SYSTEMS ARCHITECTURE

Proceedings of the Sixth
ACM European Regional Conference

THE INTERNATIONAL COMPUTING SYMPOSIUM

SYSTEMS ARCHITECTURE

Proceedings of the sixth ACM European regional conference

Published by Westbury House, the books division of IPC Science and Technology Press Limited, PO Box 63, Bury Street, Guildford, Surrey GU2 5BH, England

ISBN 0 86103 050 8

Printed in Great Britain

Foreword

The 1981 International Computing Symposium is the sixth in a biennial series organized by the European Chapters of the Association for Computing Machinery (ACM). Although the theme chosen for the conference is SYSTEMS ARCHITECTURE, the programme reflects a wide range of topics and contributions that are of current interest and concern to computing professionals. The primary goal of the symposium is to report on the state of the art and to foster the exchange of ideas among scientists, computer professionals, engineers and managers on problems, new techniques and trends of SYSTEMS ARCHITECTURE. At this conference topics under discussion will be the significant developments in computing ranging from theoretical to very practical aspects.

The distinguished session chairmen, invited speakers and authors, 71 in total, are well known in their specialist field and are of international repute. The 51 papers in this book were selected from over 160 submissions to give a balanced and authoritative view on the subject. The proceedings are divided into 13 sessions and the second day of the conference has been extended to include a panel discussion on "Formal Specification for Practical Use" and a special-interest tutorial given by Tom Gilb, "Design by Objectives".

I. S. Torsun
Programme Committee Chairman

Programme Committee

I.S. Torsun
Brunel University, UK (Chairman)
H. Savary
IRIA, France
M. F. Nicolet
Phillips AG, Switzerland
C. Jenny
IBM, Zurich, Switzerland
P. Schnupp
InterFace, Munich, Germany
T. Härder
Universität Kaiserslautern, Germany
J. Kent
Tandberg Data, Oslo, Norway
H. Hünke
GMD, Germany
M. Jackson
ITT, UK
G. Valle
Università di Bologna, Italy
B. Parslow
Brunel University, UK

Conference held in cooperation with
Computer Communications and
Microprocessors and Microsystems

Contents

*Invited paper

*Invited paper

*Invited paper

A Distributed and open architecture

Session Chairman: Dr F. K. Hanna
Electronics Laboratories, University of Kent, Canterbury, UK

MULTIPROCESSOR DESIGN AND MATHEMATICAL STRUCTURES

W. Forster

Faculty of Mathematical Studies, The University, Southampton, England

This paper first investigates abstract mathematical structures and the possibility of parallel execution of operations. We then discuss corresponding multiprocessor structures. As a concrete example the numerical solution of nonlinear equations will be considered. We utilize recently developed globally convergent algorithms for the solution of systems of highly nonlinear equations. Some aspects of these new algorithms in a multiprocessor environment will be studied and possible future improvements outlined. In the last section we examine the possibility of off-the-shelf implementation of a dedicated multiprocessor (a nonlinear equation solver) using available VLSI components.

INTRODUCTION

The requirement to achieve faster throughput of data has led to the implementation of a number of multiprocessors with various design philosophies [1]. One such design, the CRAY-1, is very efficient in handling loops. Another design, the Floating Point AP-120B, has as main feature the efficient execution of floating point arithmetic. A third design, the ICL DAP, can efficiently handle elliptic partial differential equations. Other designs, e.g. data driven multiprocessors utilizing Petri-nets, are in the prototype stage [2], [3]. The mathematical features (e.g. floating point arithmetic, etc.) around which existing multiprocessors have been designed are obviously important but seem to be a rather arbitrary collection when compared to the vast number of mathematical structures which are amenable to parallel computation. None of the existing above mentioned multiprocessors utilizes recently developed technology. The availability of comparatively low priced microprocessors, support devices and memory components allows the design of multiprocessors with a much larger number of processing elements [4]. In this paper we will first investigate some mathematical structures and the possibility of matching multiprocessor structures to these mathematical structures. We then outline briefly how these ideas can be used to design a multiprocessor specifically dedicated to the solution of systems of nonlinear equations. In the view of the author, the proposed multiprocessor structure represents the fastest known nonlinear equation solver.

MATHEMATICAL STRUCTURES AND MULTIPROCESSOR STRUCTURES

Let us consider the algebraic structure of a group. Assuming the group operation is addition, then from a mathematical point of view it is unimportant whether the elements are real numbers, vectors, or matrices. In a computational setting this independence on the elements is e.g. implemented in APL (In APL a single function may be able to cope with a scalar, vector or matrix. Variable types do not have to be specifically declared. They are inferred from their context). From a mathematical point of view the nature of the elements (real numbers, vectors,

matrices) is unimportant, but for computational purposes the precise nature of the operations to be performed requires detailed attention (e.g. memory requirements have to be considered). In a multiprocessor environment a suitable compiler (in an extended sense) could make the required number of processing elements (PE's) available (e.g. one PE if we add real numbers, n PE's if we add vectors, n^2 PE's if we add $n \times n$ matrices).
At the next level of abstraction we consider e.g. groups and structure preserving mappings between groups. At this level of abstraction we talk about categories (e.g. the category of groups, the objects are groups and the mappings are group homomorphisms). The next level of abstraction considers structure preserving maps between categories, and we talk about functors. For literature see e.g. [5], [6]. In a simplified form these concepts will be used later.
Let us consider an example of a category. We could e.g. have sets and continuous functions defined on these sets, i.e. we map sets into sets by continuous functions. This is an example of a category (objects: sets, mappings: continuous functions). If we have a large number of PE's, then we can associate each PE with an element from a set. Each PE can perform a given function evaluation. The function evaluations obtained by all PE's form another set. We have therefore succeeded to represent the category "set" in some form as a multiprocessor structure. Of course there are limitations. One has to keep in mind that mathematical structures may have an infinite number of elements, computer structures are always finite in some form.
A suitable programming language for such a multiprocessor would not only allow to define data types (e.g. sets), but would allow also the definition of operations on these data types. One would then be able to execute user defined algebras on such a machine. In PASCAL these ideas are partially realized, i.e. PASCAL allows the definition of data types (by using the TYPE declaration), but it does not allow the definition of operators on these data types.

A NONLINEAR EQUATION SOLVER

In this section we will outline the design of a dedicated multiprocessor, a nonlinear equation solver. The complexity necessary for general purpose multiprocessor structures seems to make dedicated applications more likely [7]. The solution of systems of nonlinear equations on computers is one of the more complicated problems. Until recently no globally convergent algorithm was available. In 1967 the first satisfactory algorithm for the solution of a system of nonlinear equations in n unknowns was published [8]. Since then many modifications have appeared. These algorithms use what is known as topological fixed point theorems. For details the reader is referred to the literature [9].
In this section we will briefly outline how these algorithms can be matched to suitable multiprocessor structures. The main criterion is speed (combined with low cost), i.e. we consider a user who wants to find one or more than one solution of a system of nonlinear equations for some kind of real time application (e.g. a control problem, etc.) and the user wants to find these solutions as fast as possible (using low cost microprocessor technology). The nonlinear equation solver we describe will require the following time to solve a system of n nonlinear equations:
(i) the time required for the evaluation of one value of the given nonlinear function,
(ii) plus another time interval dependend on the dimension n of the problem.
Applications in which systems of nonlinear equations with varying data are solved at regular time intervals are already known. Low cost microprocessor technology will make such computations available for a larger class of applications.

(a) A nonlinear equation solver for a one-dimensional problem

We explain the main idea on a one-dimensional example. We want to find a zero of a nonlinear function defined e.g. on the interval $0 \leq x \leq 1$, i.e. we want to find

a value x' such that $g(x') = 0$. We assume we have $N+1$ PE's, each with a reasonably sized memory.

(i) We associate each PE with a point x in [0, 1], e.g. we subdivide [0, 1] into N equal intervals.

(ii) We compute the function value $g(x_j)$ on the processing element PE_j associated with x_j .

(iii) We compare the sign of the computed function value $g(x_j)$ in PE_j with the sign of the function value $g(x_{j+1})$ computed in PE_{j+1} (i.e. the adjacent PE). If the sign of $g(x_j)$ is different from the sign of $g(x_{j+1})$, then we have found a solution.

(iv) The PE where an affirmative answer to the question of change of sign has been obtained activates a device (a bus, etc.) and sends the coordinate x_j to some output device.

This very fast method can find a zero of a one-dimensional problem in approximately the time it takes to compute one function value. If higher accuracy is required, then the same method can be employed again. This time by subdividing the interval between x_j and x_{j+1} into N parts and proceeding as before. In this manner, by repeating the above described procedure, after a couple of repeated applications a very high accuracy can be achieved.

(b) A nonlinear equation solver for an n-dimensional problem

For an n-dimensional problem, i.e. n nonlinear equations in n unknowns, the problem is a more complicated one. An introduction to these algorithms, called fixed point algorithms or pivoting algorithms, can be found in [9]. For the most recent literature see [10]. For a paper on array processors and fixed point algorithms see [11]. Instead of finding a zero $g(x) = 0$ we solve the equivalent problem $f(x) = x$ with $f(x) \equiv g(x) - x$, i.e. we compute fixed points. We will give a brief outline for two-dimensional problems. The features of two-dimensional problems can easily be generalized to n-dimensional problems. In two dimensions we triangulate e.g. a region called a simplex into smaller simplices (see Fig. 1). In n dimensions we would subdivide an n-dimensional simplex into smaller n-dimensional simplices. For the n-dimensional case we assume we have $n+1$ PE's , each with memory, for each vertex of the triangulation. The nonlinear function is assumed to be a continuous function

$$f: S^n \to S^n$$

from the n-dimensional simplex to itself. At each grid point (vertex) we compute the function value. This can be done concurrently. We attach an integer label i according to

$$i = \min_j \{j \mid f_j(x_0 , x_1 , \ldots , x_n) \leq x_j > 0\}$$

to each grid point $(x_0 , x_1 , \ldots , x_n)$.

For the one-dimensional problem we had the condition that the sign of the function values at two adjacent points has to be different. The analogous condition for the n-dimensional problem is that the vertices of a simplex carry the labels $\{0, 1, \ldots , n\}$. This can be checked in approximately $\log_2(n)$ steps.

If we want to obtain higher accuracy, we can apply a method similar to the one mentioned for one-dimensional problems. Instead of integer labels we have to use something called vector labels. The reason for this is the fact that for integer labels the solution of the problem does not have to lie inside the simplex regarded as approximate solution (for $n > 1$) , see e.g. [9]. For vector labels the solution lies inside the simplex regarded as approximate solution. A vector label is a vector

$$\ell_j = f_j(x_0 , x_1 , \ldots , x_n) - x_j$$

and we compute these vectors for each grid point.

If the zero vector is a convex combination of the vector labels attached to the vertices of a simplex, then we have a solution, i.e. if

$$\sum_{j=0}^{n} \lambda_j \ell_j = 0$$

where $\sum_{j=0}^{n} \lambda_j = 1$ and $\lambda_j \geq 0$,

then a fixed point lies inside the simplex. In other words we have to solve a system of linear equations for each simplex. The systems we have to solve are of the form

$$\begin{bmatrix} 1 & \cdots & 1 \\ \ell(y^0) & \cdots & \ell(y^n) \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \cdot \\ \lambda_n \end{bmatrix} = \begin{bmatrix} 1 \\ \cdot \\ 0 \end{bmatrix} ,$$

where $\ell(y^m)$ is the vector label of the m-th vertex y^m of the simplex under consideration. The algorithm is more complex, but we can achieve higher accuracy by repeatedly applying the method to smaller and smaller solution simplices. For a discussion of the solution of systems of linear equations on array processors see e.g. [12].

(c) Multiprocessor structure associated with nonlinear equation solver

Part of a multiprocessor structure for the two-dimensional example mentioned above is shown in Fig. 2. Each PE has its own arithmetic and logic unit (ALU) plus memory. If n is the dimension of the problem, then for each point of the triangulation we have $n + 1$ (in our two-dimensional example $n + 1 = 3$) PE's . These $n + 1$ PE's are interconnected by a bus and form a column of PE's . Apart from this bus each PE in the column is connected to $2n + 2$ (in our case 6) other PE's in the manner shown in Fig. 2. For the computation of function values the PE's in a column communicate with each other via a bus. For the computation of the labels the PE's in a column have to communicate with each other. For the determination of a solution simplex all the columns of PE's associated with the vertices of a simplex have to communicate with each other. Furthermore, all the PE's have to be connected to a master PE via a system bus.

(d) Sophisticated features for multiple solutions

We consider a further sophistication of above method. In our problem we map e.g. an n-dimensional simplex into itself. We have triangulated this simplex into smaller simplices. One can associate a group with these simplices. The map

$$f: S^n \to S^n$$

from the n-dimensional simplex to itself induces a map f_* (group homomorphism) from a group to itself. f_* is given by

$$f_*(z) = \deg f \cdot z ,$$

where z denotes an element of the group. $\deg f$ is a uniquely determined integer, called the topological degree of f . In some sense this number counts the simplices of S^n which are mapped into one simplex of S^n . The computations for f and f_* can be performed on two separate arrays of PE's. The actual calculations of $\deg f$ (on the arrays of PE's reserved for f_*) is best being

handled by performing a certain count of oriented n-simplices (we form the difference between certain positively oriented n-simplices and certain negatively oriented n-simplices). The calculation of f_* leads to an estimate on the expected number of solutions (gives a minimum number). If deg f is e.g. k , then we can use k (or possibly more) PE arrays to obtain higher accuracy for multiple solutions by refining the grid size of each of the k (or more) solution simplices concurrently (as explained earlier). A suitable mechanism (e.g. compiler) can prepare the PE arrays as soon as the topological degree is computed.
The realization of such a scheme can be considered to be an implementation of some of the abstract ideas discussed earlier in this paper. We have two categories. One category has simplices as objects and piecewise linear maps as mappings. In this category we compute our solution simplices. The second category has groups as objects and group homomorphisms as mappings. In this second category we compute the topological degree, i.e. a lower estimate for the number of solutions. The connection between these two categories is described by a functor. This implementation by a multiprocessor can be considered to reflect in some sense the abstract mathematical notions of category and functor.

(e) Using cubes instead of simplices

From the applications point of view the use of cubes instead of simplices seems to be preferable. Especially when one subdivides a given n-dimensional region it seems easier to have cubes and to subdivide these cubes. From an abstract point of view the use of n-dimensional cubes instead of n-dimensional simplices does not seem to involve too many difficulties. Unfortunately most of the theory developed (homology and cohomology theory) uses simplices for detailed discussions and only recently have some texts appeared using cubes instead of simplices. For computational purposes a large number of details is required and at present there is a gap between the abstract theory and the precise details required for algorithmic purposes. I would like to mention that H.W. Kuhn in [13] was perhaps the first one to point out the importance of cubes in a computational context.
(It is always possible to subdivide an n-dimensional cube into n! n-dimensional simplices. In order to avoid dealing with a large number of simplices one would prefer a theory based solely on n-dimensional cubes).
From an applications point of view, from a computational point of view and from the designers point of view the use of cubes seems to be preferable to the use of simplices. At present the necessary details required for computational purposes are not available.

IMPLEMENTATION USING OFF-THE-SHELF PRODUCTS

In order to reduce development costs it is desirable to use readily available components. At present there is only one microprocessor with multiprocessing capabilities on the market, namely the Intel 8086. The Intel 8086 is a 16-bit CPU with an instruction set which includes arithmetic instructions covering various types of addition, subtraction, multiplication and division. The 8086 microprocessor is supported by a number of support devices like the 8089 I/O processor, the 8288 bus controller, the 8289 bus arbiter, the 8282 and 8283 latches, etc. These support devices simplify the construction of multiprocessors considerably. Furthermore, one would have to use the 8231 arithmetic processing unit for function evaluations and the 8232 floating point processor for 32-bit and 64-bit precision arithmetic. For a dedicated application the program can reside in ROM's. The components required for one PE, including the bus interface, can be assembled on a single board. Alternatively one could use readily available boards like the iSBC 86/12A single board computer, the iSBC 310 high speed mathematics unit, the iSBC 300 32K-byte RAM expansion module, the iSBC 340 16K-byte EPROM/ROM expansion module, etc. The 8086 family is supported by a bus called multibus. The number of PE's a multibus can support is 16. For our purposes this means that one can use this multibus only for one-dimensional implementations. A reasonable

two-dimensional implementation already requires more than 16 PE's.
To summarize off-the-shelf implementation of a nonlinear equation solver one can say that suitable microprocessors plus support elements are already available.

CONCLUSION

In this paper we briefly discussed various mathematical structures and showed how they can be matched to multiprocessor structures. An application in the form of a dedicated multiprocessor, namely a nonlinear equation solver, was then outlined. Matching a multiprocessor structure to mathematical structures leads in this example to a fast nonlinear equation solver for systems of n equations in n unknowns (with possible multiple solutions). Investigations into off-the-shelf implementation of such a design conclude the paper.

REFERENCES

1 ACM Computing Surveys, vol. 9 (1977) (whole issue).

2 Gurd, J., Watson, I. 'Data driven system for high speed parallel computing; Part I: Structuring software for parallel execution'. Computer Design (June 1980), pp. 91-100.

3 Gurd, J., Watson, I. 'Data driven system for high speed parallel computing; Part II: Hardware design'. Computer Design (July 1980), pp. 97-106.

4 Mead, C., Conway, L. Introduction to VLSI systems, (chapter 8). Addison-Wesley, Reading, Mass., 1980.

5 Arbib, M.A., Manes, E.G. Arrows, structures, and functors. Academic Press, New York, 1975.

6 MacLane, S. Categories for the working mathematician. Springer, New York, 1971.

7 Enslow, P.H. (Ed.) Multiprocessors and parallel processing. Wiley, New York, 1974.

8 Scarf, H., Hansen, T. The computation of economic equilibria. Yale University Press, New Haven and London, 1973.

9 Todd, M.J. The computation of fixed points and applications. Springer, New York, 1976.

10 Forster, W. (Ed.) Numerical solution of highly nonlinear problems. North-Holland, Amsterdam, 1980.

11 Forster, W. 'Fixed point algorithms and array processors'. pp. 169-179 in reference [10].

12 Sameh, A.H., Kuck, D.J. 'On stable parallel linear systems solvers'. J.ACM, vol. 25 (1978), pp. 81-91.

13 Kuhn, H.W. 'Some combinatorial lemmas in topology'. IBM Journal of Research and Development, vol. 4 (1960), pp. 518-524.