PAWS: A Performance Evaluation Tool for Parallel Computing Systems

Daniel Pease, Arif Ghafoor, Ishfaq Ahmad,
David L. Andrews, Kamal Foudil-Bey, Thomas E. Karpinski,
Mohammad A. Mikki, and Mohamed Zerrouki
Syracuse University

PAWS (Parallel Assessment Window System) provides an interactive user-friendly environment for analysis of existing, prototype, and conceptual machine architectures running a common application.

Fifteen years ago, most large-scale scientific and engineering computations were performed on sequential von Neumann machines. Comparisons among these machines focused on running sets of common benchmarks and on ranking the machines based on the number of instructions executed per second.

However, as the number of commercial systems increased, so did the diversity of their architectural design. As each new architecture diverged from the classical von Neumann model, new languages and annotated versions of older sequential languages were developed for execution on these new machines. This made it difficult to run a standard benchmark. Not only did each benchmark require translation into each language, but the translation process and newer optimizing compilers obscured the relative merit of the results.

To date, no formal methods allow comparisons among different machines running a single common application. Furthermore, code is generally not portable among different parallel processing machines. This forces applications to be recoded for each language and each machine.

PAWS (Parallel Assessment Window System) is an experimental system for performing machine evaluation and comparisons. As shown in Figure 1, PAWS provides a user-friendly X window-based environment that allows comparisons among vastly different machines running common applications.

Figure 2 shows the PAWS block diagram, which consists of four tools: the application tool, the architectural characterization tool, the performance assessment tool, and the interactive graphical display tool. Through the application characterization tool, PAWS translates applications written in a high-level source language into a single data dependence graph. This allows users to view their applications' attributes. The dataflow graph is a machine-independent intermediate representation that can map onto different architectures.

The architecture characterization tool allows users to create, store, and retrieve descriptions of machines in a database. This approach permits users to evaluate conceptual machines before building any hardware.

The performance assessment tool (PAT) generates profile plots through the interactive graphical display tool (IGDT). It shows both the ideal parallelism inherent in the machine-independent dataflow graph and the predicted parallelism of the partitioned dataflow graph on the target machine.

Using dataflow graphical languages on dataflow machines is not new,[1,2,3] but using them for parallel computer assessment through PAWS is original. A powerful PAWS feature is its ability to associate a graph's visual display during assessment.

0018-9162/91/0100-0018$01.00 © 1991 IEEE

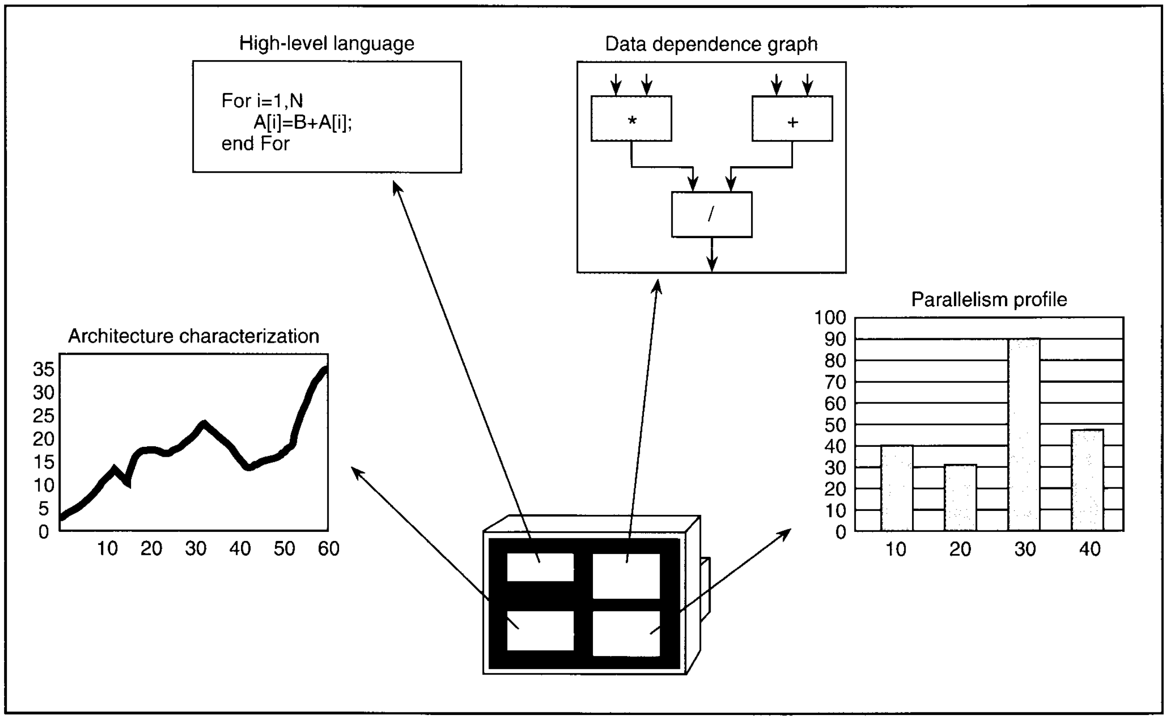


Figure 1. Parallel Assessment Window System (PAWS) environment.

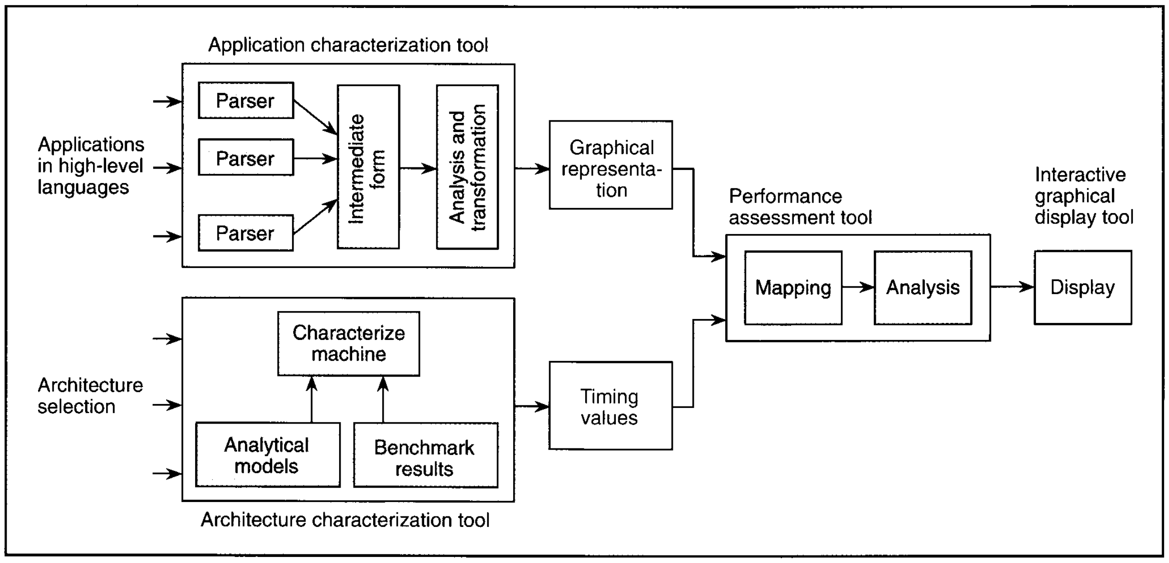

Application characterization tool
Applications in high-level languages
Parser
Parser
Parser
Intermediate form
Analysis and transformation
Graphical representa-tion
Performance assessment tool
Mapping
Analysis
Interactive graphical display tool
Display
Architecture selection
Characterize machine
Analytical models
Benchmark results
Timing values
Architecture characterization tool

Figure 2. PAWS block diagram.

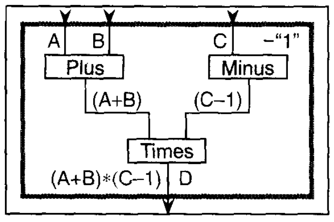
Through the windows environment, multiple windows can be opened to show the data dependence graph created from the original program, the evaluation machine's characteristics, and the application's performance metrics machine.

The application characterization tool

The application characterization tool provides the facility to evaluate the level and degree of an application's parallelism. Application characterization consists of a data dependency analysis to determine the order of program statements execution. It also identifies operations that may be executed in parallel. Parallelism within a pro-

```
Procedure Example
  A,B,C,D : integer;
begin
    D:= (A+B) * (C–1);
end Example;
```

(a)



(b)

Figure 3. Example of a simple Ada program and IF1 representation, (a) and (b).

gram may exist at different "granularity" levels. For example, arithmetic operations within a program may be independent, using different data values or variables in each expression. This parallelism level is called fine-grained parallelism. Conversely, if multiple functions, subroutines, or procedures are independent, then they can be executed concurrently. This parallelism level is called course-grained parallelism. The application characterization tool transforms programs written in high-level languages (currently Ada) into an intermediate graphical form that exposes the program's data dependencies and parallelism levels. Analysis is first performed on the intermediate form, providing insight into the program's structure and organization. The information produced during this analysis is then graphically displayed to the user for characterizing the application and mapping the application onto a machine.

The intermediate form defined to represent the program's data dependencies and parallelism must support parallelism at all granularity levels. Also, because the intermediate form is the target for mapping the application onto any machine, it should not be biased by a specific machine's limitations. Translation of applications written in a high-level language to an intermediate form with these attributes offers the following PAWS advantages:

- Provides a common target for any high-level language;
- Allows a single application to be analyzed on each machine characterized by the PAWS architecture characterization tool;
- Allows users to perform a machine-independent application analysis.

The intermediate form chosen for PAWS is Intermediate Form 1 (IF1),[4] a dataflow graphical language.

Dataflow graphs as intermediate forms. Dataflow graphs present the program's data dependencies and parallelism as graphs, clearly showing the program operations to the user. Figure 3 shows a simple Ada program and its equivalent IF1 PAWS-generated graph. The assignment to the variable D in Figure 3a is expressed in IF1 by labeling the output edge of the "times" node D. Further references to the variable D in Ada are translated in IF1 by connecting the edge labeled D to the corresponding node that uses D as an input. Figure 3 shows that Ada operations (A+B) and (C–1) can execute in parallel. The data dependency of the times operation on both the plus and minus operations is also exposed.

Description of IF1. IF1 is an acyclic graphical language developed as a target for SISAL (Streams and Iteration in a Single Assignment Language), a high-level functional programming language. IF1 hierarchical structure is well suited for graphical display. The IF1 language consists of nodes, edges, types, and graph boundaries. IF1 supports simple nodes and compound nodes. Simple nodes represent elementary operations such as addition, subtraction, and equality, while compound nodes represent complex constructs such as conditionals, loops, and the parallel construct Forall. As shown in Figure 3a, edges represents data values. The literal edges describe constants. Graph boundaries define functions, procedures, and compound nodes by encapsulating sets of nodes and edges.

Translation of Ada to IF1 in PAWS. PAWS translates Ada's regular grammar expressions into IF1 by decomposing them into IF1 operation nodes. Operands, data values, and intermediate results are transformed into IF1 edges. Table 1 shows fundamental Ada constructs mapping to their corresponding IF1 constructs. While translation of most constructs from Ada to IF1 is straightforward, issues arise in the translation due to fundamental language differences. Three main issues are compile time initializations, handling of global variables, and the single assignment rule imposed by dataflow computation. The PAWS techniques that handle these three issues are briefly described below.

While single variable initialization can be accomplished by substituting the desired initialization value into the first occurrence of the variable in the IF1 graph during runtime, compound data objects declared at compile time in Ada (arrays, records, etc.) cannot always be initialized using this approach. Instead, compound-data types are initialized with user-supplied input data during program execution. This eliminates the execution overhead caused by the compound data object's initialization during runtime.

Dataflow programs do not employ global memory. Instead, data values reside "locally" on edges between nodes. In PAWS, global variables in Ada programs

Table 1. Translation of some Ada contructs to IF1 constructs.

IF1 Ada	IF1
Arithmetic operations	Arithmetic nodes
Logical operations	Logical nodes
Array operations	AElement
	Node
	AReplace node
	ABuild node
	AFill node
If-then-else statement	Select compound node
Case statement	Select compound node
For loop statement	LoopA & forall nodes
While loop statement	LoopB & forall nodes
Functions & procedures	Subgraphs

are explicitly passed as parameters to each function using the global variables. This has the added benefit of transforming implied dependencies to explicit data dependencies.

Dataflow languages adhere to the single assignment rule, allowing variables to be written only once. PAWS solves this problem for Ada by introducing temporary variables. Once a variable is assigned a value, the value remains fixed for the remainder of the program. If reassignment is required, a temporary variable is created. After this reassignment, any read references to the original value is replaced with a reference to the temporary variable.

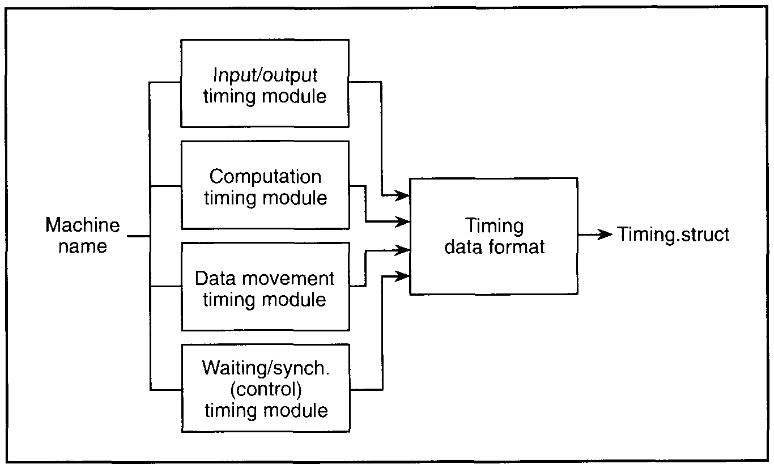

Machine name
Input/output timing module
Computation timing module
Data movement timing module
Waiting/synch. (control) timing module
Timing data format
Timing.struct

Figure 4. The top level of the parametric data structure.

The architecture characterization tool

Traditionally, machines have been classified as multiple instruction, multiple data (MIMD), single instruction, multiple data (SIMD), multiple instruction, single data (MISD), and single instruction, single data (SISD). These classifications differentiate among architectures based on instruction flow and dataflow. Also, at these classification levels, significant architectural diversity exists within each class. For example, within the MIMD class, both tightly coupled and loosely coupled systems exist. Although both are classified as MIMD, significant differences appear between the two system types. This affects how a program is written or partitioned onto the machine. The PAWS architecture characterization tool differentiates between machines within each of the classes defined above by a characterization based on

- the number and flexibility of different functional units;
- the number of processors;
- memory bandwidths and memory hierarchies; and
- the types of interprocessor communication mechanisms.

Our characterization method functionally partitions an architecture into computation, data movement and communication, I/O, and control.

An hierarchical organization of architectural parameters. Each category is continuously partitioned into subsystems until a subsystem is fine enough to be characterized by raw timing information. PAWS organizes this information in a tree data structure with the raw timing information in the leaf nodes. For timing information, we use an integrated approach based on low-level benchmarking that determines the machine's operation and behavior for each subsystem and application-dependent analytical models. Figure 4 shows the top level of this structure. As an example, the data movement subsystem shown in Figure 5 is partitioned into processor-to-processor, processor-to-memory, processor-to-peripherals, and memory-to-memory

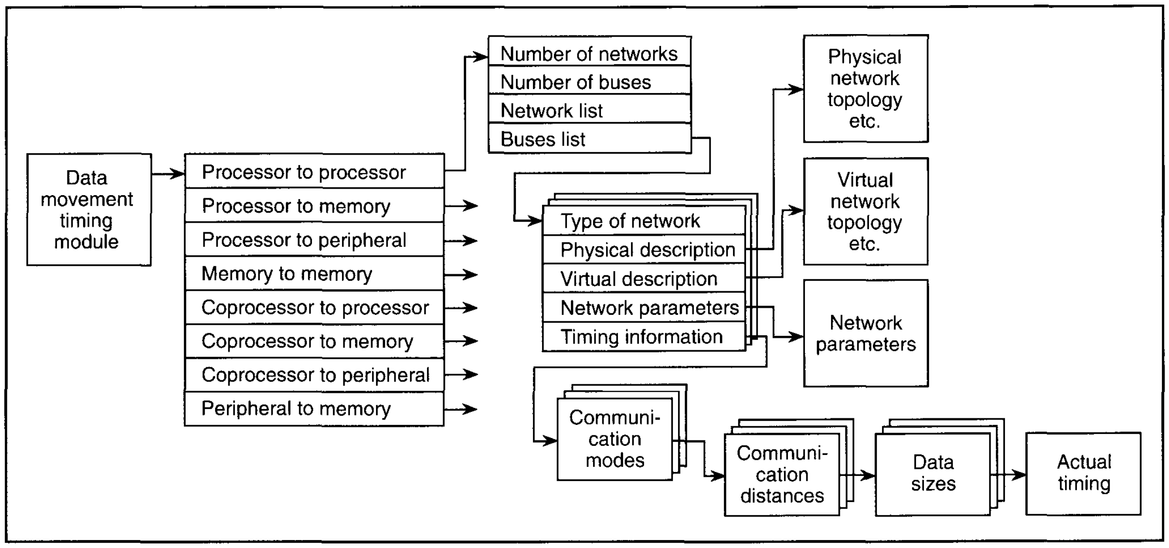


Figure 5. The data movement submodules.

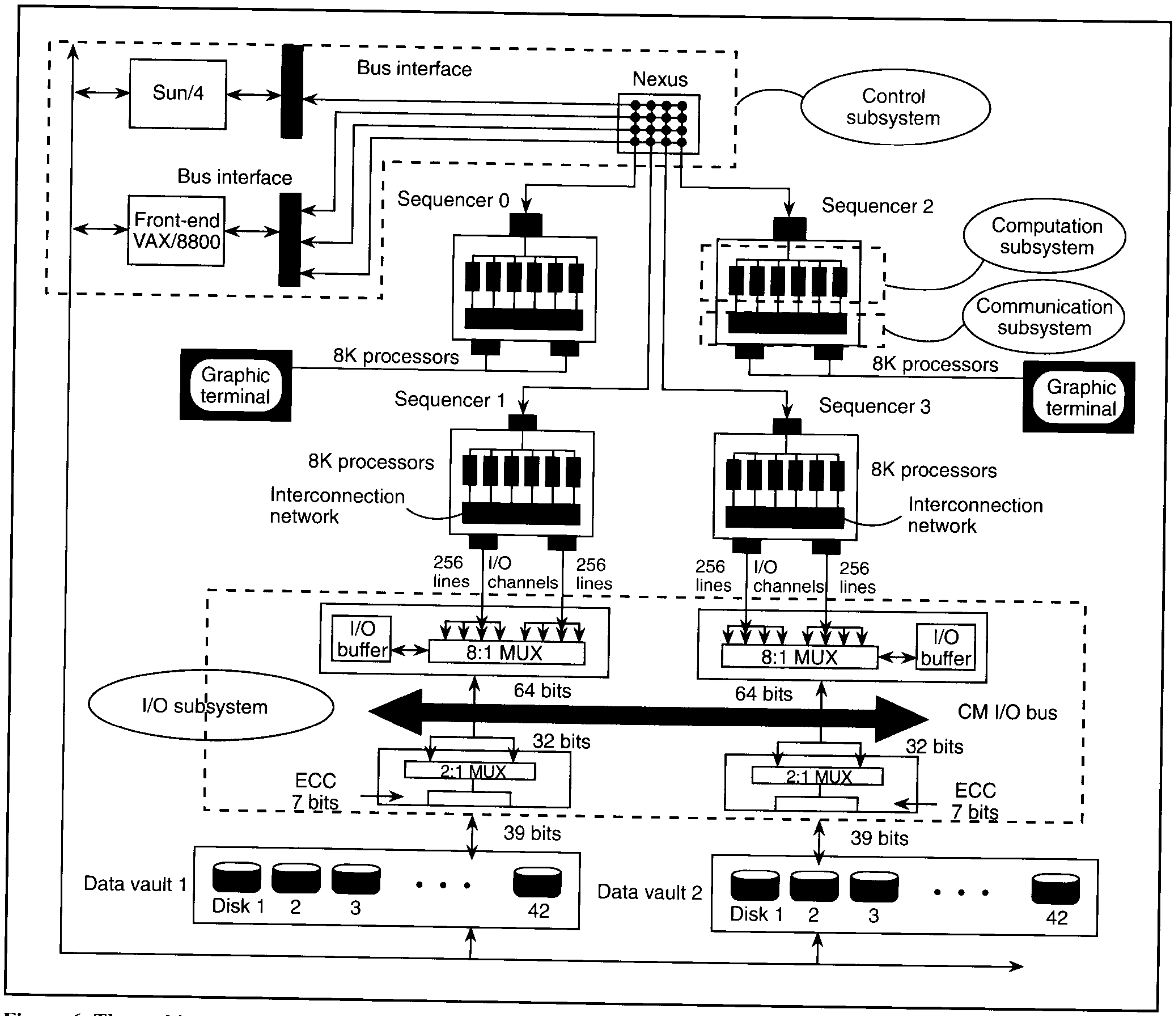


Figure 6. The architecture of CM-2 and its functional subsystems.

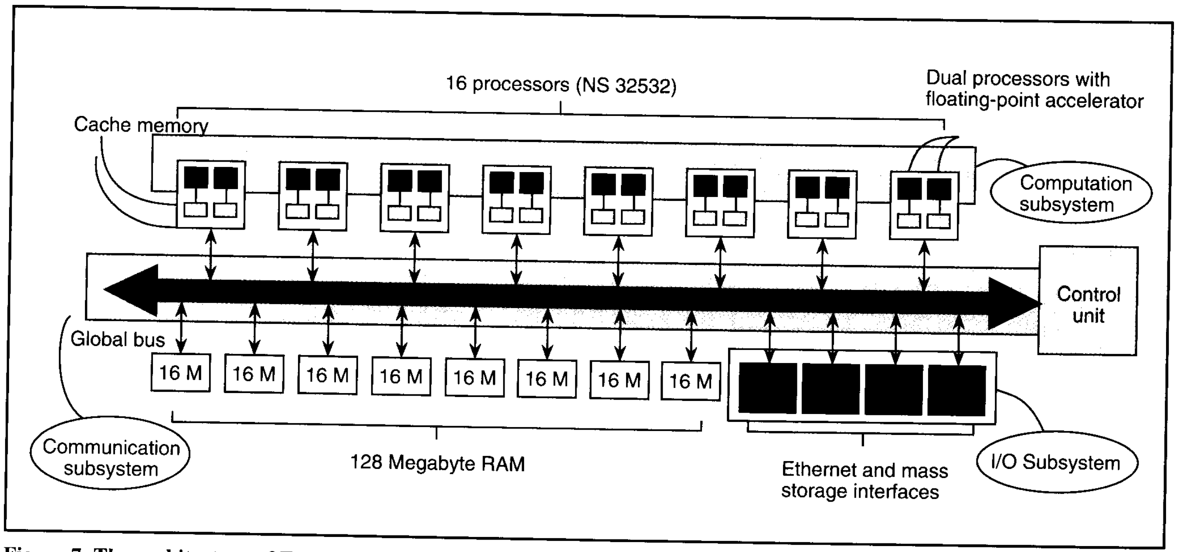


Figure 7. The architecture of Encore Multimax and its functional subsystems.

data movement subsystems. The data movement among processors can be achieved through various communication architectures such as multistage interconnection networks, bus systems, and link-oriented connections. The network subsystem describes the network's physical characteristics, such as topology. A partial decomposition of the data movement subsystem is shown in Figure 5.

The characterization of a particular machine may not need the whole data structure. Instead, the information necessary to fully characterize the machine may only require a few subtrees of the main data structure. Currently, PAWS characterizes a SIMD architecture, Thinking Machine's CM-2, and a MIMD architecture, the Encore Multimax. The CM-2, shown in Figure 6, is configured as a 32K processor machine. This figure shows the CM-2 logical partitioning into the top-level PAWS parametric data structure. The architecture of the Encore Multimax model 520[5] is shown in Figure 7, along with its various subsystems.

The block diagram of these machine's architectural characterization tool is shown in Figure 8. Users interact with this tool via the user-interactive interface module for selecting, synthesizing, or modifying the machine specification. The complete specification of any existing, conceptual, or prototype machine can be captured and stored in the PAWS architectural data structure.

Benchmarking and analytical models. To use the parametric data structure, the user interactively enters both static and dynamic timing values. Static timing values, such as arithmetic operations, are for operations that are uneffected by the runtime environment. These values are generally obtained through benchmarking. Several benchmark studies for the Connection Machine have been reported[6,7,8] and used for CM-2 characterization in PAWS. Dynamic timing values are effected by the runtime environment and are determined by analytical modeling. Levit proposed an analytical model for grid communications in the CM-2.[6] This model takes into account the geometry of physical and virtual processors,[6] the dimension of communication, and the data size. The user must provide such information to the architecture characterization tool to generate the dynamic timing values for the specified grid geometry. Similar techniques have been used to obtain the timing values for the Multimax.

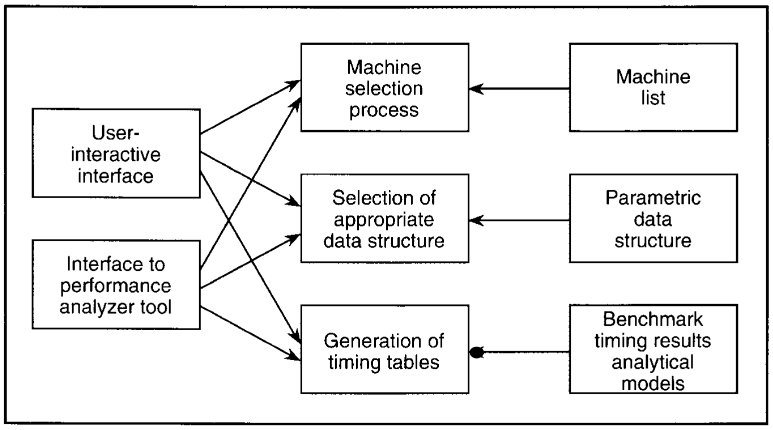

User-interactive interface
Interface to performance analyzer tool
Machine selection process
Selection of appropriate data structure
Generation of timing tables
Machine list
Parametric data structure
Benchmark timing results analytical models

Figure 8. The architectural tool and its subfunctions.

Interface between the architecture characterization tool and the PAT. The performance assessment tool obtains information from the architecture characterization tool by generating queries. Four query types correspond to the data structure's four functions. The Plus operation on the Multimax has the following format: (AMAX, computation, arithmetic, binary, plus, 32, float). The first parameter specifies the machine as Multimax. The second and the third parameters specify the type of function as arithmetic and binary. The fourth, fifth, and sixth parameters specify the name of the operation, the data size, and the data type, respectively. As a result of this query, a single timing value is returned.

A complete timing information table can also be generated instead of a single value. For example, the query (CM-2, data_movement, proc_proc, net, router, all, 4, all, 1) generates a full table of timing values for data communications using the router network on the CM-2 with Hamming distance 4, and 1- to 64-bit data size. The first parameter in the query specifies CM-2 as the target machine, the second specifies the data movement function, and the third specifies the data movement type as processor to processor. The fourth parameter specifies the network type and the fifth specifies the network name. The next two parameters describe the communication mode as "all" with a Hamming distance of 4 and data sizes of 1, 2, 4, 8, 32, and 64 bits. The last parameter defines the virtual-to-physical processor ratio.

The PAWS data structure can characterize any machine. However, query attributes are only valid if the user initializes the corresponding subdata structure for that particular machine.

Interactive graphical display tool

The interactive graphical display tool provides the user interface for accessing all PAWS tools. The IGDT has been implemented as a hierarchical menu-driven system, allowing multiple windows to be opened in a single session. The main menu shown in Figure 9 allows the user to select the three remaining PAWS tools: the application characterization tool (applications), the architecture characterization tool (architecture), and the performance assessment tool (performance). Users may simultaneously open windows containing information for each of these tools. Figure 9 shows a series of open menus, along with the IF1 graph description of the selected program "matrix1.a." The displayed graph shows nodes organized by levels. All nodes at the same level can execute in parallel. An "optimizations" window lists the user's different optimization choices during compilation.

For large applications, the number of nodes within a graph may be too large to easily display in a single window. IGDT displays graphs hierarchically, allowing users to select any compound node by placing the cursor on the node and selecting expand or collapse from the menu. Expanding a compound node takes the user into the next hierarchy level, showing simple and compound nodes of the selected compound node. Collapse reverses the process of combining nodes within the selected node. Using this approach, users can display as many or as few nodes as required.

Figure 10 shows the menus for interacting with the architecture characterization tool. The user is guided through the differ-

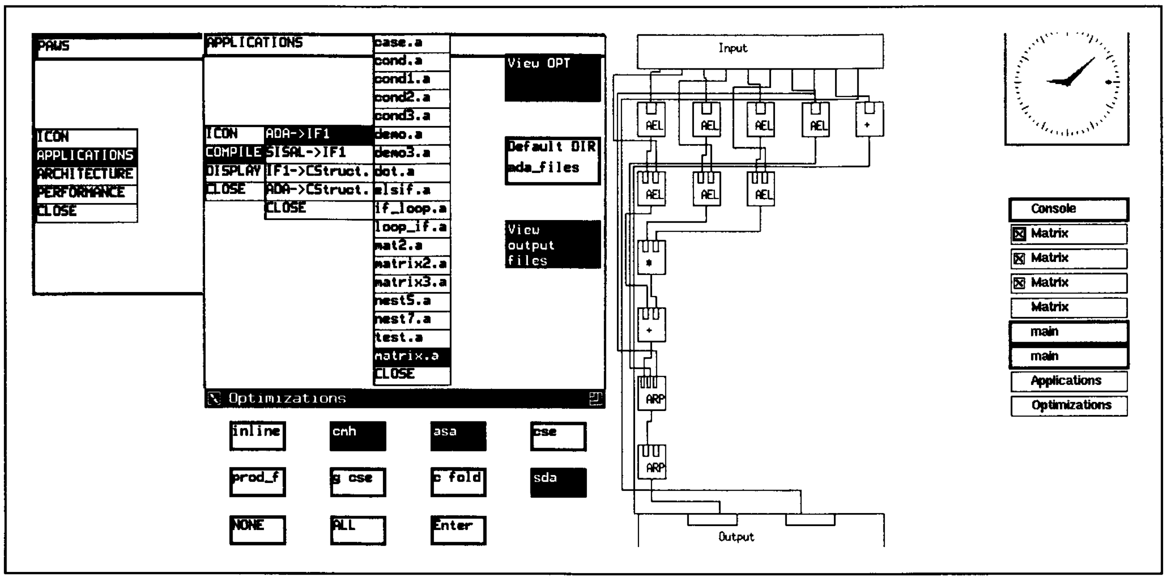
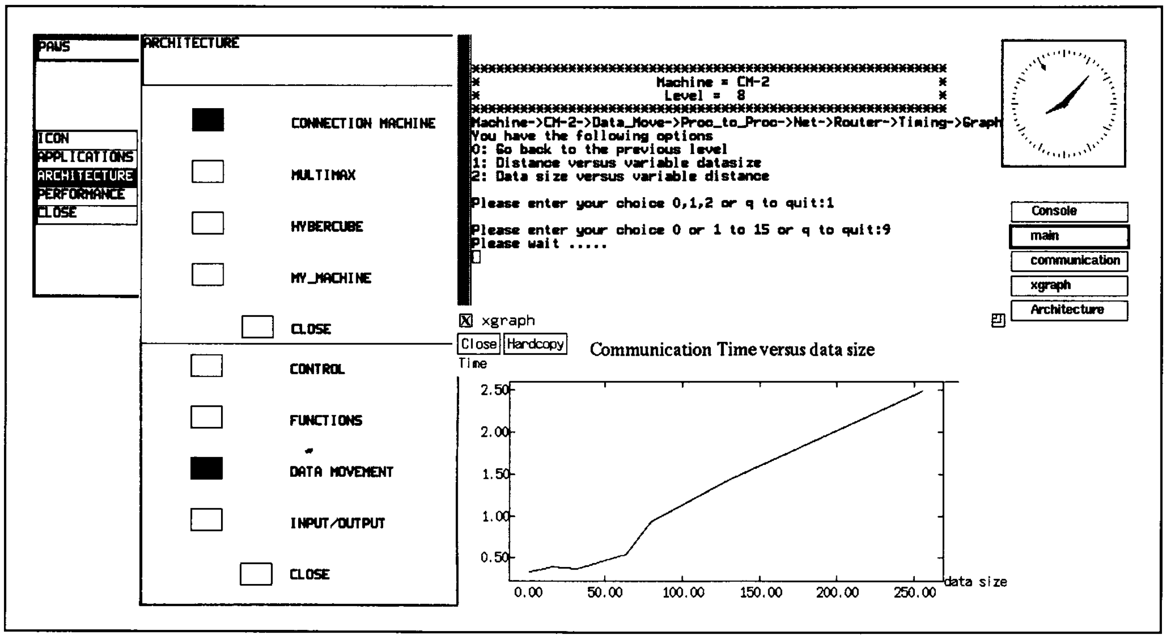


Figure 9. PAWS menus for application characterization.



```
*************************************************************
*                      Machine = CM-2                       *
*                      Level =  8                           *
*************************************************************
Machine->CM-2->Data_Move->Proc_to_Proc->Net->Router->Timing->Graph
You have the following options
0: Go back to the previous level
1: Distance versus variable datasize
2: Data size versus variable distance

Please enter your choice 0,1,2 or q to quit:1

Please enter your choice 0 or 1 to 15 or q to quit:9
Please wait .....
```

Figure 10. PAWS menus for architecture characterization.

ent levels of the parametric data structure by IGDT-generated prompts. This figure also shows the data structure's top level with the prompt for creating dynamic timing information on the CM-2 router communications network. The created timing values are passed to the performance assessment tool and displayed as graphs, as shown.

Performance assessment tool

The PAWS performance assessment tool (PAT) allows users to evaluate the performance of any application entered in the system (using the application characterization tool) by generating a set of performance metrics. These performance metrics include speedup (the average amount of computation performed in one step with unlimited processors) curves, parallelism profile curves, and execution profiles. These performance metrics are generated for both

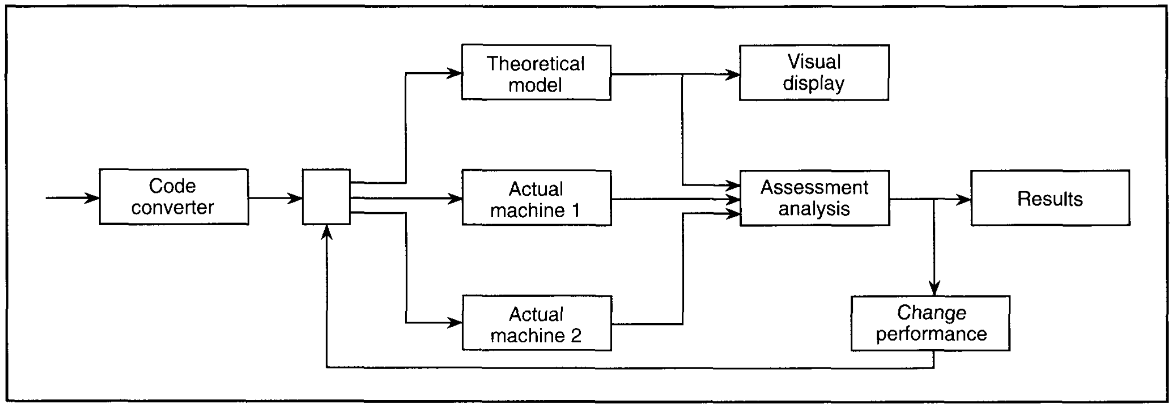


Figure 11. The performance assessment tool block diagram.

the ideal case, which represents the application's theoretical upper bounds of performance, and a set of performance metrics for the application after it has been partitioned and mapped onto a machine. An analysis of the two performance metrics sets shows the effects of mapping the application onto the machine.

Figure 11 shows PAT's overall block diagram. The block labeled "theoretical model" generates an application's ideal parallelism and speedup information. The blocks labeled "actual machine 1" and "actual machine 2" compute the predicted parallelism and speedup performance of the applications running on the specified machines after the application has been mapped onto each machine.

Master of Software Engineering at Carnegie Mellon University

Reply to

MSE Admissions Coordinator
CMU / SEI
Dept. D
Pittsburgh, PA 15213-3890

(412) 268-7713

Reader Service Number 3

Computer Architecture: A Quantitative Approach
John L. Hennessy and David A. Patterson
1990; 784 pages; cloth; ISBN 1-55860-069-8; $59.95
"[This book] is a thorough and extraordinarily wide-ranging education in that magical interface between the programmer's intentions and the electron's actions. It should be read by every software craftsman who cares about wringing the last drop of performance from his machine." *Dr. Dobb's Journal*

Introduction to Parallel Algorithms & Architectures: Arrays, Trees and Hypercubes
F. Thomson Leighton
Spring 1991; approx 500 pgs; cloth; ISBN 1-55860-117-1; $49.95
Features communication networks that form the architectural basis of almost all parallel computing; the author describes their capabilities, limitations and use in solving specific algorithmic problems.

Synthesis of Parallel Algorithms
Edited by John H. Reif
Spring 1991; approx 900 pages; cloth; ISBN 1-55860-135-X; $54.95
Each of the 25 original chapters in this book presents a specific parallel algorithm and contains a description of the fundamental problem, its solution, and analysis with complete examples.

VLSI and Parallel Computation
Edited by Robert Suaya and Graham Birtwistle
1990; 474 pages; cloth; ISBN 0-934613-99-0; $39.95

Cache and Memory Hierarchy Design: A Performance-Directed Approach
Steven A. Przybylski
1990; 223 pages; cloth; ISBN 1-55860-136-8; $39.95

AVAILABLE FROM TECHNICAL BOOKSTORES, OR, CALL TOLL FREE 800-745-7323 (US & CANADA)
Or send a US$ check or money order to **Morgan Kaufmann, 2929 Campus Dr., Ste. 260, Dept. 91, San Mateo, CA 94403.** Include shipping and handling (US/Canada: $3.50 1st volume, $2.50 each additional; Int'l.: $6.50 1st volume, $3.50 each add'l.). CA residents add sales tax. Fax orders: 415-578-0672.

COMPUTER ARCHITECTURE AND PARALLEL ALGORITHMS
MORGAN KAUFMANN PUBLISHERS

Reader Service Number 4

Mapping. The program execution time on any parallel machine is dependent on both the program operations and the users' ability to express the parallelism at the machine's correct granularity level. Therefore, to fairly compare two machines running a common application, two different mappings of the application will be required, one for each machine. In PAWS, applications are first transformed into dataflow graphs and then mapped onto a machine based on its attributes. However, guaranteeing optimal dataflow graph mapping is a nontrivial problem. Currently, PAWS uses the mapping techniques to run IF1 on MIMD machines. Research is underway for PAWS to develop new heuristic algorithms for mapping on both MIMD and SIMD machines.[9]

Generating performance parameters. Both parallelism and execution profiles are generated by performing a graph walk of an application's dataflow graph. The graph walk routine traverses the dataflow graph computing and recording each node's performance and statistical information. To handle compound nodes, the graph walk routine is implemented recursively. This recursive nature allows statistics and timing information to be recorded for individual functions, procedures, etc. Therefore, parallelism profiles and other performance parameters may be generated for a program's function, procedure, etc.

An application's recursive function calls are modeled as linear loops with a predetermined number of iterations. The number of iterations can be input interactively or estimated, based on a frequency count derived from an actual program run.

Examples. Two examples illustrate PAT's flexibility. The Ada source program for the first example, a 5 × 5 matrix multiplication, is shown in Figure 12. Figures 13a through d show the whole program, the three compound nodes (three For loops), and several simple nodes that make up the program. Figures 14a through d show the parallelism profiles for the whole

```
Begin
  for i in 1..5 loop
    for j in 1..5 loop
      for k in 1..5 loop
          c(i,j):=c(i,j)+a(i,k)*b(k,j)
      end loop;
    end loop;
  end loop;
end
```

Figure 12. Matrix multiplication program.

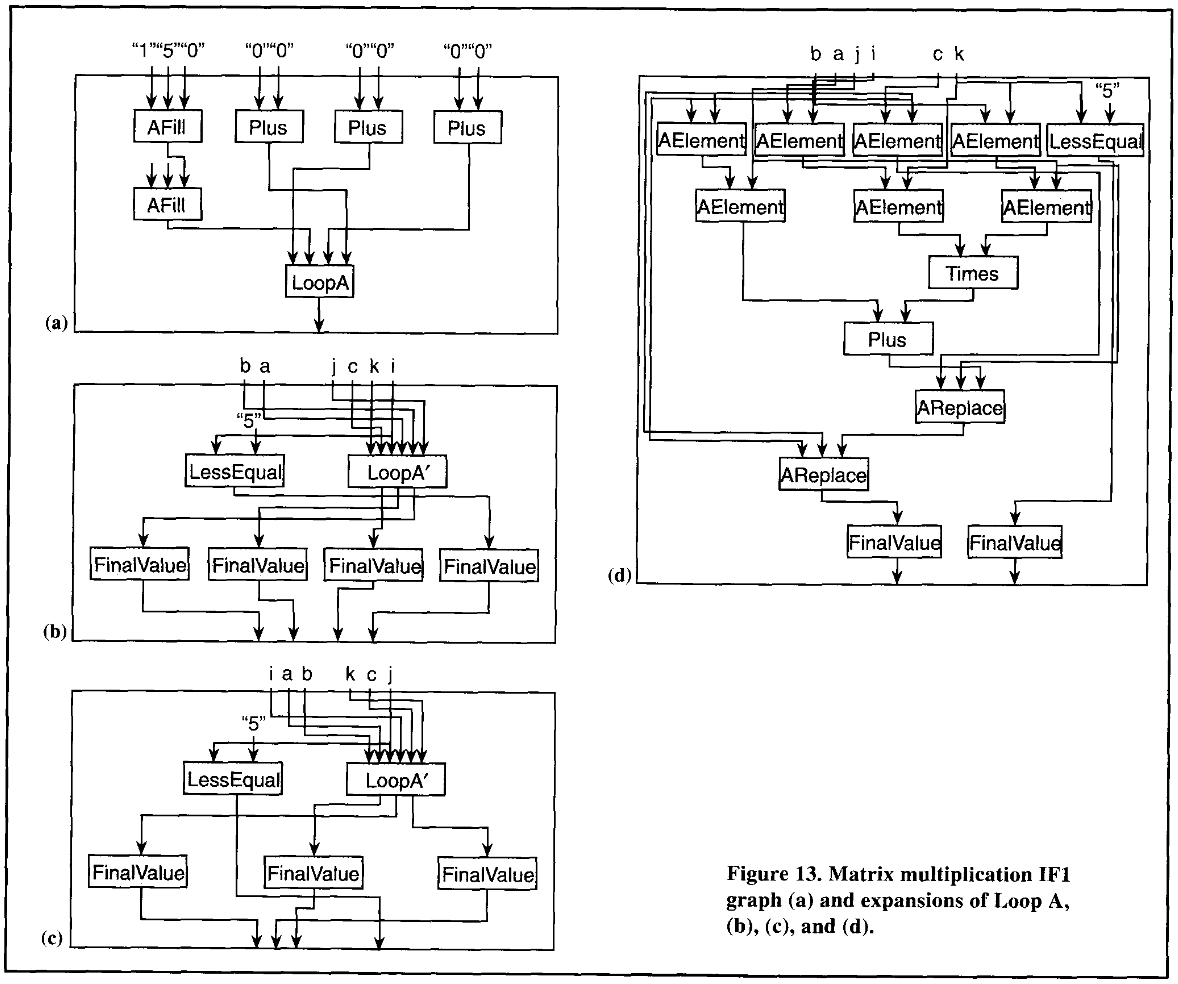


Figure 13. Matrix multiplication IF1 graph (a) and expansions of Loop A, (b), (c), and (d).

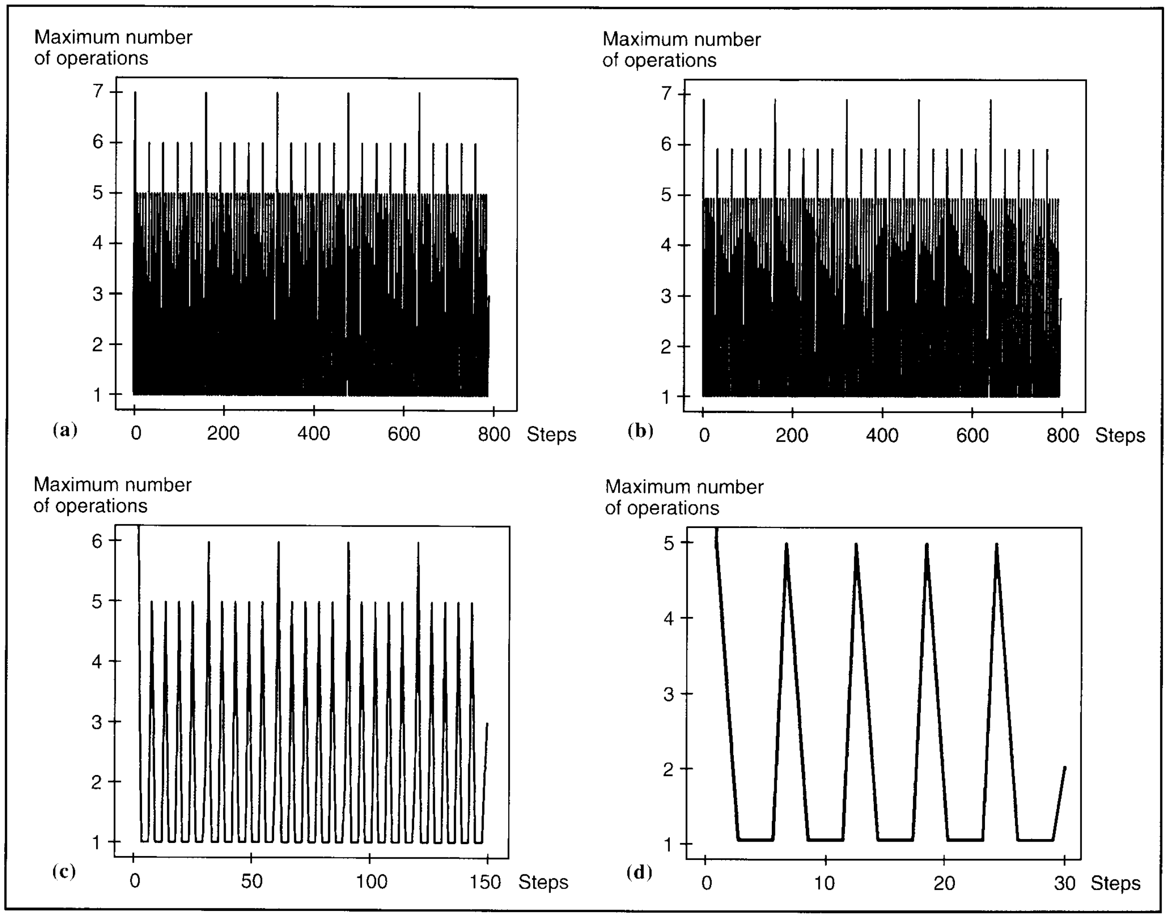


Figure 14. Profile.matrixgraph.5 (a), Profile.LoopA (b), Profile.LoopA′(c), and Profile.LoopA″ (d).

program and the three compound nodes (LoopA, LoopA′, and LoopA″, respectively). The speedup corresponding to the complete program shown in Figure 15 approaches two, the average number of operations performed at each execution step.

Figures 13(a) through (d) exhibit regular fine-grain parallelism patterns throughout the program. By identifying these patterns, a user can synthesize a machine while proceeding with investigations into this part of the code to enhance performance. Furthermore, a user performing algorithm analysis can be prompted to substitute a parallel Forall-type construct for the nested iterative constructs to obtain faster execution times and increased system efficiency. The ability to change programs and machine parameters in the PAWS architectural data structure and quickly observe the modifications' results is a powerful design tool, since modifications to existing hardware can be time consuming and cost prohibitive.

As a second example, we use a program[2] that performs binary integration of the function F as shown in Figure 16. An interesting characteristic of this program is its inclusion of recursive and function calls. This program's parallelism profile and speedup plot are shown in Figures 17 and 18. In Bohm, Gurd, and Kallstrong,[2] a similar plot was generated using a graph interpreter originally designed for IF1 programs translated from SISAL.[10] The figures show that the number of operations available for parallel execution increases geometrically during execution because the

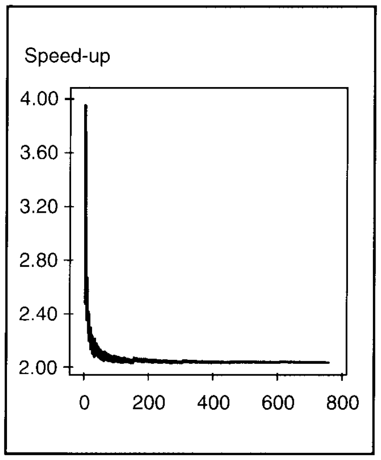


Figure 15. Speedup for matrix multiplication.

number of recursive calls doubles every time a new call is performed in the function area.

The PAWS project's objective is to provide a unified environment for users to assess various existing, conceptual, and prototype machines that execute a common application. PAWS provides a framework for users to compare various architectures for a given application and help to identify the best machine.

PAWS is a unique tool because it combines characterizations of both applications and architectures and generates for assessment various performance metrics, including parallelism profiles and speed-up information. The effects of architectural changes can be investigated through PAWs' ability to modify and store machine descriptions.

Research is underway for PAWS to develop new heuristic algorithms for mapping on both MIMD and SIMD machines.[9] ■

```
Procedure main(A,BInit:in float;r:out float)is
function F(X:float) return float is
Begin
  return 3.0*X*X*X+2.0*X*X+5.0;
end F;
function TRAP(le,Ri:float)return float is
  begin
  return(Ri-Le)*(F(Le)+F(Ri)/2.0;
end TRAP
function AREA(L,R,Est,Tol:float)return float is Mid,A1,A2,News,a:float;
  begin
Mid:=(L=R)/2.0;A1:=TRAP(l,Mid);A2:TRAP(mid,R);News;=A12+A2
  if(abs(Est-News)<Tol)then
    a:=News;
    else
  A:=AREA(L,Mid,A1,Tol/2.0)+AREA(Mid,R,A2,Tol/2.0);
  endif
  return a;
end AREA;
begin
r:AREA(A,B,TRAP(A,B),Init);
end main;
```

Figure 16. Binary integration program.

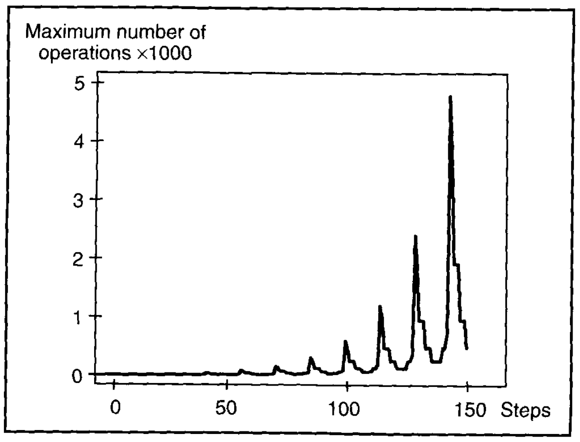


Figure 17. Parallelism profile for binary integration.

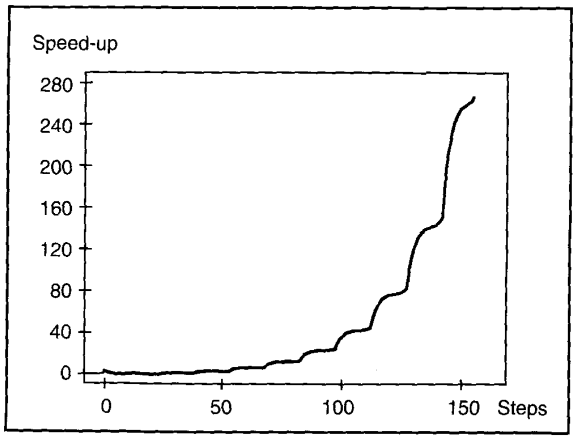


Figure 18. Speedup plot for binary integration.

Acknowledgment

This project was funded through a grant from the Rome Air Development Center under contract F306020-88-C-0136

References

1. Arvind, D.E Culler, G.K Maa, "Assessing the Benefits of Fine-Grain Parallelism in Dataflow Programs," IEEE Computer Society Press, Los Alamitos, Calif. order no. 882, pp. 60-69.

2. A.P.W. Bohm, J.R. Gurd, and M.C. Kallstrom, *Monitoring Experimental Parallel Machines*, Tech. Report, Dept. of Computer Science, Univ. of Manchester, 1988.

3. D.E. Culler, *Effective Data Flow Execution of Scientific Applications*, doctoral dissertation, Computational Science Laboratory, MIT, Cambridge, Mass., 1989.

4. "An Intermediate Form Language IF1," Lawrence Livermore National Laboratory reference manual, 1985.

5. *Multimax Technical Summary*, Tech. Report, Encore Computer Corp., Marlboro, Mass., 1987.

6. C. Levit, "Grid Communication on the Connection Machine: Analysis, Performance, and Improvements," Tech. Report 88.19, Research Inst. for Advanced Computer Science, NASA Ames Research Center, 1988.

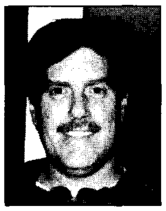
7. D.W. Myers and G.B. Adams II, "Benchmarking and Performance Analysis of the CM-2," Tech. Report 88.19, Research Institute for Advanced Computer Science, NASA Ames Research Center, 1988.

8. R. Pozo and A.E MacDonald, "Performance Characteristics of Scientific Computations

on the Connection Machine," Tech. Report CU-CS-440-89, Center for Applied Parallel Processing, Dept. of Computer Science, Univ. of Colorado at Boulder, 1989.

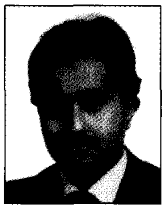
9. V. Sarkar, *Partitioning and Scheduling Parallel Programs for Multiprocessing*, MIT Press, 1989.

10. "DI: An Interactive Debugging Interpreter for Applicative Languages," Lawrence Livermore National Laboratory reference manual, 1987.

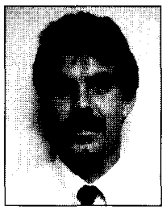


Daniel Pease joined the Syracuse University faculty in 1979 and is currently an associate professor. Pease is involved in a number of research projects related to parallel processing and assessment of parallel systems. The projects are sponsored by DARPA, RADC, IBM, and DEC. He received his BSc in 1968, and MS and PhD degrees in 1973 and 1983, respectively, all from Syracuse University . Pease is a member of IEEE.

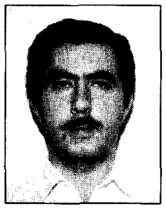
Arif Ghafoor joined the Syracuse University Department of Electrical and Computer Engineering in September 1984 and is currently an associate professor. He is a consultant to such companies as Bell Laboratories and General Electric in the areas of telecommunications and distributed systems. His research interests include design and analysis of parallel and distributed systems and telecommunication. He received his BS degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1976, and his MS, Mphil, and PhD degrees from Columbia University in 1977, 1980, and 1985, respectively. He is a senior member of IEEE and a member of Eta Kappa Nu.



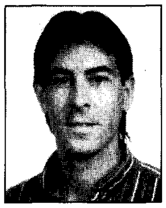
Ishfaq Ahmad has been a research assistant at the Northeast Parallel Architecture Center since January 1989. His research interests include parallel and distributed architectures, scheduling and load balancing, and performance evaluation. Ahmad won the best student paper award in systems at Supercomputing '90. Ahmad is a PhD candidate in the School of Computer and Information Science, Syracuse University. He received his BSc in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1984, and MS in computer engineering from Syracuse University in 1987. He is a student member of the IEEE Computer Society and ACM.



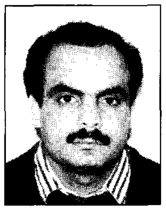
David L. Andrews is employed by the Ocean Systems Division of General Electric where he works on distributed operating systems and distributed networks. His research interests include parallel and distributed architectures. Andrews is pursuing his PhD in computer science at Syracuse University. He received his BSEE in 1983 and MSEE in 1984 from the University of Missouri at Columbia. He is a member of IEEE.



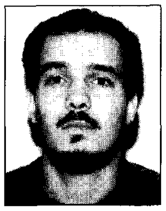
Kamal Foudil-Bey is a research assistant in the Northeast Parallel Architecture Center at Syracuse University. His primary research interests are in parallel computing, performance evaluation, and dataflow representation of algorithms on parallel computers. He is pursuing his PhD degree in performance evaluation and assessment of parallel computers at Syracuse University. He received the degree of Ingenieur D'etat in Electronique from the Ecole Nationale Polytechnique in Algiers in 1984 and the MS degree in computer engineering from Syracuse University. He is a student member of IEEE Computer Society.



Thomas E. Karpinski has been a research assistant in the Department of Electrical and Computer Engineering in the Northeast Parallel Architecture Center at Syracuse University since July 1989. His research interests include parallel and distributed languages, compiler design, and real-time systems. Currently, he is pursuing his MS in computer engineering at Syracuse University. He received his BS in computer science from the Rochester Institute of Technology, Rochester, New York, in 1986.



Mohammad A. Mikki has been a research assistant in the Department of Electrical and Computer Engineering since 1989. He is currently developing tools for displays of IF1 graphs using the X window system. His primary area of research is in parallel processing. He is a candidate for a PhD in computer engineering at Syracuse University. He received his BSc degree in general electrical engineering from Bir Zeit University in West Bank in August 1984 and his MS degree in computer engineering from Syracuse University in 1989.



Mohamed Zerrouki has been a research assistant in the Department of Computer and Electrical Engineering at Syracuse since January 1989. He is currently developing the Ada-to-IF1 converter. His primary research interests are in parallel processing and compiler design. He is a PhD candidate in computer engineering at Syracuse University. He received the Ingenieur d'Etat degree in electronics from Ecole Nationale Polytechnique d'Alger, Algiers, Algeria, in 1985, and the MS degree in computer engineering from Syracuse University in 1988.

Readers may contact Arif Ghafoor at Syracuse University, Department of Electrical and Computer Engineering, Syracuse, NY 13244. Electronic mail can be sent to ghafoor@top.cis.syr.edu.