# The Massively Parallel Processor: Architecture and Application

John E. Dorband

NASA/Goddard Space Flight Center
Image Analysis Facility/Code 635
Greenbelt, MD 20771

## HISTORY

Born from ideas conceived during the mid 1970's at NASA's Goddard Space Flight Center, the Massively Parallel Processor[1] (MPP) was built to explore the application of parallel computing power to satisfy NASA's anticipated vast space and Earth science computing needs, in particular LANDSAT with its extremely large volume of image data. Goodyear Aerospace Corporation began construction of the MPP to government specification in 1979. At that time, no previous attempt had been made at a machine that possessed such a high degree of parallelism. The MPP was delivered to Goddard in May of 1983. In 1984, NASA's Office of Space Science and Applications (OSSA) issued a nationwide request[2] for proposals, seeking scientists interested in pioneering the use of massively parallel processing for scientific computation. A working group of 39 investigators was organized and began their work in October 1985. One year later, most of these investigators presented papers on their work at the **First Symposium on the Frontiers of Massively Parallel Scientific Computation**[3] held September 24-25, 1986 at Goddard. These papers and a subsequent report[4] written by the working group documented the extremely wide variety of applications that can effectively be processed on the MPP's architecture and permanently established massively parallel processing as a viable and effective manner of satisfying the seemingly overwhelming computational needs of the future.

## ARCHITECTURE

The MPP (Figure 1) consists of a 2 dimensional array of processors, a staging memory, and an array control unit (ACU). The array unit (ARU) consists of 16,384 processors arranged as a 128x128 square grid. Each processor can communicate with its four nearest neighbors on the mesh (to the north, south, east, and west). Each processor, or processing element (PE), is bit serial and has 1024 bits of memory. This gives the ARU an aggregate memory size of 2 megabytes.

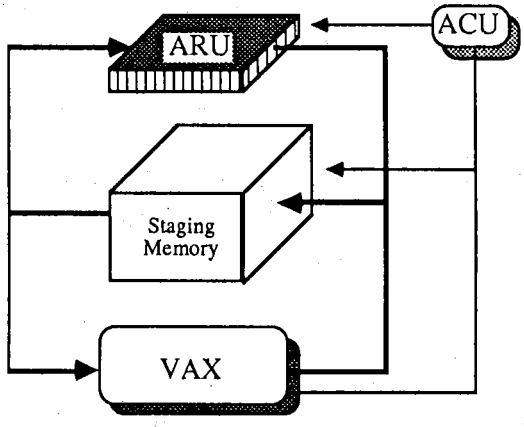


**Figure 1. MPP Organization.**

The staging memory, or stager, is an extremely versatile data reformatting mechanism. Its memory is divided into 16 banks with 2 megabytes per bank, giving it a total capacity of 32 megabytes. Data can be read in or out of the memory at 80 megabytes per second. It can simultaneously send data to and accept data from the ARU for an aggregate I/O rate of 160 megabytes per second. The ACU consists of 3 control units, the I/O control unit (IOCU), the processing element control unit (PECU), and the main control unit (MCU). The IOCU controls the data flow between the staging memory and the ARU memory. The PECU is a microcoded control unit that sends instructions and memory addresses to the PEs. The main control unit has a typical microcomputer instruction set and sends I/O requests to the IOCU and array instruction requests to the PECU.

A PE (Figure.2) is a bit serial processor. It has access to 1024 bits of nonshared memory. It contains 6 single bit registers, a 30 bit variable length shift register, a full adder, a logic unit, and routing connections to its four nearest neighbors.

The C register holds the carry output of the full adder. The A and P register are the inputs to the full adder. The B register is the output of the full adder and the input to the shift register. The shift register is used to speed up multiply, divide, floating point alignment and floating point normalization.

The P register is also an integral part of the logic unit and the routing connections. Logic operations are performed between the contents of the P register and the input to the logic unit from the common data bus with the result being stored in the P register. The P registers of all the PEs are connected in a north-east-west-south, or NEWS, grid, therefore, to communicate between processors the plane of bits contained in the P registers

slides to the north, south, east, or west. This grid of registers can be connected at opposite edges as a cylinder, a spiral, or not at all.

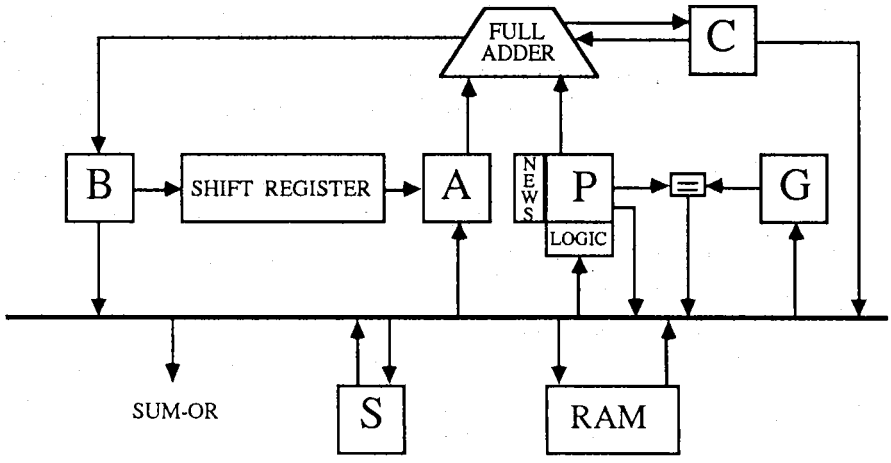


**Figure 2. Processing Element.**

The G register controls the masking of the processor during a masked instruction. This allows a processor to compute whether it will execute succeeding masked instructions. In a single instruction multiple data (SIMD) machine like the MPP, this is a very important capability because all processors receive the same instruction. Therefore the only options a processor can have other than performing the given instruction is to not perform it. This is important in such functions as sorting, generalized routing, and floating point arithmetic.

The S register is the means by which data is transferred to and from the staging memory. The S registers are connected to their neighbors to the east and west. The output of the stager is connected to the western edge of the processor array and the input to the stager is connected to the eastern edge of the processor array. Therefore, I/O is performed by sliding a bit plane (128 by 128 array of bits) in from the west or out to the east.

Two ways of using the stager's very versatile data reformatting mechanism are provided for the user though the *staging memory manager* and the *bit plane I/O* facilities. The staging memory manager treats the staging memory as a multidimensional array. This multidimensional array can be defined as having as many as 28 indices. The power of this facility is that an array may be read in with a specified ordering of the indices and read out with a different ordering. Bit plane I/O is a much simpler facility for using the stager. Bit plane I/O is based on the concept that the ARU memory can be viewed as 1024 bit planes of data. Each bit plane consists of 16,384 bits, one

bit per processor. Therefore, a bit plane is an array of 128 by 128 bits. Bit plane I/O simply treats the 32 megabytes of the stager as 16,384 bit planes of data external to the ARU memory, which can be randomly accessed and moved to and from any of the 1024 bit planes of the ARU memory.

The most interesting aspect of the MPP is obviously the array of processors, but the aspect of the MPP which really makes the efficient nature of the massively parallel architecture apparent is the parallelism of the MPP controllers. Each of the controllers of the MPP run asynchronously, whether it is the controllers of the staging memory or those of the ACU. This allows the convenient overlapping of different types of processing, for example, I/O and computation. Even within controllers there is parallel processing. The PECU has 8 registers, each of which can simultaneously be incremented or decremented. This is one reason why an array instruction is performed nearly every array clock cycle. This effectively reduces the overhead due to control to nearly zero.

But, what does this mean in terms of actual computational speed of the MPP? The MPP has a 100 nanosecond clock cycle time. This means, for instance, that it can do 163 billion bit operations per second, 55 billion boolean operations per second, 12 billion 4-bit adds per second, or 5 billion 4-bit multiplies per second. These operations are useful in such applications as cellular automata and image processing. In most scientific applications, floating point operations are very important. The MPP can perform 420 million floating point additions per second or 215 million floating point multiplies per second.

It is easily understood that the MPP is capable of large amounts of computations, since it has 16,384 operations going on at the same time. It is, however, not obvious that the MPP is at all useful in terms of large amounts of interprocessor communications, since each processor is only connected to its four nearest neighbors on the 2 dimensional grid of processors. What people fail to see is that 16,384 processors are communicating simultaneously. This gives a total aggregate data transfer rate in the array of 20 gigabytes per second. What is this massive communications bandwidth good for? Sorting can effectively use 50% of this bandwidth, 10 gigabytes/second. For instance, the MPP can sort 32,768 keys of 32 bits each in 13 milliseconds and 512K keys of 28 bits each in 1 second. No other machine to date claims to sort this fast.

## SYSTEM SOFTWARE

The MPP is a single user device. To allow multiple users access to the MPP, it has been attached to a VAX 11/780. Many developers can compile and run

their applications from a multiuser environment[5,6]. Because the MPP is a single user device, all users must wait in a queue to use it. There is also a debug/monitor(CAD)[7] program which is used to activate programs on the MPP and allows the user to perform typical debug functions such as examining memory, status, and setting breakpoints. The MPP has an assembler[8,9], and a high level language, MPP Pascal[10,11], which are used to generate executable code. It also has a FORTH environment that allows execution of MPP Parallel FORTH[12].

MPP FORTH is a dialect of UNIFORTH™ with parallel extensions added. The serial portion of MPP FORTH is as nearly identical to UNIFORTH™ as possible. The parallel functions of MPP FORTH consist of a set of serial functions that can logically be expected to be performed on a single PE, such as addition and some global functions that are inherently parallel, for example, interprocessor communications.

MPP FORTH includes two contexts: the default serial vocabulary (FORTH) and the parallel vocabulary (PARALLEL). One can easily switch back and forth between the serial and the parallel context through the use of '{' and '}'. The FORTH word '{' switches to the parallel vocabulary and '}' switches to the serial vocabulary. Thus, the expression " { + } + " means to first add the top two numbers on the data stack of each PE and then add the top two numbers on the serial data stack in the MCU.

Every PE has a data stack in its memory, but not a return stack, because programs are not run from PE memories, but from the MCU memory. Parallel variables can be stored either in the staging memory or the ARU (i.e., PE) memory as 128 by 128 arrays of elements. These elements could be 1-bit boolean values or multibit integer values. Therefore much of parallel programming in FORTH can be viewed as describing what one processor does -- it just happens to be happening on 16,384 processors.

To control a conditional expression, each processor computes a boolean value. Only processors for which the value is true process the code corresponding to the true condition, and subsequently, only processors whose value is false process the code corresponding to the false condition. This is done through the operation masking capability of each PE.

Global operations happen across all processors. These operations include such functions as global minimum (finding the minimum value from among the 16,384 top numbers on all the PE data stacks), global maximum, global OR, and the slide function which performs interprocessor communications.

## APPLICATIONS

With the ability to perform nonglobal, global, and conditional parallel operations one can start to develop a reasonably general variety of applications. Most applications on the MPP have been developed in MPP Pascal and MPP assembler, but several applications have been developed in MPP FORTH, these include the analysis of boolean delay equations, computer graphics generation by ray tracing, and pure LISP.

The Working Group, as mentioned earlier, is responsible for the majority of applications that now run on the MPP. These applications are from five main areas of science: physics, Earth science, signal & image processing, computer science, and graphics. The following list of projects are those being pursued by members of the Working Group. The ones with a * following the title were presented at the **First Symposium on the Frontiers of Massively Parallel Scientific Computation.**

### PHYSICS

Investigations on Electronic Structure and Associated Hyperfine Properties of Atoms and Condensed Matter Systems Using the MPP
*Dr. Tara Prasad Das, Department of Physics*
*State University of New York, Albany*

Numerical Calculations of Charged Particle Transport *
*Dr. James A. Earl, Department of Physics and Astronomy*
*University of Maryland, College Park*

Simulation of Beam Plasma Interactions Utilizing the MPP *
*Dr. Chin S. Lin, Southwest Research Institute*
*San Antonio, Texas*

Particle Simulation of Plasmas on the MPP *
*Dr. L.R. Owen Storey, STAR Laboratory*
*Stanford University, Stanford, California*

Phase Separation by Ising Spin Simulations *
*Dr. Francis Sullivan, Center for Applied Mathematics*
*National Bureau of Standards, Gaithersburg, Maryland*

A Study of Phase Transitions in Lattice Field Theories on the MPP *
*Dr. Peter Suranyi, Department of Physics*
*University of Cincinnati, Cincinnati, Ohio*

Wave Scattering by Arbitrarily Shaped Targets--Direct and Inverse
*Dr. William Tobocman, Department of Physics*
*Case Western Reserve University, Cleveland, Ohio*

Free-Electron Laser Simulations on the MPP *
*Dr. Scott Von Laven, Mission Research Corporation*
*1720 Randolf*
*Albuquerque, NM 87106*

The Dynamics of Collisionless Stellar Systems *
*Dr. Richard L. White, Space Telescope Science Institute*
*Baltimore, Maryland*

**EARTH SCIENCE**

Tropospheric Trace Gas Modeling on the MPP *
*Dr. Gregory R. Carmichael, Chemical and Materials Eng.*
*University of Iowa, Iowa City, Iowa*

Kalman Filtering and Boolean Delay Equations on the MPP
*Dr. Andrew Mullhaupt,*
*University of New Mexico, Albuquerque, NM*

Adapting a Navier-Stokes Code to the MPP *
*Dr. Chester E. Grosch, ICASE*
*NASA/Langley Research Center, Hampton, Virginia*

A Physically-Based Numerical Hillslope Hydrological Model with Remote Sensing Calibration *
*Dr. Robert J. Gurney, Hydrological Sciences Branch*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

A Comparison of the MPP with Other Supercomputers for Landsat Data Processing *
*Mr. Martin Ozga, Statistical Reporting Service (SRS),*
*USDA, Washington, DC*

Use of Spatial Information for Accurate Information Extraction *
*Dr. James C. Tilton, Information Analysis Facility*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

A Magnetospheric Interactive Model Incorporating Current Sheets (MIMICS)
*Mr. Elden Whipple, Center for Astrophysics & Space Sciences*
*University of California at San Diego, La Jolla, California*

**SIGNAL & IMAGE PROCESSING**

Fixed Point Optimal Nonlinear Phase Demodulation
*Dr. Richard S. Bucy, Department of Aerospace Engineering*
*University of Southern California, Los Angeles, California*

Pattern Recognition on an Array Computer
*Dr. Y. Paul Chiang, Dept. of Electrical and Computer Eng.*
*Washington State University, Pullman, Washington*

Graphic Applications of the MPP *
*Dr. Edward W. Davis, Department of Computer Science*
*North Carolina State University, Raleigh, North Carolina*

Automatic Detection and Classification of Galaxies on "Deep Sky" Pictures *
*Dr. Sara Ridgway Heap, Astronomy Branch,*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

Application of Parallel Computers to Biomedical Image Analysis
*Dr. Robert V. Kenyon, Dept. of Engineering and Computer Science*
*University of Illinois, Chicago, Ill 60680*

Comet Halley Large-Scale Image Analysis
*Dr. Daniel A. Klinglesmith, III, Science Operations Branch*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

Synthetic Aperture Radar Processor System Improvements
*Dr. Stephen A. Mango, Code 5381MA*
*Naval Research Laboratory, Washington, DC*

Development of Automatic Techniques for Detection of Geological Fracture Patterns
*Dr. H.K. Ramapriyan, Information Analysis Facility*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

Development of an Improved Stereo Algorithm for Generating Topographic Maps Using Interactive Techniques on the MPP *
*Dr. James P. Strong, Information Analysis Facility*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

Reconstruction of Coded-Aperture X-Ray Images *
*Dr. Lo I. Yin, Solar Physics Branch*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

COMPUTER SCIENCE

Diagrammatic Information-Processing in Neural Arrays *
*Dr. John A. Barnden, Computer Science Department*
*Indiana University, Bloomington, Indiana*

Sorting and Signal Processing Algorithms: A Comparison of Parallel Architectures
*Dr. Howard B. Demuth, Dept. of Electrical Engineering*
*University of Idaho, Moscow, Idaho*

Parallel Algorithms for Graph Theoretic Problems
*Dr. Susanne E. Hambrusch, Computer Science Department*
*Purdue University, West Lafayette, Indiana*

Applications on Stochastic and Reaction-Diffusion Cellular Automata *
*Dr. Harold M. Hastings, Department of Mathematics*
*Hofstra University, Hempstead, New York*

Sorting and Signal Processing Algorithms: A Comparison of Parallel Architectures
*Dr. Martin T. Hagan, School of Electrical and Computer Engineering*
*Oklahoma State University, Stillwater, Oklahoma*

Solution of Complex, Linear Systems of Equations *
*Dr. Nathan Ida, Electrical Engineering Department*
*University of Akron, Akron, Ohio*

FORTH, an Interactive Language for Controlling the MPP *[12]
*Dr. Daniel A. Klinglesmith, III, Science Operations Branch*
*NASA/Goddard Space Flight Center, Greenbelt, Maryland*

Simulating an Applicative Programming Storage Architecture Using the NASA MPP *
*Dr. John T. O'Donnell, Computer Science Department*
*Indiana University, Bloomington, Indiana*

Parallel Solution of Very Large Sparse Linear Systems *
*Dr. John H. Reif, Computer Science Department*
*Duke University, Durham, NC 27706*

Massively Parallel Network Architectures for Automatic Recognition of Visual Speech Signals
*Dr. Terrence J. Sejnowski, Biophysics Department*
*Johns Hopkins University, Baltimore, Maryland*

**GRAPHICS**

Space Plasma Graphics Animation *
*Mr. Eugene W. Greenstadt, Space Sciences Department*
*TRW, Redondo Beach, California*

Algorithmic Commonalities in the Parallel Environment *
*Dr. Michael McAnulty, Department of Computer & Information Sciences*
*University of Alabama, Birmingham, Alabama*

Animated Computer Graphics Models of Space and Earth Sciences Data Generated via the MPP *
*Mr. Lloyd A. Treinish, Data Management Systems Facility*
*NASA, Goddard Space Flight Center, Greenbelt, Maryland*

## UNUSUAL APPLICATIONS OF THE MPP

The MPP is a very regular arrangement of processors in terms of how they are interconnected: a 2-dimensional array. Many applications have previously *appeared* extremely difficult to map onto such an arrangement. Most applications attempted by the Working Group are fairly easy to map to a 2-dimensional array because they spend only a small amount of execution time communicating between processors. This does not preclude, however, executing applications which require a much larger fraction of execution time for communications. In the extreme case, the application mapping is dependent on the dynamic state of specific data values that are being processed.

Examples of applications involving irregular arrangements of data, are artificial intelligence (LISP) and graphic generation (ray tracing). In these applications the data is distributed across processors, possibly arbitrarily, but data items must be brought together at times during the execution. Typically, data rendezvous is done by generalized routing. Each data item is assigned to a specific processor. Thus, if data is needed for a computation in another processor, the data is explicitly routed from the source data's processor to the processor where the data is to be used. Therefore the source data must have available the address of the processor where it is to be moved, and the array of processors must have the appropriate hardware and software to move the data from the source processor to the destination processor.

Since the MPP has neither generalized routing hardware nor software, a different approach has been taken. This approach is more akin to virtual routing than location-dependent (physical) routing and is based on a modified version of sorting called *sort computation*. In physical routing, data is assigned to physical processors, or to virtual processors that are assigned to physical processors. In virtual routing, data is assigned to records which are not tied to specific processors. Therefore, records are free to migrate to where they can obtain or exchange data, independent of hardware configuration. This may seem impossible since the data does not explicitly know where it should go to be used, because no data reside at predefined physical locations in the array of processors. This problem is resolved by grouping records according to an internal key value. Sorting is used to migrate records of a group past each other and acts on all groups simultaneously.

Physical routing is intuitively easier to understand than virtual routing, since data is routed from a specific location to another specific location. Virtual routing allows data to move from one record to another, by giving them the same key value and effectively placing them in the same group of records, but it also allows more complex things to be done such as accumulating a value from all the records of a group into one record of the group, *sort aggregation*, or distributing the value of a record in a group to all records in that group, *sort distribution*, all without tying data to specific processors. This also means that records of data can be created and deleted almost at will and garbage collection is nearly trivial. Sort computation is accomplished by modifying the comparison routine of the sort, in such a way that it can be proven[13] that all records of each group are brought together. Note that the communication between records is done during the sort, so it is unimportant whether records of the same group reside in physically adjacent processors after the sort. Thus, faster sorts that leave records in shuffled order may be used.

Given the ability to group records according to keys and perform aggregation and distribution operations on the records of each group, applications have been developed that require the processing of irregular arrangements of data. Computer graphic generation by ray tracing and the implementation of pure LISP are examples of applications that require this capability. The pure LISP is implemented in MPP FORTH by distributing the pointer pairs that make up the LISP data structure across the processors of the MPP. Sort computation is used to bring the pointer pairs together according to the functions that must be performed on them, such as the creation of a new pointer. The basic functions of pure LISP were implemented (i.e., CAR, CDR, CONS, EQ, ATOM, COND, APPLY, EVAL, EVLIST, and LAMBDA). The MPP ray tracing approach[14] is based on an algorithm that finds the intersections of light rays and objects in a 3-dimensional space. It is done by recursively subdividing space. Records are created that keep track of whether a specific ray or object intersects a subdivision of space. If a subdivision of space is not intersected by both a ray and an object, all records associated with it are deleted. Sort computation is used to determine where this condition is true. These two applications have been implemented on the MPP using MPP FORTH.

## CONCLUSION

The MPP has a simple, yet elegant architecture. Some have granted that it is novel, and others have claimed it to be fundamental. Whichever is the case, it has shown itself to be both useful and practical. Its hardware embodiment consumes relatively low power compared to other supercomputers, does not require special cooling, has an instruction cycle time that is relatively long, thus allowing it to avoid high speed technologies, and yet it competes respectably with any currently available supercomputer in speed of execution of applications. Due to the efficiency of the architecture, ease of construction and variety of applications, it is conceivable that within a few years computational accelerators more powerful than the MPP will be available at reasonable cost to be placed inside desktop workstations.

## REFERENCES

All MPP documents, exclusive of [1] may be obtained from the MPP User Support Office, Code 635, NASA/Goddard Space Flight Center, Greenbelt, MD 20771.

[1] *The Massively Parallel Processor*, J.L. Potter, ed., ISBN: 0-262-16100, MIT Press, 1985.

[2] NASA Space Science & Applications Notice, *Computational Investigations Utilizing the Massively Parallel Processor*, December 1984.

[3] *Proceedings of the First Symposium on the Frontiers of Massively Parallel Scientific Computation*, J. Fischer, ed., NASA Conference Proceedings 2478, 1987.

[4] *Report from the MPP Working Group to the NASA Associate Administrator for Space Science and Applications*, J. Fischer, C. Grosch, M. McAnulty, J. O'Donnell, O. Storey, eds., NASA Technical Memorandum 87819, 1987.
[5] *Computing on the MPP at the Goddard Image and Information Analysis Center*, February 1986.
[6] *MPP User's Guide*, February 1986.
[7] *Control and Debug (CAD) User's Manual*, GER 17142, May 1983.
[8] *MPP Main Control Language - MCL*, September 1985.
[9] *MPP PE Array Language - PEARL*, January 1986.
[10] *MPP Pascal User's Guide*, September 1986.
[11] T. Busse, Opsahl, T., Abeles, J., "MPP Pascal: Mapping a Language to the Machine."
[12] *MPP Parallel FORTH User's Guide*, J.E. Dorband, September 1986.
[13] J.E. Dorband, *Sort Computation and Conservative Image Registration*, Ph.D. Thesis, Department of Computer Science, Pennsylvania State University, December 1985.
[14] J.E. Dorband, *3-D Graphic Generation on the MPP*, Proceedings of the 2nd International Conference on Supercomputing, Vol. II, pg 305-309, 1987.