CmpE 511 Computer Architecture Term Paper
The Message-Driven Processor

by
Çetin Meriçli

January 3, 2006

Contents

Chapter 1

Introduction

The Message-Driven Processor is a processing node specially designed for multi-computer applications. The MDP consist of a 36-bit processor, memory and a set of network ports.

The notion of "Message-Driven" processing in the MDP is that the processing is performed in response to received messages from other nodes. Contrary to many message-passing architecture, there is no explicit "Receive" function. Instead, for each received message, a task is created and dispatched with proper attributes (priority etc.).

The desigh of the processing node relies on two major goals:

- Implementing a general purpose processing element natively providing adequate synchronization, naming and communication mechanisms.
- Creating a cost effective component that is both inexpensive and plentiful like a jellybean candy. Hence, a multi-computer made of such components would be called a *Jellybean Machine* or *J-Machine*.

For nearly a half of a century, processing elements have evolved a set of sequential mechanisms that are appropriate for many sequential programming models. But, with the rise of the popularity of parallel machines, it became clear that building concurrent machines with sequential processing elements is not appropriate because the sequential processing elements are just not good at most parallel computation models. So, this deficiency of hardware support for concurrent computation is covered with implementing such mechanisms at the software level which is a costly way.

The MDP supports a broad range of parallel programming models such as:

- Shared Memory
- Data Parallel

- Dataflow
- Actors
- Explicit Message Passing

The Message-Driven Processor is developed at Artifical Intelligence Laboratory of M.I.T. in joint efford with Intel. At the 1992, a 512-node J-Machine was in daily use at M.I.T. and there were preparations for upgrading it to 1024 nodes.
The rest of the paper is organized as follows: In the Chapter 2, the architecture of MDP is examined in the system level. Chapter 3 focuses on the instruction architecture and Chapter 4 discusses the network architecture. Chapter 5 gives brief information about the two experimental programming languages implemented for J-Machine and then we conclude and post some remarks in Chapter 6.

Chapter 2

System Architecture

The Message-Driven Processor can be viewed from two different points of view:

- Hardware View
- Software View

2.1 Hardware View

The MDP appears as a VLSI component equipped with:

- A processor
- A memory port
- Six bidirectional network ports
- A diagnostic port

The memory port provides a direct interface to up to 1M-word of ECC DRAM memory. The diagnostic port is used to issue supervisory commands and to read and write MDP memory. The network ports are used to connect MDPs together into a 3-Dimensional mesh network. Each port connects directly to the opposite port on an adjacent MDP node.

2.2 Software View

To a systems programmer, a bare J-Machine appears as a set of node memories and register files that can be operated on by an instruction set including mechanisms for communication, synchronization and naming. These mechanisms are used to build a programming model. For instance, a shared

memory model giving a single shared address space to the programmer can be built.

The MDP supports communication using a SEND instruction for message formatting, a fast network for delivery, automatic message buffering, and task creation upon message arrival.

Synchronization in the MDP is performed using message dispatch and presence tags. Presence tags are used to synchronize on the arrival of data. By tagging the locations appropriately, the reading of a location before the actual value of that location is calculated is prevented.

The MDP supports naming with segmented memory management and with translation instructions.

Chapter 3

Instruction Set Architecture

The MDP has an instruction set covering conventional microprocessor instruction-set architecture with the special extensions for supporting concurrent processing. Specifically, the MDP provides efficient hardware mechanisms for communication, synchronization and naming.

3.1 Register Set

The MDP provides three different register sets for supporting three levels of execution: background, priority 0 and priority 1. The MDP executes at the background level when there are no pending messages. A task is created when a message is received and the execution level of the task is determined according to the priority of the received message. At any time, the task with the highest priority is executed within the MDP. So, if a message with a higher priority is received, the actual task being executed is suspensed and the execution levels and the tasks being executed are swithced as well as the corresponding register set.

3.2 Tags

Tags in the MDP are used for type checking and synchronization. The memory is organized as 36-bit words in which the 32-bits are used for storing the actual value and the remaining four bits are used for tags. Tag values are defined for primitive user data types as well as for system data types. Four tags are user-definable. If type checking is enabled, the MDP checks operand tags to determine which form of an instruction to execute (which can be interpreted as hardware-level polymorphism). If the operands are not compatible with the instruction, an exception is raised.

Two tags, FUT and CFUT are used for inter-task synchronization. When a memory location is allocated, the location is marked initially with a CFUT tag denoting the location is currently empty. When a task produces a value

for that location any time later, the CFUT tag is replaced with corresponding tag. FUT tag is used for global synchronization where CFUT is used for local synchronization.

3.3 Instructions

The MDP uses 17-bit, fixed format, three-address instructions. Each instruction specifies an operation, two register operands and a third operand that may be a register, a memory location or a constant. Two instructions are packed into each 36-bit word. Any instruction-stream word not tagged as an instruction is interpreted as a constand and loaded into the register R0. This provides a very efficient way for loading 36-bit arbitrary constants.

3.4 Naming

Naming is supported in MDP via translation instructions and segmented addressing. The ENTER instruction enters an arbitrary translation from a 36-bit key to a 36-bit data value in set-associative cache mapped into the on-chip memory. The XLATE instruction looks up the data value associated with a key. These instructions can be used to translate an object's name into a physical segment descriptor or a node number to support a global virtual address space.

3.5 Communication

The MDP provides hardware level support for end-to-end communication including formatting, injection, delivery, buffer allocation, buffering and task scheduling. The MDP transmits a message using a series of SEND instructions. An injected message is delivered to the destination node by the network. At the destination, it is buffered in a hardware FIFO queue. There are separate queues for the messages having separate priorities.

3.6 Task Scheduling

When a message reaches to the head of the queue, a task is created to handle it by changing the thread of control and creating a new addressing environment. The dispatch mechanism is used directly to process messages requiring low latency. Other messages specify a handler that locates the required method, using the translation mechanism described above, and then transfers control to it.

Chapter 4

Network Architecture

The Message-Driven Processor contains a network interface and a router that support a communication network closely integrated with the processor. In a J-Machine, the network provides end-to-end message delivery with low latency (¡ 2 μ s in a 4K-Node network) and high bandwidth (288 MBits/sec per channel).

4.1 Structure

The J-Machine network is a 3-dimensional grid, with bidirectional channels. Each MDP has six bidirectional network ports for three main axis (+X, -X, +Y, -Y, +Z, -Z) and each port is connected to the opposite port of another MDP node (i.e. +X is connected to -X port of another MDP).

4.2 Engineering

The 3-dimensional mesh topology was chosen as the most efficient arrangement subject to constraints of wiring density and component pinout. These constraints set the width of the six bidirectional channels per MDP node at 9 data bits plus 6 control bits. The J-Machine is built as a stack of boards with dense board-to-board interconnections to implement the 3-D network with short wires.

4.3 Routing & Flow Control

The J-Machine uses deterministic dimension-order routing. All messages are first routed in X dimension, then in Y and then in Z. Since messages are routed in dimension order and messages running in opposite directions along the same direction do not block (due to bidirectional structure), the network is provably deadlock free.

The network also uses blocking flow control to resolve contention for a physical channel. When a message arrives at a router path that is already in use by a message of the same priority, it is blocked. The blocked message compresses into routers along its path, occupying one node per word of the message. When the blockage clears, the message uncompresses and proceeds to its destination, at a rate of one hop per cycle.

Chapter 5

Programming Model

The J-Machine is intended as a platform for software experiments in fine-grained parallel programming. Fine-grained programs typically execute 10-100 instructions between communication and synchronization actions. Two languages have been implemented to the J-Machine: The actor language Concurrent Smalltalk and the dataflow language Id.

5.1 Concurrent Smalltalk

Concurrent Smalltalk is a parallel object-oriented programming language with support to asynchronous message send and distributed objects. Functions and methods in the language are compiled into MDP assembly code by an optimizing compiler and uses a library called COSMOS at run-time. COSMOS provides the following features:

- Global virtual namespace
- Object-based memory management
- Support for distributed objects
- Low overhead context switching

COSMOS is designed to run efficiently in fine-grain concurrent computation where tasks are very short (40 user instructions) and data objects are very small (8 words). The Concurrent Smalltalk compiler and the COSMOS runtime system also provide floating-point arithmetic, simple arrays and garbage collection.

5.2 Dataflow Implementation

Id is a functional programming language originally designed for dataflow architectures. An Id program can be converted into a dataflow graph, in

which operations are represented with nodes and the dependencies are represented with arcs. Originally, these dataflow graphs were executed directly on specialized dataflow machines. More recently, researchers have begun compiling dataflow graphs to run on general-purpose parallel machines.

Dataflow programs are well-suited for large parallel computers, because since each of the fine-grained tasks can be as small as a single dataflow operator, makes it easy to mask communication latency with task switching.

Chapter 6

Conclusions & Comments

In this survey, The Message-Driven Processor is examined from both software and hardware perspectives. The MDP is a simple yet very powerful processing node for multi-computer applications. The MDP has two clear advantages:

- It provides instruction level support for communication, naming and synchronization.
- It has a network interface consisting of six network ports and has been designed solely for 3D-Mesh networking topology.

The main idea is using high number of cost effective processing nodes instead of a small number of highly complicated and expensive processing units. Therefore, the processor part of the MDP is not necessarily as fast as the high-end processors of the time but it has an advantage of being designed specially for message-based parallel programming.
Unlike traditional von Neumann processors, the processing in MDP is triggered on receiving messages from other nodes. For each received message, a task is dispatched.
As a personal opinion, everything proposed about the MDP is very simple and very reasonable. It is not acceptable and understandable that it could not become popular although it has been designed by M.I.T. and Intel which are two big shots.

Bibliography

[1] William J. Dally, J.A. Stuart Fiske, John S. Keen, Richard A. Lethin, Michael D. Noakes, Peter R. Nuth, Roy E. Davison, Gregory A. Fyler. "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms," *IEEE Micro*, vol. 12, no. 2, pp. 23-39, March/April, 1992.

[2] Michael D. Noakes, Deborah A. Wallach, and William J. Dally, "The J-Machine Multicomputer : An Architectural Evaluation," In Proceedings of the 20^{th} International Symposium on Computer Architecture, May 1993

[3] Daniel Maskit and Stephen Taylor, "A Message-Driven Programming System for Fine-Grain Multicomputers," *SoftwarePractice & Experience* , Vol. 24, Issue 10, pp. 953 - 980, 1994

Prepared in LaTeX 2_ε by RCJ