The Message Driven Processor: An Integrated Multicomputer Processing Element

William J. Dally, Andrew Chien, J.A. Stuart Fiske, *Greg Fyler, Waldemar Horwat, John S. Keen, Richard A. Lethin, Michael Noakes, Peter R. Nuth, and D. Scott Wills

Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139

*Platform Architecture Group, Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95052-8131

Abstract

The Message-Driven Processor (MDP) is an integrated multicomputer node. It incorporates a 36-bit integer processor, a memory management unit, a router for a 3-D mesh network, a network interface, a 4K-word x 36-bit SRAM, and an ECC DRAM controller on a single 1.1M transistor VLSI chip. The MDP is not specialized for a single model of computation. Instead, it incorporates efficient primitive mechanisms for communication, synchronization, and naming. These mechanisms support most proposed parallel programming models. Each processing node of the MIT J-Machine consists of an MDP with 1MByte of DRAM. MDPs have been operational since June 1991 and J-Machines built from them have been online since July 1991.

1 Introduction

The Message-Driven Processor (MDP) is a multicomputer node incorporating a 36-bit integer processor, a memory management unit, a 3-D router, a communications controller, a 4K x 36-bit SRAM, and a DRAM controller with ECC on a single 1.1M transistor chip. An MDP, with optional external DRAM, forms a processing node of the MIT J-Machine, a distributed-memory, MIMD, concurrent computer [6]. A J-Machine is a 3-D mesh of up to 65,536 message-driven processors (MDPs).

The MDP supports a broad range of parallel programming models (including shared-memory, data parallel, dataflow, and explicit message-passing) by providing low-overhead primitive mechanisms for communication, synchronization, and translation [5]. Communication mechanisms are provided that permit a user-level task on one node to send a message to any other node in a 4K-node machine, and dispatch a task in response to the message, all in < 3μs. Synchronization is provided by presence tags on all storage locations. Three separate register sets allow fast task switching. The translation mechanism maintains bindings between arbitrary names and values; it may be used to support a global virtual address space. These mechanisms have been selected to be both general and amenable to efficient hardware implementation.

The MDP builds on previous work on multicomputer design. Like the Caltech Cosmic Cube [14], the Intel iPSC [2], and the N-CUBE [13], each MDP in the J-Machine has a local memory and communicates with other nodes by passing messages. Like shared-memory multiprocessors [4, 1, 9], the J-Machine provides a global virtual address space. The same IDs (virtual addresses) are used to reference local (on the same node) and remote (on another node) objects. Like the Caltech MOSAIC [10], and the Intel iWARP [3], a J-Machine node is a single chip processing element integrating a processor, memory, and a communication unit.

2 Instruction Set Architecture

The MDP extends a conventional microprocessor instruction-set architecture (ISA) with instructions to support parallel processing. Specifically, the MDP
provides efficient hardware mechanisms for communication, synchronization and naming.

Register Sets: The MDP provides separate register sets to support rapid switching between three execution levels: background, priority 0 (P0), and priority 1 (P1). The MDP executes at the background level when there are no pending messages. Each arriving message creates a task and initiates execution at P0 or P1 depending on the priority of the message. The MDP executes the highest priority task at any point in time. Thus, the arrival of a P1 message while the MDP is executing a P0 task will cause the MDP to switch execution levels (and thus register sets).

The register set at each priority level includes:

- 4 general-purpose data registers: R0–R3
- 4 address registers: A0–A3
- 4 ID registers:
  - 1 instruction pointer: IP

Most instructions operate on the general registers R0–R3. Each address register A0–A3 contains a segment descriptor consisting of a base and a length field. Memory addresses are specified by an offset and an address register. For example, the operands [R0, A1] and [3, A2] specify an indexed access to the segment described by A1 and a displacement of 3 into A2’s segment, respectively. ID registers are used to hold object IDs. The instruction pointer includes process status bits that control virtual addressing, type checking, and fault handling. Placing these bits in the instruction pointer enables control and execution state to be changed by loading a single register. Quick task switching within an execution level is facilitated by the relatively small size of each register set.

Tags: Tags in the MDP are used for type checking and synchronization. Every 36-bit word of register and memory state holds a 32-bit value and a 4-bit tag that indicates the type of the value. Tag values are defined for primitive user data types (such as SYMBOL, INTEGER, BOOLEAN, etc.) and for system data types (such as IP, ADDR (a segment descriptor), MSG (a message header), etc.).

Two tags, FUT and CFUT, support inter-task synchronization. A location is initially marked empty by writing it with a CFUT tag. When a task produces the value for the location, it overwrites the CFUT with the final value and tag. Any attempt to read from the location before the value is produced invokes the CFUT fault handler which typically suspends the reading task until the location is written. CFUTs are synchronizing variables that cannot be copied, while FUTs may be copied.

Instructions: The MDP executes 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 17-bit instructions are packed in each 36-bit word. Any instruction-stream word not tagged as an instruction is loaded as a constant into register R0. This provides a very efficient means to load arbitrary 36-bit constants. The MDP instruction set includes the usual set of data movement instructions, arithmetic and logic instructions, and branch instructions. Special instructions are provided to support synchronization, naming, and task scheduling, some of which are described below.

Naming: is supported in the MDP via translation instructions and segmented addressing. Addressing memory through segment descriptors permits arbitrarily sized objects to be relocated and protected. The ENTER instruction enters an arbitrary translation from a 36-bit key to a 36-bit data value in a set-associative cache (translation table) mapped into the on-chip memory. The XLATE instruction looks up the data value (if any) associated with a key. These instructions may be used to translate virtual object-IDs into physical segment descriptors or node numbers to support a global virtual address space. A segmented memory management unit may be used with the translation mechanism to provide protected addressing.

Communication: To achieve low-overhead communication for fine-grain messages, the MDP provides hardware support for end-to-end message delivery. Message formatting, injection, delivery, buffer allocation, message buffering, and task scheduling are managed in hardware to reduce overhead. The SEND instructions allow the processor to directly inject messages into the network. The network is discussed in detail in a companion paper in this conference.

Task Scheduling: When a message reaches the head of the highest priority non-empty queue, a task is created to handle it by changing the thread of control and creating a new addressing environment. Every message header contains a message opcode which is loaded directly into the IP to start a new thread of control. The message itself is the initial addressing
environment for the task. Creating a task to handle a message takes only 3 cycles.

The dispatch mechanism is used directly to process messages requiring low latency (e.g., combining and forwarding). Other messages (e.g., remote procedure call) specify a handler that locates the required method (using the translation mechanism described above) and then transfers control to it.

Further details of the MDP ISA are described in [11, 7].

3 Implementation

The MDP was implemented conservatively to minimize risk. Among other things, this led to a simple microarchitecture, a conservative synchronous router, the extensive use of standard cells, and a generous non-overlap period for the two-phase clock. The major subsystems in the MDP are shown in Figure 1, and are described below.

The Control, Prefetch, and RALU together provide most of the functionality of a conventional processor with tags. The Control unit generates state sequences that control the operation of the remaining blocks, while the Prefetch unit fetches and decodes an instruction sequence. The RALU contains the general registers and ALU.

The Address Arithmetic Unit (AAU) generates all memory addresses for data read and write, instruction fetches, network enqueuing, and task dispatch. All memory management and message queue buffer management functions are performed in this block.

The Internal Memory is a 4K x 36-bit 6-transistor per cell SRAM. It includes two 144-bit wide input row buffers for enqueuing network data, one output row buffer for reading instructions, and comparators for implementing set associative access. This subsystem is actually two modules, the full-custom RAM array, and the standard-cell RAM interface. Having both the Prefetch and the Network Input units access memory 144-bits at a time reduces memory contention. The External Memory Interface generates timing signals, multiplexes addresses and data, and performs error detection and correction to interface standard DRAM components to the MDP.

The Network Input, Network Output, and Router modules are responsible for inter-processor communication. The network input and output modules handle messages coming in from and going out to the network, respectively, while the router handles routing and flow control to deliver messages across the network.

Finally the Diagnostic Port permits control of the processor (e.g., reset, halt, single step, run,...) and access to both the on-chip and off-chip RAMs. Commands, addresses and data are scanned in and out serially.

Most instructions take from one to three cycles to execute under hardwired control. Prefetch and decode of the next instruction are overlapped with the last cycle of an instruction's execution; however, instruction execution is not pipelined. The final chip has a transistor count of 1.1M, a die area of 1.5cm², and is currently operating at 16MHz, with the network operating at 32MHz.

System Design: The J-Machine processing board measures 20.5 by 26 inches and contains 64 processing nodes. Each node consists of an MDP chip (in a 168-pin PGA package) and three 4Mbit DRAMs. Elastomeric connectors are used to provide the interconnect between boards. Each pair of nodes shares a set of elastomeric connectors to communicate with the corresponding nodes on the boards above or below the board in a stack. The chassis contains a stack of 16 processor boards, power supplies, and distribution bus
bars. A 4096-node system can be built by combining four chassis.

In addition to the processor board and chassis, we have also designed a diagnostic interface board, a SCSI disk interface. We are in the process of designing a distributed graphics frame buffer, and an SBus interface. More details of the J-Machine system design are given in [12].

4 Conclusion

The MDP has been built to demonstrate the utility of general-purpose communication and synchronization mechanisms in a multicomputer building block. Its mechanisms have been shown to efficiently support dataflow [15] and object-oriented programming [6] models using a shared address space. Our studies have shown that the use of a few simple mechanisms provides orders of magnitude lower communication and synchronization overhead than is possible with multicomputers built from off-the-shelf microprocessors.

Acknowledgment

The research described in this paper was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-88-K-0738 and N00014-91-J-1698, in part by a National Science Foundation Presidential Young Investigator Award, grant MIP-8657531, with matching funds from General Electric Corporation, IBM Corporation and AT&T Corporation, and in part by assistance from Intel Corporation. Many members of the MIT Concurrent VLSI Architecture Group, in particular Scott Furman, Shaun Kaneshiro, Ellen Spertus, and Deborah Wallach, contributed to the design and programming of the J-Machine. Discussions with Chuck Seitz, Steve Ward, Anant Agarwal, and Tom Knight were helpful during the early stages of the J-Machine project.

References


