MasPar

The MasPar MP-1 Architecture

Tom Blank

MasPar Computer Corporation

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.

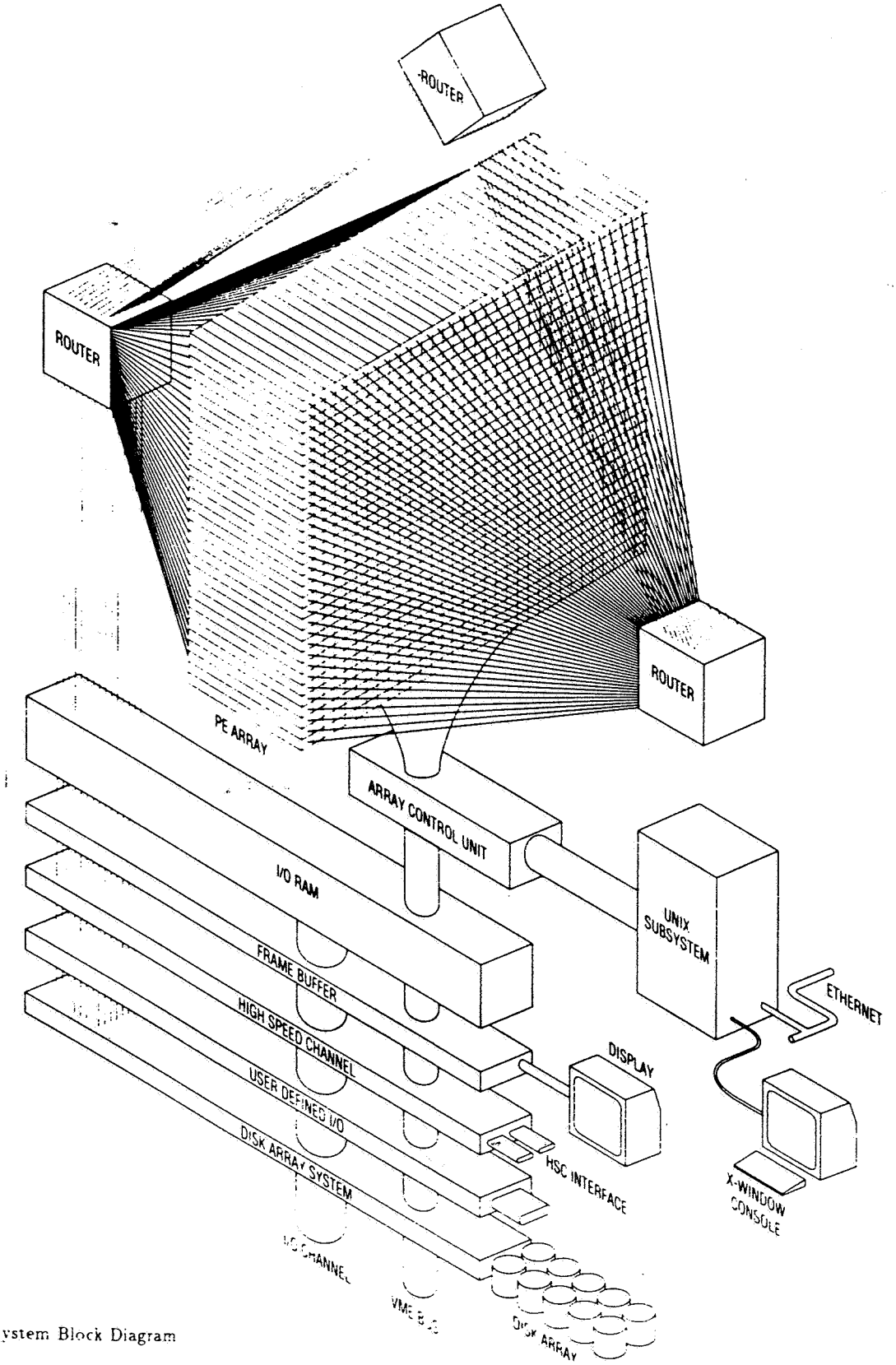


Figure 1: MP-1 System Block Diagram

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* → *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, TCP/IP, 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.

The Design of the MasPar MP-1: A Cost Effective Massively Parallel Computer

John R. Nickolls

MasPar Computer Corporation
749 North Mary Avenue
Sunnyvale, CA 94086

Abstract

By using CMOS VLSI and replication of components effectively, massively parallel computers can achieve extraordinary performance at low cost. Key issues are how the processor and memory are partitioned and replicated, and how interprocessor communication and I/O are accomplished. This paper describes the design and implementation of the MasPar MP-1, a general purpose massively parallel computer system that achieves peak computation rates beyond a billion floating point operations per second, yet is priced like a minicomputer.

Massively Parallel System

Massively parallel computers use more than 1,000 processors to obtain computational performance unachievable by conventional processors [1,2,3]. The MasPar MP-1 system is scalable from 1,024 to 16,384 processors and its peak performance scales linearly with the number of processors. A 16K processor system delivers 30,000 MIPS peak performance where a representative instruction is a 32-bit integer add. In terms of peak floating point performance, the 16K processor system delivers 1,500 MFLOPS single precision (32-bit) and 650 MFLOPS double precision (64-bit), using the average of add and multiply times.

To effectively apply a high degree of parallelism to a single application, the problem data is spread across the processors. Each processor computes on behalf of one or a few data elements in the problem. This approach is called "data-level parallel" [4] and is effective for a broad range of compute-intensive applications.

Partitioning the computational effort is the key to high performance, and the simplest and most scalable method is data parallelism. The architecture of the MP-1 [5] is scalable in a way that permits its computational power to be increased along two axes: the performance of each processor, and the number of processors. This flexibility is well matched to VLSI technology where circuit densities continue to increase at a rapid rate. The scalable nature of massively parallel systems protects the customers' software investment while providing a path to increasing performance in successive products [6].

Because its architecture provides tremendous leverage, the MP-1 implementation is conservative in terms of circuit complexity, design rules, IC geometry, clock rates, margins, and power dissipation. A sufficiently high processor count reduces the need to have an overly aggressive (and thus expensive) implementation. Partitioning and replication make it possible to use low cost, low power workstation technology to build very high performance systems. Replication of key system elements happily enables both high performance and low cost.

Array Control Unit

Because massively parallel systems focus on data parallelism, all the processors can execute the same instruction stream. The MP-1 has a single instruction stream multiple data (SIMD) architecture that simplifies the highly replicated processors by eliminating their instruction logic and instruction memory, and thus saves millions of gates and hundreds of megabytes of memory in the overall system. The processors in a SIMD system are called processor elements (PEs) to indicate that they contain only the data path of a processor.

The MP-1 array control unit (ACU) is a 14 MIPS scalar processor with a RISC-style instruction set and a demand-paged instruction memory. The ACU fetches and decodes MP-1 instructions, computes addresses and scalar data values, issues control signals to the PE array, and monitors the status of the PE array. The ACU is implemented with a microcoded engine to accommodate the needs of the PE array, but most of the scalar ACU instructions execute in one 70 nsec clock. The ACU occupies one printed circuit board.

Processor Array

The MP-1 processor array (figure 1) is configurable from 1 to 16 identical processor boards. Each processor board has 1,024 processor elements (PEs) and associated memory arranged as 64 PE clusters (PECs) of 16 PEs per cluster. The processors are interconnected via the X-Net neighborhood mesh and the global multistage crossbar router network.

The processor boards are approximately 14" by 19" and use a high density connector to mate with a common backplane. A processor board dissipates less than 50 watts; a full 16K PE array and ACU dissipate less than 1,000 watts.

A PE cluster (figure 2) is composed of 16 PEs and 16 processor memories (PMEM). The PEs are logically arranged as a 4 by 4 array for the X-Net two-dimensional mesh interconnection. Each PE has a large internal register file shown in the figure as PREG. Load and store instructions move data between PREG and PMEM. The ACU broadcasts instructions and data to all PE clusters and the PEs all contribute to an inclusive-OR reduction

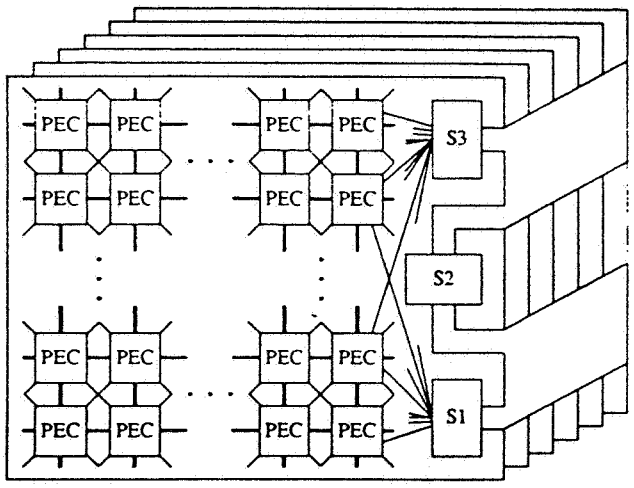


Figure 1. Array of PE Clusters

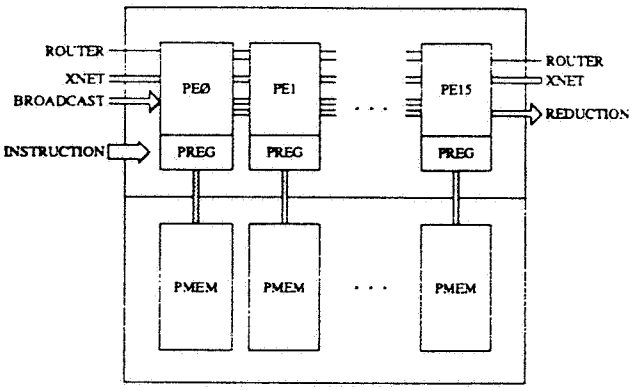


Figure 2. PE Cluster

tree received by the ACU. The 16 PEs in a cluster share an access port to the multistage crossbar router.

The MP-1 processor chip is a full-custom design that contains 32 identical PEs (2 PE clusters) implemented in two-level metal 1.6μ CMOS and packaged in a cost effective 164 pin plastic quad flat pack. The die is 11.6 mm by 9.5 mm, and has 450,000 transistors. A conservative 70 nsec clock yields low power and robust timing margins.

Processor memory, PMEM, is implemented with 1 Mbit DRAMs that are arranged in the cluster so that each PE has 16 Kbytes of ECC-protected data memory. A processor board has 16 Mbytes of memory, and a 16 board system has 256 Mbytes of memory. The MP-1 instruction set supports 32 bits of PE number and 32 bits of memory addressing per PE, so the memory system size is limited only by cost and market considerations.

As an MP-1 system is expanded, each increment adds PEs, memory, and communications resources, so the system always maintains a balance between processor performance, memory size and bandwidth, and communications and I/O bandwidth.

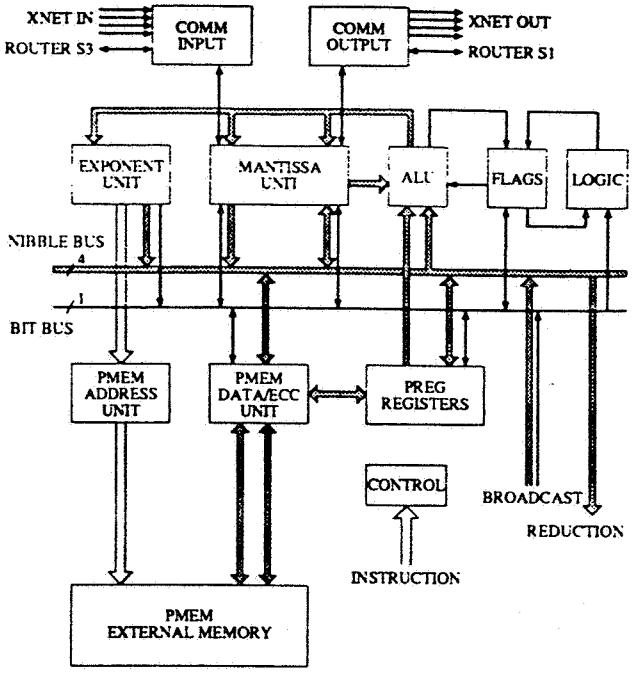


Figure 3. Processor Element and Processor Memory

Processor Elements

The MP-1 processor element (PE) design is different than that of a conventional processor because a PE is mostly data path logic and has no instruction fetch or decode logic. SIMD system performance is the product of the number of PEs and the speed of each PE, so the performance of a single PE is not as important as it is in conventional processors. Present VLSI densities and the relative tradeoffs between the number of processors and processor complexity encourage putting many PEs on one chip. The resulting design tradeoff between PE area and PE performance tends to reduce the PE architecture to the key essentials.

Each PE (figure 3) is designed to deliver high performance floating point and integer computation together with high memory bandwidth and communications bandwidth, yet have minimal complexity and silicon area to make it feasible to replicate many PEs on a single high-yield chip.

Like present RISC processors, each PE has a large on-chip register set (PREG) and all computations operate on the registers. Load and store instructions move data between the external memory (PMEM) and the register set. The register architecture substantially improves performance by reducing the need to reference external memory. The compilers optimize register usage to minimize load/store memory traffic.

Each PE has 40 32-bit registers available to the programmer and an additional 8 32-bit registers that are used internally to implement the MP-1 instruction set. With 32 PEs per die, the resulting 48 Kbits of register occupy about 30% of the die area, but represent 75% of the transistor count. Placing the registers on-chip yields an aggregate PE/PREG bandwidth of 117 gigabytes per second with 16K PEs. The registers are bit and byte addressable.

Each PE provides floating point operations on 32 and 64 bit IEEE or VAX format operands and integer operations on 1, 8, 16, 32, and 64 bit operands. The PE floating point/integer hardware has a 64-bit MANTISSA unit, a 16-bit EXPONENT unit, a 4-bit ALU, a 1-bit LOGIC unit, and a FLAGS unit; these units perform floating point, integer, and boolean computations. The floating point/integer unit uses more than half of the PE silicon area but provides substantially better performance than the bit-serial designs used in earlier massively parallel processors.

Most data movement within each PE occurs on the internal PE 4-bit NIBBLE BUS and the BIT BUS (figure 3). During a 32-bit or 64-bit floating point or integer instruction, the ACU microcode engine steps the PEs through a series of operations on successive 4-bit nibbles to generate the full precision result. For example, a 32-bit integer add requires 8 clocks: during each clock a nibble is fetched from a PREG register, a nibble is simultaneously obtained from the MANTISSA unit, the nibbles are added in the ALU, and the sum is delivered to the MANTISSA unit. At the same time, the ALU delivers a carry bit to the FLAGS unit to be returned to the ALU on the next step. The ALU also updates bits in the FLAGS unit that indicate overflow and zeroness.

The different functional units within the PE can be simultaneously active during each micro-step. For example, floating point normalization and de-normalization steps use the EXPONENT, MANTISSA, ALU, FLAGS, and LOGIC units together. The ACU issues the same micro-controls to all PEs, but the operation of each PE is locally enabled by the E-bit in its FLAGS unit. During a floating point operation, some micro-steps are data-dependent, so the PEs locally disable themselves as needed by the EXPONENT and MANTISSA units.

Because the MP-1 instruction set focuses on conventional operand sizes of 8, 16, 32, and 64 bits, MasPar can implement subsequent PEs with smaller or larger ALU widths without changing the programmer's instruction model. The internal 4-bit nature of the PE is not visible to the programmer, but does make the PE flexible enough to accommodate different front-end workstation data formats. The PE hardware supports both little-endian and big-endian format integers, VAX floating point F, D, and G format, and IEEE single and double precision floating point formats.

Along with the PE controls, the ACU broadcasts 4 bits of data per clock onto every PE nibble bus to support MP-1 instructions with scalar source operands. The PE nibble and bit bus also drive a 4-bit wide inclusive-OR reduction tree that returns to the ACU. Using the OR tree, the ACU can assemble a 32-bit scalar value from the OR of 16,384 32-bit PREG values in 8 clocks plus a few clocks of pipeline overhead.

Processor Memory

Because only load and store instructions access PMEM processor memory, the MP-1 overlaps memory operations with PE computation. When a load or store instruction is fetched, the ACU queues the operation to a separate state machine that operates independently of the normal instruction stream. Up to 32 load/store instructions can be queued and executed while PE computations proceed, as long as the PREG register being loaded or stored is not used by the PE in a conflicting way. A hardware interlock mechanism in the ACU prevents PE operations from using a PREG register before it is loaded and from changing a PREG register before it is stored. The optimizing compilers move loads earlier in the instruction stream and delay using registers that are being stored. The 40 registers in each PE assist the compilers in obtaining substantial memory/execution overlap.

The PMEM processor memory can be directly or indirectly addressed. Direct addressing uses an address broadcast from the ACU, so the address is the same in each PE. Using fast page mode DRAMS, a 16K PE system delivers memory bandwidth of over 12 gigabytes per second. Indirect addressing uses an address computed locally in each PE's PMEM ADDRESS UNIT and is a major improvement over earlier SIMD architectures[7] because it permits the use of pointers, linked lists, and data structures in a large processor memory. Indirect addressing is about one third as fast as direct addressing.

X-Net Mesh Interconnect

The X-Net interconnect directly connects each PE with its 8 nearest neighbors in a two-dimensional mesh. Each PE has 4 connections at its diagonal corners, forming an X pattern similar to the Blitzen[8] X grid network. A tri-state node at each X intersection permits communications with any of 8 neighbors using only 4 wires per PE.

Figure 1 shows the X-Net connections between PE clusters. The PE chip has two clusters of 4 by 4 PEs and uses 24 pins for X-Net connections. The cluster, chip, and board boundaries are not visible and the connections at the PE array edges are wrapped around to form a torus. The torus facilitates several important matrix algorithms and can emulate a one-dimensional ring with two X-Net steps.

All PEs have the same direction controls so that, for example, every PE sends an operand to the North and simultaneously receives an operand from the South. The X-Net uses a bit-serial implementation to minimize pin and wire costs and is clocked synchronously with the PEs; all transmissions are parity checked. The PEs use the shift capability of the MANTISSA unit to generate and accumulate bit-serial messages. Inactive PEs can serve as pipeline stages to expedite long distance communication jumps through several PEs. The MP-1 instruction set [5] implements X-Net operations that move or distribute 1, 8, 16, 32, and 64 bit operands with time proportional to either the product or the sum of the operand length and the distance. The aggregate X-Net communication rate in a 16K PE system exceeds 20 gigabytes per second.

Multistage Crossbar Interconnect

The multistage crossbar interconnection network provides global communication between all the PEs and forms the basis for the MP-1 I/O system. The MP-1 network uses three router stages shown as S1, S2, and S3 in figure 1 to implement the function of a 1024 by 1024 crossbar switch. Each cluster of 16 PEs shares an originating port connected to router stage S1 and a target port connected to stage S3. Connections are established from an originating PE through stages S1, S2, S3, and then to the target PE. A 16K PE system has 1024 PE clusters, so each stage has 1024 router ports and the router supports up to 1024 simultaneous connections.

Originating PEs compute the number of a target PE and transmit it to the router S1 port. Each router stage selects a connection to the next stage based on the target PE number. Once established, the connection is bidirectional and can move data between the originating and target PEs. When the connection is closed, the target PE returns an acknowledgement. Because the router ports

are multiplexed among 16 PEs, an arbitrary communication pattern takes 16 or more router cycles to complete.

The multistage crossbar is well matched to the SIMD architecture because all communication paths are equal length, and therefore all communications arrive at their targets simultaneously. The router connections are bit-serial and are clocked synchronously with the PE clock; all transmissions are parity checked. The PEs use the MANTISSA unit to simultaneously generate outgoing router data and assemble incoming router data.

The MP-1 router chip implements part of one router stage. The router chip connects 64 input ports to 64 output ports by partially decoding the target PE addresses [9]. The full-custom design is implemented in two-level metal 1.6μ CMOS and packaged in a 164 pin plastic quad flat pack. The die is 7.7 mm by 8.1 mm, and has 110,000 transistors. Three router chips are used on each processor board.

A 16K PE system has an aggregate router communication bandwidth in excess of 1.5 gigabytes per second. For random communication patterns the multistage router network is essentially equivalent to a 1024 by 1024 crossbar network with far fewer switches and wires.

Conclusion

Through a combination of massively parallel architecture, design simplicity, cell replication, CMOS VLSI, conservative clock rates, surface mount packaging, and volume component replication, the MasPar MP-1 family delivers very high performance with low power and low cost. The massively parallel design provides cost effective computing for today and a scalable growth path for tomorrow.

References

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

[2] Kenneth E. Batcher, "Design of a Massively Parallel Processor", *IEEE Transactions on Computers*, vol C-29, pp. 836-840, Sept 1980.

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

[4] David L. Waltz, "Applications of the Connection Machine", *Computer*, pp. 85-97, January 1987.

[5] Tom Blank, "The MasPar MP-1 Architecture", *Proceedings of IEEE Compcon Spring 1990*, IEEE, February 1990.

[6] Peter Christy, "Software to Support Massively Parallel Computing on the MasPar MP-1", *Proceedings of IEEE Compcon Spring 1990*, IEEE, February 1990.

[7] Ken Batcher, "The Architecture of Tomorrow's Massively Parallel Computer", *Frontiers of Massively Parallel Scientific Computing*, NASA CP 2478, September 1986.

[8] Edward W. Davis and John H. Reif, "Architecture and Operation of the BLITZEN Processing Element", *3rd Intl. Conf. on Supercomputing*, vol III, pp. 128-137, May 1988.

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

Software To Support Massively Parallel Computing on the MasPar MP-1

Peter Christy

MasPar Computer Corporation
Sunnyvale, CA

Abstract

Creating a commercially successful, massively parallel, minicomputer requires careful attention to the problems of application design and programming. The software product set was a key design focus for the MasPar computer from the beginning. Key issues discussed are the following:

- The Programming Model
- Software Philosophy
- Parallel Virtuality
- High-Level Languages and Their Compilers
- Application Porting and Adaptation
- Programming Support

The Programming Model

Arbitrary existing programs will not necessarily run effectively on a massively parallel machine. For scientific and engineering applications, a reasonable analogy for a SIMD massively parallel machine is a vector register machine with the register length equal to the size of the processor array (for the MP-1 family [1] [2], 1,024 to 16,384 processors; much larger than typical vector registers). Most existing codes have short average vector lengths -- of the order of 25. Therefore, perfect vectorization of such code results in utilizing a few percent of the parallel array at best. From the outset, we have assumed that most existing application code must be modified so that a high-degree of parallelism is expressed. We did not attempt to create tools that would automatically discover massively parallelism in arbitrary preexisting applications (the proverbial "dusty decks").

Such a reprogramming constraint would historically be a death warrant for a new product. In the case of the MP-1, we believe the following facts justify the reprogramming for many applications:

- Diminishing performance improvement in conventional machines forces all high-performance application designers to consider new architectural alternatives. There is growing belief that some form of massive parallelism is the best approach to continuing performance and price/performance improvements. The issue of expressed use of parallelism is common to all these architectures, not just the MP-1.
- A VLSI massively parallel machine offers extraordinary price/performance improvements today. The peak power of the MP-1 is up to 100-1000 times that of a conventional machine of equivalent price. We feel confident that many applications will achieve minimally a factor of 10 improved price performance after all the application conversion and reprogramming is done. A factor of 10 in price performance justifies a significant adaptation effort in most applications.
- We have developed effective tools for porting existing applications with minimal modifications (these are discussed in more detail later). To adapt an application for massively parallel execution, focus must be placed on the computation kernel of the application, but often the majority of the existing code can be run intact.
- For Fortran applications, we provide a compiler for a subset of Fortran 88 in which the parallelization is conveniently expressed in the array notation provided in that language. An application can be rewritten in Fortran 88 and run across a broad variety of computers, including traditional scalar and vector machines, as Fortran 88 compilers become broadly available.
- Finally, and most importantly, broad parallelism is inherent in most of the applications currently running on high-performance computers. Creating programs that take advantage of a massively parallel computer means taking advantage of the natural parallelism in the problem, which usually is in the form of uniform calculations on a multidimensional mesh of data.

MasPar and other massively parallel computer vendors are using the term "data parallel" [3] to connote the general concept of utilizing the parallelism inherent in the problem data to take advantage of many parallel processors. The important contrast is with "control" parallelism, in which a program is analyzed, either manually or by compiler tools, and independent sections of the program are run concurrently on multiple processors. Supercomputer applications typically exhibit limited control parallelism, whereas they use data meshes that are much bigger than even the largest parallel arrays.

Software Philosophy

Historically, many new computers have made their debut in the form of fast hardware with minimal software. As desirable as that may be for a new computer company, in today's software intensive world, such a raw computer is unaffordable because of the cost of programming.

A better model today is that of the RISC computer developers and their use of the UNIX® operating system. A new RISC processor with an effective C compiler can quickly port a version of UNIX to new hardware, and in turn quickly deliver the numerous software capabilities available in a portable UNIX form.

MasPar's view of computer design has been similar to that of the RISC providers in the sense that our strategy has focussed on accelerating the widespread use of the machine. This resulted in the following product goals:

- Utilize compiler technology effectively to mask architectural issues
- Utilize existing languages
- Support a standard operating system
- Assure a rich set of support software as soon as possible
- Integrate into existing computer networks

In the MasPar architecture, the software focus is most evident in the design of the Array Control Unit which represents a significant architectural advance in the design and use of massively parallel computers. The most successful massively parallel machine to date – the Thinking Machines Corporation Connection Machine™ [3] – drives the parallel array with a microprogrammed sequencer, which is in turn driven by macro instructions issued by the front-end computer. The macro instructions are complex instructions, including an implied "virtuality" (see following section also) which causes that instruction to execute repeatedly on multiple data items within each processing element (or alternatively, as if there were a larger number of virtual processing elements).

MasPar has taken the alternative view of replacing the microprogrammed sequencer with a fully programmable computer – the Array Control Unit or ACU [1] [2]. An optimizing compiler can generate much better ACU code sequences to drive the array for parallel computation, than can be achieved with a preprogrammed sequencer (see more below). We have taken the RISC approach of depending on good compiler technology rather than preprogrammed, complex instructions.

A broad variety of software and services are available with the MasPar system because it incorporates a conventional UNIX computer (a DEC™ VAX™ ULTRIX™ workstation). Rather than defocus our development by developing our own scalar computer and version of UNIX, we chose to simply utilize an existing UNIX workstation. This UNIX subsystem is the basis for our development environment, providing all the conventional UNIX tools, and linking the MasPar system into the available network-based resources. The MasPar development software runs under ULTRIX on the workstation.

The MasPar programming model utilizes the front-end UNIX system where appropriate in the support of massively parallel applications. Preexisting code, such as window-based interaction code or network-based data access code, can continue to run on the MasPar UNIX front-end, untouched, while only the computational kernel of the application is adapted for parallel execution and migrated to run on the data parallel machine. The MP-1 has been designed so that communication between the front-end resident segment of an application and the parallel segment of the application is through directly mapped hardware queues, without any operating system intervention, so that communication is very efficient.

Scalar segments of an application can run either on the front-end or the ACU. In some cases most of the application scalar code runs on the front-end, for example, if the application utilizes complex UNIX resources such as X-Windows. In other cases the application runs largely on the ACU. For example, a signal processing and analysis application might run entirely on the ACU for maximum speed and real-time responsiveness. A later section on Programming Languages gives a more detailed view of the programming options available.

Parallel Virtuality

The MasPar system advances the concept of parallel virtuality by using both optimizing compiler technology and architectural design. In contrast, in the Connection Machine, parallel virtuality is largely an instruction set or architectural concept. As has been proven in the use of RISC computers, there can be very significant benefit from using compile time optimizations to augment what can be done at machine design time or program execution time. The same is true in the effective realization of a parallel virtuality.

Virtuality is best understood in its absence. Earlier massively parallel machines, such as the Goodyear/NASA MPP [4] [5] and earlier versions of the AMT™ DAP™ [6], had the physical dimensions of the processor array clearly visible in application programs. The lowest dimension of a data array was necessarily the dimension of the physical processor array (e.g., on a 128 by 128 array, the lowest dimension of a data array had to be 128). It is generally agreed that the lack of virtually is a problem for effective and flexible application programming, and certainly an impediment to the broad use of these machines. The important cases are when the data array is larger than the physical array available, or when software created for one sized array must be moved to an array of a different size.

The Connection Machine solves this problem by making virtuality an architectural concept – instructions specify the virtuality of the operation. For example, a virtual to physical ratio (V/P ratio) of 16 would create a virtual processor array 16 times larger than the physical array, in which case a parallel ADD instruction would be executed 16 times on each PE, once for each virtual PE. With instruction set virtuality, higher level software, such as compilers, treats the system as having an array size which is more directly suitable for the problem being solved, largely eliminating the issue of array size as a programming issue. Such architectural virtuality also permits software to remain unchanged when the physical array size changes. The program assumes the same virtual array size. Within the hardware system, the V/P ratio changes and with in the size of the loop on each PE.

MasPar has taken a different approach utilizing optimizing compiler technology [7] as well as elements of architecture and machine design. Instead of building virtuality into the instruction set, we use techniques analogous to the management of vector register sets in supercomputers such as the Cray. A Cray programmer freely uses data vectors that are much longer than the size of the vector registers [8]. The optimizing compiler uses techniques such as "strip mining" to break longer vector into data strips that fit in the vector registers, and subsequently optimizing the loop code to minimize data motion in and out of the vector registers. Without such optimization, a Cray application programmer would be forced into explicitly coding the vector register length. The VLSI implementation of the MasPar machine provides a PE register set with the equivalent opportunity for performance optimization. Just as the Cray compiler restructures

code so that data motion in and out of the vector registers is minimized, the MasPar compilers optimization use of the PE registers. This class of optimizations is not possible with an instruction set concept of virtuality. It depends necessarily on the complex analysis of the application done by the compiler.

In summary, everyone agrees that parallel virtuality in some form is important. MasPar implements virtuality with the support of compiler optimizations rather than simply as an instruction set, or architectural feature (although the architecture is designed to assure that compiler generated virtuality is efficient).

Programming Languages and Their Compilers

If massively parallel computers are to play a significant role in practical use, they will be programmed in conventional languages, or dialects of conventional languages. MasPar provides adaptations of C and Fortran suitable for programming massively parallel machines.

The three languages provided -- MasPar Fortran (MPF), MasPar C (MPC), and the MasPar Parallel Application Language (MPL) -- divide into two categories.

- MasPar Fortran and MasPar C are languages that generate code for all parts of the MasPar system (the front-end system, the ACU and the parallel array) from a single program. Although three independent, synchronized, code streams are created from the program, the programmer does not see any seams or boundaries. Furthermore, the MasPar Symbolic Debugger (see later) reintegrates these different code streams into the appearance of a single program, for the convenience of symbolic debugging. MPF and MPC provide the greatest ability to ignore the details of the computer architecture, and deal in a high-level, application-related programming abstraction.
- MPL only generates code for the parallel subsystem (the ACU and PE array). MPL is also C-derived but remains more in the style of C as a "power" language for direct, high-level control of hardware. Typically an MPL program is used in conjunction with a jointly designed front-end UNIX program to form a composite application. The MPL programmer typically deals with a visible seam between front-end and back-end code, although the MasPar Symbolic Debugger provides means for multi-windowed, coordinated symbolic debugging of the composite application. MPL is particularly valuable for porting existing applications, as will be discussed below (see Porting).

MasPar Fortran is based on Fortran 77, with the parallel and array features of Fortran 88 added. Most importantly, in this dialect, arrays are a first-order language concept. Array calculations can be expressed as simple expressions. Arrays can be used either in their entirety, or through a powerful sectioning mechanism which permits parts of an array to be flexibly specified and utilized. The MPF compiler automatically generates PE array code for the parallel computations in the code. Data is moved between the front-end and ACU automatically, as needed.

In addition to the kinds of optimization typically found in a supercomputer optimizing compiler (including strip mining and the related loop transforms), the MPF compiler includes a category of new optimizations which analyze data placement and motion, and generate code to dynamically restructure arrays within a computation. Data placement optimization has always been a key goal for compilers. The effective temporary allocation of variables to registers is a key data placement optimization in modern RISC compilers. The distributed memory in a massively parallel computer generates the need for additional data placement optimizations.

Although any data in the PE array can be accessed simply by using the global router mechanism, there is a significant latency difference between data that is the local PE registers, data that is in the local data memory of that PE (roughly 10 times slower), and data that is in the local data memory of another PE (roughly 100 times slower than in the local register). A fundamental goal of the MPF compiler is to make the best possible use of the PE register, high-speed memory. Ideally all variables needed in a computation will already exist in the registers of the same PE. Failing that, the data should be the local data memory of that PE.

The latency of the distributed memory requires data placement optimization for maximum computation, however the enormous aggregate bandwidth of the global router subsystem provides a powerful mechanism for solving the problem. When optimal, the compiler generates code to reshape a data array (remap the elements into PEs and PE memory) to speed the subsequent array parallel computations. In contrast to most Fortran systems, data location in MPF is dynamic, and data is moved to optimize computation. Data location and motion analysis is also used to optimize the location of temporary intermediate results during computations. Using symbol information from the compiler, the MasPar Symbolic Debugger unwinds these optimizations and supports symbolic debugging of the optimized code.

In summary, MPF is a subset of a standard Fortran dialect which permits effective coding of the MasPar computer without losing the ability to run the code on other computers, all of which will support the Fortran 88 language over time. The programmer largely ignores the issues specific to the MasPar design and focuses instead only on the application problem, leaving the task of adapting the code to this architecture to an optimizing compiler (as is always the case with high-performance computers, there is benefit in understanding how the the compiler/computer system operates, and programming in a style that leverages that understanding).

MasPar C is an extension of ANSI C in which the parallel array can be programmed explicitly in a virtual (size independent) model. The best existing model for MPC is Thinking Machines Corporation's C*™ language.

Finally, MasPar provides a simpler, massively parallel C, MPL, for programming the ACU and parallel array. MPL is designed much more in the traditional view of C as a high-level language which doesn't get in the way of doing what you want. MPL is based on K&R C, with three basic additions (to the degree possible, the semantics and syntax of K&R C are preserved):

- A variable can be declared as plural, in which case it is instantiated on all PEs; otherwise it is instantiated on the ACU. Expressions can mix plural and non-plural variables in which case the scalar variables are promoted to plural (broadcast to the PEs) and the computation done in parallel. Reduction operators exist for moving from plural data to scalar.
- Control structure semantics are adapted to a SIMD computation model. If the control expression of a conditional structure is plural, then the expression is evaluated in a plural form, and only a subset of the then active PEs evaluates the expression as true. In SIMD control structures, rather than bypassing unselected code structure, the active set of PEs is subdivided, and each subset executes the relevant code. One subset will execute the THEN clause; the disjoint subset the ELSE clause. Plural control structures such as SWITCH are transformed similarly into subsetting the active set of PEs. In general, programming in MPL can be thought of as writing a C program which executes on each PE independently but synchronously. The key exception

is that scalar code within these plural control structures is different from traditional C since scalar code in both the THEN and ELSE clause will typically be executed if the control expression is plural.

- Language syntax has been added to support use of the X-Net (neighbor communications) and the global router. This syntax supports access to a plural variable on a neighbor PE, or an arbitrary PE, respectively. In the case of X-Net communication, the most visible constraint is that the data motion direction is SIMD -- all PEs will communicate with the same direction neighbor (e.g., the Northern neighbor) -- or systolic, and not data sensitive.

  The use of the global router provides perhaps the most profound contrast between SIMD and MIMD programming. Global router communications provide a means of generalized data exchange throughout the array. However because the parallel program execution is synchronous (i.e., each PE executes the same program and at each instant, the same instruction of the same program), data messages can only be received at the point the program where they are being sent. Therefore all of the very difficult issues of interprocessor communications and synchronization (e.g., interrupts, buffering, critical code sections) are reduced to a single language construct. Aside from the segments of a program where data is being sent, data will not be received. It sounds trivial and obvious until you watch a massively parallel application being created, at which point the practical advantages of the SIMD model are enormous in terms of program design and debugging simplifications.

Because the base language is C, and the extensions minimal and consistent with scalar semantics, trained C programmers program comfortably in MPL very quickly.

Application Porting and Adaptation

The use of a mature UNIX front-end computer tightly integrated with the ACU, and the MPL language, provide a remarkably effective means for application porting.

The first step in such a port is to move an existing application to execute on the UNIX front-end. Because of the spreading use of UNIX and the commonality between systems, typically this is a straightforward and relatively simple task. Because the front-end is a VAX running ULTRIX, the VMS Fortran compiler (called "fort" under ULTRIX) may be used. Since VMS Fortran is broadly used within the scientific and engineering community, this is a significant advantage. Optionally other standard UNIX languages may be used.

Next, the key computation kernels of the application are identified, and critical data structures and their computations recoded in MPL. Access to these MPL subroutines from the original application is comparable to calling a C subroutine in a normal UNIX applications, although functions must be used to explicitly move data between the front-end and back-end since the memory spaces are disjoint. Most of the application code is typically not in the computation kernel, so the amount of time to port an application can be remarkable small. The porting effort does require that the numerical methods be adapted for massively parallel execution. Fortunately, in the last decade a wealth of basic research has been done on machines like the Goodyear MPP, the DAP and the Connection Machine so algorithm design involves adaptation of published methods, not necessarily the invention of new methods

This process often goes quickly, and demonstrates significant speed-up and cost performance increase. At some point, diminishing returns will be be encountered, as the scalar computation remaining prevents significant overall applications speed-up. This effect has been coined "Amdahl's Law" and is sometimes given as the reason why parallel machines cannot succeed.

We interpret Amdahl's Law in a quite different manner. Each new generation of computer (e.g., minicomputers, personal computers and workstations) has spawned an entirely new style of application. Minicomputers did not just run existing batch programs cheaper. Instead the real-time design of minicomputers spawned new, highly interactive applications. One could probably adapt Amdahl's Law to demonstrate that minicomputers could not succeed because the gain on existing batch applications was insufficient, but that analysis would have clearly missed the point.

Similarly, we believe the issue in the use of massively parallel computers is not simply how effectively the existing computation can be parallelized (the naive interpretation of Amdahl's Law), but rather how additional computation can be utilized in a new version of the application. For example

- How can geophysical exploration be made more effective if computation is an order of magnitude less expensive, and if the "supercomputer" can be deployed on the exploration truck?
- How can the VLSI design process be speeded if there is an order of magnitude more computation available and complex checking and analysis can be done on an ongoing basis rather than as a batched verification process?

We believe that the implications of Amdahl's Law for massively parallel machines will be yet another adaptation of applications, this time to utilize much more computation.

Programming Support

The MasPar programming system goes well beyond what has been traditionally expected in a supercomputer (for which the tools have traditionally been relatively primitive). Clearly programmability is the key issue in the success of massively parallel computers. Believing significant improvements could be achieved in the programability of high-performance computers, we invested heavily in developing application development tools.

The low price of the MasPar system compounds the problem of programmability. If a computer of a given power is offered at a much lower price, then the relative cost of programming goes up in the ratio of the old to new price (e.g., if the power of a $10M computer is offered at $500,000, the relative cost of programming increases by a factor of 20). We had to improve programmability to keep our computer realistically affordable at a minicomputer price.

Part of the MasPar software investment went into the design of high-level languages and implementation of optimizing compilers. Ideally, massively parallel computers can be programmed in familiar high-level languages, and optimizing compilers will do most of the work of mapping the algorithm into efficient code for a specific computer architecture. The efficiency of such compiler generated code will improve over time as the specific optimization and transformation issues are better understood (as has been the history of Cray compilers, for example) We are very pleased at what we have achieved to date, which validates the strategy of counting on compiler technology.

Additionally we have invested in a graphical, window-based, programming environment, which permits symbolic debugging of compiler-optimized code, and graphical viewing of both application execution and machine operation.

Our key assumption was that workstations and network standards were a major force, and here to stay (which has clearly proven to be the true), and that by utilizing the interactive power of a workstation, and the graphical remote access available over a local area network, we could significantly improve the programmability of a high-performance computer. We designed a programming system that depends on a graphical display (e.g. cannot be used from a remote teleprinter connected by a 300 baud line).

A graphical, interactive programming environment has the same basic advantages in a high-performance computer system as it does in a personal computer or workstation, and yet few high-performance computer vendors have developed such tools. One potential problem would be access to the programming system from a remote station. Early workstations had limited, and vendor-specific, remote access capabilities. It was our judgement that in the 1990 timeframe in which we would introduce our product, it was reasonable to count on high-performance, local-area networks and graphical, virtual-terminal connections. In early 1988, we bet on the broad acceptance and success of X-Windows (over SUN™ NEWS™ for example), which at the time was not an assured decision, but has since proven to be a good choice.

By using a graphical interface we have created a programming environment that brings much of the convenience of programming personal computers or workstations to a machine with supercomputer performance. The convenience of the programming system is very important given that ultimately the success of a machine like the MP-1 depends on many applications being successfully ported or adapted to run. The easier the program design or adaptation task, the more quickly that will occur.

The MasPar programming environment includes tools for browsing and navigating through complex application codes, source level debugging, and various tools for visualization the application and machine execution. Since we assume that a programmer is using a graphical display terminal, we reason that the more we can use graphical displays to explain what the machine is doing, and the more quickly a programmer will internalize what it means to do effective design and implementation of massively parallel application. For example, a simple graphical tools displays the E-bit (activity bit) for each PE on a time sampled basis providing an immediate and intuitive demonstration of the parallel effectiveness of the application executing.

The key contributions in the MasPar Programming Environment are these:

- A graphical, multiwindowed, point-and-select interface.
- A modified UNIX object file format that
  - permits symbol access on demand rather than all at once (the modification in symbol structure greatly reduces the debugger start-up delay on large applications)
  - permits incremental delinking and relinking of a single object module for incremental rebuilding of an application (greatly speeding the development cycle of large applications).
  - contains a dual code stream – the front-end code and ACU code (simplifying the management of dual code stream application development).
  - includes greatly enhanced symbol information for debugging (enabling the symbolic debugging of optimized code).
- A symbolic debugger (and compilers) that permit symbolic debugging of highly optimized code. The debugger handles both single source programs with front-end and back-end code components, or dual, cooperating, source programs, one for the front-end and one for the back-end.
- Interactive browsing tools for the convenient, graphical navigation through complex applications.
- Data inspection tools to examine large data structures and arrays graphically.
- Machine animation tools that create graphical depictions of the parallel machine operation, showing for example, the current set of active processors.

Summary

VLSI technology and massively parallel architecture are combined in the MP-1 to offer what would be considered supercomputer performance at a minicomputer price. The challenge of the MasPar software system is to permit supercomputer programming with minicomputer convenience. We feel that our products, which include massively parallel dialects of common languages, optimizing compilers, and a graphical programming environment, integrated with a network, UNIX implementation, to have accomplished that goal.

References

[1] Tom Blank, "The MasPar MP-1 Architecture", *Proceedings of the IEEE Compcon Spring 1990*, IEEE, February 1990.

[2] John Nickolls, "The Design of the MasPar MP-1, A Cost-Effective Massively Parallel Computer", *Proceedings of the IEEE Compcon Spring 1990*, IEEE, February 1990.

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

[4] Kenneth E. Batcher, "Design of a Massively Parallel Processor", *IEEE Transactions on Computers*, vol C-29, pp. 836-840, Sept 1980.

[5] Ken Batcher, "The Architecture of Tomorrow's Massively Parallel Computer", *Proceedings of the First Massively Parallel Processing Conference*, 1987.

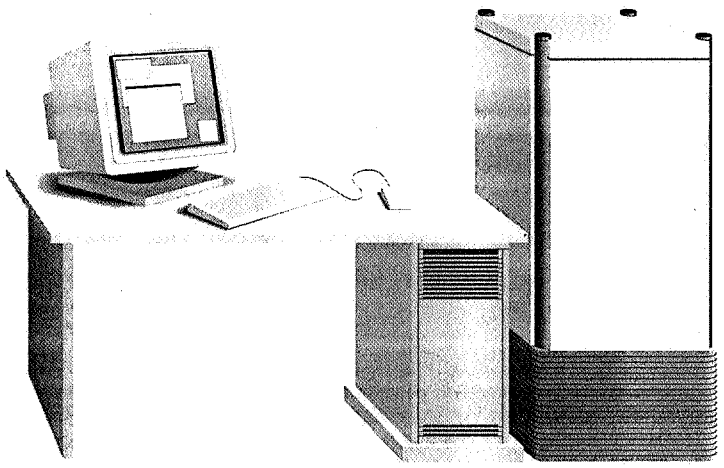
[6] S.F. Reddaway, "DAP, A Distributed Array Processor", First Annual Symposium on Computer Architecture, IEEE/ACM, Florida, 1973.

[7] Michael Wolfe, *Optimizing Supercompilers for Supercomputers*, MIT Press, 1989.

[8] Randy Allen and Ken Kennedy, "Vector Register Allocation", Rice University Department of Computer Science, COMP TR86-45, 1986.

MASPAR

MasPar 1200 Series Computer Systems



The MP 1200 Series consists of five entry-to-large configurations from the MasPar MP-1 Family of data-parallel computers. Each of these systems breaks new ground in price/performance, scalability, and I/O capacity – all working in concert to create a new class of high-performance computers.

Highlights

❒ **Exceptional Price/Performance.** The MP 1200 Series provides up to 26,000 MIPS and 1,300 MFLOPS – at minicomputer prices. Never before has so much performance been made available within the minicomputer price range.

❒ **System Scalability.** The MP 1200 Series can be configured with 1,024, 2,048, 4,096, 8,192, and 16,384 Processor Elements at initial installation or through convenient on-site upgrades. Thus, the MP 1200 Series achieves a new level of scalability in high performance computing – from an entry system providing 1,600 MIPS to a maximum configuration offering 26,000 MIPS – and all within the same system package.

❒ **Very High I/O Capacity.** The complete range of MP 1200 Series computers offer peak I/O throughput of 200 megabytes per second through the MasPar I/O Channel (MPIOC), and with additional customer-designed hardware, a fully configured 1200 Series system can achieve 1.3 gigabytes per second through direct connection into the MP-1's Global Router communications system.

MasPar 1200 Series Computer Systems

Specifications

DATA PARALLEL UNIT

Hardware

CPUs

Array Control Unit (ACU)
Processor Element Array (PE Array) Configurations:
1,024 Processor Elements (PEs)
2,048 Processor Elements
4,096 Processor Elements
8,192 Processor Elements
16,384 Processor Elements

Peak Performance

PE Array Size	MIPS	32-Bit MFLOPS	64-Bit MFLOPS
1K PEs	1,600	82	36
2K PEs	3,200	164	73
4K PEs	6,400	325	145
8K PEs	13,000	650	290
16K PEs	26,000	1,300	580

Peak performance ratings are based upon integer 32-bit addition (MIPS) and the average of floating point multiply and add (MFLOPS).

Memory

16 - 256 megabytes ECC protected total PE data memory (16 kilobytes per PE)
(System memory scales at the rate of 16 megabytes per 1K PE Array)
192 bytes register memory/PE
1 megabyte physical ACU instruction memory
Up to 4 gigabytes ACU virtual instruction memory memory
128 kilobytes ACU data memory

Inter-processor Communications

1.1-18 gigabytes/second peak X-Net , eight-way, nearest-neighbor communications
(between neighboring PEs on a 2 dimensional grid, torroidally wrapped at the edges)
80-1300 megabytes/second peak Global Router communications
(between arbitrary PEs or between PEs and I/O)
Performance scales linearly with the number of PEs in the system

Backplane Slots

16 slots available for PE Boards (1,024 PEs/Board)
16 slots available for I/O RAM and controllers

I/O Channel Options

MasPar I/O Channel (MPIOC)
- 64 bit data bus
  200 megabyte/second transfer rates
- 256 megabytes maximum dedicated I/O RAM buffer

Capability for development of VME-based controllers
MP-1 Global Router
Experienced I/O system designers may, by special arrangement with MasPar, have direct access to the global router's 1,024 maximum I/O connections for a peak bandwidth of 1.3 gigabytes per second.
- 64 - 1,024 Router connections
- 80 - 1300 megabyte/second peak transfer rates
  (Performance scales linearly with the number of PEs in the system)

Physical Characteristics

Enclosure

55.5"H x 23.5"W x 32.5"D
< 800 lbs

Power requirements

Line Voltage: 220V/240V 1Ø
Frequency: 60 Hz/50 Hz
Current Rating: 30A
Maximum Power Consumption: 800-3700W depending on configuration

UNIX SUBSYSTEM

Hardware

VAXstation 3520

2 CVAX Processors
16 MB ECC memory (Optional expansion to 32 MB)
1 332 Mbyte RZ55 SCSI 5-1/4"disk (Optional expansion to 664 Mbyte disk)
1 296 Mbyte TK70 streaming cartridge tape
Ethernet™ (ThinWire™ or thick wire, transceiver included)
105-key keyboard
3-button mouse
2 serial lines

19" color monitor

66 Hz refresh
1280 x 1024 x 8 planes image resolution
Optional additional 16 color planes
Optional 24 bit Z-buffer
Optional Gouraud shading and depth cueing

Physical Characteristics

Enclosure

CPU: 27"H x 21"W x 18"D, 108 lbs
Color Monitor:18.5"H x 20"W x 22"D, 75 lbs
Keyboard: 17"W x 2"H x 7"D

Power Requirements

Line Voltage: 120V/240V 1Ø
Frequency: 60 Hz/50 Hz
Maximum power consumption: 670 W

SYSTEM SOFTWARE

Note: All MasPar and Digital Equipment Corporation supplied software products include a license for 4 active users unless otherwise noted.

Operating System

ULTRIX™ with symmetric multiprocessing (SMP), compliant with OSF Level 0 specifications
IEEE POSIX standard

The MasPar Execution Environment integrates operation of the UNIX subsystem and the DPU.

Network Communications

- Ethernet™
- TCP/IP
- NFS™
- DECwindows™ (includes support for X 11 Windows)
- DECnet™

Graphics

- Programmer's Hierarchical Interactive Graphics System (PHIGS) - Run-time

Languages

- ☐ VAX C
- ☐ Ultrix C
- ☐ MasPar Parallel Application Language (MPL)
- ☐ Optional languages:
  - MasPar Fortran (MPF)
  - VAX Fortran

MasPar Programming Environment (License for one active user)

- ☐ MasPar Symbolic Debugger
- ☐ Machine Animator
- ☐ Data Animator
- dbx and dxdb (for debugging serial code only)

Utilities

- All standard ULTRIX Worksystem software utilities are available

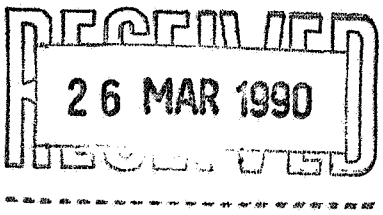
Remote Service Access

- Integrated Telebit Trailblazer™ T9000 Modem with automatic error correction to 19,200 bps for remote login and uucp access.

MasPar Computer Corporation
749 North Mary Avenue
Sunnyvale, California 94086
408-736-3300
FAX: 408-736-9560

PL004.0490

MASPAR



DATA-PARALLEL RESEARCH INITIATIVE

At MasPar we believe that data-parallel computing is the best choice for more and more high performance users. And today, we offer a complete massively parallel solution for a wide variety of people who want high-performance scalable systems at affordable prices. For more details on the MasPar MP-1 family of massively parallel computer systems, simply return the enclosed business reply card.

If you have a research project that can utilize a data parallel computer system then we have some good news for you. Earlier this year, MasPar Computer Corporation and Digital Equipment Corporation announced the Data Parallel Research Initiative. Universities and other research organizations are invited to participate with MasPar and Digital in this Initiative.

Institutions whose proposed projects are accepted into the Initiative program may purchase MasPar computer systems and software at discounts approaching 45%, with the option to purchase additional equipment from Digital at discounts of up to 75%. In addition to these generous discounts the principal investigators on the project are invited to participate in an annual conference to present the results of their work.

Complete details on participation in this Research Initiative can be found in the enclosed document and if you are interested in submitting a proposal please contact me to obtain the necessary application forms.

Yours faithfully

Neil Rowlands

MasPar Computer Corporation

MasPar Computer Corporation
First Base, Beacontree Plaza, Gillette Way, Reading RG2 0BP, Berkshire, England / (0734) 753388 / FAX (0734) 313939

MASPAR

Data-Parallel Research Initiative

European Program
February 1990

MasPar Computer Corporation
First Base, Beacontree Plaza, Gillette Way, Reading RG2 0BP, Berkshire, England (0734) 753388 FAX (0734) 313939

Data-Parallel Research Initiative

In January 1990, MasPar Computer Corporation and Digital Equipment Corporation announced MasPar's participation in Digital's Data-Parallel Research Initiative. This program, which was introduced by Digital in May 1989, seeks to advance the art and science of data-parallel computing by sponsoring research and development on data-parallel applications, programming tools, and network access to data-parallel machines in heterogeneous computing environments. Universities and other research organizations are invited to participate with MasPar and Digital in this Initiative.

This program description provides full details on the Data-Parallel Research Initiative and a format for proposals. Proposals will be accepted at any time; investigators should plan to complete the research proposed under the Initiative by June 30, 1992.

Motivation

Current applications of data-parallel computer systems include molecular modeling, computational fluid dynamics, structural and thermal analysis, image processing, signal processing, geophysical analysis, VLSI system analysis, optimization, and information management. Among the objectives of the Initiative are encouragement of innovation in algorithms and data models to achieve enhanced performance in these applications, and stimulation of new applications in other areas of engineering, scientific, and commercial computation. Of particular interest are applications which would produce substantial economic impact if they were widely available on data-parallel machines.

Significant gains have been made in parallel programming languages and compilers for many architectures, including those of data-parallel machines. The Initiative hopes to stimulate tool-related research which will assist the partitioning of problems for data-parallel machines, facilitate the conversion of existing (and typically vectorized) codes, and further improve the efficiency or usability of data-parallel machines.

The concentration of computational power which is typical of data-parallel machines makes them attractive as application servers for compute-intensive applications, just as the concentration of I/O resources makes a machine an attractive data server for I/O-intensive applications. The Initiative expects to sponsor research which enables and demonstrates convenient, high-speed access to data-parallel machines in heterogenous computing environments.

Program Features

Universities and other research organizations are invited to submit proposals describing the research to be undertaken, including a description of the technology to be developed, a plan for its dissemination in the data-parallel research community, and a statement of the resources required for the work. The research may be begun at any time, but projects should be planned for completion by June 30, 1992. The specific form of this proposal and the mechanisms of its evaluation are described below.

Participating institutions are granted the opportunity to purchase MasPar MP-1 data-parallel computer systems, and Digital products required for the research at substantially reduced prices which reflect the sponsors' commitment to the Initiative. Normally, institutions are expected to provide for personnel and other operating costs of their Initiative projects.

Technical collaboration with MasPar and/or Digital may be a component of the project, or not, as the project requires, but the technology developed by the project is to be available to both Digital and MasPar, directly or by placing the technology in the public domain. The research results of the project are to be disseminated through publications in technical journals and presentations at conferences. To further encourage sharing of information within the data-parallel research community, Digital will sponsor an annual conference at which the interim results of current projects and the final results of projects completed within the year will be presented. The first of these conferences will be held in Fall 1990. Travel support for principal investigators' participation in these conferences will be provided by Digital.

Scope of the Initiative

The Data-Parallel Research Initiative is open to universities, non-profit research organizations including government agencies, and corporate research organizations. Participants must be free to share the research results of their Initiative project with the data-parallel research community through publication and presentation, and they must present a plan to share the technology developed in their project with Digital and MasPar.

The Initiative is open to research institutions in all areas of the world, subject to any export restrictions of the U.S. government which may apply and to the ability of both MasPar and Digital to support their products in the location of the institution. Neither of these limitations is expected to apply to institutions in the U.S. , Canada, Western Europe, or Japan; in other areas, an initial inquiry directed to MasPar is a desirable first step to assure that these possible limitations do not preclude participation.

Institutions whose proposed projects are accepted into the Initiative program may purchase MasPar computer systems and software and Digital hardware and software, which are required for the project and specified in the proposal, at significant discounts.

MasPar systems acquired for an approved Initiative project are subject to a special price list which incorporates substantial discounts. Under the Initiative, additional MasPar software or service options are also available at substantial discounts. Installations of more than one MP-1 at a project site may be supported, given specific research justification. Digital workstations or other systems, network hardware, and software may be included in the proposal's equipment requirement, where their use is coordinated with that of an MP-1 in the proposed project, at discounts of 75%. Post-warranty maintenance of computer systems purchased under the Initiative can be prepaid through the project period at discounted rates, as well.

The Proposal Process

The proposal process for the Data-Parallel Research Initiative is straightforward, and proposals may be submitted at any time. Normally the proposal review and qualification process requires no longer than six to eight weeks.

Consultation

Questions about the Initiative or its proposal process may be addressed to any of the MasPar or Digital contacts indicated below, at any time. Applicants outside the U.S., Canada, Europe, and Japan should inquire about eligibility.

Applicants should consult with a MasPar sales representative in their geographic area for detailed information about MasPar system configurations, pricing, and availability. A MasPar quotation detailing the MP-1 system(s) and any additional MasPar hardware, software, or service options, and showing the Initiative prices for these components, must be included with the proposal.

In many cases, discussion of the proposed research with MasPar field applications personnel or corporate applications staff will be useful, in advance of preparing a proposal.

Where Digital workstation options, additional workstations or other systems, network hardware, or software are required for the research, applicants should also consult with their local Digital sales representative. A separate Digital quotation for these items must also be submitted with the proposal.

Proposal Submission

Proposals consist of four parts, a cover sheet, a narrative of approximately 10 pages, quotation(s), and optional supporting information. The cover sheet indicates separately the total costs, and Initiative allowances, of the MasPar and Digital components of the equipment requested for the project; these components are listed in the narrative of the proposal, corresponding to the referenced quotations.

The detailed layouts of the cover sheet and narrative are available, in either printed or electronic format, upon request from MasPar. To assure prompt and consistent evaluation of your proposal, the suggested format should be followed exactly.

The cover sheet and narrative should be submitted via electronic mail to Digital's External Research Program at:

eerp@europe.enet.dec.com

The MasPar quotation, and the Digital quotation if additional Digital components are required for the project, should be included with the cover sheet and narrative if the quotations are available in electronic form. If a quotation is not available in electronic form, it should be sent in hardcopy form, referencing the proposal by name and date, to

Digital Equipment Corporation
European External Research Program
12 av. des Morgines
Case postale 176
1213 Petit-Lancy 1
Switzerland

FAX +41 (22) 792.25.03

If you do not have access to an electronic mail system, your proposal will receive prompt attention if it is submitted on machine-readable magnetic media (standard magnetic tape or diskette). Proposals should be submitted in hard-copy form only if neither electronic mail nor magnetic media is available; in this case an original and one copy should be submitted. Hardcopy and magnetic media proposals are to be sent to the above address.

Please also send copies of your proposal to both your MasPar and Digital sales representatives.

Proposal Review

Proposals will be reviewed as they are received by a panel of Digital and MasPar technical staff. Normally a commitment to support the proposal can be made within eight weeks of its submission.

Notification

The principal investigator and the contract administrator of the proposing institution will be notified as soon as a decision is reached. Orders for required equipment, conforming to the quotations submitted with the proposal with any amendments agreed upon during the review process, may be placed with MasPar (and, if additional Digital components are included, with

Digital for these components) as soon as funds are available to the proposing institution.

Relationship to Other MasPar Programs

MasPar's participation in the Data-Parallel Research Initiative coordinates with MasPar's ongoing UniversityPartners Program, which provides discounts for purchases of MasPar systems by universities and non-profit research institutions and establishes cooperative research and development relationships with these institutions. Institutions whose projects are awarded Initiative grants will be designated UniversityPartners, which will make them eligible for discounted purchases of MP-1 system upgrades, peripherals, and extended service during the project period and beyond.

Discounts under the Initiative exceed those which are normally available under the UniversityPartners program. The larger discounts offered by the Initiative, and the opportunity to purchase related equipment from Digital at a 75% discount, make it appropriate for a qualifying institution to first seek support for an intended purchase by submitting an Initiative proposal Should such a proposal not be supported, the UniversityPartners discount structure remains available.

In some cases, where a research institution lacks sufficient funds for an Initiative purchase, sponsorship by a private corporation or foundation may be possible. Inquiries are invited from interested sponsors, and from research institutions who require sponsorship.

Contacts

For response to questions about this program, contact:

Neil Rowlands, Director of European Sales
MasPar Computer Corporation
First Base, Beacontree Plaza
Gillette Way, Reading
Berkshire RG2 0BP, England
Tel. +44 (734) 753388
Fax +44 (734) 313939

or

Deborah Murith, Program Manager
European External Research Program
Digital Equipment Corporation
12 av. des Morgines
Case postale 176
1213 Petit-Lancy 1
Switzerland
Tel. +41 (22) 709.48.76
Fax +41 (22) 792.25.03
murith@europe.enet.dec.com

For detailed information about MasPar systems, and for assistance in submitting a proposal to the Data-Parallel Research Initiative, contact your local MasPar sales representative.

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.

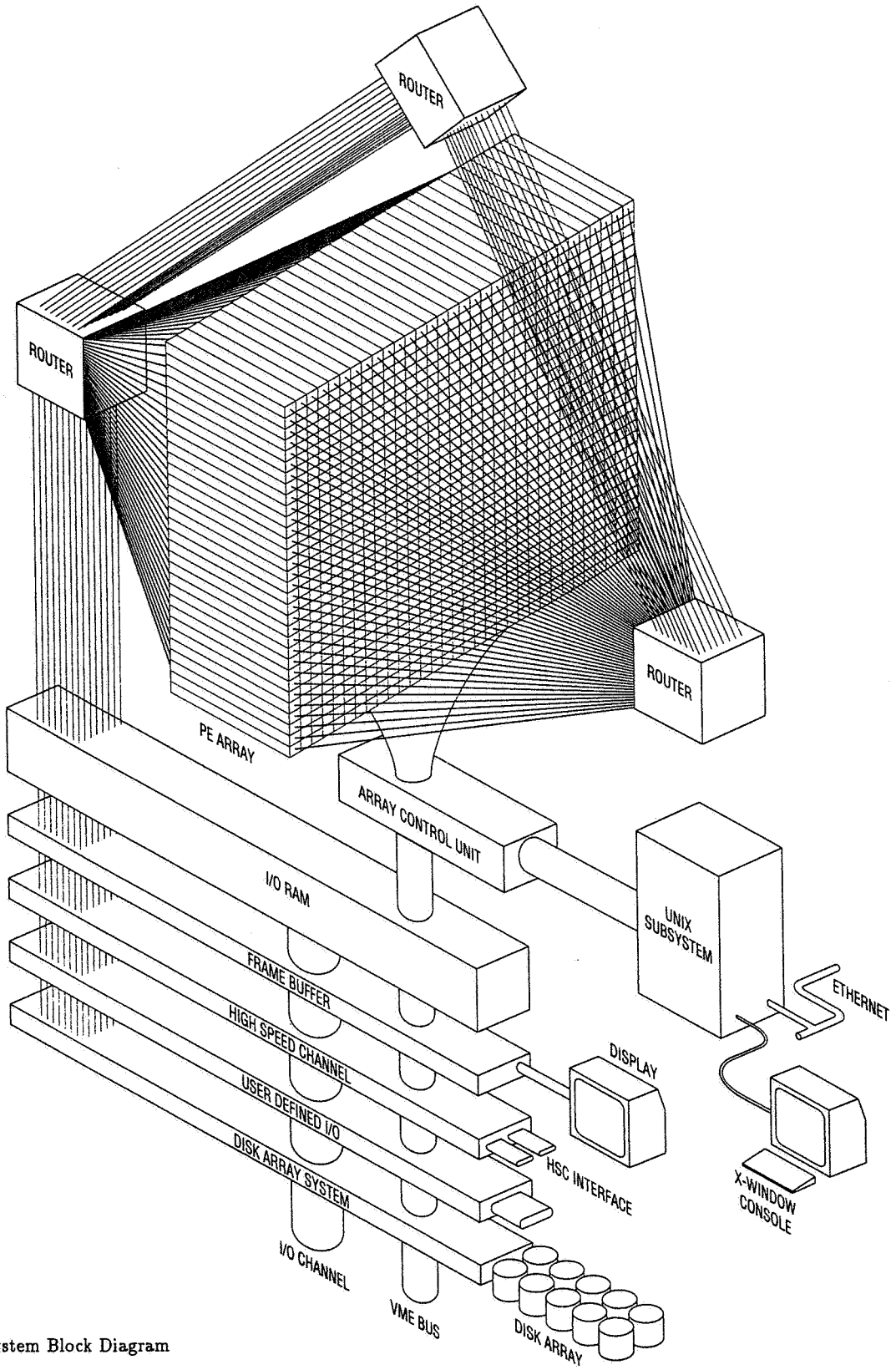


Figure 1: MP-1 System Block Diagram

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.

UNIX Subsystem (USS)

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

Communication Mechanisms

e following sections describe the five major communications chanisms. Included are descriptions of the programming del and instructions.

.1 USS to ACU

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

eues 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.

ared 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.

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

1.2 ACU to PE Array

vo basic capabilities are required for data movement between e ACU and PE Array: data distribution, DIST, and array nsensus detection which uses a global OR, GOR. An example age:

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

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

.4.3 PE Array: XNet

Net communications provide all PEs with a direct connec- ion to its eight nearest neighbors in a two dimensional mesh. pecifically, each PE is connected to its neighbors to the: North, Northeast, East, Southeast, South, Southwest, West, and North- vest. Processors located on the physical edge of the array have oroidal 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.

The Design of the MasPar MP-1: A Cost Effective Massively Parallel Computer

John R. Nickolls

MasPar Computer Corporation
749 North Mary Avenue
Sunnyvale, CA 94086

Abstract

CMOS VLSI and replication of components effectively, y parallel computers can achieve extraordinary ance at low cost. Key issues are how the processor and are partitioned and replicated, and how interprocessor ication and I/O are accomplished. This paper describes gn and implementation of the MasPar MP-1, a general massively parallel computer system that achieves peak ition rates beyond a billion floating point operations per yet is priced like a minicomputer.

Massively Parallel System

ely parallel computers use more than 1,000 processors to computational performance unachievable by conventional sors [1,2,3]. The MasPar MP-1 system is scalable from to 16,384 processors and its peak performance scales with the number of processors. A 16K processor system s 30,000 MIPS peak performance where a representative tion is a 32-bit integer add. In terms of peak floating point mance, the 16K processor system delivers 1,500 MFLOPS precision (32-bit) and 650 MFLOPS double precision (64- sing the average of add and multiply times.

fectively apply a high degree of parallelism to a single ation, the problem data is spread across the processors. processor computes on behalf of one or a few data elements problem. This approach is called "data-level parallel" [4] is effective for a broad range of compute-intensive ations.

ioning the computational effort is the key to high rmance, and the simplest and most scalable method is data elism. The architecture of the MP-1 [5] is scalable in a way permits its computational power to be increased along two the performance of each processor, and the number of ssors. This flexibility is well matched to VLSI technology circuit densities continue to increase at a rapid rate. The le nature of massively parallel systems protects the ners' software investment while providing a path to sing performance in successive products [6].

se its architecture provides tremendous leverage, the MP-1 mentation is conservative in terms of circuit complexity, rules, IC geometry, clock rates, margins, and power ation. A sufficiently high processor count reduces the need to have an overly aggressive (and thus expensive) implementation. Partitioning and replication make it possible to use low cost, low power workstation technology to build very high performance systems. Replication of key system elements happily enables both high performance and low cost.

Array Control Unit

Because massively parallel systems focus on data parallelism, all the processors can execute the same instruction stream. The MP-1 has a single instruction stream multiple data (SIMD) architecture that simplifies the highly replicated processors by eliminating their instruction logic and instruction memory, and thus saves millions of gates and hundreds of megabytes of memory in the overall system. The processors in a SIMD system are called processor elements (PEs) to indicate that they contain only the data path of a processor.

The MP-1 array control unit (ACU) is a 14 MIPS scalar processor with a RISC-style instruction set and a demand-paged instruction memory. The ACU fetches and decodes MP-1 instructions, computes addresses and scalar data values, issues control signals to the PE array, and monitors the status of the PE array. The ACU is implemented with a microcoded engine to accommodate the needs of the PE array, but most of the scalar ACU instructions execute in one 70 nsec clock. The ACU occupies one printed circuit board.

Processor Array

The MP-1 processor array (figure 1) is configurable from 1 to 16 identical processor boards. Each processor board has 1,024 processor elements (PEs) and associated memory arranged as 64 PE clusters (PECs) of 16 PEs per cluster. The processors are interconnected via the X-Net neighborhood mesh and the global multistage crossbar router network.

The processor boards are approximately 14" by 19" and use a high density connector to mate with a common backplane. A processor board dissipates less than 50 watts; a full 16K PE array and ACU dissipate less than 1,000 watts.

A PE cluster (figure 2) is composed of 16 PEs and 16 processor memories (PMEM). The PEs are logically arranged as a 4 by 4 array for the X-Net two-dimensional mesh interconnection. Each PE has a large internal register file shown in the figure as PREG. Load and store instructions move data between PREG and PMEM. The ACU broadcasts instructions and data to all PE clusters and the PEs all contribute to an inclusive-OR reduction

MP-1 Family Data-Parallel Computers

The MasPar MP-1 Family of data-parallel computers makes the vast power of massive parallelism affordable and practical for a wide range of developers, scientists, and engineers.

Outperforming many supercomputers yet priced like a minicomputer, MP-1 products provide a price/performance improvement of *up to a hundredfold* over conventional architectures. Numerous innovations contribute to the MP-1 breakthroughs – full-custom VLSI silicon, massive component replication, dense packaging technologies, and distributed memory and processing elements. In addition, MP-1 capabilities scale readily with customer needs and budget.

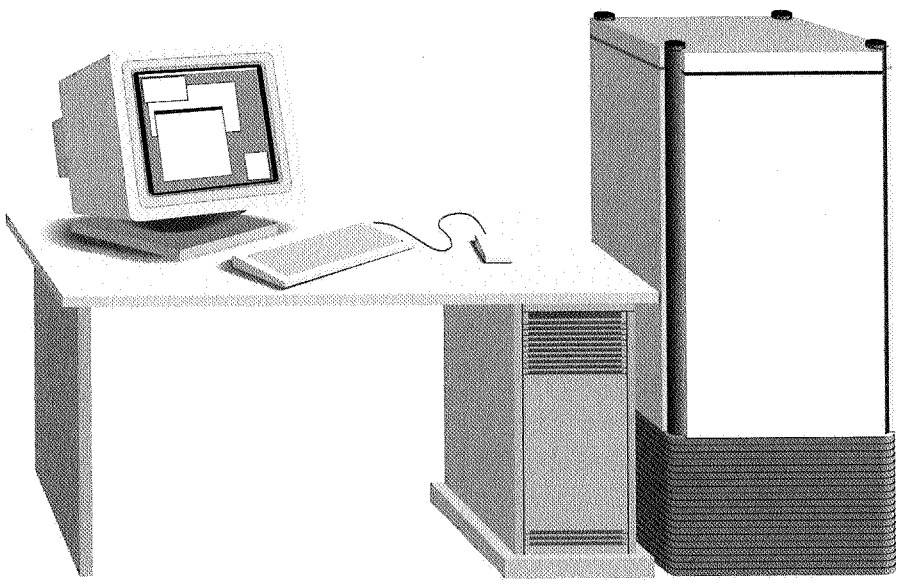
The MP-1 Family achieves an equally *dramatic breakthrough in programmability*. Its robust software development environment provides unique support for data-parallel programming. High-level languages and a suite of advanced development tools greatly facilitate the creation of effective parallel programs and the porting of existing applications. As a result, software developers can readily adopt an enduring parallel programming paradigm to exploit the power of massive parallelism as never before.

MP-1 products couple the advantages of this powerful processing and programming environment with the utility of a workstation. The systems are based on the UNIX® operating system and feature a graphical, interactive point-and-select interface built on the X Window System™ standard. They function as single-user systems or as networked computational resources. They are ideal for departmental use.

Highlights

- **Massively parallel architecture.** By employing thousands of processors to compute data-intensive problems, MP-1 computers provide users with an exceptionally powerful and expandable computing environment .

- **Quantum leap in price/performance.** MP-1 computers deliver exceptional power and breakthrough price/performance – as low as *$30 per MIP and $600 per Mflop* in a fully configured system.



MP 1200 Series System

- **Scalable performance.** The MP-1 system readily scales to accommodate demands for increased processing power, memory size and bandwidth, I/O bandwidth, and processor communications – *without requiring changes to application software.*

- **Family of products.** Available in configurations from 1,024 processors to 16,384 processors, MP-1 systems present a wide range of price/performance options: up to 30,000 MIPS and 1,500 Mflops.

- **Small footprint, low power.** Occupying a fraction of the floor space of traditional high-performance computers, MP-1 systems fit into an office environment without any special support. Depending on configuration, they use standard 110-volt or 220-volt AC power.

- **Unprecedented I/O capacity.** A fully configured MP-1 data-parallel computer can support peak I/O rates of up to 230 megabytes per second. With specially designed customer hardware, the architecture can support data rates of *up to 1.5 gigabytes per second*.

- **Data Parallelism.** The MP-1 Family achieves remarkable performance through its ability to operate on thousands of data elements in parallel. This effective paradigm for using massive parallelism simplifies the job of parallel programming and provides enduring protection for software.

- **Modern graphical programming environment.** Robust, interactive, graphical software tools – symbolic debuggers; visualizers of data, and processor behavior; and optimizing compilers – all enhance productivity for MP-1 application developers.

- **Mainstream languages with data-parallel extensions.** MasPar™ has enhanced Fortran and C with data-parallel facilities so that developers can produce effective parallel programs in a simplified manner. In addition, the MasPar Parallel Application Language (MPL) affords direct program control over the data-parallel hardware. MPL code integrates easily with many existing languages.

- **Industry-standard software.** The familiar UNIX operating system, the X Window System, and other standard protocols like TCP/IP and NFS,™ provide a powerful context for users and developers.

MP-1 Hardware

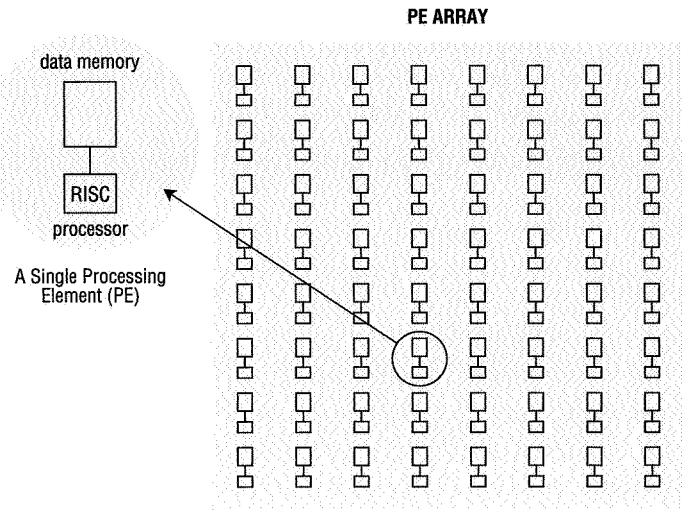
MP-1 architecture consists of four subsystems:

- The Processor Element (PE) Array
- The Array Control Unit (ACU)
- A UNIX subsystem with standard I/O
- A high-speed I/O subsystem

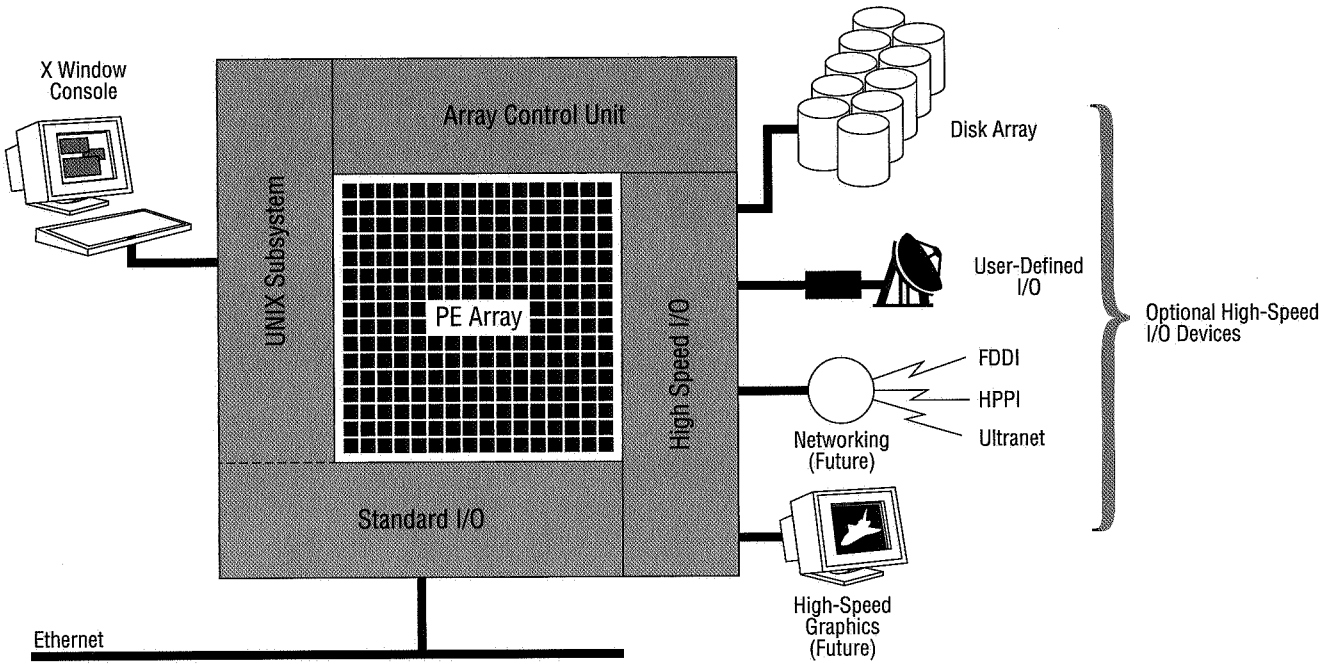
Working together transparently under the top-level control of UNIX, these subsystems provide a single, integrated computing environment for users. Together, they supply a wide range of computational power – from traditional serial processing on the UNIX subsystem to massively parallel computing at rates of up to 30,000 MIPS – while ensuring efficient overall behavior.

Processor Element Array. The computational heart of the MP-1 architecture is the PE Array. It is the source of the MP-1 Family's ability to operate on thousands of data elements in parallel.

Each PE in the array is a MasPar-designed, register-based, load/store RISC processor with dedicated data memory and execution logic. Each processor operates at 1.8 MIPS (32-bit integer add) and 90 Kflops (average of 32-bit floating point



Distributed Processors and Data Memory



MP-1 System Block Diagram

multiply and add). The PEs operate on standard integer data sizes of 1, 8, 16, 32, and 64 bits, and standard floating point operands of 32 and 64 bits. This capability produces highly efficient machine utilization. Extensive use of CMOS VLSI allows extremely dense packaging – 32 processors per chip, with forty 32-bit registers per PE.

By operating in a single instruction multiple data fashion (SIMD), thousands of PEs can work on a single problem simultaneously. Each PE has its own data memory and can share data with other processors through a global communications system. The PEs receive instructions from the Array Control Unit.

MP-1 systems currently support PE arrays with 1,024 to 16,384 processor elements. Peak performance ranges from 1,800 to 30,000 integer MIPS (32-bit add) and from 94 to 1,500 single precision Mflops (average of add and multiply). Double precision (64-bit) floating-point performance ranges from 41 Mflops to 650 Mflops. The aggregate PE Array data memory ranges from 16 to 256 megabytes. Designed with a scalable architecture, the MP-1 Family is positioned to exploit advances in semiconductor technology to allow increases in the number of PEs, PE processing power, or any of several other dimensions of system performance.

Array Control Unit. Highly focused on computation, the processor chips in the PE Array consign to the Array Control Unit most control and memory management functions ordinarily found in conventional processors. The ACU controls the operation of the PE Array as well as communications among the PEs and the rest of the MP-1 system. A dedicated, programmable control processor, the ACU is the *only* place where instructions are issued and decoded. This division of labor allows for maximum efficiency.

At the heart of the ACU is a register-based, load/store RISC processor designed by MasPar. The unit contains its own separate data and instruction memories and has 4 gigabytes of demand-paged virtual instruction memory, allowing it to operate independently from the UNIX Subsystem processor. Another independent functional unit – the Memory Machine – performs PE Array load and store operations while the ACU broadcasts such processor instructions as add, subtract, multiply, and divide to the PEs for execution.

Interprocessor Communications. Processor communications can become the most important determinant of performance in distributed-memory machines such as the MP-1. Accordingly, MasPar provides three highly efficient mechanisms for communicating with elements in the PE Array. These are:

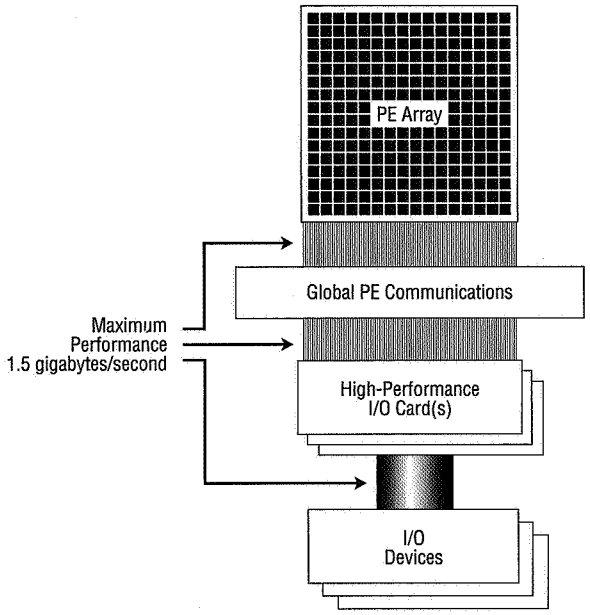
- ACU-PE Array communications
- X-Net nearest neighbor communications
- Global Router communications

Under the first mechanism, the ACU broadcasts values to all processors in the array simultaneously and performs global reductions on parallel data to recover scalar values from the array. The second mechanism, X-Net, supports high-speed data movement to and from the eight nearest neighbors of each PE. Useful for moving data arranged in a uniform array, X-Net Communications operates at a peak bandwidth of more than 24 gigabytes per second in the largest MP-1 configuration.

The Global Router mechanism handles communications of arbitrary connections among processors within the PE array. By providing a multi-stage hierarchical crossbar switch, the Router concurrently establishes links, each of which will enable any two processors to send or fetch data. In an MP-1 system configured with 16,384 processors, this connection mechanism supports 1,024 simultaneous links and operates at an aggregate bandwidth of 1.5 gigabytes per second.

UNIX Subsystem. A VAXstation™ 3520 manages program execution, user interface, and network communications for an MP-1 system. When there is a need for massively parallel execution, it invokes the ACU and PE Array. This scalar processor is configured with two CPUs, runs ULTRIX,™ Digital Equipment Corporation's UNIX implementation, and includes Ethernet™ hardware and TCP/IP, NFS,and DECwindows™ software, which is based on the X Window System, Version 11. The subsystem also features 16 megabytes of memory, an eight-plane color frame buffer, and a range of standard I/O devices including a 19-inch 1280x1024 high-resolution monitor, standard keyboard and three-button mouse, a 332-megabyte SCSI 5-1/4-inch disk drive, and a 296-megabyte streaming cartridge tape. Options include up to 16 megabytes of additional memory as well as more disk storage, a 24-plane color frame buffer and such graphics software as PHIGS, PEX, and GKS.

I/O Subsystem. The MP-1 Family provides a wide range of I/O performance that ranges from standard workstation speeds to a 230 megabyte/sec 64



Massively Parallel I/O Architecture

bit channel (MPIOC), and then up to a maximum of 1.5 gigabytes per second with special user-designed hardware.

The MP-1 high-speed I/O subsystem utilizes the same innovative technology that provides global communications within the PE Array. It performs massively parallel I/O with performance that scales with configuration size. It provides support for high-performance disk arrays with multiple gigabytes of storage, and is also designed to support frame-buffer graphics systems, and such external interfaces as FDDI and HPPI. The subsystem's open interface specification allows customer access to the MP-1's high-performance I/O for interfacing of custom I/O devices that operate at up to 1.5 gigabytes per second.

The MP-1 Family. The MP-1 Family consists of two series: the MP 1100 and MP 1200 series. Both series offer a range of performance, memory, I/O bandwidth, and processor communications. Both utilize two cabinets: a data-parallel unit consisting of the ACU, PE Array, PE Communications and high-speed I/O; and a UNIX subsystem consisting of a VAXstation 3520 ULTRIX workstation.

The MP 1100 Series includes three configurations, with 1,024, 2,048, or 4,096 processors. Similar configurations ranging up to 16,384 processors are available in the MP 1200 Series.

All members of the MP-1 Family have a footprint of less than seven square feet and fit into a

standard office environment. Running on 110 or 220 volt AC power, depending on configuration, MP-1 computers need no special facilities.

MP-1 Software

While the MP-1 Family uses many interconnected processing units to achieve high performance, it features a single, integrated software environment. Every element of this environment is geared toward fostering the use of data parallelism as an effective programming style.

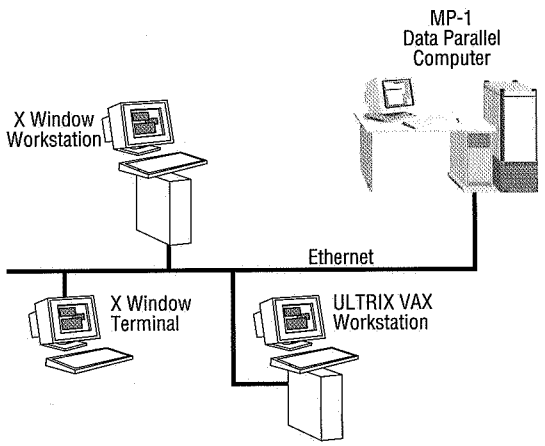
Standards. Wide use of standards provides a familiar setting for developing and running data-parallel applications:

- **Operating system.** The MP-1 software environment is based on the UNIX operating system. The choice of this standard as a foundation gives MP-1 users access to hundreds of utilities and applications.

- **User interface.** The MP-1 software environment is built around the X Window System. This windowing system allows MP-1 products to support graphics standards. It is fully integrated with the DECwindows System and supports access from X Window terminals and workstations.

- **Communications and networking.** The MP-1 software environment supports the protocols essential to a shared network environment. These include Ethernet, TCP/IP, NFS, uucp, BSD network services, and X Windows among others. MP-1 systems also accommodate users of DECnet.™

Network Computing Resource



- **Languages.** The MP-1 software environment supports mainstream programming languages, including ULTRIX C, and VAX Fortran. In addition, MasPar provides data-parallel programming languages that are full adaptations of C and Fortran.

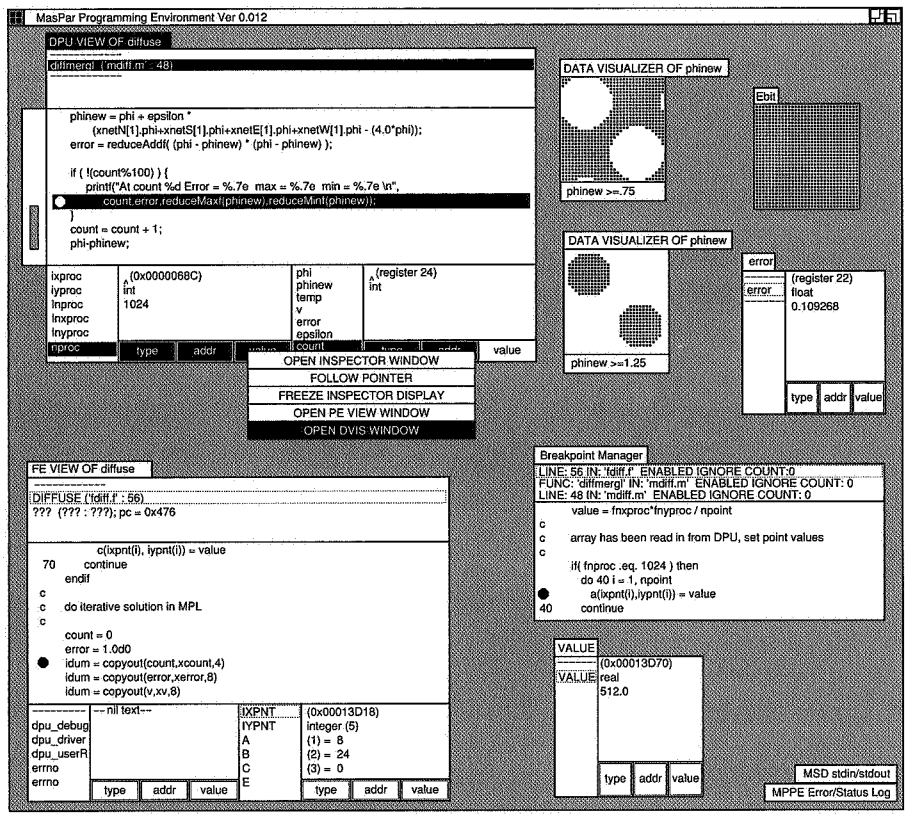
MasPar has made several innovative contributions to this familiar environment to provide special support for data-parallel programming. These include high-level programming languages and their compilers, and a comprehensive suite of development tools.

MasPar Languages. Built on the ANSI Fortran 77 standard language definition, MasPar Fortran (MPF) provides facilities to program in a high-level language and generate code for data-parallel execution. Optimization occurs automatically; the process is transparent. But for those developers interested in fine-tuning applications, MasPar also provides direct programming access to the data-parallel hardware through the MasPar Parallel Application Language (MPL).

MPF is based on the Fortran 77 ANSI standard with parallel and array extensions from ANSI Fortran 8X. The MPF optimizing compiler translates Fortran array and parallel code into optimized data-parallel machine code for execution on the PE Array. By integrating arrays into the language and providing control structures, these additions enable MPF to support arrays as first-class language elements. MPF allows a single program to direct operations of the UNIX Subsystem, the ACU and the PE Array as a single, seamless system.

The MasPar Parallel Application Language (MPL) provides the most direct form of access to the ACU and PE Array. A high-level language derived from C, MPL allows developers to explicitly control data placement and operations on the PE Array from the ACU.

MPL is useful for adapting UNIX applications for data-parallel execution in a simple, incremental fashion. The conversion process involves recoding parallel data structures and subroutines for data manipulation while leaving the remainder of con-



The MPPE Screen During a Debugging Session.

trol, user interface, and non-compute-intensive code intact. The resulting MPL code integrates with existing ULTRIX C or VAX Fortran programs through subroutine calls.

MasPar Programming Environment. Known as MPPE,™ this comprehensive set of development tools provides tremendous assistance to programmers of data-parallel code. It brings unprecedented ease-of-use to the process of programming, adapting and debugging large, compute-intensive, massively parallel applications.

MPPE provides a single context for software development, integrating myriad steps into a fluid, graphical process. It is an easily learned environment, featuring an interactive, window-oriented, point-and select interface. Built on the X Window standard, it can be used from any networked terminal or workstation that supports this protocol.

The MPPE makes it easy to visualize, analyze and optimize the behavior of MP-1 processors, algorithms and data. The tools are used as code is written or ported, edited, compiled, debugged, and tuned. These tools include a symbolic debugger, a machine animator, and data visualizers, among others.

- **Symbolic Debugger.** A tool for examining and controlling the execution of an application, MasPar's source-language symbolic debugger is *always available*. Programmers experience no lag while the debugger is loaded. Nor is there a need to restart a program after a problem is encountered. The debugger offers incremental access to program symbols as needed, with minimal startup overhead, and it also debugs optimized output from MasPar compilers without recompilation.

- **Visualizers and Animators.** The MPPE serves to open graphical "windows" into the behavior of large data structures and the processor array itself. A machine animator lets developers watch hardware activity during execution, providing immediate feedback on how well the program is utilizing the PE Array. A data visualizer allows programmers to examine large data arrays and watch data values change as the program executes.

Many other components of the MP-1 software environment further enhance the effectiveness of

the MPPE. The MasPar compilers were designed so the debugger can operate truthfully on optimized code. The symbol tables were also designed so that information for specific functions and subroutines can be retrieved incrementally. This feature provides greater efficiency by improving debugger initialization speed.

The MP-1 Family... For the '90s and Beyond

The MP-1 Family delivers a combination of assets that is unique in massively parallel computing:

- **Aggressive price/performance.** MP-1 computers have set a new standard for the cost of both integer and floating-point computation – $30 per MIPS and $600 per Mflop in the largest MP-1 configuration.

- **Outstanding programmability.** With its extensive set of graphical tools, the MasPar Programming Environment is designed to make data parallelism an intuitive programming style.

- **Vast computational power.** Its massively parallel design allows the MP-1 Family to match users' needs. The MP-1 computational unit ranges from 1,024 to 16,384 processors and offers up to 30,000 MIPS and 1,500 Mflops.

- **Long-term growth path.** The MP-1 Family architecture protects your investment through its scalability to greater numbers of processors, memory, and I/O to accommodate larger and larger amounts of data.

- **Small footprint.** An MP-1 system readily fits into an office or lab setting. It uses 110- or 220-volt power and requires no special cooling.

These features are packaged into network-accessible systems of exceptional utility. MP-1 systems deliver high-performance computation in the familiar context of industry-standard software and an interactive, graphical point-and-select user interface.

Together, these assets will make it easier to use data-parallel computing to unlock new levels of productivity and performance.

MasPar Computer Corporation
749 North Mary Avenue
Sunnyvale, California 94086
408-736-3300
FAX: 408-736-9560

PL006.0190