TM-0862

A Pipelined Microprocessor for Logic Programming Languages

by
H. Nakashima, Y. Takeda, K. Nakajima,
H. Andou & K. Furutani (Fujitsu)

February, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Pipelined Microprocessor for Logic Programming Languages*

Hiroshi Nakashima Yasutaka Takeda
Katsuto Nakajima Hideki Andou
Kiyohiro Furutani

Information Systems and Electronics Development Laboratory,
Mitsubishi Electric Corporation,
5-1-1 Ofuna, Kamakura, 247, Japan,

Phone: +81-467-44-6136 (Office)
+81-466-23-0022 (Home)
e-mail: hiroshi@mjv870.isl.melco.co.jp

*Summary of the paper submitted to ICCD'90.

1 Introduction

In the Japanese Fifth Generation Computer Project, a large scale parallel inference machine, PIM/m, is being developed[1]. In PIM/m, up to 256 processor elements are connected to form a two-dimensional mesh network.

The processor element has a pipelined microprocessor specialized to the execution of logic programming languages. The microprocessor, called PU (Processing Unit), is also used as a key component of AI workstations. Thus, PU has capability to execute two different type logic programming languages, KL1[2] for PIM/m and ESP[3] for the AI workstation. KL1 is a parallel logic programming language based on flat-GHC[4], and is very powerful to represent parallel processes communicating each other. ESP is the language in which Prolog and object oriented language features are combined, in order to make it easy to construct large and practical AI programs.

Since the execution mechanisms of both languages stand on *unification*, data typing and dereference are very important operations for efficient implementation. For these operations, PU has powerful mechanisms to manipulate tagged data. Especially, the pipelined data typing and dereference are the most unique features of PU. Those mechanisms greatly contribute to the high performance, 833 KLIPS[†] for KL1 *append*, and 1282 KLIPS for ESP.

In this paper, the hardware architecture of PU is described, focusing on its data typing and dereference mechanisms.

2 Hardware Architecture

2.1 PIM/m and Its Processor Element

In PIM/m, up to 256 processor elements (PE) are connected to form a two-dimensional mesh network, as shown in Figure 1. Up to 32 SCSI ports are available to connect disks and front end processors (FEP). FEP is an AI workstation which will provide a comfortable programming environment for PIM/m users. It also acts as an intelligent controller of interactive I/O devices.

[†]Kilo Logical Inference Per Second.

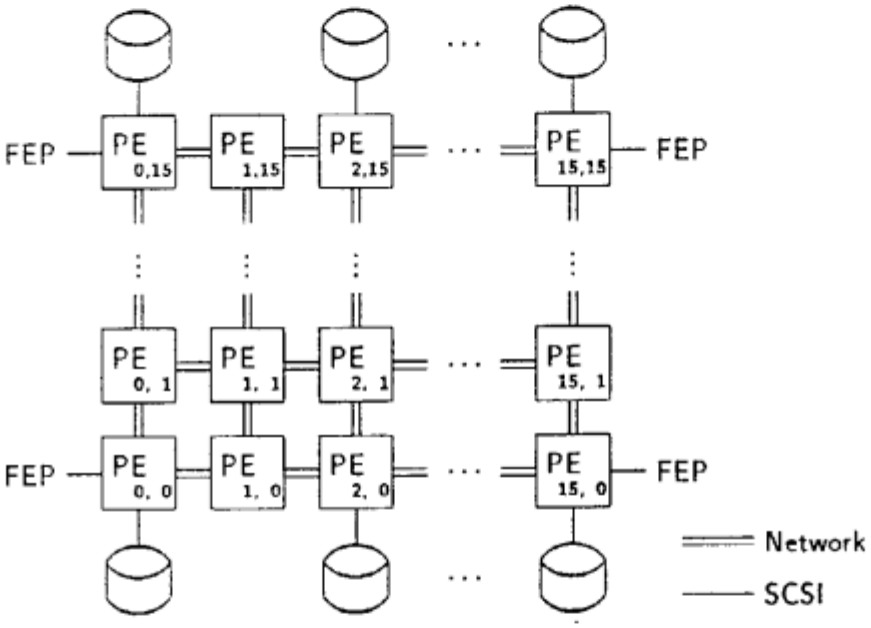

Figure 1: PIM/m System Configuration

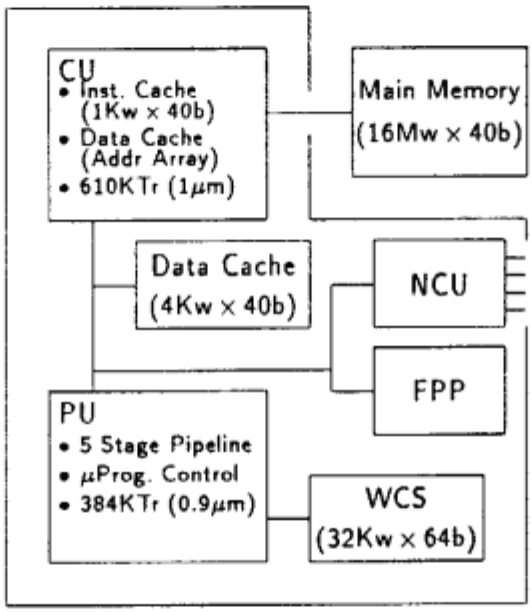

Figure 2: Processing Element of PIM/m

Each PE has three VLSI chips, PU (Processing Unit), CU (Cache Unit) and NCU (Network Control Unit), as shown in Figure 2. PU is a 40-bit microprocessor, which executes KL1 and ESP under the control of a microprogram stored in 32K-word WCS. The architecture of PU is described in the following sections.

CU contains a 1K-word instruction cache and a 4K-word data cache[‡]. Address translation buffers for instructions and data are also installed on CU. The size of each buffer is 64, and the configuration is 2-set set associative.

NCU has a switching circuit, which has four bidirectional channels connecting adjacent processor elements and two buffers for message packets to/from the processor element. The packet transmission and buffering are automatically performed without any interruption of the execution of PU and CU.

2.2 Processing Unit (PU)

Figure 3 shows the configuration of PU. PU executes WAM[§]-like instructions for KL1 and ESP[2, 5, 6]. Argument and temporary registers (A_n/X_n) are implemented as a register file. Another register file, WR, contains WAM's control registers, except for a program counter and two structure pointers which are hardware counters. Each register is 40 bit width, including 8 bit tag to represent data types.

PU has five pipeline stages, **D**, **A**, **R**, **S** and **E**.

The **D** (Decode) stage has a RAM table for instruction decode. Each entry of the table contains the start address of the microprogram routine for an instruction, and the *nano-code* to control the following stages. This RAM decoder makes it easy to develop the microprogram.

The **A** (Address Calculation) stage calculates the operand address by adding two of following resources, according to the nano-code.

- An operand field of the instruction.
- Program counter.
- A_n/X_n specified by an operand field.

[‡]The data array of the data cache is not included.

[§]Warren Abstract Machine.

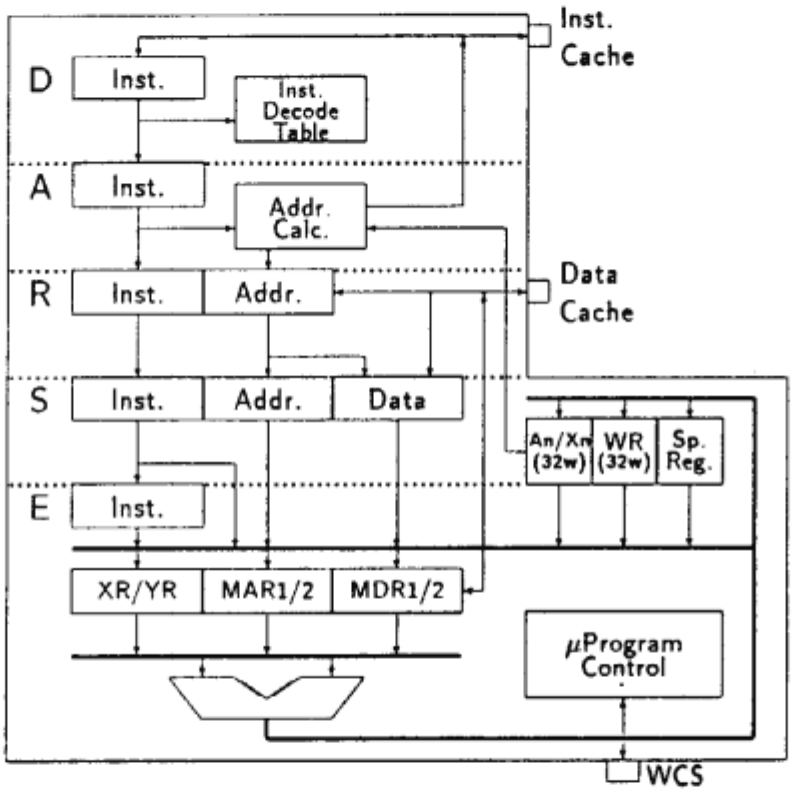

Figure 3: Configuration of PU

- Two special address registers, which are contained in WR.

In the KL1 execution, the special address registers contain the address of the alternative, and the base of a frame representing a goal, the execution of which is postponed. In the ESP execution, they contain the address of continuation, and the base of environment. The A stage also controls instruction fetch, including conditional and unconditional branch operations.

The R (Read Data) stage fetches an operand from data cache using the calculated address, if necessary.

The S (Set Up) stage selects three operands from the following resources and transfers them to the E (Execution) stage, according to the nano-code.

- An operand field of the instruction.
- The operand fetched by the R stage and its address.
- A_n/X_n specified by an operand field.
- WR.
- Two structure pointers.

In conventional pipelined processors, the operand set up operation is performed by the stage like R. PU, however, has an additional special stage, S, for the operation. The reason for introducing the S stage is that it is required for the pipelined data typing and dereference, as discussed latter. Longer pipeline will have some drawbacks, because longer time will be taken to recover pipeline break. The drawbacks, however, will not be so serious, because the break is mainly caused by backtracking in logic programming languages. That is, backtracking takes many cycles which are enough to re-fill all the stages with instructions to be executed after the backtracking.

The E stage has two pipelined phases controlled by microinstructions. The first phase contains A_n/X_n, WR, and special registers including the program counter and structure pointers. This phase is shared by the S and E stages for the operand set up. The second phase has two temporary registers (XR/YR), two memory address registers (MAR1/2), and two memory data registers (MDR1/2). Two of those registers are input to ALU, and the result is written into registers in the first and/or second phase.

2.3 Data Typing and Dereference

Data typing and dereference are very important for efficient implementation of logic programming languages. Both data typing and dereference are performed by checking the tag of data and changing the control flow according to the result. PU has powerful mechanisms, including the pipelined data typing and dereference, for these operations.

The **E** stage has the following microprogram operations for tag checking.

(1) Two-way conditional jump. The jump condition is obtained by comparing the tag of a register with an immediate value or the tag of another register.

(2) Three-way jump. The tag of MDR1 or MDR2 is compared with an immediate value and *reference* tag.

(3) Multi-way jump. A RAM table, which contains jump offsets, is looked up by the tag of MDR1 or MDR2.

Those operations requires two machine cycles. The first cycle makes the jump condition or offset, and the second generates the jump address and fetches the microinstruction.

The pipelined data typing and dereference, which are most unique features, mainly depends on the **S** stage.

The **S** stage has the following three functions for data typing.

(1) Modify the microprogram entry address comparing the tag of the operand fetched by the **R** stage with a immediate value.

(2) Set up the offset of a multi-way jump, which can be performed by the first microinstruction, looking up the RAM table by the tag of the operand fetched by the **R** stage.

(3) Set up the two-way jump condition, which can be examined by the first microinstruction, comparing the tag of an operand transferred to the **E** stage with a immediate value.

The first two functions requires the special stage between the **R** and **E** stages.

The **S** stage also performs dereference. When the dereference from A_n/X_n is ordered, the **R** stage fetches the operand if the A_n/X_n contains *reference*

pointer, while it always fetches the operand in the case of the dereference from memory. In both cases, the **S** stage examines the tag of fetched data, and repeatedly reads memory until a non-reference data is obtained.

In the KL1 execution, the state of the reference path is examined using MRB (Multiple Reference Bit)[7], during the dereference. MRB is a part of the tag, and indicates that there are (or may be) other pointers to same data object, as shown in Figure 4. Thus, if the MRBs of all the pointers on a reference path are off, the pointers and the terminal data can be reclaimed as garbage, when the unification of the terminal is completed.

In order to support this incremental garbage collection, the **S** stage passes the following information to the **E** stage.

SRP (Single Reference Path): MRBs of all the pointers and terminal are off.

COL (Collectable): MRBs of the first two pointers are off.

Moreover, these information can be used for the modification of the micro-program entry address, combining the data typing result. Thus, the **E** stage can easily decide whether the reclamation of the pointers and terminal is necessary, as shown in Figure 4.

3 Performance Evaluation

As described in the previous section, the most unique features of PU are the pipelined data typing and dereference. To evaluate the efficiency of these features, the performance of *append* in KL1 and ESP is estimated by simulation. Moreover, the performance of alternative architectures is also evaluated. Figure 5 shows the performance of the following architectures.

(1) **PSI-II**: The performance of PSI-II[6], which is the predecessor of PU. Its CPU is also used as the element processor for a middle scale parallel inference machine, Multi-PSI/v2[1, 8]. The data typing and dereference are performed by the following tag comparison and microprogram conditional jump operations.

   (a) Output the content of a register to a data bus.

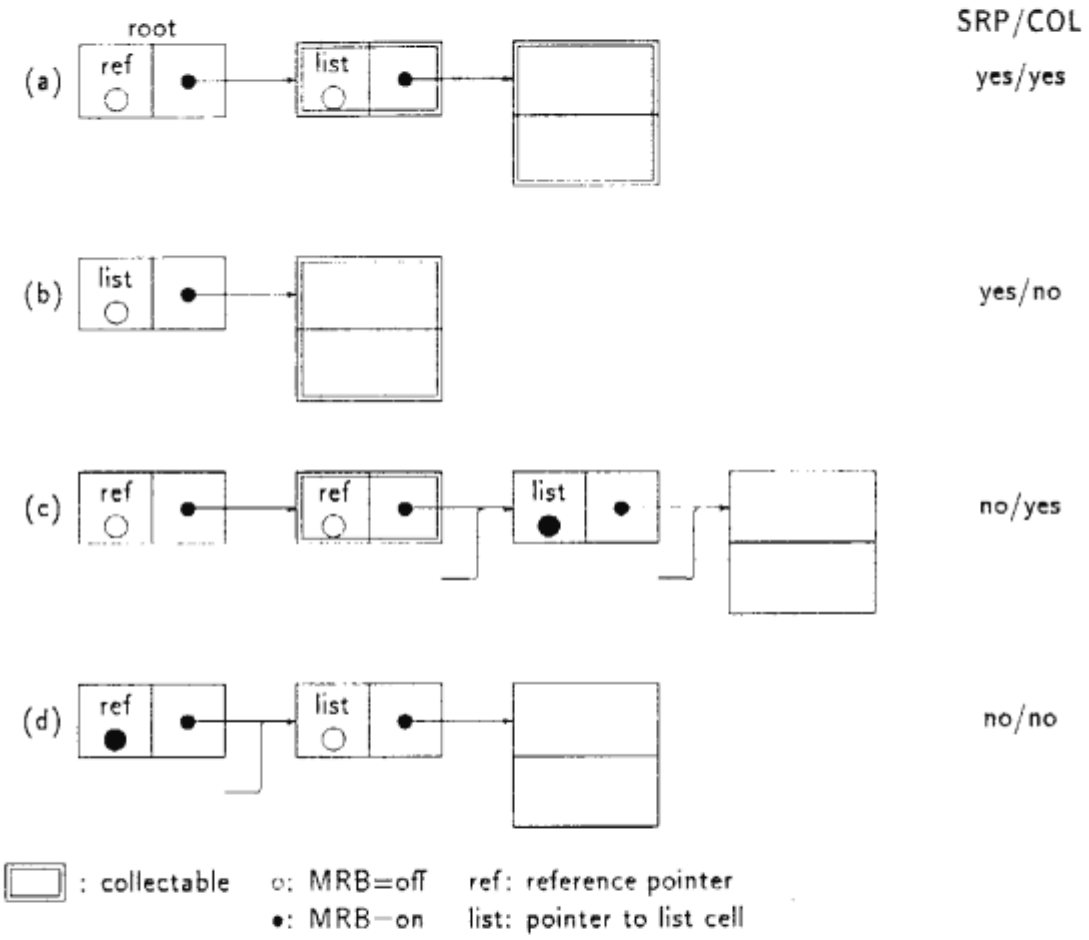

Figure 4: MRB and Realtime Garbage Collection

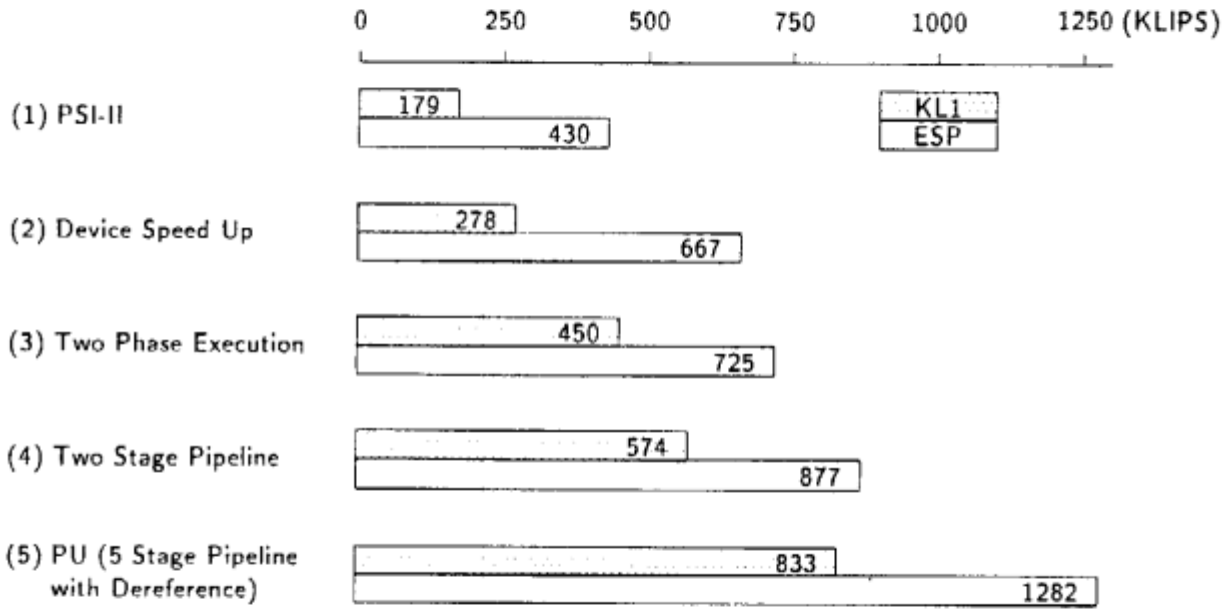

Figure 5: Performance Improvement by Pipeline Architecture

(b) Compare the tag of the data bus with a immediate value.

(c) Generate the next microinstruction address examining the comparison result.

(d) Fetch the next microinstruction from WCS.

These operations are performed in one cycle, 155 ns, and the path for them is one of the critical paths. The recursive clause of *append* is executed in 36 cycles for KL1, and 15 cycles for ESP.

(2) **Device Speed Up**: The performance assuming that the logic gates of PSI-II is as fast as those of PU. The cycle time will be 100 ns or more.

(3) **Two Phase Execution**: The performance assuming that the critical path is split into two pipeline phases, one of which consists of (a) and (b), and the other consists of (c) and (d). This configuration is the same as the E stage of PU, the cycle time of which is 60 ns. The recursive clause will be executed in 37 cycles for KL1, and 23 cycles for ESP.

(4) **Two Stage Pipeline**: The performance assuming that (a) and (b) are performed by a pipeline stage preceding the **E** stage. This configuration is similar to the combination of the **S** and **E** stages of PU, but the dereference is not pipelined. The recursive clause will be executed in 29 cycles for KL1, and 19 cycles for ESP.

(5) **PU**: The performance of PU, which has the pipelined dereference mechanism. The pipelined data typing is more efficient than that of (4), because any conditional jumps are not required in the **E** stage if the data typing is performed on dereferenced data. The recursive clause is executed in 20 cycles for KL1, and 13 cycles in ESP.

The results show that the pipelining reduces both the machine cycle and execution steps. Especially, the advantage of the pipelined data typing and dereference will be clear by comparing (5) with (4).

Table 1 shows the *append* performance of PU, and other machines for parallel and sequential logic programming languages. It will be clear that PU is one of the fastest machines for both languages.

Table 1: Performance in Append

	KLIPS PLPL/SLPL
PIM/m(PU)	833/ 1282
PSI-II[6]	179/ 430
Pegasus[9]	—/≈350
PLM[10]	—/≈400
CHI-II[11]	—/ 490
KCM[12]	—/ 833
IPP[13]	—/ 1035
IP1704[14]	—/ 1100
PIM/p[15]	≈600/—

PLPL: Parallel Logic Programming Language

SLPL: Sequential Logic Programming Language

4 Conclusions

The architecture of a pipelined microprocessor for logic programming languages is presented. It has very high performance, 833 KLIPS in KL1 *append* and 1282 KLIPS in ESP, owing to the pipelined data typing and dereference.

The first chips of PU and CU have been fabricated, and now are being evaluated in single processor environment. NCU and other components to construct a multiprocessor system are being developed. The first PIM/m system will be completed on the first quarter of 1991.

References

[1] S. Uchida, K. Taki, K. Nakajima, A. Goto and T. Chikayama, Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project. *Proc. Intl. Conf. on Fifth Generation Computer Systems 1988* , 1988.

[2] Y. Kimura and T. Chikayama, An Abstract KL1 Machine and its In-

struction Set. *Proc. 4th IEEE Symp. on Logic Programming*, 1987.

[3] T. Chikayama, Unique Features of ESP. *Proc. Intl. Conf. on Fifth Generation Computer Systems 1984* , 1984.

[4] K. Ueda, Guarded Horn Clauses: A Parallel Logic Programming Languages with the Concept of a Guard. *TR 208, ICOT*, 1986.

[5] D. H. D. Warren, An Abstract Prolog Instruction Set. *Technical Report 309, Artificial Intelligence Center, SRI International*, 1983.

[6] H. Nakashima and K.Nakajima, Hardware Architecture of the Sequential Inference Machine: PSI-II. *Proc. 4th IEEE Symp. on Logic Programming*, 1987.

[7] T. Chikayama and Y. Kimura, Multiple Reference Management in Flat GHC. *Proc. 4th Intl. Conf. on Logic Programming*, 1987.

[8] Y. Takeda, H. Nakashima, K. Masuda, T. Chikayama and K. Taki, A Load Balancing Mechanism for Large Scale Multiprocessor Systems and Its Implementation. *Proc. Intl. Conf. on Fifth Generation Computer Systems 1988* , 1988.

[9] K. Seo and T. Yokota, Design and Fabrication of Pegasus Prolog Processor. *Proc. Intl. Conf. on Very Large Scale Integration*, 1989.

[10] T. P. Dobry, A. M. Despain and Y. N. Patt, Performance Studies of a Prolog Machine Architecture. *Proc. 12th Intl. Symp. on Computer Architecture*, 1985.

[11] S. Habata, R. Nakazaki, A. Konagaya, A. Atarashi and M. Uemura, Co-Operative High Performance Sequential Inference Machine: CHI. *Proc. 1987 Intl. Conf. on Computer Design*, 1987.

[12] H. Benker, J. M. Beacco, M. Dorochevsky, Th. Jeffré, A. Pöhlmann, N.Noyé and B. Poterie, KCM: A Knowledge Crunching Machine. *Proc. 16th Intl. Symp. on Computer Architecture*, 1989.

[13] S. Abe, T. Bandoh, S. Yamaguchi, K. Kurosawa, and K. Kiriyama, High Performance Integrated Prolog Processor IPP. *Proc. 14th Intl. Symp. on Computer Architecture*, 1987.

[14] K. Maeda, et al., Mechanisms for Achieving Parallel Operations in a Sequential VLSI AI Processor. *Proc. 3rd Annual Parallel Processing Symp.*, 1989.

[15] T. Shinogi, K. Kumon, A. Hattori, A. Goto, Y. Kimura and T. Chikayama, Macro-Call Instruction for the Efficient KL1 Implementation on PIM. *Proc. Intl. Conf. on Fifth Generation Computer Systems 1988* , 1988.