Implementation and Analysis of

Multi-ALU Processor

Youyong Zou yzou1@cs.umbc.edu
Rong Yu ryu@cs.umbc.edu
Shuang Zeng szeng1@cs.umbc.edu
Kejian Hu khu1@cs.umbc.edu

Abstract

Much of the improvement in computer performance has come from architectural advances that increase parallelism. Historically, parallelism has been exploited either at the instruction level (ILP) with a single instruction or by partitioning applications into thousands of instructions. The ILP in applications is restricted by control flow and data dependencies. For multicomputers, there is limited coarse thread parallelism at small problem sizes and in many applications. There is a middle level parallelism that can fill the parallelism gap between these extremes -- efficient communication and synchronization mechanisms implemented in the Multi-ALU Processor (MAP) chip. It provides an opportunity for greater speedup. MAP including a thread creation instruction, register communication.

1 introduction

Modern computer systems extract parallelism from problems at two extremes of granularity: instruction-level parallelism (ILP) and coarse-thread parallelism. VLIW and superscalar processors exploit ILP with a grain size of a single instruction, while multiprocessors extract parallelism from coarse threads with a granularity of many thousands of instructions.
The parallelism available at these two extremes is limited. The ILP in application is restricted by control flow and data dependencies, and the hardware in superscalar designs is not scalable. Both the instruction scheduling logic and the register file of a superscalar grow quadratically as the number of execution units is increased. For multicomputers, there is limited coarse thread parallelism at small problem sizesand in many applications.

Middle-threads close the parallelism gap between the single instruction granularity of ILP and the thousand instruction granularity of coarse threads by extracting parallelism with a granularity of 50-1000 instructions. This parallelism is orthogonal and complementary to coarse-thread parallelism and ILP. Programs can be accelerated using coarse threads to extract parallelism from outer loops and large co-routines, middle-threads to extract parallelism from inner loops and small subcomputations, and ILP to extract parallelism from subexpressions. As they extract parallelism from different portions of a program, coarse-threads, middle-threads, and ILP work synergistically to provide multiplicative speedup.

These three modes are also well matched to the architecture of modern multiprocessors. ILP is well suited to extracting parallelism across the execution units of a single processor. Middle-threads are appropriate for execution across multiple processors at a single node of a

parallel computer where the interaction latencies are on the order of a few cycles. Coarse-threads are appropriate for execution on different nodes of a multiprocessor where interaction latencies are inherently 100s of cycles.

The Multi-ALU Processor (MAP) chip provides three on-chip processors and methods for quickly communication and synchronizing among them. A thread executing on one processor can directly write to a register on another processor. Threads synchronize by blocking on a register that is the target of a remote write or by executing a fast barrier instruction.

This report is for cmsc611 project, we will design a MAP DLX-like processor.

2. Multi ALU Technology

2.1 Thread Control

Invoking a thread on a remote processor is typically an expensive operation, requiring thousands of instructions to set up a stack and initialize system data structures. The MAP chip implements a fast *fork* instruction which invokes a thread on a remote cluster by automatically writing a remote program counter and updating the thread control registers. The example below starts a thread on cluster 1, using the address 20 as the running start point.

Fork #1, #20

Combined with the ability to write directly to the registers of the remote cluster, the fork instruction allows a remote procedure to be started quickly.

2.2 Communication

In coarse grained multiprocessors, communication between threads is exposed to the application as memory references or messages, both of which require many cycles to transmit data from one chip to another. In the MAP chip, threads on separate clusters will communicate through the shared registers. Since the data need not leave the chip to be transferred from one thread to another, communication is fast and well suited to middle-level threads.

The MAP chip implements register-register communication between clusters, allowing one cluster to write directly into the register file of another cluster, via the Cluster Switch. Register-register transfers are extremely fast, requiring only one more cycle to write to a remote register than to a local register. The result of any arithmetic operation may be sent directly to a remote register, without interfering with memory references or polluting the cache. Since the size of the register file limits the storage for communicated values, register communication is particularly suited to passing small amounts of data quickly, such as transferring signals, arguments, and return values between threads.

The following example send the value in register R2 to DLX#1's register R3.
Send #1, R2,R3

2.3 Synchronization

Barrier instruction: The simplest synchronization mechanism implemented by the MAP is the cluster barrier instruction *cbar*. The cluster barrier instruction stalls a thread?s execution until the threads on the other two clusters have reached a *cbar* instruction. Threads waiting for cluster barriers do not spin or consume any execution resources. The *cbar* instruction is implemented using six global wires per thread to indicate whether a *cbar* has been reached, and whether it has been issued.

3. Implenmentation

3.1 Specification

The specification part of the project report chronicals the implementation of the cpu inVHDL. Initial assumptions were made from the requirements regarding the instructions. The following lists these assumptions. for type instructions do:
Result <= For shift register and immediate type instructions assume the amound shifted is rs2 and the data to be shifted is rs1 or Immediate. The amount shifted is limited to 7 bits. For all unsigned adds and subtracts assume these instructions do not raise an exception on overflow.For subtracting use addition with a subtraction layer and CarryIn =1. The subtraction layer changes the second operand in this way: Operand2 <= Operand2 xor CarryInExt; CarryInExt = 111..1, or 232 - 1. Initial assumptions were also made about JALR, JR, J, and JAL. These were part of the initial project report and will not be reprinted here.

3.2 MAP design

The three parts of MAP is Clusters, Cluster-switch and Instrument memory. There are three clusters in our system. Every cluster is a basic DLX chip. So, this MAP has all the feature DLX has.

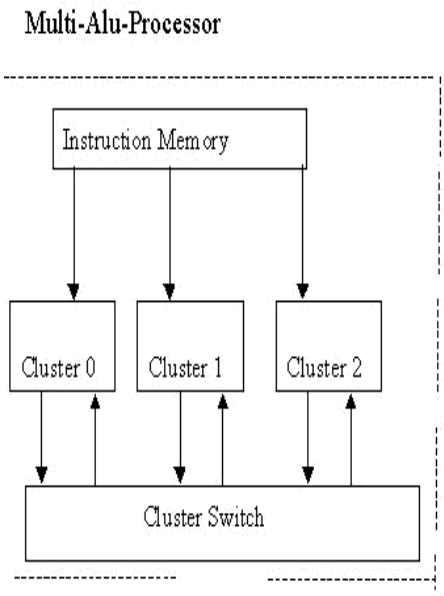
Instrument memory store all the

instrument used by MAP, every cluster can fetch instrument from it directly, so, compare with superscalar or VLIW , instrument fetch speed is speedup.

Cluster 0 is the main cluster, that is , it will control the running of other two clusters through fork and barrier instrument.

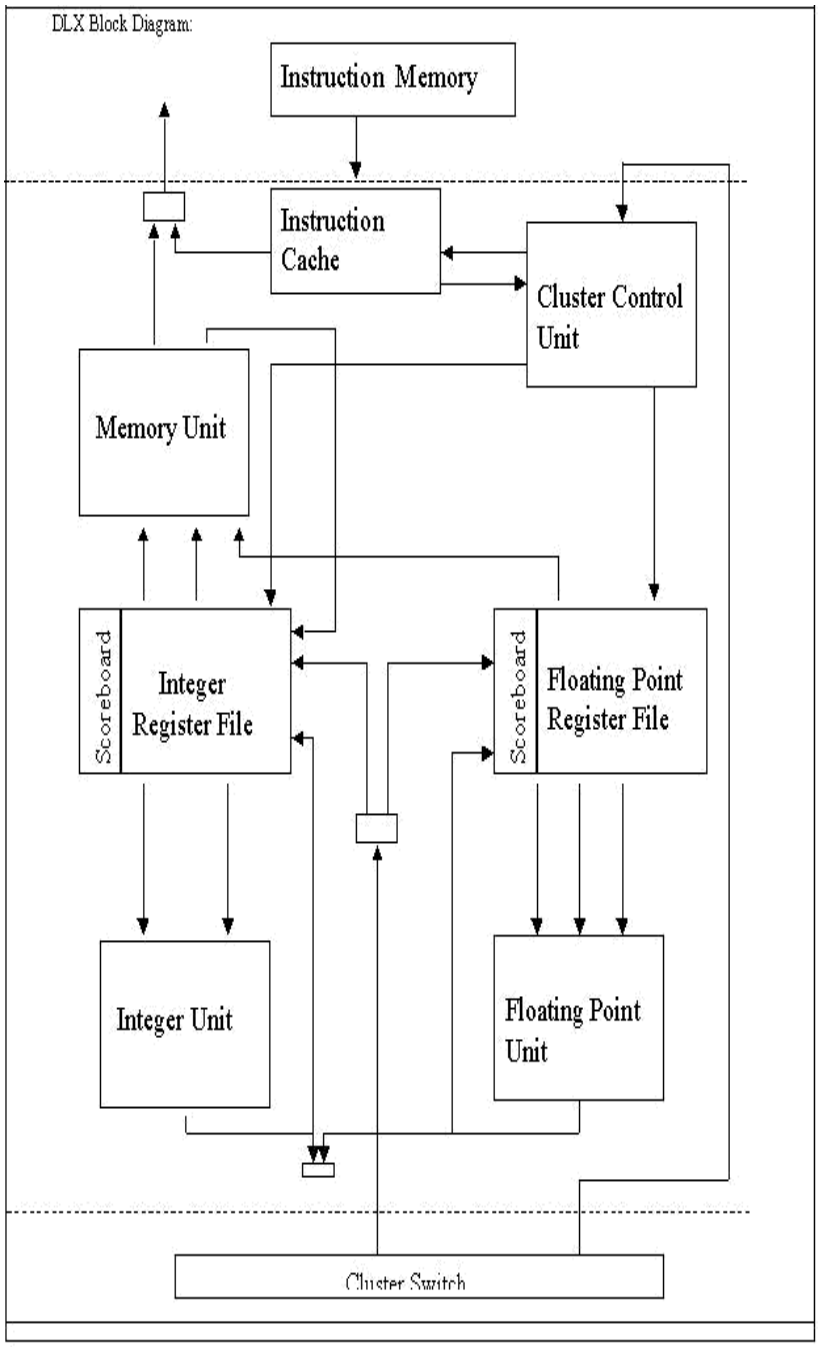
Cluster switch will transfer register data between clusters.

Multi-Alu-Processor

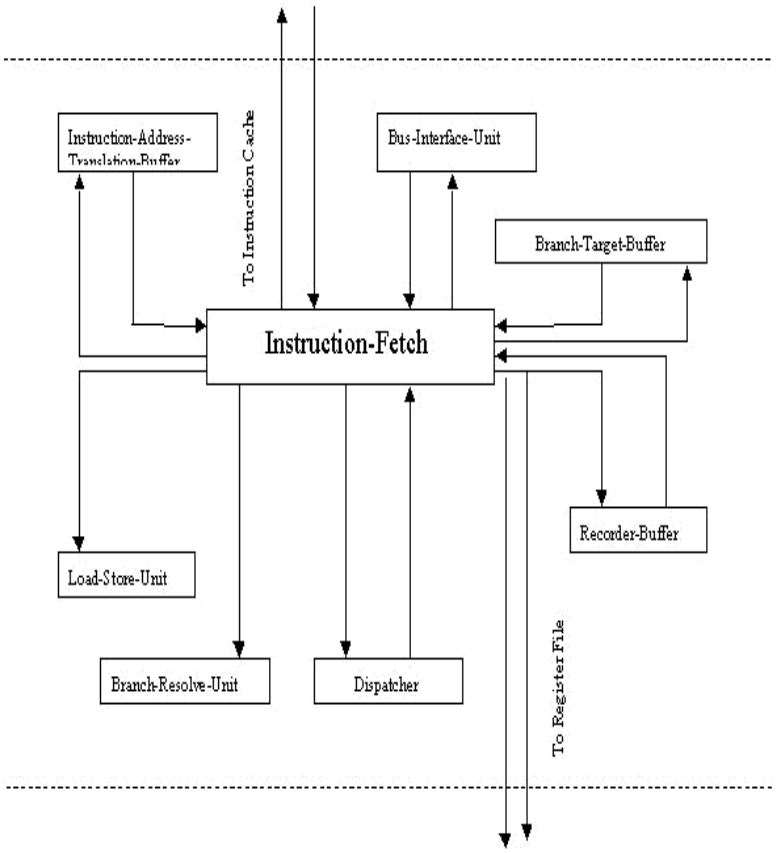


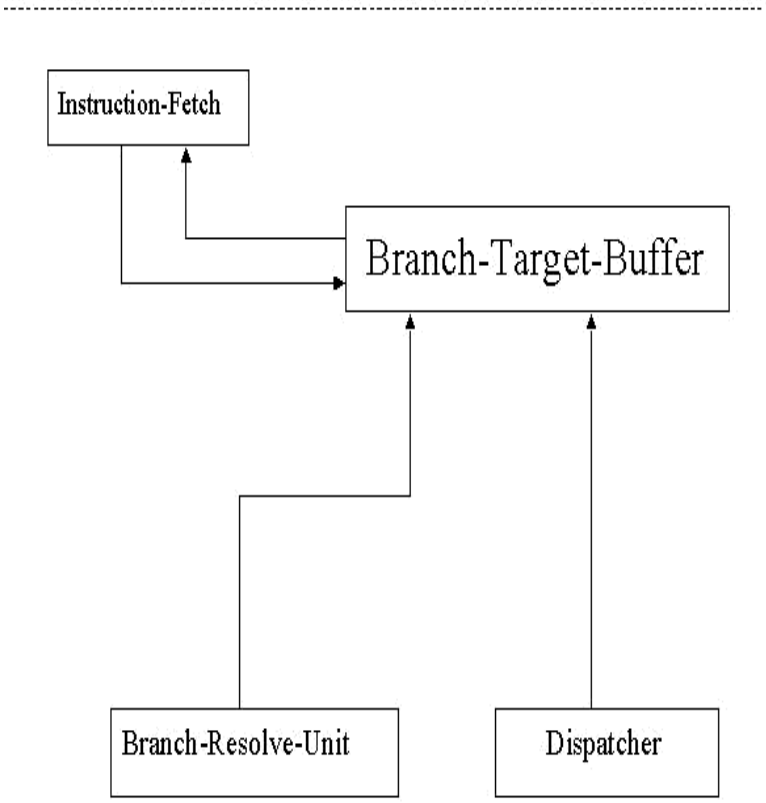
3.2 DLX design

DLX Block Diagram:



Cluster Control Unit(1):

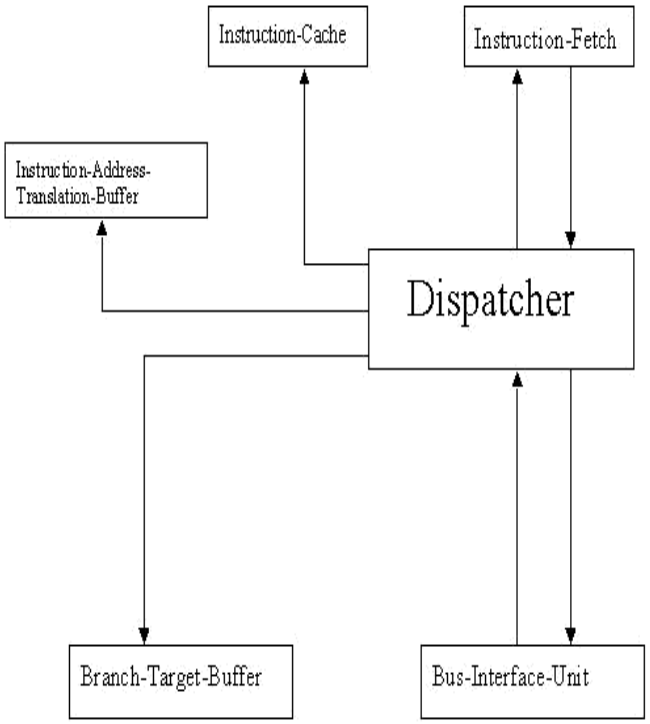

Instruction-Address-
Translation-Buffer
To Instruction Cache
Bus-Interface-Unit
Branch-Target-Buffer
Instruction-Fetch
Recorder-Buffer
Load-Store-Unit
Branch-Resolve-Unit
Dispatcher
To Register File



Cluster Control Unit(2):

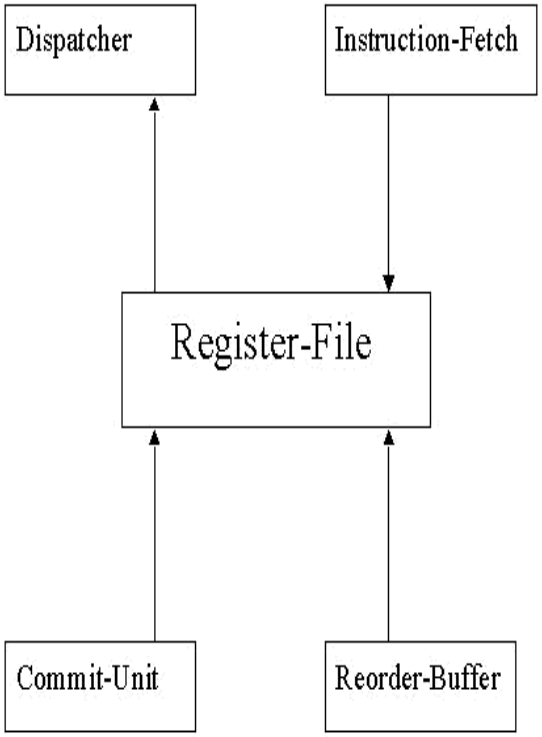
(Interface of the Branch-Target-Buffer)

Cluster Control Unit(3):

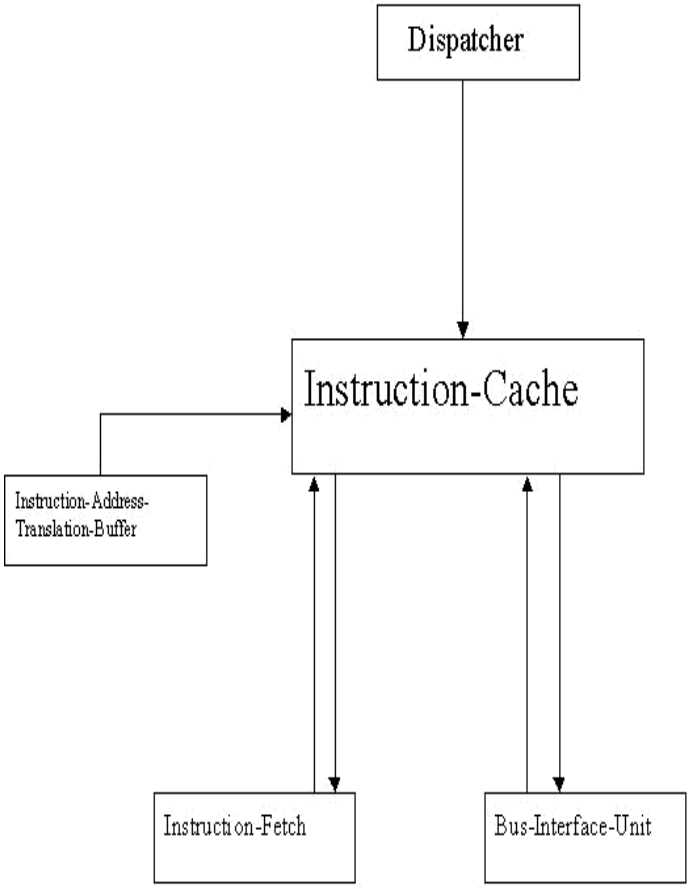


(Interface of the Dispatcher)

Interface of the Register-File:



Interface of the Instruction-Cache:



3.3 single dlx unit

3.3.1 Control unit

The control was implemented by assigning logical expressions to control signals. The logical expressions are one or more encoded instructions ORed together. When the logical expression evaluates to true, the control signal is asserted. In other words, when the input to the control is the opcode for add any control signal containing the encoding for the add

instruction will be asserted. In actuality a simplification of the instruction set reduces the number of encodings necessary to represent blocks of instructions.

The rest of the control is done in this way for both the alu and cpu control and the result are the control *entities control and AluControentity control_unit is*

```
    port(IR : in bit_vector(5 downto 0);
        lclock,stall : in bit;
        if_id_reset: out bit;
        c : out bit_vector(7 downto 0));
end control_unit;
```

3.3.2 register file

Register file consists of a set of registers that can be read or written by suppling two regis-ter numbers to be accessed.

The main element of the register file is the R-S clocked latch. It is a cross-coupled struc-ture that stores data signals. The output gets the values of the input when the clock signal is asserted and set and reset are both desserted.

```
entity registers is
  port (read0addr,read1addr,writeaddr: in bit_vector(4 downto 0);
      writedata: in bit_vector(31 downto 0);
      write_en,rf_clock,rf_reset: in bit;
      read0data,read1data: out bit_vector(31 downto 0));
end registers;
```

3.3.3 memory

There are two memory units in our CPU. One is instruction memory and the other is data memory. Instruction memory and Data memory are built using SRAM.
SRAM is an array of D filp flops. SRAMs have a fixed access time to any datum, though the read and the write access characteristics are often different. To initiate a read or write acces, the writeEnable or readEnable should be asserted. The SRAM read access time is usually specified as the delay from the time that readEnable is asserted and the address lines are valid until the time that the data are on the output lines.For writes we must supply the data to be written and the address , as well as the writeEn-able signal to cause the memory to write the data into the given locations. When the wri-teEnable is asserted the data will be written into the desognated location. The time to complete a write is the combination of the set_up times, holds_time, and the writeEnable pulse width.entity data_memory is
data memory is defined in data memory.vhdl:

```
  port (read0addr,read1addr,writeaddr: in bit_vector(4 downto 0);
      writedata: in bit_vector(31 downto 0);
      write_en,rf_clock,rf_reset: in bit;
      read0data,read1data: out bit_vector(31 downto 0));
end data_memory;
```

instrument memory is defined in instr_memory.vhdl :

```
entity instr_memory is
  port (read0addr,read1addr,writeaddr: in bit_vector(4 downto 0);
```

writedata: in bit_vector(31 downto 0);
write_en,rf_clock,rf_reset: in bit;
read0data,read1data: out bit_vector(31 downto 0));
end instr_memory;

3.3.4 Pipeline DataPath:

The datapath of the pipelined CPU is seperated into five stages as named below:
1. IF: Instruction Fetch.
2. ID: Instruction Decode and Register Fetch.
3. EX: Execution and effective address calculation.
4. Mem: Memory access.
5. WB: Write Back.

In a Pipeline CPU registers are inserted in between each stages of the pipes inoder to save the previous state.
The components are: if, if_id, id, id_exe, exe, exe_mem, mem,mem_wb,wb.

4. test

Not availavle now.

5. conclusion

bibliography

1. Exploiting fine-grain thread level parallelism on the MIT multi-ALU processor

Authors: *Keckler, Stephen W.; Dally, William J.; Maskit, Daniel; Carter, Nicholas P.; Chang, Andrew; Lee, Whay S.*
Author Affiliation: *Stanford Univ.*
Source: *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA Jun 27-Jul 1 1998 1998 Sponsored by: IEEE IEEE Comp Soc p 306-317 0884-7495*

2. Effects of explicitly parallel mechanisms on the multi-ALU processor cluster pipeline

Authors: *Chang, Andrew; Dally, William J.; Keckler, Stephen W.; Carter, Nicholas P.; Lee, Whay S.* **Author Affiliation**: *Stanford Univ.*
Source: *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors Oct 5-7 1998 1998 Sponsored by: IEEE IEEE p 474-481*

3. Dual-ALU CRISC architecture and its compiling technique

Authors: **Chou, Hong-Chich; Chung, Chung-Ping; Cheng, Shyi-Chyi**
Author Affiliation: **Chiao Tung Univ**
Source: **Computers & Electrical Engineering v 17 n 4 1991 p 297-312 0045-7906**

4. Designing multi-ALU microprocessors from LSI slices

Authors: **Palagin, A.V.; Slobodnyanyuk, T.F.; Yusifov, S.I.**
Source: **Cybernetics (English Translation of Kibernetika) v 23 n 5 Sep-Oct 1987 p 658-665 0011-4235**

5 .Concurrent event handling through multithreading

> ***Authors:*** **Keckler, Stephen W.; Chang, Andrew; Lee, Whay S.; Chatterjee, Sandeep; Dally, William J.**
> ***Author Affiliation:*** **Univ of Texas at Austin**
> ***Source:*** **IEEE Transactions on Computers 48 9 1999 IEEE p 903-916 0018-9340**

6. Processor coupling: Integrating compile time and runtime scheduling for parallelism

> ***Authors*****: Keckler, Stephen W.; Dally, William J.**
> ***Source:*** **Conference Proceedings - Annual Symposium on Computer Architecture May 1992 Sponsored by: IEEE Computer Soc; ACM SIGARCH Publ by IEEE p202-213**

7. M-machine multicomputer

> ***Authors:*** **Fillo, Marco; Keckler, Stephen W.; Dally, William J.; Carter, Nicholas P.; Chang, Andrew; Gurevich, Yevgeny; Lee, Whay S.**
> ***Author Affiliation:*** **Massachusetts Inst of Technology**
> ***Source:*** **International Journal of Parallel Programming v 25 n 3 June 1997 Plenum PublCorp p 183-212 0885-7458**

8. POWER2 floating-point unit: architecture and implementation

> ***Authors:*** **Hicks, T.N.; Fry, R.E.; Harvey, P.E.**
> ***Source:*** **IBM Journal of Research and Development 38 5 Sep 1994 IBM p 525-536 0018-8646**

9. Rotary pipeline processors

> ***Authors:*** **Moore, S.; Robinson, P.; Wilcox, S.**
> ***Author Affiliation:*** **Univ of Cambridge**
> ***Source:*** **IEE Proceedings: Computers and Digital Techniques 143 5 Sep 1996 IEE p 259-265 1350-2387**

10. The MIT Multi-ALU Processor

> ***Authors:***Keckler, Stephen W., Dally, William J., Chang, Andrew, Carter, Nicholas P., and Lee,
> Whay Sing
> ***Source:*** **ftp://cva.stanford.edu/pub/publications/hotchips97.ps.gz**

11. Thread Scheduling Mechanims for Multiple Context Parallel Processors

Authors: **Fiske, J. Stuart A., PhD Thesis**
Source**: ftp://cva.stanford.edu/pub/publications/stuart_phd_thesis.ps.Z**.

12. A quantitative comparison of parallel computation models

Authors: Ben H. H. Juurlink and Harry A. G. Wijshoff
Source: ACM digital library 1996