Database Machines and Knowledge Base Machines

Edited by

Masaru Kitsuregawa
Hidehiko Tanaka

Kluwer Academic Publishers

DATABASE MACHINES AND KNOWLEDGE BASE MACHINES

edited by

Masaru Kitsuregawa
University of Tokyo
Hidehiko Tanaka
University of Tokyo

KLUWER ACADEMIC PUBLISHERS
Boston/Dordrecht/Lancaster

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for the UK and Ireland:
Kluwer Academic Publishers
MTP Press Limited
Falcon House, Queen Square
Lancaster LA1 1RN, UNITED KINGDOM

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

Database machines and knowledge base machines / edited by Masaru Kitsuregawa.
p. cm. — (The Kluwer international series in engineering and computer science ; 43. Parallel processing and fifth generation computing)
Contains papers presented at the Fifth International Workshop on Database Machines.
ISBN 0-89838-257-2:
1. Electronic digital computers—Congresses. 2. Data base management—Congresses. 3. Expert systems (Computer science)--Congresses. I. Kitsuregawa, Masaru. II. Hidehiko Tanaka. III. International Workshop on Database Machines (5th : 1987 : Tokyo, Japan) IV. Series: Kluwer international series in engineering and computer science ; SECS 43. V. Series: Kluwer international series in engineering and computer science. Parallel processing and fifth generation computing.
QA76.5.D2687 1988 87-29646
004—dc19 CIP

Printed in the United States of America

DATABASE MACHINES AND KNOWLEDGE BASE MACHINES

THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

PARALLEL PROCESSING AND FIFTH GENERATION COMPUTING

Consulting Editor

Doug DeGroot

Other books in the series:

PARALLEL EXECUTION OF LOGIC PROGRAMS
John S. Conery ISBN 0-89838-194-0

PARALLEL COMPUTATION AND COMPUTERS FOR ARTIFICIAL INTELLIGENCE
Janusz S. Kowalik ISBN 0-89838-227-0

MEMORY STORAGE PATTERNS IN PARALLEL PROCESSING
Mary E. Mace ISBN 0-89838-239-4

SUPERCOMPUTER ARCHITECTURE
Paul B. Schneck ISBN 0-89838-234-4

ASSIGNMENT PROBLEMS IN PARALLEL AND DISTRIBUTED COMPUTING
Shahid H. Bokhari ISBN 0-89838-240-8

MEMORY PERFORMANCE OF PROLOG ARCHITECTURES
Evan Tick ISBN 0-89838-254-8

PREFACE

This volume contains the papers presented at the Fifth International Workshop on Database Machines. The papers cover a wide spectrum of topics on Database Machines and Knowledge Base Machines. Reports of major projects, ECRC, MCC, and ICOT are included. Topics on DBM cover new database machine architectures based on vector processing and hypercube parallel processing, VLSI oriented architecture, filter processor, sorting machine, concurrency control mechanism for DBM, main memory database, interconnection network for DBM, and performance evaluation. In this workshop much more attention was given to knowledge base management as compared to the previous four workshops. Many papers discuss deductive database processing. Architectures for semantic network, prolog, and production system were also proposed.

We would like to express our deep thanks to all those who contributed to the success of the workshop. We would also like to express our appreciation for the valuable suggestions given to us by Prof. D. K. Hsiao, Prof. D. J. DeWitt, and Dr. H. Boral. The workshop was sponsored by the Information Processing Society of Japan and the Institute of New Generation Computer Technology, with the support of Japan Electronic Industry Development Association, in cooperation with the Association for Computing Machinery, Japanese Society for Artificial Intelligence, and Japan Society for Software Science and Technology. We would like to thank all those who gave us their support, including many companies which supported us financially. We are grateful for the assistance we received from the Mampei Hotel. We wish to thank Miss Y. Tasaku of Inter Group for taking care of all the arrangements for the workshop and also Mr. D. Childress and Mr. Y. Yamamoto of Kluwer Academic Publishers for publishing the proceedings. We, on behalf of the program committee, wish to express our gratitude to the many others who contributed to the success of the workshop.

Program Chairman *M. Kitsuregawa*
General Chairman *H. Tanaka*

CONTENTS

I Project Research for Knowledge Base Machines

ICM3: Design and evaluation of an Inference Crunching Machine

Jacques Noyé, Jean-Claude Syre, et al.

ECRC - European Computer-Industry Research Centre GmbH
Arabellastr. 17 D-8000 Muenchen 81 West Germany

ABSTRACT

The ICM (Inference Crunching Machines) Project is a research project conducted at ECRC to design and evaluate the architectures of processors dedicated to Prolog. Although there is a real trend in developing co-processors for AI, little has been done to tailor the abstract Prolog machines known in the literature to the real features of existing hardware. ICM3 is one example of such an effort to modify the software Prolog machine, leading to a powerful and efficient implementation in hardware. After an introduction giving the framework of the ICM Project, we describe the modified abstract machine, then the architecture of ICM3, emphasizing its unique features (asynchronous prefetch unit, dereferencing and unification unit). Some functional and gate level simulation results follow. We conclude with comments on what we learned from ICM3, and introduce the next project under way at ECRC, in the Computer Architecture Group.

INTRODUCTION

This paper presents the architecture and performance evaluation of ICM3 (Inference Crunching Machine), a co-processor dedicated to Prolog. ICM3 is one output of the ICM research project, begun at ECRC in 1985, which also involved H. Benker, T. Jeffré, G. Watzlawik, A. Poehlmann, S. Schmitz, O. Thibault and B. Poterie as full time researchers of the Computer Architecture Group.

When we started the ICM Project, several other important research studies were under way at other places: the PSI I machine [12, 10] was starting running at a tremendous speed of 100 Klips (Kilo logical inferences per second), peak rate, 30 in a sustained regime, a major breakthrough compared to the conventional 1 to 3 Klips of current interpreters. Then machines running compiled Prolog were introduced. The CHI [7] machine was revealed with a peak performance of 280 Klips (OK, it was in ECL, but an interesting milestone). At about the same time, the Berkeley Group with Al Despain [5, 4] , and other people from California (Evan Tick [9]), were announcing an incredible estimate of 450 Klips (still peak performance). Most of these machines (including the more recent X-1 of Xenologic [8, 3], and to some extent too, the PSI-II of ICOT [6]) are more or less a direct mapping of the quasi "standard" WAM (Warren Abstract Machine) defined by David Warren in 1983 [11]. The achieved sustained performance, in the order of 100 to 150 Klips (already 6 to 8 times better than pure software Prolog systems), together with the idea that the hardware potentialities of the WAM were still to be explored, motivated the ICM project.

The project started by investigating the various system architectures possible by

associating a hardwired engine and a software Prolog environment running on a conventional Host system. We summarize below:

- In a back end processor system, the designer is free to define a format for a machine word. On the other hand, he will have to implement a complete ad-hoc software system to compile, emulate, and run the programs. While the communication with the Host system is easy, it will not allow a tightly-coupled execution (e.g. for data base or external language extensions).

- In a co-processor system, the constraints are different: the memory system being shared, the designer is faced with terrible problems of pre-defined word lengths, memory bus throughput, host compatibility, and will also face the operating system to solve his problems of communication. The software effort can be small (if the machine runs in a "one-shot" fashion, i.e. a single query is posed and completely solved by the co-processor), or it can be high, if one allows a bidirectional control of the program between the co-processor and the host system. A co-processor is more attractive for flexibility and extensions.

The ICM3 machine [4] belongs to the second class (we have another design, called ICM4 [1], corresponding to the first class of system architectures). Thus it is a co-processor, sharing the memory system of a host machine, and running compiled Prolog programs in a one-shot manner. The other requirements for ICM3 were the following:

- Achieve a peak performance of more than 400 Klips (rather easy), and a sustained performance of more than 200 (this is less easy), by really tuning the hardware design to the fundamental mechanisms involved in a Prolog execution.

- Be connectable to a 32-bit host processor, with the constraints this implies on word format, memory, ...

- Use a conventional technology.

- Be a vehicle of research to learn lessons from various design choices. In fact some of the choices were made to evaluate a feature, not really because the feature was already proven excellent.

This paper is organized as follows: Section 2 describes the abstract machine, stressing the differences with the conventional WAM where necessary. Section 3 introduces the hardware architecture of ICM3. Section 4 focusses on the Tag Flag Calculator, an important source of speed-up. Section 5 presents performance results, with comparisons, where possible, with existing Prolog systems. Section 6 gives a qualitative evaluation of ICM3, and section 7 concludes by introducing the future activities of the ICM research team, part of the Computer Architecture Group of ECRC.

ABSTRACT MACHINE

The ICAM (ICM Abstract Machine) is based on the ideas initially presented by D.H.D. Warren in [20]. While preserving the main organization of the WAM, we modified it along two different lines. First, it was completed and enhanced, taking into account the experience of ECRC Prolog, a compiled Prolog system developed in the Logic Programming Group [9, 17]. Second, it was finely tuned to a hardware implementation. The following short description gives some examples of these two points.

Data formats

A Prolog term is represented by a 32-bit word composed of an 8-bit tag and a 24-bit value. Within the tag, the type field (4 bits) gives the type of the object, the mark field (4 bits) was reserved for future use (garbage collection).

Nine different Prolog types are defined: the bound and unbound variables, the compound terms (lists and structures) and a number of constant types (atoms, functors, 16-bit short integers, 32-bit long integers and reals). A tenth type corresponds to internally used data pointers (for instance tops of stacks).

Memory layout

The ICAM manages six memory areas:

- The Code Area statically stores compiled Prolog programs.
- The Local Stack stores the AND/OR tree corresponding to a Prolog execution, which is represented by a stack of frames, the environment frames (AND-level) and the choice point (or backtrack) frames (OR-level).
- According to the structure copy technique, the Global Stack mainly stores compound terms created during unification for argument passing purposes.
- The Trail stores the addresses of the bindings which have to be reset on backtracking.
- The Registers hold the current state of the computation (program pointers, tops of stacks, pointers to the current choice point and environment frames...) and are used for argument passing purposes. These Registers, which are the most frequently accessed objects, are actually held in hardware registers. There are 16 Argument Registers, and 9 State Registers. E points to the current environment frame, B to the current backtrack frame, T to the top of the Local Stack, TG to the top of the Global Stack, TT to the top of the Trail, GB to the Global Stack backtrack point. P is the Program Pointer, CP (Continuation Program Pointer) points to the next instruction when a clause is completed, BP points to the next instruction in case of backtracking (Backtrack Program Pointer), and S is a Structure pointer to the Global Stack used during unification.
- The PDL (Push-down List) is a small stack used by the general unification

procedure. It is always empty between two ICAM instructions.

Generally, the Local and Global Stacks as well as the Trail expand towards higher addresses, as more procedures are invoked, and contract on backtracking. In addition, the Tail Recursion Optimization performs an early space recovery on the Local Stack.

Environment handling

In the WAM, when a clause comprises several goals, an environment is allocated before head unification by setting the register E (current environment) to the top of the Local Stack and pushing the continuation information (this information defines what to do next if the current goal succeeds). The bindings of the permanent variables (i.e. the variables which must survive the resolution of the first goal) are then stored in the second part of the frame during unification.

This makes it possible, as an extension to the Tail Recursion Optimization, to trim the environment. Let us suppose that a permanent variable occurs for the last time in the second goal of a clause. This clause will be compiled such that this variable will be the last one to be pushed on the Local Stack. If, before solving the second goal, this variable is still at the top of the Local Stack, the corresponding location can be recovered.

These features bring a number of drawbacks. First, each time a location is trimmed, the possibility of a dangling reference has to be checked, which is quite costly. Second, the top of the Local Stack is computed dynamically, necessitating, in a deterministic state, an access via the CP register (the continuation program pointer) to the Code Area. In a hardware implementation, this access to the Code Area disturbs the process of prefetching the next instructions. The difficulty can be circumvented, as in [7], by adding a new register holding the dynamic size of the current environment. Unfortunately, this must then be stored in the environment and choice point frames which may nullify the expected memory gain.

To overcome these drawbacks, trimming has been abandoned. As in ECRC Prolog, E points to the top of the environment frame, and T is always Max(E,B). Moreover, the allocation of environments (i.e. pushing the continuation information and setting E, now to the top of the Local Stack) can be delayed until unification has succeeded. In case of failure, unnecessary work is avoided.

Choice point handling

Three refinements (the last two inspired by ECRC Prolog) have been brought to choice point handling.

First, the backtrack program pointer BP becomes a State Register. In the same way CP is saved in the environment frame (the AND frame), BP is saved in the choice point frame (the OR frame). Additionally, both the BP and CP registers can be held in the Prefetch Unit part of the machine. This results in a notable speedup in instruction fetch, since these registers are now immediately available.

Secondly, we have introduced the management of shallow backtracking. When a clause head unification may shallow backtrack (i.e. there is at least one alternative clause), the compiler guarantees that the argument registers are not modified. In case of failure, neither the continuation (E and CP) nor the argument registers have to be restored.

The indexing scheme has been modified in order to eliminate the possibility of creating two choice point frames for the same call. This scheme saves time, and speeds up cut operations, too.

The instruction set

The kernel ICAM instruction set comprises 69 instructions, inspired from the WAM, and mostly implemented on a 32-bit word. The AND-level instructions are responsible for environment management and goal sequencing. The OR-level instructions deal with choice point management and clause sequencing as well as with the cut operation. The indexing instructions filter the candidate clauses using as a key a compiler determined argument. The get instructions perform the head unification of non-nested terms. The put instructions are responsible for argument passing of non-nested terms and the unify instructions deal with nested term unification and argument passing.

Lastly, the built-in instructions implement most of the Prolog built-in predicates as direct microcode calls. This allows register allocation optimization and extends the effectiveness of shallow backtracking.

ARCHITECTURE OF ICM3

The functional architecture of ICM3 (figure 1) looks like a conventional computer: a Prefetch Unit (PRU) is connected to a Code Cache (COCA), and an Execution Unit (EXU), is connected to a Data Cache (DACA). Thus instructions and data are separately accessed and managed, and the PRU is asynchronous to the EXU. Both units cooperate using basically two flags: NIP (Next Instruction Please) informs the PRU that the EXU has reached a point where the current instruction will run to its end without problems (such as a fail operation), so that the PRU can supply the right next instruction. PRURDY informs the EXU that this instruction is supplied in a decoded form directly executable by the EXU. Both units are separately microprogrammed.

The Execution Unit

Beside the basic Register File and the ALU, there exists a number of specific boxes dedicated to tasks involving specific Prolog operations, as shown in figure 2. The Tag Flag Calculator is described in the next section.

The Register File. This is made of AMD29334 four-port 64x32-bit register chips, and contains the Argument Registers (up to 16), the State Registers (E, B, T, TG, TT, GB, S), a 32 word PDL (or bottom of the PDL), and intermediate or scratch registers used by the microprograms.

The Register Address Control. This unit provides the register addresses for the