FIFTH GENERATION COMPUTER ARCHITECTURES

EDITED BY
J. V. Woods

FIFTH GENERATION COMPUTER ARCHITECTURES

Proceedings of the IFIP TC 10 Working Conference on
Fifth Generation Computer Architectures
Manchester, U.K., 15-18 July, 1985

edited by

J. V. WOODS
University of Manchester
U.K.

1986

NORTH-HOLLAND
AMSTERDAM · NEW YORK · OXFORD · TOKYO

ISBN: 0 444 87987 0

Published by:
ELSEVIER SCIENCE PUBLISHERS B.V.
P.O. Box 1991
1000 BZ Amsterdam
The Netherlands

Sole distributors for the U.S.A. and Canada:
ELSEVIER SCIENCE PUBLISHING COMPANY, INC.
52 Vanderbilt Avenue
New York, N.Y. 10017
U.S.A.

Library of Congress Cataloging-in-Publication Data

IFIP TC 10 Working Conference on Fifth Generation Computer Architectures (1985 : Manchester, Greater Manchester)
Fifth generation computer architectures.

1. Computer architecture--Congresses. 2. Electronic digital computers--Congresses. I. Woods, J. V. (John Vivian), 1939- . II. IFIP TC-10. III. Title.
QA76.9.A73I36 1985 004.2'2 86-4443
ISBN 0-444-87987-0

PRINTED IN THE NETHERLANDS

FOREWORD

This book contains a collection of the papers that were presented at the Working Conference on Fifth Generation Computer Architectures held at the University of Manchester Institute for Science and Technology from July 15th to 18th, 1985. The Conference was sponsored by Technical Committee TC-10 of the International Federation for Information Processing (IFIP).

The aim of the Working Conference was to draw together 50 or so of the world's leading reseachers in the area of novel computer architecture so that they could present ongoing work and disseminate their recent results. The following areas were thought to be of interest: architectures for functional and logic programming, parallel dataflow and reduction machines, inference architectures and human-machine interfaces. Both formal and informal submissions were sought, the latter being progress reports on projects of general interest. The Programme Committee invited three speakers to give keynote talks and selected 15 of the 51 submitted papers for formal presentation. A number of informal presentations were also selected. Authors were invited to prepare written versions of their papers which would be considered for publication in the Proceedings. Referees' comments were sought and the resultant changes included.

The Programme Committee felt that the main objective of the Working Conference had been achieved in that many leading researchers attended and disseminated their latest work. The discussion sessions were particularly lively and informative. There was an interesting mix of the famous and the less well-known which added an extra dimension to the event. All in all the Conference was another testimony to the benefit of the international collaboration that is fostered by IFIP.

Of course, these events always rely on the hard work of the few altruistic persons who can turn such an idea into a reality. We would like to offer our sincere thanks to everyone who assisted in making the Working Conference a success: firstly to our late Conference Chairman, Professor Tohru Moto-oka; secondly to the Chairman of IFIP TC-10, who was also our Chairman of Local Affairs, Professor David Aspinall; thirdly to our Proceedings Editor and Conference Secretary, Dr. Viv Woods; and finally to the Members of the Programme Committee and all the local helpers who contributed to the smooth running of the conference.

Professor Tohru Moto-oka

It is a matter of great sadness that these Proceedings must open by reporting that the Conference Chairman, Professor Moto-oka, of the University of Tokyo, died on the 11th November, 1985. Professor Moto-oka was renowned throughout the world for his leading role in the Japenese Fifth Generation Computer Systems Project and he was a key member of IFIP TC-10. His achievements already attest to his stature, but it is particularly sad that he will not be able to see the final fruits of his endeavours in the above areas. He will be greatly missed.

Hideo Aiso and John Gurd
Programme Committee Chairman and Deputy

TABLE OF CONTENTS

PART A

LOGIC PROGRAMMING ARCHITECTURES

Fifth Generation Computer Architectures, J.V. Woods (ed.)
Elsevier Science Publishers B.V. (North-Holland)

A NEW PARALLEL INFERENCE MECHANISM BASED ON SEQUENTIAL PROCESSING

Yukio Sohma, Ken Satoh, Kouichi Kumon,
Hideo Masuzawa, Akihiro Itashiki

Artificial Intelligence laboratory
Fujitsu Laboratories Limited
Kawasaki, Japan

We propose a new parallel inference mechanism which we call the KABU-WAKE method. In this method, an inference is made by a depth-first search in a processor element. If there is a request from another processor, a job is split up between PEs for OR parallel inference.

Thus, the overhead in each processor and for communication between processors can be minimized.

Through our experimental system, we found that the KABU-WAKE method was particularly effective for applications with large search trees.

1. INTRODUCTION

This paper discusses a new parallel inference method, and gives an evaluation of this method using our experimental system.

We have been researching the feasibility of realizing a high-speed inference machine that uses parallel processing. We propose a new parallel inference mechanism based on the idea of sequential inference, but extended further to include parallel inference. Chapter 2 explains our basic concept for parallel inference. Chapter 3 explains the features and operating principles of the KABU-WAKE method. Chapter 4 explains an experimental system dedicated to the KABU-WAKE method. Chapter 5 summarizes the results of experiments using this system.

2. OUR APPROACH TO PARALLEL INFERENCE

Inference speed can be improved by operating a number of identical processing elements (PEs) in parallel. However, the processing efficiency of each PE is also very important. Currently, there are already a number of speed-up techniques for single PE, i.e., sequential inference. Thus, we developed a parallel inference method based on sequential inference, to take full advantage of existing technology.

3. THE KABU-WAKE METHOD

The KABU-WAKE method is one of the parallel inference mechanisms currently being researched by the Fifth Generation Computer Project.

3.1 Features

The KABU-WAKE method has the following features:

- Each PE sequentially processes a search tree by a depth-first search.
- Each PE splits its current tree for OR parallel processing only when requested by another PE.

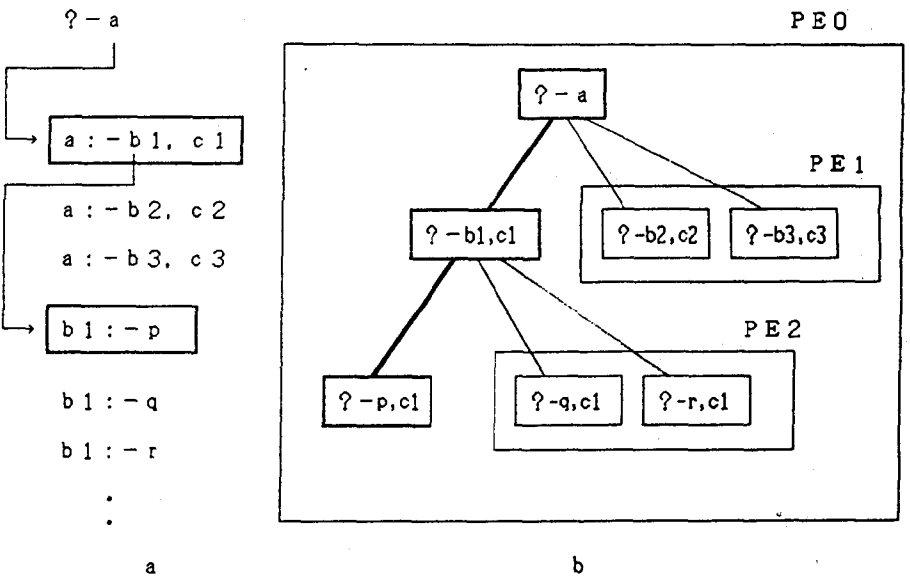

Fig.1 Basic mechanism of KABU-WAKE

These features have the following advantages:

- Low overhead in each PE
  In a PE, processing is the same as for sequential inference -- no special processing is necessary for parallel inference. Therefore, each PE works at the same speed as for sequential inference.
- Little communication between PEs
  A job is split and passed only when requested by

another PE, which reduces the amount of required communication. In addition, since a job is split near the root of the search tree, granularity of the job can be made as big as possible.

- Limited number of OR processes
  Since the number of OR processes is limited to the number of PEs, there is no danger of an unexpected increase in the number of OR processes.

3.2 Principles of Operation

The operating principles of the KABU-WAKE method are explained here with some reference figures. Figure 1-a is an example of a data base and inquiry written in Prolog. We will omit arguments of the predicate to simplify the explanation. The arrow shows the ordinary flow of sequential inference. Figure 1-b shows the operating principles of the KABU-WAKE method. The part indicated by bold lines shows the processing of one PE and corresponds to the processing indicated by the arrow in Figure 1-a, which is a sequential inference.

If a job request is made by another PE, the job is split at the branch closest to the root of the search tree and half is passed to the requesting PE. If another request comes, the job is split again at the branch next closest to the root of the tree and half is passed to the second requesting PE. The job can be further split up among other PEs.

4. EXPERIMENTAL SYSTEM

We built an experimental parallel inference machine to test the effectiveness of the KABU-WAKE method quantitatively. We designed a hardware configuration suitable for our method and installed the KABU-WAKE interpreter on that hardware.

4.1 System Configuration

Figure 2 shows the hardware configuration. The system consists of 16 PEs one of which is used for I/O. PEs are connected by two kinds of exclusive networks.

The system components are as follows:

- PE:
  This is a processing element in which a parallel inference interpreter based on the KABU-WAKE method is installed. If a PE receives a request from another PE during inference, the PE splits its current job and passes half to the other PE.
  Each PE has a copy of the entire data base.

- CONT network:
  This is a communication route for requesting a job. PE status information, whether a PE is processing, is circulated on this network. A PE which is performing an inference can check the status information to give a free PE a part of its job.

- DATA network:
  This is a communication route for transferring a split job.

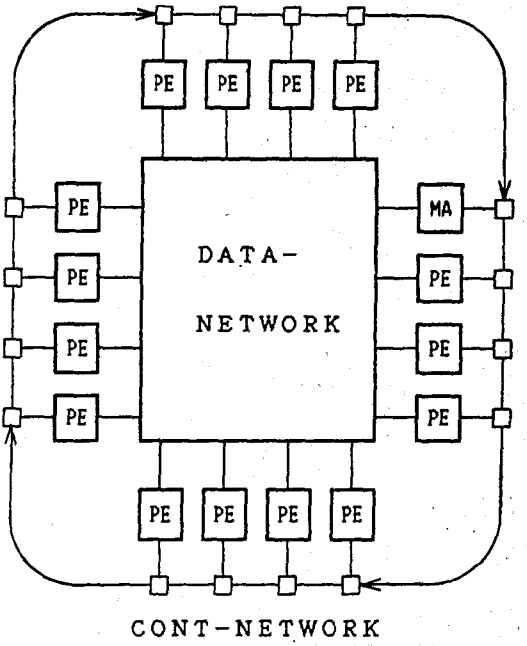

Fig.2 System Configuration of Experimental machine

4.2 Implementation

The following points must be implemented in the system:

(1) Unbinding of variables

When a search tree is split, the status of the variables in the split tree must be as if all the processes located left below of the split position in the search tree had failed and backtracked, i.e., unbinding.

To speed up this unbinding operation, we reserved an area for each variable which memorizes the time it was bound. Each time a subgoal is made, a new level number, corresponding to the depth of subgoal, is assigned. For variables bound during the processing of that subgoal, the same level number as that of the subgoal is tagged. When a tree is split, we can determine whether binding should be released by comparing the level of the subgoal

to be split with the variable's level number.

In this method, the time required to release variables for splitting is proportional to the number of variables in the subgoal to be split. Using a trailing stack is another alternative, but we think it would take more time.

(2) Use of rule numbers

When a tree is split and transferred, it is transferred in the form of a subgoal. However, since some of the definitions for that subgoal are already being processed, the other PEs must start from subsequent definitions. Therefore, we adopted a method in which a special predicate called a rule number is prepared and attached to the subgoal before transfer, to indicate where to start execution.

(3) Control of requests

Some jobs cannot be split even when a request is received from another PE. If this happens frequently, PE performance will deteriorate.

To prevent this, we used a request reception flag to indicate whether a job can be split. This enables a PE to concentrate on job without being disturbed by other PEs.

(4) Protocols between PEs

For the communication between PEs, we have two kinds of dedicated networks, as explained above. On the control network, 16 time slots, one fixed location for each PE are reserved. Each PE can access one time slot at a time, and that time slot is shifted to next PE every 250 ns.

Each PE is also provided with dedicated hardware (CNA: Control Network Adapter) to access the control network. Each CNA sets its PE's activity status (busy/idle) on its assigned time slot. When the CNA in a busy PE identifies an idle PE, it sets a flag to indicate local PE for splitting a job.

The PE then splits its job and part of the job is passed to the idle PE via the Data network.

(5) Management of activities

In our system, there is no special method for managing PE activities. The system automatically tries to offload the busy PE, but only if there is an idle PE in the system.

5. RESULTS OF THE EXPERIMENT AND THEIR EVALUATION

Data is still being collected from the experimental system. Although it is too early to make a full evaluation, we would like to give a preliminary evaluation.

5.1 Degree of Total Performance Improvement

We gathered data using an n-queen problem as a benchmark program. The values were obtained by averaging several or several tens of measured values.

As expected, the execution time varies for each measurement, because of non-determinism of hardware behavior. This is illustrated in table 1.

Table 1 Deviation of measured data

	Execution time (σ)	number of communications (σ)
6Q	1090 (47) ns	114 (6)
7Q	2798 (113) ns	199 (25)
8Q	9331 (81) ns	250 (20)
9Q	40056 (117) ns	358 (34)
10Q	191252 (300) ns	413 (55)

The figure in parenthesis is the standard deviation.

Figure 3 shows the relationship between the number of PEs and execution time. For a problem with a large search tree, such as an 8-,9-, or 10-queens problem, the execution time decreases almost linearly as the number of PEs increases.

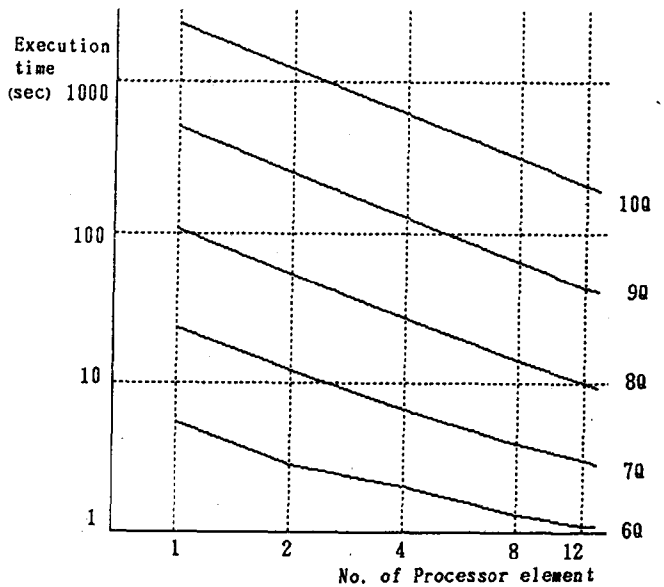

Fig.3 Speed up factor due to parallel inference

Based on the same data, Figure 4 shows the performance improvement ratio due to the increase of the number of PEs. The average activity ratio per PE is about 99% when thirteen PEs are doing parallel inference for a 10-queens problem.

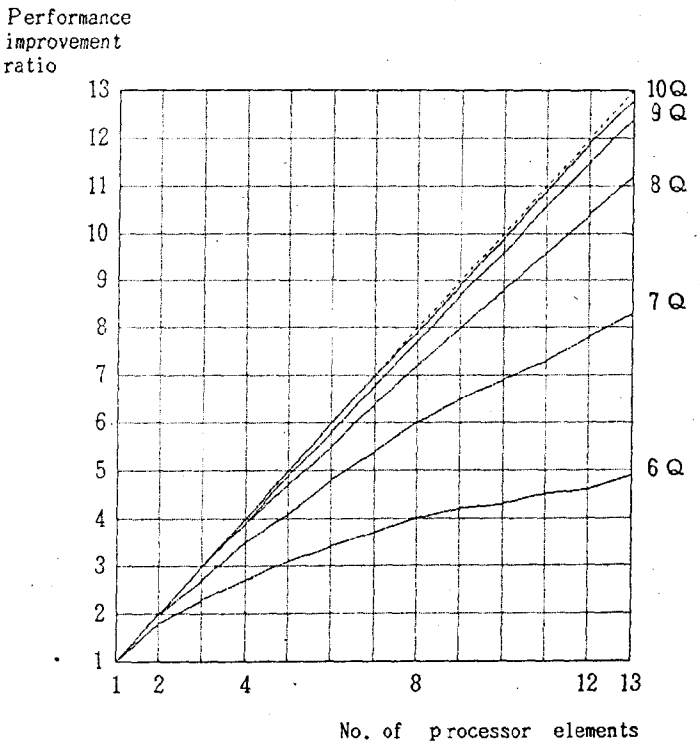

Fig.4 Performance improvement ratio due to parallel inference

5.2 Detailed Analysis

Detailed analysis of the collected data has yielded the following results:

(1) PE performance

Absolute performance

One PE has a performance of approximately 1 KLIPS, almost equal to C-prolog on a vax11/780.

Overhead during sequential execution

We tried to create the system based on sequential

inference, to reduce the overhead for parallel processing as much as possible. As mentioned before, however, overhead is required for memorizing binding times of variables.

Therefore, we measured the percentage of the binding time processing out of the total processing. When the benchmark program (n-queen, quicksort) was executed using one PE, the value was less than 6%. In other words, when continuous processing is performed in one PE, our parallel inference interpreter performs about as well as a sequential inference interpreter.

Overhead during parallel execution

We also evaluated the processing time of one PE when a job is executed in parallel by several PEs.

We sampled the processing of a PE in time series to check the type of processing, for a 7-queens problem executed by 13 PEs. Figure 5 shows the results.

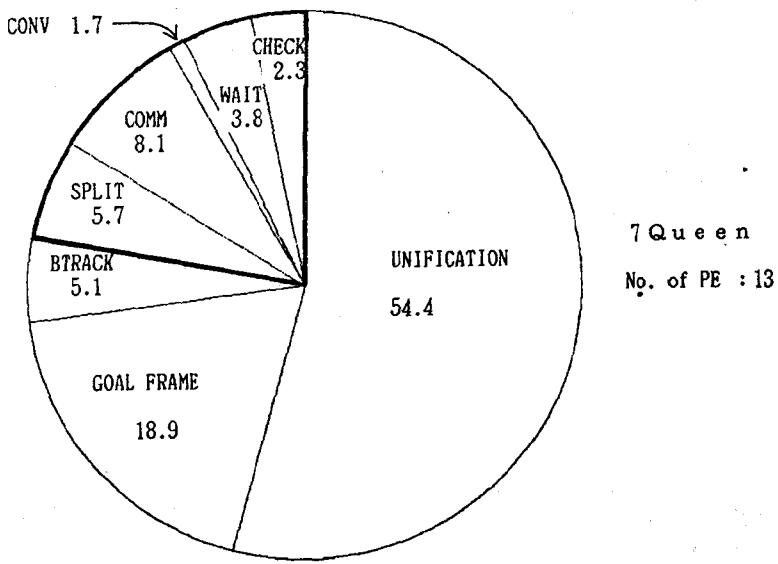

Fig.5 Details of execution time in processor element

The abbreviations in the graph have the following meaning:

UNIFICATION : Unification
GOAL FRAME : Creation of frame on stack
BTRACK : Undo for backtracking