The MasPar MP-1 Architecture

Tom Blank

MasPar Computer Corporation
Sunnyvale, CA

Abstract

This article describes the MasPar MP-1 architecture, a massively parallel SIMD (Single Instruction Multiple Data) machine with the following key characteristics: scalable architecture in terms of the number of processing elements, system memory, and system communication bandwidth; "RISC-like" instruction set design which leverages optimizing compiler technology; adherence to industry standard floating point formats, specifically VAX^{TM} and IEEE floating point; and an architectural design amenable to a VLSI implementation. The architecture provides not only high computational capability, but also a mesh and global interconnect style of communication.

The techniques and subsystems of the MP-1 are described including the interconnection mechanisms. Companion papers describe the software system and provide a description of the hardware implementation.

1 Introduction

MasPar Computer Corporation has designed and implemented a high performance, low-cost, massively parallel computing system called the MP-1. The system works in a SIMD (Single Instruction Multiple Data) fashion. Previous machines with similar characteristics are the MPP[1], DAP[4], Blitzen[2], CM[3], DEC MPP[6], and the VBMP[5]. Unique characteristics of the MP-1 architecture are the combination of: a scalable architecture in terms of the number of processing elements, system memory, and system communication bandwidth; "RISC-like" instruction set design that leverages optimizing compiler technology; adherence to industry standard floating point design, specifically VAX and IEEE floating point; and an architectural design amenable to a VLSI implementation.

Figure 1 shows a block diagram of the MasPar system with five major subsystems. The following briefly describes each of the major components with a more detailed description later in the paper:

The Array Control Unit (ACU) The ACU performs two primary functions: either PE Array control or independent program execution. The ACU controls the PE Array by broadcasting all PE instructions. Independent program execution is possible since it is a full control processor capable of independent program execution.

The Processor Element Array (PE Array) The PE Array is the computational core of the machine. All instruction dispatch to the PE Array is from the ACU.

Communication Mechanisms The communication mechanisms provide the following key capabilities:

- The X network for communication with neighboring processors. All connections are on a 2-D mesh.
- The global router network permits random processor-to-processor communication using a circuit-switched, hierarchical crossbar communications network.
- Two global busses: a common bus on which the ACU broadcasts instructions and data to all or selected processors, and a logical OR-tree which consolidates status responses from all the processors back to the ACU.

The $UNIX^{R}$ Subsystem (USS) Provides UNIX services to the data parallel system. For example, all job management and low speed network access (e.g. ethernet) is performed by the USS.

The I/O Subsystem Supports high speed I/O performance. A channel style architecture is used allowing overlapped computation and I/O operations.

2 Machine Computational Model

Based on the previous architecture block diagram, the system can be accurately viewed as having two instruction streams, the UNIX Subsystem (USS) and the ACU, and three locations for data: the USS, the ACU, and the PE Array. In the SIMD fashion, all PE instructions reside in the ACU instruction memory.

Since two instruction streams are required for the system, two basic programming approaches are possible and are both are supported:

- One application code is automatically distributed across the USS and the ACU with the data partitioned across the USS, ACU, and PE Array. All interprocess communication is automatically handled by the compiler.
- Two application codes are provided, one for the USS, and one for the ACU/PE Array where all communication between the two processes is explicitly controlled by the programmer.

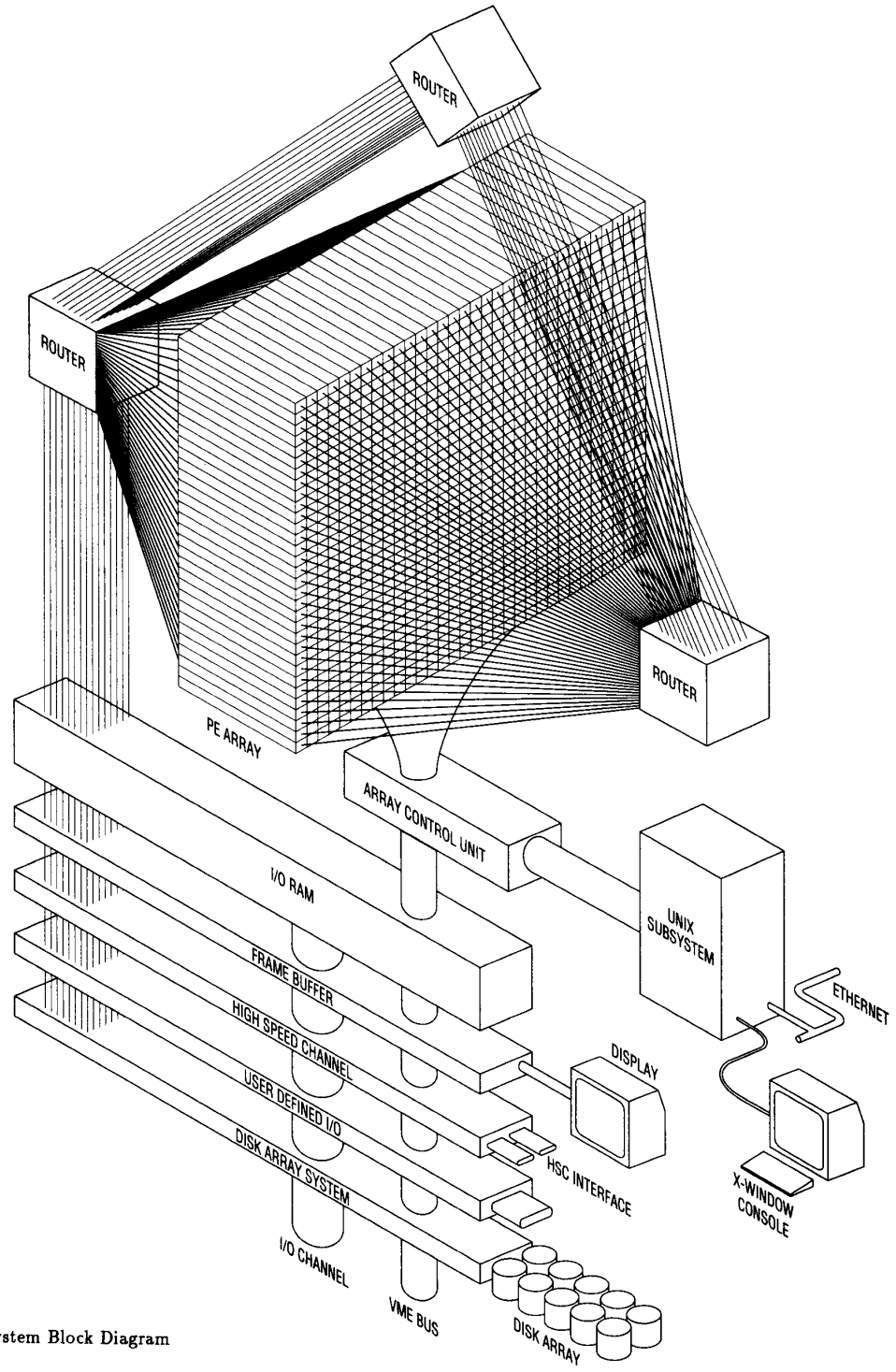

Figure 1: MP-1 System Block Diagram

Common to both programming approaches are two different interaction models: synchronous and asynchronous both with architectural and software support. In the synchronous model, either the USS or the ACU/PE Array is actively running at one instant. Similar to UNIX remote procedure calls (RPC), a subroutine calling convention allows straight forward control flow transfer between the two hardware processes.

In contrast, the asynchronous model allows both the USS and ACU/PE Array to operate concurrently. Support for a FORK/JOIN model are provided.

3 MP-1 Architecture

This section describes in more detail the basic architectural subsystems including the basic instruction set model.

3.1 Array Control Unit (ACU)

The ACU, a custom processor, both executes instructions that cause computation in the PE Array and executes instructions that cause computation only in the ACU itself. The following list describes the major architectural characteristics:

- Harvard style architecture with separate instruction and data spaces.
- 32-bit, two address, load/store, simple instruction set
- 4 Gigabyte, virtual, instruction address space, using 4,096-byte pages.

Table 1 shows the basic ACU instruction types where each instruction uses one instruction word with a two address three operand style format: *src op dst* $\rightarrow$ *dst*. In this load/store style machine, all operations are only within the register set with only load and store operations into memory. Instructions typically execute in one or two clocks.

Instruction Types	Examples
Memory:	`Load, Store`
Logical:	`AND, OR, XOR`
Arithmetic:	`ADD, SUB`
Control:	`Branch, Jump to subroutine (JSR)`

Table 1: ACU Instruction Set

The ACU has a microcoded implementation of this RISC like instruction set due to the additional control requirements of the PE Array. In the next section describing the PE Array, PE instructions typically require more than one clock including floating point instructions which are well suited to a microcode implementation.

3.2 Processor Array

The processor array is the computational core. Each PE has on-chip registers, and off-chip memory using a basic load/store style instruction set design. During a computation, all PEs execute the same instruction stream (which is broadcast by the ACU), unless they have been programmed to idle.

The basic PE components follow:

Integer and Floating Point ALU Both the integer and floating point unit share the computational PE core. Floating point hardware is included for both 32 and 64 bit floating point numbers capable of VAX D, F, and G formats; and IEEE standard floating point. Further, both big and little endian conventions are supported. All PE calculations are done in a scalar fashion without pipeline latency.

Communications Interface Three interfaces are provided: glo-bal router connections, nearest neighbor connections, and connections to global ACU signals. Section 3.4 contains further details.

Register Set In contrast to typical processor architectures, the PE register set can be addressed as bits, bytes, 16-bit words, 32-bit words, or 64-bit words depending on the PE instruction used. The current implementation has 40 32-bit registers. Both floating point and integer values are stored in the register set.

Main Memory Each PE has a private data store with full ECC (remember that only data is stored in the PEs; all instructions are stored in the ACU).

Control Logic Minimal control logic is required in each PE since the majority of the instruction decode logic is in the ACU and shared by all PEs. The control unit performs two primary functions: simple decode of ACU broadcast microinstructions, and conditional instruction execution. Conditional instruction execution allows individual processors to decide based on internal data whether it should execute the current instruction.

The PE instruction set is nearly identical to the ACU in that all instructions are two address, three operand instructions using a load/store model. All execution instructions (e.g. `add`, `sub`, etc.) operate only out of the register set and only `load` and `store` operations access memory. The following table contains the basic PE instruction types and examples:

Instruction Types	Examples
Memory:	`LD, ST, LDX, STX`
Logical:	`AND, OR, XOR`
Integer:	`ADD, SUB, MUL, DIV`
Floating Point:	`FADD, FSUB, FSQRT`
Control:	`Turn PEs on/off`

Table 2: PE Instruction Set

Different instructions are provided for both single (32-bit) and double (64-bit) precision floating point numbers. For integers, different instructions are provided for 1, 8, 16, 32 and 64 bit calculations designed specifically to support high level compiled languages like Fortran and C (a more detailed discussion of the compilers are provided in a companion paper).

Two very important instructions are `LDX` (load indirect) and `STX` (store indirect) which allow PEs to simultaneously access different memory locations. This capability allows important data structures like queues and look-up tables to be used.

3.3 UNIX Subsystem (USS)

An important aspect of the system is the use of an existing computer system (specifically a VAXstation 3520 $ULTRIX^{TM}$ workstation) that follows existing industry standards (e.g. X windows, TCPIP, etc.). The USS provides a complete, network and graphics based, software environment in which all the MasPar tools and utilities (e.g. compilers) execute. Part of the application executes as a conventional workstation application; most of the "operating system" functions are provided by the workstation's UNIX software.

3.4 Communication Mechanisms

The following sections describe the five major communications mechanisms. Included are descriptions of the programming model and instructions.

3.4.1 USS to ACU

Three different types of interactions occur between the UNIX Subsystem (USS) and the Array Control Unit (ACU) which use three different types of hardware support. All are based on a standard bus interface (VME). The following describes each mechanism:

Queues Hardware queues are provided which allows USS processes to quickly interact with the process running on the ACU. The programming model is similar to UNIX pipes but with hardware assist.

Shared Memory The shared memory mechanism overlaps ACU memory addresses with USS memory addresses. This provides a straight forward mechanism for processes to share common data structures like file control blocks etc.

DMA A DMA mechanism is provided that permits fast bulk data transfers without using programmed I/O.

3.4.2 ACU to PE Array

Two basic capabilities are required for data movement between the ACU and PE Array: data distribution, `DIST`, and array consensus detection which uses a global OR, `GOR`. An example usage:

```
while (array_value > error_limit)
    array_value = find_better_value();
```

In words, each PE gets a copy of the common `error_limit` value and compares it to a PE specific data value. Then, all PEs put the logical result of the expression evaluation onto an OR tree allowing the ACU to decide if any PEs need to go through the loop again.

3.4.3 PE Array: XNet

XNet communications provide all PEs with a direct connection to its eight nearest neighbors in a two dimensional mesh. Specifically, each PE is connected to its neighbors to the: North, Northeast, East, Southeast, South, Southwest, West, and Northwest. Processors located on the physical edge of the array have toroidal wrapped edge connections.

Three basic instruction types are provided to use the nearest neighbor connections:

`XNET` The XNET instruction moves an operand from source to destination a specified distance in all active PEs. The instruction time is proportional to the distance times the operand size since all communication is done using single wire connections.

`XNETP` The XNETP instruction is pipelined so that a collection of PEs move an operand from source to destination over a specified distance. However, the pattern of active and inactive PEs is very important since active PEs transmit data and inactive PEs act as pipeline stages. The instruction time is proportional to the distance plus the operand size due to its pipelined nature. For example if every 16th PE in a row is active, the XNETP instruction could move data between the active PEs providing a very high performance non-blocking communication mechanism. This mechanism is similar to the ideas proposed in [7].

`XNETC` The XNETC instruction is pipelined and is very similar to the XNETP instruction except that a copy of the operand is left in all PEs acting as pipeline stages (e.g. the inactive PEs). Again, the instruction time is proportional to the distance plus the operand size.

3.4.4 PE Array: Global Router

The global router is a circuit switched style network organized as a three stage hierarchy of crossbar switches. This mechanism provides direct point to point bidirectional communications. The network diameter is 1/16 the number of PEs which requires a minimum of 16 communication cycles to do a permutation with all PEs. The basic instruction primatives are:

`ropen` open a connection to a destination PE

`rsend` move data from the originator PE to the destination PE

`rfetch` move data from the destination PE to the originator PE

`rclose` terminate the connection

The best analogy for using this network is the telephone system where people who want to make a call use the following steps:

1. People who want to make a call pick up their phone
2. Dial a phone number
3. If busy, hangup and try again later (go back to step one)
4. If connection completes, have a nice conversation
5. When call completes, hangup

The usage sequence for the MasPar router is as follows:

```
while (PEs_want_to_communicate) {
    ropen
    rsend
    rfetch
    rsend
```

```
    .
    .
    .
rclose

}
```

3.4.5 PE Array to I/O Subsystem

Since the global router provides high performance random PE to PE communication, the global router is also used to provide a high performance communication mechanism into the I/O subsystem. The interface is achieved by connecting the last stage of the global router to an I/O device, the I/O RAM (described in section 3.5). The programming model is identical to the model described for using the global router in section 3.4.4.

3.5 Array I/O System

Referring back to figure 1, the I/O subsystem uses the following key components: the global router connection into the PE Array (over 1 GB/sec), a large I/O RAM buffer (up to 256MB), and a high speed (230MB/sec) data communications channel between peripheral devices, a bus for device control (not for data movement). Using output as an example, the model for using the I/O subsystem follows these steps:

1. Device is opened by the USS (all I/O devices are UNIX controlled)
2. The ACU moves data into the I/O RAM through the global router.
3. Either the USS or an I/O Processor (IOP) schedules data movement from the I/O RAM to the device (e.g. Disk); data through the MPIOC and control on the VME bus.
4. The USS is notified when the transaction is complete.

Note that all transactions from the I/O RAM to external I/O systems can occur asynchronously from PE Array operations. This is a key attribute since data can move into the I/O RAM at speeds over 1 GB/sec then move at I/O device speeds, typically in the tens of megabytes per second or less, without effecting the performance of the PE Array. These hardware mechanisms can support either typical synchronous UNIX I/O or newer (and faster) asynchronous I/O software models.

4 Summary

A key attribute of the MP-1 system architecture is that the system characteristics all are scalable. Specifically, as the performance increases (more PE boards are added), the system memory increases, and the communications bandwidth increases. Each PE board increase the system capability while keeping performance, communication, and memory balanced. System "bottlenecks" are not introduced as the number of processors are increased.

The architectural subsystems have been designed so that the various computational tasks are distributed to specialized units. Examples include: the ACU is specialized for controlling the PE Array, the PE is optimized for both floating point and integer calculations. Further, hardware software tradeoffs have been made that leverage existing software technology. Key examples are both the ACU and PE instruction sets that closely resemble current RISC style instruction sets. The advantage in following this instruction set design is that complexity is moved out of the hardware design and out of the microcode design and into the compiler. Less complex hardware allows both a faster and less expensive design. Further advantages of moving the complexity into the compiler leverages optimizing compiler technology with the tremendous advantage of optimizing data placement, register allocation, and eliminating unnecessary work.

References

[1] K.E. Batcher, "Design of a Massively Parallel Processor", IEEE Trans. on Computer, Sept 1980, pp. 836-840.

[2] E. Davis, J. Reif, "The Architecture and Operation of the BLITZEN Processing Element", 3rd Intl. Conf. on Supercomputing, May 1988.

[3] W.D. Hillis, *The Connection Machine*, MIT Press, 1985.

[4] S.F Reddaway, "DAP A Distributed Array Processor", First Annual Symposium on Computer Architecture, (IEEE/ACM), Florida, 1973.

[5] William T. Blank, A Bit Map Architecture and Algorithms for Design Automation, PhD thesis, Stanford University, September 1982.

[6] R. Grondalski, "A VLSI Chip Set for a Massively Parallel Architecture", International Solid State Circuits Conference, February 1987.

[7] C.M. FiDuccia, R. M Mattheyses and R.E. Stearns, "Efficient Scan Operators for Bit-Serial Processor Arrays", Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation, October 1988.