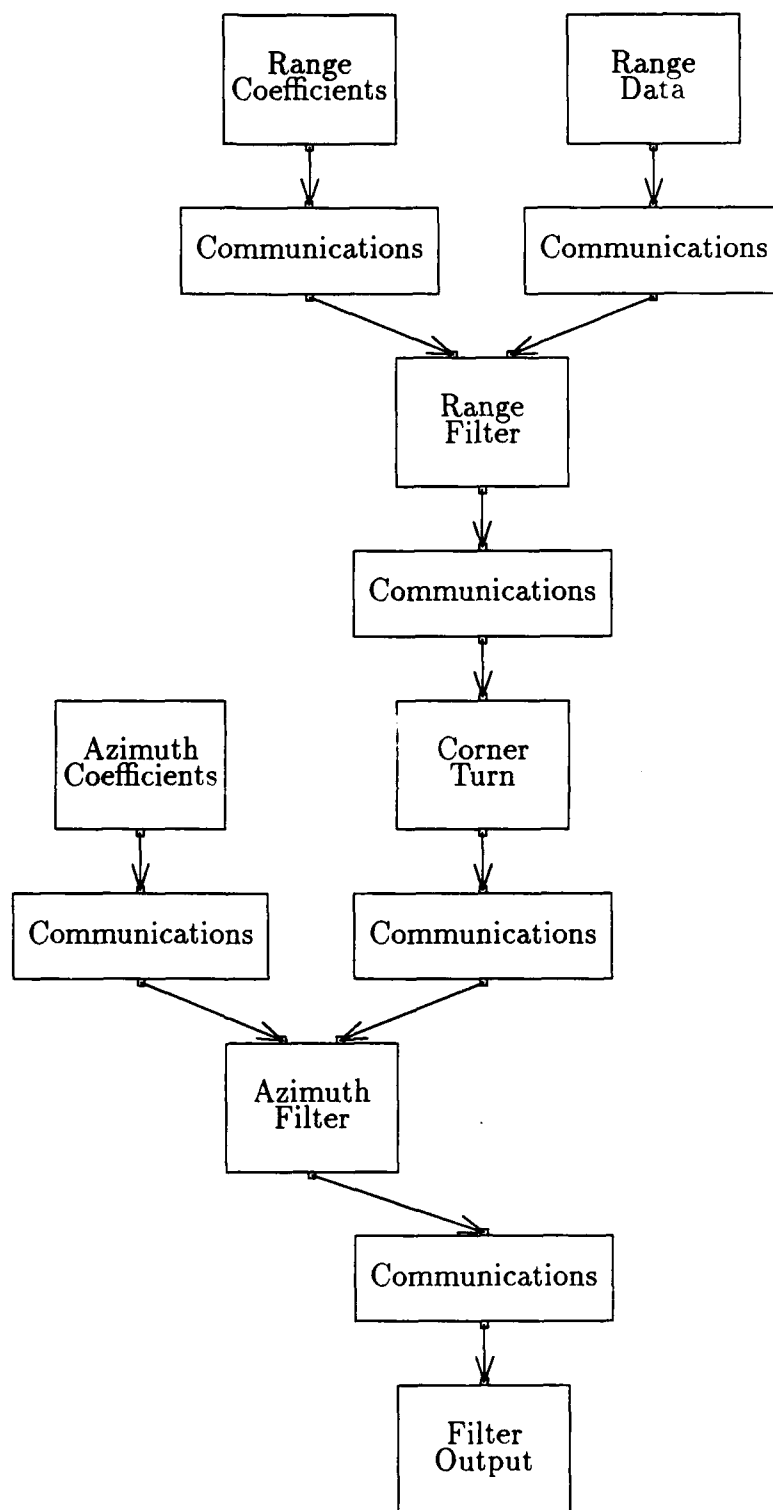


Figure 2.4: SAR Graph Adapted to FTPP Architecture

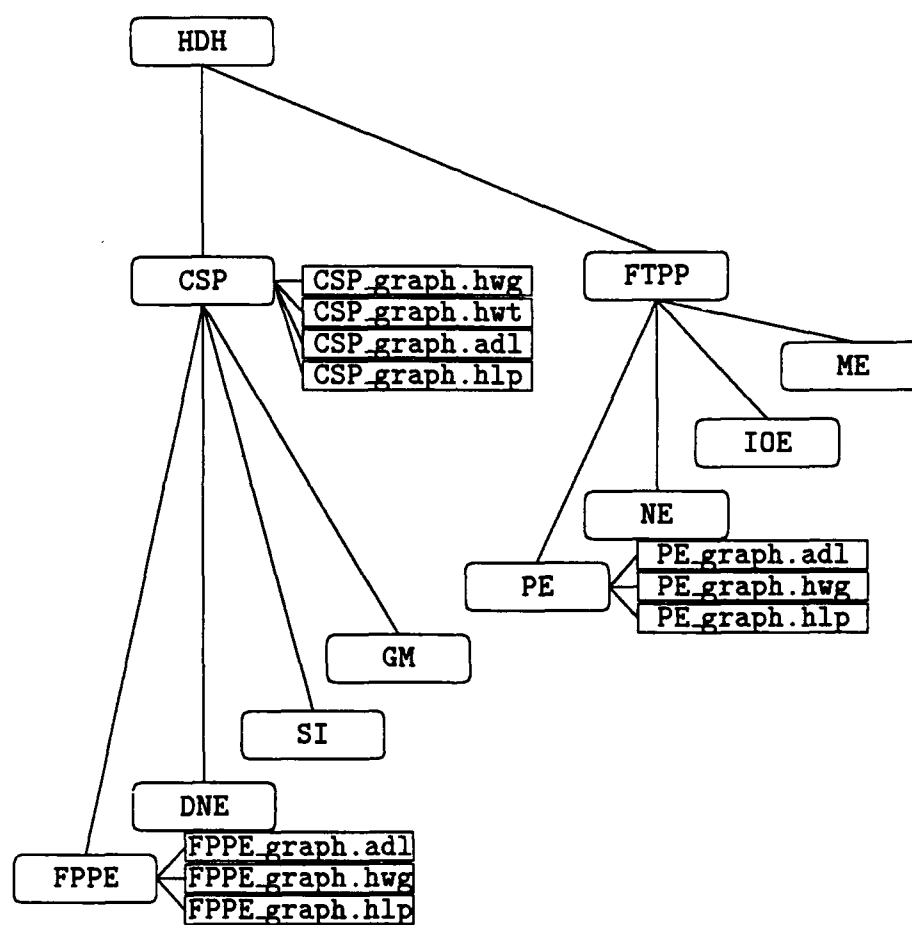


Figure 2.5: Hardware Domain Hierarchy

The modules of the CSP considered in this demonstration are the Floating Point Processing Element (FPPE), the Global Memory (GM), the Sensor Interface (SI), the Element Supervisory Unit (ESU), the TM and PI Bus, and the Data Network Element (DNE).

The components of the FPHP are the Processing Element (PE), the Input/Output Element (IOE), the Network Element (NE), and the Memory Element (ME). All components are grouped into clusters which consists of Fault-Masking Groups (FMG) which are elements tightly-coupled to preserve highly-reliable characteristics.

For the demonstration, the hardware modules onto which the SAR functions are mapped fall into four classes: processing element, memory element, input/output element and bus. Table 2.1 contains the architecture modules and their classifications.

Hardware Classification (<i>module_class</i>)	Architecture	
	CSP	FTPP
Processing Element (PE)	FPPE	PE
Memory Element (ME)	GM	ME
I/O Element (IOE)	SI	IOE
Bus (BUS)	DNE	NE

Table 2.1: Architecture Hardware Module Classifications

2.6.2.3 The Transformation Rule Base

This section describes the graph transformation rules which are used to demonstrate the capability of GTS to complete the design of the system initially defined by Figure 2.3. Two sets of transformation rules were developed for the demonstration.

The first set is designed to optimize the performance of any algorithm graph built from filters for a generic multiprocessor architecture. This set of rules is used to customize the SAR graph by:

- Implementing the filters as a pipeline with an FFT, a vector multiply, and an inverse FFT.
- Partitioning the FFT operations into smaller parallel FFTs which can be distributed on multiple processors.
- Segmenting the corner-turning memories to allow parallel access.

The second set is designed to adapt a generic algorithm to the FTPP architecture. This set of rules is used to customize the SAR graph by:

- Implementing each processing function with Triple Modular Redundancy (TMR).
- Including the communications costs for transmission of the data both between fault containment regions and between FTPP processor clusters.

GTS selects and applies a sequence of graph transformations to the algorithm data flow graph in order to develop a final design which is specialized for the input goals and/or hardware constraints that have been specified. GTS transform rules include evaluation rules which evaluate the filter process and determine what transform to select to implement it.

2.7 AIVD Reviews

The AIVD project was punctuated by a series of project reviews. The AIVD project was unusual in that each review included one or more prototype demonstrations. These demonstrations allowed the project reviewers to get a hands-on feel for the system that was evolving and to provide real-time direction in the design of the system.

The reviews were:

Kickoff Meeting At the kickoff meeting, an initial version of the Graph Transformation System was demonstrated by Octy. In this version, the rules were encoded in Prolog, and only a single rule and a single match location could be processed. At this review, the plans for the proposed AIVD development were presented and discussed.

Preliminary Design Review The major focus of the Preliminary Design Review was to present the design of AIVD. Two documents were presented: a draft User Manual and a System Requirements document. The User's Manual focused on the issues of the user interface for the system. The design of the Attribute Definition Language was presented in detail, and the relationship between the ADL and the VHDL was discussed.

Critical Design Review At the Critical Design Review, three tools were demonstrated: a new version of the Graph Transformation System, an initial version of the Attribute Definition Language Evaluator, and an initial version of the Intelligent User Interface. These tools were demonstrated in concert with the ADAS tools processing a model of the AWACS Track While Scan algorithm.

The Graph Transformation System was shown reading the ADAS graph version of the Transformation Rule Base. A first iteration of the AIVD specifications was presented.

Test Plan Review At the Test Plan Review, an initial version of the enhanced Software to Hardware Mapper was demonstrated, along with revised implementations of the Attribute Definition Language Evaluator and the Graph Transformation System. The revised AIVD specification document was presented and discussed, and plans for testing AIVD were described.

Final Review At the Final Review, the AIVD workstation was installed at Army LABCOM, and all of the tools were shown working together to perform the system demonstration.

3. THE AIVD DEVELOPMENT APPROACH

3.1 Overview of Approach

The AIVD development approach is based on four novel principles:

Parallel Construction of Prototypes and Specifications At each review of the AIVD project, different versions of the prototype tools were demonstrated. At the same time, each of the two major documents (the User's Manual and the System Specifications) went through two major revisions. This parallel development of prototypes and specifications ensured that the documents were consistent with the final delivered product, and that the review teams (and the tool developers) could effectively guide the design through feedback based on hands-on experience with the prototypes. This approach was particularly valuable given the geographic distribution of the development team. Defining a series of versions of the tools, so that the system could be "grown" rather than "built," required more design effort at the beginning, but ensured a product that met expectations at the end of the project.

Loosely Coupled Design In order to foster parallel development of different tools, and to reduce the effort required for integration of the tools, two decisions were made early in the project that defined the interfaces between the tools. All the tools would communicate through two mechanisms: the commercial ADAS data base files and commercial ADAS script files. These interfaces remained both simple and stable for the life of the project, and made integration of the tools relatively easy.

Based on Commercial ADAS The AIVD system was built around the latest commercial version of ADAS. Two of the ADAS tools were replaced. The Intelligent User Interface replaced the ADAS graph editor EDIGRAF. The enhanced Allocator of Software to Hardware replaced the commercial version of ASH. As a result of this development, all of the other commercial ADAS tools can be used with the AIVD tools. Either EDIGRAF or the IUI can be used to build the software graphs and the transformation rule base.

Application of AIVD on a Parallel Project An important resource for any tool builder is a set of tool users who are willing to work with prototype versions of the tools. The AIVD development team was fortunate to have application projects available which needed the capabilities of the AIVD tools even in prototype form. The results of using one of the AIVD tools is described below.

3.2 An Application of AIVD

During the development of the AIVD tools, a parallel project was started at RTI called "The Design and Assessment of High Performance, High Reliability Systems" (DAHPHRS). This project was done under NASA contract NAS1-17964, Task 20. The contract monitors were Lt. Gale Paeper from RADC and Wayne Bryant from NASA/LaRC. The RTI project leader was Charlotte Scheper. There were two sub-contractors: Honeywell SRC under the direction of Dr. Sudah Yalamanchili and Virginia Polytechnic Institute under the direction of Dr. Gail Gray.

DAHPHRS involved the evaluation of the performance and reliability of two algorithms on three different architectures. The Attribute Definition Language was used to capture the performance effects of the mission parameters. The fault-tolerant transformations and communications transformations developed for the AIVD final demonstration were based on the needs of this project.

3.2.1 Description of Application

The DAHPHRS application consisted of two battle management C^3I algorithms. The algorithm described below is used to manage Directed Energy Weapons (DEWs). The processing is designed to allocate DEWs to targets (which are enemy ICBMs). The processing will be done in space and on a battle management platform, which will be communicating with multiple sensor platforms and weapons platforms. The sensor platforms (the BSTS's) will be acquiring and tracking the targets while they are in boost phase. Thus the information available to the battle management algorithm will be information on the expected position of the targets, the target values, and information about the status and characteristics of the weapons.

The performance requirements of this system were determined by response time requirements. In general, 5 seconds would be allowed for the system to respond to a request for target sequencing instructions. A fall-back requirement was a response time of 30 seconds.

There were four major mission parameters defined for the problem:

The Number of Targets The number of targets ranged over the values of 25, 50, 100, and 200.

The Number of Weapons The number of weapons ranged over the values of 5 and 10.

The Number of Clusters The number of clusters was fixed at 10.

The Redundancy Level The redundancy level is the number of shots that must be taken at each target to ensure a kill. For this study, the redundancy level was fixed at 3.

These parameters were used to determine processing times and data structure sizes. In turn, the data structure sizes were used to determine communication costs.

3.2.2 The High Level Model

The top level data flow model of the Weapons to Targets Assignment and Target Sequencing (WTA/TS) algorithm is shown in Figure 3.1. There are four major functions shown:

Target Cluster Definition This function groups the targets into clusters based on the distance between the targets. The DEW's cannot be aimed rapidly in widely divergent directions. In order to minimize the time between shots and thus maximize the number of shots taken by each DEW, the targets are clustered, and then weapons are assigned to targets within a cluster.

Weapon to Cluster Assignment This function assigns the weapons to the clusters. In order to achieve the required level of redundancy, each cluster must be assigned multiple weapons. The number of weapons depends upon the number of targets in the cluster and on the level of redundancy. No weapon is assigned to more than one cluster.

Weapon Assignment This function assigns the targets in the cluster to the weapons committed to the cluster. This function can be performed in parallel, with the assignments for each cluster proceeding independently.

Target Sequencing This function determines the sequence of shots for each weapon to ensure that each target assigned to the weapon is shot at the best time to eliminate the target. This is a complex problem, since both the targets and the weapons are moving independently, and the weapon will have a limited time for firing. This function can be performed in parallel, with the assignments for each weapon proceeding independently.

These functions work on seven major data structures:

Interdistance Matrix The interdistance matrix describes the distances between the targets.

Target Processing Time The target processing time matrix describes how long the directed energy beam from each weapon must dwell on each target in order to disable the target.

Weapon Slew Time The weapon slew time matrix describes how long it will take to shift each weapon to point at each target.

Kill Probability Matrix The kill probability matrix describes the probability that each weapon will destroy each target.

Utilization Matrix This matrix describes how much energy each weapon must expend to destroy each target.

Target Due Date Vector This vector describes the time when this target will be out of range, hidden, or will complete boost phase (and hence be much more difficult to detect).

Target Value Vector This vector describes the priority for destroying each target.

Figure 3.2 shows the Attribute Definition Language program for the WTA/TS graph. This program defines the values of the mission parameters, collects the hardware parameters, and defines the data structure sizes in terms of the mission parameters. The variables defined in the graph program are inherited into any node or arc ADL file in the hierarchy that wants to use this information. Figure 3.5 shows the ADL program for a node in the Target Cluster Definition subgraph. This ADL program uses the variables which it has inherited from the WTA/TS graph ADL program.

The first step was to build an ADAS simulation of the WTA/TS graph assuming a single 1 MIPS processor. This configuration clearly did not meet the performance requirements. With 25 targets and 10 weapons, execution time exceeded 10 seconds. With 25 targets and 20 weapons, execution time approached 40 seconds. With 200 targets and 10 weapons, execution time exceeded 340 seconds. With 200 targets and 20 weapons, execution time exceeded 360 seconds. This study also pointed out the relative dependence of the model on different parameters. Changes in the number of weapons affected the response time linearly, while changes in the number of targets affected the response time exponentially.

The second step was to analyze the utilization of the processing resources to determine the bottlenecks and see if a parallel implementation could improve performance sufficiently to meet the requirements. Figure 3.3 shows the distribution of resource utilization by the different major functions at different numbers of targets. An interesting phenomenon shows up. For small numbers of targets (e.g., 25) the Weapons to Target Cluster Assignment Function uses most of the single processor's time, but

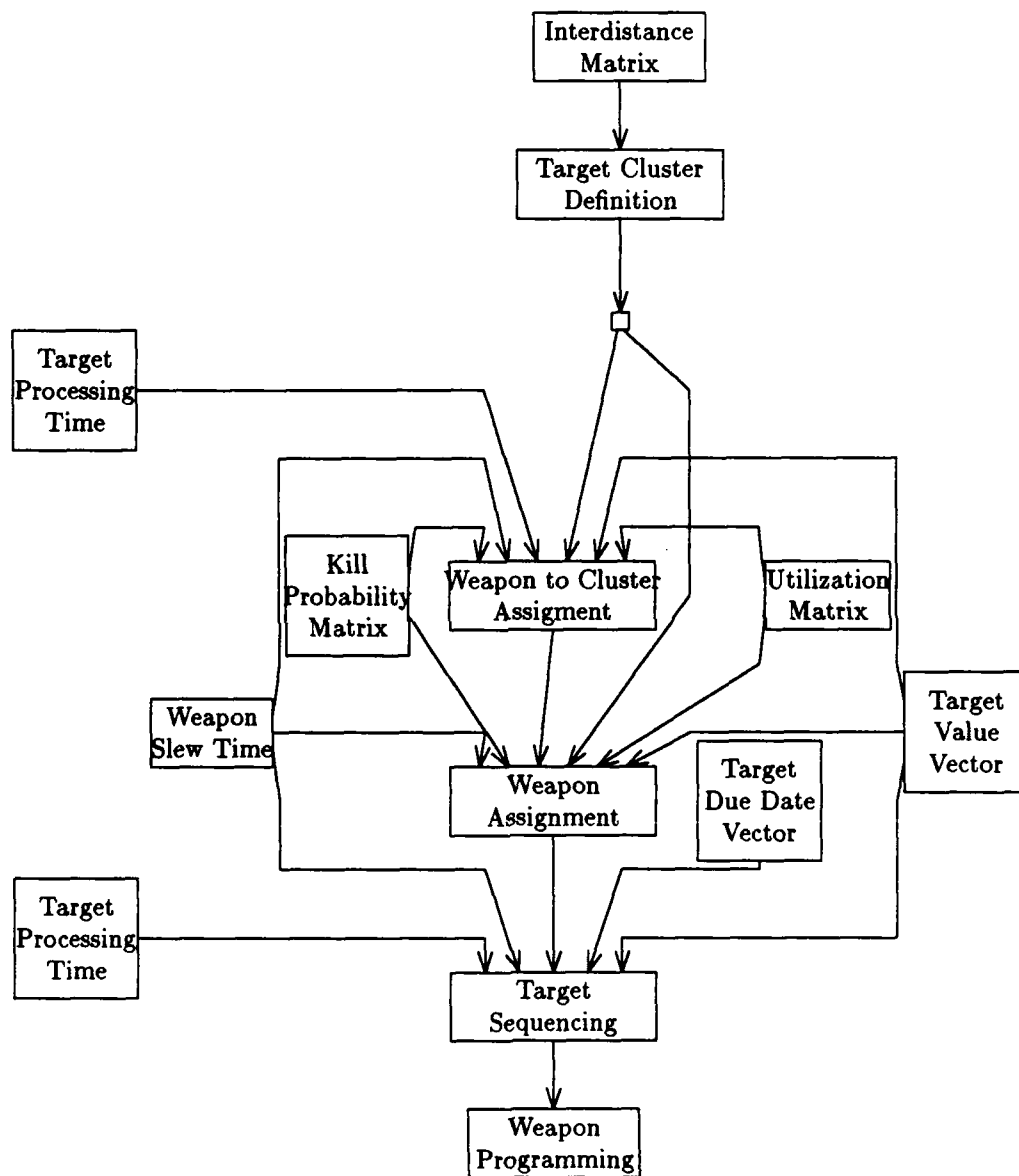


Figure 3.1: Top Level ADAS Graph of WTA/TS Algorithm

```
int: -- Key Input Parameter Declaration
nbr_targets, -- The number of targets (N).
nbr_clusters, -- The number of clusters (C)
nbr_weapons, -- The number of weapons (W).
redundancy, -- Weapon assignment redundancy (R).

-- Hardware Parameter Declaration
fpt_mult, -- Floating point MPYs per millisecond
mips, -- Instructions per millisecond
io_rate, -- Floating point I/Os per millisecond
read_fpt; -- Floating point READs per millisecond

-- Data Structure Sizing Calculations
dim_size = nbr_targets * nbr_targets;
di_size = nbr_targets;
Gc_size = nbr_clusters;
Lwc_size = redundancy * nbr_clusters;
Lwt_size = redundancy * nbr_targets;
pij_size = nbr_targets * nbr_weapons;
Tij_size = nbr_targets * nbr_weapons;
tip_size = nbr_targets;
Uij_size = nbr_targets * nbr_weapons;
```

Figure 3.2: WTA/TS Graph ADL Program

for larger numbers of targets, the Target Cluster Definition function uses most of the resources. A second aspect is that there are simple techniques for making the Weapon to Target Assignment and the Target Sequencing functions parallel, but they use very little resources. In order to meet the response time requirements, the Target Cluster Definition function must be made parallel.

3.2.3 The Detailed Model

The third step was to investigate the Target Cluster Definition function to see what could be done to make it execute in parallel. Figure 3.4 shows the data flow of the TCD function. There are three major modules in the TCD function:

Gradient, LaGrange, and Median This module uses linear programming techniques to create clusters based on the distances between targets. The median distances between targets in a cluster are also computed. The linear programming solution defines probabilities that targets should be assigned to clusters.

Cluster Formation This module uses the median information and the cluster probabilities to assign targets to clusters. Once the clusters are defined, the clustering is evaluated. If the cluster size is sufficiently close to the goal, then the cluster definitions are passed on to feasible cluster definition. Otherwise, another linear programming iteration is performed.

Feasible Cluster Definition This module creates lists of target assignments for each cluster.

The TCD function uses only one major data structure, the Target Interdistance Matrix. The structure of the algorithm features a loop, where a linear programming approach to the solution is iteratively refined in order to achieve the integer solution desired.

Figure 3.5 shows the Attribute Definition Language program for the GLM node. This program uses the values of the mission parameters and data structure sizes which it inherits from the WTA/TS graph ADL. It also inherits the TCD iteration count TCD_LIc from the TCD graph. The inherited values are used to determine the firing delay attribute for the node and the consume, threshold, initial, and produce attributes for the ports of the GLM node. This allows the user of the system to be hidden from the ADAS attributes and terminology and to concentrate instead on the definition of the mission and hardware parameters.

Figure 3.6 shows the relative use of a single processor's computing resources by the three different modules as a function of the number of targets. While the relative

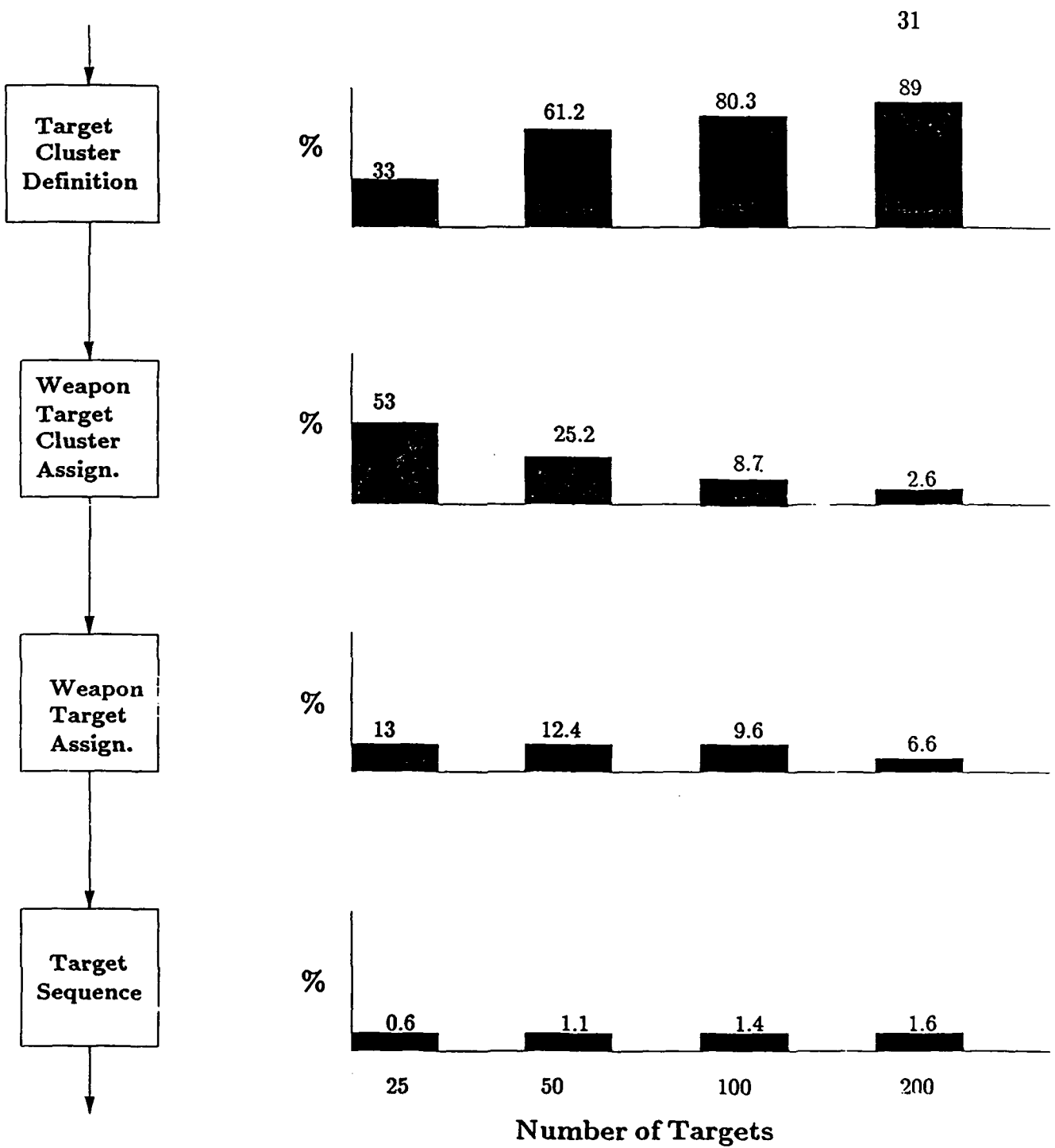


Figure 3.3: WTA/TS Process Utilization

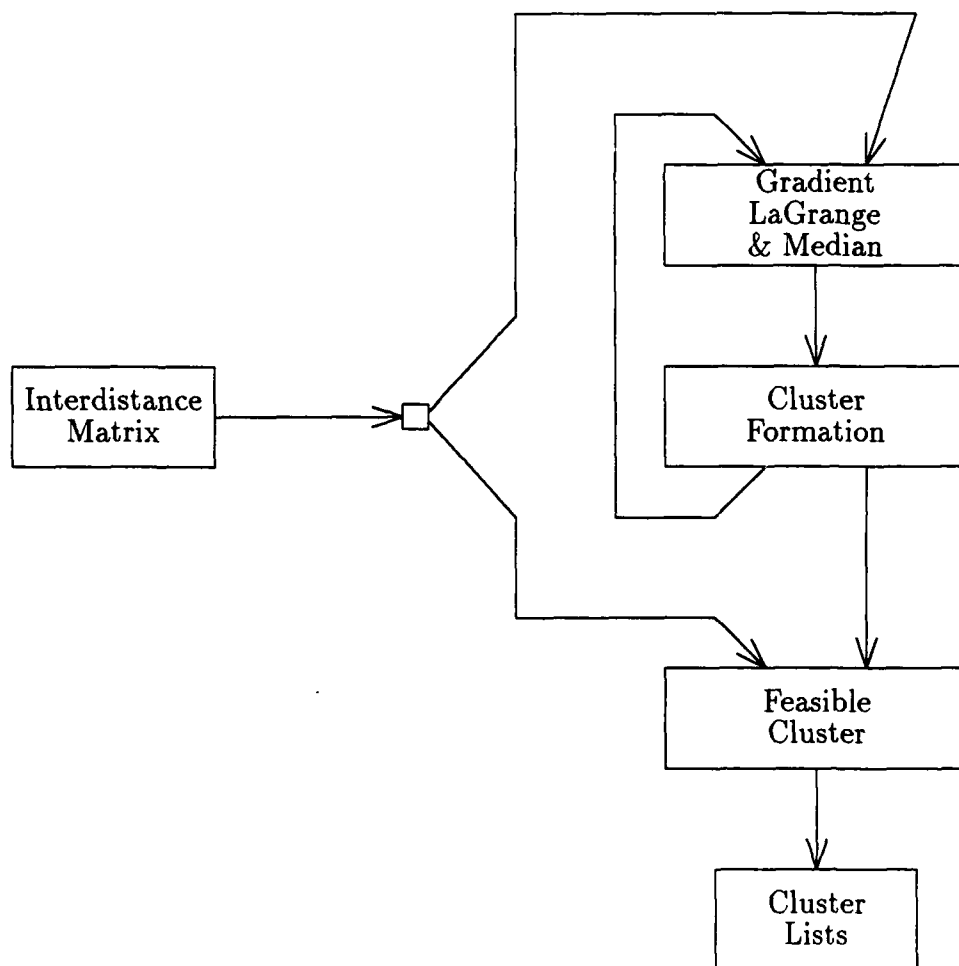


Figure 3.4: ADAS Subgraph of Target Cluster Definition Algorithm

```

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

-- Consumes an entire dim matrix each iteration
token_consume_rate(in0) = TCD_LIc * xim_size;
firing_threshold(in0) = token_consume_rate(in0);
-- initial_token_count(in0) = token_consume_rate(in0);
-- Consumes an entire xim matrix each iteration
token_consume_rate(in1) = dim_size;
firing_threshold(in1) = token_consume_rate(in1);
-- Produces an entire Cim matrix each iteration
token_produce_rate(out0) = TCD_LIc * Cim_size;

```

Figure 3.5: GLM Node ADL Program

utilizations of the functions in the WTA/TS graph varied considerably as a function of the number of targets, the utilizations of the modules within the TCD remained relatively constant. Unfortunately, while there are ways of partitioning the GLM module so that it can execute in parallel, there are no obvious ways of partitioning the Cluster Formation module. Since Cluster Formation has the highest utilization, making GLM parallel can improve performance by at most 40%. This limits the benefits of parallel architectures for this problem.

3.2.4 Software Partitioning and Parallelism

The algorithms which had been developed were originally implemented on a sequential processor. In order to meet the response time performance requirements, the algorithms needed to be adapted to take advantage of the available computing resources in the multiprocessor environment addressed by the DAHPHRS study. Two techniques for improving the performance of the algorithm were evaluated: pipelining and parallelism.

A cursory evaluation of pipelining the algorithms at the macro level indicated that this approach would not be effective. The primary issue at the macro level

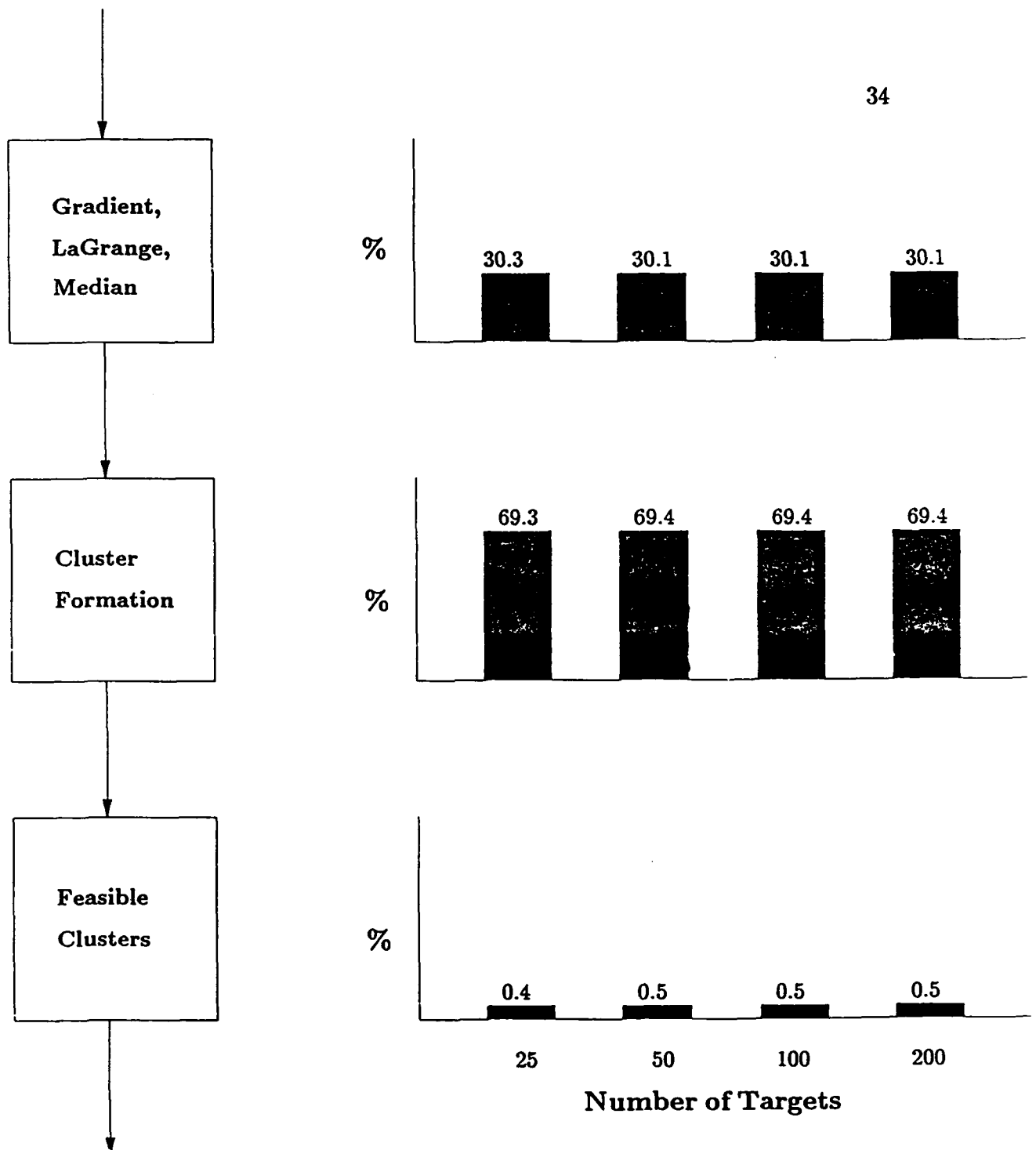


Figure 3.6: Target Clustering Process Utilization

was response time, not throughput. Pipelining increases system throughput at the possible cost of an increase in response time. At a lower level of detail, pipelining could be very effective if the individual processors supported pipelining.

At the macro level, the algorithm provided several opportunities for coarse grain parallelism. These opportunities were captured in the form of parallel subgraphs for the top level operations. These parallel subgraphs were designed to serve as models for graph transformation rules. Simulation of these models revealed that the utilization of some inherently sequential tasks limited the benefits of parallelism.

As described above, the limiting factor in this case is the ability to make the Cluster Formation module parallel. Without a parallel version of Cluster Formation, at most a factor of 2 speedup can be achieved. This is not nearly enough to achieve the desired performance requirements.

3.2.5 Communication and Fault Tolerance

A second set of issues related to this project is the cost of fault tolerance. The cost of fault tolerance was evaluated in the context of the FTPP architecture, which provides both parallel processing and fault tolerance. The FTPP approach to fault tolerance involves the use of Triple Modular Redundancy and quadruplicate voting. There is an obvious cost in terms of additional hardware required for TMR, but there is also a significant performance cost imposed by the need for frequent voting of intermediate results.

In order to evaluate the overhead costs of communication and fault tolerance, a parallel algorithm was constructed (which introduced communications costs) and mapped onto a FTPP architecture with multiple clusters and multiple TMR processor groups per cluster. The GLM algorithm was partitioned into four parallel processes. Figure 3.7 shows three different mappings of the parallel version of the GLM and shows the additional costs imposed by the fault tolerance algorithms of the FTPP. The three different mappings are based on different distances for the communications. The first pair of bars shows a one cluster system, where no intercluster communications are required. The second pair of bars shows a two cluster system, where half of the interprocessor communication requires intercluster communications. The third pair of bars shows a four cluster system, where all interprocessor communications requires intercluster communications, and the longest communications must traverse of four clusters, including the source and the sink clusters. The cost without fault tolerance was modeled by assuming that the fault tolerance algorithms could be performed in zero time.

Three conclusions emerge from examining Figure 3.7. The first is that there are significant overhead costs in distributing the parallel modules across multiple clus-

ters in the FTPP architecture. The second is that significant performance overhead is associated with the FTPP implementation of fault-tolerant communications. The third is that the proportion of execution time associated with the FTPP implementation of fault-tolerant communications increases as the distance of the communication increases. This makes intercluster communication involving more than two clusters very unattractive.

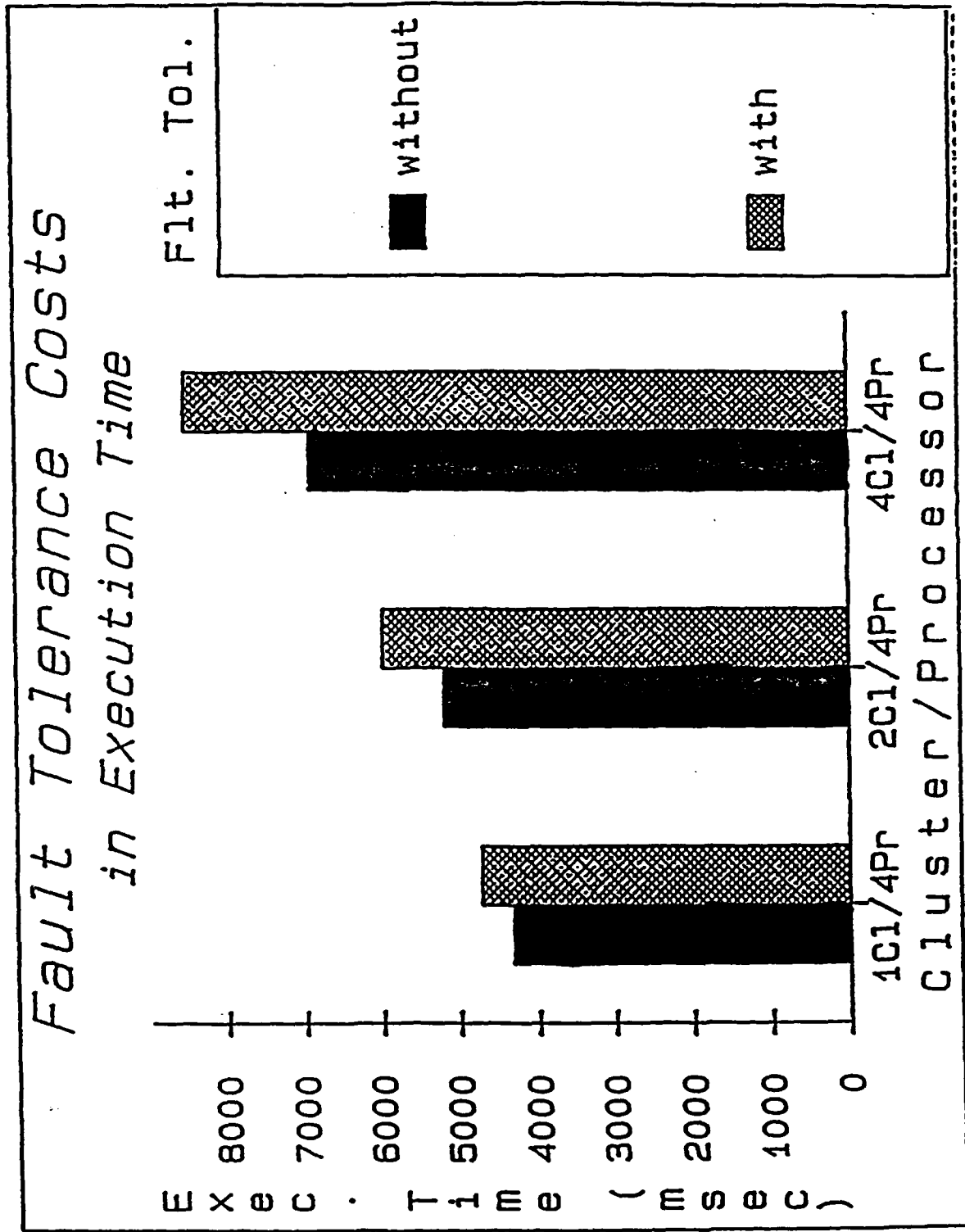


Figure 3.7: Performance Costs of Fault Tolerance

4. FURTHER WORK

The goal of the AIVD effort was to develop prototype tools which would use artificial intelligence techniques to extend the ADAS tool set capabilities. Experience with AIVD indicates that we have succeeded in meeting our goal. There is still a long way to go before AIVD is a standard part of every system designer's toolkit. In this section, some of the steps that still need to be taken to make AIVD reach its full potential are discussed. These steps fall into three categories:

- Further application of AIVD tools.
- Enhancements of AIVD tools.
- Converting AIVD to a production quality system.

4.1 Further Application of AIVD Tools

RTI will continue to use the AIVD tools in their current configuration in order to support its work on the modeling of systems. Several projects will begin shortly which will make use of these capabilities. These projects range from further work on *C³I* systems for the Strategic Defense Initiative, to fault-tolerant, distributed avionics systems for military and commercial aircraft.

AIVD is designed to support the use of reusable libraries of both algorithms and architectures. As RTI uses the AIVD tools, it will be accumulating design experience in the form of algorithm and architecture libraries and rule bases for using those libraries.

4.2 Further Enhancement of AIVD Tools

Experience with AIVD points out the need for more work in developing these tools to make them more "user friendly." This section describes some of those techniques.

4.2.1 Develop a System Experiment Environment

Some of the early design work on AIVD focussed on the need for a System Experiment Environment (SEE). In keeping with the AIVD design approach of simplifying the interfaces, a very rudimentary version of an SEE was developed for the AIVD demo. The demo uses VMS command files to construct scenarios for running multiple tools,

each in its own window. While this approach is sufficient for the demo, a full-time user would prefer an integrated environment, where the equivalent of the command files are generated graphically and interpreted interactively. This type of environment requires extensive work in integrating the AIVD tools which were developed independently into a common user environment.

The SEE would support batch experiments, where the user interactively defines a complete experiment *a priori* and then submits the experiment definition to the workstation operating system for a batch run. The batch run will produce the final edited results without intervention by the user. The SEE should also support interactive experiment execution where the user can monitor the experimental process and modify the approach during the middle of the experiment.

In order to avoid overwhelming the user with huge quantities of data, the SEE should provide a set of data reduction techniques which will allow the user to edit the results from many simulations into tables which contain critical results. These tables can be read by other tools or plotted in formats readily accessible to the user.

4.2.2 Enhance the Graph Transformation System

The current Graph Transformation System provides the user with a powerful tool for creating system data flow graphs. More experience is needed to fully understand the capabilities and limitations of the current tool. However, even at this stage in its use, there are several parts of the system which could be improved. One is the way that the transformation process is controlled. Both the rule selection and the match and transform evaluation processes need to be extended. In particular, an enhanced GTS should be able to compare specific instantiations of different patterns. Different representations for higher level control techniques (i.e., the selection of pattern and transformation sets) need to be evaluated and implemented. Also, an understanding of how to define very high-level goals that can be understood by both the mapping program and GTS needs to be developed.

An enhanced GTS should permit more interaction with the mapping tool and the simulator. In particular, an enhanced GTS should be able to process the same set of design goals and constraints that the mapping tool uses. Also, GTS should get feedback from the mapping tool which will allow it to generate graphs which more accurately reflect the communications overhead for multihop transmissions.

Different pattern matching strategies need to be evaluated in order to reduce the total cost of pattern matching as part of the transformation process. Strategies which are cost-effective should be implemented.

Different mechanisms for user interaction with the GTS need to be evaluated so that a user can get a better understanding of where the transformation process stands.

These mechanisms include the display of the transformation status, the display of proposed matches, and the display of the current transformation rules.

An enhanced GTS should be able to access static performance analysis capabilities that have been developed for ADAS. These packages would allow GTS to estimate the performance of an intermediate transformed graph and suggest transformations based on the available performance information.

4.2.3 Enhance the Software to Hardware Mapping Program

More work is needed in enhancing the ASH in order to allow the user to interact with the ASH and in order to allow the ASH to deal with additional real world constraints.

Further work is needed to improve communications between the ASH and the GTS. In particular, the insertion of communications nodes, which is done by the GTS, needs to be based on information provided by ASH, including mapping information. This could lead to an iterative process where GTS and ASH are called alternatively in order to create a graph which can be mapped properly and can also meet the performance modeling capabilities which the communications node libraries can provide.

Different mapping goals and constraints shall be investigated to determine if they can be used effectively in the mapping process. One particular area of importance for new goals and constraints is in the area of supporting fault-tolerant computing. The mapping program needs to support constraints which require that certain computations be mapped to different fault-masking groups of the architecture. These types of constraints will be developed and implemented.

4.3 Converting AIVD to Product Quality

The current implementation of AIVD is a prototype. Prolog was used to provide rapid prototyping capabilities. All of the AIVD tools should be converted to C in order to allow portability, better performance, and compatibility with the other ADAS tools. The decision to build AVID as a set of loosely coupled tools needs to be reversed in a product version. Finally, a great deal more testing and better error handling capabilities are needed.

5. CONCLUSIONS

The goal of the AIVD effort was to develop prototype tools which would use artificial intelligence techniques to extend the ADAS tool set capabilities. Preliminary experience with AIVD indicates that we have succeeded in meeting this goal. Feedback at the AIVD demo indicates that many users believe that AIVD will provide a major increase in the productivity of ADAS users. This improvement in productivity will arise from many factors. The use of the ADL and GTS will reduce the amount of time that users spend on tedious, repetitive, and error-prone steps. The use of ADL and GTS to build models will not only reduce the work for users, but will also help to document the models that are produced. Furthermore, the formal documentation provided by the ADL descriptions and the rule bases will help users in verifying that the models they build are correct. Finally, the ADL descriptions and GTS transformations will help make ADAS models much more reusable. It is in this final capability that the largest potential for increased productivity lies. However, the greatest benefits of reusability will not be achieved until libraries of algorithm and architecture descriptions have been constructed. AIVD provides some tools for building those libraries, but other projects will have to populate them.

The AIVD experience seems to validate the development approach used for this project. A focus on prototypes, and a pattern of supporting multiple releases of only two documents, rather than a more formal development based on freezing requirements, specifications, designs, etc., paid off. The use of very simple interfaces may have reduced the convenience of the tools, but it made integration of the tools into a prototype system much easier.

Finally, although AIVD accomplished a lot, there is much more that needs to be done. AIVD defines a skeleton for algorithm and architecture libraries, but did not populate them. Further work is needed in integrating the tools into a System Experiment Environment.