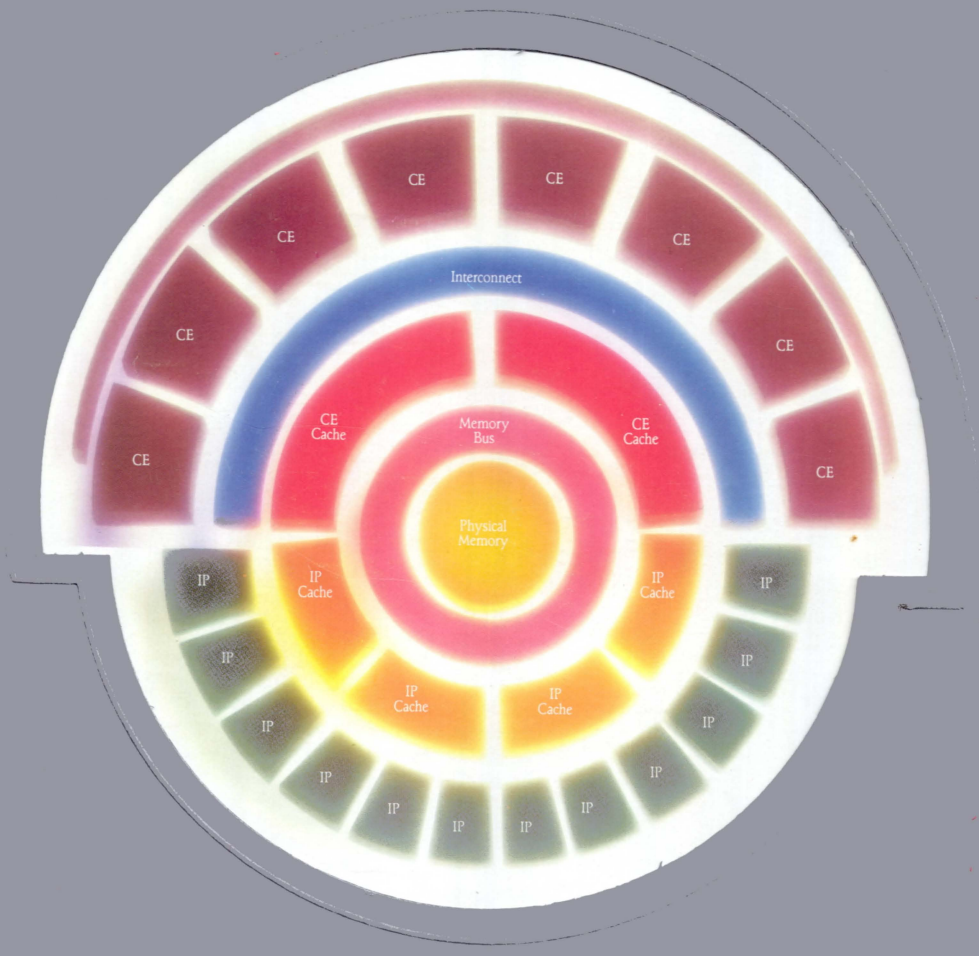
FX/Series Product Summary

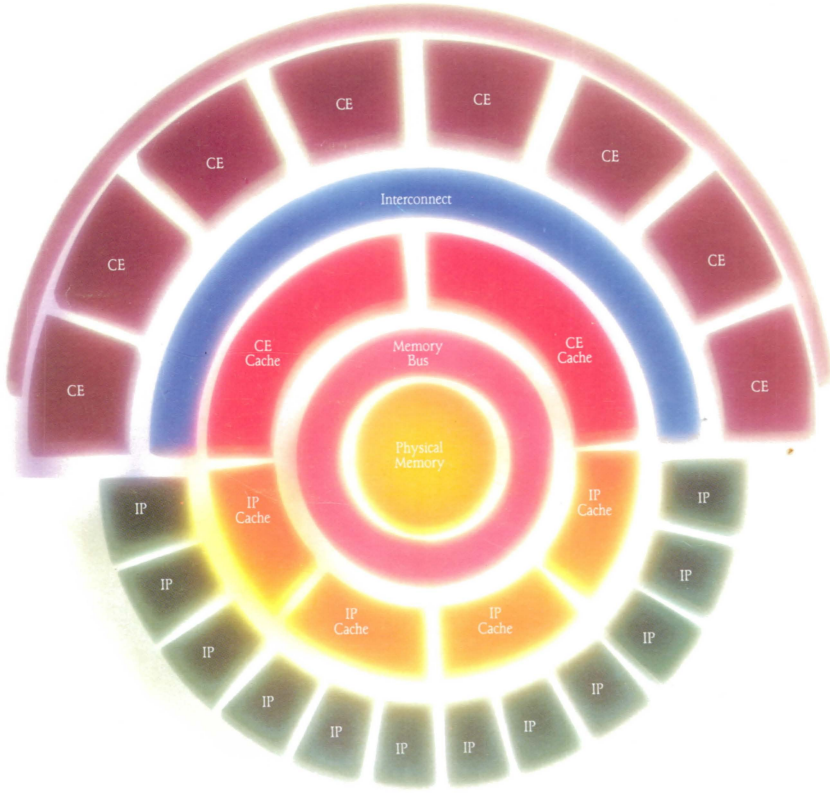


ALLIANT

ALLIANT FX/Series

Product Summary

June 1987



Alliant Computer Systems Corporation
One Monarch Drive • Littleton • Massachusetts 01460
617-486-4950

Revision History:

Original Release - January 1985

Revised - June 1985

Revised - October 1986

Revised -June 1987

Printed in the United States of America

Contents

Chapter 1 – Introduction

Chapter 2 – Parallel Processing and Alliant Concurrency

Chapter 3 – Alliant System Description

Chapter 4 – Concentrix Operating System

Chapter 5 – Alliant Networking Supercomputing Resources (ANSR)

Chapter 6 – FX/Fortran Compiler

Chapter 7 – FX/Ada Development System

Chapter 8 – Reliability, Availability, and Serviceability

Chapter 9 – System Expansion

Tables

Figures

CHAPTER 1

Introduction

Parallel processing, or the application of multiple processors to a single task, is not a new idea. The potential of using multiprocessor systems to reduce time-to-solution for individual compute-bound problems has long intrigued computer scientists. However, until now, engineers and scientists have been forced to rewrite existing software to take advantage of parallel processing.

Alliant Computer Systems Corporation offers a line of parallel processing systems that deliver on the potential of parallel processing. The FX/Series™ systems are the first to run existing applications in parallel with little or no reprogramming. They are based on fundamental advances in computer architecture, operating system, and compiler design. With Alliant systems, the simple addition or deletion of processors alters the level of performance for both production and development jobs.

Product Line

The Alliant product line includes two general purpose time-sharing systems targeted at complex engineering and scientific applications:

- The FX/8™ computer combines parallel processing with vector processing, the technology pioneered on supercomputers, to provide fast time-to-solution for large compute-intensive problems, and sustained throughput for a large number of users. The FX/8 can be expanded in the field to an eight-processor configuration that provides 94.4 MFLOPS (millions of floating point operations per second) double precision (64-bit) peak performance in a 49-inch wide, air-cooled cabinet.

- The FX/1™ computer is a single-processor version of the FX/8 that provides 11.8 MFLOPS double precision peak performance in an integrated desk-high package. The FX/1 is ideal as an application engine, multiuser computer, or computational server on a network of engineering workstations.

The two systems use identical hardware and software and are fully compatible, offering the customer investment protection and an easy upgrade path from the FX/1 to the FX/8. To upgrade, the user simply installs the FX/1 hardware modules into the FX/8 chassis. Code developed on the FX/1 runs in parallel on the FX/8 with no reprogramming, recompiling, or relinking.

Three Forms of Parallelism

The Alliant architecture is based on three distinct but interconnected resource classes, all of which run in parallel with each other.

- The **interactive processors** (IPs) comprise an expandable pool of computers that execute interactive user jobs and the operating system. The IPs maintain system responsiveness and allow the computational elements to concentrate on the compute-intensive portions of user applications.

- The **computational elements** (CEs) are 5050 KWhetstone single precision (32-bit), 4270 KWhetstone double precision (64-bit) general purpose microprogrammed computers each of which delivers 11.8 MFLOPS peak 32- and 64-bit performance. Each CE supports integrated vector, IEEE floating point, and parallel processing control instruction sets with dedicated hardware. Under operating system control, individual CEs multiprocess smaller production jobs and development jobs such as compilations.

- The **computational complex** introduces a new form of parallel processing, "Alliant concurrency," that groups up to eight computational elements (CEs), into an entity that is treated by the operating system as a single resource. The CEs in the complex simultaneously execute a single application, transparently to the user, to reduce time-to-solution.

Dedicated hardware on each CE is used to control the scheduling and synchronization of multiple CEs in a computational complex, and allows parallel processing to occur with extremely low overhead. A single instruction, automatically inserted by the FX/Fortran™ compiler, switches a user program from serial to parallel execution in less than two microseconds. Efficient control makes it possible for FX/Fortran to exploit the numerous opportunities for fine-grained parallelism that exist throughout engineering and scientific programs.

Alliant concurrency enables FX/Fortran to optimize more of a program than is possible on a vector-only computer. In addition to optimizing vectorizable loops, non-vectorizable loops are automatically optimized by FX/Fortran for high speed parallel execution on the FX/8. For example, data-dependent code that must run sequentially on advanced vector-processing supercomputers runs in parallel on the FX/8 with the data dependency synchronized in dedicated hardware.

Alliant concurrency decreases the time-to-solution for a single application as additional CEs are added. Users can start with a small system and upgrade computational performance in the field without obsolescence of hardware or software.

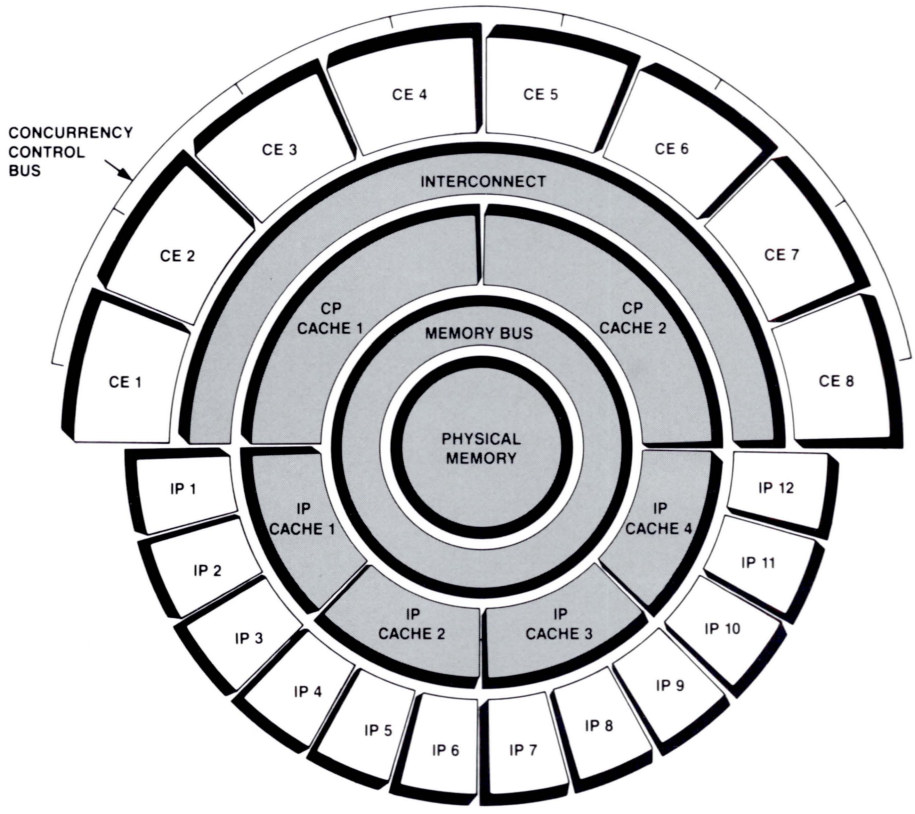


Figure 1-1: The Alliant architecture includes three resource classes. The interactive processors (IPs) run interactive user jobs and the operating system. Computational elements (CEs) multiprocess smaller production jobs and compilations. The computational complex uses Alliant concurrency to apply multiple processors to a single job, reducing time-to-solution for computationally intense applications.

Multiuser Computing

Unlike production computing, multiuser computing requires that many smaller production jobs, and development jobs such as compilations and edits, receive system resources in a timely manner. In most computer systems, a large production job adversely impacts system throughput, and a large number of user jobs degrades system responsiveness.

In addition to Alliant concurrency, Alliant systems support multiprocessing. Alliant multiprocessing automatically utilizes multiple processors to run multiple jobs simultaneously, thereby increasing system throughput and responsiveness. By combining Alliant concurrency with multiprocessing, FX/Series systems deliver near-supercomputer performance to production jobs, while maintaining the high level of throughput and responsiveness necessary for efficient multiuser computing.

Concentrix™ Operating System

The Concentrix operating system is a full native port of the Berkeley UNIX® operating system extended for high performance computing. Concentrix supports a multiuser environment and contains extensive software development and documentation tools. Extensions include multiprocessor support and support for the large physical and virtual memory requirements of technical applications.

Concentrix schedules and allocates jobs to run on the computational complex, individual CEs, and individual IPs. Compute intensive jobs are run on the computational complex or individual CEs. Interactive jobs, input/output tasks, and other operating system activities are scheduled for any available IP.

Programming Languages

The FX/Fortran compiler allows Alliant parallel processing to be used to full advantage without special programming skills. The FX/Fortran programmer simply selects "optimize" at compile time. Applications written for conventional scalar and vector computers port quickly and remain compatible with industry standards yet automatically take advantage of parallel and vector processing.

The FX/Fortran compiler implements the full ANSI Fortran-77 programming language (ANSI X3.9-1978). In addition, the language includes most of the extensions found in industry-standard compilers such as VAX Fortran, and implements the array processing and dynamic array extensions for the next ANSI Fortran standard known as Fortran 8X.

Originally implemented by the U.S. Government, the Ada™ programming language is specifically aimed at improving software reliability, portability, and maintainability while significantly reducing overall system costs. It was designed for uses ranging from large complex systems to embedded real-time applications. Ada naturally takes advantage of Alliant parallel processing through its task structure. Ada tasks within a program can run in parallel on multiple computational elements (CEs), interactive processors (IPs), and the computational complex, resulting in very fast execution of applications. The FX/Ada™ Development System includes the Ada compiler, a screen-oriented symbolic debugger, library maintenance utilities and programming tools, and a runtime system.

FX/C is an ANSI compatible highly optimizing C compiler. C is described by Kernighan and Richie[1] as:

> a general-purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators. C is not a "very high level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.

The Pascal compiler is based on the Berkeley UNIX 4.2 compiler for the DEC™ VAX™. It is an implementation of the Pascal language described in the Jensen-Wirth *Pascal Report*.

FX/C and Pascal provide programmers access to concurrency and vector processing through explicit mechanisms. FX/Ada programs can incorporate routines written in the other languages which utilize both concurrency and vectorization. A concurrent-call library routine allows for parallel execution of individual subroutines. In addition, a highly tuned library of vector operations is callable from these languages.

All Alliant languages share a common calling convention. Routines can be written in any of these languages and call each other directly.

Third-Party Software Products

A broad range of application programs, system software, and utilities for the FX/Series systems are available from the leading software vendors. Application software is available for electrical and mechanical CAE, earth resources, chemistry, image processing, earth resources and other scientific and engineering fields. System software and tools include a relational database management system and fourth generation language, the Mainsail systems design language, graphics software tools, and spreadsheet and scheduling software. Multiple third-party math libraries also run on the FX/Series systems.

Reliability, Availability, and Serviceability

Alliant has implemented extensive manufacturing quality programs such as electrostatic discharge protection, component pre-testing, and ongoing reliability testing. At the systems level, a diagnostic bus interconnects diagnostic processors on each module. The diagnostic operating system and run-time error reporting allow confidence in production results.

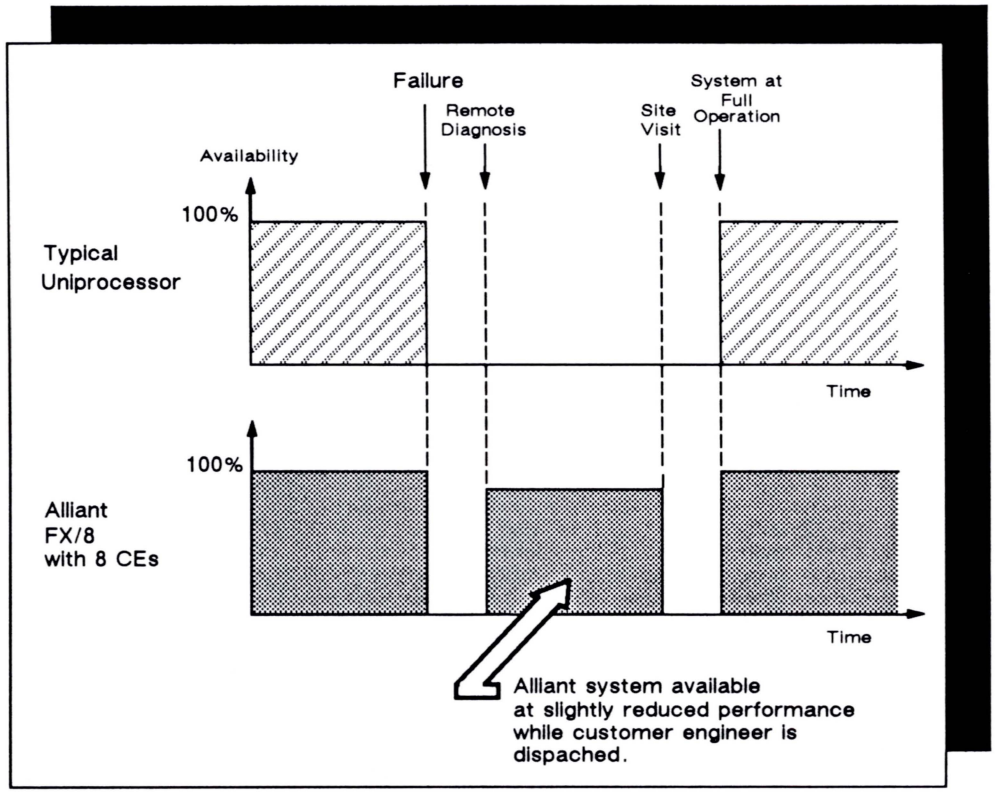


Figure 1-2: Alliant computers provide a high level of system availability.

Alliant parallel processing provides the added advantage of a very low mean time to repair. Most hardware failures simply result in reduced performance after restart. Alliant systems can continue to do useful work when conventional computers would experience system failures and costly down time. In addition, the remote maintenance feature of Alliant systems reduces system down time.

Delivered Parallel Processing Performance

Alliant FX/Series systems deliver high performance for a wide range of compute-intensive scientific and engineering applications.

The LINPACK[1] benchmark, developed by Jack J. Dongarra of Argonne National Laboratory, measures delivered performance solving dense systems of linear equations. LINPACK represents one of a class of equation solvers that forms the computationally-intense kernel for a large number of scientific and engineering applications. Figure 1-3 shows the delivered performance of the FX/8 running LINPACK in full (64-bit) precision and illustrates how Alliant concurrency efficiently applies multiple processors simultaneously to the same job, transparently to the user, to achieve a high level of delivered performance.

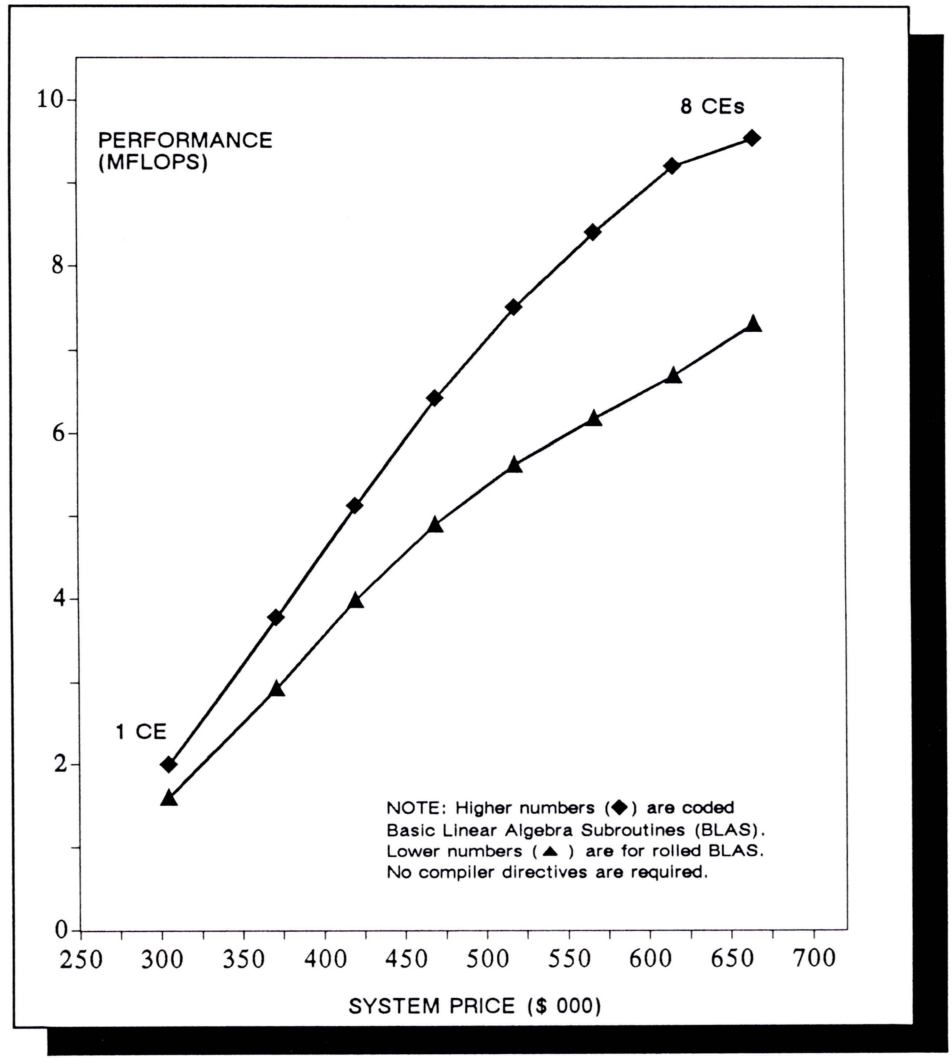


Figure 1-3: The industry standard LINPACK benchmark demonstrates the range of performance available with the FX/8.

Delivered Multiprocessing Performance

The FX/8 maintains a high level of throughput for smaller production jobs and development jobs such as compilations by automatically multiprocessing them on any available computational element. Figure 1-4 illustrates the "theoretical" maximum multi-user throughput of one and eight-processor systems. It depicts minimum degradation in time-to-solution for one "additional job" as the background workload is increased. To simplify the model it assumes all jobs are the same and there is no operating system overhead or memory contention.

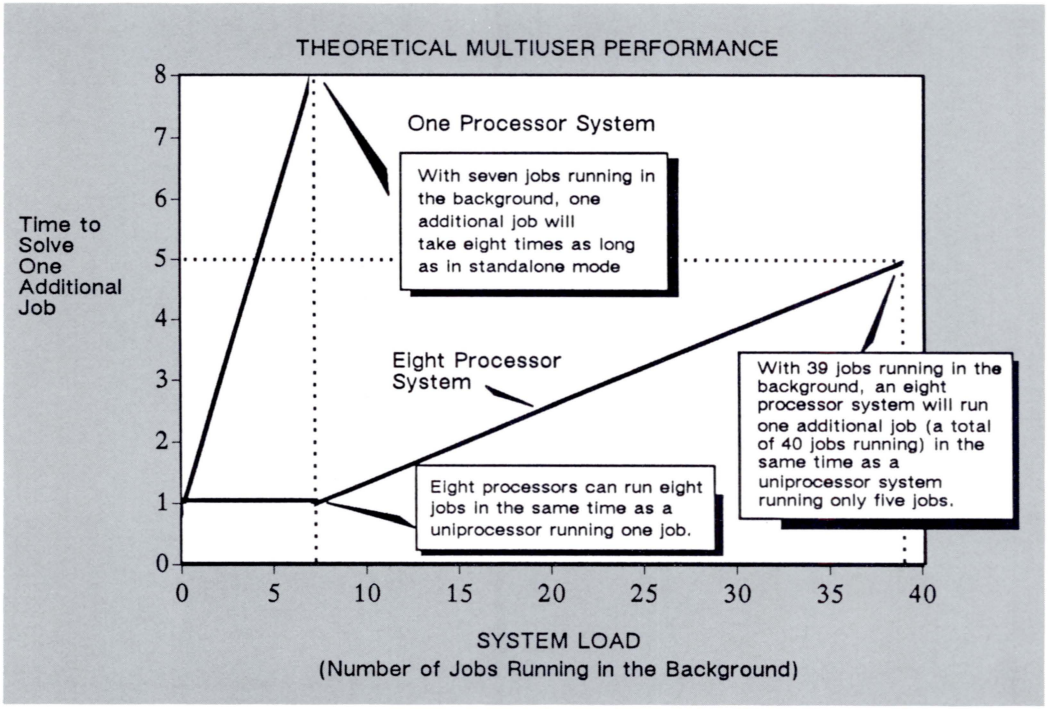


Figure 1-4: Multiple processor systems offer the potential for significantly greater throughput for multiuser computing.

An eight-CE FX/8 can be used as an example of this theoretical multiprocessing model. With seven jobs running in the background, a new user logging-in to execute an "additional job" would theoretically expect to get the full resources of a dedicated processor. The total of eight jobs would all execute at the same speed as if they were running stand-alone. On a uniprocessor system, the same eight jobs would share the single processor and on average each take eight times as long to execute.

Theoretical peak performance can never be attained in any real multiprocessing system because of operating system overhead and memory contention. However, Alliant systems minimize these effects by using IPs to run interactive jobs and the operating system, and perform system I/O. This frees the CEs for multiprocessing of individual programs. To provide the memory bandwidth required to support multiple processors operating simultaneously, the FX/8 uses a 376-MB-per-second cache that can supply the full bus bandwidth of all eight CEs operating in parallel. Combined, these features enable FX/Series computers to deliver throughput that often approaches theoretical and exceeds by a wide margin the performance available from uniprocessor systems.

Figure 1-5 illustrates the actual multiprocessing efficiency of the FX/8. It shows the runtime for one computational fluid dynamics job while multiple (N) copies of that job are running on the system.

For example, on an eight-CE system with 15 jobs running in the background, the best possible run-time for the 16th job is twice the run-time of a single job. Figure 1-5 shows that FX/8 system performance in this heavily loaded environment is very close to the theoretical, even for a large number of users.

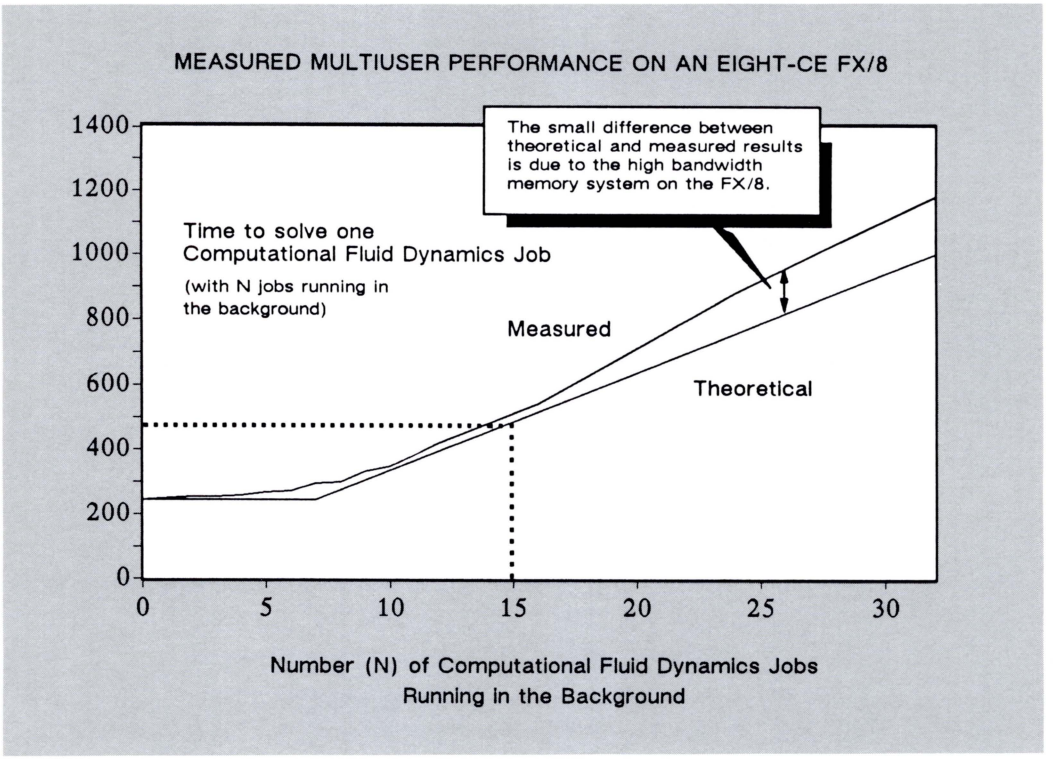


Figure 1-5: An Alliant FX/8 can operate in a multiprocessing mode delivering very high multiuser thoughput.

REFERENCES

[1] B. W. Kernighan and D. M. Richie, *The C Programming Language*, Prentice-Hall Inc., Englewood Cliffs, N. J. 1978.

[2] J. J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment*, Technical Memorandum No. 23, Argonne National Laboratory, (March 26, 1987).

CHAPTER 2

Parallel Processing and Alliant Concurrency

To date, commercial multiprocessor systems exist primarily for multitasking applications. Development of a parallel processing architecture for the compute-intensive applications found in engineering and science has proved difficult for two reasons:

- Hardware designers have typically sought parallelism at the instruction level where intrinsic parallelism is minimal due to the sequential nature of most in-line code.
- Academic research in parallel systems has centered around totally new approaches, rather than extensions to existing systems. The result has been proposals for new languages, radically new architectures, and systems incapable of supporting existing applications written in standard languages.

In 1976, Dr. David Kuck, now director of the Center for Supercomputer Research and Development at the University of Illinois, discussed some of the problems in existing multiprocessor machines, and how these problems might be overcome in future designs[1]:

> Compilers for these machines have been uniformly difficult to write – mainly for two reasons. First, it has been difficult in ordinary programs to discover operations that can be executed simultaneously on such machines. Second, the organizations of these machines have often made it difficult to implement array operations in efficient ways.
>
> The first [problem] can be solved by using several compilation techniques aimed at parallelism detection and exploitation.... For most ordinary FORTRAN programs, this will probably lead to theoretical speedups that grow (nearly) linearly in the number of processors used in a computation.
>
> The second problem is closely related to the first. If one can transform most programs into array form, then proper hardware is necessary to support their fast execution. This means we must have a control unit capable of executing array instructions in a more or less direct way.... Furthermore, we must have a high-bandwidth memory that provides conflict-free access to arrays.... Finally, sufficiently high processing bandwidth must be provided - most likely by a combination of parallel and pipeline techniques.

Kuck concluded, "Our experiments lead us to conclude that multioperation machines could be quite effective in most ordinary FORTRAN computations."

In 1983, Dr. Ingrid Bucher of Los Alamos National Laboratory discussed the effect that new technology might have on the next generation of supercomputers, and postulated on their architecture[2]:

> It appears unlikely that advances in semiconductor technology or even the transition to Josephson junction technology will bring the speedup of two orders of magnitude needed for scientific computing within the next decade. It is therefore safe to assume that the next generation of supercomputers will achieve most of their speedup by more parallelism, probably in the form of several scalar or vector processors operating in parallel asynchronously and connected to a common memory. Communications overhead and Amdahl's law will limit the number of cooperating processors to probably not more than 16.

Compilers that detect parallelism, multiple vector processors connected to a common high-bandwidth memory, and efficient communications between processors were all predicted to form the basis of the next generation of parallel processing supercomputer architectures. All of these can be found in the Alliant FX/8 – today.

Through a combination of advanced architecture, operating system, and compiler technology, the FX/8 represents the realization of a machine that had previously been only conceptualized: the first computer specifically designed for transparent scalable parallel processing of scientific applications.

Parallelism in Supercomputer Architectures

In her 1983 paper, Ingrid Bucher describes the use of parallelism in three types of supercomputer architectures[2,]:

> *Single Instruction-Multiple Data Stream Machines (SIMD).* These machines [are typically pipelined vector processors that] are capable of performing identical operations on arrays of data at very high speed. We will call this mode of processing *synchronous parallel processing.*
>
> *Multiple Instruction-Multiple Data Stream Machines (MIMD)* consisting of several processors each processing its own instruction stream. We call this mode of operation *asynchronous parallel processing.*
>
> *Multiple Vector Processors* [such as the Alliant FX/8] featuring several vector processors connected to a common fast memory. These machines will be capable of both synchronous and asynchronous parallel processing.

SIMD Machines and Instruction Level Parallelism

Today's computers apply instruction-level parallelism to achieve enhanced performance.

The simplest form of instruction-level parallelism is **pipelining**, which allows the different phases of instruction execution (instruction fetch and decode, effective address calculation, operand fetch, etc.) to occur in parallel with other instructions. Pipelining permits several instructions (one for each phase of the pipeline) to be in different stages of execution at the same time.

The pipelining process is aided by the use of **multiple execution units**. A single instruction stream is broken down into two or more functional areas (floating point multiply and divide, integer add/subtract, load/store, logical operations, etc.) any or all of which can be operating in parallel.

Vectorization (synchronous parallel processing) is the form of instruction-level parallelism exploited by SIMD machines. Vectorization allows multiple data elements to be processed by a single machine instruction. The first truly successful vector processor, the Cray-1™, was announced in the mid 1970's. Today, almost all supercomputers use vector processing to achieve high levels of peak performance.

However, vector processing has a limitation. Very few real applications are highly vectorizable. Consequently vector processors only deliver a small percentage of their peak performance for most applications. This effect is known as Amdahl's Law, or the "vector bottleneck."

Amdahl's Law

Amdahl's Law demonstrates that with vector processing alone, time-to-solution becomes dominated by the non-vectorizable portion of the code.

In trying to predict the performance of the next generation of supercomputers, Bucher develops an extended version of Amdahl's Law that relates three different types of execution speed to the total execution time[2]:

> The *sequential speed* S_{seq}, characterizing the rate at which a machine can process code that has to be processed in sequential mode, either for reasons of logic or because it is too costly to vectorize or parallelize it.
>
> The *synchronous speed* S_{syn}, characterizing the rate at which a machine can process code that lends itself to vectorization or synchronous parallel processing with small granularity. This speed is dependent on the vector length and on other characteristics of the workload.
>
> The *asynchronous parallel speed* S_{asyn}, characterizing the rate at which the machine can process code that is unsuitable to vectorization or synchronous parallel processing but can be parallel processed asynchronously in large chunks, as for example Monte Carlo and Particle-in-Cell (PIC) simulations. The asynchronous parallel speed will depend on the number of processors available, the percentage of the tasks that can be parallelized, the amount of communication including overhead, the number of synchronization steps, and the choice of algorithm. It might be considerably lower than its upper limit, the product of sequential speed and the number of processors.
>
> Let
>
> F_{seq} be the fraction of the workload that can be processed in sequential mode only,
>
> F_{syn} be the fraction that can be vectorized, and
>
> F_{asyn} the fraction that can be parallelized asynchronously;
>
> then the time required to run this workload is proportional to
>
> $$T = F_{seq}/S_{seq} + F_{syn}/S_{syn} + F_{asyn}/S_{asyn} = 1/S_{eff}$$
>
> where S_{eff} is the workload-dependent, effective speed of the machine. This equation is an extension of Amdahl's Law[4,5]. It implies that the slowest of the execution speeds will critically influence the effective speed unless the weight factor F associated with it is negligibly small.

Bucher draws the following conclusions regarding the computational speed of supercomputers[2]:

> The slowest characteristic processing speed ... acts like a bottleneck. The most effective way to speed up a machine is to increase this speed or to decrease the fraction of work associated with it. Both solutions are often very difficult.
>
> Speeding up the fastest characteristic speed of a supercomputer [the vector processing speed] will markedly improve its effective speed only if the fraction of the workload running at that speed is close to 1. If this is not the case, installation of additional vector pipes on a vector computer is not useful.

Figure 2-1 graphically illustrates Amdahl's law. It shows that an application must be highly vectorizable in order for a vector computer to deliver a large percentage of its peak performance.

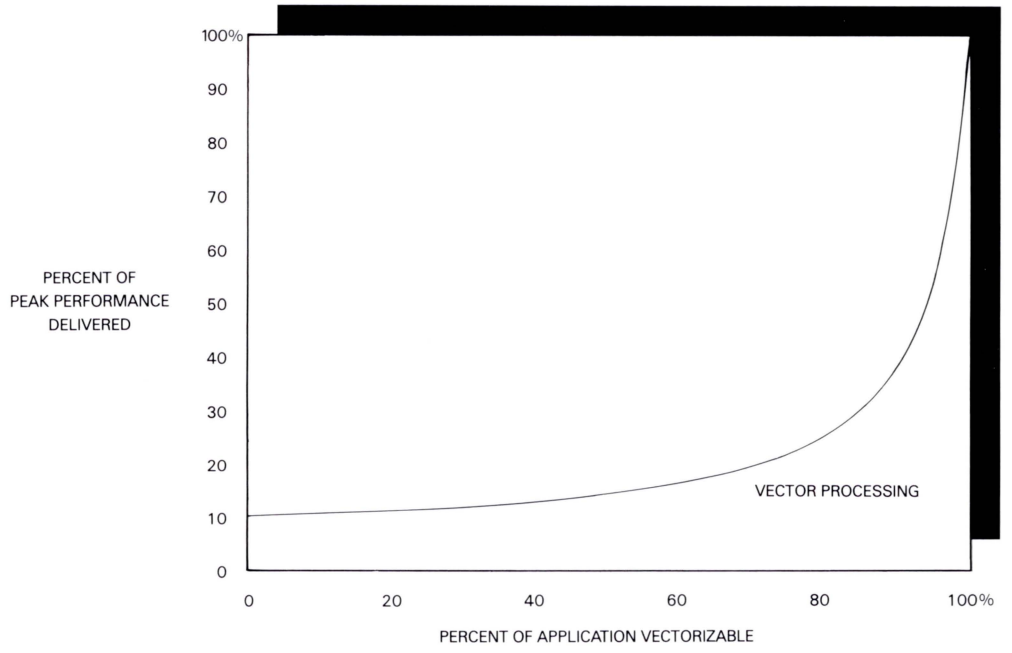


Figure 2-1: Delivered performance vs. percent of application vectorizable

MIMD Machines and Asynchronous Parallel Processing

Bucher's version of Amdahl's law shows that breaking the "vector bottleneck" requires speeding up those parts of a program that cannot run in vector mode. To achieve this, computer designers have turned to multiprocessor architectures.

Most multiprocessor systems are designed primarily for running multiple jobs on multiple processors. They increase system throughput, but do not decrease time-to-solution for a single job. Bucher's equation points out the difficulty that such systems encounter when trying to run a single complex application. First, the sequential speed of a single processor is typically low, so that any portion of the code that runs on a single processor runs slowly. Second, these systems typically do not support vector processing, so that no portion of the code can be processed synchronously.

Bucher's definition of asynchronous parallel processing refers to the simultaneous execution of "large chunks" of code on multiple processors. These "large chunks" could include loops, especially those that contain calls to subroutines. Bucher warns that communication, including "overhead" and the "number of synchronization steps" affects the overall speed of asynchronous parallel processing.

Use of the operating system to initiate and suspend asynchronous tasks executing across multiple processors requires that each task be fairly long in relation to the overhead of start-up and synchronization if any efficiencies are to be expected. Software based start-up and synchronization of multiple jobs executing on a multiprocessor can be complex and inefficient, resulting is very low asynchronous parallel processing speeds.

In researching whether parallel processing could help break the "vector bottleneck" for large complex codes, Bucher's group asked the following questions[2]:

> (i) Can scientific computations be parallelized asynchronously in large chunks?
>
> (ii) How frequent and time-consuming are necessary communications between parallel processes?
>
> The answer to the first question is positive. We have studied three kinds of tasks that seem fairly representative of the work at our Laboratory: Monte Carlo simulations, a PIC code, and the solution of partial differential equations. We observe that:
>
> - Physical problems are naturally parallel. Asynchronous parallelization of codes seems to be conceptually simpler than vectorization.
> - Most of the computational work of all three codes could be parallelized at the subroutine level. This is very important. Because we do not expect compilers to recognize this type of parallelism in the near future, changing codes by hand must not be too time consuming.
> - The work could be evenly distributed between any number of processors. This is important for achieving maximum efficiency. Typically, each processor is executing the same code, although not all may choose the same path through it.

Even here, the assumption was that the programmer would need to analyze and restructure the code to create subroutines that were free of dependencies. Also left to the programmer was the allocation and synchronization of tasks running on multiple processors.

To answer the second question, Bucher and her colleagues ran a complex application program on a system consisting of eight array processors attached to a mainframe host, and determined that, "Although communications between processors were infrequent and short, operating system overhead to initiate communications was disastrous."

Bucher's group concluded by identifying several architectural features that they felt were necessary for efficient asynchronous parallel processing, and speculated on the performance that such a design could achieve[2].

> The connection of each processor to a common memory is an important architectural feature, especially for Monte Carlo computations and partial differential equations. Efficient synchronization hardware accessible to the user program is another important requirement. With these features our preliminary results indicate that, for a small number of processors, speedup factors nearly equal to the number of processors are achievable.

Multiple Vector Processors

The third type of architecture described by Bucher, multiple vector processors, refers to a multiprocessor system comprised of several pipelined vector processors. This type of architecture combines both forms (synchronous and asynchronous) of parallel processing, resulting in the highest level of performance. Examples of this type of computer are the Cray X-MP™, the Cray-2™, and the Alliant FX/8.

The Alliant Architecture

The Alliant architecture utilizes parallel processing in all its forms to decrease time-to-solution for large applications while providing sustained throughput for smaller production jobs and development tasks.

Each computational element (CE) uses a five-stage pipeline and multiple functional units to achieve a high level of scalar (sequential) performance. Instruction fetch,

address calculation, memory reference, integer calculation, scalar floating point add, and scalar floating point multiply are all performed by separate functional units. Any or all of these units can operate in parallel with each other.

Each CE contains hardware to support a full vector (synchronous parallel) processing instruction set. Up to eight CEs can execute a 32-element 64-bit vector operation simultaneously for an effective vector length of 256 64-bit data elements. CE vector operations utilize up to a 12-stage vector pipeline when running at peak rate.

Multiprocessing occurs on both the CEs and the IPs. Individual CEs can be declared by the system manager to be "statically detached" from the computational complex. These CEs are treated as individual computers by the operating system and are available to run smaller production jobs or development jobs such as compilations. Similarly, if there is no work to be done on the computational complex, the complex can dynamically "break apart" into multiple CEs available for multiprocessing of individual jobs.

Finally, Alliant concurrency carries parallel processing one step further by distributing iterations of individual loops across multiple processors for simultaneous execution. Alliant concurrency goes beyond vector processing. By speeding up portions of the code that would normally run in serial mode on vector machines, Alliant concurrency helps to break the "vector bottleneck" and allows the FX/8 to deliver greater absolute performance and faster time-to-solution than comparably priced vector computers.

The computational complex introduces a new form of parallel processing, "Alliant concurrency," that groups up to eight computational elements (CEs), into an entity that is treated by the operating system as a single resource. The CEs in the complex simultaneously execute a single application, transparently to the user, to reduce time-to-solution.

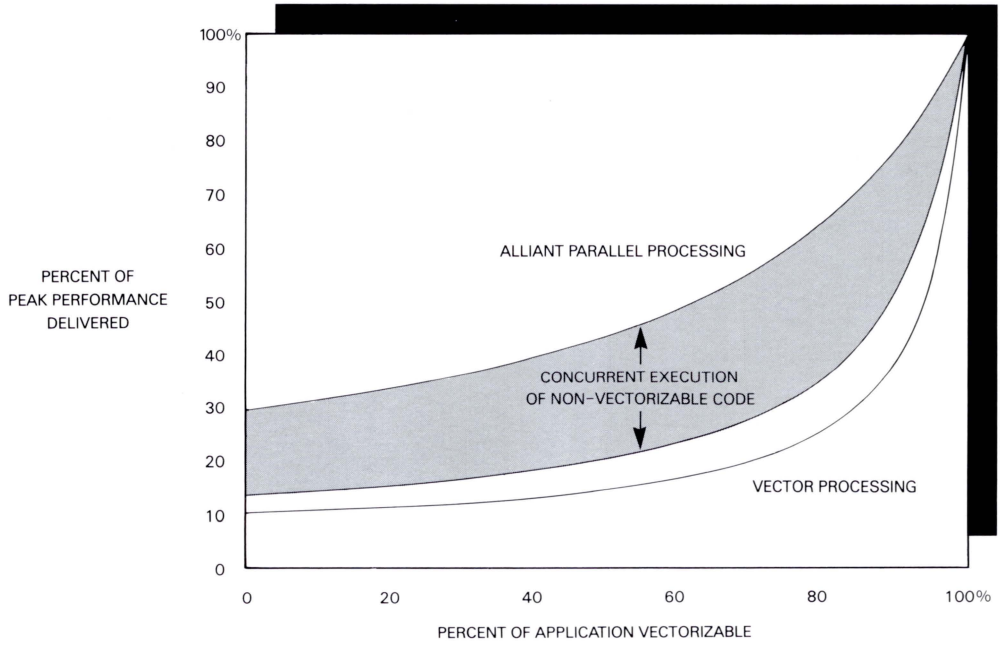


Figure 2-2: Alliant concurrency enables the FX/8 to deliver a greater percentage of its peak performance to an application than vector computers.

Parallel Processing Based on Iterative Constructs

Alliant concurrency uses the program loop as the instruction stream to be executed in parallel. This innovative approach:

- requires no source code reprogramming. Loops and loop dependencies are easily identified by the FX/Fortran compiler.
- results in high parallel processing efficiency. Task assignment and dependency synchronization can be handled in dedicated hardware.
- delivers high performance. Vector processing and parallel processing are easily combined.
- adapts to a wide range of applications. Loops with conditional code, data dependencies, subroutine calls, potential feedback and loop exits can be optimized for parallel processing. These loops run serially on conventional vector computers.

Iteration Assignment in Alliant Concurrency

Figure 2-3 shows how Alliant systems assign multiple CEs to the execution of a DO loop in a single program.

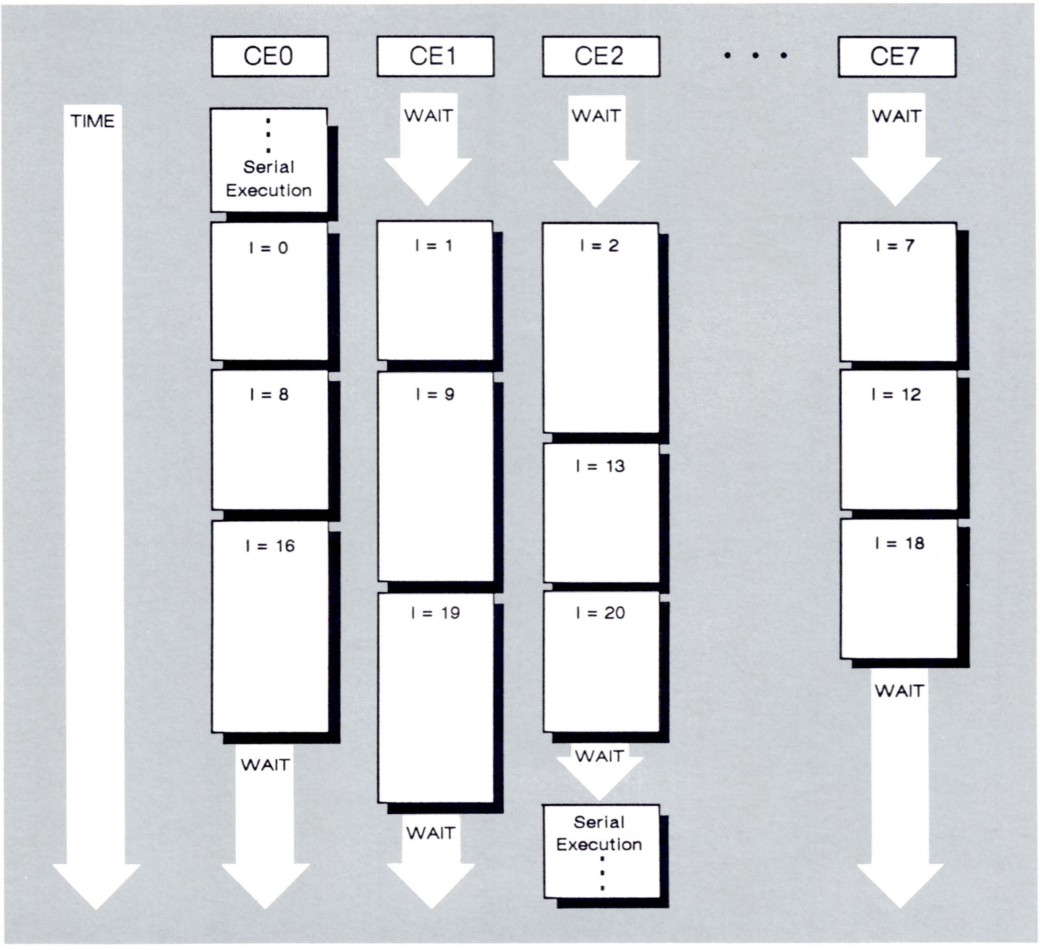


Figure 2-3: Iteration assignment in an 8-CE configuration

During compilation, the FX/Fortran compiler inserts special concurrency control instructions into those sections of code that can be executed in parallel by multiple CEs. Concurrent code includes not only loops, but advanced array operations such as A = B where A and B are arrays.

Execution is initially serial. One computational element begins executing while the others wait. Alliant concurrent execution begins when the active CE executes an instruction inserted during compilation. At that point, concurrency control data is transmitted over a dedicated concurrency control bus. Each CE then becomes self scheduling, executing the next iteration as soon as it completes its current iteration. When all iterations are completed, serial execution continues.

Parallel processing efficiency is high with Alliant systems because Alliant concurrency is self-scheduling and handled in dedicated hardware.

Concurrent Processing of Dependent Loops

Loops in which calculations in one iteration depend on results from a prior iteration inhibit traditional vector processing. For example, if the first iteration of a loop stores into an element of an array and the next iteration loads from the same element, the load must occur after the store to obtain correct results.

The simplest way to insure that the load follows the store is to run the loop serially, losing all concurrency. Alliant concurrency, however uses special instructions and additional dedicated hardware to synchronize dependencies.

The loop shown in Figure 2-4 contains a data dependency for the array F. Each iteration loads the value F(I), which was stored during the preceding iteration as F(I+1).

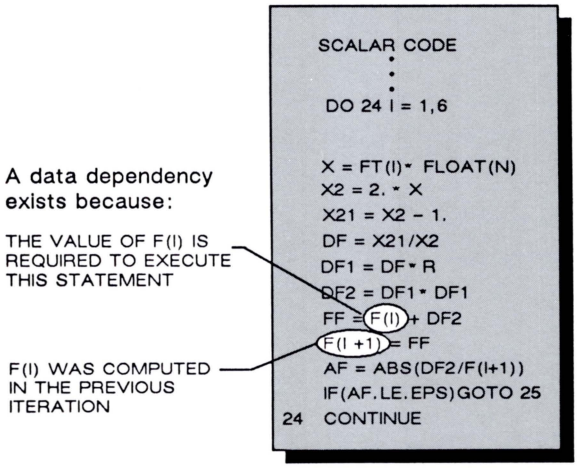


Figure 2-4: A data dependency between iterations can be efficiently processed with Alliant concurrency control based in hardware.

Figure 2-5 illustrates how Alliant concurrency maintains parallel processing efficiency in this loop.

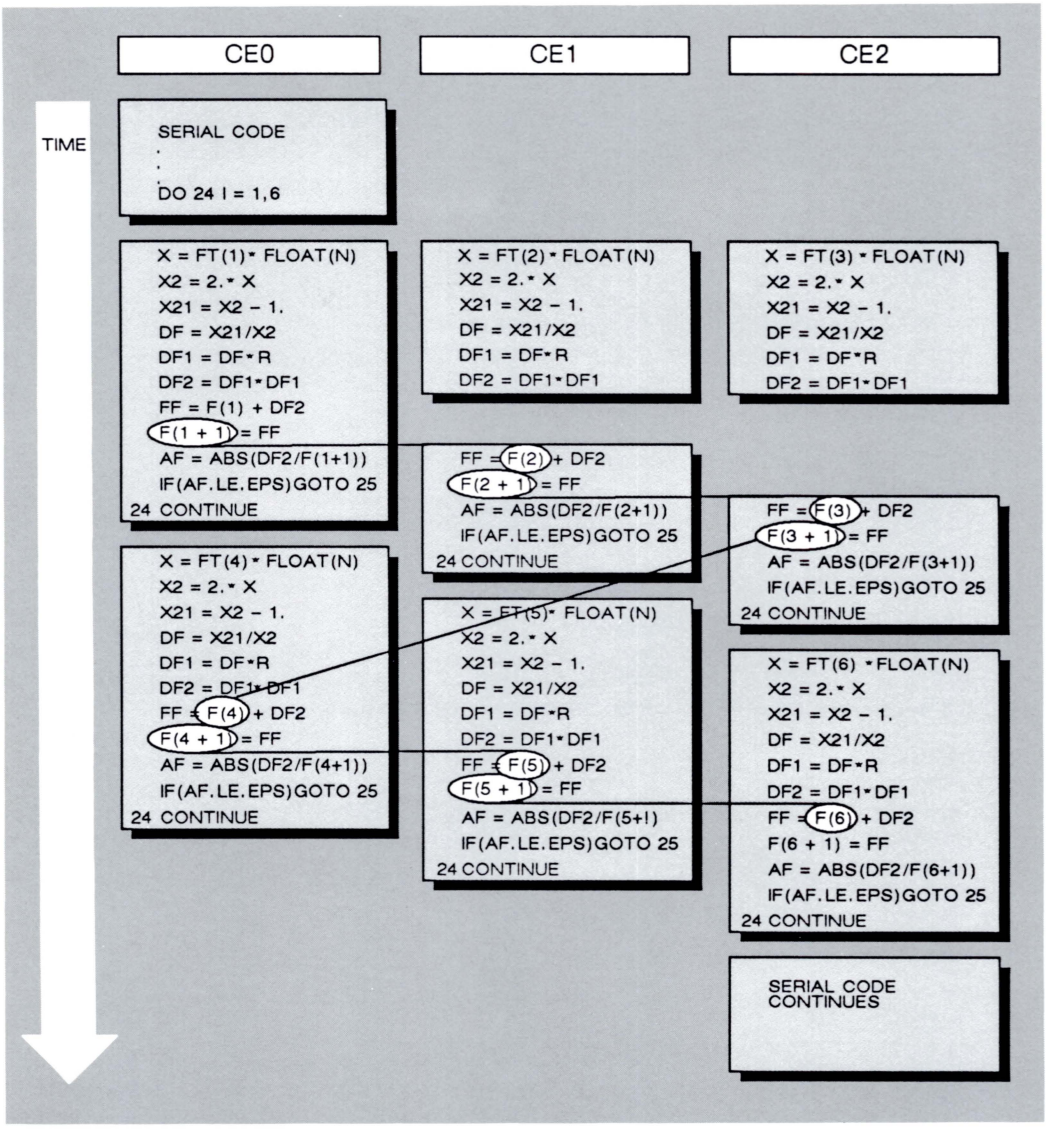


Figure 2-5: Concurrent processing of a DO loop with a data dependency achieves high efficiency as data dependencies become synchronized.

In this three-CE example, all CEs begin processing their first iterations (1, 2, 3) together. CE0 processes the full iteration. CE1 processes its iteration until it requires a value from CE0's iteration; CE2 processes its iteration until it requires a value from CE1's iteration.

Once CE0 has calculated and stored a new F(I+1), CE1 resumes execution of its iteration. Now two CEs are in operation: CE0 is completing iteration 1; CE1 has just begun completing iteration 2. CE2 is idle, awaiting a new value of F(I+1) from CE1's iteration.

Once CE1 has calculated the new value of F(I+1), CE2 resumes execution.

At this point parallel execution reaches maximum efficiency. When CE0 finishes iteration 1, it can immediately begin and complete iteration 4. By the time CE0 reaches the

data dependency and requires a value from iteration 3, CE2 has already calculated the value. The process continues until all iterations are completed. This "cascading" effect results in very high processor utilization.

Concurrent Processing of Various Loop Constructs

In addition to synchronizing loops containing data dependencies, Alliant concurrency can execute loops containing conditional branches, loop exits, subroutine calls, and potential iteration dependencies. In loops with conditional exits, Alliant concurrency always ensures that the correct iteration finishes a loop, that results from subsequent iterations executing in other CEs are not stored, and that final results are identical to serial execution.

Many applications nest loops. In a uniprocessor scalar computer, the "deepest" loop is processed first, the next deepest is incremented, then the deepest loop is processed until all iterations are completed. Nested loops are often bottlenecks in compute-intensive programs.

In executing nested loops, Alliant concurrency vectorizes the inner loop, if possible, and processes the next outer loop concurrently. This form of execution, concurrent-outer-vector-inner (COVI), results in maximum delivered performance for the FX/8 system.

Alliant Concurrency Control

Alliant concurrency is controlled in hardware at execution time. Additional CEs can be added in the field without recompilation or relinking of programs.

Hardware concurrency control allows a program to initiate, synchronize, and suspend concurrent processing with minimum overhead. Efficient control of concurrency achieves four critical functions in applying parallel processing to existing applications:

- Operation of one to eight CEs on a single program.
- Efficient allocations of CEs.
- Efficient synchronization between CEs.
- Straightforward compiler code generation.

Concurrency initiation, synchronization, and suspension are accomplished by a concurrency control unit (CCU) in each CE and an interconnecting concurrency control bus. The concurrency control bus provides a high-speed communication path between CCUs that is independent of program data and instruction paths.

Parallel Processing Efficiency

Supercomputer designers and engineers recognize that good price/performance parallel processing computers must be efficient in CPU utilization. As a result, parallel processing efficiency (Ep) becomes a major implementation issue in the design of these systems, as shown in the following equation.

$$Ep = \frac{Sp}{P}$$

Where: Ep = Parallel processing efficiency
Sp = Speedup over a single processor
P = Number of installed processors

Parallel processing efficiency is a measure of how much of the available computational resource can be used to do useful work versus how much is lost in waiting or in managing the parallelism itself. As Ep approaches the value of 1, overall system price/performance rises.

Parallel Processing Control

The two components of parallel processing control are the assignment of work to multiple processors and the synchronization of that work to ensure its proper completion. For system price/performance to be high, both components must be handled efficiently.

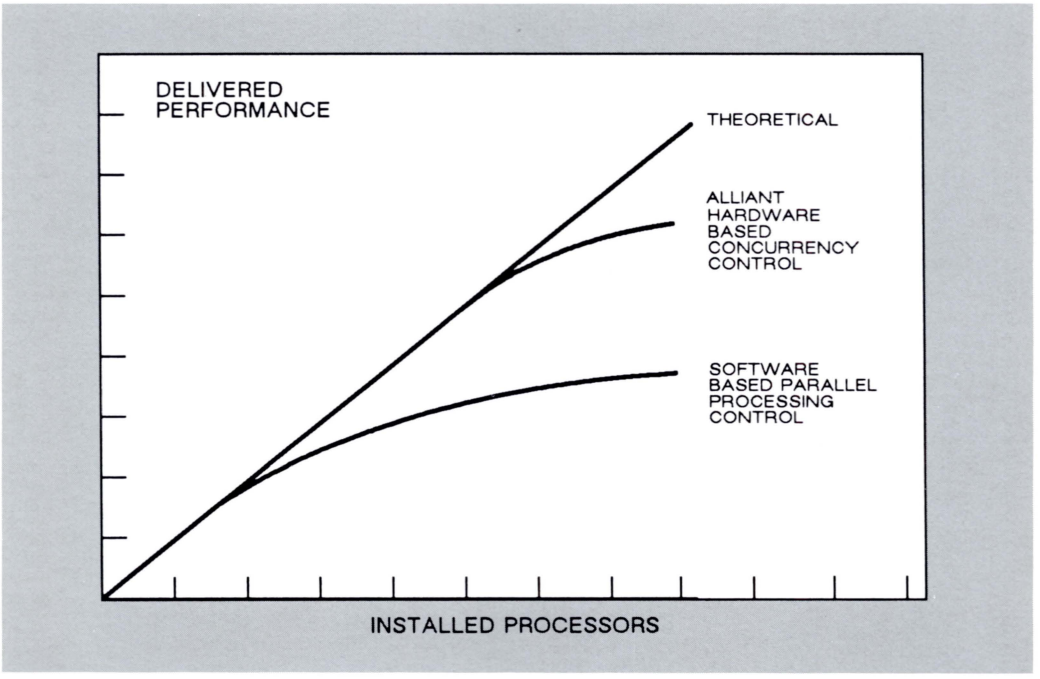


Figure 2-6: Software overhead can quickly negate the benefits of parallel processing.

Prior to the FX/8, proposed designs based parallel processing control in software alone. By any standard, the theoretical efficiency of software controlled systems is low. Alliant concurrency control is based in dedicated hardware so that for any given application, the speedup over a single processor can approach the number of processors installed.

An Architecture With A Future

The rapid advances in semiconductor technology over the last decade have enabled many computer vendors to "ride the technology curve" in order to improve system price/performance. For example, engineering workstations originally based on Motorola's MC68000 have migrated to the MC68010, and most recently to the MC68020. Technology injections such as this play a crucial part in the long term success of a computer company.

However, over the last decade, the companies that have had the most impact are those that first differentiate themselves with architectural innovation and then ride the technology curve. Figure 2-7 shows that the last major architecture injection was the advent of vector processing on machines such as the Cray-1.

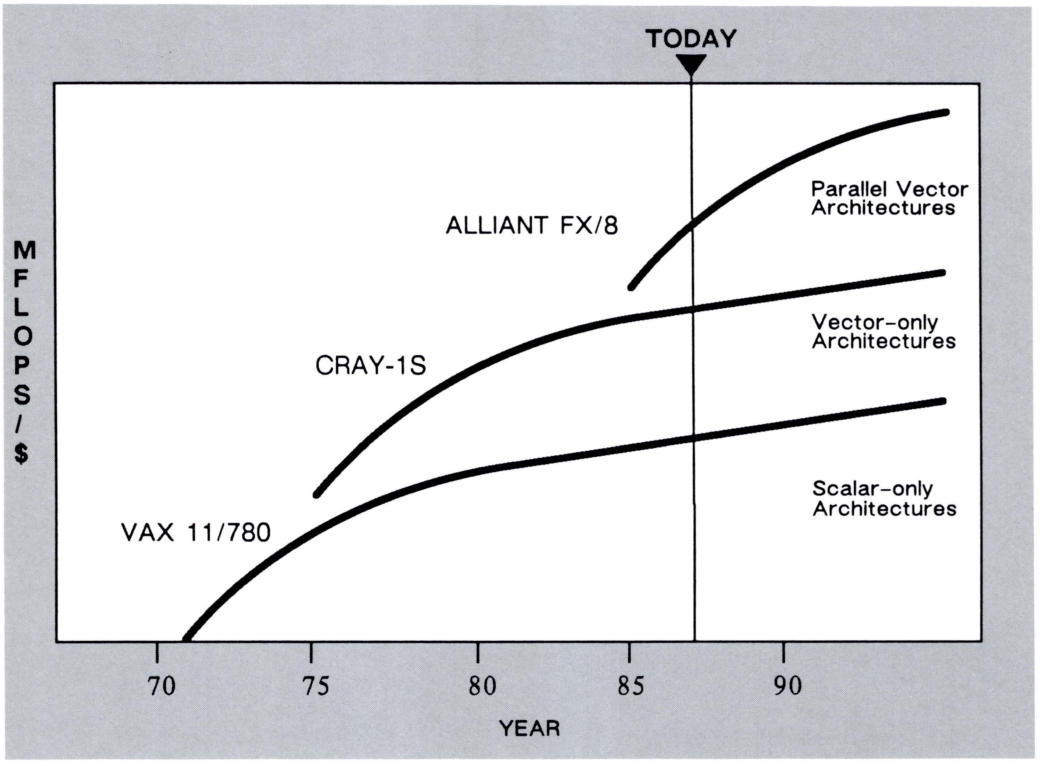


Figure 2-7: Since Alliant is at the beginning of an accelerating technology curve, rapid advances in parallel processing technology will continue to provide large increases in price/performance.

The Alliant FX/Series computers are based on fundamental advances in computer architecture design. The FX/Series parallel processing architecture ensures that:

- high performance systems can be constructed from moderately priced components using the latest available technology.
- little or no reprogramming of existing applications is required.
- performance is easily expanded.
- a degree of fault tolerance results from module redundancy.
- a range of systems compatible at the object code level is available.
- interactive performance is sustained with processors dedicated to operating system and interactive jobs.

Alliant FX/Series computers offer a long-term growth path. Users can add additional computational elements, computational processor caches, interactive processors, interactive processor caches, and memory modules as they are needed to maintain the desired level of performance.

REFERENCES

[1] D. Kuck, *Parallel Processing of Ordinary Programs*, Advances in Computing, Vol. 15 , M. Rubinoff and M.L. Yovits, Eds., Academic Press, 1976.

[2] I. Y. Bucher, *The Computational Speed of Supercomputers*, The Proceedings of ACM Sigmetrics Conference on Measurements and Modeling of Computer Systems, 1983.

[3] M. J. Flynn, IEEE Trans. Comp., C-21 1972.

[4] G. M. Amdahl, AFIPS Conf. Proc., 30 1967.

[5] J. Worlton, *A Philosophy of Supercomputing*, Los Alamos National Laboratory report 1981.

CHAPTER 3

Alliant System Description

All FX/Series computers execute the same system and application software and are configured from the same set of hardware modules. From the perspective of the user, the computers differ only in delivered performance and configurability. In contrast, most other established manufacturers offer product lines with different hardware architectures, different manufacturing and support requirements, and often different or incompatible software.

Computational Elements

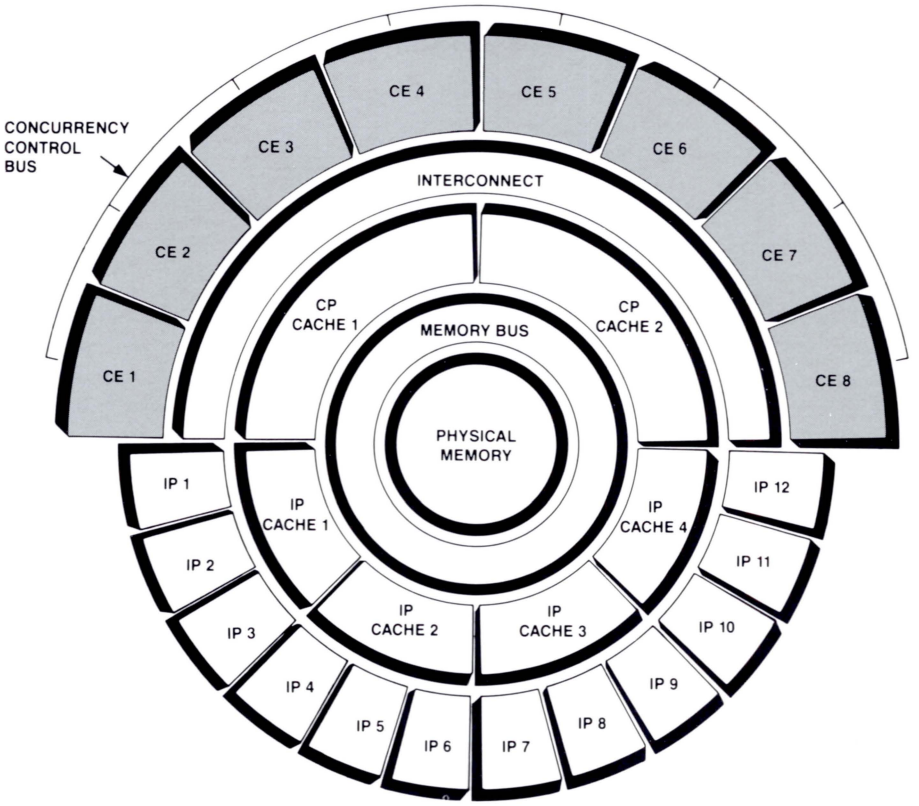


Figure 3-1: The computational elements

The computational element (CE) is the computational building block of all Alliant systems. Each CE is a microprogrammed, pipelined processor with integrated floating point and vector instruction sets. Individually or grouped in a computational complex, CEs deliver high absolute performance to single applications.

Table 3-1: Peak computational performance

PEAK PERFORMANCE			
		SINGLE CE	COMPLEX OF 8 CEs
Scalar	32-bit	5.05 MIPS	40.4 MIPS
	64-bit	4.27 MIPS	34.1 MIPS
Vector Floating Point	32-bit	11.8 MFLOPS	94.4 MFLOPS
	64-bit	11.8 MFLOPS	94.4 MFLOPS

Performance

Individual CEs execute the industry-standard Whetstone benchmark at 5050 KWhetstones single precision (32-bit) and 4270 KWhetstones double precision (64-bit). In vector mode, each CE executes floating point instructions at the peak rate of 11.8 million floating point operations per second (MFLOPS) for both single and double precision. A complex of eight CEs can approach a performance level equal to eight times the performance of one CE.

Instruction Set

Each CE is a CMOS gate array implementation of a full scalar architecture, with additional hardware for IEEE floating point operations, vector operations, concurrency control, and virtual memory support. Each CE is a microprogrammed computer with pipelined data and control paths.

The vector instructions, which are integral to the CE instruction set, include logical, integer, and floating point data types, and special instructions for data movement and compound operations, including gather and scatter. Instructions are provided to move vectors between vector registers and memory, perform element-by-element arithmetic, perform comparison and logical operations on operands in vector registers, and perform reduction functions such as summing the elements of a vector. Instructions can operate on one or two vectors, a scalar and a vector, a scalar and two vectors, and two scalars and a vector.

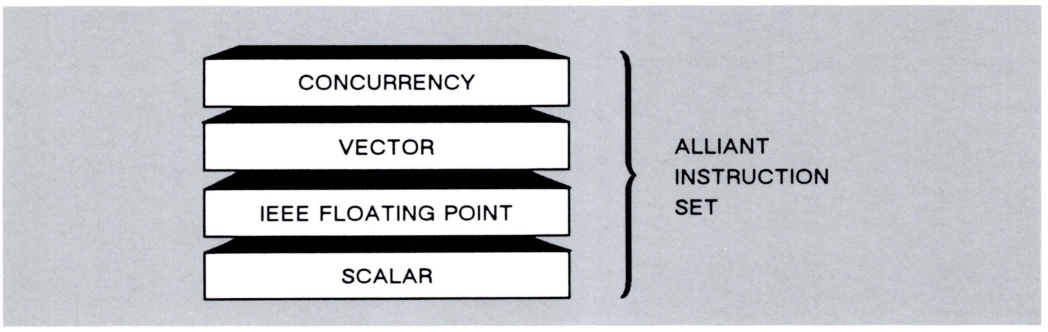


Figure 3-2: The Alliant computational element instruction set

The Alliant FX/Fortran compiler generates scalar, floating point, vector, and concurrency instructions to provide the programmer transparent access to the computational performance of vector and concurrent processing.

The scalar architecture implements two processor modes for instruction execution: supervisor mode for the operating system kernel, and user mode for application software and operating system utilities. Supervisor mode enables the execution of privileged instructions, for example, instructions that affect the allocation of system resources. The two processor modes permit the execution of sensitive instructions without compromising system security.

Data Formats and Registers

All floating point operations, including vector instructions, support single and double precision binary data formats in conformance with IEEE Proposed Standard 754, Version 10.0. Integer data formats are supported for conversion instructions.

Table 3-2: Supported data types

DATA TYPE	ELEMENT SIZE
Binary floating point double	64
Binary floating point single	32
Binary integer longword	32
Binary integer word	16
Binary integer byte	8
Binary coded decimal	4
Bit	1
Bit field	n

CE Design

The computational element is organized into four main functional systems:

- Pipelined instruction unit
- Pipelined vector and floating point unit
- CE switch
- Concurrency control unit

Instruction Unit

The instruction unit is a five-level pipelined processor that allows instructions to be executed every machine cycle provided no resource or data conflicts exist.

The instruction unit includes the instruction cache, control section, instruction processor, and address translation unit.

The instruction cache on each CE consists of 16KB of 85-nanosecond RAM and is addressed on each CE with logical addresses. The instruction cache holds a copy of the current instruction stream and improves execution time for code constructs such as loops that stay within the instruction cache. When an instruction fetch operation detects a miss, the instruction cache controller initiates a load.

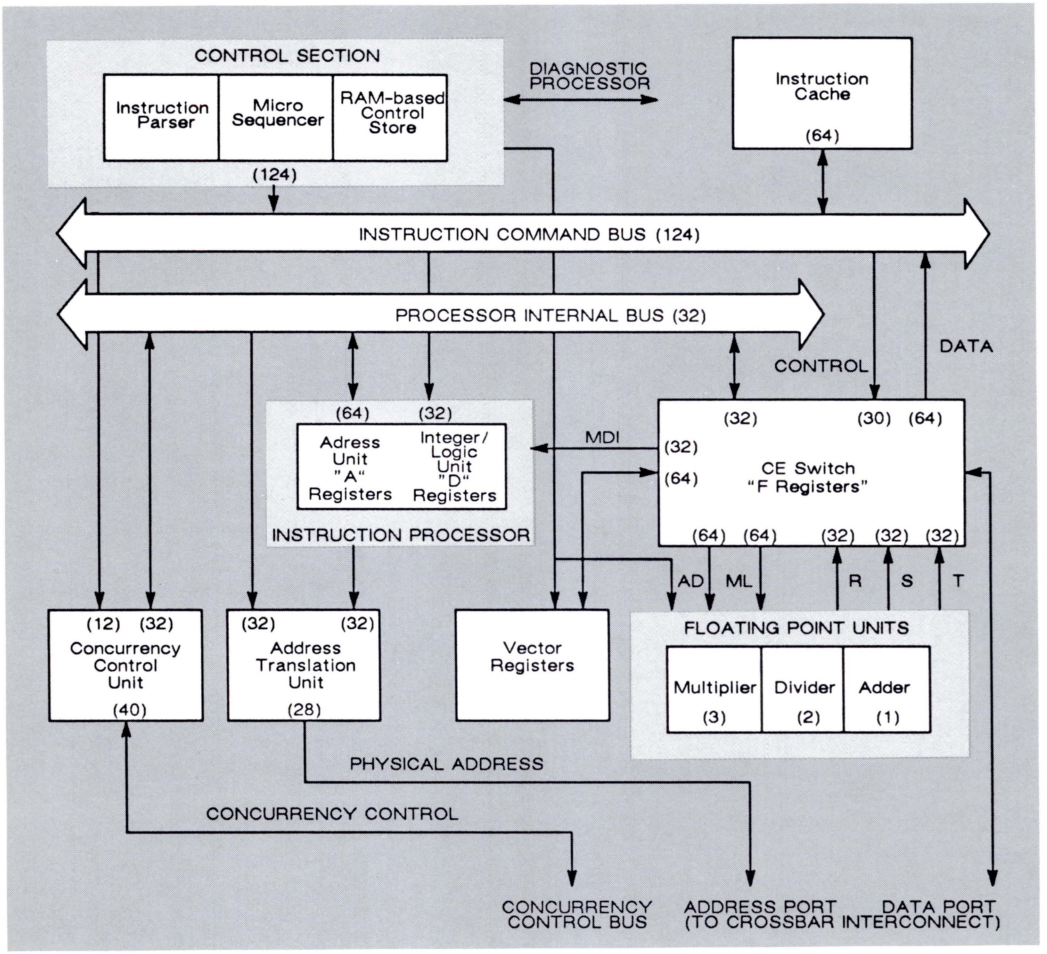


Figure 3-3: Computational element block diagram. Number in parentheses indicate bus widths in bits.

The control section consists of an instruction parser, a microsequencer, and a RAM-based control store. The instruction parser receives opcodes from the data path and decodes them to generate control store microaddresses. The parser also stores the instruction fields (residuals) of the opcodes that are in various stages of execution, checks for dependencies between instructions, and prevents a new instruction from starting when dependencies exist. The parser includes a branch prediction unit that anticipates the most likely flow of program control and prefetches instructions. Branch prediction increases execution speed, especially in programs that contain loops.

The microsequencer and control store memory array perform most of the complex control functions of the CE. These units contain registers for delaying the execution of certain fields of a microword, logic for decoding delayed fields, and sequencing logic for controlling microtraps and unaligned memory references.

The instruction processor consists of the address unit and the integer/logic unit. The address unit is implemented as a pair of 8000-gate CMOS gate arrays and consists of two identical 16-bit slices. The unit contains the instruction buffer, the address data path, and the address unit control logic. The instruction buffer receives the output of the instruction caches, latches it, and rotates 16-bit words to align the opcode, immediate data, immediate addresses, and displacements. The address data path contains eight address registers, the program counter (PC), a 32-bit adder, a variable shifter,

multiplexors, and temporary registers for implementing various addressing modes. The address unit control logic generates address data path controls from opcodes. These opcode-dependent controls are used during the second cycle of each instruction, and can be overridden for subsequent microcode controlled operations.

The integer/logic unit is an 8000-gate CMOS gate array. The unit's 32-bit data path contains eight data registers, four integer temporaries, an ALU, and a full barrel shifter. The unit also contains a programmable two-cycle microinstruction delay shift register to aid in macroinstruction pipelining.

The address translation unit (ATU) performs logical-to-physical address translation and contains the logical address register, which is loaded from the address unit. The ATU includes a translation cache, which stores recently used address translations, and access checking hardware.

Floating Point and Vector Unit

Each CE contains an on-board floating point and vector unit implemented in custom NMOS VLSI and 8000-gate CMOS gate arrays.

Table 3-3: CE floating point processors

OPERATIONAL UNIT	COMPONENTS	TECHNOLOGY USED	PRECISION (BITS)	170 NANOSEC. CYCLES	
				SCALAR	VECTOR
Floating Point Add/Subtract/Convert	1	NMOS VLSI	32	2	1
			64	3	1
Floating Point Multiplication	2	NMOS VLSI	32	2	1
			64	3	1
Floating Point Division	2	8000-GATE CMOS GATE ARRAY	32	9	4
			64	16	7.5
Integer Multiplication	1	8000-GATE CMOS GATE ARRAY	32	4	1

Each CE contains several register sets to handle floating point and vector operations, minimizing cache and memory references.

Table 3-4: CE floating point and vector registers

REGISTER NAME	QUANTITY	LENGTH	WIDTH	PRECISION
Floating Point Data	8	1	64	32/64
Floating Point Status	1	1	32	na
Floating Point Control	1	1	32	na
Vector Data	8	32	64	32/64
Vector Length (d4)	1	1	32	na
Vector Increment (d5)	1	1	32	na
Vector Mask (d6)	1	1	32	na

Each of eight 64-bit floating point data registers holds a 32-bit or 64-bit precision floating point number.

Floating point status and control registers support condition codes (the results of floating point comparison and test instructions), exception codes (branch or set on unordered condition, operand error, overflow, underflow, divide by zero, and inexact result), and trap enabling for exception codes.

Floating point operations include move, conversion, arithmetic, testing and branching, and a hardware implementation of square root. Microcoded transcendentals include ATAN(X), SIN(X), EXP(X), ALOG(X), and COS(X) for both single and double precision.

The eight vector registers each contain 32 64-bit wide elements. Each element holds any data type supported by Alliant, including single or double precision floating point and 8-,16-, and 32-bit integer and logical data types.

A vector instruction set similar to that used on the Cray X-MP supports efficient sparse matrix operations with hardware scatter/gather instructions. A wide variety of multi-operation instructions (dyadic and triadic) are also supported.

Data register 4 is the vector length register and specifies how many vector elements a vector instruction processes. For vectors greater in length than 32, the vector length register is used to divide transparently the vector into subsets of 32 elements that are processed iteratively. For example, a vector of 100 elements is processed in four iterations by decrementing the vector length register by 32 after each iteration.

Table 3-5: Example of processing a vector of length 100

ITERATION	VALUE OF d4	NUMBER OF ELEMENTS PROCESSED
1	100	32
2	68	32
3	36	32
4	4	4

For vector operations, data register 5 is the vector increment register and can be set to specify the stride between vector elements. An increment of two, for example, accesses every other element. The increment can also be negative, in which case elements come from successively lower memory locations.

Data register 6 is the vector mask register and provides flexibility in reduction, merge, split, and sparse data gathering operations. The vector mask register allows element-by-element control of vector operations and is typically set with a vector compare instruction. Vector instructions then use the vector mask register to indicate which vector elements are generated or stored.

Concurrency Control Unit

The concurrency control unit (CCU) is an 8000-gate CMOS gate array that connects the CEs of the computational complex through an independent concurrency control bus. The CCU controls Alliant concurrency (see chapter 2) and assures high parallel processing efficiency. The CCU interfaces with the instruction unit of a CE and up to seven other CCUs to transparently control up to eight CEs running concurrently.

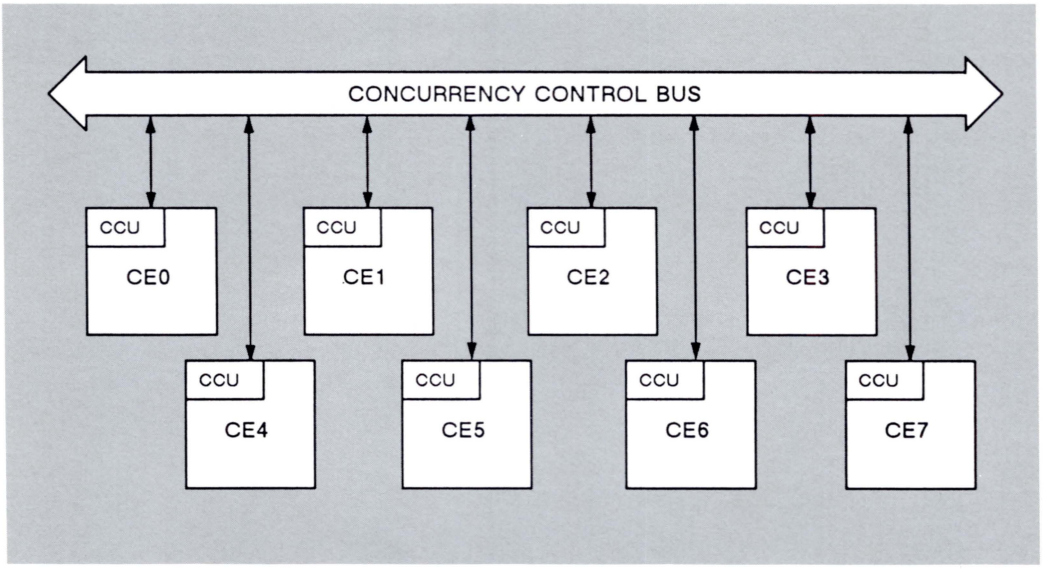


Figure 3-4: Each computational element (CE) contains a concurrency control unit (CCU). CCUs communicate with each other over a dedicated concurrency control bus to control parallel execution.

Since all parallel processing is controlled at execution time, there is no compile- time dependency on the size of the complex. Once programs are compiled to run concurrently, they will execute on a complex of from one to eight CEs. This unique approach allows for field upgradable performance by simply adding additional CEs with no reprogramming, recompiling, or relinking necessary.

Interactive Processors

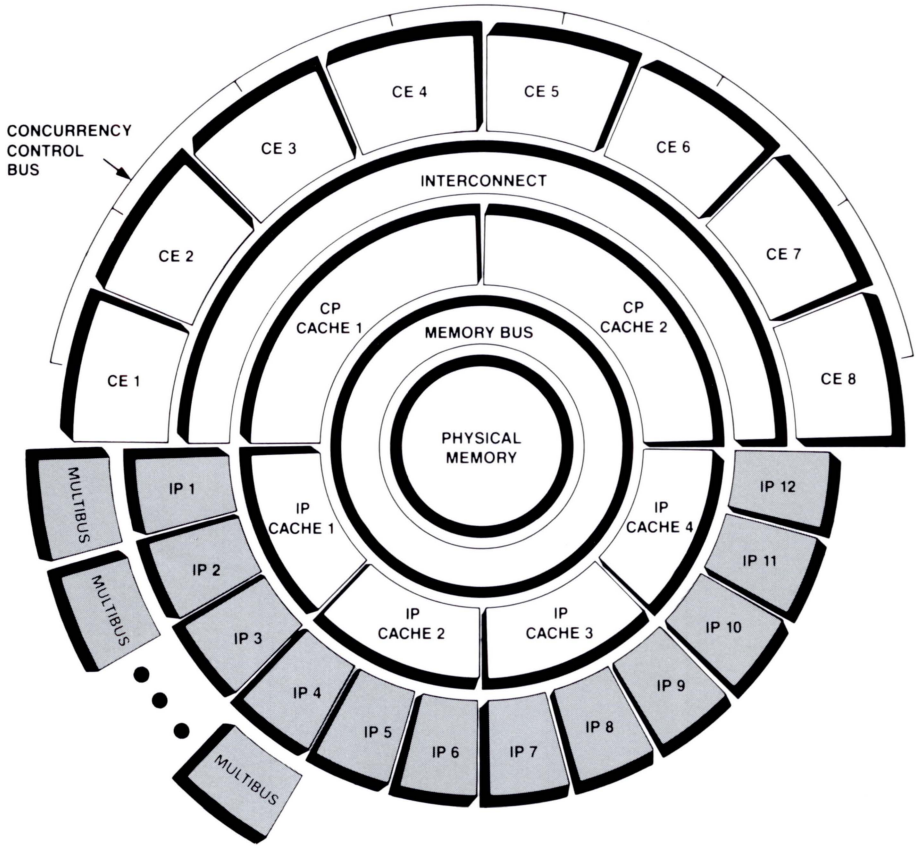


Figure 3-5: Up to 12 interactive processors (IPs) can be configured on an FX/8 system.

Interactive processors (IPs) constitute a separate class of field expandable computing resources in Alliant systems. Two IPs can be configured in an FX/1, up to 12 in an FX/8. IPs off-load the computational complex by executing interactive user jobs, input/output, and other operating system activities in parallel with each other, detached CEs, and the computational complex.

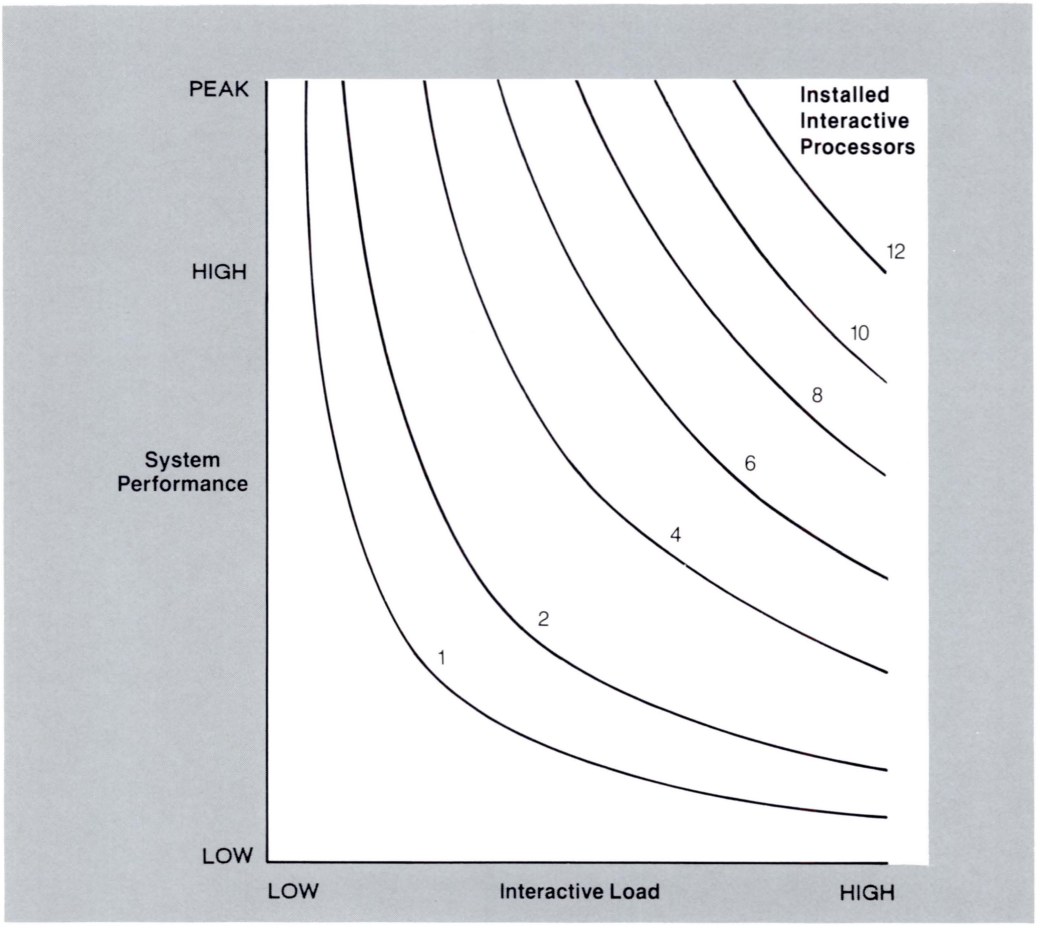


Figure 3-6: Interactive Processors can be added to maintain high responsiveness as the interactive load grows.

IP Design

The IP is an industry-standard Multibus™ card containing a Motorola MC68012 microprocessor operating at 11.76 MHz. In addition, the card contains a virtual memory address translation unit, an I/O map, local parity-protected RAM, power-up EPROMs, and console and remote diagnostic serial ports. The IP interfaces with the IP cache, which provides access to global memory, and to a Multibus (IEEE 796 compatible), which provides access to I/O devices.

Each IP has 512 KB of local memory to cache frequently used operating system kernel text pages, and accesses global memory through a virtual memory architecture identical to that of a CE.

The IPs support the full physical memory of all Alliant systems and are designed with separate address spaces for supervisor and user modes. Virtual address space for users is two gigabytes.

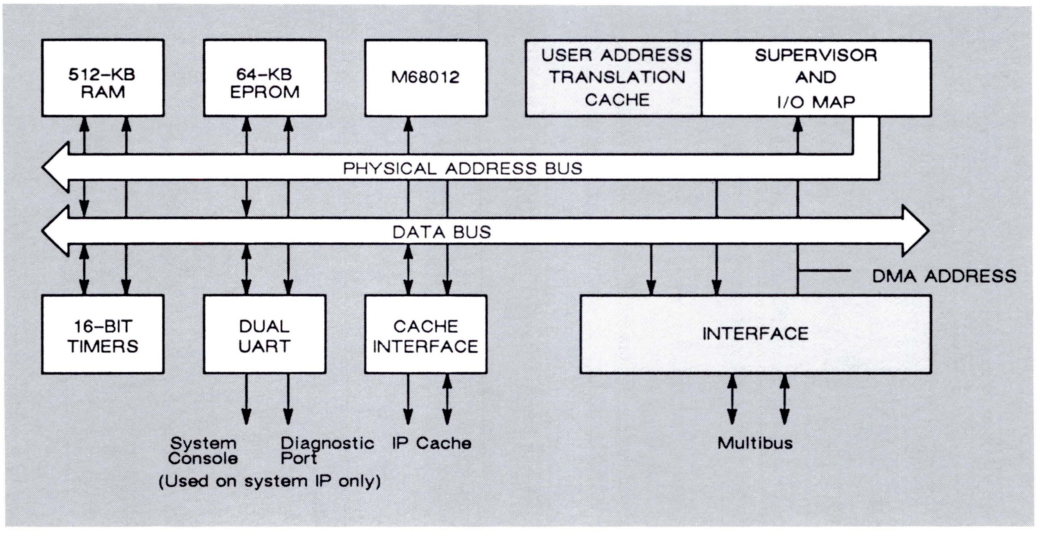


Figure 3-7: Interactive processor block diagram

A direct memory access channel, operating through the I/O map, enables Multibus devices to perform direct memory accesses, including cross-page transfers, anywhere within the physical address space.

EPROMs with a total capacity of 64KB accommodate power-up and boot code to support two initial program load devices. Two RS232 ports provide a console interface and a remote diagnostic port with integral support for an external modem.

System Interactive Processor

The system interactive processor is identical to other IPs except that it connects to the system console and remote diagnostic port. The system IP bootstraps the system, executes diagnostic software, controls the system diagnostic bus (an independent serial bus described in Chapter 8), and handles other system housekeeping tasks.

A local program-controlled clock allows the system IP to function as a totally self-contained processor so that it can diagnose the rest of the Alliant system. The local clock is used at power-up until the IP determines that the system clock is operating correctly. The system IP, the master console, and its boot device comprise the minimum hardcore of the system necessary to support the running of system-wide diagnostics.

IP Performance

A Multibus DMA device can transfer data at 2.5MB per second with the Motorola 68020 processor accessing the IP cache. High system I/O throughput is achieved with multiple IPs. Memory access time is 255 nanoseconds for local memory and 340 nanoseconds for global memory. With both the Motorola 68020 processor and a DMA device transferring data, each IP can sustain a bandwidth of 5MB per second to its associated cache.

Cache And Memory Systems

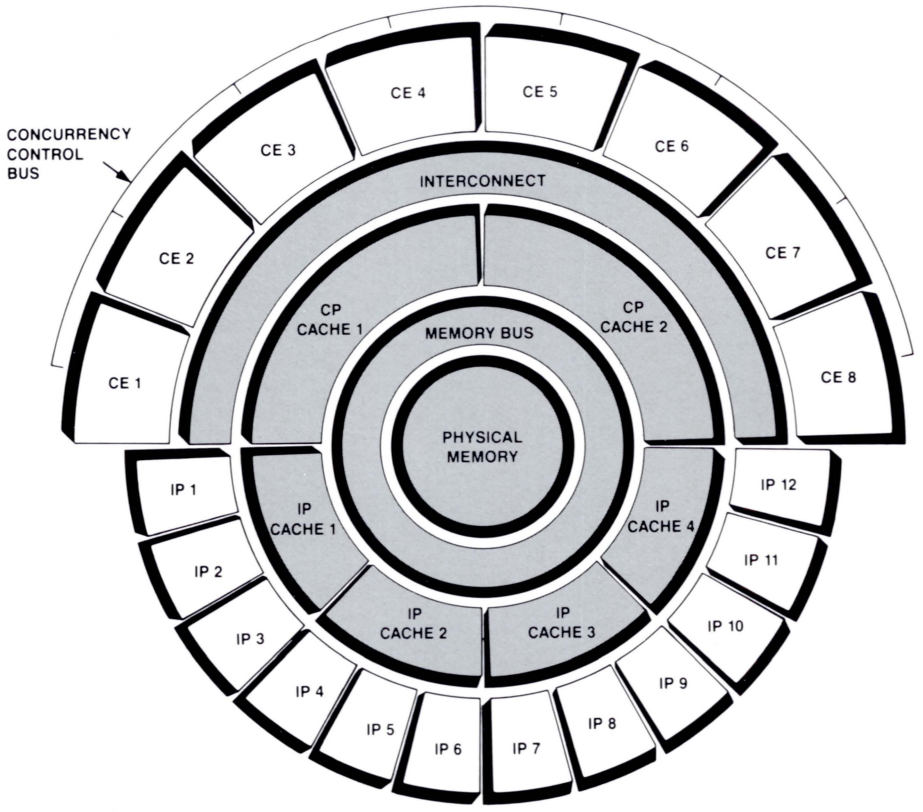


Figure 3-8: The Alliant cache and memory system provides sufficient bandwidth to support up to eight CEs accessing memory simultaneously with no bottlenecks.

Memory system design has proven to be the downfall of many parallel architectures. When multiple high performance processors have to get to the same portion of memory simultaneously, they contend for available memory bandwidth. This type of memory system "lockout" is a primary cause of parallel processing inefficiency.

Alliant has solved the "memory bottleneck" problem by employing parallelism in the memory system design. By combining conventional inexpensive memory with advanced caching techniques, the FX/8 achieves supercomputer memory system performance at superminicomputer prices.

High Performance Memory Systems

Supercomputers such as the Cray X-MP require very high memory bandwidth to support multiple high performance processors. This bandwidth is achieved by interleaving a large number of memory modules. The result is high, but costly, performance.

Superminicomputers typically use architectural features such as various forms of caching to achieve their memory system performance. For example, upon a cache miss, a superminicomputer cache might obtain the entire block containing the required word. Because the next transfer generally is for the next sequential word, the cache design results in reduced transfer time. In addition, the cache holds blocks that have been accessed recently. Code loops are accelerated since the probability is high that the loops are in the cache.

In addition to increased speed, the Alliant memory architecture uses cache for:

- a high degree of interleaving
- a great degree of accessibility
- avoiding memory lockouts on a common bus

Computational Processor Cache

The FX/8 computational processor cache (CPC) serves as an interleaved high-speed physical memory buffer for the computational complex. A single cache module can serve as a two-way interleaved 256KB cache for up to four computational elements; two caches provide a four-way interleaved 512KB cache with a maximum bandwidth of 376MB per second.

As opposed to a distributed cache architecture (a cache on each processor), the shared interleaved cache approach allows for:

- Implicit prefetch: Concurrent processing of a loop implies that several processors can each use a portion of the data fetched from any one cache block. A distributed cache architecture forces data to be passed back through memory when one processor needs data residing in a cache on another processor.
- Avoidance of bank contention in main memory: The cycle time of the cache memory is much less than that of main memory, so that conflicts arising from multiple processors attempting to access the same code or data can be resolved much faster.
- Fast data passing between processors: The results of one loop iteration are immediately available to a different processor executing the next iteration of the same loop. This helps to achieve high efficiency in processing data dependent loops.

The CPC is designed to support multiple cache accesses in parallel. A typical uniprocessor superminicomputer cache keeps supplying data until it encounters some type of unresolvable conflict such as a cache miss. Then it has to stop until the conflict is resolved. In the Alliant architecture, if one processor stops because of a cache miss, the others must be able to continue. To support this, the CPC continues to supply data while resolving a conflict. Up to three cache accesses can be pending in each of its four banks, for a total of up to 12 accesses pending in the entire cache.

Interactive Processor Cache

The interactive processor (IP) caches are field expandable on the FX/8 from one to four 32KB modules of 85-nanosecond memory, for a maximum of 128KB. Each cache supports up to three interactive processors and contains features designed to maximize interactive performance and availability.

The IP caches interface the interactive processors and their associated I/O devices with the Alliant memory bus at a bandwidth of 94MB per second. The IP caches block and unblock global memory, maintain maximum bus efficiency, and enhance interactive processor performance by accelerating program code fetches and I/O transfers.

IP caches maintain system-modify bits to enhance system performance by minimizing disk writes during paging. The modify bits maintain a current record of the status of every physical page in global memory. The modify bit logic monitors the memory bus to record when a page of global memory is modified. All caches in the system broadcast over the memory bus when they are going to write a block of memory for the first time. A cache that wants to write any given block must first obtain a "unique" copy of the block. Thus, while any cache may have a copy of a block, only one cache can have a writable copy. A modify bit is set only if a processor is going to modify a page, thus eliminating I/O activity for pages that are not modified.

Each IP cache supports three external ports: two 11.76MB-per-second 16-bit ports for interactive processors and one 64-bit port. In the FX/1, the 64-bit port supports a single computational element with a bandwidth of 47.06MB per second. In the FX/8, the 64-bit port is reconfigured to an 11.76MB-per-second 16-bit port and used by a single interactive processor.

Table 3-6: Interactive processor cache I/O bandwidth

INTERFACE TYPE	CONTINUOUS BANDWIDTH (MB PER SEC)		
	EACH	MAXIMUM AVAILABLE FX/1	FX/8
Memory Bus to IP Cache	94.1	94.1	188
IP Cache to FX/1 CE	47.0	47.0	na
Cache to 16-bit port	11.7	23.4	35.1
IP Cache to IPs	5.0	10.0	60.0
IP to Multibus	2.5	5.0	30.0

The IP cache contains the master system clock and a high resolution counter that can be used by both users and the operating system for performance and timing analysis. The master system clock runs at 85 nanoseconds. The clock can be margined at the system console, or remotely through the remote diagnostic port, to 81 or 100 nanoseconds by service engineers to help in isolating intermittent faults.

Cache Coherency

All processors in the system (both CEs and IPs) share a common view of global memory that is independent of the multiple cache architecture. For example, an editing job running on an IP attached to a certain IP cache can get swapped out and resume execution on an IP attached to a different IP cache. Alliant cache coherency guarantees that the editing job will automatically have the most current copy of the data when it resumes execution.

Some systems achieve cache coherency by not caching certain data, or by using numerous cache flushing operations. Both of these methods are inefficient. Alliant solves this problem by implementing the entire memory system coherency scheme in hardware. Hardware cache coherency is totally invisible to the programmer, the processors in the system, and the operating system. This means that any job can execute on any processor at any time without having to worry about "stale" or invalid data being used.

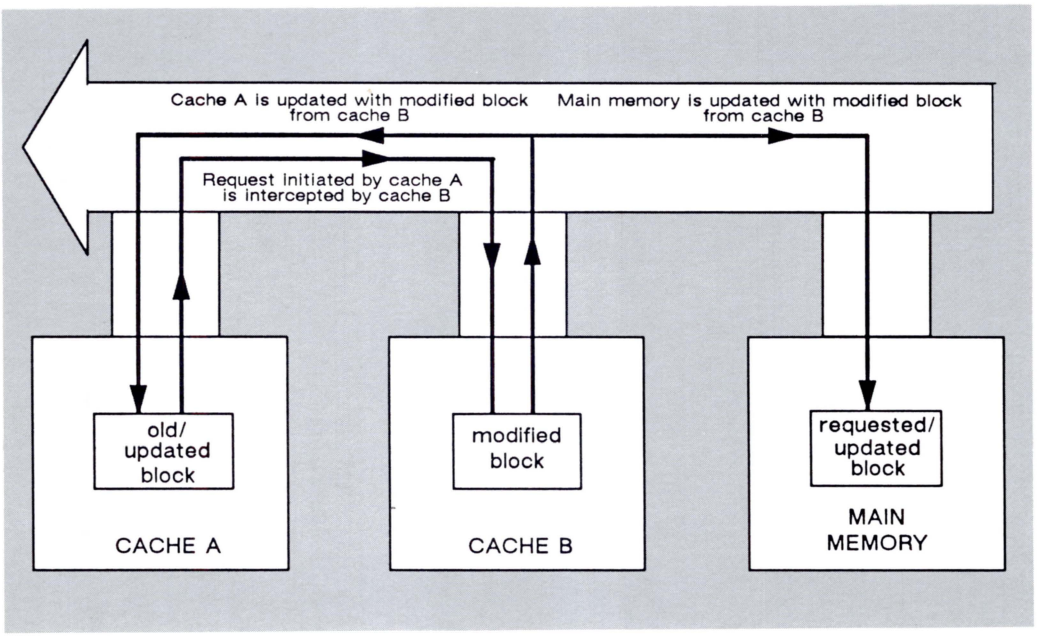


Figure 3-9: Cache coherency is automatically handled in hardware on Alliant systems.

Figure 3-9 shows the hardware coherency scheme in action. Cache A has just requested a copy of a block from main memory. However, a copy of this block has been modified and is currently residing in cache B. Cache B monitors the memory address bus and detects that a request is being made for a copy of a block that it has modified. Cache B intercepts the memory read operation initiated by cache A and sends a copy of its modified block to cache A and to main memory. If cache A has requested a unique (or writable) copy of the block, then cache B will automatically invalidate its copy of the block.

Write-Back Cache Architecture

The write-back architecture of the Alliant caches reduces memory bus traffic by updating main memory only when a modified block is replaced in the cache or when a processor connected to another cache initiates a read for modified data. The write-back design allows multiple consecutive stores such as vectors to occur at the full bandwidth of the computational processor cache, which is twice that of main memory.

The write-back design is equally efficient for memory reads and writes. A write-through cache design stores all data back immediately to main memory. While solving such problems as main memory bank contention for read operations, the write-through design would not provide the same advantage for write access.

Another advantage of the write-back design is efficient utilization of the memory bus bandwidth. Because all traffic on the main memory bus is block traffic, the memory does not have to deal with individual words or bytes. This means that the true effective transfer rate of the memory bus is equivalent to its peak rate of 188MB per second.

Crossbar Interconnect

The crossbar interconnect dynamically connects up to eight computational elements with up to four cache ports. The crossbar design avoids the "lockout" problems associated with multiple processors contending for memory over a common bus.

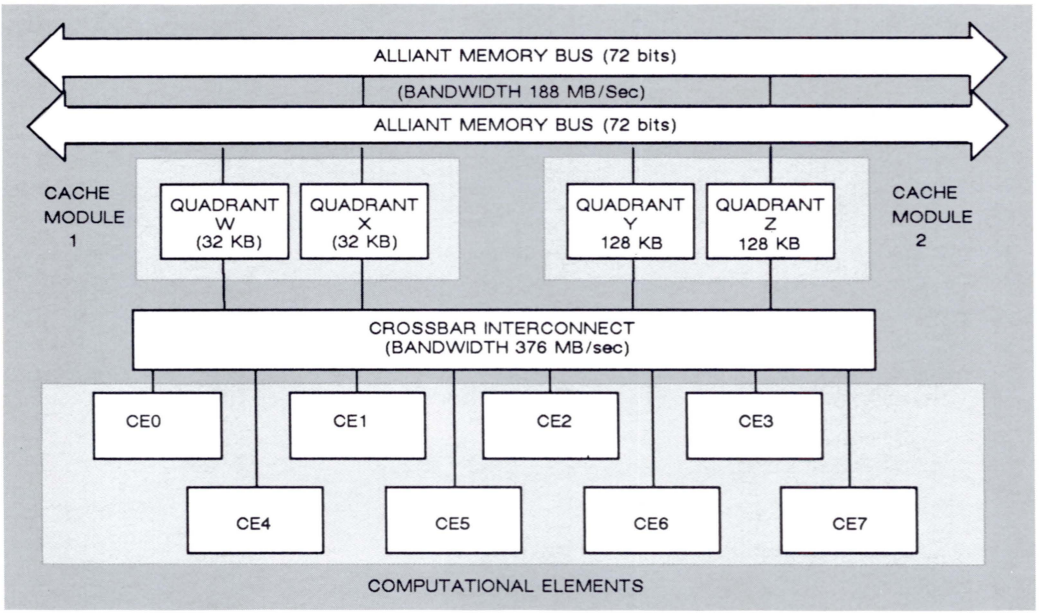


Figure 3-10: The crossbar interconnect dynamically connects up to eight computational elements with up to four cache quadrants providing a peak bandwidth of 376 MB per second to the computational complex.

The crossbar interconnect provides sustained bandwidth of 376MB per second to the computational processor cache when required data is in the cache. The interconnect is a 4-by-8 address and data switch implemented in 24 2600-gate gate arrays. The computational processor cache and the crossbar interconnect are designed to support multiple computational elements and are therefore used only on the FX/8. The FX/1 does not require the interconnect as each processor has a dedicated cache port.

Alliant Memory Bus

The Alliant memory bus is a high-speed, synchronous memory access bus that consists of two 72-bit-wide data paths (64 bits of data, plus eight bits for single-bit error detection and correction and double-bit error detection), a 28-bit address bus, and a control bus. The data busses are bidirectional and are driven by memory modules, the computational caches, and the IP caches.

Data is always transferred as four eight-byte words (a "cache block"), with the first word transferred on data bus A. Thereafter, the data alternates between data bus A and data bus B for four cycles. Another transfer can be interleaved with the first transfer to allow a maximum of eight eight-byte words to be transferred in four cycles. A data bus cycle time of 85 nanoseconds provides a total bus bandwidth of 188MB per second.

The data busses use a "convenient word first" protocol that accelerates accesses that may not be to the first word of a cache block. The least significant bit of the address points to the pair of eight-byte words within the cache block. Data transfers can be

ordered 0–1–3–2 or 3–2–0–1. (The numbers refer to the eight-byte word address within the cache block.) Any desired word is at most only one word away.

The memory bus allows up to eight block accesses to be in progress at any one time in order to provide effective use of its available bandwidth.

The address bus has twice the bandwidth required to keep the data busses at maximum utilization, which permits data bus masters to use several types of address-only transfers to control the multicache environment.

Memory Modules

Physical memory in Alliant systems currently uses either 256Kb or 1 Mb dynamic RAMs and is field expandable in 8 or 32-megabyte modules. Four-way interleaving on each memory module permits any module to supply the full bus bandwidth of 188MB per second for sequential read accesses, and 80% of the bandwidth, or 150MB per second, for sequential write accesses. Each module can be dynamically reconfigured to either 3/4 or 1/2 capacity in order to bypass hard component failures.

Memory modules also monitor and test the parity of the address bus.

Front Panel

All Alliant products use a common front panel module (Figure 3-11). The front panel controls the switches and power supplies, and contains a Motorola 68701 processor that connects to the system diagnostic bus (see chapter 8).

All switches, except the power switch, are membrane type. Non-volatile RAMs maintain their state across power failure so that, for example, the system can be configured to boot automatically from a specified device after a power failure leaving remote diagnostics enabled.

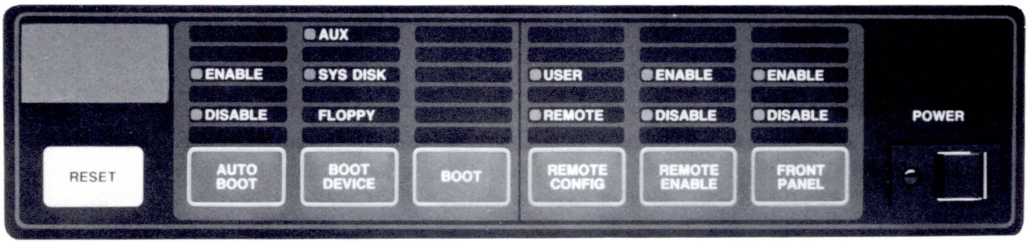


Figure 3-11: The system front panel module is based on a Motorola 68701 processor.

The front panel also contains a battery backed-up clock calendar that is read when the system is booted and reset, and power supply controls that are used to margin the system and isolate intermittent failures.

All Alliant systems contain thermal sensor receivers to monitor operating temperature. The system console indicates when the temperature limit is near the maximum safe level. Power is removed from the system when the maximum temperature is exceeded.

Product Families

All Alliant systems are compatible and upgradable. Only delivered performance and the level of expandability distinguish the two families.

FX/1

The FX/1 computer consists minimally of a CE, an IP, an interactive processor cache, and 8MB of physical memory. The FX/1 can be expanded with an additional IP and another 8 or 32MB (for up to 64MB) of physical memory. The FX/1 is a multiprocessing system where the CE and IPs are independent processors that share a common operating system kernel. Each processor executes a user job or operating system process in parallel with the other.

The FX/1 delivers 5.05 KWhetstones and 11.8 MFLOPS peak floating point performance in a desk high package that takes less physical space and power than many workstations.

Designed for use in an office or laboratory environment, a complete FX/1 system is only 72.4 cm high, 34.3 cm wide, and 63.5 cm deep. Weighing only 57 kilograms fully configured, the FX/1 runs on 15-amp, 115-volt wall current and has a heat output of less than 1200 watts. In the system cabinet, space is provided for a Multibus chassis, and three 5 1/4" peripherals. With an I/O expansion cabinet, an FX/1 system can be expanded to accommodate two 268MB SMD Winchester disk drives. An optional tri-density tape drive is also available.

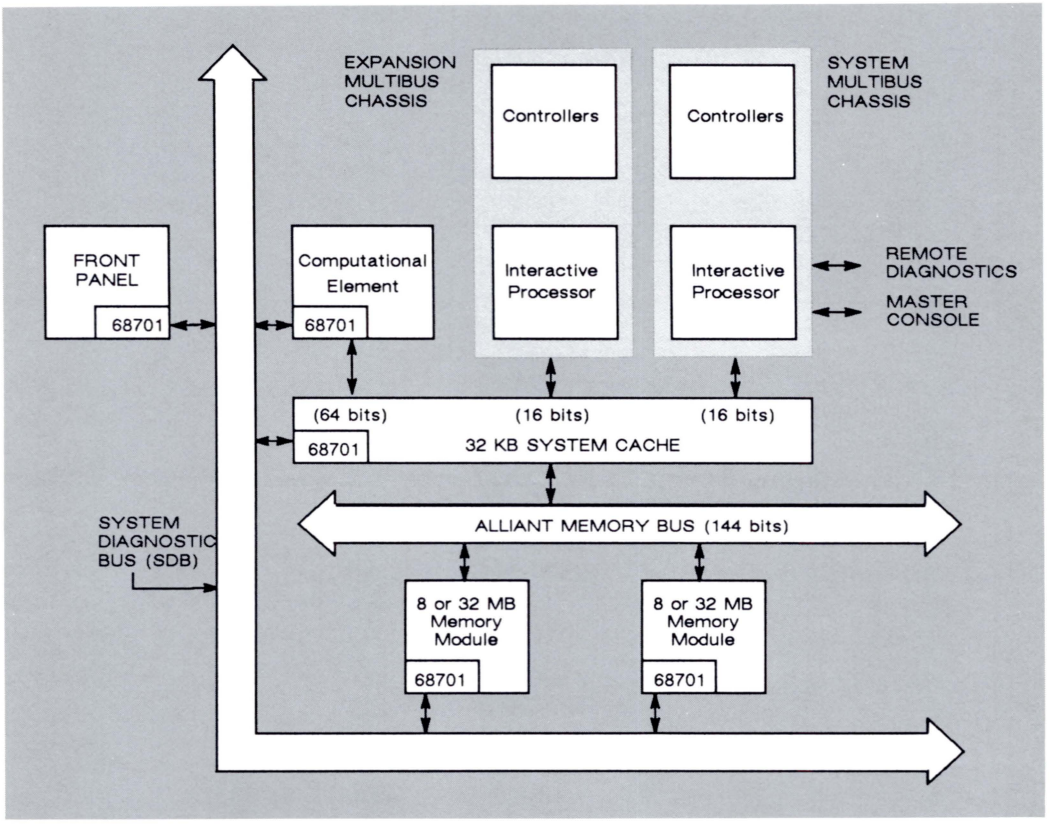


Figure 3-12: Alliant FX/1 block diagram

FX/8

The FX/8 computer adds Alliant concurrency to the multiprocessing used on the FX/1. An FX/8 can grow from the performance of an FX/1 to an eight-CE system with delivered performance that approaches eight times that of an FX/1.

The FX/8 allows for incremental field expansion of both computational and interactive processing capabilities. An entry-level system consists of a CE, a computational cache, 16MB of memory, an IP cache and at least two IPs. As computational needs increase, additional CEs can be added and the computational cache expanded in the field. As the interactive load increases, additional IPs and IP caches can be added. In addition, physical memory can be increased to a maximum of 256MB.

FX/8 systems are configured in a system cabinet 110 cm high by 80 cm wide. Space is provided in the system cabinet for up to eight CEs, 512KB of computational processor cache, 128KB of interactive processor cache, and 256MB of memory. A fully configured system dissipates less than 5000 watts

FX/8 IPs and their associated peripheral controllers are installed in Multibus chassis mounted in I/O expansion cabinets. Additional expansion cabinets can be installed to support large numbers of peripheral devices. Interactive performance is also easily expanded.

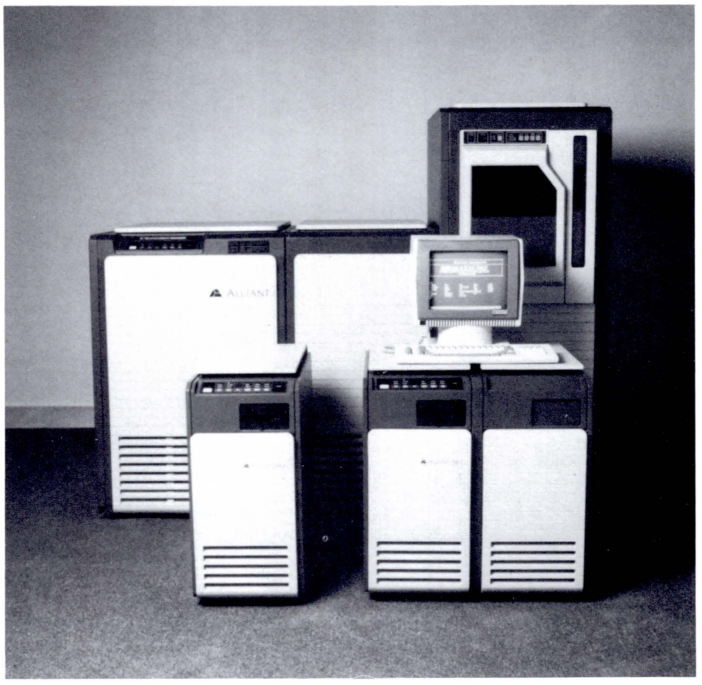


Figure 3-13: The Alliant FX/Series: the FX/8 (background), the FX/1 (left foreground), and the FX/1 with expansion cabinet (right foreground)

CHAPTER 4

Concentrix Operating System

The Concentrix Operating System is the Alliant implementation of the UNIX operating system. Concentrix is a general-purpose, time-sharing operating system for scientific and engineering applications.

Concentrix is based on the 4.2BSD version of the UNIX operating system, the extended UNIX developed at the University of California at Berkeley.

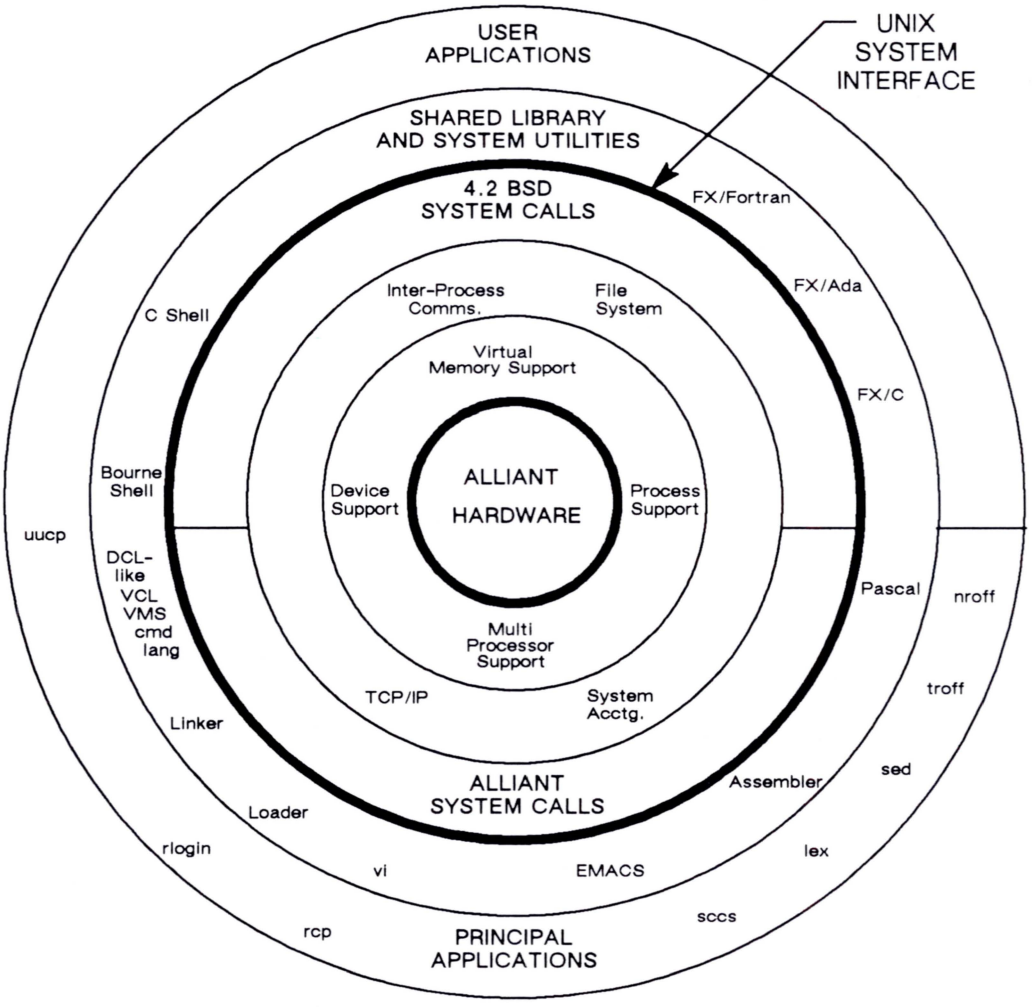


Figure 4-1: The Concentrix Operating System extends the Berkeley UNIX operating system with support for multiple processors and large physical and virtual memory.

Concentrix Kernel

The Concentrix operating system is structured in three layers: the kernel, the shell, and a range of languages and utilities. The kernel is the machine-dependent core of Concentrix and contains the basic hardware interfaces, process control software, and memory management software. The Concentrix kernel has been enhanced for improved performance and supports the following features:

- Multiple computing resources
- User-tunable scheduler
- Multiprocessing
- Macrotasking
- Mapped file I/O
- Demand-paged virtual memory
- Copy-on-write process creation
- Shared user library
- Shared memory
- Fast file system
- Track-allocated file system
- Striped file system
- High bandwidth paging/swapping I/O
- Device auto-configuration
- Workstation networks
- Integral networking
- Real-time applications

Support for Three Classes of Computing Resources

Concentrix maintains queues of ready-to-run jobs for three separate classes of computing resources: IPs, detached CEs, and the computational complex. Concentrix schedules all three resource classes simultaneously, and switches jobs between resource classes to maintain optimum computational throughput.

The IP queue contains operating system processes and interactive user jobs such as editors. Concentrix schedules these jobs for any available IP. If no jobs are waiting to run in the detached CE or computational complex queue, Concentrix utilizes these resources to offload the IPs. This automatic load balancing assures sustained interactive performance.

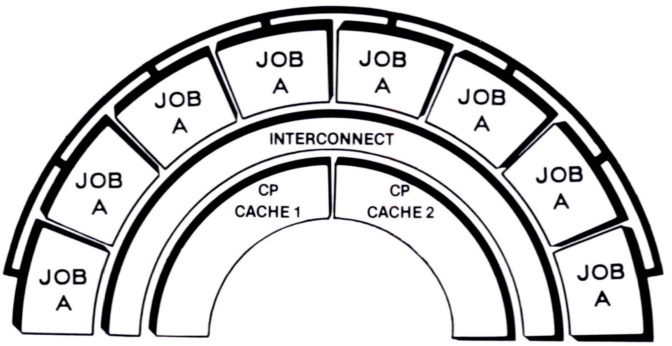
Any number of CEs can be declared to be "statically detached" from the complex at system boot time. The "detached" CE queue contains jobs identified by the user as not requiring the complex for execution, such as smaller production jobs and compilations. Concentrix schedules these jobs for any available CE.

The computational complex consists of up to eight CEs that work on one job simultaneously to reduce time-to-solution. Concentrix views the complex as a single computing resource, and maintains a queue of ready-to-run complex jobs. To the user, the complex appears as a normal time-shared computing resource. If a job running on the complex requires system services or reaches the end of its time slice, Concentrix "collapses" the complex, places the job back on the complex run queue, and starts another complex job.

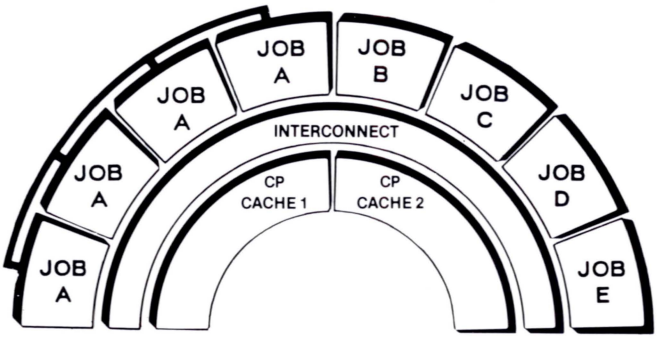
User Tunable Scheduler

Under control of Concentrix, the computational complex can quickly break apart into separate "detached" CEs. These "dynamically detached" CEs become available to Concentrix as additional resources for jobs in the detached CE queue. Through software control, the system manager can specify the percentage of time that the complex runs in both "attached" and detached modes. For example, during the day, when the program development and interactive user load is high, detached CEs can be favored to insure high user productivity, while still maintaining the complex as a resource for production execution. Three modes of execution are available through an easily tuned scheduler:

- **Traditional complex mode** - In complex mode, all installed CEs are always applied to the execution of a single program. This mode is ideal for production installations where the job mix is dominated by computationally intense programs that demand the fastest possible time-to-solution.



- **Static detached mode** - In this mode, one or more of the installed CEs is detached from and operates in parallel with the complex. Static detached mode insures that a fixed resource is always available to single CE jobs.



- **Dynamic mode** - In this mode, CEs in the complex are load and time-multiplexed between "together" as a complex and "detached" as independent processors. In dynamic mode, the user allows the operating system to maximize the utilization of all system resources providing support for both single CE and complex jobs.

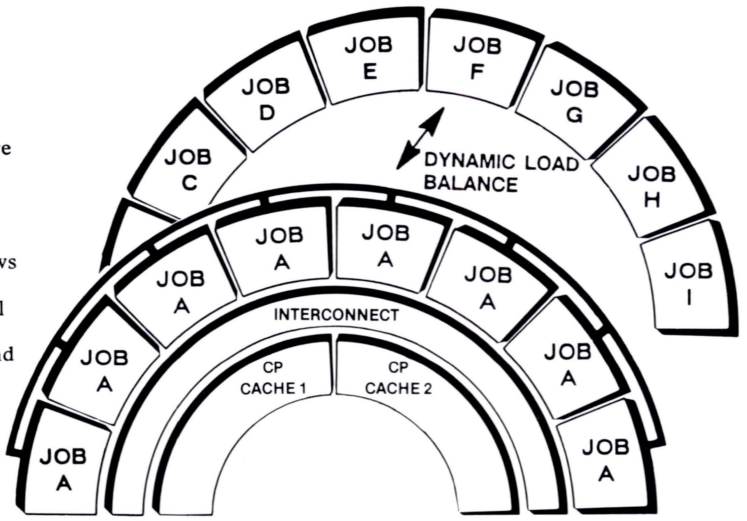


Figure 4-2: Concentrix supports three modes of execution through an easily tuned scheduler.

Multiprocessing

Detached CEs (statically and dynamically) comprise a resource class that is used by Concentrix to support multiprocessing. Multiprocessing delivers high system throughput by allowing multiple UNIX processes, each with its own logical address space, to run simultaneously on multiple detached CEs. These processes can either be independent user programs or tasks within a single computationally intense program.

Multiprocessing on detached CEs takes place in parallel with activity on the computational complex and IPs. Multiprocessing achieves:

- Increased throughput – Multiple jobs can run simultaneously with each other.
- High multiuser responsiveness – Many small jobs can run simultaneously and take advantage of CE time when no jobs are executing concurrently.
- Support for a mix of production and development work – Compilations, which do not run concurrently, can share the resources of the system with programs that need the performance of Alliant concurrency.
- Expandability - Adding additional CEs increases total system throughput.

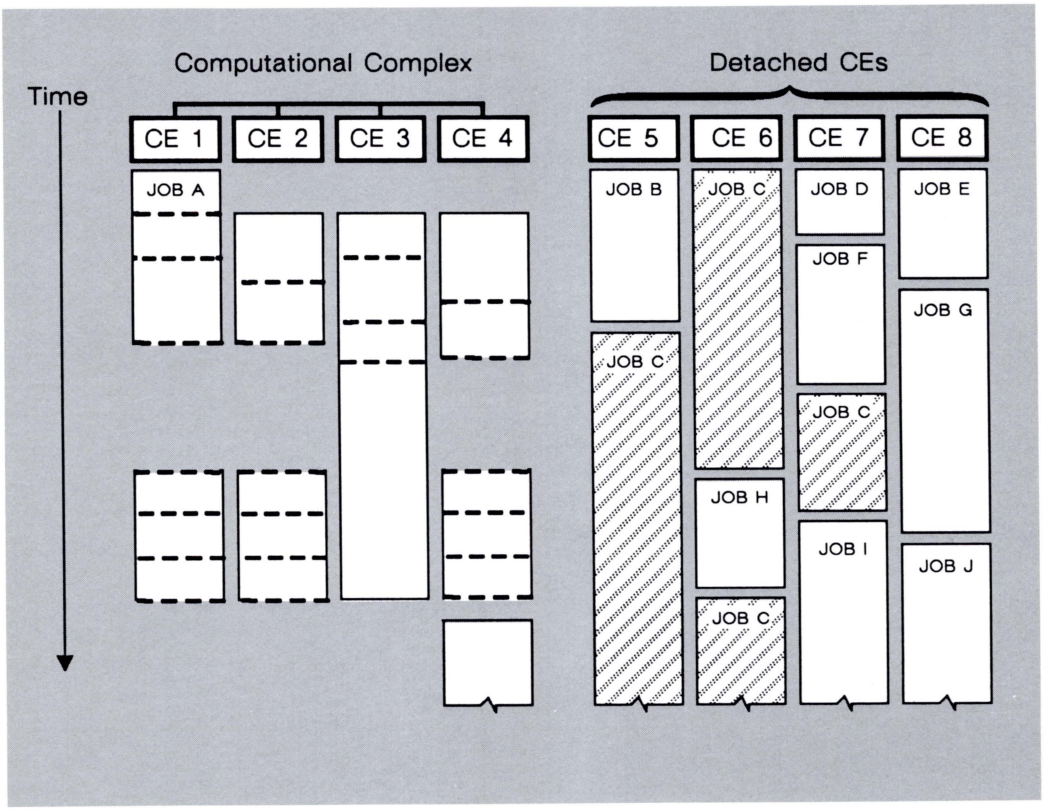


Figure 4-3: A complex of four CEs runs one job concurrently while four detached CEs multiprocess smaller jobs. Job C depicts a macrotasking program.

Macrotasking

The multiprocessing capabilities of the FX/8 allow Alliant users to take advantage of macrotasking. Macrotasking (sometimes referred to as multitasking) is "the process of dividing a program into segments that are then executed simultaneously on multiple processors[1]."

Macrotasking achieves high performance in many applications where parallelism matches natural problem boundaries. For example, in complex circuit simulation problems, separate processes can be generated for each device being simulated. These processes can then be executed simultaneously on detached CEs.

Concentrix supports two different forms of macrotasking. The standard UNIX "fork" utility can be used to generate task-level processes that are multiprocessed on detached CEs. The fork utility duplicates a process with the only difference between the two executing processes being the value returned by the fork utility. Fork returns the process identifier of the subprocess to the parent process, and a value of zero to the subprocess. Each subprocess has access to the full state of the parent process as it existed at the time of the fork. Using library features that support shared memory and process synchronization (locking), programmers can develop single applications that execute as coordinated, but independent, processes on detached CEs.

In Figure 4-4, the parent process assigns a different task to each of four subprocesses, then exits the loop and continues its own processing.

```
INTEGER PID, FORK
.
.
.
DO  I = 1, 4
      PID  =   FORK( )
      IF   ( PID .EQ. 0 )   THEN
           CALL    TASK ( I )
           STOP
      END    IF
END DO
```

Figure 4-4: The UNIX fork command generates task-level processes that are multiprocessed on detached CEs.

This form of macrotasking on Alliant systems is fundamentally the same as macrotasking on other parallel vector processing supercomputers. As a result, Alliant systems can be used as a development environment for applications targeted at supercomputers such as the Cray X-MP and Cray-2.

Alliant's FX/Ada provides a form of macrotasking tailored for Ada. The task is a natural construct of the Ada language. The FX/Ada runtime system automatically and dynamically assigns tasks to the appropriate resources (CEs, IPs, or complex) that have been allocated to the Ada program.

In addition to task-level multiprocessing on detached CEs, macrotasking can also occur within the computational complex. This form of macrotasking uses the concurrent call (CNCALL) directive and dedicated concurrency control hardware to achieve extremely low overhead. The CNCALL directive allows the computational complex to execute each iteration of a loop on a separate CE even when those iterations contain one or more subroutine calls. Efficient data sharing and synchronization are made possible by having all CEs in the complex share the same address space and a common cache.

```
 1              Program Macrotask
 2
 3              Parameter (N_TASK = 4)
 4
 5              CALL INPUT_DATA
 6
 7     CVD$ CNCALL
 8            DO 10 ITASK=1,N_TASK
 9                 IF (ITASK .EQ.1) THEN
10                    CALL TASK1
11                 ELSEIF (ITASK .EQ. 2)THEN
12                    CALL TASK2
13                 ELSEIF (ITASK .EQ. 3) THEN
14                    CALL TASK3
15                 ELSEIF (ITASK .EQ. 4) THEN
16                    CALL TASK4
17                 ELSE
18                    STOP
19                 ENDIF
20         10  CONTINUE
21
22             CALL OUTPUT_DATA
23
24             END
```

Figure 4-5: The CNCALL directive is used to achieve macrotasking within the computational complex. Each iteration of the loop calls the task (subroutine) corresponding to the current iteration number.

Mapped File I/O

Standard UNIX I/O uses a buffering system that moves data from the disk to a buffer and then to the user address space (Figure 4-6). This slows disk I/O by introducing considerable system overhead into all disk I/O operations. Concentrix uses mapped file I/O to overcome these limitations.

Mapped file I/O uses the high-speed virtual memory system to manage physical memory as a "window" into a disk file (Figure 4-6). When a program reads or writes data, the virtual memory system transfers a large block of data directly into or out of the user address space, bypassing the buffers used in other UNIX systems.

Mapped file I/O utilizes direct transfers between disk and the user's address space to maximize I/O speed. Bypassing the operating system buffer pool reduces processor involvement in data movement, making more processor cycles available for user program activity.

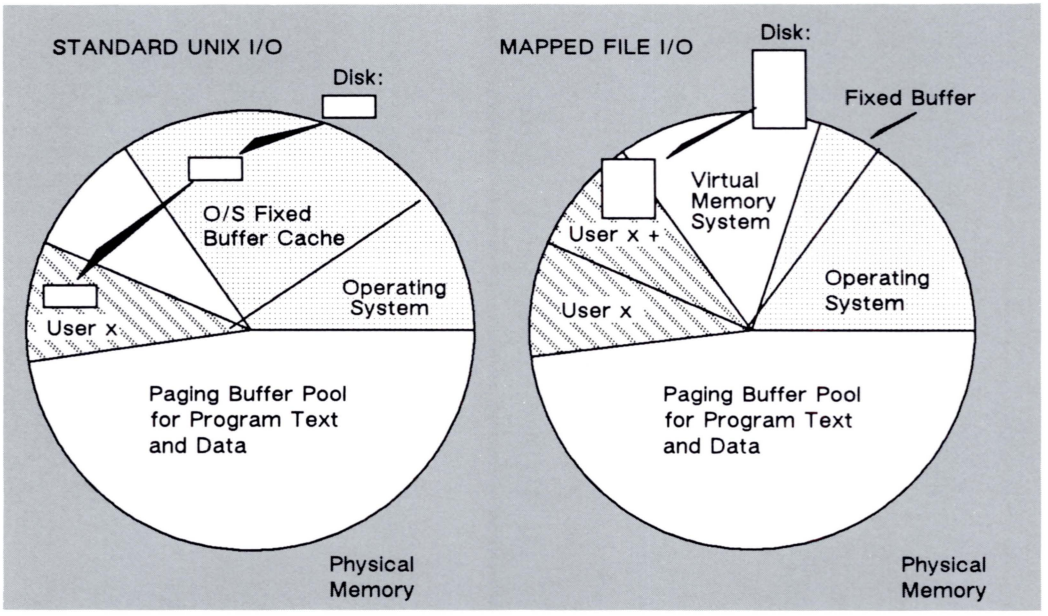


Figure 4-6: Standard UNIX I/O uses a buffering system for disk I/O. Alliant mapped file I/O allows direct transfers between disk and physical memory, bypassing the the buffered I/O of standard UNIX

Mapped file I/O is transparently available through FX/Fortran I/O for both sequential and random files. Existing programs that use the Fortran READ and WRITE statements can take advantage of mapped file I/O without modification. In addition, programs written to access UNIX standard I/O directly continue to run without modification, coexisting with Concentrix mapped file I/O.

Demand-Paged Virtual Memory

Concentrix memory management is demand-paged and supports a physical address space of 256MB and a virtual address space up to 2GB per process.

Copy-On-Write Process Creation

Concentrix supports shared code and data between processes with a copy-on-write feature speeding process creation. In Concentrix, only pages that are written are copied, resulting in low overhead process creation and efficient use of physical memory.

Shared Library

The Concentrix operating system links pointers, rather than actual library routines, into executable programs. As a result, object programs have smaller disk images and Alliant supplied library support is isolated for maintenance.

Shared Memory

Concentrix allows independent processes running on IPs, detached CEs, or the computational complex to share memory. For detached CE or IP processes, library routines provide support for shared memory, mapped files, and a process synchronization locking capability to administer mutual exclusion within a shared memory segment.

Within the computational complex, all processors implicitly share an identical address space since the complex is a single resource that only runs one process at a time.

Fast File System

Concentrix supports the fast file system of 4.2BSD UNIX, which includes techniques for clustered file placement, resulting in improved disk transfers.

Track-allocated and Striped File Systems

In addition to mapped file I/O, Concentrix provides two facilities for very high system I/O throughput: track allocation and disk striping. Both are intended primarily for file systems that contain large files, typically application database files, or highly-active file systems such as /tmp.

In track allocation, the basic unit of disk space allocation is a full track, rather than 4 KB as in the standard file system. The actual size of a track varies among disk types; typically it is 32 KB. The larger unit size results in fewer reads and writes per I/O request and less disk fragmentation.

In disk striping, a track allocated file system is distributed over multiple disks so that I/O requests on a single file can be performed by several disk controllers working simultaneously.

Gordon Bell, director of the Supercomputer Project for the National Science Foundation observed that:

> ...we have no parallelism in the disk area yet; that's called balance, and that's the stuff that never gets talked about. [2]

Alliant's disk striping fulfills this need by extending Alliant's parallel architecture to the file system.

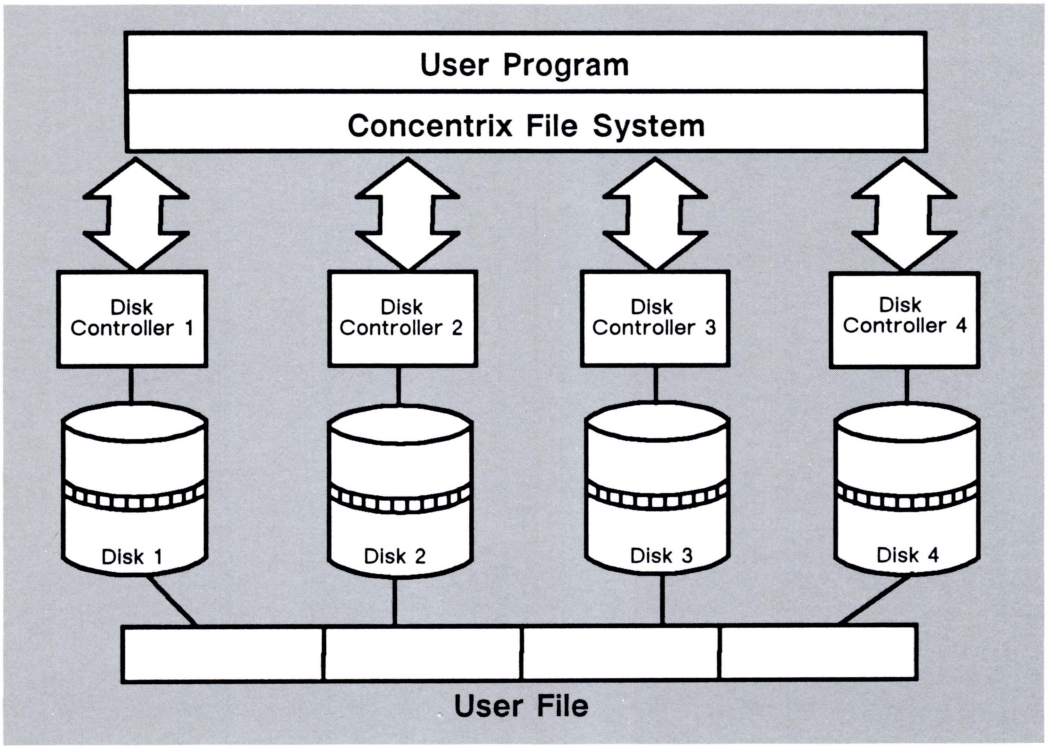


Figure 4-7: Alliant's disk striping distributes individual files across multiple disks so that up to four disks can be accessed in parallel by a single application to provide a 400% improvement in disk I/O performance.

High Bandwidth Paging and Swapping I/O Support

While the large physical memory of Alliant systems minimizes swapping, Concentrix provides an additional large-block read/write capability that results in very fast paging and swapping when required. Scatter-read and gather-write I/O can be accomplished in blocks of up to 1MB with a single transfer.

Device Auto-configuration

At system boot time Concentrix performs an auto-configuration. Concentrix interrogates all IPs for Alliant supported devices, and automatically performs the logical to physical mapping of device to device name. Custom user devices can be accommodated in a standard configuration file. Devices can be moved from one Multibus to another. Reconfiguration is automatic when the system is rebooted.

Real-time Support

To provide support for real-time applications, the Concentrix scheduler supports the assignment of dedicated computing resources to identified user processes. Concentrix has the ability to "lock" a process into a single computing resource (computational complex, detached CE, or IP). These processes can use shared memory for interprocess communication, avoiding the use of Concentrix kernel services and assuring deterministic response.

Figure 4-8 illustrates a real-time system that uses all three types of FX/8 computing resources. The shared memory environment allows fast data passing between processors and avoids context switching and operating system kernel services, essentially implementing a zero-latency real-time system.

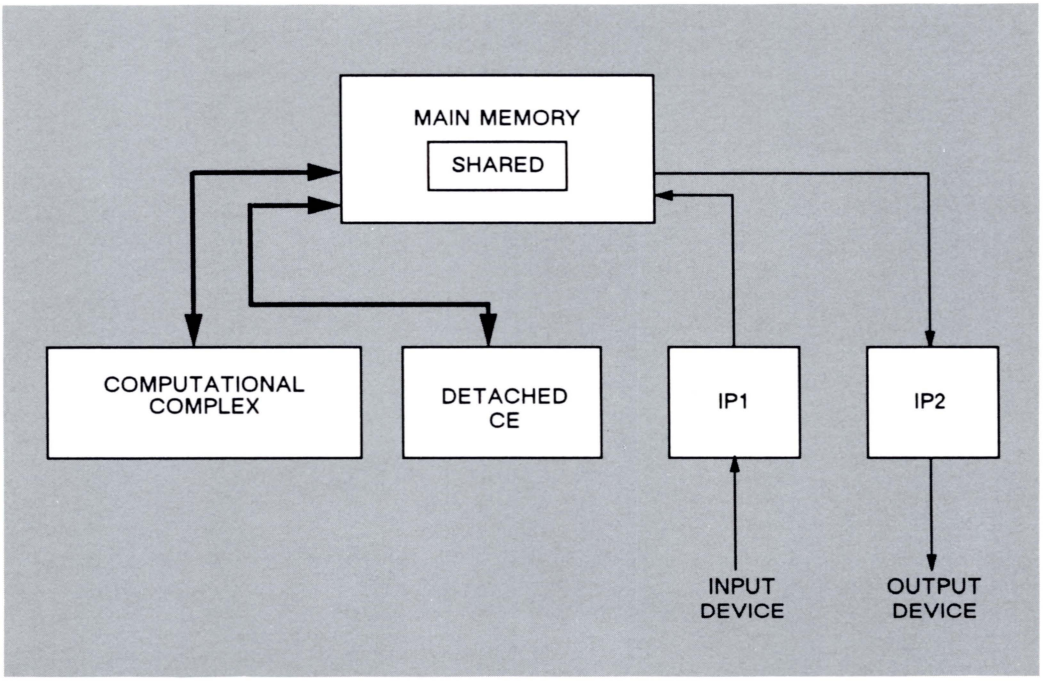


Figure 4-8: Real-time processing on an FX/8

Concentrix Shell

The shell is a command language interpreter that is also a powerful programming language. Typically the shell executes commands by creating processes. Shell commands can be grouped into a program called a shell script. Utilities and capabilities within the shell include control-flow primitives, parameter passing, variable and string substitution, and programming constructs such as "if-then-else," "case," "for," and "while" that provide great flexibility in building applications. Pipes can be used to connect the output of one program to the input of another program. Many different filters are available to modify an input stream, and a user can easily redirect standard input or output to other devices or files.

Concentrix implements the C shell of 4.2BSD UNIX as well as the AT&T Bourne Shell.

Concentrix Languages and Utilities

Concentrix supports several high level languages, an assembly language, a wide range of programming tools and text editors, and special system utilities such as the system monitor.

Programming Tools

Concentrix includes a group of utilities that provide a highly organized and efficient programming environment. The Source Code Control System (SCCS) helps control and account for changes to text files (typically source code and documentation). SCCS maintains an audit trail that includes time and date recording of all changes to a given data base by authorized users.

Make is a program that automates many of the activities of program development and maintenance. Using stored information on inter-file dependencies and command sequences, make updates program files by performing only the necessary operations.

To facilitate program development, Concentrix supports high-level language and assembly language debuggers. The high-level debugger, dbx, enables program developers easily to explore, display, and modify variables within a running program. As the programmer steps through the program, dbx displays the source code, making it easier to monitor execution and isolate bugs. The assembly language debugger, adb, allows users to examine stack traces, view the contents of registers, and set breakpoints at individual assembly language instructions.

Other Concentrix utilities include document preparation utilities (nroff, troff, sed, and lex), a parser generator, calculator functions, high-level language macro facilities, and pattern manipulation facilities.

System Monitor

The Concentrix system monitor, mon, monitors the utilization of system resources and displays status information in real time. Mon has three modes of operation:

- In family mode, mon monitors a target process and all of its descendent processes.
- In system mode, mon monitors all processes in the system.
- In computing resource mode, mon displays a bar graph showing the state of each installed IP, detached CE, and the computational complex.

Text Editors

Concentrix provides two full-screen text editors, vi and CCA EMACS™. Vi is the display oriented interactive screen text editor that is supplied with the 4.2BSD version of UNIX.

The major features of CCA EMACS include:

- multiple windows and buffers, which permit several files to be displayed and edited simultaneously.
- direct UNIX command execution from a window.
- advanced editing tools for formatting, abbreviation expansion, case changes, transpositions, and other operations.
- a keystroke macro command facility.
- an extraction mechanism for creating tag files.
- an interactive spelling checker.
- source compare via multiple windows.
- automatic backup and undo facilities.
- an on-line help facility.

Languages

Concentrix supports Fortran, Ada, C, Pascal, and the Alliant assembler. (Fortran and Ada are discussed in Chapter 6 and Chapter 7.)

FX/C is based on the Bell Laboratories Portable C compiler and has been extended to be ANSI X3J11 compatible. It supports the 32-bit CE scalar instruction set and generates code that makes direct use of the CE floating point hardware. The global optimizer includes advanced optimization features such as:

- constant computation and propagation
- redundant expression elimination
- global register allocation for frequently used variables
- dead store elimination
- induction variable elimination
- reduction of multiplication to addition where speedup will occur
- invariant code motion for DO loops and loops formed from GOTOs
- instruction scheduling to maximize pipeline performance
- label chaining for multiple labels at one location and for branches to branches
- branch removal where the branch is to the next instruction
- displacement minimization
- code sequence optimization

The Pascal compiler is based on the Berkeley UNIX 4.2 compiler for the DEC VAX. It is an implementation of the Pascal language described in the Jensen-Wirth *Pascal Report*. It supports the 32-bit CE scalar instruction set and Alliant floating point instructions.

FX/Fortran, FX/C, and Pascal allow the programmer to add explicit concurrency to applications by invoking routines to perform general forms of synchronization and memory sharing. Access to CE vector capabilities is accomplished with calls to a li-

brary of vector operations that have been tuned to take maximum advantage of both vectors and concurrency.

All languages share a common calling convention. Routines can be written in any language and call each other directly.

The Alliant assembler supports the 32-bit scalar instruction set and Alliant instructions for floating point, vectors, and concurrency.

Run-time profiling (see Chapter 6) and call-trace analysis are easily invoked for high-level languages. They quickly highlight the specific routines and loops where time is spent in running the program with actual data. Profiling details time spent at the routine or source-code statement level. Program developers can therefore focus optimization efforts on the code having the greatest potential for improved performance.

Documentation

The Concentrix documentation set is organized into small (7 X 9 inch) manuals that treat subsets of the three major topics: basic system use, software development, and system administration. The manuals are organized logically and functionally so that users can find necessary information quickly.

Additional manuals are available for the FX/Series architecture and each of the programming languages.

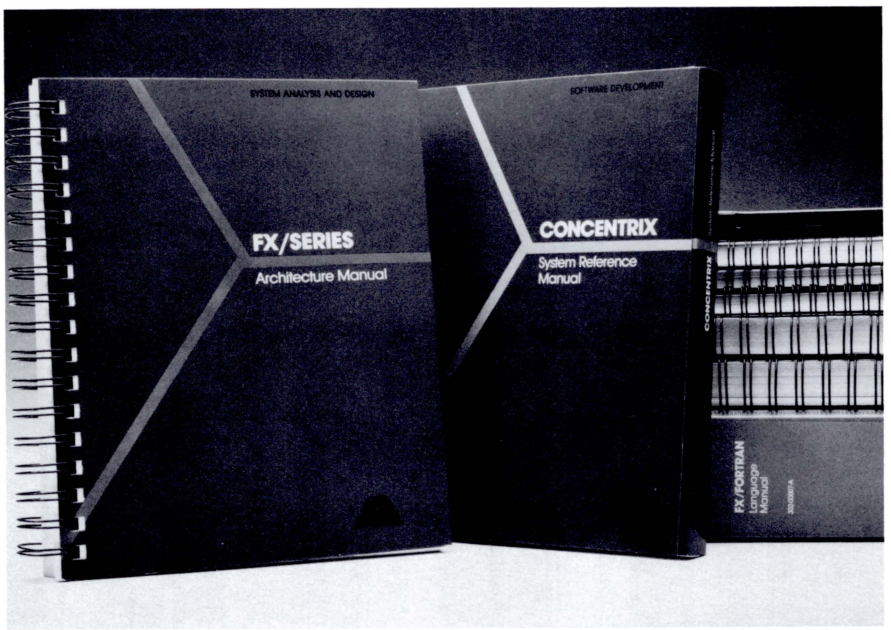


Figure 4-9: FX/Series documentation

REFERENCES

[1] Multitasking User Guide (SN0222), Cray Research Inc., Mendota Heights, Minnesota.

[2] Computerworld, December 22, 1986.

CHAPTER 5

ANSR – Alliant Network Supercomputing Resources

The scientific and engineering computing environment is evolving towards a three-tiered structure consisting of desktop systems such as PCs and workstations, high-performance interactive minisupercomputers such as the Alliant FX/Series, and traditional supercomputers such as Cray systems.

Planning for this environment raises several important issues:

- Disk storage – How do I provide large amounts of disk storage for my PCs and workstations?
- Computational services – How do I combine interactive workstation graphics with the computational power of the minisupercomputer?
- System integration – How do I integrate these systems into existing sites consisting of VAX/VMS™, Cray, and IBM systems?

ANSR™ is a set of industry-standard products that enable the *interactive* coupling of PCs and workstations with FX/Series systems, while allowing integration into traditional environments.

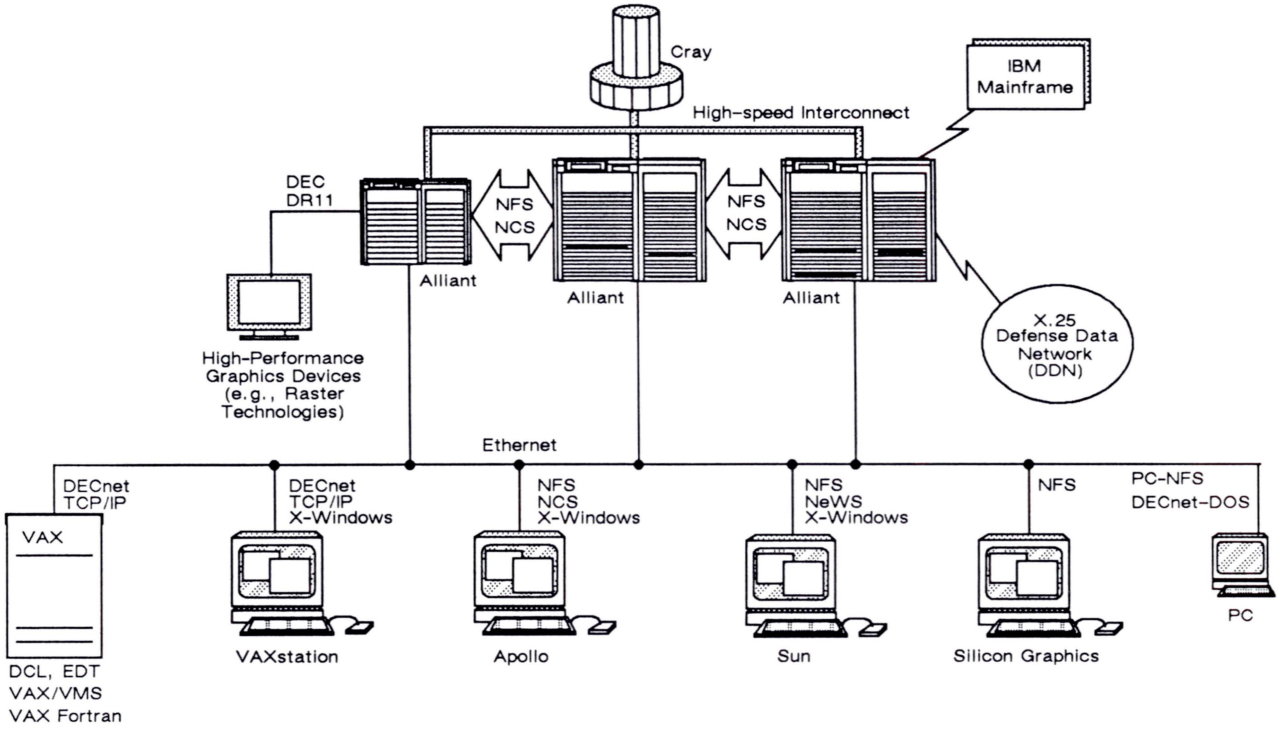


Figure 5-1: Alliant's network supercomputing environment

ANSR products for *interactive network supercomputing* include the Network File System (NFS™); the Network Computing System® (NCS®); and the X-Window System™ and NeWS™ for network graphics.

ANSR *interconnect products* include DECnet™-compatible networking (with Digital Command Language and EDT editor emulation), support for Hyperchannel™ high-speed networking, the HASP protocol, and X.25/DDN (Defense Data Network) wide-area networks.

The Alliant FX/Series architecture is well-suited to providing compute and I/O-intensive services in this network supercomputing environment. Multiple CEs can handle the demands of multiple compute-intensive jobs from the network; multiple IPs can handle the operating system, file I/O, graphics, and network operations.

Interactive Data Sharing:

Network File System (NFS)

NFS allows Alliant systems to be used as high-performance compute and file servers in multi-vendor heterogeneous networks. Large file systems and back-up facilities resident on the Alliant system become a network resource accessible to all users.

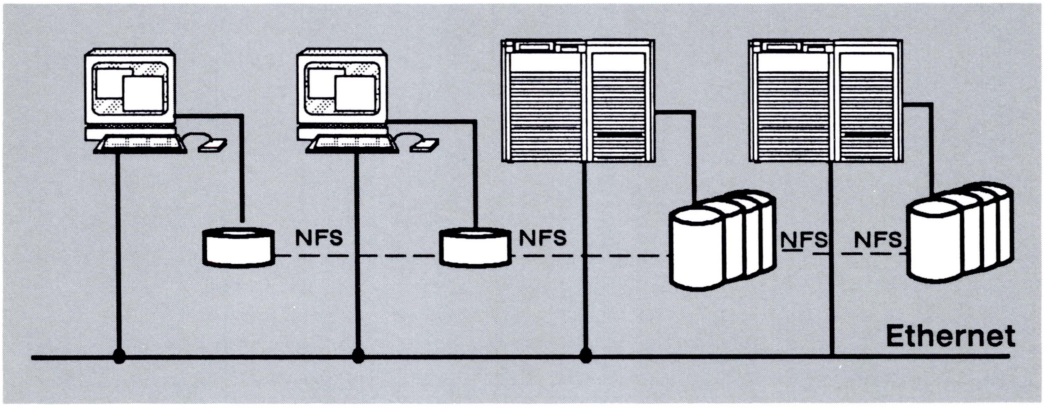


Figure 5-2: The Network File System (NFS) allows large disk farms on Alliant systems to appear local to workstation users.

With NFS, all shared files – no matter where they are located – appear to users as a part of a single file system that can be transparently accessed from across the network.

In typical user environments, shared file systems are physically mounted on Alliant disks and logically mounted on workstations using NFS. Local programs running on the workstation (performing interactive pre-processing, for example) access these files in the same way that remote programs executing on the Alliant compute server (such as analysis programs) access them.

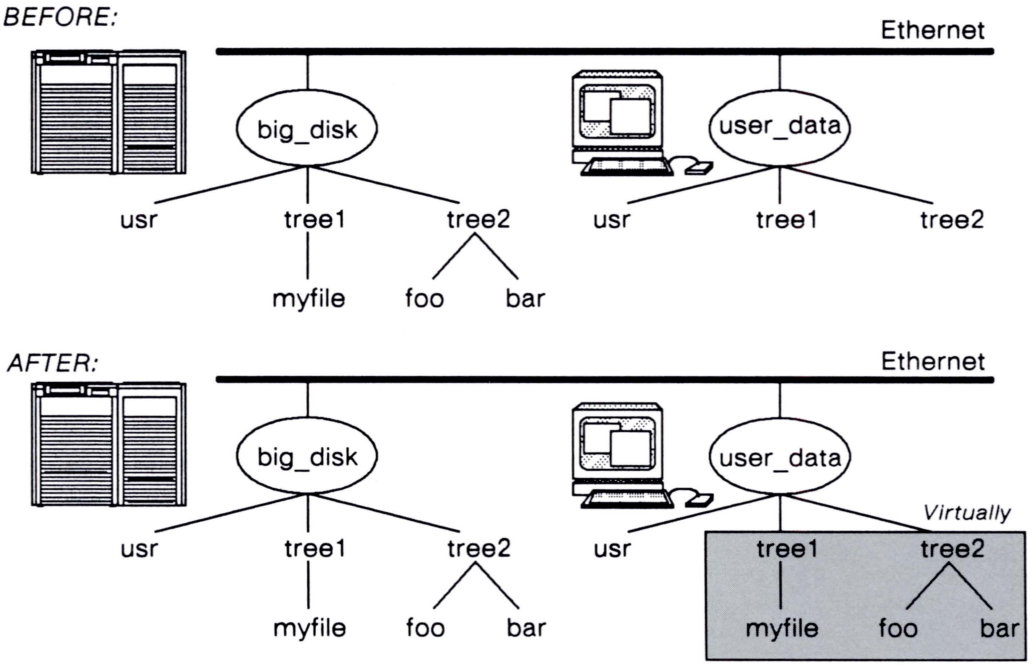


Figure 5-3: NFS provides transparent access to remote files over the network.

NFS provides:

- Ease-of-use and increased productivity. All files appear local to users, eliminating the need for cumbersome, non-interactive file transfer operations.
- Substantial disk storage savings and enhanced data security. Only one copy of data needs to be maintained on the network. Data can be stored centrally on large, expandable Alliant disk farms, and can be backed up on a regular basis by the system administrator.
- Consistent network-wide files. Since multiple copies of data do not exist, the risk that old or corrupted data may inadvertently be used is reduced.

Interactive Compute Sharing:

Network Computing System (NCS)

NCS is a set of tools that allow users to develop applications that use computing resources throughout a network.

With NCS, applications can be decomposed into cooperating subroutines that execute on appropriate resources. Compute- and I/O-intensive subroutines are executed on Alliant minisupercomputers, rather than on workstations, producing faster response times and more effective utilization of the organization's computing resources.

NCS allows *network compute sharing* to occur in conjunction with the *network data sharing* provided by NFS.

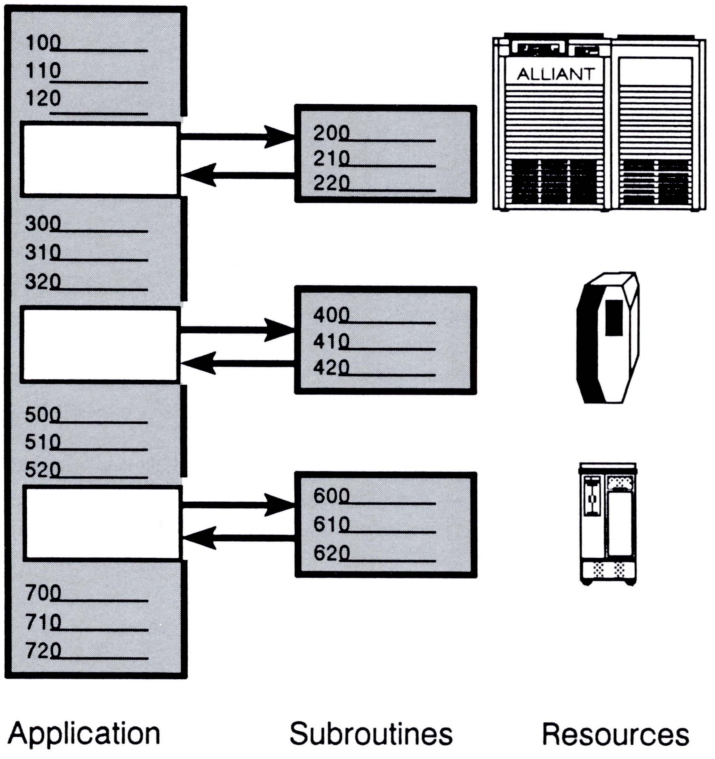


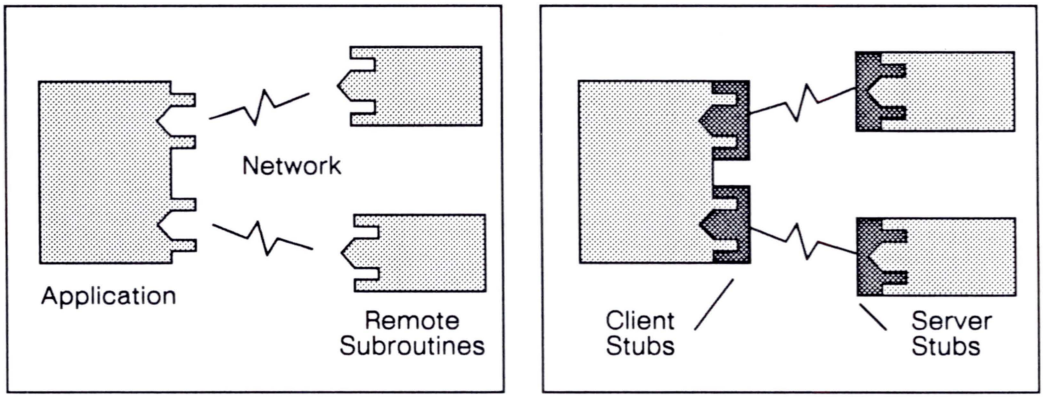
Figure 5-4: The Network Computing System (NCS) provides tools to decompose applications into subroutines that execute on different compute resources across the network.

NCS, an open system designed for multi-vendor environments is based on three major components:

- A remote procedure call (RPC) facility that allows programs running locally to call procedures implemented on remote hosts.

- A compiler, known as the *Network Interface Definition Language (NIDL),* that converts high-level subroutine interface definitions into portable C-language source code. This code is then compiled into a set of *stubs* that look like the local subroutines to the calling program, and look like the calling program to the remote procedure.

- A set of software tools that let applications determine at runtime which computers can provide the services they require. For example, NCS allows Alliant compute servers offering specific services – such as execution of a particular compute-intensive procedure – to register their capabilities with the *location broker;* clients then query the location broker at runtime to determine which hosts to use for particular RPC calls to that procedure.



In network-based applications, individual procedures can run on computers throughout the network.

Stub routines let local and remote parts of the application communicate as one application. NCS provides a compiler that generates client and server stubs automatically.

Figure 5-5: Remote procedure calls in the NCS environment

Interactive Network Graphics:

The X-Window System and NeWS

These packages enable applications executing on Alliant systems to generate graphics on workstations over the network. In this particular approach to network computing, the application is partitioned to execute on the Alliant system, except for graphics and windowing operations, which are executed remotely on the workstation.

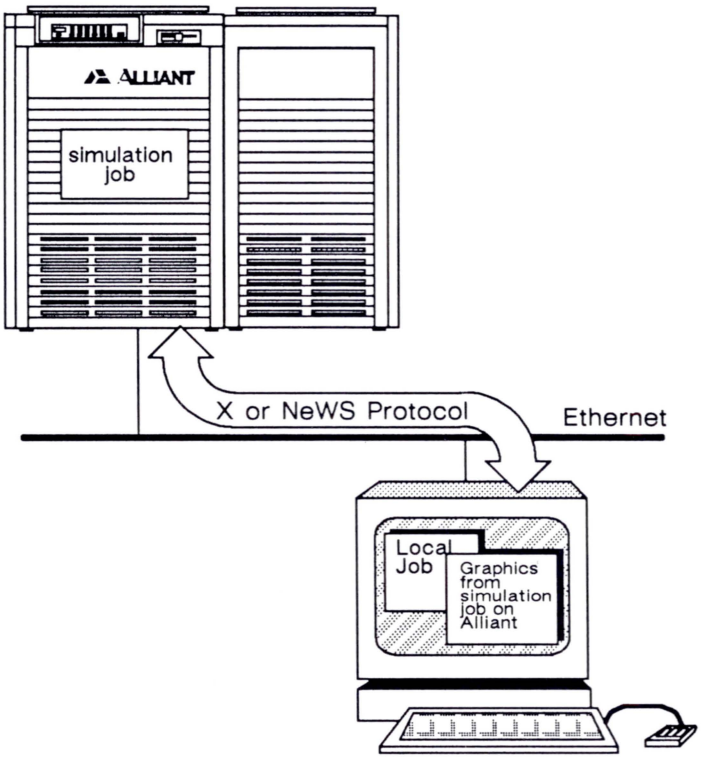


Figure 5-6: The X-Window System and NeWS provide graphics capabilities to applications residing on Alliant minisupercomputers.

This approach minimizes the amount of network-related development required by the application developer. The application simply makes calls to a local graphics and windowing library (for example, "open_window"). The library handles all interprocess communication to software running on the remote workstation that executes the request. The workstation software, known as the background server process, is supplied by most workstation vendors.

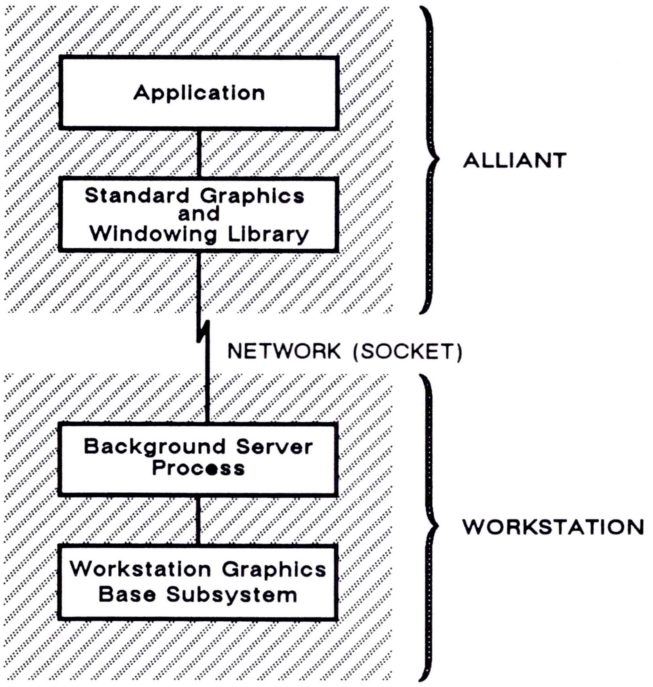


Figure 5-7: Applications make calls to a standard graphics and windowing library with the X-Window System and NeWS. The network socket mechanism is used to transparently transfer these calls to a background server process running on the workstation.

NeWS goes one step further by allowing applications running on the Alliant system to load code into the workstation. The code, written in the PostScript graphics language, executes locally, providing high-performance for repetitive or highly interactive graphics operations.

The X-Window System and NeWS allow supercomputing applications running on Alliant systems to take advantage of the ease-of-use features commonly found on PC or workstation applications, such as windowing, pop-up menus, bit-map graphics, and icons. Workstation users linked to larger systems are not limited to text-only windows when communicating with remote applications.

ANSR Interconnect Products

VAX Compatibility:

Fortran, DCL-Like User Interface, EDT Editor, DECnet-Compatible Networking

Alliant's VAX compatibility software allows the easy integration of Alliant systems into existing VAX/VMS environments. Users can retain their investments in VAX hard-

ware, software, and user training, while taking full advantage of the parallel vector processing capabilities offered by the FX/Series.

Alliant's FX/Fortran compiler is VAX compatible, allowing VAX applications to port quickly to Alliant systems with little or no recoding required.

The Digital Command Language (DCL™)-like user interface provides VMS users with most of the commands and on-line HELP found on VMS systems. An optional "teach" mode aids in migration by echoing equivalent UNIX commands when DCL commands are typed. The EDT emulator provides full EDT functionality, including keypad functions, user defined keys, journaling, and context-sensitive help.

Alliant's DECnet-compatible software enables Alliant systems to function as transparent DECnet end-nodes in a Phase IV Ethernet™ environment. Users can perform common DECnet functions such as remote login ("set host"), file transfer, remote directory access, and task-to-task communications. Digital's ncp network management functions are also provided.

These products allow users to be productive without the immediate need to learn new commands and editors, rewrite programs, or learn new networking functions.

High-Speed Networking:
Hyperchannel

The Hyperchannel is a 50 Mbps link commonly used in supercomputer environments.

Alliant systems are well-suited as "bridges" between desktop systems and Cray back-end supercomputers. The FX/Series provides the interactive services that desktop systems require, as well as the capability to receive large amounts of data from the Cray and redistribute it to individual desktop systems. Alliant's vector and parallel processing architecture is also well suited as an adjunct development environment for Cray applications.

Alliant's Hyperchannel support is based on the NETEX protocol and BFX (Bulk File Transfer) program developed by Network Systems Corporation.

Wide Area Supercomputing:
X.25 and Defense Data Network (DDN)

Alliant's support for X.25 networks such as Telenet, and DDN networks such as MILnet and ARPAnet, extend network supercomputing to remote users connected over packet-switched data networks (PSDNs). Alliant's wide-area networking support conforms to the CCITT 1980 recommendations for X.25 and X.29.

Alliant's DDN implementation combines standard TCP/IP facilities with X.25, allowing local area networks in different geographic locations to be connected together, and appear to users as if they were a single logical network. It also provides users with familiar DoD utilities such as FTP, Telnet, and SMTP over X.25 networks.

IBM Support:
HASP

Alliant's HASP product emulates an IBM 360/20 multileaving remote job entry workstation. HASP enables FX/Series systems to send jobs and data files to a HASP central system (such as IBM, CDC, Honeywell, Sperry, Cray) for execution or file transfer. Data received from the central system is passed to a user-tailorable shell program for routing or other post-processing. The HASP software supports speeds up to 64Kbps on local or remote dedicated lines.

Network User Applications

TCP/IP and its associated DoD utilities, operating over Ethernet, have become de facto standards for multi-vendor networking. These facilities are offered, along with basic BSD 4.2 networking functions, as part of the standard Concentrix distribution, providing capabilities for file transfer, remote login, and electronic mail. TCP/IP and DECnet can coexist on the same Alliant system and same Ethernet, providing simultaneous access to both types of environments.

Alliant Networking and the ISO Model

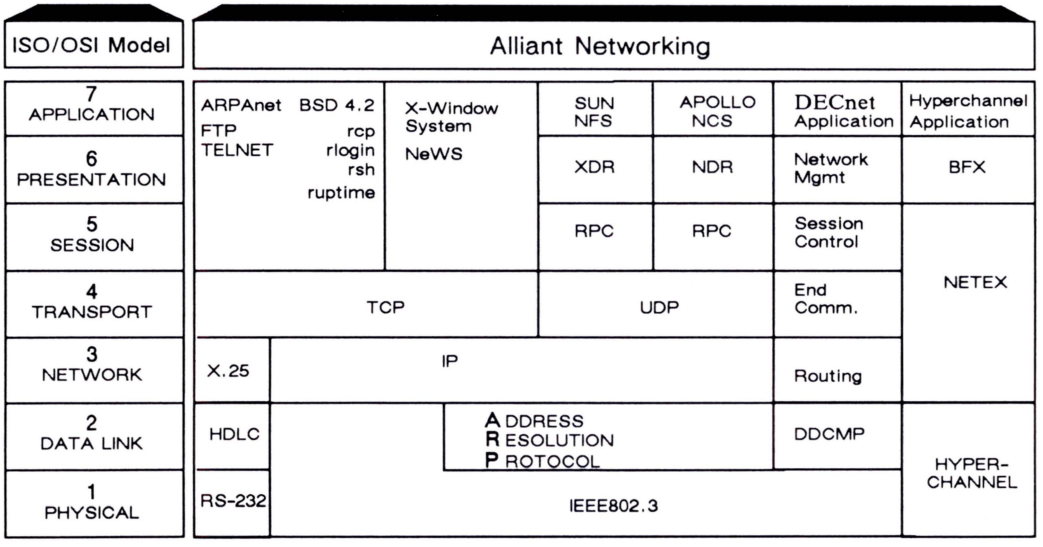


Figure 5-8: Alliant networking and the ISO model

CHAPTER 6

FX/Fortran Compiler

The FX/Fortran compiler is a highly efficient optimizing compiler designed to run on Alliant FX/Series computers. FX/Fortran automatically detects the potential for vector and parallel processing in standard Fortran code and generates instructions that use the concurrency and vectorization features of the hardware to full advantage.

At compile time, FX/Fortran permits the programmer to control the optimization process at the program (global), subprogram, and loop levels, and provides feedback on the optimization process.

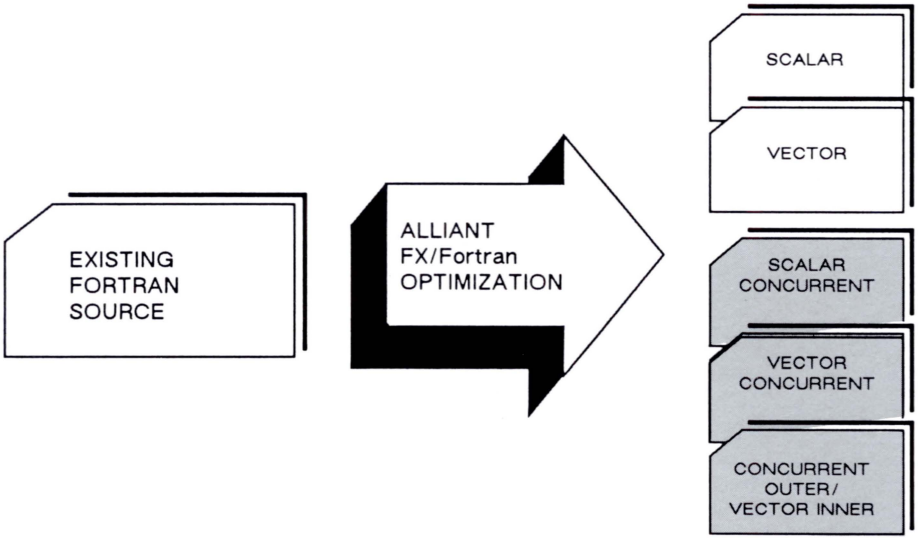


Figure 6-1: Automatic employment of parallelism by FX/Fortran

FX/Fortran is an ANSI standard Fortran-77 compiler that contains most of the extensions of VAX/VMS Fortran-77 and other industry-standard compilers. In addition, FX/Fortran extends the language to permit assignment and other operations on full arrays and the use of full arrays in intrinsic functions. These extensions make source code simpler and more structured, and clarify the intent of the programmer to the compiler. The extensions closely follow the proposed updates to the Fortran standard contained in Standing Document 8, Version 104, of the ANSI X3J3 Committee (published May, 1987), also known as the Fortran-8x committee.

FX/Fortran Code Generation

As shown in Figure 6-1, FX/Fortran is capable of generating code that executes in five different modes.

A scalar is a single element or value, such as S, A(I), or 1.0. Scalar processing uses instructions that operate on single elements of data at any one time. Most superminicomputers have scalar-only architectures. Each CE has a full scalar instruction set that operates on both 32- and 64-bit data elements.

A vector is a series of elements, such as all the elements of the array A, or every fourth element of the array B. Vector processing, first introduced on supercomputers, uses instructions that operate on an entire vector. Each CE supports a full 64-bit vector processing instruction set.

The Alliant proprietary modes of optimization are scalar concurrent, vector concurrent, and concurrent-outer-vector-inner. These modes are generated by FX/Fortran to take advantage of the low-overhead hardware-controlled parallel processing available on the computational complex.

In scalar-concurrent mode, each of up to eight CEs in the computational complex continuously executes the next available iteration of a loop until the entire loop has been executed. In some cases, hardware synchronization mechanisms are automatically employed to synchronize dependencies between iterations. As a result, loops that would be forced to run in serial mode on traditional vector computers often can be optimized for scalar-concurrent execution on the FX/8.

In vector-concurrent mode, the iterations of a loop are divided over the available CEs and each CE operates on its portion of the loop using its vector hardware. For example, a 256-element array is processed on eight CEs by concurrently executing on each CE single 32-element vector instructions.

Where loops are nested, FX/Fortran runs the innermost loop in vector mode and the next outer loop in concurrent mode. Taken as a whole, the two loops run in concurrent-outer-vector-inner (COVI) mode. An array operation within a loop also executes in COVI mode. The array operation executes in vector mode; the loop iterations run concurrently. An array operation on a multidimensional array also executes in COVI mode. The leftmost dimension executes in vector mode; the next dimension executes in concurrent mode. COVI execution results in the highest delivered performance on the FX/8.

FX/Fortran Optimizations

FX/Fortran optimizes programs for concurrency, vectorization, and scalar (global) performance. The compiler analyzes programs for data dependencies and generates optimized executable code that can run on the computational complex or a detached CE.

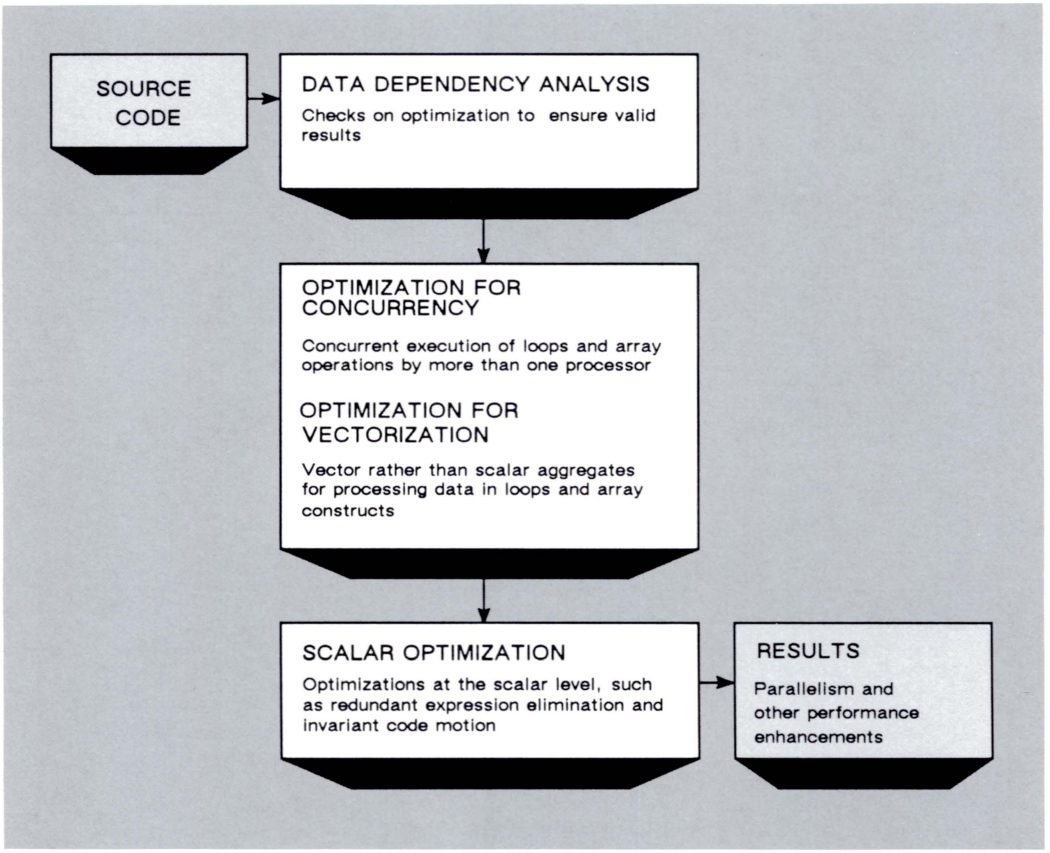


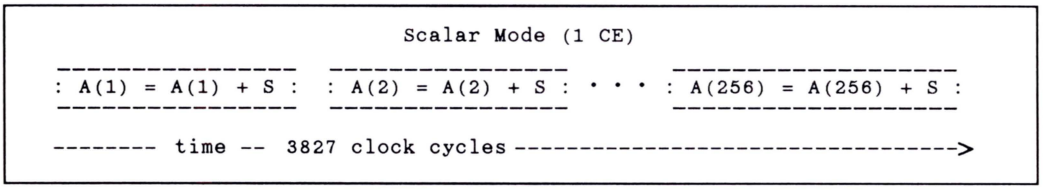
Figure 6-2: FX/Fortran optimization process

The FX/Fortran compiler optimizes the following operations for vectorization and concurrency:

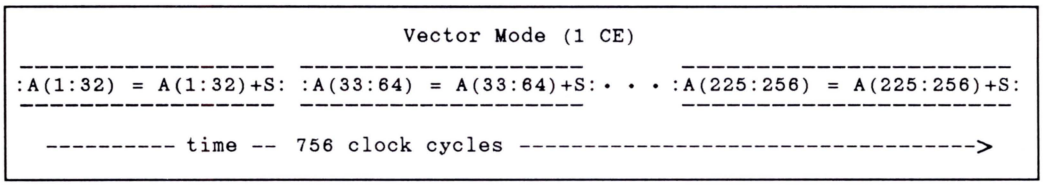
- DO loops execute in vector-concurrent mode. Do loops that contain restricted statements or data dependencies may execute in vector mode, scalar-concurrent mode, or scalar mode depending on the nature of the restrictions or dependencies. Array operations execute in vector-concurrent mode except that concurrency may be limited due to restrictions or data dependencies.
- Nested DO loops and multidimensional array operations run in vector-concurrent or concurrent-outer-vector-inner mode.
- DO WHILE loops run in scalar-concurrent mode.

Optimization Examples

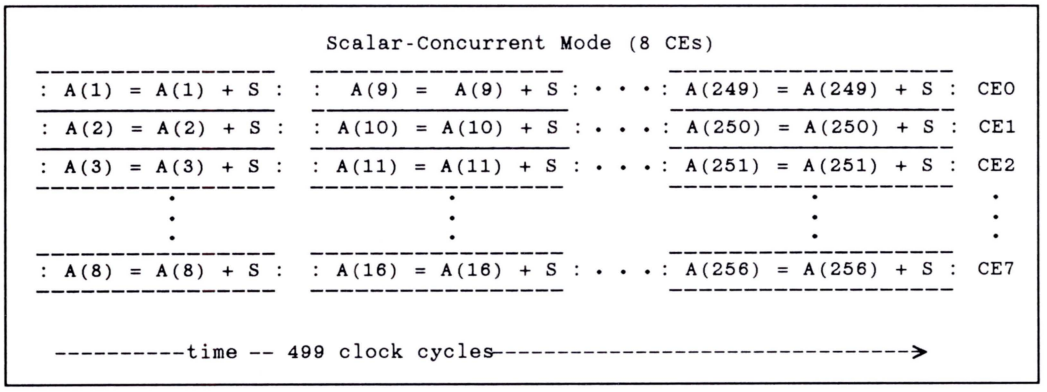
Figure 6-3 shows how FX/Fortran generates code to execute a simple DO loop in scalar, vector, scalar-concurrent, and vector-concurrent modes.



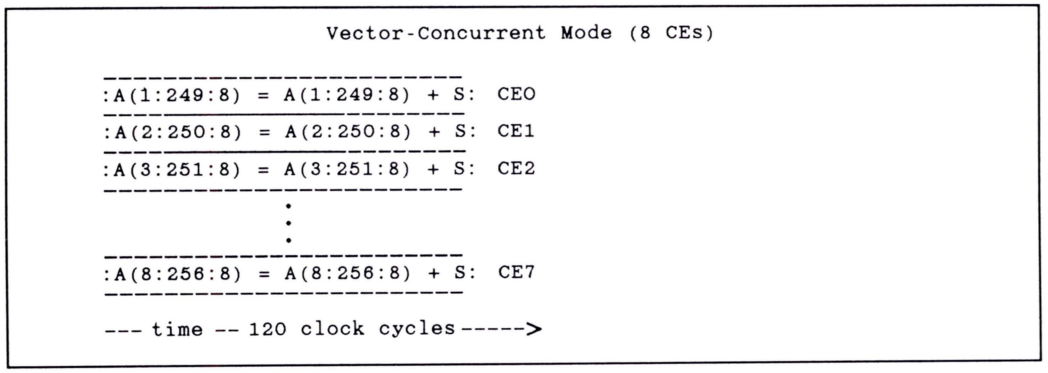
With vectorization and concurrency optimizations turned off, FX/Fortran generates code that executes the loop in scalar (or serial) mode on a single CE with each iteration executed after the previous one completes.



With concurrency turned off, FX/Fortran generates code that executes the loop in vector mode with eight vector instructions executed serially on a single CE. Each vector instruction works on 32 iterations of the loop.



With vectorization turned off, FX/Fortran generates code that executes the loop in scalar-concurrent mode with each iteration of the loop being executed on a separate CE.



With full optimization, FX/Fortran generates code that executes the loop in vector-concurrent mode making direct use of both vector and parallel processing.

Figure 6-3: FX/Fortran can optimize an ordinary DO loop for four different modes of execution.

Figure 6-4 shows the delivered performance of the FX/8 when running a loop in scalar, scalar-concurrent, and vector-concurrent modes. This loop is similar to the DAXPY subroutine from the Basic Linear Algebra Subroutines (BLAS). This operation constitutes the inner loop of many linear algebra algorithms, and is also the most time consuming subroutine in many commercial structural analysis packages.

```
                DO 100  I=1,N
100               Z(I) = Y(I) + X(I)*A
```

Vector-Concurrent to Scalar Ratio	Loop Iterations	Delivered 64-bit MFLOPS for 8 CEs 0----------4.0---------8.0--------12.0--------16.0
0.3	1	|V | | | | | | | |
0.4	2	|V | | | | | | | |
0.8	4	|VC | | | | | | | |
1.5	8	|sVC | | | | | | | |
1.8	12	| sVC | | | | | | | |
2.4	16	| s VC | | | | | | |
4.6	32	| s |CV | | | | | | |
6.4	48	| s | C V | | | | | |
8.0	64	| s |C | V | | | | | |
9.4	80	| s | C | V | | | | |
10.7	96	| s | C | | V | | | | |
11.4	112	| s | C | | V | | | | |
14.8	128	|s | |C | V | | | |
12.3	144	| s | |C | V | | | |
14.5	160	| s | C | | | V | | | |
14.5	176	| s | |C | | V | | | |
17.5	192	|s | |C | | V | | | |
16.0	208	| s | C | | | | V | | |
16.0	224	| s | C | | | | V | | |
17.8	240	| s | C | | | | | V | |
17.8	256	| s | |C | | | | V | | |

Long vector lengths: s = Scalar, C = Scalar-Concurrent, V = Vector-Concurrent

Figure 6-4: Delivered performance varies depending on the mode of optimization.

Figure 6-5 is an example showing nested loops in which the inner loop executes in vector mode and the outer loop is executed concurrently. Consider the following:

```
          DO 10 J = 1,8
             DO   10 I = 1,256
10             A (I,J) = A(I,J) + S
```

which also can be written in FX/Fortran as:

```
A(1:256,1:8) = A(1:256,1:8) + S
```

FX/Fortran automatically compiles either of the above to execute in concurrent-outer-vector-inner mode.

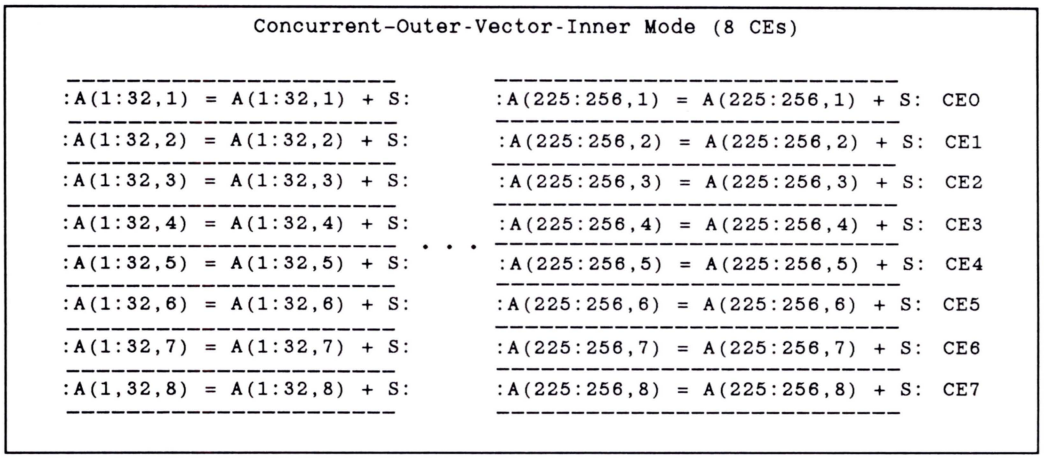


Figure 6-5: FX/Fortran optimizes nested loops for concurrent-outer-vector-inner (COVI) execution.

Figure 6-6 shows that the FX/8 achieves a high level of delivered performance when running nested loops in concurrent-outer-vector-inner mode, even when the number of loop iterations is small.

```
      DO 100  J=1,N
        DO 100  I=1,N
100       Z(J) = Z(J) + X(I)*Y(J) + W(I)*A
```

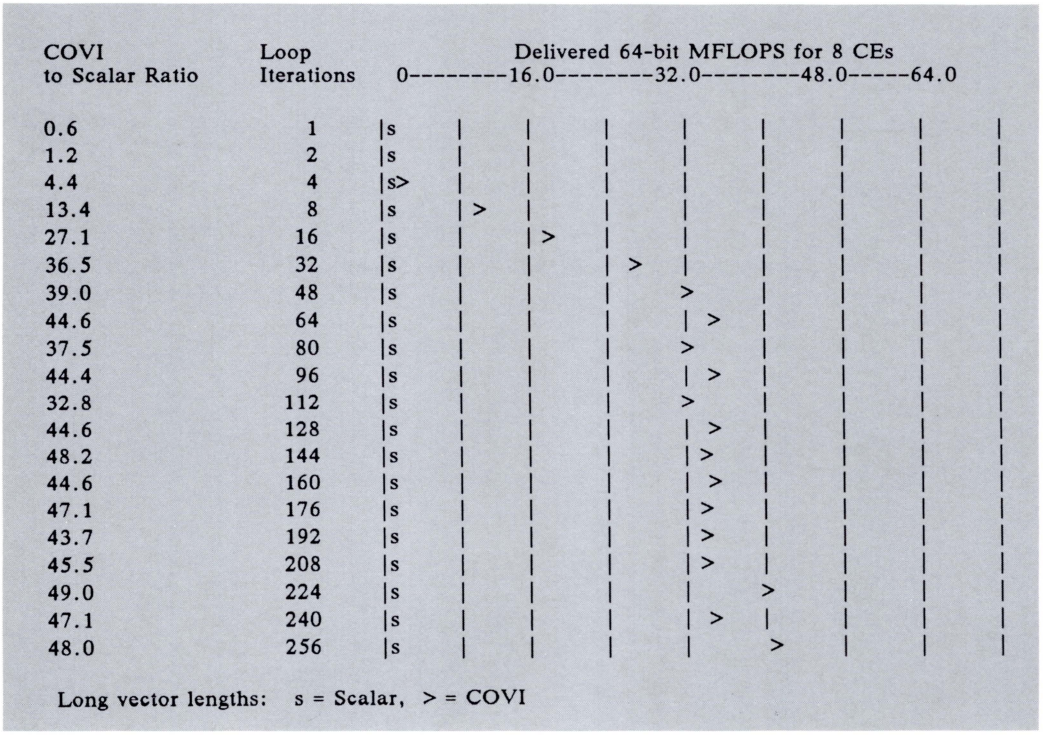


Figure 6-6: Delivered performance for a nested loop

Do Loop Allowable Statements

The inclusion of references to statement functions and intrinsic functions in loops does not prevent optimization. The inclusion of references to external procedures in loops prevents the compiler from using concurrency, unless the inclusion is explicitly allowed by a directive or FX/Fortran command line option.

Table 6-1: Fortran statements allowable in loops optimized for vector and concurrent processing by the FX/Fortran compiler

Statements That Do Not Inhibit Vector or Concurrent Execution	Statements That Inhibit Vectorization But Do Not Inhibit Concurrent Execution
Data Assignment	ELSE IF
COMMENT	Nested Block IF with Nesting to a Level Greater than 3
CONTINUE	
GOTO Forward Label Within Loop	GOTO Label Outside Loop
Arithmetic IF to Forward Label Within Loop	RETURN
Block IF, ELSE IF, END IF	STOP
Logical IF	

Array Extensions for Optimization

Standard Fortran-77 compilers require that arrays be manipulated in a series of scalar operations. To move more than one element of array B to array A, a program must perform a series of scalar assignment operations.

```
      DO 12 I = 1,N
12       A(I) = B(I)
```

The FX/Fortran compiler permits operations on full or partial arrays of more than one element, so that the example above can be written:

```
A(1:N) = B(1:N)
```

FX/Fortran extensions permit operations on array sections with indexed (shown in the example) and vector-valued section selectors. Array expressions can be used in assignment statements and as primaries in expressions.

FX/Fortran extends all Fortran-77 intrinsic elemental functions to array operations. For example, the following statement calculates the square root of N elements in array A:

```
A(1:N) = SQRT (A (1:N))
```

In addition, the extensions include more than 25 array transformational and inquiry functions. The following example sums N elements of array A:

```
S = SUM (A (1:N))
```

General Fortran Optimization

In addition to vectorization and concurrency, the FX/Fortran compiler contains a broad range of global optimization capabilities that are automatically performed on source code when the global optimization option is enabled:

- Constant computation and propagation
- Redundant expression elimination
- Global register allocation for frequently used variables
- Dead store elimination
- Induction variable elimination
- Reduction of multiplication to addition where speedup will occur
- Invariant code motion for DO loops and loops formed from GOTOs
- Instruction scheduling to maximize pipeline performance
- Label chaining for multiple labels at one location and for branches to branches
- Branch removal where the branch is to the next instruction
- Displacement minimization
- Code sequence optimization

Language Extensions

The FX/Fortran compiler contains extensions to Fortran-77 to increase the utility of the language and to provide compatibility with other implementations of Fortran.

The extensions include:

- Source program formatting with tab formatting conventions
- DATA statements in executable code
- INCLUDE statement
- Many data type extensions
- Bit intrinsic functions
- Hexadecimal, octal, and Hollerith constants
- Control logic extensions such as DO WHILE and END DO statements
- Extended range on DO loops
- Input/output extensions such as ACCEPT and TYPE statements, namelist-directed I/O, additional format descriptors, field descriptor defaults, short field termination, and additional file attributes

Performance Tuning

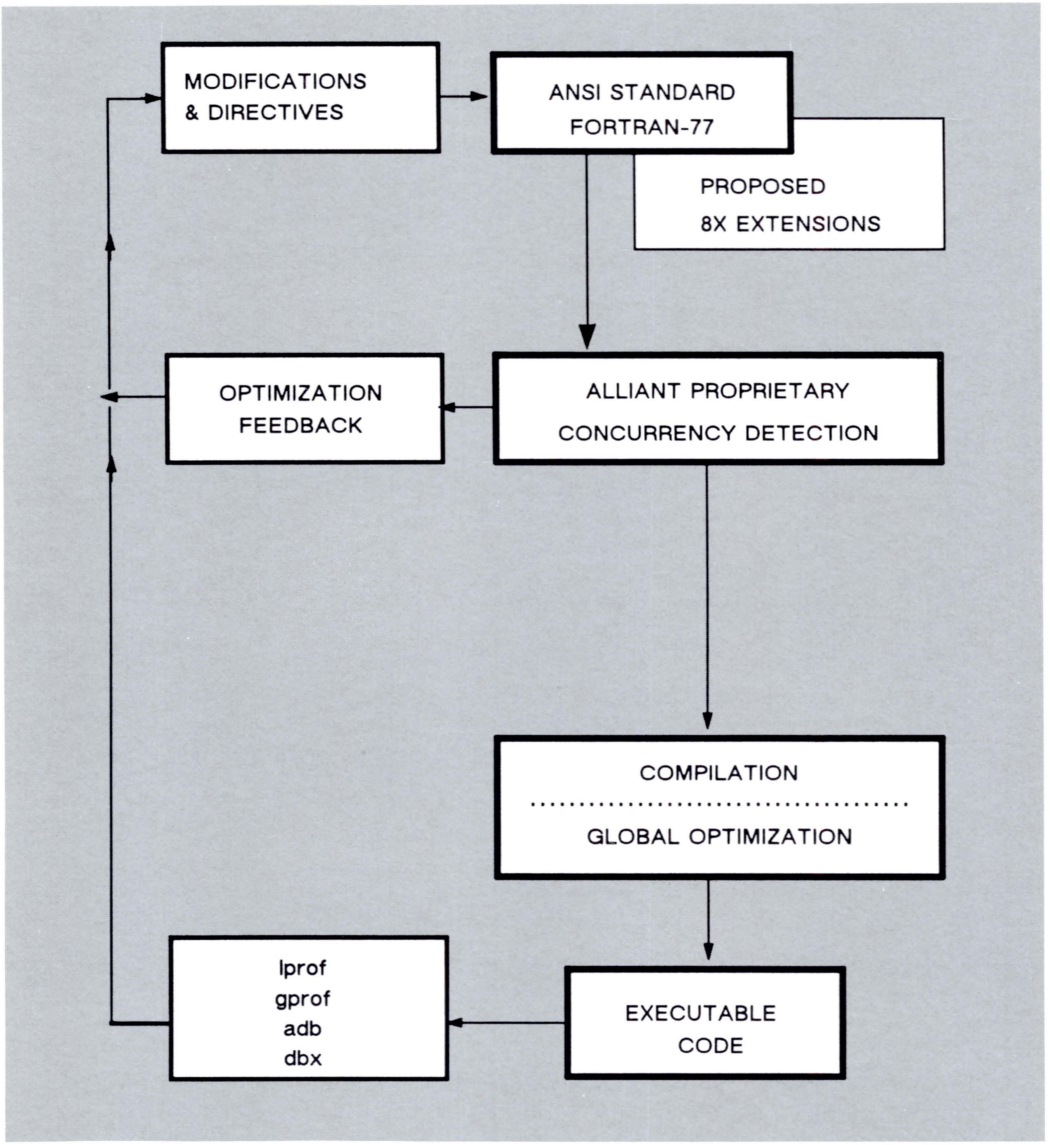


Figure 6-7: FX/Fortran provides extensive performance tuning tools.

The FX/Fortran development environment provides the programmer with extensive optimization feedback in the form of messages and listings, compiler directives, and profilers at both the loop and line level (Figure 6-7).

Messages and Listings for Parallel Optimization

The FX/Fortran compiler provides facilities for users to monitor and direct the optimization process. Messages notify the user of conditions that affect optimization, and summarize the degree of optimization attained for each routine. Optimization can then be modified by the programmer with compiler directives.

```
    1                Subroutine Demo
    2                common f(99),ft(99),n,r,eps
    3
    4                do 24 i=1,6
    5                   x = ft(i) * float(n)
    6                   x2 = 2. * x
    7                   x21 = x2 - 1.
    8                   df = x21 / x2
    9                   df1 = df * r
   10                   df2 = df1 * df1
   11                   ff = f(i) + df2
Line 11       Informational message # 1237 dependency to line #12
                     concurrent and vector loop optimization inhibited by
                     feedback to array elements -- F
   12                   f (i+1) = ff
   13                   af = abs(df2/f(i+1))
   14                   if (af .1e. eps) goto 25
   15           24      continue
   16           25      continue
   17                   return
   18                   end

            ----------DO-LOOP SUMMARY FOR ROUTINE DEMO---------
     LABEL   INDEX   START   END  NEST   COMMENT       CD? DP?  ITERATIONS

        24     I          4   15     1  CONCURRENT          Y          6

       0 VECTOR          1 CONCURRENT      0 VECTOR-CONCR      0 CON. OUTER
       0 NOT OPTMZD      0 NOT INNER       0 TOO SHORT         0 DEPENDENT
       1 TOTAL           1 EXAMINED        1 OPTIMIZED
```

Figure 6-8: FX/Fortran output showing optimization summary

Figure 6-8 demonstrates an FX/Fortran program listing with optimization enabled. The listing consists of two sections. The first section is a line-numbered listing of the original Fortran source, including any error, warning, or informational messages caused by statements. The second section is a loop summary containing information about the DO loops in the program and the optimization or transformations for vector or concurrent execution performed by the compiler.

The first column of the loop summary shows the label of the statement ending the loop and the second the index variable of that loop. The START and END columns list the line numbers of the first and last statements of that loop in the source listing. The NEST column displays the level of nesting of each loop. The COMMENT column describes the type of optimization performed or the reason that no optimization was

performed. The CD? column contains a "Y" if there is a conditional within that loop, and DP? similarly indicates if a data dependency exists in an optimized loop. The last column contains a constant, variable, or expression indicating the number of iterations of the loop (ignoring a possible early exit).

Compiler Directives

The FX/Fortran compiler provides the programmer with twelve optimization directives that can be inserted in source code to control optimization at the global, routine, and loop level. Compiler directives appear as comments so that a program containing directives remains standard and portable.

Optimization directives allow the programmer to override the compiler in situations where optimization can be enhanced based on information not available to the compiler.

Table 6-2: FX/Fortran compiler directives

DIRECTIVE*	DESCRIPTION
ALTCODE (n) ALTCODE (n) CONCURRENT ALTCODE (n) VECTOR **NOALTCODE**	Generate alternate scalar, concurrent, or vector code for short loops.
ASSOC ASSOC(D) **NOASSOC**	Optimize associative transformations.
CNCALL **NOCNCALL**	Allow subroutine and function references in loops.
CONCUR NOCONCUR	Optimize for concurrency.
DEPCHK NODEPCHK	Check for data dependencies between loop iterations.
EQVCHK NOEQVCHK	Check for data dependencies caused by equivalence statements.
PERMUTATION (a,...)	Supress checks for data dependencies caused by indirect addressing.
RELATION (int .r. int)	Supress checks for feedback depending on the possible value of a subscript.
LSTVAL NOLSTVAL	Save last values of original indexes and promoted scalars.
SHORTLOOP NOSHORTLOOP	Generate a single vector indtruction for known short loops.
SYNC NOSYNC	Check for synchronization problems between loop iterations.
VECTOR NOVECTOR	Optimize for vectorization.
* defaults shown in bold	

Figure 6-9 shows an example of conditional code where a "NOVECTOR" directive is added to suppress vectorization. In vector mode, all the square root calculations would be performed, and only those where a(i) was greater than or equal to x would be stored. In this case, the programmer knows that a(i) is rarely greater than or equal to x, so the "NOVECTOR" directive is used to force scalar-concurrent mode and avoid the unnecessary computation of many square roots.

```
 1
 2            Subroutine Loop15
 3            dimension a(10)
 4            data x/0/
 5      cvd$ novector           ! concurrency only
 6            do 15 i=1,10
 7               if(a(i).lt.x) go to 15
 8               a(i)=sqrt(a(i))
 9       15   continue
10            end


------------------  LOOP SUMMARY FOR ROUTINE LOOP15 ------------------
  LABEL    INDEX    START    END NEST   COMMENT       CD? DP?   ITERATIONS

     15    I            6      9  1   CONCURRENT                 10
```

Figure 6-9: Compiler directives are used during the optimization process.

Profiling Tools

The gprof profiler identifies time-consuming subroutines that can benefit from further optimization and provides information about time spent in the execution of these subroutines. Gprof lists the percent of actual program execution time attributed to a routine, the cumulative seconds spent executing all routines, the actual number of seconds spent executing each routine, and the number of calls to each routine.

The line-level profiler, lprof, allows a programmer to generate a profile of a subroutine on a line-by-line basis. Lprof is a very useful tool if a time-consuming subroutine has many loops and branches that make it difficult to tell which loops are executed most often.

```
test.f          test            84              0.01 sec        0.3%
test.f          test            120             0.01 sec        0.3%
test.f          test            121             0.06 sec        2.0%
test.f          test            131             0.01 sec        0.3%
test.f          test            132             0.07 sec        2.3%
test.f          test            133             0.01 sec        0.3%
test.f          test            135             0.01 sec        0.3%
```

Figure 6-10: Output from the line level profiler (lprof)

Loop Transformation Performance

In general, the best performance results from loops that are processed in both concurrent and vector modes. As discussed above, the FX/Fortran compiler fully optimizes both single loops and nested loops. Purely concurrent execution is also available for certain loops that cannot be executed in vector mode. Because Alliant concurrency can optimize loops containing data dependencies and other conditions that inhibit vector execution, loop transformation performance exceeds the performance of compilers limited to vectorization alone, as shown in Table 6-3.

Table 6-3: FX/Fortran identifies and optimizes loops that cannot be optimized by vector computers.

MACHINE/ COMPILER CODE	TOTAL LOOPS	LOOPS TRANSFORMED			
		CRAY X-MP CFT 1.15	FUJ.VP V10L20	NEC-SX vsn24	ALLIANT REV 2.0
BARO	88	61	66	60	83
EULER	92	50	51	50	69
MHD	52	26	28	28	40
SHEAR	71	35	38	36	46
VORTEX	32*	23	25	23	25
All 5 Codes	335	195	208	197	263

Cray, Fujitsu, NEC numbers - R. Mendez, private communication (August 1986)

* no I/O version of Vortex (version with I/O has 34 loops)

FX/Series Scientific Library

The FX/Series scientific library routines were written by Alliant to take optimal advantage of concurrency and vectorization on FX/Series computers. Two examples of the high performance of these routines are shown in Figures 6-11 and 6-12. The routines are callable from FX/Fortran, FX/C, FX/Ada, and Pascal and fall into the following groups:

- **Convolvers** – The convolution of two vectors arises frequently in probability, engineering, and other problems. Convolvers are used in filtering signals for smoothing and noise elimination, where one vector represents the signal and the other vector represents the filter.
- **Fourier transforms** – The Fourier transform is a basic tool in digital signal processing and analysis. Application examples include spectrum estimation, auto-correlation estimation, and discrete convolution.
- **Toeplitz solver** – The resolution of Toeplitz linear systems is a basic step in parametric signal processing for auto-regressive modeling, linear prediction, spectrum estimation, infinite impulse filter design, and other applications.

- **Basic Linear Algebra Subprograms** Level 1 (BLAS1) – The BLAS1 perform certain basic vector operations of numerical linear algebra.
- **Basic Linear Algebra Subprograms** Level 2 (BLAS2) – The BLAS2 extend the BLAS1 to matrix-vector operations in order to provide for efficient and portable implementations of algorithms for high-performance computers.
- **Basic Linear Algebra Subprograms** Level 3 (BLAS3) – The proposed BLAS3 further extend the BLAS1 and BLAS2 to matrix-matrix operations with the aim of providing more efficient and portable implementations of algorithms on high-performance computers, especially those with hierarchical memory and parallel processing capability. The FX/Series Scientific Library incorporates the most important subset of the BLAS3.
- **Various Linear Algebra Routines** – These vector and matrix operations include block-LU factorization, orthogonal factorization, triangular system solvers, and eigenvalue problem solvers.

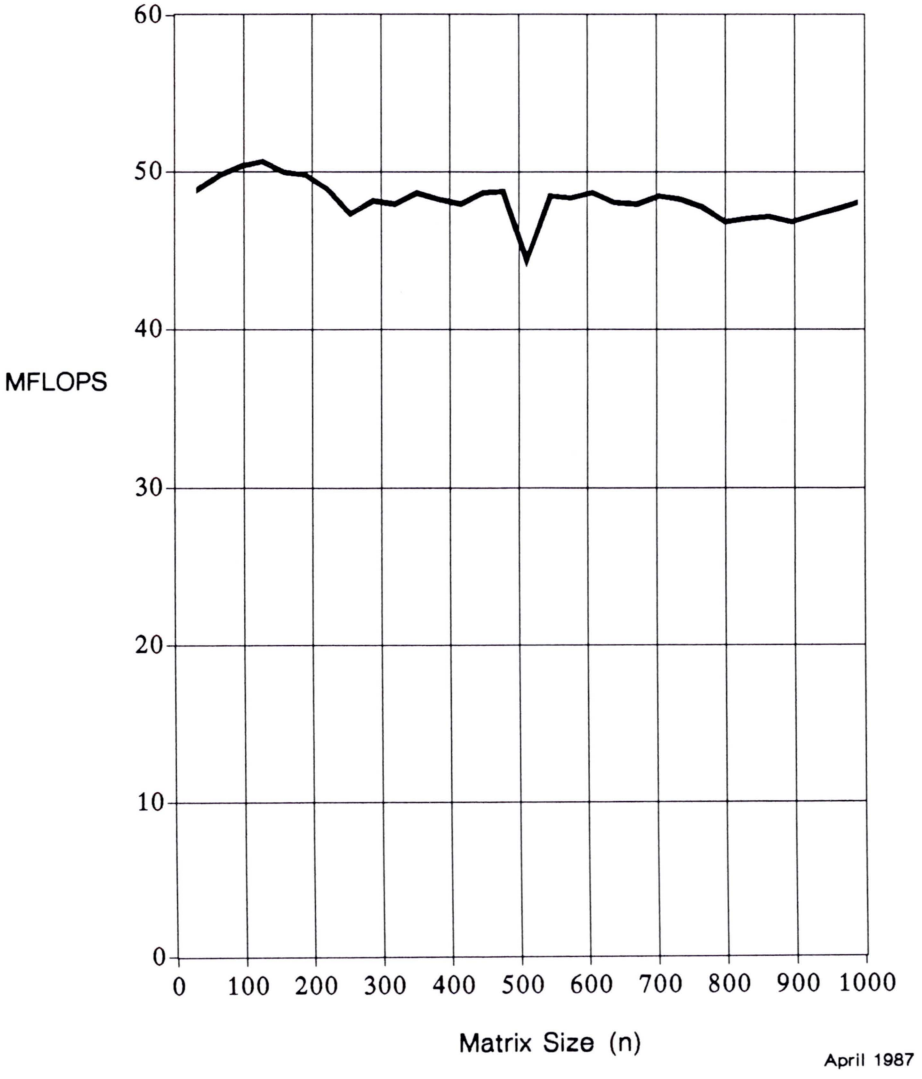


Figure 6-11: Performance for the multiplication of two double-precision *n*x*n*-matrices on an Alliant FX/8 with 8 CEs

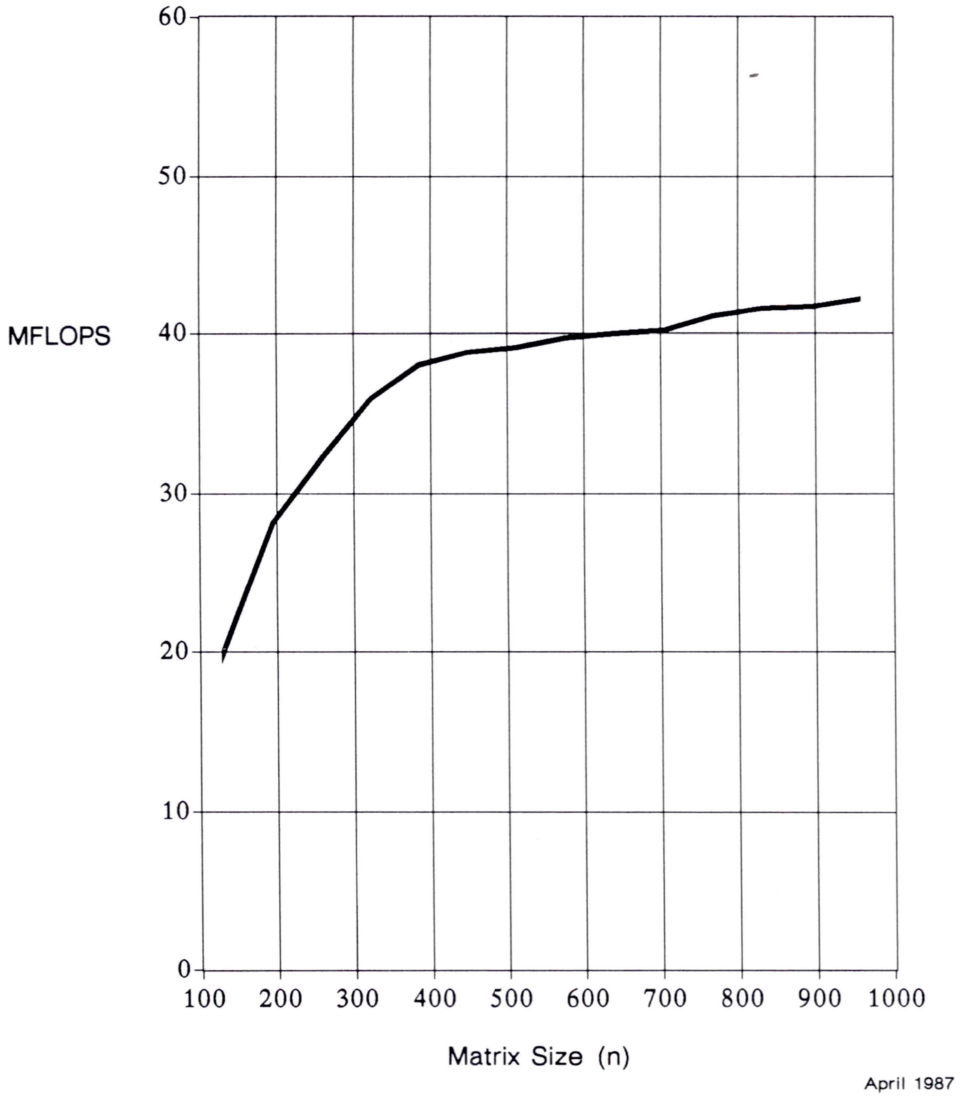


Figure 6-12: Performance for block LU decomposition for double precision *nxn* matrix on an Alliant FX/8 with 8 CEs

In addition to the FX/Series Scientific Library, Alliant customers can choose from a wide range of libraries developed by third party software vendors and made available on the FX/Series. These math libraries include the IMSL libraries, the Numerical Algorithms Group (NAG) library, Math Advantage from Quantitative Technology Corporation (QTC) and ParaEig and ParaLin from Paracomp.

CHAPTER 7

FX/Ada Development System

FX/Ada Development System

The FX/Ada Development System for the Alliant FX/Series of systems is centered on a high performance, production-quality Ada compiler that is fully compliant with ANSI/MIL-STD-1815A (validated June 1987). FX/Ada provides a total Ada production environment which includes the Ada compiler, a screen-oriented symbolic debugger, library maintenance utilities and programming tools, and a runtime system, all utilizing advanced technology to provide both ease of use and efficiency.

The FX/Ada Compiler, along with the extensive Ada tool set, operates as reentrant, shareable processes under Concentrix, making efficient use of Concentrix facilities on the FX/Series Systems.

FX/Ada, an enhanced version of the Verdix Ada Development System, combines Alliant's unique parallel processing architecture with the parallel constructs inherent in Ada.

FX/Ada Development System for FX/Series Systems

The FX/Ada Compiler is truly production-quality. It has been carefully designed and implemented to maximize both compilation speed and runtime efficiency. On a single CE the Compiler processes an average of 1,800 Ada source statements per minute, as measured by compiling the Dhrystone, based on total user plus system CPU time. An FX/8 with 8 CEs supports eight parallel compilations to provide an aggregate average compilation speed of 13,000 Ada source statements per minute.

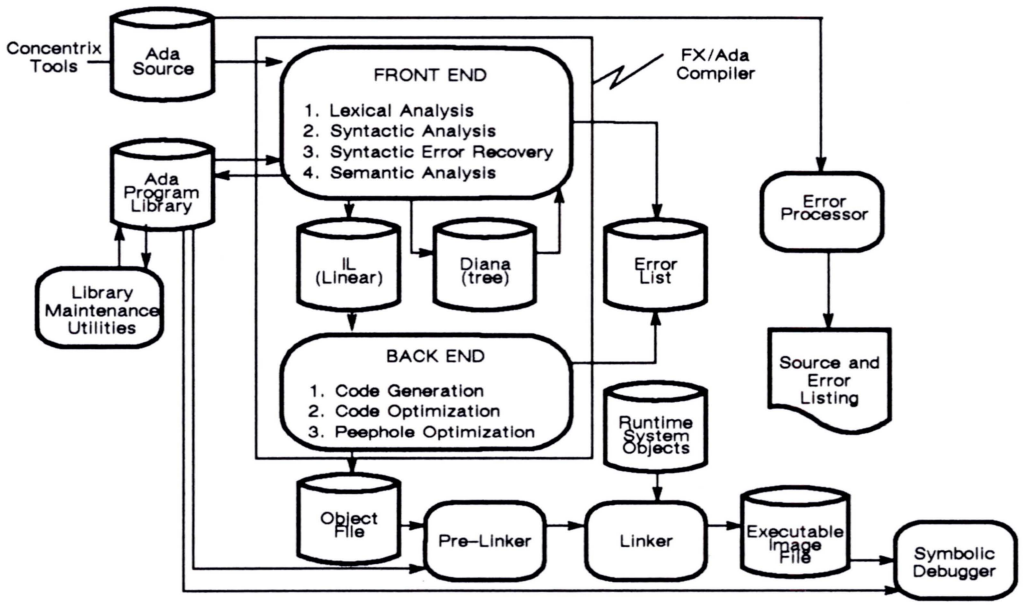


Figure 7-1: FX/Ada Development System

The FX/Ada Compiler is easy to use, utilizing innovative syntactic error-recovery techniques to maximize the number of errors found per compilation. To provide further help with semantic error messages, FX/Ada generates a concise description of the error and directs the user to a specific subsection of the Ada Reference Manual for a more detailed explanation.

The Compiler generates native code for the FX/Series System. This provides object performance substantially higher than intermediate code (pseudo-code) compilers and also allows machine-dependent optimization to be performed. An optimizer boosts executive performance. Further improvements to the optimizer are planned.

Most of the features of chapter 13 (representation clauses) of the U.S. government's Ada Language Reference Manual have been implemented. These features allow the programmer to "get closer to the machine", through functionality such as interrupt handling, which is important for realtime applications.

Library Management

The implementation of an Ada program library must be efficient in order to support the development of large-scale programs. FX/Ada's library management facilities organize, manage, manipulate, and display Program Library information in a way that enhances the compiler's performance. The FX/Ada library utilities provide for library and sublibrary creation and deletion; library dependency link creation, link removal, and cross referencing.

In addition, the FX/Ada Development System permits Ada Program Libraries to be hierarchically organized. This structure enables programmers to work without interference on local versions of individual program units, while retrieving the remainder of the program from higher-level libraries. This capability further facilitates the support of large development projects.

Program Creation

To facilitate program creation and maintenance, FX/Ada provides a utility (similar to UNIX's "make") that populates and maintains a program library with the program dependency information derived from Ada source files. This utility uses information contained in the program library to verify consistency and to perform the minimal compilations necessary to build an Ada unit.

FX/Ada provides a link preprocessor for the purpose of insuring the consistency and integrity of the program to be linked. This facility locates all dependent units required in a user program, verifies that all units are up-to-date, and builds additional tables for runtime support and control. The link pre-processor then calls the Concentrix loader to link the compiled units to produce an executable program image.

FX/Ada Debugger

The FX/Ada Debugger provides a symbolic, interactive, debugging facility with both line and screen modes. The debugger has access to the program's symbol table (the Diana) so that it can use its knowledge of the program's entities to recognize user types and variables, scopes and/or tasks, and to provide true high-level language debugging capabilities.

The Debugger has sophisticated breakpoint capabilities. Breakpoints may be conditional, based on the value of variables in the running program. The Debugger also provides single-step execution on either a source line or machine instruction basis, where the user optionally can skip over procedure invocations.

Because it is important for the programmer to have access to the Ada source code and machine instructions during a debugging session, the Debugger is integrated with the breakpoint and stepping capabilities, giving the user a source "window" into the program at all times. Since much of debugging involves the user switching attention between the executing program and the program source code, this facility has been made simple, convenient, and fast.

Like all the tools in the FX/Ada Development System, the Debugger is designed to be easy to learn and natural to use, and an on-line help facility is provided.

FX/Ada Runtime System

The FX/Ada Runtime System provides comprehensive support for tasking, exception handling, and input/output. The runtime system is linked with the user's program. It controls subsequent program execution and provides the principal interfaces to Concentrix. The resource utilization of the Runtime System has been enhanced by the availability of shared generics.

Parallel Tasking

Ada is a programming language that provides for parallel execution of multiple program tasks, making it ideal for the Alliant FX/Series systems. FX/Ada version 1.1 supports the parallel execution of tasks on multiple computational elements (CEs) and interactive processors (IPs), resulting in very fast execution of applications. The user simply specifies at execution time the resources (number of CEs and IPs) to apply to the job. Additionally, the user can bind tasks to specific resources to achieve maximum control and high performance.

Mix of Languages

FX/Ada shares a common calling convention with FX/Fortran, FX/C, and Pascal. Therefore, routines can be written in any of these languages and call each other directly. This feature enables programmers to use existing Fortran routines while building a new application with new routines written in Ada. This capability protects the user's software investment and enables quick application development.

Additionally, these other language routines can use hardware vectorization and concurrency to achieve very high performance for critical sections of an application.

Summary

The FX/Ada Development System is a production-quality system intended for the large-scale development of both application and systems software. It is based on a mature, widely-used compiler, thus utilizing proven technology. As such, the FX/Ada Development System goes beyond the requirements of the Ada language standard and provides a truly usable development and execution capability.

CHAPTER 8

Reliability, Availability, and Serviceability

Alliant systems incorporate conservative design practices, extensive logical and electrical simulation, and low-power high-density CMOS gate array logic. All elements are important factors that contribute to the integrity of the system.

Component testing and environmental preconditioning prior to assembly, voltage and clock-margined testing, and extended system burn-in help to assure that Alliant systems are consistently manufactured to specification.

In the event of failure, diagnostic and fault-tolerant features support rapid fault isolation and repair.

Diagnostic System

A layered set of diagnostic functions maintain high availability and serviceability in the field. Every Alliant system includes an independent hardware and software diagnostic system that supports local and remote fault diagnosis and system configuration.

The diagnostic system is controlled by the Diagnostix™ operating system, a diagnostic subset of Concentrix, that executes in the system interactive processor (IP).

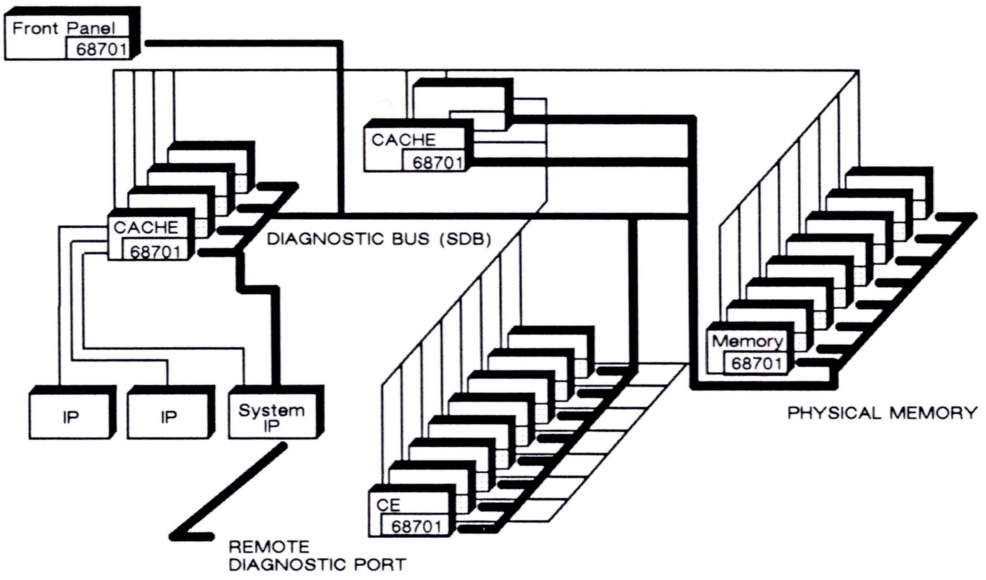


Figure 8–1: Independent diagnostic processors on each module and a dedicated system diagnostic bus and remote port assure high system confidence and low mean time to repair.

Diagnostix is totally memory-resident in the system IP and from there can initiate system start-up or diagnostic analysis. During system start-up or reset, Diagnostix initiates verification tests of all system modules, controls the loading of microcode over the independent system diagnostic bus (SDB), sizes and configures the system, and manages the start-up and initialization of Concentrix. Diagnostix also supervises the execution of all diagnostic programs, including a complete set of microcoded diagnostics that perform gate-level testing for each system hardware module and functional diagnostics that test system architecture integrity. System exercisers that emulate actual worst case run-time conditions run under Concentrix. Running in sequence, these diagnostic programs thoroughly test the entire system.

All system modules contain a dedicated Motorola 68701-based diagnostic processor that communicates over the SDB. Each diagnostic processor contains dedicated error registers for fault capture, and non-volatile memory that maintains module and system configuration data including serial number, revision level, and unique diagnostic information. The independent SDB provides error capture without using system data buses, aiding symptomatic fault evaluation and isolation.

The system's error logging facility is supported by extensive fault detection hardware on all major system data busses and memory arrays. Single-bit error correction, double-bit error detection, and memory scrubbing are supported within the system memory. All power supplies are monitored by the diagnostic system for out-of-range voltage conditions and input power brown-out or loss. In addition, the environmental status is monitored for out-of-range conditions.

Diagnostix User Interface

The user interface to Diagnostix is similar to the C Shell in Concentrix. A full range of shell commands and run-time routines are provided by Diagnostix to examine, emulate, and exercise all system functions. The interface provides complete access to the diagnostic system through a directory structure similar to the tree-structured directories of Concentrix.

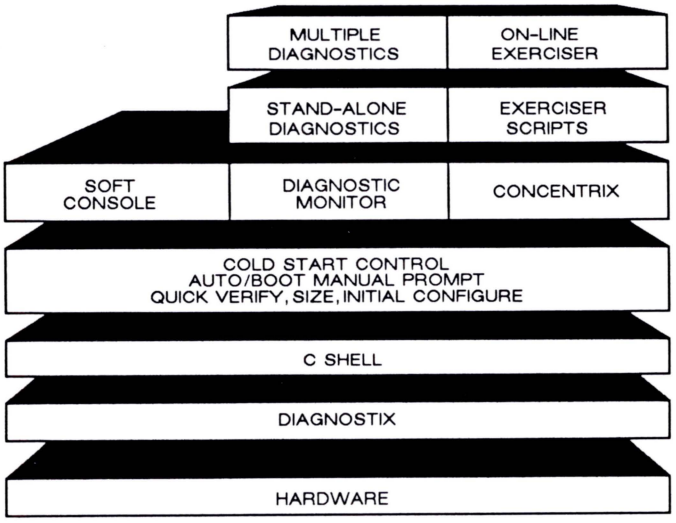


Figure 8-2: A layered set of diagnostic tools provide for high system confidence and low mean time to repair.

The soft console can examine the hardware configuration as if it were a tree. Each machine is the root directory of the tree, with module type the second level, and individual modules the third. Registers (or RAM arrays) on each module are accessi-

ble to the user, and form the fourth level. Accessing different modules of the machine is identical to changing directories in Concentrix. Commands similar to UNIX find, cd, pwd, and ls allow for selective examination of hardware control and status registers. Access can be to a single register, all registers on a single module, or identical registers on different modules. For example, the command CE.D0 accesses the D0 registers in all installed CEs.

Fault Tolerance

The FX/8 typically provides sufficient system redundancy to support a significant degree of fault tolerance. All replicated system elements (CEs, IPs, caches, and memory modules) are locally and remotely reconfigurable under Diagnostix so that faulty elements can be logically disconnected from the system, allowing operation to resume after a failure.

The computational complex, for example, can be reconfigured should a CE experience a hard fault. Degraded operation can continue after Diagnostix has disabled the failing computational element.

IPs are similarly reconfigurable, provided failures do not disable input/output paths to peripheral controllers. In that case, the controllers on the disabled bus must be manually reconfigured. The Alliant architecture assigns all devices a board-level address so that peripheral controllers attached to an IP can be moved to another IP with no reconfiguration.

The eight-megabyte memory module is organized into four two-megabyte quadrants. The 32-megabyte memory module is organized into four eight-megabyte quadrants. Up to two quadrants can be selectively disabled by Diagnostix in the event multiple hard errors occur within a quadrant.

System Serviceability

Serviceability is enhanced through the use of only six system module types in a fully configured FX/8 system, and four module types in the FX/1. Software-controlled clock and voltage margining provides a tool for improved isolation of intermittent problems and early fault detection during scheduled maintenance. All configuration parameters are specified through software, eliminating the need for hardware switches or jumpers except on peripheral controllers.

All Alliant systems support diagnostic sessions that can be invoked locally or remotely under Diagnostix or Concentrix. Diagnostic sessions minimize system downtime and maintenance costs by giving Alliant's customer service personnel direct access to the customer's system during problem evaluation and resolution. Once identified, hardware problems can be resolved through module replacement or, in many instances, remote reconfiguration. Degraded operation can continue until the failing component is replaced. In addition, regularly scheduled remote monitoring sessions perform predictive maintenance checks and error log analysis, minimizing unscheduled downtime.

Concentrix contains a broad range of system error logging and on-line diagnostic capabilities, permitting on-line fault analysis and system maintenance. In addition, Concentrix contains an on-line system acceptance program that duplicates Alliant's final acceptance stress test. This program is run on-site to assure full system integrity before customer acceptance at installation or upgrade.

System Upgrades

Since all unique system and module parameters (for example, serial numbers, configuration parameters, revision levels) are set and interrogated by software, the system is easily reconfigured or upgraded. The modular system construct also ensures that extensive downtime is avoided whenever customers upgrade the relative performance of their systems with add-on hardware.

CHAPTER 9

System Expansion

The Alliant architecture provides for field expansion of all system resources. This flexibility and the use of industry-standard I/O allow a range of options in configuring and expanding systems.

FX/8 System Building Block

Every Alliant FX/8 system begins with a system building block (SBB). The FX/8 SBB, combined with required disk, tape, and system options, is a fully operational system. No additional hardware is required. All of the details of the configuration, including cables, connections, and blank panels are supplied. The system designer must consider only function and performance.

FX/8 System Expansion

Alliant FX/8 systems can be factory configured or field expanded for a wide range of technical applications. Cache and physical memory can be added as program size and user loads grow. IPs can be added to improve interactive and operating system performance and to support peripheral expansion. More importantly, additional CEs can be added to the FX/8 for expansion of delivered computational performance.

FX/1 Systems

The FX/1 system building block can be combined with the required disk, tape, and system software options to form a complete system. Complete FX/1 packaged systems are also available.

Peripheral Expansion

Alliant peripherals include disk and tape systems, terminals and printers, and local area networking products. Multibus controllers for the peripherals are supported by the IPs.

To ensure sustained I/O and operating system performance, the system designer must consider two factors:

- The balance between the CPU cycles and memory bandwidth of each IP, and the requirements (or load) of the attached peripheral controllers.
- The availability of sufficient system-wide IP resources to service operating system activity not associated with the peripheral controllers.

Alliant customer service personnel can provide guidelines when configuring new systems or upgrading existing systems to achieve a desired level of performance.

Specifications

The following pages detail physical specifications for Alliant systems and expansion options. Tables 9-1 and 9-10 describe the FX/8 and the FX/1 system building blocks. Each table is followed by specification sheets for typical peripheral expansion options. Tables 9-17 through 9-22 contain specifications for peripherals and controllers that can be configured with any Alliant system.

Table 9-1: FX/8 System Building Block and Expansion

FX/8 System Building Block

Computational Element
Two 8MB Memory Modules or one 32MB memory Module
One 256MB Computational Processor Cache Memory
One 32KB Interactive Processor Cache Memory
System Floppy Disk and Controller
System Cabinet with power and expansion for maximum system (see configuration rules)
System Interactive Processor and Multibus Memory Module
All power, cooling, and cabling
Installation, Documentation, and 90-day Warranty

Each FX/8 System Building Block must be ordered with:

FX/8 Peripheral Building Block, or
FX/8 Tapeless Node
Concentrix UNIX Operating System
Master Console (printing and video terminal)

System Expansion

Additional Computational Element
8 or 32MB Memory Module
256KB Computational Processor Cache
32KB Interactive Processor Cache

Software

Concentrix (16, 32, and 64 users)
FX/Ada Compiler
FX/C Compiler
FX/Fortran Compiler
Pascal Compiler
Network File System
VAX/VMS DCL-like User Interface
DECnet
X.25
TCP/IP (included with Concentrix)
HASP
Hyperchannel
Network Computing System
X Windows
NeWS
Scientific Library

Peripheral Expansion

Disk Systems

Dual 8" Winchester Disks
10 1/2" Winchester Disk
SMD Disk Controller

Terminals & Printers

Video Terminal
Console Printer
600 LPM Printer and Controller
1200 LPM Printer and Controller

Magnetic Tape Systems

50 IPS Tri-Density Tape Drive and Enclosure
125 IPS Tri-Density Tape Drive and Enclosure
Magnetic Tape Controller

I/O Expansion

Interactive Processor
Configurable Multibus Chassis
Communication Controller (16 lines)
Synchronous Controller
Ethernet Controller
Additional I/O Expansion Cabinet

System Cabinet and I/O Expansion Cabinet

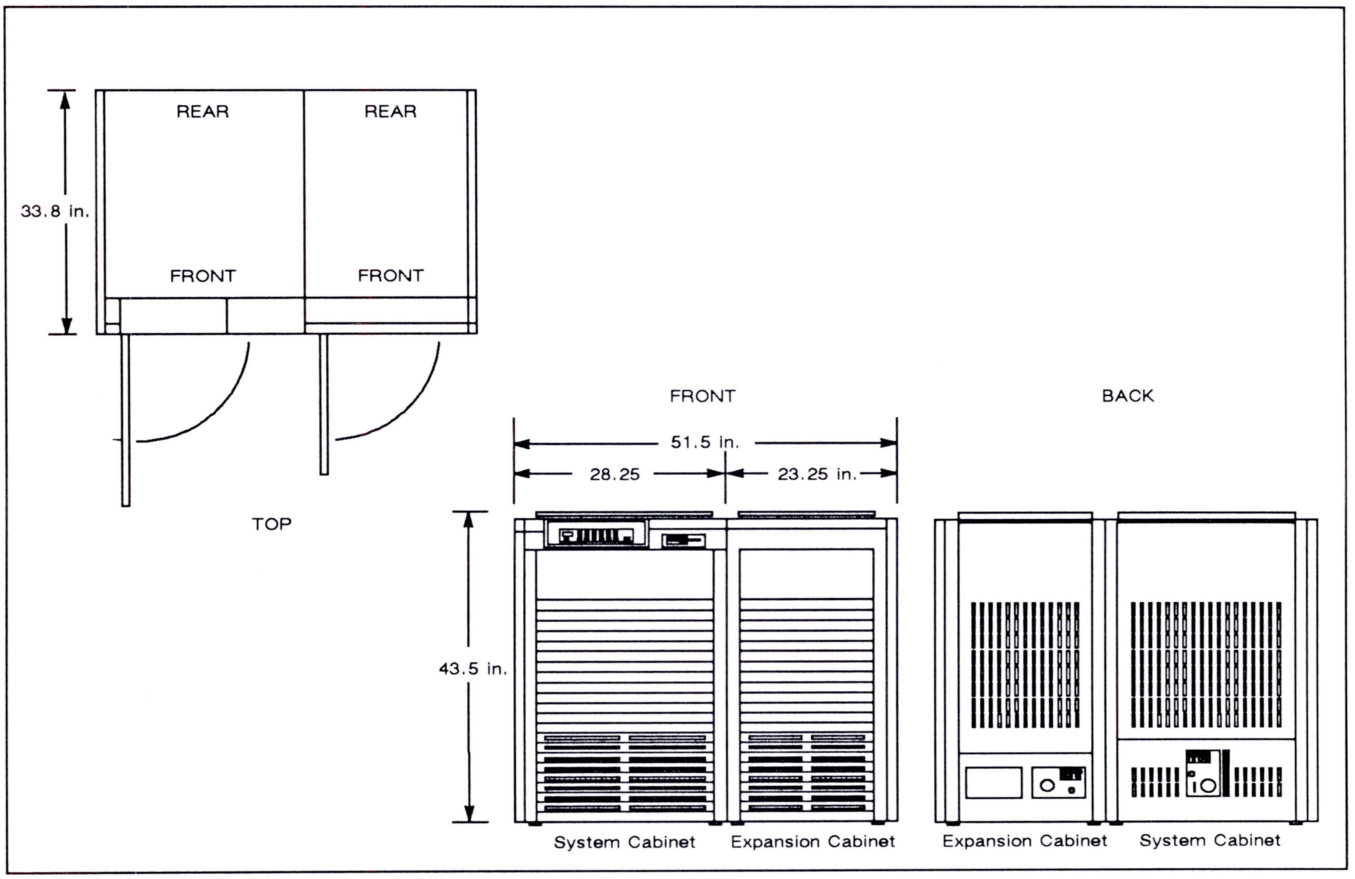


Table 9-2: FX/8 System Cabinet Specifications

SYSTEM CABINET PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT	110.5 CM (43.5 in.)
WIDTH	75.0 CM (29.5 in.)
DEPTH	85.8 CM (33.8 in.)

Service Access

FRONT	107 CM (42 in.)
REAR	107 CM (42 in.)

Weight

BUILDING BLOCK	180 KG (400 lbs.)
MAX CONFIG	225 KG (500 lbs.)

Heat Output

BUILDING BLOCK	1900 Watts (6500 BTU/hr.)
MAX CONFIG	4950 Watts (16,900 BTU/hr.)

SYSTEM CABINET CONFIGURATION

Expansion Modules	In SBB	Max
COMPUTATIONAL ELEMENT	1	8
8/32-MB MEMORY	2/1	8/8
256-KB CP CACHE	1	2
32-KB IP CACHE	1	4

Available Slots	Memory	Cache	CE
PROVIDED IN SYS CAB	8	6	8
USED IN BUILDING BLOCK	2/1	2	1
AVAILABLE FOR EXPANSION	6/7	4	7

Slot Utilization	Memory	Cache	CE
COMPUTATIONAL ELEMENT	0	0	1
8/32MB MEMORY	2/1	0	0
256KB CP CACHE	0	1	0
32KB IP CACHE	0	1	0

Table 9-2: FX/8 System Cabinet Specifications (Continued)

SYSTEM CABINET PHYSICAL SPECIFICATIONS

Power	USA	INT**
VOLTAGE	120/280V	220/380V*** 240/415V***
TOLERANCE (VOLTS)	-15% + 10%	-15% + 10%
PHASES	3	3
BRANCH CIRCUIT REQ.	30A	20A
CURRENT (AMPS)	24	15
FREQUENCY	60 ± 2Hz	50 ± 2Hz
CONNECTOR (NEMA)*	L21-30P	NA
CONNECTOR (Hubbell)*	2811	NA
RECEPTACLE (NEMA)	L21-30R	NA
RECEPTACLE (Hubbell)	2810	NA
CABLE LENGTH	12 Feet	12 Feet
POWER AVAILABLE	6.6 KVA	6.6 KVA

*Connector Supplied
**For Japan consult with customer representative
***No restrapping required

Environmental	Operating	Storage
TEMPERATURE °C	0 to 32	-40 to 66
°F	32 to 90	-40 to 150
RELATIVE HUMIDITY (NON-CONDENSING)	40% to 90%	10% to 95%
ALTITUDE	2400 M (8000 ft)	3000 M (10,000 ft.)

SYSTEM CABINET CONFIGURATION

Available Power

PROVIDED IN SYSTEM CABINET	3000 Watts
USED IN BUILDING BLOCK	675 Watts
AVAILABLE FOR EXPANSION	2325 Watts

Power Utilization

COMPUTATIONAL ELEMENT	190 Watts
8MB MEMORY	160 Watts
256KB CP CACHE	165 Watts
32KB IP CACHE	155 Watts
MIB MODULE	140 Watts

System Floppy Disk Drive*

CAPACITY 655KB

* Multibus Controller included (requires one Multibus slot)

Table 9-3: FX/8 I/O Expansion Cabinet Specifications

I/O EXPANSION CABINET PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT 110.5 CM (43.5 in.)
WIDTH 59.9 CM (23.25 in.)
DEPTH 85.8 CM (33.8 in.)

Service Access

FRONT 107 CM (42 in.)
REAR 107 CM (42 in.)

Weight Empty 90 KG (200 lbs.)

Power	USA	INT**
VOLTAGE	125/250V center tap	220V*** 240V***
TOLERANCE (VOLTS)	-15% + 10%	-15% + 10%
PHASES	1	1
BRANCH CIRCUIT REQ.	20A	15A
CURRENT (AMPS)	15	12
FREQUENCY	60 ± 2Hz	50 ± 2Hz
CONNECTOR (NEMA)*	L14-20P	Stripped End
CONNECTOR (Hubbell)*	2411	Stripped End
RECEPTACLE (NEMA)	L14-20R	NA
RECEPTACLE (Hubbell)	2410	NA
CABLE LENGTH	12 Feet	12 Feet
POWER AVAILABLE	2.3 KVA	2.3 KVA

*Connector Supplied
**For Japan consult with customer representative
***No restrapping required

Heat Output (Max) 2100 Watts (7200 BTU/hr.)

I/O EXPANSION CABINET CONFIGURATION

Multibus

ONE MULTIBUS CHASSIS PROVIDED IN SBB
SEE MULTIBUS CHASSIS SPECIFICATIONS (Pg. 7-9)

Peripheral Rack Space Utilization

MULTIBUS CHASSIS	22.3 CM (8.75 in.)
DUAL 8" WINCHESTER DISKS	13.25 CM (5.25 in.)
10 1/2" WINCHESTER DISK	26.5 CM (10.5 in.)

Wiring: Controller to Peripheral

INTERNAL TO INTEGRATED SYSTEM

Multibus Chassis Mounting

MOUNTED IN LOWEST AVAILABLE POSITION (FROM BOTTOM FIRST)

Peripheral Mounting

STANDARD R.E.T.M.A. 19" RACK
AVAILABLE SPACE 75.6 CM (31.5 in.)

Terminal Connectors 32 AVAILABLE

FX/8

Tape Drive Cabinet (50 IPS) Specifications

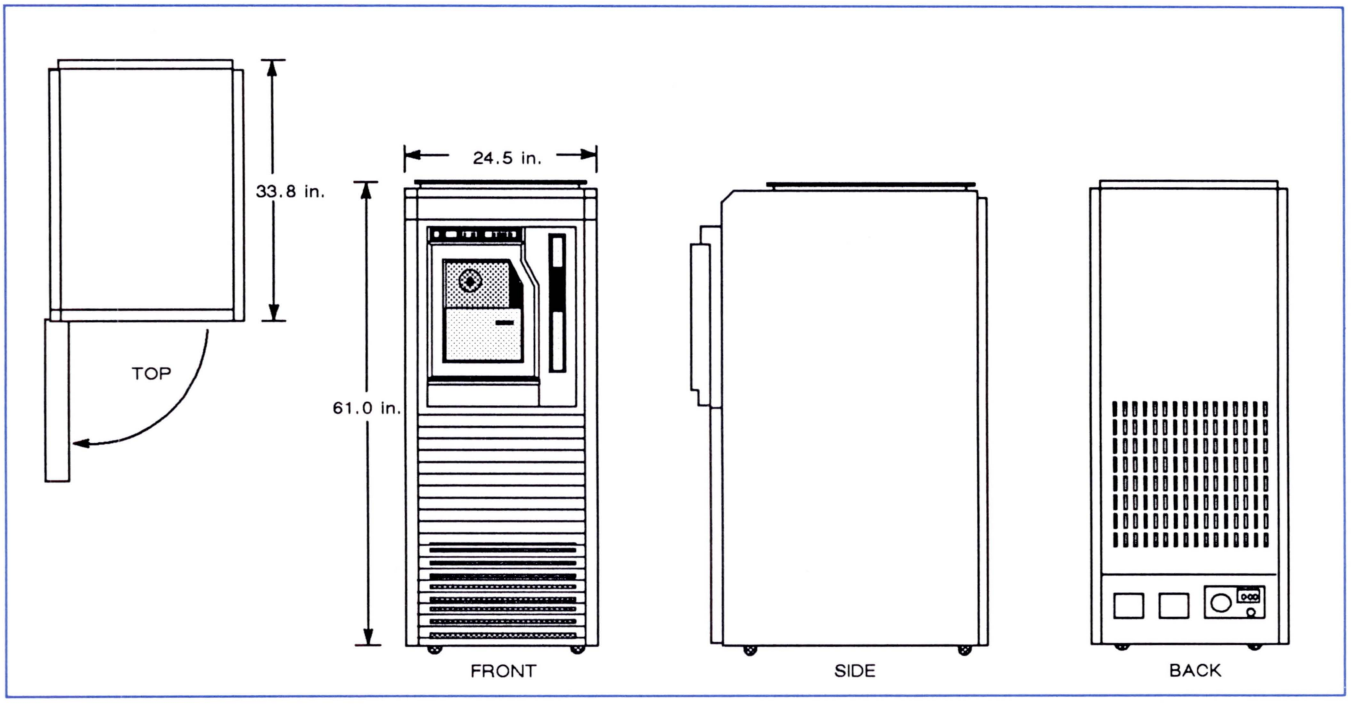


Table 9-4: FX/8 Tape Drive Cabinet (50 IPS) Specifications

PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT	155 CM (61 in.)
WIDTH	62.2 CM (24.5 in.)
DEPTH	85.8 CM (33.8 in.)

Service Access

FRONT	107 CM (42 in.)
REAR	107 CM (42 in.)

Weight 146 KG (325 lbs.) (includes tape drive)

Power	**USA**	**INT****
VOLTAGE	125/250V center tap	220V*** 240V***
TOLERANCE (VOLTS)	−15% + 10%	−15% + 10%
PHASES	1	1
BRANCH CIRCUIT REQ.	20A	15A
CURRENT (AMPS)	15	12
FREQUENCY		50 ± 2Hz
CONNECTOR (NEMA)*	L14-20P	Stripped end
CONNECTOR (Hubbell)*	2411	Stripped end
RECEPTACLE (NEMA)	L14-20R	NA
RECEPTACLE (Hubbell)	2410	NA
CABLE LENGTH	12 Feet	12 Feet
POWER AVAILABLE	2.3 KVA	2.3 KVA

*Connector Supplied
**For Japan consult with customer representative
***No restrapping required

Heat Output (max) 2100 Watts (7200 BTU/hr.)

CONFIGURATIONS

Peripheral Mounting

STANDARD R.E.T.M.A.	19" RACK
TOTAL MOUNTING SPACE AVAILABLE	120.0 CM (47.25 in.)
USED BY TAPE	62.2 CM (24.5 in.)
AVAILABLE	57.8 CM (22.75 in.)

Peripheral Rack Space Utilization

MULTIBUS CHASSIS	22.2 CM (8.75 in.)
WINCHESTER DISK	26.5 CM (10.5 in.)

Wiring: Controller to Peripheral

INTERNAL TO INTEGRATED SYSTEM

Multibus Mounting Chassis

WILL BE MOUNTED IN LOWEST AVAILABLE POSITION (FROM BOTTOM FIRST)

Terminal Connectors 32 AVAILABLE

FX/8

Tri-Density Tape Drive (50 IPS) Specifications

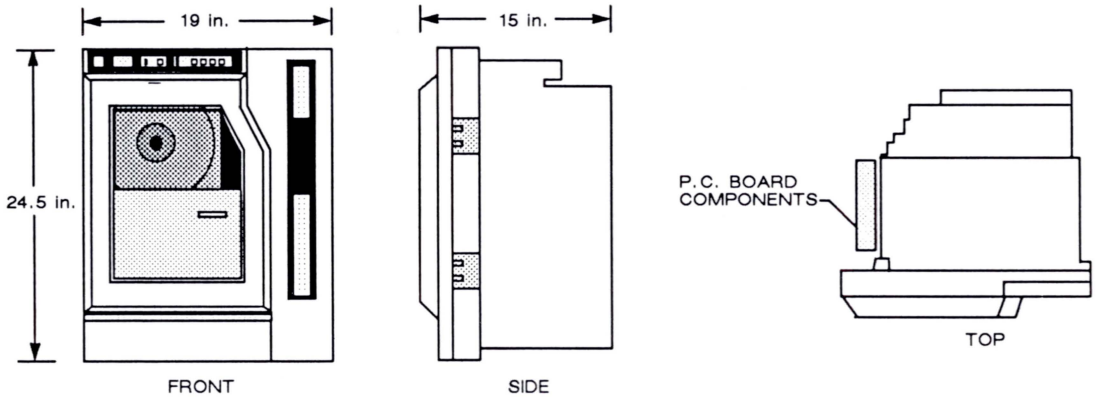


Table 9-5: FX/8 Tri-Density Tape Drive (50 IPS) Specifications

FUNCTIONAL SPECIFICATIONS

Tape Speed	50 IPS
Recording Density	6250 BPI/GCR 1600 BPI/PE 800 BPI/NRZI

Access Time	**Read**	**Write**
GCR	3.4 MS	3.0 MS
PE/NRZI	5.0 MS	3.0 MS

Transfer Rate	
6250	312.5KB/sec.
1600	80.0KB/sec.
800	40.0KB/sec.

Rewind (2400' reel)	2 min.
Reel Sizes (inches)	6/7/8.5/10.5
Density Select	Manual or Software
MTBF (design)	6000 hrs.
Expansion	1 Master 3 Slaves

PHYSICAL SPECIFICATIONS

Dimensions (rack)

HEIGHT 62.3 CM (24.5 in.)
WIDTH 48.3 CM (19.0 in.)
DEPTH 38.1 CM (15.0 in.)

Weight 52 KG (115 lbs.)

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	5 to 43	–45 to 70
°F	41 to 110	–50 to 160
RELATIVE HUMIDITY (NON-CONDENSING)	30% to 80%	5% to 95%

Power Consumed 440 Watts

CONTROLLER SPECIFICATIONS

Drives Supported 4

Data Verification

800 BPI	Horizontal and Vertical Parity
1600 BPI	Phase encoding
6250 BPI	Group encoding

Transfer Mode	DMA
FIFO Buffer	8KB
Board Type	Multibus (one slot)

FX/8

Tape Drive Cabinet (125 IPS) Specifications

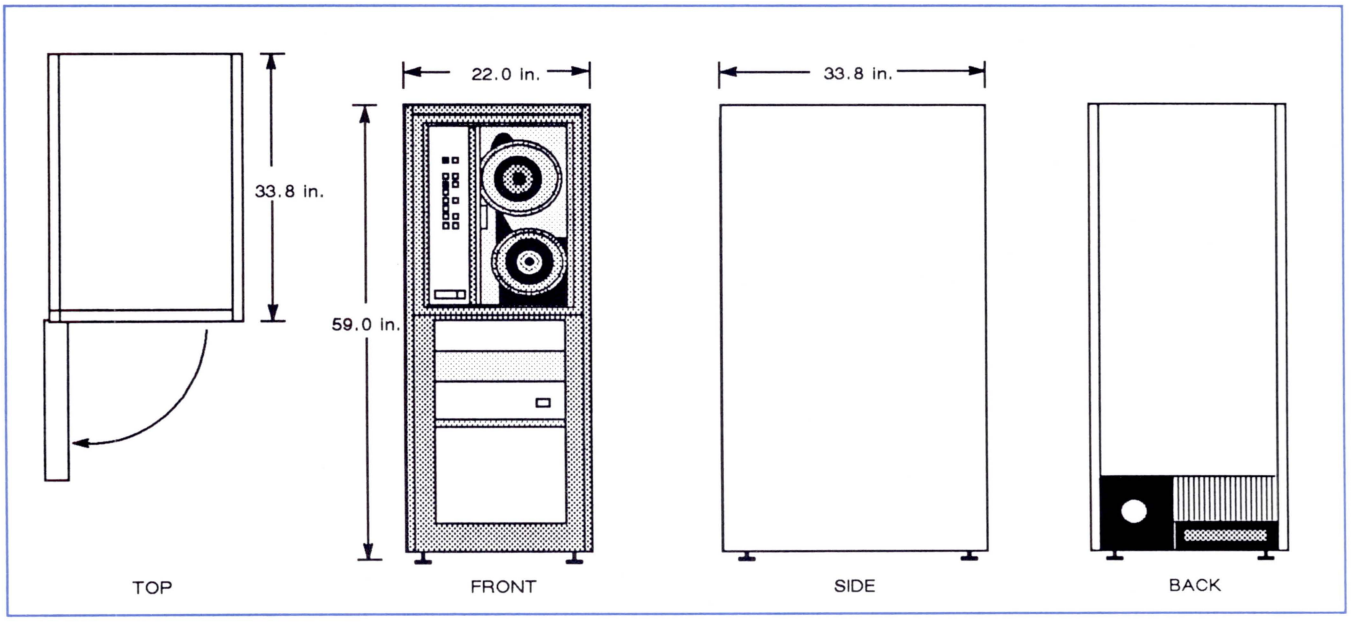


Table 9-6: FX/8 Tri-Density Tape Drive (125 IPS) Specifications

PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT	149.9 CM (59.0 in.)
WIDTH	56.0 CM (22.0 in.)
DEPTH	85.8 CM (33.8 in.)

Service Access

FRONT	107 CM (42 in.)
REAR	107 CM (42 in.)

Weight 146 KG (325 lbs.) (includes tape drive)

Power	**USA**	**INT****
VOLTAGE	125VAC center tap	220V 240
PHASES	1	1
BRANCH CIRCUIT REQ.	30A	20A
CURRENT (AMPS)	14	7
FREQUENCY	60 ± 2Hz	50 ± 2Hz
CONNECTOR (NEMA)*	L5-30P	L6-20P
CONNECTOR (Hubbell)*	2611	2321
RECEPTACLE (NEMA)	L5-30R	L6-20R
RECEPTACLE (Hubbell)*	2610	2320
CABLE LENGTH	12 Feet	12 Feet

*Connector Supplied
**For Japan consult with customer representative

Heat Output (max) 1300 Watts (7200 BTU/hr.)

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	5 to 40	-45 to 70
°F	40 to 100	-50 to 160
RELATIVE HUMIDITY (NON-CONDENSING)	30% to 80%	5% to 95%

FUNCTIONAL SPECIFICATIONS

Tape Speed 125 IPS

Recording Density
6250 BPI/GCR
1600 BPI/PE
800 BPI/NRZI

Access Time	**Read**	**Write**
GCR	1.2 MS	1.2 MS
PE/NRZI	3.0 MS	3.0 MS

Transfer Rate

6250	781.25KB/sec.
1600	200.0KB/sec.
800	100.0KB/sec.

Rewind (2400' reel) 1 min.
Reel Sizes (inches) 6/7/8.5/10.5
Density Select Manual or Software
MTBF (design) 6000 hrs.
Expansion 1 Master
3 Slaves

CONTROLLER SPECIFICATIONS

Drives Supported 4

Data Verification

800 BPI	Horizontal and Vertical Parity
1600 BPI	Phase encoding
6250 BPI	Group encoding

Transfer Mode DMA
FIFO Buffer 8KB
Board Type Multibus (one slot)

FX/8

Configurable Multibus Chassis Specifications

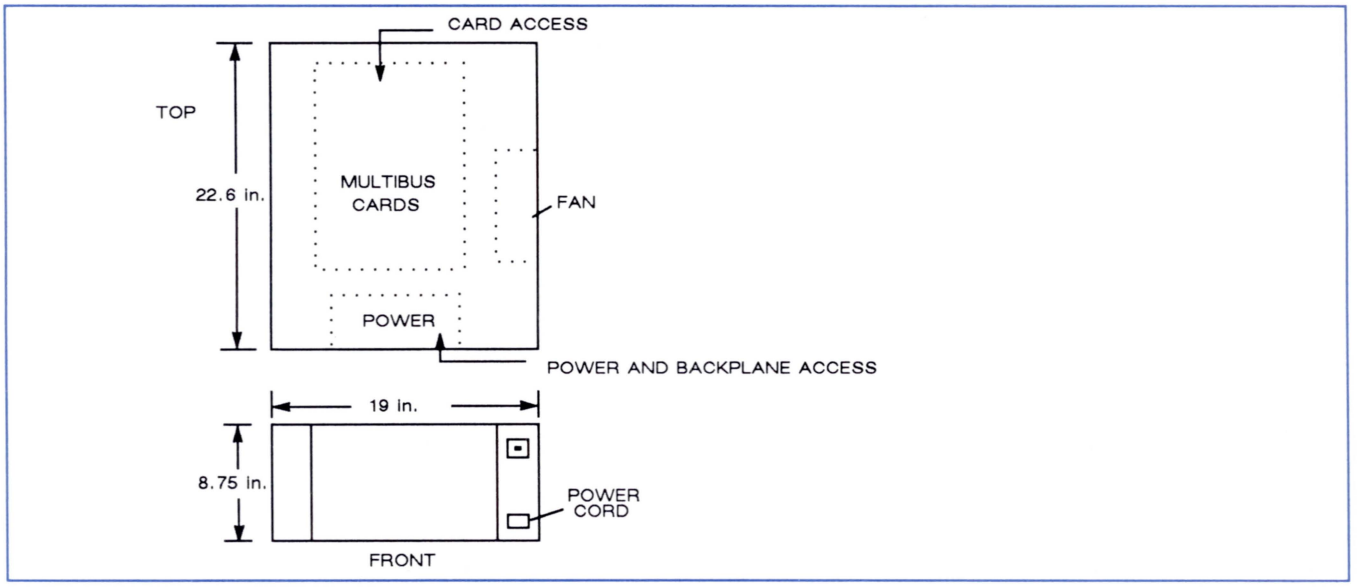


Table 9-7: FX/8 Configurable Multibus Chassis Specifications

PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT 22.2 CM (98.75 in.)
WIDTH STANDARD 19" RETMA RACK
DEPTH 57.4 CM (22.6 in.)

Weight (empty) 34 KG (75 lbs.)

Power

OUTPUT POWER	500 Watts Max	
DC OUTPUT	VOLTS	AMPS (Max)
	+5	80
	-5	5
	+12	8
	-12	8

Environmental	**Operating**	**Storage**
TEMPERATURE °C	0 to 32	-40 to 66
°F	32 to 90	-40 to 150
RELATIVE HUMIDITY (NON-CONDENSING)	40% to 90%	10% to 95%
ALTITUDE	2400 M (8000 ft.)	3000 M (10,000 ft.)

Controls Circuit Breaker
Multibus Reset

Backplane

P1 and P2 Connections Supplied

MULTIBUS CONFIGURATIONS

Slot Spacing 1.53 CM (0.6 in.)

Total Multibus Slots 13

Jumper Reconfigurable as:

Number Of Logically Independent Multibus Chassis	Slots per section 1	2	3
1	13	NA	NA
2	9	4	NA
2	5	8	NA
3	5	4	4

Notes:

(1) CHASSIS SUPPLIED WITH THREE MULTIBUS TERMINATORS

(2) CHASSIS MUST BE INSTALLED IN I/O EXPANSION OR TAPE DRIVE CABINET IMMEDIATELY ADJACENT TO SYSTEM CABINET

(4) IP0 MUST CONTAIN:

Systems Controller
SMD Disk Controller

FX/8

Dual 8" Winchester Disk Rack Specifications

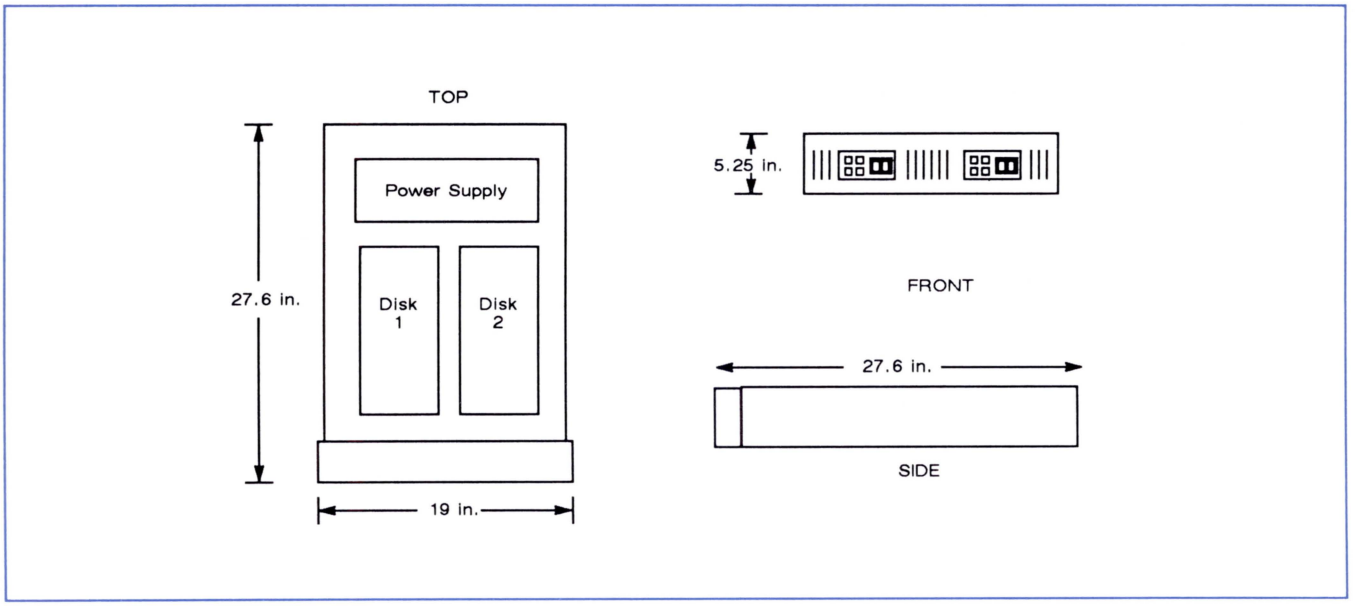


Table 9-8: FX/8 8" Winchester Disk Specifications

DISK FUNCTIONAL SPECIFICATIONS

Capacity	FORMATTED	268MB
	UNFORMATTED	337.1MB
Data Transfer Rate		2.36MB/sec.
Cylinders		823
Tracks Per Cylinder		10
Sectors Per Track		64
Spare Sectors Per Track		4
Bytes Per Sector		512
Rotation Speed		3600 RPM
Average Rotational Latency		8.3 MS
Positioning Time		
	TRACK TO TRACK	5 MS
	AVERAGE	20 MS
	MAXIMUM	40 MS
Track Density		683 TPI
Recording Density		19,734 BPI
Disk Reliability Specifications		
	MTBF (power on)	10,000 hrs.
	MTTR	30 min.

Note: 2 disks per rack

RACK PHYSICAL SPECIFICATIONS

Dimensions

HEIGHT 13.25 CM (5.25 in.)
WIDTH 48.3 CM (19.0 in.)
DEPTH 70.1 CM (27.6 in.)

Weight 65 KG (144 lbs)

Power Consumed 600 Watts

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	5 to 40	–40 to 60
°F	41 to 104	–40 to 140
RELATIVE HUMIDITY (NON-CONDENSING)	20% to 80%	5% to 95%
ALTITUDE	3000 M (10,000 ft.)	12,000 M (40,000 ft)

CONTROLLER SPECIFICATIONS

Drives Supported	4
Data Verification	32-bit ECC 11-bit Correction
Transfer Mode	DMA
FIFO Buffer	8KB
Interface Type	SMD (modified)

10 1/2" Winchester Disk Specifications

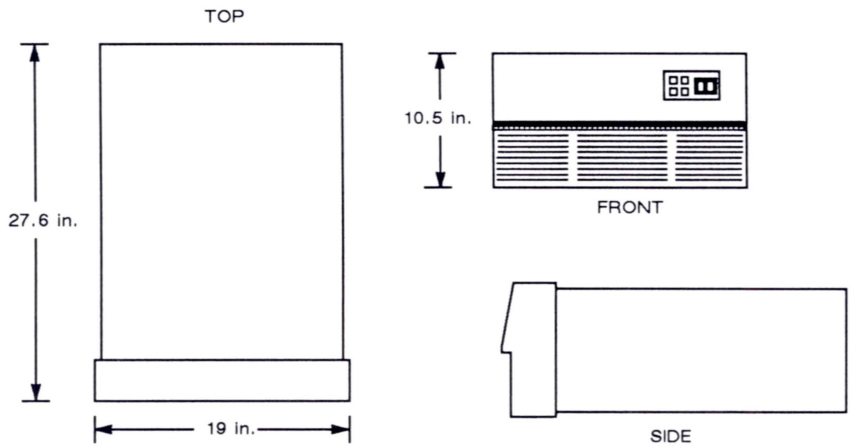


Table 9-9: FX/8 10 1/2" Winchester Disk Specifications

DISK FUNCTIONAL SPECIFICATIONS

Capacity	FORMATTED	379MB
	UNFORMATTED	474MB
Data Transfer Rate		1.859MB/sec.
Cylinders		842
Tracks Per Cylinder		20
Sectors Per Track		44
Spare Sectors per Track		2
Bytes Per Sector		512
Rotation Speed		3961 RPM
Average Rotational Latency		7.5 MS
Positioning Time		
	TRACK TO TRACK	5.5 MS
	AVERAGE	18 MS
	MAXIMUM	35 MS
Track Density		880 TPI
Recording Density		12,790 BPI
Disk Reliability Specifications		
	MTBF (power on)	20,000 hrs.
	MTTR	30 min.

DISK PHYSICAL SPECIFICATIONS

Dimensions

HEIGHT 26.7 CM (10.5 in.)
WIDTH 48.3 CM (19.0 in.)
DEPTH 70.1 CM (27.6 in.)

Weight 65 KG (144 lbs)

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	10 to 40	–40 to 60
°F	50 to 104	–40 to 140
RELATIVE HUMIDITY (NON-CONDENSING)	20% to 80%	5% to 95%
ALTITUDE	3,000 M (10,000 ft.)	12,000 M (40,000 ft.)

Power Consumed 600 Watts

CONTROLLER SPECIFICATIONS

Drives Supported	4
Data Verification	32-bit ECC 11-bit Correction
Transfer Mode	DMA
FIFO Buffer	8KB
Interface Type	SMD (modified)

Table 9-10: FX/1 System Building Block and Expansion

FX/1 System Building Block

Computational Element
8 or 32-MB Memory Module
32-KB Cache Memory
System Floppy Disk and Controller
System Cabinet with power and cooling
System Interactive Processor
6-slot Multibus Chassis
Installation, Documentation, and 90-day Warranty

Each FX/1 System Building Block must be ordered with:

138-MB 5 1/4" Winchester Disk, or
268-MB 8" Winchester, (requires FX/1 Expansion Cabinet), or
379-MB 10 1/2" Winchester (requires FX/8 I/O or Tape Cabinet)
45-MB 5 1/4" Streaming Cartridge Tape, or
Tri-Density Tape Drive and Controller
Concentrix UNIX Operating System
Master System Console (Video Terminal and Printer)

System Expansion

8 or 32-MB Memory Module

Software

Concentrix (16, 32, and 64 users)
FX/Ada Compiler
FX/C Compiler
FX/Fortran Compiler
Pascal Compiler
Network File System
VAX/VMS DCL-like User Interface
DECnet
X.25
TCP/IP (included with Concentrix)
HASP
Hyperchannel
Network Computing System
X Windows
NeWS
Scientific Library

Peripheral Expansion

Disk Systems

5 1/4" Winchester Disk
8" Winchester Disk
10 1/2" Winchester Disk
SMD Disk Controller

Terminals & Printers

Video Terminal
Console Printer
600 LPM Printer and Controller

Magnetic Tape Systems

45-MB 5 1/4" Streaming Cartridge Tape
Tri-Density Tape and Enclosure
Magnetic Tape Controller

I/O Expansion

Interactive Processor
Communications Controller (8 or 16 lines)
Synchronous Communications Controller
Ethernet Controller
Expansion Cabinet

FX/1

System Cabinet and I/O Expansion Cabinet

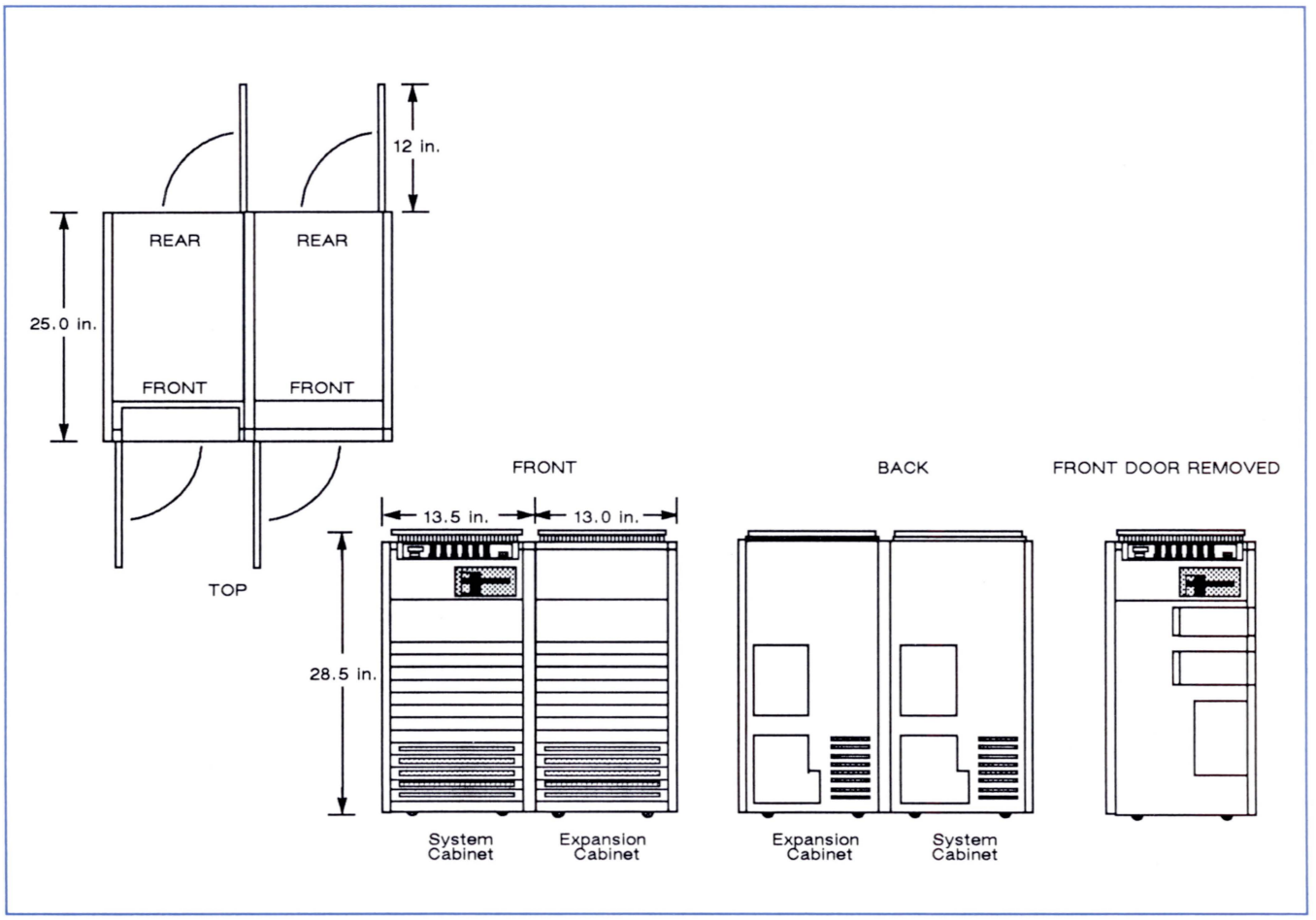


Table 9-11: FX/1 System Cabinet Specifications

SYSTEM CABINET PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT	72.4 CM (28.5 in.)
WIDTH	34.3 CM (13.5 in.) main only
DEPTH	63.5 CM (25.0 in.)

Service Access*

FRONT	38 CM (15 in.)
REAR	92 CM (36 in.)

*System mounted on casters

Weight

BUILDING BLOCK	45.4 KG (100 lbs.)
MAX CONFIG	56.8 KG (125 lbs.)

SYSTEM CABINET PERIPHERAL CONFIGURATION

Peripherals Supplied (see system peripherals)

5 1/4" SYSTEM FLOPPY
SYSTEM PERIPHERALS CONTROLLER

Peripheral Mounting Space Provided

5 1/4" 138MB WNCHESTER DISK
5 1/4" 45MB CARTRIDGE TAPE

Multibus

6 TOTAL SLOTS PROVIDED
2 SLOTS USED IN SBB
4 AVAILABLE SLOTS

Table 9-11: FX/1 System Cabinet Specifications (Continued)

SYSTEM CABINET PHYSICAL SPECIFICATIONS

Heat Output

BUILDING BLOCK	825 Watts (4200 BTU/hr.)
MAX CONFIG	1155 Watts (5880 BTU/hr.)

Power	USA	INT*
VOLTAGE	120V	220/240V
TOLERANCE	-15% + 10%	-15% + 10%
PHASES	1	1
BRANCH CIRCUIT REQ.	15A	15A
CURRENT (AMPS)	12	10
FREQUENCY	60 ± 2Hz	50 ± 2Hz
CONNECTOR (NEMA)	5 - 15P	as required by country
CONNECTOR (Hubbell)	5266	
RECEPTACLE (NEMA)	5-15R	
RECEPTACLE (Hubbell)	5262	
CABLE LENGTH	6 Feet	6 Feet
POWER AVAILABLE	1.7 KVA	1.7 KVA

*For Japan consult with customer representative

SYSTEM CABINET PHYSICAL SPECIFICATIONS

Environmental	Operating	Storage
TEMPERATURE °C	5 to 32	-40 to 66
°F	41 to 90	-40 to 150
RELATIVE HUMIDITY (NON-CONDENSING)	40% to 90%	10% to 95%
ALTITUDE	2400 M (8000 ft.)	3000 M (10,000 ft.)

SYSTEM CABINET PERIPHERAL CONFIGURATIONS

Configuration Options	Slots Used	Max Allowed
COMMUNICATIONS SUBSYSTEM**	2	1
SYNCHRONOUS COMM. CTRLR**	1	1
ETHERNET CONTROLLER	1	1
SYSTEM PRINTER	1	1

Connections Provided	Type
ETHERNET	DA15
REMOTE DIAGNOSTIC PORT	DB25
MASTER SYSTEM CONSOLE	DB25
TERMINALS *	DB25 *

* See controller specification
** Maximum of either 8 async or 8 sync ports

Table 9-12: FX/1 I/O Expansion Cabinet Specifications

I/O EXPANSION CABINET PHYSICAL SPECIFICATIONS

Cabinet Dimensions

HEIGHT	72.4 CM (28.5 In.)
WIDTH	34.3 CM (13.5 In.)
DEPTH	63.5 CM (25.0 in.)

Service Access

FRONT	38 CM (15 in.)
REAR	92 CM (36 in.)

Weight

BUILDING BLOCK	31.8 KG (70 lbs.)
MAX CONFIG	68.2 KG (150 lbs.)

Heat Output

MAX CONFIG	725 Watts (3500 BTU/hr.)

Power	USA	INT**
VOLTAGE	120V	220/240V
TOLERANCE	-15% +10%	-15% +10%
PHASES	1	1
BRANCH CIRCUIT REQ.	15A	15A
CURRENT (AMPS)	12	10
FREQUENCY	60 ± 2Hz	50 ± 2Hz
CONNECTOR (NEMA)	5 - 15P	as required by country
CONNECTOR (Hubbell)*	5266	as required by country
RECEPTACLE (NEMA)	5-15R	
RECEPTACLE (Hubbell)	5262	
CABLE LENGTH	6 Feet	6 Feet
POWER AVAILABLE	1.5 KVA	1.5 KVA

*connector supplied
**For Japan consult customer representative

I/O EXPANSION CABINET CONFIGURATION

Winchester Disk Drive Expansion

268MB 8"	Two Maximum Per System Requires SMD Controller
138MB 5 1/4"	One Allowed in I/O Cab Two Maximum Per System

Multibus

ONE SIX-SLOT CHASSIS
REQUIRES INTERACTIVE PROCESSOR

Multibus Chassis Configuration Options	Slots Used	Max Allowed Chassis	Sys.
INTERACTIVE PROCESSOR *	1	1	2
COMM. SUBSYSTEM **	2	1	1
SYNCHRONOUS COMM. CTRLR**	1	1	2
ETHERNET	1	1	1
SYSTEM PRINTER	1	1	1
SMD DISK DRIVE	1	1	2

* Required to use expansion cabinet Multibus

Connections Provided	Type
ETHERNET	DA15
TERMINALS*	DB25

*See controller specification
** Maximum of either 8 async and 2 sync ports or - 8 sync ports

FX/1

5 1/4" Winchester Disk Specifications

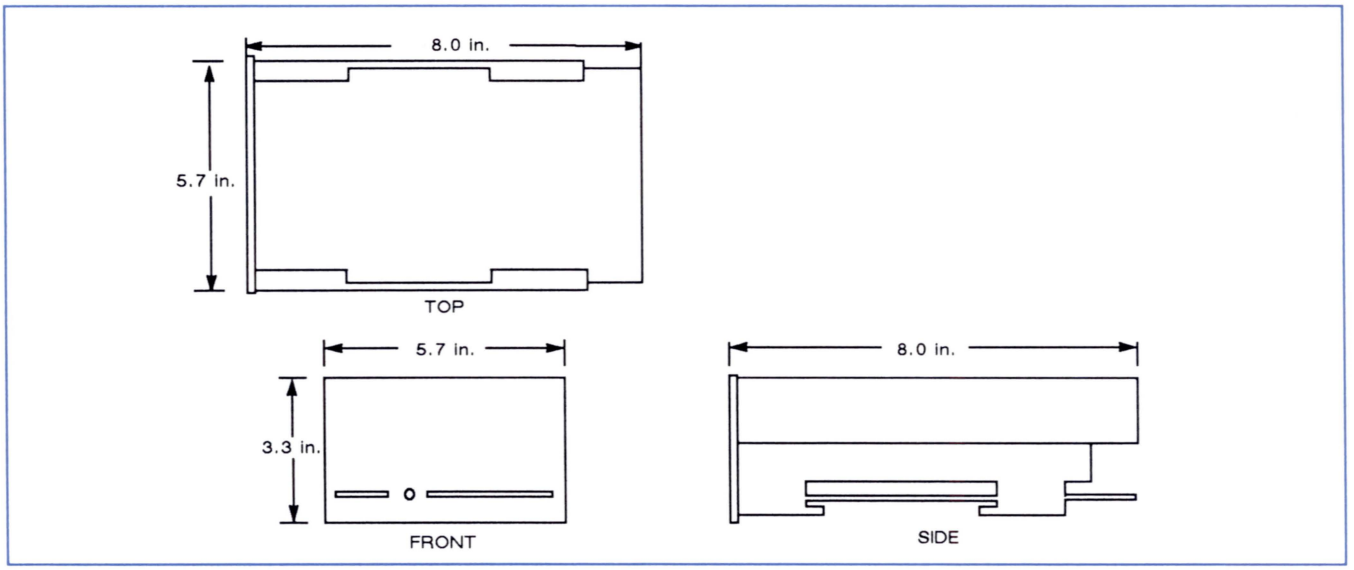


Table 9-13: FX/1 5 1/4" Winchester Disk Specifications

DISK FUNCTIONAL SPECIFICATIONS

Capacity	FORMATTED	134MB
	UNFORMATTED	172MB
Data Transfer Rate		1.25MB/sec.
Cylinders		823
Tracks Per Cylinder		10
Sectors Per Track		32
Bytes Per Sector		512
Rotation Speed		3600 RPM
Average Rotational Latency		8.3 MS
Positioning Time		
	TRACK TO TRACK	8 MS
	AVERAGE	25 MS
	MAXIMUM	50 MS
Track Density		850 TPI
Recording Density		20,400 BPI
Disk Reliability Specifications		
	MTBF (power on)	10,000
	MTTR	30 min.

DISK PHYSICAL SPECIFICATIONS

Dimensions

HEIGHT 8.4 CM (3.3 in.)
WIDTH 14.5 CM (5.7 in.)
DEPTH 20.3 CM (8.0 in.)

Weight 3 KG (6.0 lbs.)

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	5 to 45	-40 to 60
°F	41 to 113	-40 to 140
RELATIVE HUMIDITY (NON-CONDENSING)	20% to 80%	5% to 95%
ALTITUDE	3000 M (10,000 ft.)	12,000 M (40,000 ft.)

CONTROLLER SPECIFICATIONS

Drives Supported	2
Data Verification	32-bit ECC 11-bit Correction
Transfer Mode	DMA
FIFO Buffer	4KB
Interface Type	ESDI

FX/1

5 1/4" Cartridge Tape Specifications

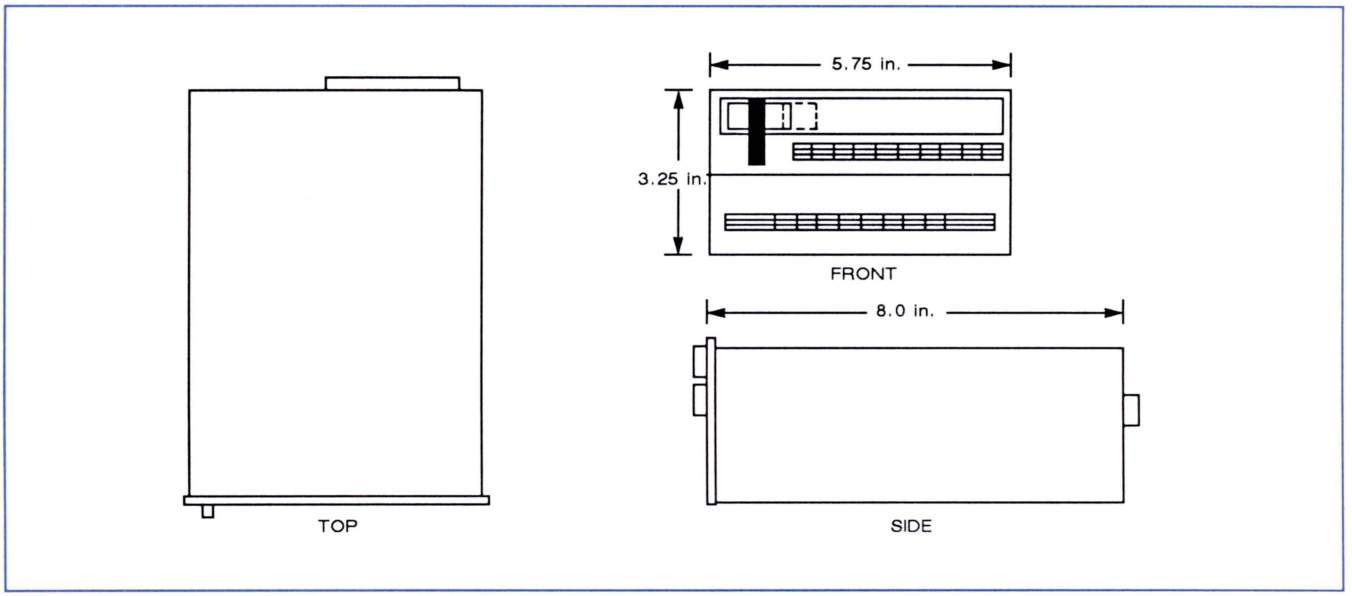


Table 9-14: FX/1 5 1/4" Cartridge Tape Specifications

FUNCTIONAL SPECIFICATIONS

Drive Performance

Capacity	45MB (DC300 XL/P or equivalent)
No. Recorded Tracks	9
Data Transfer Rate	85KB/sec. avg.
Tape Speed	
READ/WRITE	90 IPS
REWIND	90 IPS

PHYSICAL SPECIFICATIONS

Dimensions

HEIGHT	8.26 CM (3.25 in.)
WIDTH	14.6 CM (5.75 in.)
DEPTH	20.3 CM (8.0 in.)

Weight 2 KG (4.5 lbs.)

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	5 to 45	-30 to 60
°F	41 to 113	-22 to140
RELATIVE HUMIDITY (NON-CONDENSING)	20% to 80%	0% to 90%
ALTITUDE	-300 M (-1000 ft.)	4500 M (15,000 ft.)

Power

DC VOLTAGE	+24 VDC ± 5% @ 1.0 AMP (typical) +5 VDC ± 5% @ 1.5 AMP (max)
POWER DISSIPATION	19.5 Watts, Nominal 66 BTU/hr.

FX/1

8" Winchester Disk Specifications

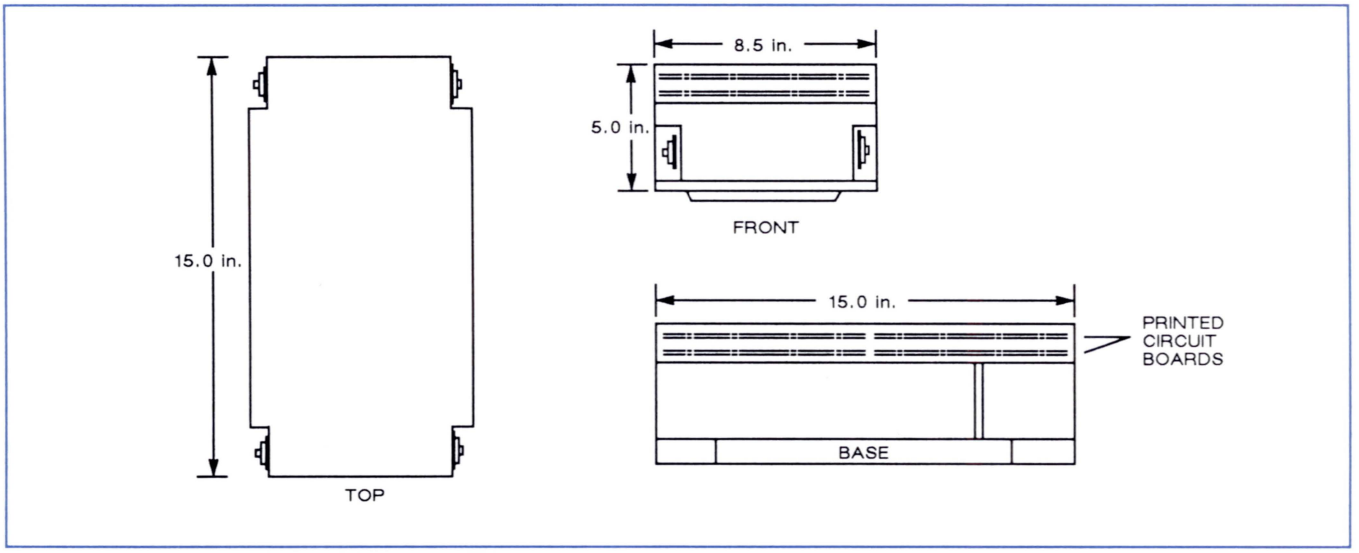


Table 9-15: FX/1 8" Winchester Disk Specifications

DISK FUNCTIONAL SPECIFICATIONS

Capacity	FORMATTED	268MB
	UNFORMATTED	337.1MB
Data Transfer Rate		2.46 MB/sec.
Cylinders		823
Tracks Per Cylinder		10
Sectors Per Track		64
Spare Sectors Per Track		4
Bytes Per Sector		512
Rotation Speed		3600 RPM
Average Rotational Latency		8.3 MS
Positioning Time		
	TRACK TO TRACK	5 MS
	AVERAGE	20 MS
	MAXIMUM	40 MS
Track Density		683 TPI
Recording Density		19,734 BPI
Disk Reliability Specifications		
	MTBF (power on)	10,000 hrs.
	MTTR	30 min.

DISK PHYSICAL SPECIFICATIONS

Dimensions

HEIGHT 12.7 CM (5.0 in.)
WIDTH 21.6 CM (8.5 in.)
DEPTH 38.1 CM (15.0 in.)

Weight 14 KG (31 lbs.)

Environmental	**Operating**	**Non-Operating**
TEMPERATURE °C	5 to 40	–40 to 60
°F	41 to 104	–40 to 140
RELATIVE HUMIDITY (NON-CONDENSING)	20% to 80%	5% to 95%
ALTITUDE	3000 M (10,000 ft.)	12,000 M (40,000 ft)

CONTROLLER SPECIFICATIONS

Drives Supported	2 (requires FX/1 expansion cabinet)
Data Verification	32-bit ECC 11-bit Correction
Transfer Mode	DMA
FIFO Buffer	8KB
Interface Type	SMD

System Peripherals

Specifications (Supplied With System Building Blocks)

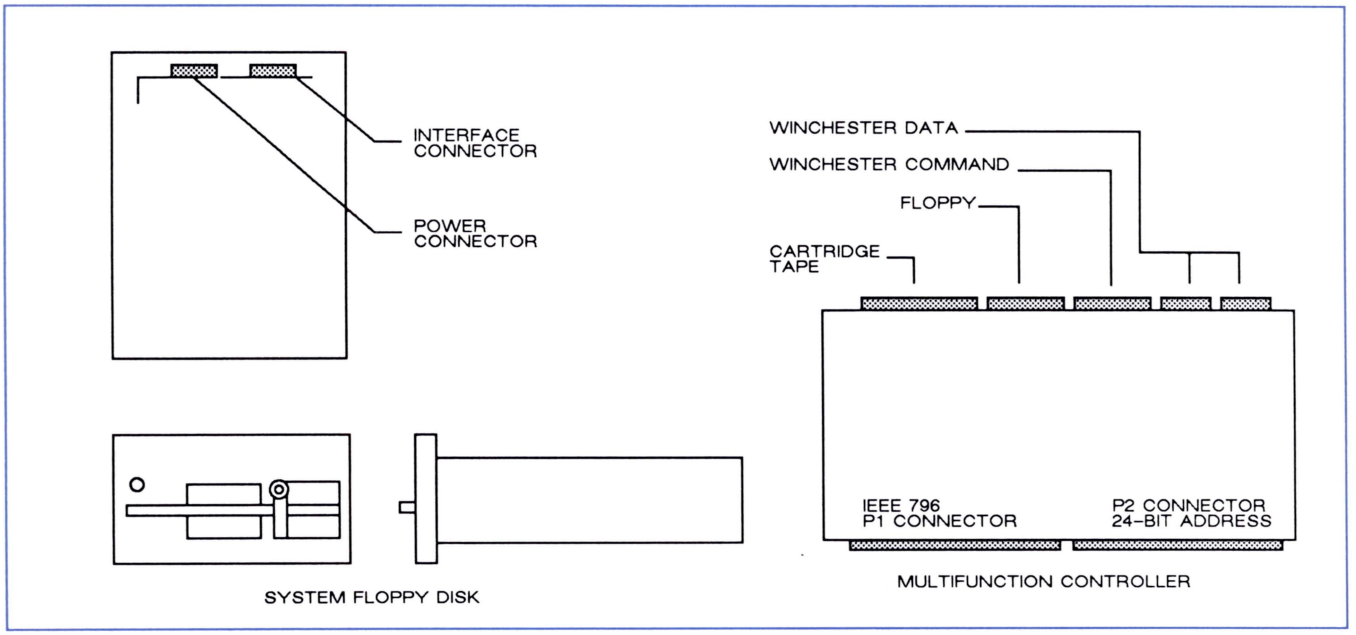


Table 9-16: System Peripherals Specifications

FLOPPY DISK DRIVE

System Application

MICROCODE LOAD
SYSTEM INITIALIZATION
SOFTWARE INSTALLATION
DIAGNOSTIC LOAD

Capacity (formatted)	655KB
Transfer Rate	31.25KB/sec.
Formatting	
SECTORS PER TRACK	8
BYTES PER SECTOR	512
Average Latency	100 MS
Positioning Time	
TRACK TO TRACK	3 MS
AVERAGE	94 MS
Reliability Specifications	
MTBF (POWER ON HOURS)	10,000
MTTR	30 min.

MULTIFUNCTION CONTROLLER

Bus Interface:	Multibus
Slots Required:	1 Multibus

Interface	**Standard**	**Max Devices FX/1**	**FX/8**
FLOPPY	SA460	1	1
WINCHESTER	ESDI	2	0
TAPE	QIC-02	1	0

Data Transfer Rates (max)

WINCHESTER	1.25MB/sec.
FLOPPY	32KB/sec.
TAPE	87KB/sec.

Error Detection/Correction

FLOPPY/TAPE	CRC Checksum
WINCHESTER	32-Bit ECC Polynomial

Formatting

WINCHESTER SECTOR SIZE	512 Bytes
SECTORS PER TRACK	32
FLOPPY SECTOR SIZE	512 Bytes
SECTORS PER TRACK	8

Synchronous Communications Controller

Specifications

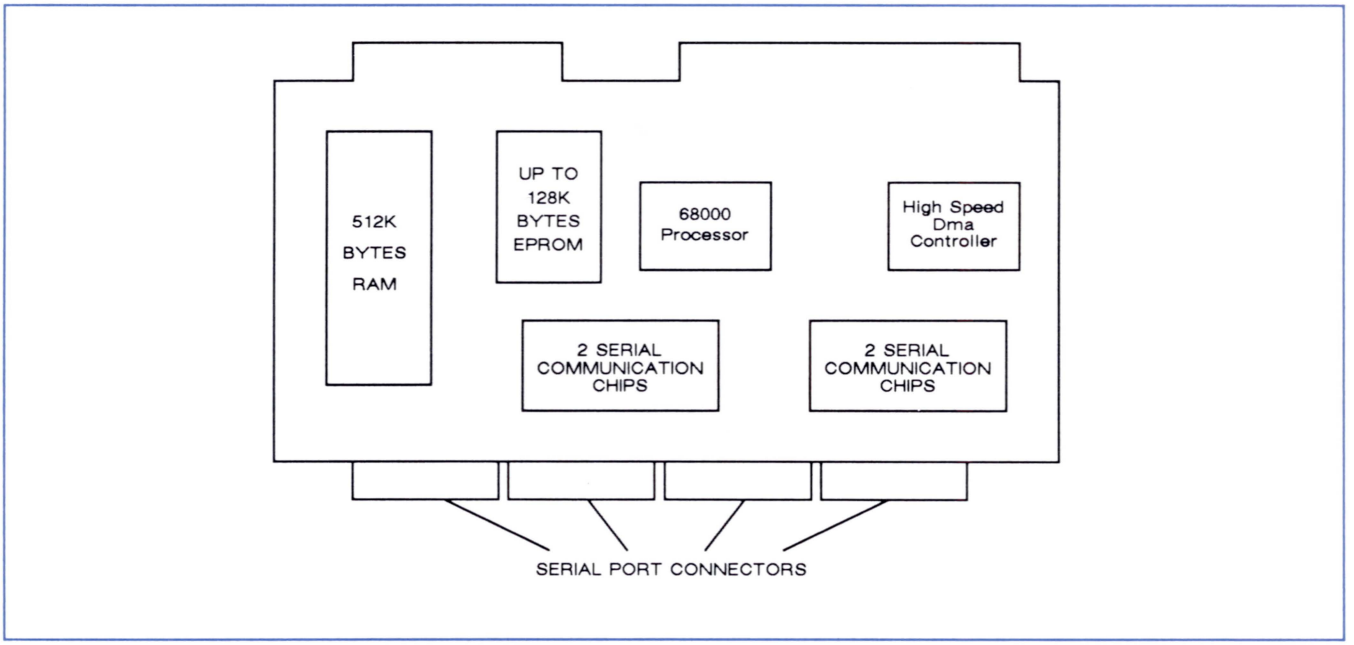


Table 9-17: System Multichannel Communications Controller Specifications

FUNCTIONAL SPECIFICATIONS

Protocol/Standards Supported

HASP, X.25, and IP-over-X.25
RS232, RS449 or V.35 (internally or externally clocked)

Architectural Features

HIGH SPEED 68000 PROTOCOL PROCESSOR
2 HIGH SPEED (64 BITS PER SECOND), DMA SUPPORTED SERIAL LINES
6 NON-DMA LINES (19.2K BITS PER SECOND)
ONBOARD HARDWARE DIAGNOSTICS
512K BYTES ONBOARD RAM
UP TO 128K BYTES ONBOARD EPROM
4 CHANNEL HIGH SPEED DMA CONTROLLER

Capacities

5 ACTIVE SERIAL LINES MAXIMUM PER BOARD
4 BOARDS MAXIMUM, PER SYSTEM
192K BIT PER SECOND AGGREGATE THROUGHPUT

PHYSICAL SPECIFICATIONS

Board Type Multibus

Slots Required 1 Multibus

Dimensions

HEIGHT 30.5 CM (6.7 in.)
WIDTH 17.2 CM (12.0 in.)

Weight 583GM (21 oz.)

Terminal Connections

INTERNAL CABLES Supplied

Serial Line Termination Daughter board for desired electrical interface. Affects availability of async ports.

Power Required

+5 Volts @ 4.5A, typical
+/- 12 Volts depending on serial interface requirements

System Multichannel Communications Controller

Specifications

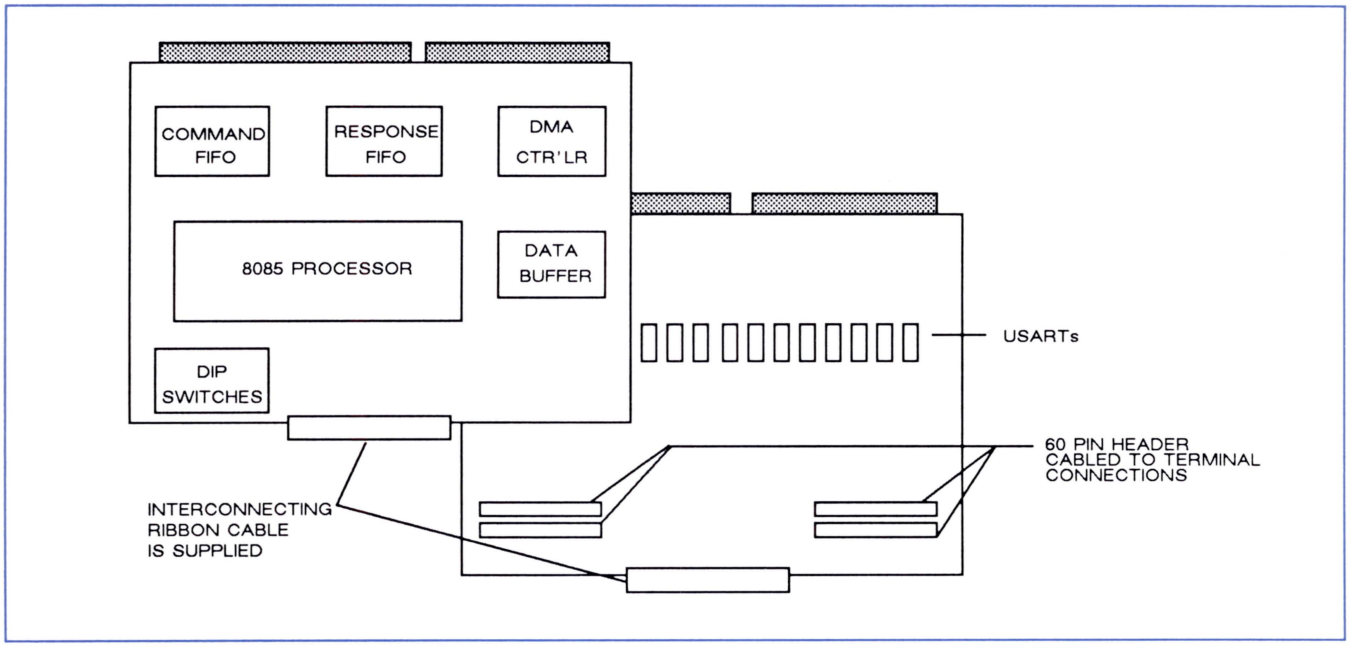


Table 9-18: System Multichannel Communications Controller Specifications

FUNCTIONAL SPECIFICATIONS

Architectural Features

DMA TRANSFER MODE
ON-BOARD MICROPROCESSOR
16-KBYTE BUFFER
SOFTWARE CONFIGURABLE BUFFER
ON-BOARD DIAGNOSTICS

Data Transfer Rates

9600 BAUD (16 lines, 100% load)
15 KBYTES PER SECOND AGGREGATE

Supported Baud Rates

50, 75, 110, 134.5, 150,
300, 600, 1200, 1800
2400, 4800, 9600, 19200

PHYSICAL SPECIFICATIONS

Board Type Multibus

Slots Required 2 Multibus

Terminal Connections

INTERNAL CABLES	Supplied
CONNECTOR TYPE	DB25 (male)

FX/1:
8 connectors available in System Cabinet
8 additional connectors available in Expansion Cabinet when ordered with Model M403 Controller
Connectors installed when Controller is ordered

FX/8:
32 connectors available per I/O Expansion or 50 IPS Tape Cabinet
96 maximum connectors per system
Connectors installed when Controller is ordered

Controller Availability

	16 Line	8 Line
FX/1	YES	YES
FX/8	YES	NO

Power Required	**16 Line**	**8 Line**
+5 Volts	4.7A	4.0A
+12 Volts	250 mA	250 mA
-12 Volts	200 mA	200 mA

System Printer

Specifications

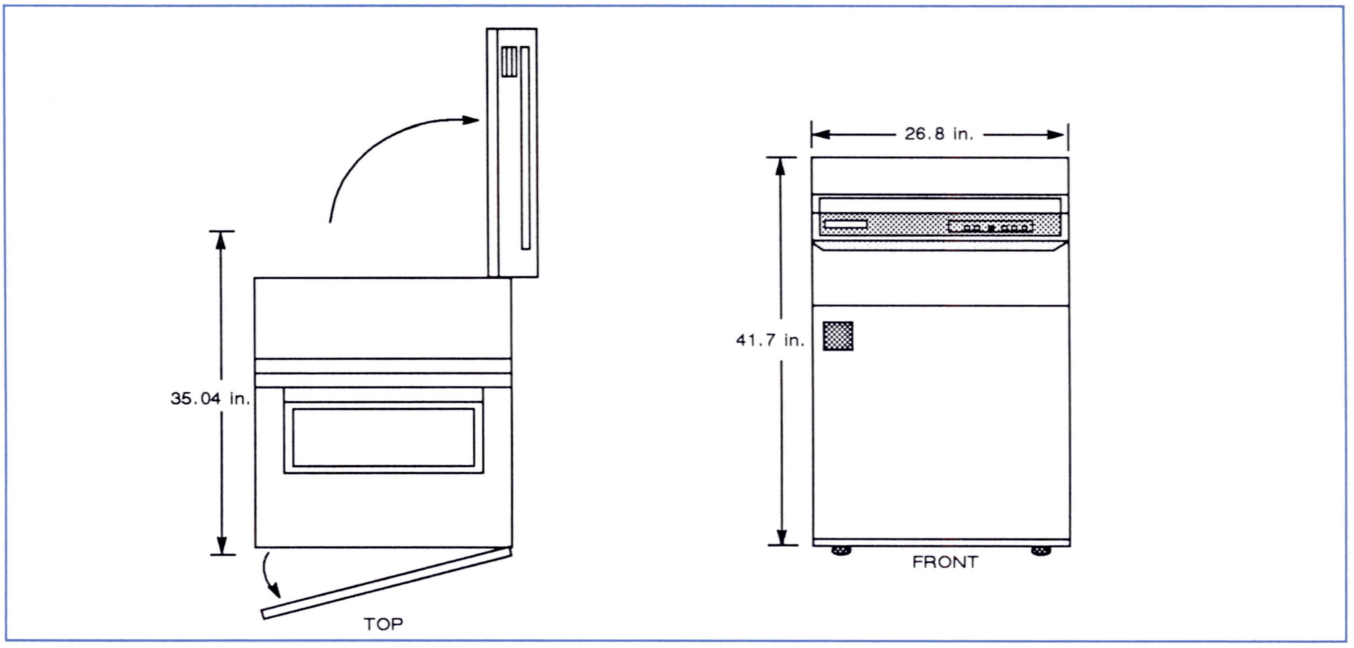


Table 9-19: System Printer

PERFORMANCE CHARACTERISTICS (600/1200)	
Printing Speed:	600/1200 LPM (64 character set) 420/910 LPM (96 character set)
Print Positions	132 OR 136 SELECTABLE
Print Format	
HORIZONTAL SPACING	2.54 MM (0.1 in.)
VERTICAL SPACING	4.23 MM (1/6 in.) OR 3.18 MM (1/8 in) switchable
Forms	
FORM FEEDING	18 MS (6 LPI) or 15.5 MS (8 LPI) for single line advance 324.2 MM/s (16.7 in/s) for continuous feeding
FORM WIDTH	76.2 MM (3 in.) to 431.8 MM (17 in.)
FORM LENGTH	76.2 MM (3 in.) to 381 MM (15 in.)
COPIES:	6 maximum (including original)
RIBBON:	30 MM x 100 M (1.2 in. x 328 ft.) continuous loop
Reliability Characteristics	
MTBF	6,000/4,000 hrs.
Interface	
DATAPRODUCTS, CENTRONICS	

PHYSICAL SPECIFICATIONS (600/1200)	
Dimensions	
HEIGHT	105.9 CM (41.7 in.)
WIDTH	68.1 CM (26.8 in.)
DEPTH	8.90 CM (39.5 in.)
Weight	130/140 KG (286/308 lbs.)
Power Requirements	
USA POWER SUPPLY	90 – 132 Volts 60 Hz
UNIVERSAL POWER SUPPLY (Optional)	90 – 132 Volts 180 – 250 Volts 50/60 Hz
CORD TYPES	Stripped end
POWER CONSUMPTION	600/1000 VA
ACOUSTIC NOISE	55 dBA (ECMA standard sound pressure level)

CONTROLLER SPECIFICATIONS	
Printers Supported	1
Transfer Rate	600KB
Board Type	MULTIBUS (one slot)

System Local Area Network (Ethernet) Controller

Specifications

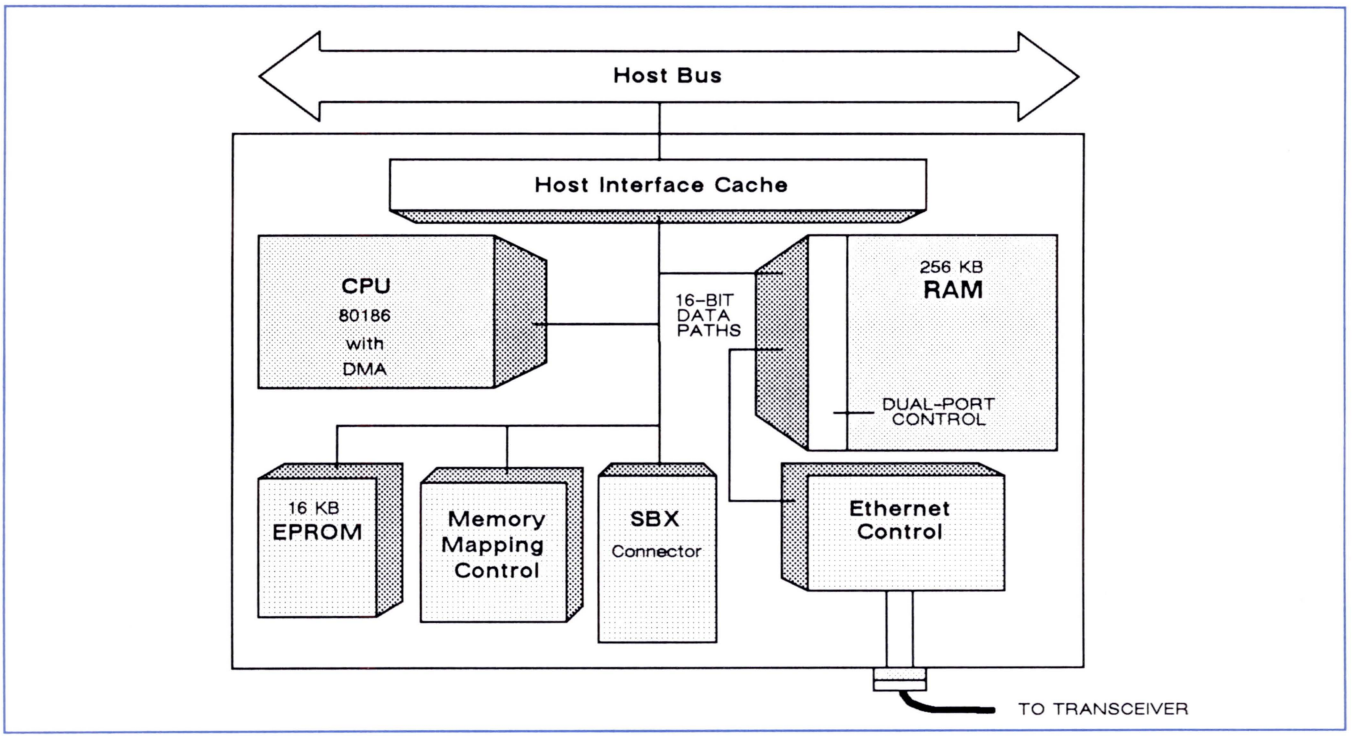


Table 9-20: System Local Area Network (Ethernet) Controller

SYSTEM ETHERNET CONTROLLER

CPU	8 MHz 80186
RAM	256KB
EPROM	16KB
Ram Dual-ported CPU/Network	Yes
Host Interface Cache	One Word
Multiple Segment Memory Mapping	Four independent segments 64-KB segments
Host Memory Address Support	20 or 24 Bits
Board Number	One
Board Size	Standard Multibus

Current (w/ max memory)	
@ +5V	4.75 A
@ +12V	0.5 A
Operating Environment	0.50C, 0-90% Hum. w/o Cond.
Standards Supported	Ethernet IEEE 802.3
Protocol Supported	TCP/IP DECnet
Transceiver Cable*	Supplied

* transceiver not supplied

Alliant Computer Systems Corporation
One Monarch Drive • Littleton • Massachusetts 01460
617-486-4950

ALLIANT

Computer Systems Corporation