ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/312578784

Report on Workshop on Design andPerformance Issues in Parallel Architectures

Technical Report · March 1986

DOI: 10.13140/RG.2.2.19057.28004

CITATIONS
0

READS
2

1 author:

Stephen Kaisler
George Washington University
55 PUBLICATIONS **545** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Some Ideas on Machine Learning Research Applied to CyberSecurity View project

Project Parallel Architectures Workshop View project

All content following this page was uploaded by Stephen Kaisler on 20 January 2017.

The user has requested enhancement of the downloaded file.

Report on THE WORKSHOP ON DESIGN & PERFORMANCE ISSUES IN PARALLEL ARCHITECTURES

March 17-18, 1986

Satish K. Tripathi, Steve Kaisler[1], Sharat Chandran[2], Ashok K. Agrawala

Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

[1]Information Sciences and Technology Office, DARPA

[2]Center for Automation Research, University of Maryland

Machines that perform computations in parallel have come into vogue today partly prodded by technology and user needs. In the early spring of '86, a workshop was held under the auspices of the University of Maryland Institute for Advanced Computer Studies (UMIACS) to investigate the design and the not-usually-addressed issue of the performance of these machines. This report serves as a record of the workshop though it does not promise to be a transcript of the various sessions. About a dozen presentations interspersed with sprited open-forum discussions have been paraphrased here. It is hoped that this report remains faithful to the proceedings.

The workshop, as introduced by Prof. Satish K. Tripathi of the University of Maryland, revolved around the following three themes.

1. As computational needs increase, better performance will be gained through the utilization of parallel architectures.
2. Given parallel architectures, there is a substantial need for good software development environments oriented towards parallel programming.
3. The development of architectures to meet applications requirements implies a need for workload analysis.

In holding this workshop, exchange of ideas was made possible between experts in the industry and universities. Simultaneously, students were exposed to leading researchers in parallel processing.

The events have been organized into two parts. In Section 1, we summarize the presentation of invited speakers. In Section 2, group oriented information processing discussions have been condensed. Appendix 1 presents a chronological description of the workshop. Appendix 2 lists the workshop participants and their affiliations.

1 Central Issues in Parallel Processing

1.1 Modeling Systems for Parallel Computations

Professor Jim Browne, of the University of Texas, opened by describing an evolutionary simulation modeling system for modeling parallel computations. The simulation modeling system is based upon a template for a model of parallel computation which covers all of the commonly discussed models, e.g., data flow, functional and applicative model, petri nets, graph models, and linear recurrence relations. The fundamental contention was that our ability to reason is constrained by the language we use. Thus, if our models of parallel computation are inadequate, our understanding will be severely degraded.

Jim proposed that a model should encompass a specification of the elements from which a computation and an abstract machine definition can be composed. The elements form the basic constituents of a parallel computation. The abstract definition is the device upon which one simulates the execution of the model or the specifications for each element in a programmable form. Sequences of abstract machines of increasing levels of resolution are then, simply, computations. To execute such a computation, it becomes necessary to map between abstract machines (see figure 1). Mapping replaces an operation in a source abstract machine with a code template for the target machine which contains parameters. A mapping becomes fully bound whenever the sequence of operations in the target machine is fully determined.

In conclusion, Jim suggested that a unified view of parallel computation would view all computations in terms of dependency graphs. A framework for designing models of parallel computation is shown in figure 2.

1.2 General Purpose Parallel Machines

Professor Arvind, of MIT, focused on the characteristics of a general purpose parallel machine. His goal is to demonstrate the feasibility of general purpose parallel machines in which all processors cooperate to solve the problem provided the problem has "sufficient parallelism".

Arvind believes that the programming model should not reflect the number of processors in the machine nor should it reflect the vagaries of the interconnection topologies among the processors. It should promote higher performance by adding more processors, i.e. performance should be scalable. It should support the dynamic allocation of storage in order to accommodate varying sizes of problems. So Arvind argues that a general purpose parallel computer should insulate the user from changes in the machine configuration resulting from the availability of more resources and the removal of defective components. The latter objective allows more than one type of component processor but insists that the user need not be aware how many of a specific type of these processors there are. Ultimately, if the last of a certain type fails, one must be able to simulate its operations on one of the another types of processors in the system.

How does one insulate the user in this manner? Should one (a) rewrite the application ? (b) recompile the applications code or, perhaps, rewrite the compiler? (c) not rewrite the application code or the compiler but reinitialize the resource managers? These options proceed from the worst to the best case in Arvind's view. Current multiprocessors come in three flavors: (1) 2-8 vector machines in a box. (2) 8-16 processors on a high speed bus. (3) more than 16 processors connected by a multistage switching network. All the above have the average memory latency higher than in the uniprocessor setting. Synchronization in the other key issue in parallel programming typified by (1) forks and joins. (2) producer-consumer problems. (3) mutual exclusion (e.g., serial usage of resources). Thus to recapitulate,

1. As the machine size increases, latency in memory accesses increases and latency in communications with other processors increases.
2. A task running on one of the processors needs to wait for synchronization with other tasks due to the nature of the computation.

Both increase processor idle times.

Arvind points out the problems with multiprocessors based on Von Newman architectures:

1. Minimizing latency cost increases the synchronization cost, and vice versa.
2. The overhead of multiprocessing increases with the size of the machine (i.e. the number of processors).

Arvind suggests the use of functional languages, I-Structures and data flow graphs in his data flow machines to build "real" parallel machines.

1.3 Programming Available Parallel Processors

Ms. Ann Hayes, of the Los Alamos National Laboratory (LANL), presented the parallel processing work at the lab. LANL has considerable parallel processing capability to support its various missions and activities. Among the systems are Cray Research machines, including an X/MP-48, an Intel Hypercube and a Denelcor HEP.

LANL recently performed a study that had the following goals:

1. Find efficient algorithms for specific multiprocessor architectures.
2. Map realistic codes onto multiprocessors.

3. Develop mathematical models for parallel processing computations.

Among the problems studied were Particle-in-Cell (PIC) simulations, hydrodynamics models, Monte Carlo simulations, and chaotic methods(e.g., dynamical systems).

On their 8-processor Denelcor HEP, they found a speedup of 7 using 12 or more processes on the PIC problem.

LANL also considered porting the PIC code to the Hypercube. The PIC code does not vectorize well. Though the computations inherent are essentially parallel. Their conclusions were:

1. It was more difficult to map the PIC code onto the Hypercube than the HEP because:
   (a) Load balancing required more communications.
   (b) More code restructuring was involved for the dynamic memory system.
   (c) Special communication algorithms need to be written to achieve optimal execution.
2. As P, the number of processors, gets large, the PIC code implementation is dominated by the communications costs. That is, the communications time is approximately 1/(log P) whereas the computation time is approximately 1/P.

Ann noted the following issues that one had to consider in moving existing codes (such as PIC) to new parallel processors:

1. You must preserve the large investment in FORTRAN. For many scientists, it is the only language they will ever program in and all of their critical programs are written in some dialect of it (which is not necessarily ANSI standard conforming).
2. How does one evolve from modestly parallel to massively parallel computers. One hopes that the scale up in performance is linear without major (re)programming effort.
3. One needs to evaluate the trade-offs between vectorization and parallelization. Some codes do not vectorize well. Can the same assertion be made for parallelization?
4. How does one deal with the communications problem?

1.4 Graph Reduction and Multiprocessing

Professor Bob Keller, of the University of Utah, wants to to develop multiprocessing architectures which are scalable, have minimal configuration sensitivity (i.e. modular), support general purpose medium grain computation, and are language influenced (by Lisp).

The model he considers is the Graph Reduction Evaluation Model, a functional model based on Lambda-Calculas. It supports medium grain tasking. Tasks are distributable across processors which leads to coherent data management. Module communications is via streams of data.

Bob Keller has developed the REDILISP language for the REDIFLOW system. REDIFLOW stands for the reduction data flow system. It is a model using stream-based programming. The benefit of this model is that nothing gets done (i.e. computed) until somebody (e.g., a processor) demands it.

A graph modeling a particular computation needs to be distributed across the available computational units. The graph dynamically grows and shrinks as pieces of the computation are encountered. Thus, as the system encounters a predicate (function call), it replaces the predicate by its graph which corresponds to the function body. This new piece of graph is spread across the available computational units. Execution is by means of graph reduction, i.e. as something is computed, a corresponding piece of the graph disappears from the computational units because it is no longer needed.

He suggests that the load distribution of the computational effort can be automated. One monitors the load at various processors. When a processor gets heavily burdened, the work should spill over to less burdened or standby processors. His approach is modeled after the notion of pressure - the buildup of a certain amount of work on a specific processor.

Bob postulated the following theorem: Under steady-state conditions, the propagated pressure yields the minimum number of hops to an underutilized node. He noted that the proof is contained in papers describing the REDIFLOW system and the graph reduction model. Saturation occurs when the external pressure equals the upper limit on each processor. Thus, work stays put on the processor and it rejects incoming work because it cannot accommodate it. This approach yields enhanced locality if the system is properly load balanced.

Bob noted the following issues to consider:

1. Current machines scale, but the overhead associated with that scaling is rather large.
2. Is hardware support for lazy evaluation really worth it? All we have so far are simulation/emulation model, but no real hardware.
3. What are the tradeoffs among communications delay, multiplexing level and cost, and the (intra-)switch interface delay.
4. How does one encapsulate destructive operations?

1.5 Architecture Selection using Analytic Modeling

Professor Ken Sevcik, of the University of Toronto, posed the following question: Which architecture will be most effective for my problem? Using this question as motivation, he formulated the following goals: to propose a framework for the selection of architectures and to identify some specific subproblems associated with the selection process. His methodology is depicted in figure 3. The following discussion is keyed to the components of that figure.

1. Alternative Architectures

   There exists a wide diversity of parallel architectures. What we need is one or more rules of thumb for identifying suitable candidate architectures that match characteristics of algorithms that we wish to run.

2. Architecture Characteristics

   Ken proposes that we identify the resources and their quantities, speed, sizes, etc. Characteristics to consider for an architecture are:

   (a) Parallelism among functional units.
   (b) Vector operations.
   (c) Communications paths.
   (d) Memory location.
   (e) Control.
   (f) I/O paths and devices.

What we need is a formal specification of the kind of parallelism supported by the architecture.

3. Structure of Algorithms

   Ken suggests that the data dependency graph is the ultimate representation. To detect parallelism, we can do several things:

   (a) Use very smart compilers.
   (b) Use language extensions (such as PRAGMAs in ADA).
   (c) Use special purpose languages tailored to the problem.

   To represent parallelism, we can:

   (a) Add detail until the desired level of description is reached.
   (b) Use code blocks.
   (c) Use vector sequences.
   (d) Use independent iterations.
   (e) Use process splitting.

   We need to indicate the regularity and locality of communication. We also need to represent parallelism in the algorithm concisely, but in an architecture-independent way.

4. Resource Demands

   A computation structure is composed of the architectural characteristics and the algorithmic structure. The objective is to map the computation onto the processing elements (PE) of the architecture. We must be concerned about the demands on each resource of the system and how communication and synchronization among the PEs are achieved.

5. Performance Prediction

   In the sequential case, we can use the standard queuing network model. For the parallel case, blocking and synchronization considerations may be obtained from the detailed computation graph. We can aggregate pieces to give the time and overall resource usage. Ken suggests that Stochastic Petri Net Analysis may help. From the aggregated computation graph, we can then identify the process splitting and synchronization requirements.

1.6 Architectures for AI

Professor Leonard Uhr, of the University of Wisconsin, concentrated on the development of appropriately process-structured architecture for the perceptual recognition of real world objects in real time. The perception problem is to recognize and describe scenes of complex moving objects. It involves handling enormous amounts of information and must recognize highly structured objects over unknown variations working in real time. The perception program can be represented as a data-flow graph and the problem of developing an appropriate multi-computer topology can be treated as one of finding a multi-computer that can efficiently execute the graph as it scans over it. A 1-node graph, the uniprocessor Von Neumann computer, can scan over any graph but needs far too much time.

A 1-dimensional pipeline (e.g., PICAP, Cyto, deAnza, Vicom, PIPE) can effectively execute local operations on arrays of iterated information of any dimension. A 2-dimension array (e.g., CLIP, DAP, MPP) can handle 2,3 and N-dimensional arrays with *speedups* proportional to the number of processors; but they can be very slow at moving information together for global operations.

Len proposed a pyramid architecture. He suggested viewing pyramids as 2-dimensional pipelines through which information flows and successively gets transformed. Len very graphically described the steps involved in recognizing a mundane object such as a house with the help of slides. On the pyramid a *bottom-up* approach was very naturally presented. Other architectures, like 1-dimensional strings (pipes, rings, arrays), 2-dimensional arrays (DAP, MPP), and shared memory machines (Butterfly, Ultracomputer) were also discussed with their relevant advantages and disadvantages.

2 Short Presentations

Short presentations were instrumental in outlining specific problems. Gordon Lyon of NBS discussed their problems in their ongoing project on the measurement of high performance computers. They would like to compare different "products". The systems architect would like to know where "bottlenecks" lie. The user wants the "best-bang for the buck" and the applications programmer wants the ease of fine tuning his programs.

Dr. Herb Schwetman's presentation showed that MCC's experience with the "no-frills" 12 processor Unix based Sequent machine was good. David Black of CMU presented the programming environment PIE and the operating system MACH. The C-Based Process Oriented Simulation Language (CSIM) seems to be a good package to test quasi-parallel execution of parallel processes.

Professor Pete Sewart, of the University of Maryland, presented the difficulties of programming on a direct connection machine like the ZMOB and suggested the CRAB architecture, which seeks to combine the best of both programming paradigms - the shared memory and the message passing model.

3 Group Discussions

3.1 Architecture (Ashok Agrawala, Moderator)

Many issues came up in the session on architecture. The main problem is how to tailor existing problems, e.g., large scientific programs, large AI problems, large "day to day" programs (airline reservations) to a given architecture. In some sense, architecture may be classified by the application they are best suited to. The issue of virtual memory is also important in the parallel context but no machine described seem to satisfy this requirement. The discussion took a turn when the audience started predicting the future of machines. The Denelcor HEP had an attractive architecture but it did not survive! The consensus seemed to be that general purpose machines will survive if they have good programming environments. Special purpose machines will survive if they offer significant performance improvements over general purpose machines. This brought us to the next two group discussions.

3.2 Software (Herb Schewtman, Moderator)

Processor level parallel programming models are too hard to use. They do not handle massive parallelism and are like multiprogrammed operating systems without good tools.

A good programming language does not say anything about parallelism. Declarative languages are therefore not a good choice. It was also important however to have programs that would take an existing sequential code and "somehow" parallelize it. (Vectorizing compilers do exist...)

The general consensus was that "it's good to have lots of software" that

1. Performs parallel compilation.
2. Performs parallel verification.
3. Promotes host-processor cooperation.

3.3 Performance Issues (Ken Sevcik, Moderator)

The important issues identified were:

1. Speedup and efficiency when using many processors.
2. Will multiprocessing still be a consideration?
3. Intermediate storage.
4. Input/Output from and to the environment.

Evaluation techniques include graph manipulation, timed petri nets and queuing network models.

Another problem is to decide when one needs more processors i.e., capacity planning is difficult. Testing benchmark programs remains popular but we should not fall in the trap of building big machines and testing on toy programs. Obscure bugs have been known to be found both in hardware and software only while tackling problems involving massive parallelism.

It was noted that Fault Tolerance would especially be a consideration in industrial applications.

Miscellaneous exchanges:

- "Parallelism is in the eyes of the beholders"
- Would "graphical programming" *à* la VLSI be more suited than conventional textual programs?
- Can the computation f(a) + g(b) be described as high (or low) level programming?

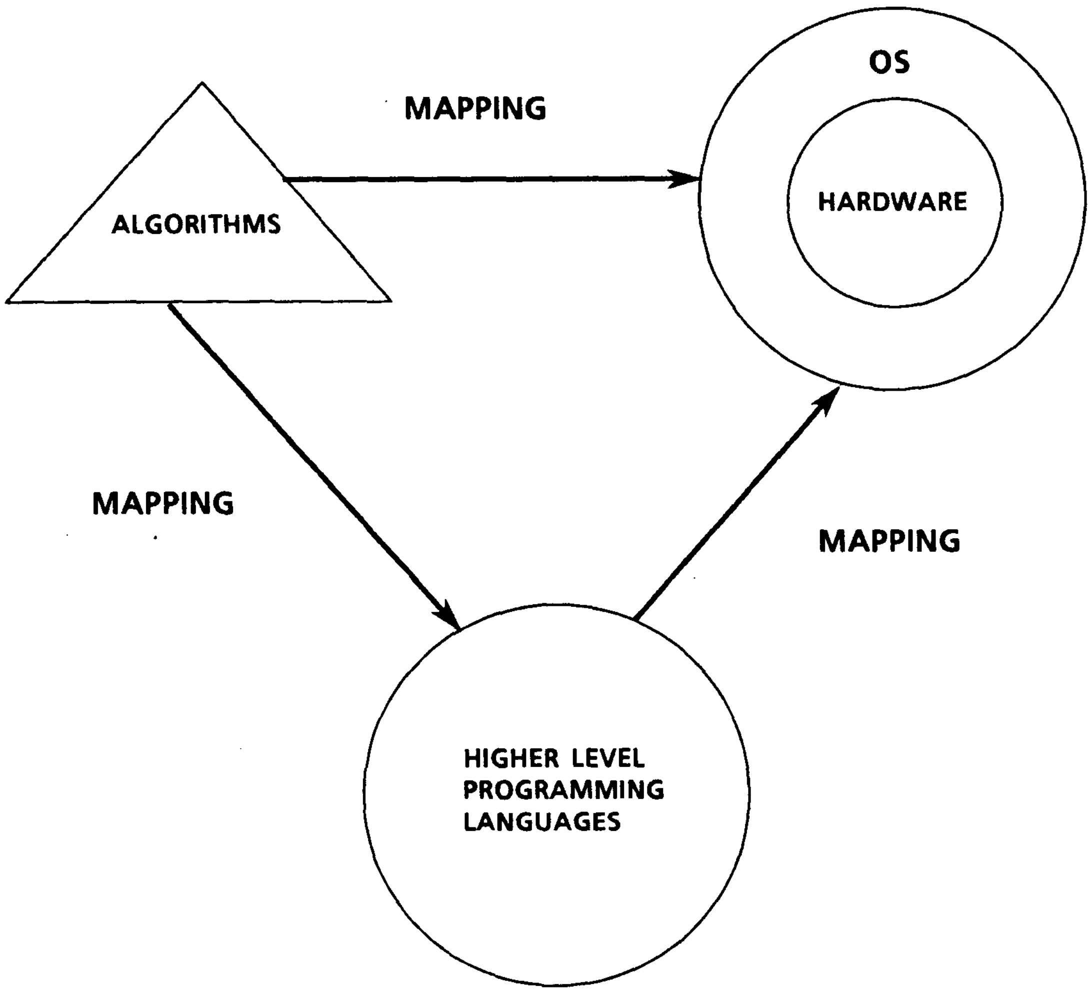
OS
MAPPING
HARDWARE
ALGORITHMS
MAPPING
MAPPING
HIGHER LEVEL PROGRAMMING LANGUAGES

FIGURE 1

DESIGN FRAMEWORK

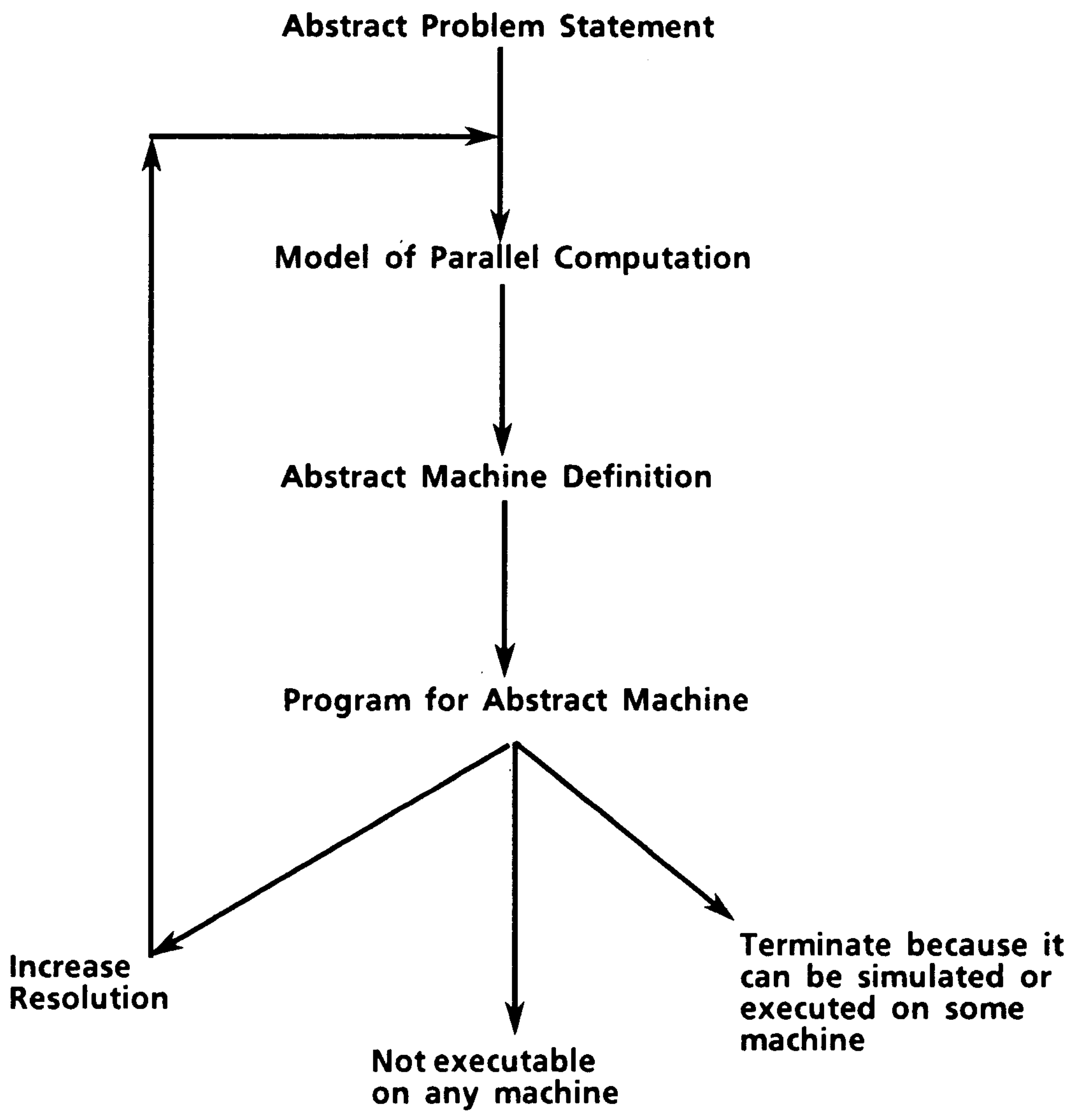
Abstract Problem Statement
Model of Parallel Computation
Abstract Machine Definition
Program for Abstract Machine
Increase Resolution
Not executable on any machine
Terminate because it can be simulated or executed on some machine

FIGURE 2

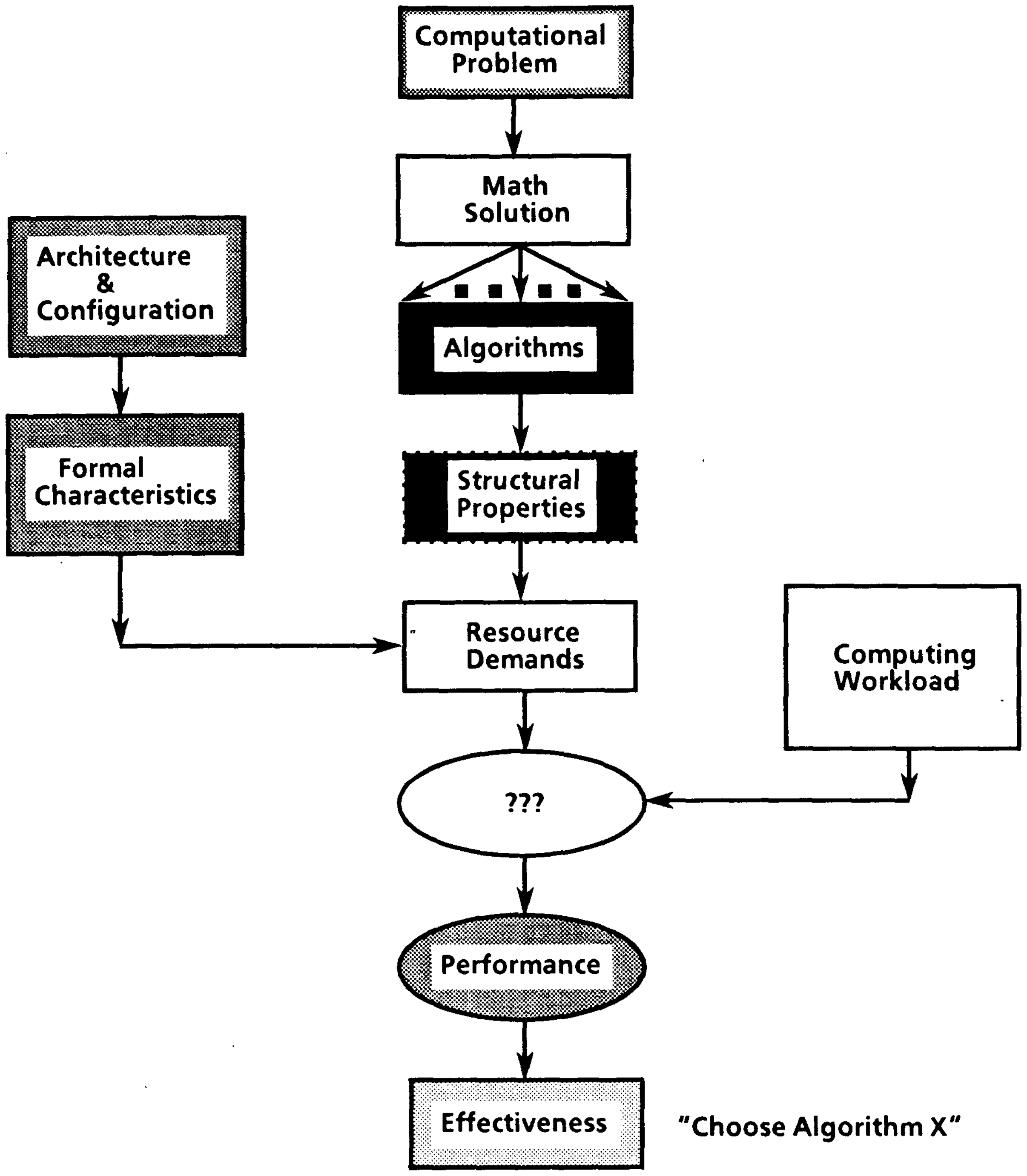
Computational Problem
Math Solution
Architecture & Configuration
Algorithms
Formal Characteristics
Structural Properties
Resource Demands
Computing Workload
???
Performance
Effectiveness
"Choose Algorithm X"

FIGURE 3

Appendix 1

A Schedule: March 17

Welcome: Prof. Kirwan, Vice Chancellor for Academic Affairs, University of Maryland; Prof. Basili, Chairman, Department of Computer Science; Prof. Davis, Acting Director, UMIACS.

Opening Remarks: Prof. Agrawala, Prof. Tripathi, University of Maryland.

Modeling Systems for Parallel Computations: Prof. Browne, University of Texas.

General Purpose Parallel Machines: Prof. Arvind, MIT.

Programming Parallel Processing Computers: Ms. Hayes, Los Alamos Laboratories.

Graph Reduction and Multiprocessing: Prof. Keller, University of Utah.

Architecture Selection using Analytic Modeling: Prof. Sevcik, University of Toronto.

Architectures for AI: Prof. Uhr, University of Wisconsin.

B Schedule: March 18

- Short Presentations.
- Theme Discussions.
- Closing Session.

Appendix 2

C Workshop Participants

Name	Affiliation
Mark Abrams	University of Maryland
George Adams	RIACS
Ashok Agrawala	University of Maryland
Arvind	Massachusetts Institute of Technology
Don Bailey	SAIC
Vic Basili	University of Maryland
David Black	Carnegie Mellon University
Jim Browne	University of Texas
Bob Carpenter	NBS
Sharat Chandran	University of Maryland
Bernie Chern	NSF
Tim Clausner	University of Maryland
Larry Davis	University of Maryland
Ken Dritz	Argonne Labs
Howard Elman	University of Maryland
Martin Ferranti	CDC
Rod Fontecilla	University of Maryland
John Gannon	University of Maryland
Don Harvey	PAR Technologies
Ann Hayes	Los Alamos
William Hopkins	SDC/Burroughs
Pankaj Jalote	University of Maryland
B.N. Jain	University of Maryland, Baltimore County
Steve Kaisler	DARPA
Laveen Kanal	University of Maryland
Robert Keller	University of Utah
Cheeha Kim	University of Maryland
Brit Kirwan	University of Maryland
Clyde Kruskal	University of Maryland
T. Lakshman	University of Maryland

Ron Larsen	University of Maryland
Gordon Lyon	NBS
David Mount	University of Maryland
Diane O'Leary	University of Maryland
Brigitte Plateau	University of Maryland
Maya Ramamurthy	University of Maryland
Kelvin Rawls	PAR Technologies
Dieter Rombach	University of Maryland
Beverly Sanders	University of Maryland
Paul Schnek	SRC/DOD
Herb Schwetman	MCC
Srinivasan Sekar	University of Maryland
Ken Sevcik	University of Toronto
Jagdish Sharma	University of Maryland
Carl Smith	University of Maryland
G.W. Stewart	University of Maryland
David Stotts	University of Maryland
Ashok Thareja	University of Maryland
Satish Tripathi	University of Maryland
Len Uhr	University of Wisconsin
Richard Upton	PSC
Mary Vernon	University of Wisconsin
Harold Weford	Vitro Corp
Nam Woo	Bell Labs
John Zahorjan	University of Washington

● ● ● ●

"All he needs is a good swift kick in the pants."

When inappropriate behavior worsens or persists it can be a warning sign of a mental illness.

Learn to recognize these important warning signs. It can be the first step to healing the sickness.

- Marked personality change over time.
- Confused thinking; strange or grandiose ideas.
- Prolonged severe depression; apathy; or extreme highs and lows.
- Excessive anxieties, fears or suspiciousness; blaming others.
- Withdrawal from society, friendlessness; abnormal self-centeredness.
- Denial of obvious problems; strong resistance to help.
- Thinking or talking about suicide.
- Numerous, unexplained physical ailments; marked changes in eating or sleeping patterns.
- Anger or hostility out of proportion to the situation.
- Delusions, hallucinations, hearing voices.
- Abuse of alcohol or drugs.
- Growing inability to cope with problems and daily activities such as school, job, or personal needs.

For an informative booklet, write: The American Mental Health Fund, P.O. Box 17700, Washington, D.C. 20041. Call toll free: 1-800-433-5959. In Illinois, call: 1-800-826-2336.

Learn to see the sickness. Learning is the key to healing.

THE AMERICAN MENTAL HEALTH FUND

Ad Council