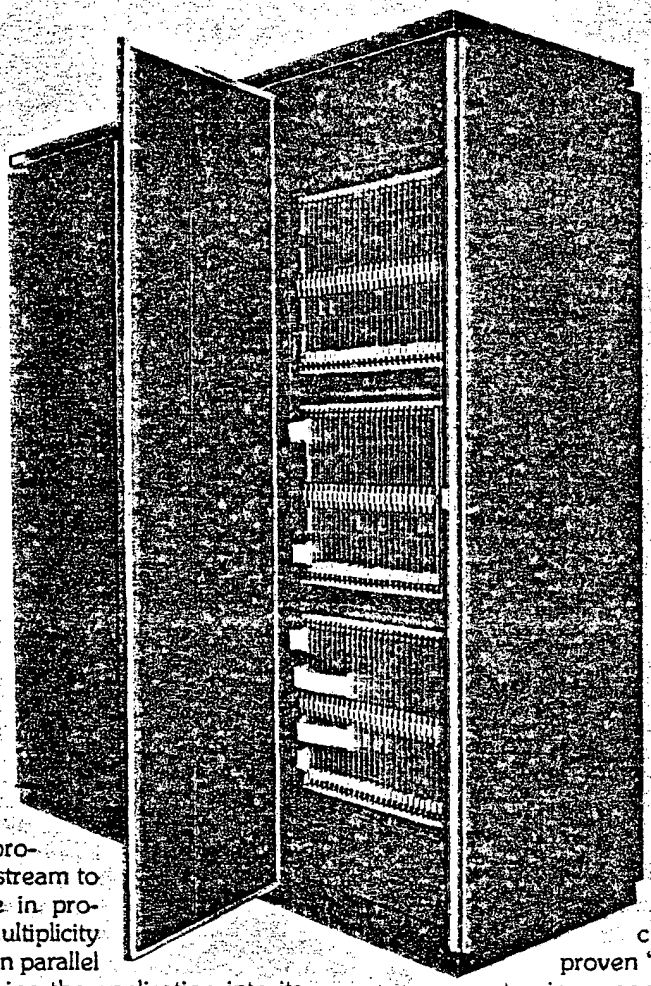
Attachment B

Heterogeneous Element Processor

Denelcor, Inc.

Tomorrows' Computer Is Here . . . Today

Denelcor's Heterogeneous Element Processor (HEP) is a large-scale (64-bit) high-speed digital computer whose architecture makes all other supercomputer architecture obsolete. HEP provides a totally new computing environment: high-speed, parallel processing of heterogeneous data elements. HEP has been designed for use in scientific and/or commercial applications which can effectively utilize processing speeds of ten million to 160 million instructions per second. HEP achieves this throughput because of its design which implements the Multiple Instruction Stream Multiple Data Stream (MIMD) architectural concept for the first time in a commercially available system.



HEP makes available to the user up to 1,024 independent instruction streams or processes, each with its own data stream to be used concurrently for use in programming applications. This multiplicity of instruction streams running in parallel enables and encourages breaking the application into its component parts for parallel processing. Other features of the HEP design provide the synchronization necessary to facilitate cooperation between concurrent processes, and eliminates the precedence delays which often occur when parallel processing is attempted using more conventional data processing equipment. An equal number of Supervisory Processes are available for processing the privileged functions necessary to the support of the User Processes for a total of 2,048 independent instruction streams.

The many capabilities of the HEP hardware are fully supported by HEP System Software so that the potential performance of the system is realized with relative ease. Using the available System Software, programming HEP is very similar to programming a conventional system, and only minimal additional programmer training is required.

In addition to the obvious design goals of fast throughput and the ability to solve very large and complex problems, HEP is designed for ease of operation and to be highly effective across the full range of general-purpose computing applications.

HEP hardware is modular and field expandable.

HEP achieves its high speed performance through advanced architectural concepts rather than through unproven "leading edge technology" electronic components. This provides the user benefits in economy and reliability.

HEP Parallel Fortran is designed for maximum similarity to existing languages, with logical extensions as necessary to implement the advanced features of HEP.

HEP is designed for ease of maintenance in the event of hardware malfunction. Maintainability features are an integral part of the hardware design, including an on-board maintenance diagnostic system which implements an Interactive Maintenance Language for diagnostic purposes.

Heterogeneous Element Processor

Denelcor, Inc.

Evolution of Computer Architecture

The earliest computers executed a single instruction at a time, using a single piece of data. The architecture of these machines, called SISD (for Single Instruction, Single Data Stream) computers, was straight-forward, and well suited to the technology of the times. As technology advanced and computer users required greater performance, SISD machines were made faster and faster, using newer and better components and designs. But a fundamental problem remained. Although the execution of a computer instruction is physically composed of several parts — instruction fetch, operand fetch, execution and result store — the SISD computer could only perform one of these at a time, since each step depended on the completion of the previous one. Thus, three-fourths of the expensive hardware stood idle at any given time, waiting for the rest of the hardware to finish operation.

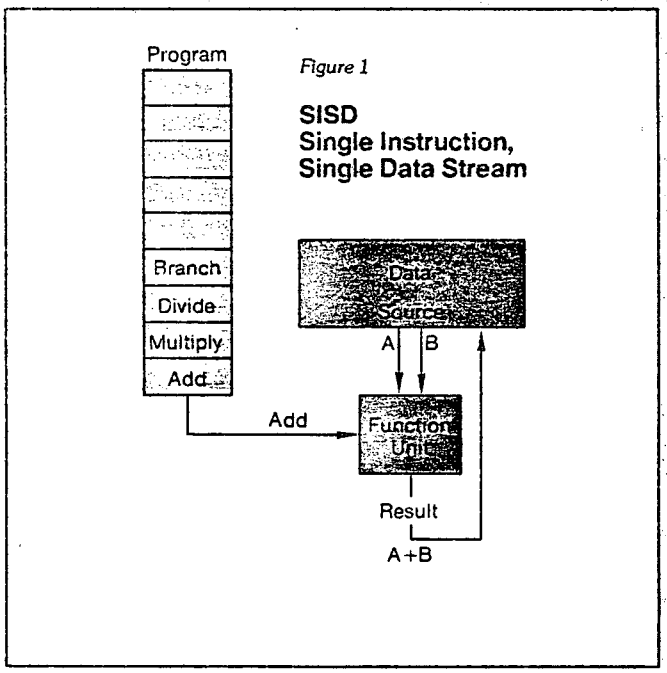


Figure 1

SISD
Single Instruction,
Single Data Stream

SISD designers attempted to remedy this by a technique called "look-ahead", in which instruction fetch for the next instruction was overlapped with some portion of the execution of the current instruction. This provided some performance improvement. However, digital computer programs, particularly those written in higher level languages, contain large numbers of test and branch instructions, in which the choice of the next instruction depends on the outcome of the current instruction. In such cases, "look-ahead" offers no speedup, and introduces substantial complexity to make sure that the partial execution of an incorrect next instruction does not contaminate the computation.

Another approach to increasing the speed of computation was to make multiple copies of portions of the SISD hardware. In this approach, called SIMD (for Single Instruction, Multiple Data Stream), the operand fetch, execution and result store portions of the hardware were replicated, so that the execution of a single instruction caused several values to be fetched, computed upon and the answers stored. For certain problems, this provided a substantial performance improvement. With sufficient hardware, entire vectors of numbers could be operated upon simultaneously. However, as with "look-ahead" SISD machines, the occurence of test and branch instructions, among others, required the machine to wait for the total completion of the instruction before proceeding. The test and branch itself could make no use of the replicated hardware.

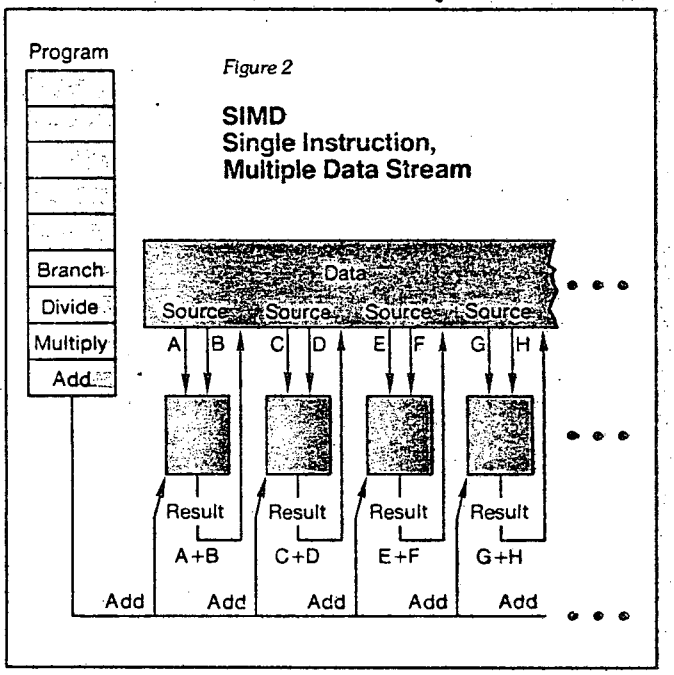


Figure 2

SIMD
Single Instruction,
Multiple Data Stream

In addition, two new problems were created by the SIMD architecture. Substantial portions of most programs are not vector-oriented. The computation of iteration variables and array subscripts is a scalar problem, for which SIMD offers no speedup, and the collection of operands across arrays is an addressing problem which many SIMD architectures do not handle. As a second problem, if an SIMD computer has a fixed quantity of replicated execution modules (adders, etc.), and if the length of the vector which the user wishes to operate on differs from the vector length of the machine, performance suffers and software complexity increases. The cost of computation remains high since the hardware is often not fully utilized.

Heterogeneous Element Processor

Denelcor, Inc.

Evolution of Computer Architecture

Continued difficulties with the implementation of high performance, cost effective computation using single instruction machines have led to the development of a new concept in computer architecture.

This concept, called MIMD (for Multiple Instruction, Multiple Data Stream) architecture, achieves high performance at low hardware cost by keeping all processor hardware utilized executing multiple parallel programs simultaneously. For example, while an add is in progress for one process, a multiply may be executing for another, a divide for a third; or similar functions may be executing simultaneously, such as multiple adds or divides. In MIMD architectures, cooperating programs are often called "processes". Independent programs may contain one or several processes.

processes. Since this arbitration of the state of memory locations is handled by hardware and without affecting the execution of unrelated instructions, the communication delay is short and the overhead is small.

MIMD computers may be used to execute either SISD or SIMD programs. SISD programs are just MIMD programs with no inter-program communication. Execution of multiple identical MIMD programs is equivalent to execution of an SIMD program.

In the SIMD case, MIMD computers may match the vector lengths exactly, while using remaining resources for unrelated computation. Thus, high efficiency may be maintained even through scalar portions of the code. But the major application of MIMD computers lies in problems of

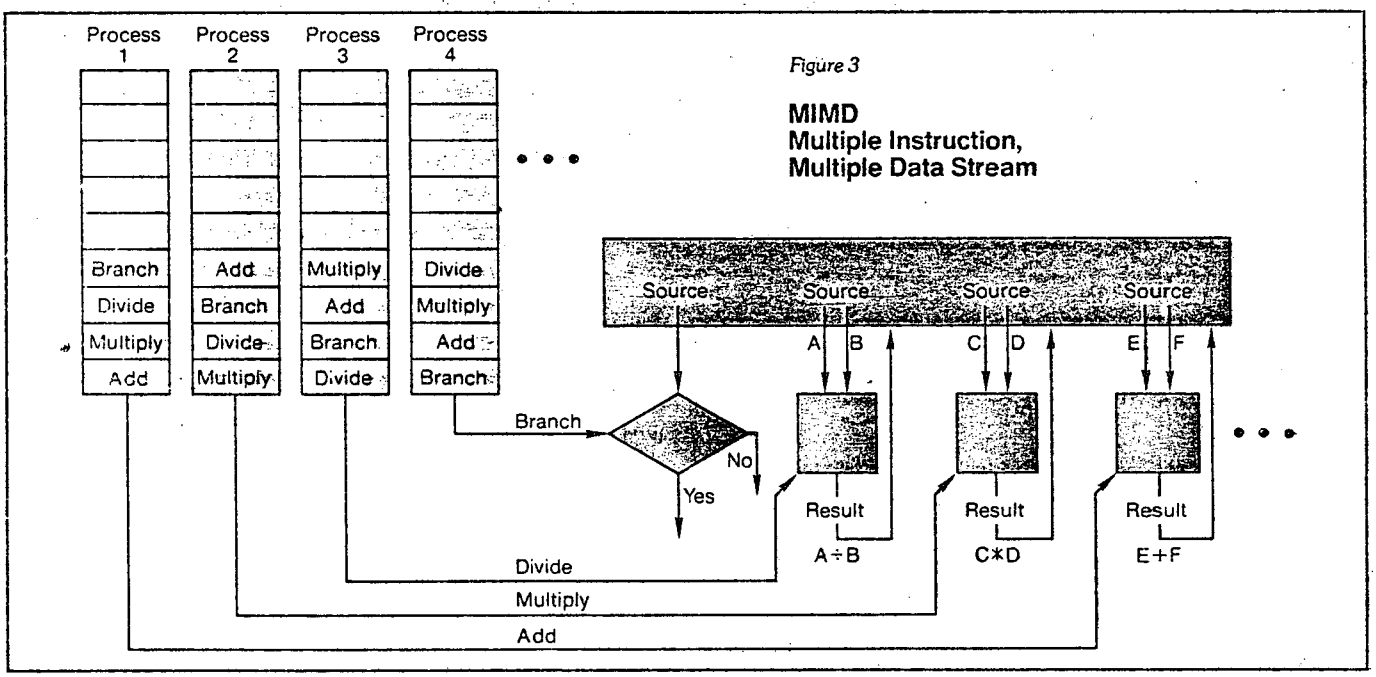


Figure 3

MIMD
Multiple Instruction,
Multiple Data Stream

Because the multiple instructions executed concurrently by an MIMD machine are independent of each other, execution of one instruction does not influence the execution of other instructions and processing may be fully parallel at all times.

Successful MIMD architectures (figure 3) also provide low-overhead mechanisms for inter-process communication. In these architectures, data locations may contain not only a value but a state. Processes may synchronize by waiting for input locations to have the "full" state. Result storage may wait for output locations to attain the "empty" state resulting from consumption of their contents by other

sufficient complexity that straightforward vector computation is not feasible. In these cases, which include continuous simulation and complicated partial differential equation solutions, MIMD architecture offers the only possible method of achieving significant parallelism. Denelcor's Heterogeneous Element Processor system is the only commercially available MIMD computer.

Heterogeneous Element Processor

Denelcor, Inc.

HEP Architecture

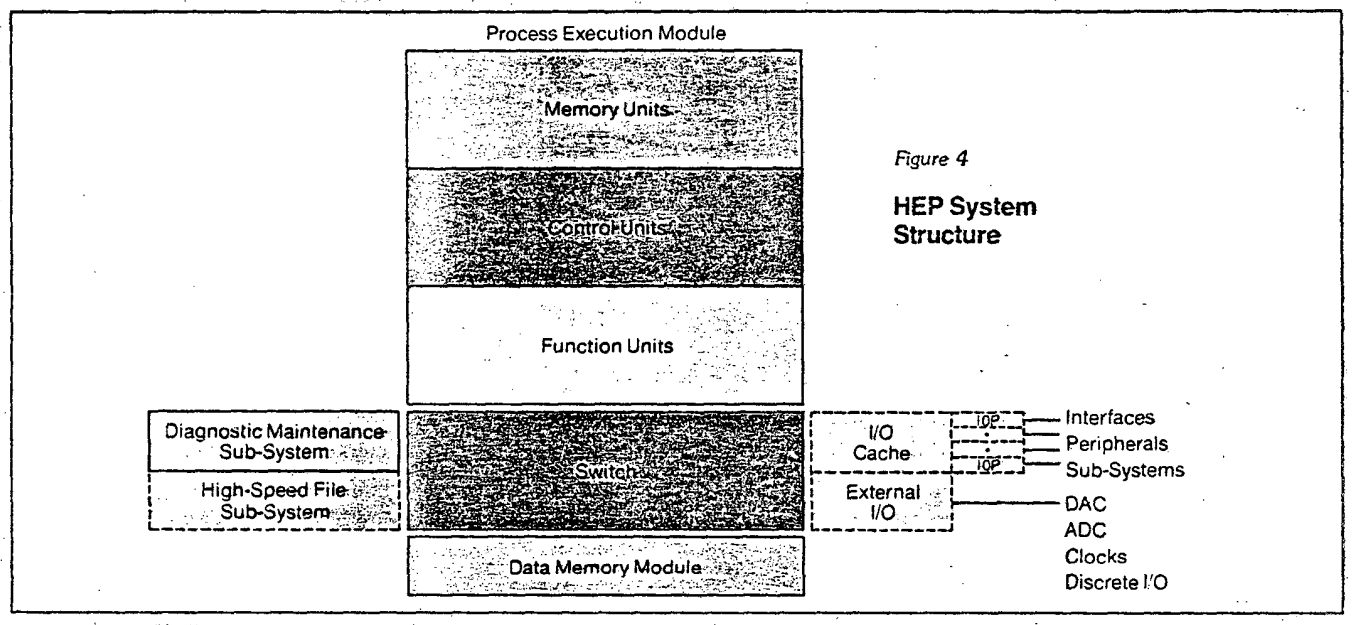


Figure 4

HEP System Structure

The HEP computer system consists of process execution modules (PEMs), data memory modules and support processors interconnected by a high-speed data switch network. All data memory modules are accessible by all PEMs. Thus, processes executing in parallel in one or several PEMs may cooperate by reading and writing shared information in the data memories. Parallel processes synchronize and pass information back and forth using the full/empty attribute of each data memory location. HEP instructions may automatically wait for an input data memory location to be full before execution, and leave the location empty after execution. Instructions may also wait for an output location to be empty before execution and leave it full after execution. This communications discipline allows processes to conveniently and unambiguously pass information to other processes while executing. The full/empty attribute ensures that reads and writes of inter-process variables will alternate and no information will be lost. For locations used exclusively within a process, the full/empty attribute is ignored and memory may be accessed conventionally.

Both normal and synchronized memory access are available to the Fortran programmer as well as the assembly programmer. Software modules in both Fortran and assembler programs may be distributed across several PEMs to achieve increased throughput. In general, design of a parallel program is not affected by whether the program will run in one or several PEMs.

In HEP, creation and termination of parallel processes in an MIMD program is a hardware capability directly available to the programmer. Processes are created or terminated in 100 nanoseconds by execution of a single HEP instruction. Thus, processes may be created at any point in a program where additional parallelism is required, and terminated as soon as their function is accomplished. Up to 64 user processes may exist simultaneously within each PEM in a HEP system.

In order to efficiently manipulate data, each PEM contains 2048 internal general purpose registers. PEMs automatically detect and flag normal arithmetic errors (overflow, underflow, etc.) and may generate traps on occurrence of these errors. Programs in a HEP system are protected from each other and relocated in memory by a set of relocation/protection registers in each PEM. This allows multiprogramming in a HEP system with full isolation of one user from the next.

All data and instruction words in a HEP are 64 bits long, although PEM data memory reference instructions allow partial word and byte addressing. The memory bandwidth in a HEP system is 20 million words/second per PEM, including the data switch network. Each PEM executes up to 10 million instructions per second. The architecture of the switch network allows up to 128 memory modules of up to one million words each and up to 16 PEM's. This range of system configurations results in speeds up to 160 million instructions per second on 64 bit data and memory sizes up to one billion bytes.

Heterogeneous Element Processor

Denelcor, Inc.

HEP Architecture & Software

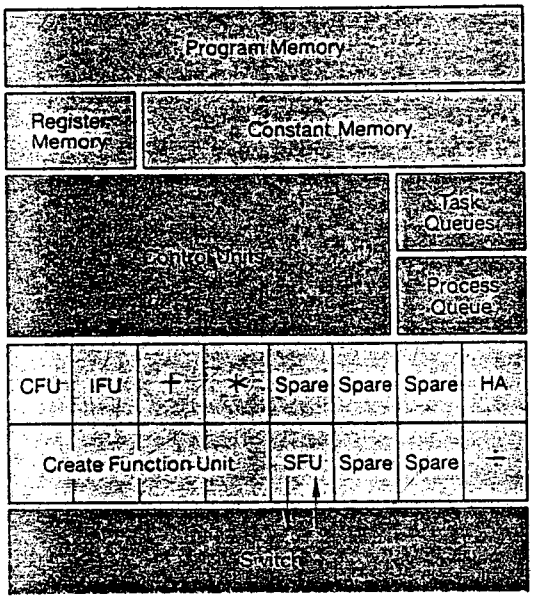


Figure 5

Process Execution Module Structure

HEP systems may include high-speed real-time I/O devices connected to the data switch network. These devices operate at memory speeds up to 80 million bytes/second. Normal I/O devices are connected to the HEP system through support processors. Thus, standard commercial I/O devices and controllers may be used for routine I/O functions. All standard I/O devices are accessible through the HEP operating system and Fortran I/O.

HEP Software

HEP systems support a batch operating system with Fortran and assembler programming languages. The HEP operating system provides input and output spooling, batch job scheduling, and full operator control of the system.

HEP Fortran is an extended ANSI Fortran IV with added parallel capabilities. The Fortran programmer has access to all standard Fortran formatted and unformatted I/O capabilities. In addition to the relaxation of syntax common to many Fortran compilers, HEP Fortran provides the programmer with the means for explicit parallel programming. A math library is also available which generates parallelism in the evaluation of known functions.

The HEP Assembly Language allows the user to access all of the capabilities of the system in an efficient manner. HEP Assembly Language subroutines may be included in a Fortran job to improve the efficiency of certain heavily used sections of code. Assembly Language programs have direct access to all hardware capabilities, including the direct creation and termination of arbitrary processes.

The HEP Link-Editor binds programs and subroutines into processes, tasks, and jobs. The input is from either HEP Fortran or HEP Assembler. The output is HEP machine executable code which is input to the HEP Loader at execution time. The HEP Link-Editor runs as a user job in the HEP PEM.

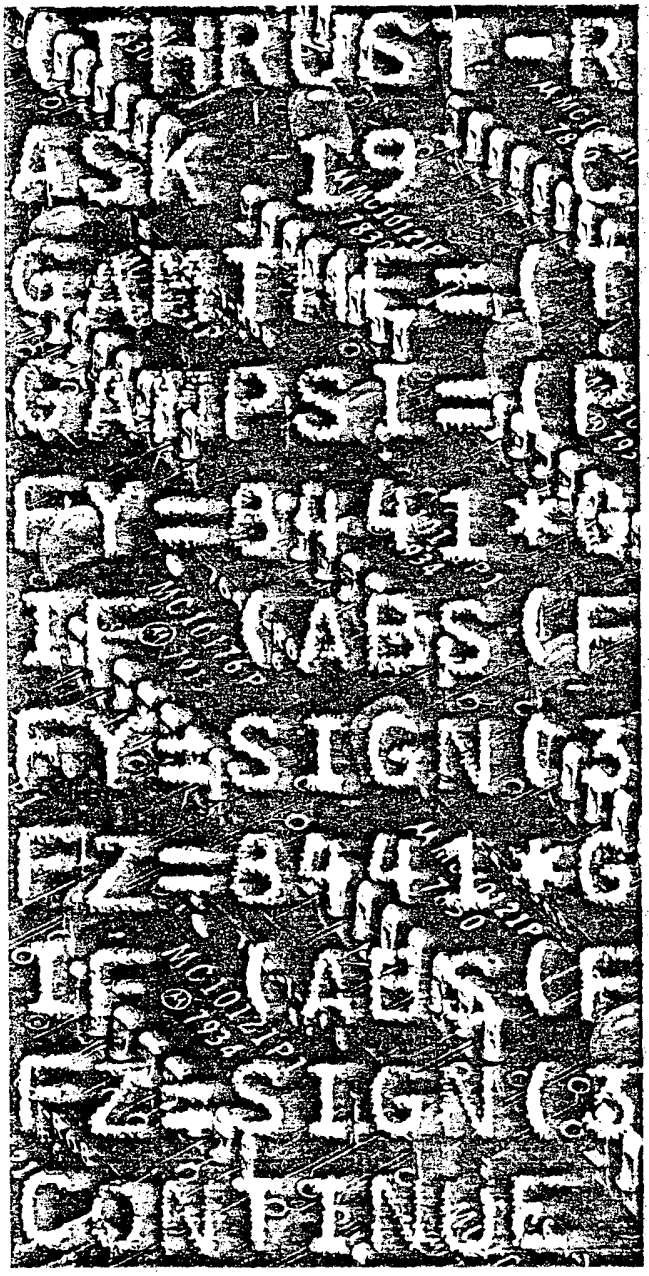
The HEP File System provides a large volume, high data rate I/O capability via the HEP Switch to a HEP System with multiple Process Execution Modules (PEM). Sequential access to information stored in multiple moving-head disk files is provided to the system at data rates from 80 megabytes per second (the maximum input rate for the switch), to approximately 1 megabyte per second (the rotating storage data rate). Random access to information is provided with comparable bandwidth, depending on the logical file size and the access patterns.

The HEP Interactive Maintenance Language (IML) provides a sophisticated yet easy-to-use language for debugging the HEP System. It is used in conjunction with maintenance hardware, wither test slots in the HEP main frame or off-line test fixtures. The language is procedure-oriented, thus permitting complex functions to be coded into higher order procedures.

Heterogeneous Element Processor

Denelcor, Inc.

HEP Applications



The most obvious area of HEP application is the multiprogramming of ordinary SISD algorithms. This application does not use the inter-process communications features of HEP, but fully utilizes its computing capacity. Since HEP's parallel architecture allows more complete utilization of its hardware, the cost effectiveness of HEP multiprogramming is higher than for other machines of comparable performance. Another benefit of HEP's effectiveness at conventional computation is that it can easily run all jobs at a facility — not just those which are sufficiently large or important to be written in parallel.

The application for which HEP was originally designed was the solution of systems of ordinary differential equations, such as those describing flight dynamics problems. In these problems, a substantial system of dissimilar equations must be solved, often in real-time. Many of the functional relationships in the equations are empirically derived and must be repetitively evaluated by multi-dimensional interpolation in lookup tables. Historically, such problems could only be solved, with limited precision and great expense, using analog computers. The HEP MIMD architecture is the first commercially available digital technology capable of effectively addressing these problems.

Another application area well suited to HEP is the solution of partial differential equations describing continuous media. These equations, which occur in fluid dynamics and heat transfer problems, are typically modelled using a grid of lattice points within the continuous medium. The behavior at a point is a function of the values at its neighbors. The HEP's architecture allows these problems to be solved with full parallelism, even in the presence of irregular or time-varying lattice geometry, or with complex functional relationships between lattice points.

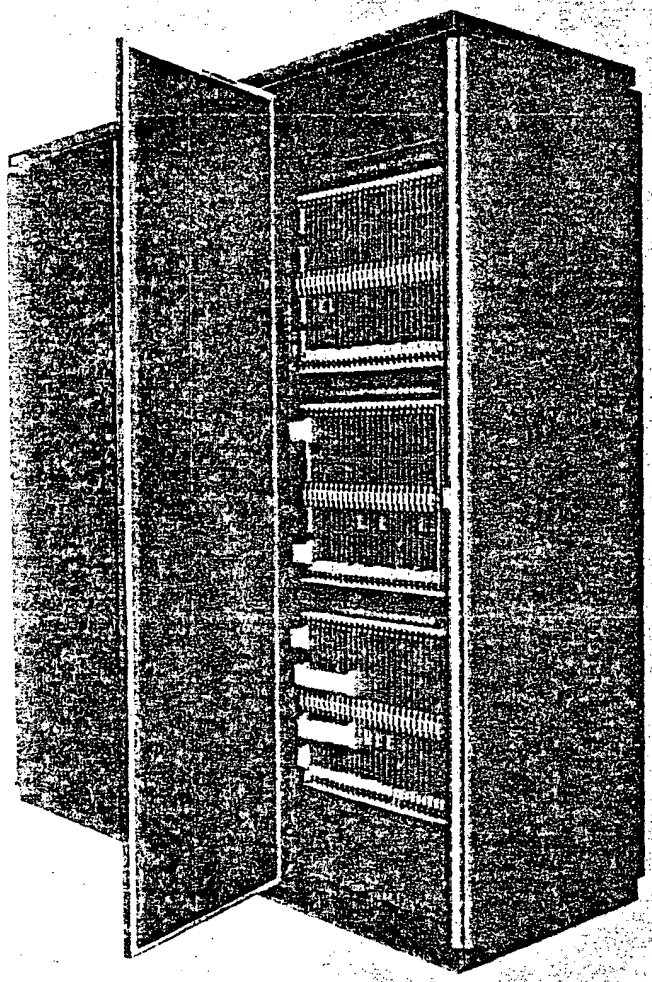
A fourth, and very general, application area for HEP is that class of problems for which a large number of discrete elements must be modeled or computed upon. Examples of such problems are tree and table searches, multiparticle physics problems, electric power distribution, and fractional distillation simulations. In all cases, complex behavior at a number of sites must be modeled, and interaction between the sites is critical to the result. Such problems are easily solved on the HEP.

The computing requirements for each of these applications are different. To effectively supply the range of capabilities needed, the HEP system is available in several configurations.

Heterogeneous Element Processor

Denelcor, Inc.

HEP Summary



HEP's building block architecture offers total flexibility enabling the user to start with the exact amount of computer power needed. As computing requirements grow, HEP's field-expandability allows the user to easily and economically add hardware and software modules to accommodate the largest of applications. These advanced features clearly place HEP in the forefront of digital computer technology and provide strong competition for existing computer systems, both scalar and vector.

The evolution of HEP is a natural result of Denelcor's on-going commitment to meet the market needs with state-of-the-art, high-quality systems.

- 10 Million to 160 Million Instructions per Second, Scalar or Vector.
- 2,048 to 32,768 General Purpose, 64-bit Registers
- 262,000 to one Billion Bytes Memory Capacity
- Parallel Computing in Fortran
- Fail-Soft Architecture
- Unbounded I/O Rates
- Real-Time Synchronized

Denelcor, Inc.
3115 East 40th Avenue
Denver, Colorado 80205
303-399-5700
TWX: 910-931-2201

Approved For Release 2007/07/17 : CIA-RDP86B00689R000300040031-0