NASA Contractor Report 172197

ICASE

NASA-CR-172197
19830026342

THE FEM-2 DESIGN METHOD

Terrence W. Pratt
Loyce M. Adams
Piyush Mehrotra
John Van Rosendale
Robert G. Voigt
Merrell Patrick

Contract Nos. NAS1-17070, NAS1-17130
August 1983

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

THE FEM-2 DESIGN METHOD

Terrence W. Pratt
Department of Applied Mathematics and Computer Science
University of Virginia
Charlottesville, VA 22901

Loyce M. Adams
Piyush Mehrotra
John Van Rosendale
Robert G. Voigt
Institute for Computer Applications in Science and Engineering

Merrell Patrick
Department of Computer Science
Duke University
Durham, NC 27706

Abstract

The FEM-2 parallel computer is being designed using methods differing from those ordinarily employed in parallel computer design. The major distinguishing aspects are: (1) a top-down rather than bottom-up design process, (2) the design considers the entire system structure in terms of layers of virtual machines, and (3) each layer of virtual machine is defined formally during the design process. The result is a complete hardware/software system design. The basic design method is discussed and the advantages of the method are considered. A status report on the FEM-2 design is included.

This work supported in part by the National Aeronautics and Space Administration under NASA Contracts NAS1-17130 and NAS1-17070 while the authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665. The first author was also supported in part by NSF Grant MCS78-00763.

N83-34613#

Introduction

The Finite Element Machine [1,2] is an array of microprocessors, originally designed as a special purpose parallel computer for solution of problems in structural analysis using finite element methods. The authors are currently in the process of designing a successor, FEM-2, aimed at essentially the same applications.

Parallel Machine Design

In most parallel machine design, the basic hardware decisions are fixed at an early stage of the design, long before the software organization and external environment have been considered in detail. This approach often leads to major problems at later stages, where the software and external supporting environment must be distorted to match the already fixed hardware organization. The general approach of early decision on hardware, followed by later detailed software design is seen in the original FEM [1,2], and in most other designs reported in the literature, e.g., Blue CHiP [3], TRAC [4], MPP [5] to name a few. This design approach is basically a "bottom-up" approach.

In the FEM-2 design, an alternative "top-down" approach has been adopted. While the use of a top-down approach to system design is not novel, the particular form this has taken in the FEM-2 design is novel, in the context of parallel computer design. Two aspects are of note:

a. FEM-2 is considered to be composed of layers of virtual machine. Each layer defines the view of the system available to one class of users. Four layers of virtual machine are currently conceived: (1) The applications user's machine (e.g., as defined by the interactive command language), (2) the applications programmer/numerical analyst's machine (e.g., as defined by the applications language), (3) the systems programmer's machine (e.g., as defined by the operating system structure), and (4) the hardware itself (which if microprogrammed may include another layer of virtual machine).

b. Each layer of virtual machine is formally specified during the design process, using the methods of H-graph semantics [6] to construct a formal model of each layer. The advantages of this formal specification are explained below.

A virtual machine is composed of (1) various types of data objects, (2) various operations on those data objects, (3) various sequence control mechanisms for specifying the order of the operations, (4) various data control mechanisms for controlling access to data objects by the operations, and (5) storage management mechanisms for determining the placement and movement of data and code during program execution.

The FEM-2 Virtual Machines

Although complete virtual machine descriptions cannot be given here, a brief sketch will indicate the general type of results from this design approach. Considering each of the four levels of virtual machine, some typical data objects, operations, control mechanisms, and storage management methods are listed below.

Application User's Virtual Machine

The FEM-2 user would typically be a structural engineer using the system as an interactive workstation that allows one to store the description of a structural model, to invoke applications packages to analyze the model, and to display the results. The following is a partial list of the virtual machine components at this level.

Data objects:

Structure/substructure model

Grid description

Node/element description

Load set

Displacements of nodes

Stresses on elements

Operations:

Define structure model

Generate grid

Define elements

Solve structure model/load set for displacements

Calculate stresses

Data base operations (store model in DB/retrieve)

Sequence control:

Direct interpretation of user commands

Data control:

Workspace (user local data)

Data base (long-term storage; shared data)

Storage management:

Dynamic storage allocation for models, results, workspaces, etc.

Data movement between data base and workspace

Numerical Analyst's Virtual Machine

The numerical analyst is a research user who views the machine in terms of a high-level language that allows him to specify directly the data structures, operations and their sequences, and the parallelism in the linear algebra necessary to implement efficiently a structural engineer's application. We assume as a base a sequential language such as Fortran, Pascal, or Ada, and only mention some of the new constructs needed for effective control of the parallel processing and data distribution in the parallel system.

Data objects:

Windows on arrays (e.g., row, column, block descriptors, for remote access to non-local data) see [6] for more details

Operations:

Tasks (programmer-defined parallel procedures)

Window operations: create window, access/assign data visible in a window

Broadcast data to a set of tasks

Linear algebra operations: inner product, vector operations, etc.

Sequence control:

Forall loops -- do all iterations in parallel if possible

Pardo...end -- do all statements in parallel

Task control: initiate a task, pause, resume a paused task, terminate

Remote procedure call - location determined by location of data visible in a window

Data control:

All data owned by a single task

Data accessible non-locally only via windows

Windows may be transmitted as parameters, further partitioned, stored as values of variables, etc.

Tasks may communicate through windows

Storage management:

Dynamic creation of data objects by a task

Data lifetime = lifetime of owner task

Dynamic creation of multiple task replications

Local data of a task retained over pause/resume

System Programmer's Virtual Machine

By specifying the run-time representation of tasks, their scheduling, the communication between them, and the storage representation of the data, the system programmer's virtual machine is used to implement the numerical analyst's virtual machine. The following is a partial list of the virtual machine components.

Data objects:

Code blocks/constants blocks

Task/procedure activation records (local data)

Window descriptors

Storage representations for scalars, arrays, etc.

Messages from tasks:

initiate K replications of a task of type T

pause and notify parent task

resume a child task

terminate and notify parent

remote procedure call

remote procedure return

load code/constants

Operations:

Usual sequential operations: arithmetic, procedure call, etc.

Library routines for linear algebra operations

Format and send message (one of the 7 types above)

Decode and execute message (e.g., an initiate task message may require the following steps: find code for task, allocate an activation record, copy parameters from the message queue into activation record, enter task in ready queue)

Sequence control:

Usual sequential language control structures

Data control:

Usual sequential language structures

Storage management:

General heap with variable size blocks

Hardware architecture

The requirements imposed by the upper levels of virtual machine suggest that the architecture should be chosen to effectively support:

Large scale dynamic task initiation

Remote access to local data (through windows)

Large messages (between tasks, and from a task to the operating system)

Irregular communication patterns

Large storage requirements; dynamic allocation

Fast linear algebra operations (to extract the low-level parallelism available in these operations)

In addition, several additional requirements are imposed independently:

Use off-the-shelf hardware/software if possible

Provide a way to extend the system to larger configurations easily

Provide reconfigurability to isolate faulty hardware components

Provide multi-user access

From these requirements, an architecture is evolving that is configured as clusters of processing elements organized around a shared memory. Sets of clusters communicate through a common communication network. Within each cluster, one PE runs the operating system kernel, which fields incoming messages and assigns available PE´s to process them. Messages arriving in the input queue of any cluster can be processed by any available PE. Since this architecture will be described at length in other papers, no detailed design is given here.

Formal Specification of Virtual Machines

By formally specifying the data objects, operations on those data objects, control mechanisms, and storage management techniques of each virtual machine level, a detailed software/hardware design can be obtained that specifies the function of each level as well as its implementation on the next lower lever. Our research uses the methods of H-graph semantics [7] for making this formal specification. H-graph semantics is a mathematical modeling method for software/hardware systems that can be used to construct a precise mathematical model of each virtual machine level. The data objects are modeled as hierarchies of directed graphs (H-graphs) in which the nodes represent abstract storage locations and the arcs represent access paths. Data types are modeled using formal "H-graph grammars," a type of BNF grammar in which the "language" defined is a set of H-graphs representing a class of data objects. Operations (procedures) on the data objects are modeled as "H-graph transforms," which are functions defining transformations on the H-graph models of data objects. H-graph transforms may invoke each other in the usual manner of subprogram calling hierarchies to determine the overall flow of control in a model of a virtual machine.

In the FEM-2 design process, each layer of virtual machine is designed first, starting with the top layer and considering each layer as defining the requirements that must be satisfied by the design at the level below. Several iterations through the four levels are made, adjusting the design to find an appropriate mix of hardware and software at each level. As the design begins to "firm up", the individual virtual machines are defined formally. The precise formal definitions are then used as the basis for simulations of the various virtual machine levels. Simulations to measure the storage, processing, and communication patterns in typical FEM-2 applications and to determine the ease of programming the machine at the various levels are of particular importance. The ultimate result is to be a detailed design of the hardware and software, completely specified at each level in terms of its function and its implementation on the next lower level of virtual machine.

Conclusion

A major advantage of the top-down, layers of virtual machine, design approach is that it forces a design of the entire system structure, including I/O (virtual) devices, global control strategies, interfaces with the outside environment, etc. at an early design stage. It also allows the potential parallelism at various levels to be considered in detail: parallelism in user requests for simultaneous solution of several independent problems, parallelism in the substructure analysis of a larger structure, parallelism in the finer structure of solution of a particular system of simultaneous equations, etc. A third advantage is that the entire design process may be iterated, adjusting the design of each virtual machine level, until the proper match of hardware and software organizations is found.

Current Status

The FEM-2 design effort has been underway since December 1982. At present the first iteration of the design of the four layers of virtual machine is nearing completion. Several scenarios of use of the numerical analyst's virtual machine have been carried out in detail, using a detailed design of a typical algorithm to get quantitative estimates of processing requirements, storage requirements, and communication requirements for a typical large-scale application. One such analysis is reported in [8]. H-graph semantics definitions of the various levels are being constructed.

References

[1] H. Jordan, "A Special Purpose Architecture for Finite Element Analysis," Proc. 1978 IEEE Conf. on Parallel Proc.

[2] O. Storaasli, et al. "The Finite Element Machine: An Experiment in Parallel Processing," Research in Struct. & Solid Mechanics, NASA Conf. Pub. 2245, Wash. D.C., 201-217, October 1982.

[3] L. Snyder, "Introduction to the Configurable, Highly Parallel Computer," IEEE Computer, Jan. 1982.

[4] M. Sejnowski, at al. "An Overview of the Texas Reconfigurable Array Computer", AFIPS Proc. 1980 NCC, 631-641.

[5] K. Batcher, "Design of a Massively Parallel Processor," IEEE Trans. on Comps., Sept. 1980, 836-840.

[6] P. Mehrotra, "Distributed Processing of Large Arrays," Ph.D. Thesis, University of Virginia, 1982.

[7] T. Pratt, "Formal Specification of Software Using H-graph Semantics," Rept. 83-2, Appl. Math & Comp. Sci., U. of Va., Jan. 1983.

[8] L. Adams and R. Voigt, "A Methodology for Exploiting Parallelism in the Finite Element Process," Proc. NATO Advanced Research Workshop on High Speed Computation, Julich, West Germany, June 1983, Springer-Verlag.

1. Report No. NASA CR-172197	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle The FEM-2 Design Method		5. Report Date August 1983 6. Performing Organization Code
7. Author(s) Terrence W. Pratt, Loyce M. Adams, Piyush Mehrotra, John Van Rosendale, Robert G. Voigt, Merrell Patrick		8. Performing Organization Report No. 83-41 10. Work Unit No.
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665		11. Contract or Grant No. NAS1-17070, NAS1-17130 13. Type of Report and Period Covered Contractor report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		14. Sponsoring Agency Code

15. Supplementary Notes Additional support: NSF Grant MCS78-00763.
Langley Technical Monitor: Robert H. Tolson
Final Report

16. Abstract

The FEM-2 parallel computer is being designed using methods differing from those ordinarily employed in parallel computer design. The major distinguishing aspects are: (1) a top-down rather than bottom-up design process, (2) the design considers the entire system structure in terms of layers of virtual machines, and (3) each layer of virtual machine is defined formally during the design process. The result is a complete hardware/software system design. The basic design method is discussed and the advantages of the method are considered. A status report on the FEM-2 design in included.

17. Key Words (Suggested by Author(s)) parallel computing virtual machine programming languages	18. Distribution Statement 61 Computer Programming and Software 62 Computer Systems Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	12	A02

End of Document