



Fermi National Accelerator Laboratory

FERMILAB-Conf-88/165

A Floating Point Engine for Lattice Gauge Calculations*

D. Husby, R. Atac, A. Cook, J. Deppe,
M. Fischler, I. Gaines, T. Nash, T. Pham, and T. Zmuda
Advanced Computer Program
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

E. Eichten, G. Hockney, P. Mackenzie, H. B. Thacker, and D. Toussaint
Theoretical Physics Group
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

November 1988

*Talk presented at the 1988 IEEE Nuclear Science Symposium, Orlando, Florida, November 8-13, 1988.



A FLOATING POINT ENGINE FOR LATTICE GAUGE CALCULATIONS

D. Husby, R. Atac, A. Cook, J. Deppe, M. Fischler, I. Gaines, T. Nash, T. Pham, and T. Zmuda
Advanced Computer Program
Fermilab National Accelerator Laboratory
Batavia, IL 60510 USA

E. Eichten, G. Hockney, P. Mackenzie, H. B. Thacker, and D. Toussaint
Theoretical Physics Group
Fermilab National Accelerator Laboratory
Batavia, IL 60510 USA

The latest in low cost computing solutions from the Fermilab Advanced Computer Program is targeted at Lattice Gauge theory calculations and delivers supercomputer performance at a fraction of the cost. A typical system with 256 processors, 2.5 Gigabytes of memory, and 64 Gigabytes of on-line tape storage, delivers a peak performance of 5 billion floating point operations per second. The programming environment, Canopy, provides a comprehensive, hardware independent, distributed processing platform from within the more familiar environments of FORTRAN, C, and UNIX. This paper describes the individual processing elements of the system and gives a brief description of the Canopy software.

Introduction

Recent advances in computer technology have made it possible to build a processing engine that is sufficiently powerful, flexible, and inexpensive to be dedicated to solving certain problems in the field of Quantum Chromodynamics (QCD) theory. Several such machines have been / are being built, each of which uses a slightly different approach to solving the problem [1-5].

As the most recent entry to the race, the Fermilab Advanced Computer Program (ACP) Multiple Array Processor System (ACPMAPS) has been able to take advantage of some of the most recent developments in computer technology including:

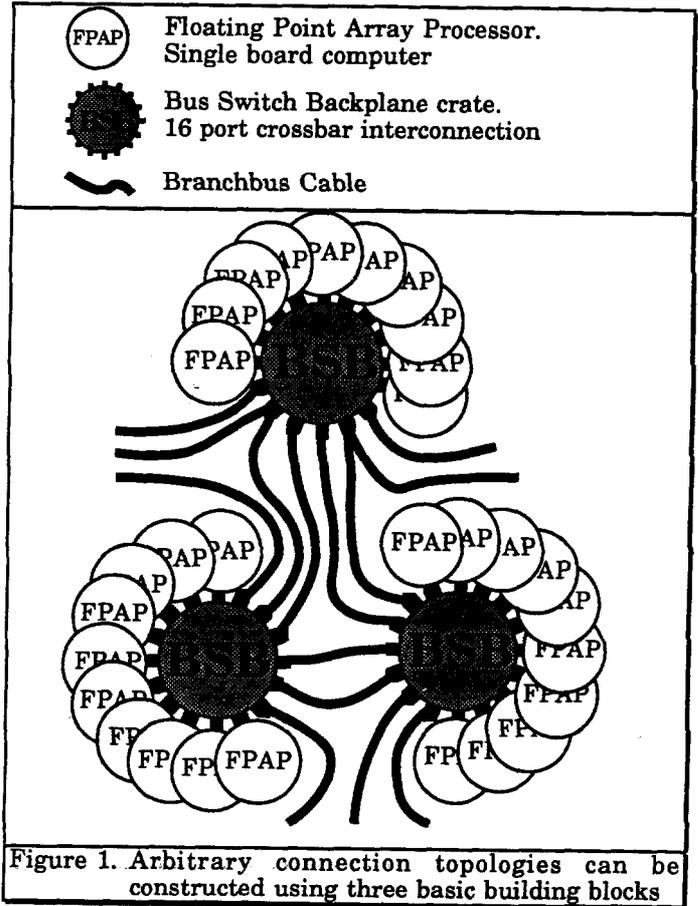
- 1) Fast floating point processors from Weitek, Inc. that are supported with high level language compilers.
- 2) Large, fast dynamic memory chips.
- 3) High speed communication chips from Texas Instruments that implement a full 16 port crossbar switch.
- 4) High density, low cost magnetic tape drives from Exabyte, Inc. that allow up to 2 Gigabytes of data to be stored on an 8mm video cassette cartridge.

In addition, the ACPMAPS approach is somewhat unique in that one of its goals is to create a productive environment for the development of new algorithms.

System Overview

As shown in figure 1, a system is built from three basic building blocks:

- 1) Processors, called FPAP's (Floating Point Array Processor), which have a peak performance of 20 Megaflops, a data memory capacity of 8 Megabytes, an instruction memory capacity of 2 Megabytes, and can be plugged into...



- ... 2) Interconnection crates, called BSB's (Bus Switch Backplane), which have 16 ports (slots) and support an aggregate data rate of 160 Megabytes per second or 20 megabytes per second per channel. Each of the 16 ports is identical and can accept either an FPAP or...
- ... 3) A connection to another BSB crate, called a Branchbus [6], which has a peak data rate of 20 Megabytes per second. The Branchbus is a block transfer bus that allows up to 16 devices to communicate across twisted pair, 50 signal, ribbon cable.

In principle, these building blocks can be assembled to create arbitrarily complex interconnection networks with as many as 2048 FPAP's. The ACP will support standard configurations of 15 to as many as 256 FPAP's.

This paper focuses primarily upon describing the FPAP and supporting software. The BSB and Branchbus have applications in other ACP systems and are described in a companion paper [7].

The FPAP Hardware

As shown in figure 2, the FPAP contains four major subsystems:

Microprocessor

The processing power of the FPAP comes from the Weitek XL microprocessor. The XL processor consists of three separate chips to handle instruction sequencing, integer processing, and floating point operations. The processor is microcoded and pipelined so that on each 100 nanosecond cycle it can generate an instruction address, generate a data memory address, transfer a piece of data from memory, and do two floating point operations with corresponding register operations.

The 64-bit wide microcode is very well designed and provides many complex instructions (though it is called a RISC processor), all of which execute in 100 nanoseconds. The instruction set includes: very low overhead branching, looping, and subroutine calls, indexed memory addressing with pre and post increments, 3-address arithmetic and logical operations, and bit field manipulations.

Instruction Memory

The instruction memory uses page-mode dynamic RAM's to deliver a 64-bit instruction on almost every 100ns cycle. An extra two cycles are needed to fetch an instruction if it is not in the same 512 word page as the previous instruction. Since program execution is sequential and localized, the 200ns penalty will occur on less than 0.5% of instruction fetches.

Data Memory

The data memory also uses page-mode DRAM's and is capable of delivering a 32-bit word every 100ns with an additional 2 cycle penalty for crossing a 1024 word page boundary. Since data accesses are not inherently local, a second access mode is available. This non page-mode access takes 2 machine cycles, however there is no penalty for crossing a page boundary. The programmer can switch between modes by accessing a control register.

BSB Interface

The FPAP communicates to the outside world via its BSB interface. BSB protocol is a fairly simple block transfer protocol. Each message consists of a two word header followed by as many as 64K data words. Byte parity and handshake signals are used to insure data integrity.

The BSB interface does many operations automatically, including opening a channel to a destination, retrying a failed open request, and notifying the microprocessor via interrupts when an error occurs.

As a master, the CPU controls BSB operations via a set of control and status registers (see table 1). The CPU is also responsible for generating the memory addresses that are used to transfer data between the memory and BSB. This allows the CPU to scatter or gather blocks of data from non-sequential locations in memory as the transfer is in progress.

As a slave, the BSB interface requires no CPU intervention unless an error occurs. Data is transferred to sequential memory addresses starting at the address specified in the second word of the block header. Bits in the second header word can also halt, start, or interrupt the CPU.

TABLE 1. BSB CONTROL AND STATUS REGISTERS

REG.	FUNCTION
OPEN	Open a channel to a slave node. This is the first of two header words and contains an 11-bit destination node address and a 16-bit transfer count.
ADR_RD	Send a 32-bit memory address to the slave and prepare to receive data from the slave.
ADR_WR	Send an address and prepare to send data.
REOPEN	Terminate a transfer and start a new one.
CLOSE	Terminate a transfer and close the channel.
STATUS	Read the status of the channel.
CLEAR	Clear errors.

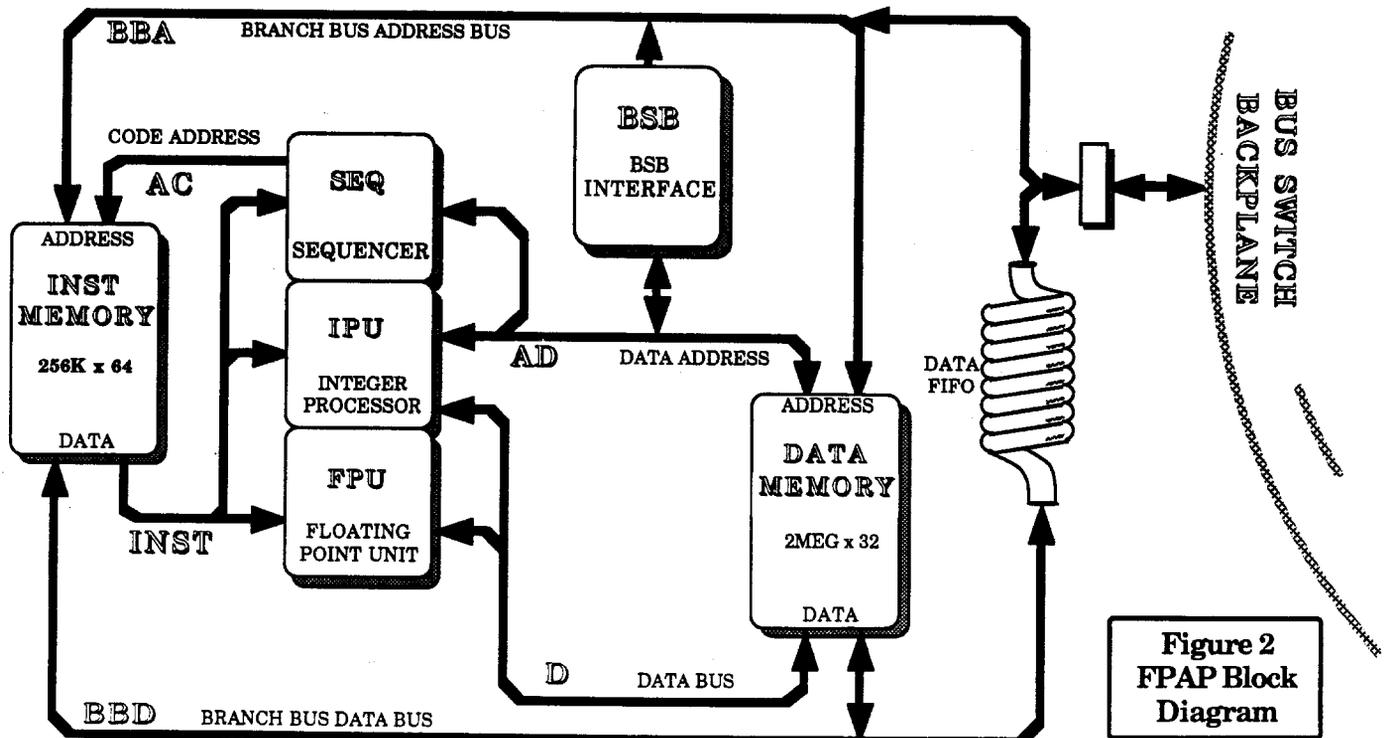


Figure 2
FPAP Block
Diagram

CANOPY

The Canopy environment is designed to aid in the development of lattice and grid oriented programs. To the user, it appears as a set of subroutines and pre-processor directives that can be called from within a FORTRAN or C program. Internally, Canopy maintains the structures, pointers, and communication routines necessary to distribute a lattice across multiple processors.

A lattice, in the context of Canopy, is a set of interconnected sites (see figure 3). The structure of Canopy permits arbitrary connectivity, however currently available routines support hypercube lattices of arbitrary dimension and size.

A site is a point on a lattice. Each site contains a number of user defined data structures called fields. Sites also contain system information such as pointers to neighboring sites and synchronization data.

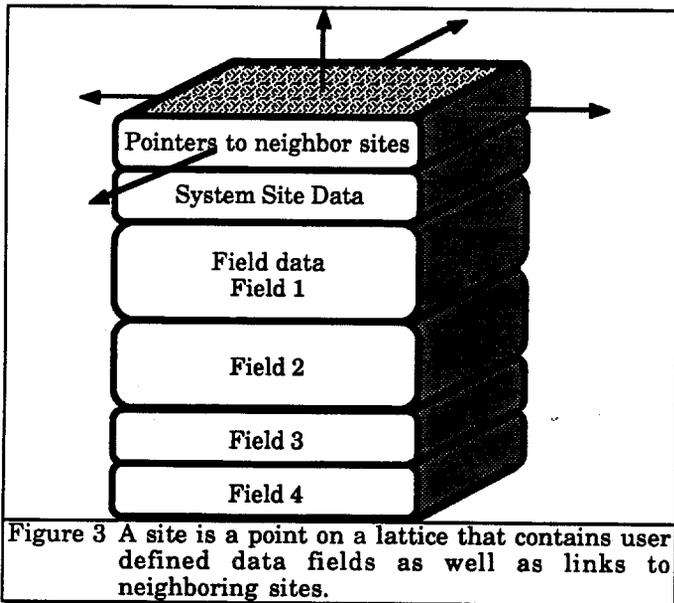


Figure 3 A site is a point on a lattice that contains user defined data fields as well as links to neighboring sites.

Canopy provides routines to support most aspects of lattice gauge calculations. Some representative routines are described here.

Describing a lattice:

Several tools facilitate the definition of a lattice. In addition to describing lattices and fields, there are functions to describe sets of sites, directions, paths, and fields that reside on links between sites (`link_fields`). Some of the basic functions have the form:

```
lattice_id =          /* Identifier of new lattice */
define_periodic_lattice(
  ndims,              /* Number of dimensions */
  sizes[ndims] )     /* Array containing the size of */
                    /* each dimension */

field_id =           /* Identifier of new field */
define_site_field(
  lattice_id,        /* Id of lattice that contains field */
  size )            /* Number of bytes in field */

complete_definitions
/* End of definitions, distribute the defined */
/* lattices among the available processors */
```

Processing

Canopy allows processing to be distributed among multiple processors. The primary facility for distributed processing, `do_task`, is a remote subroutine call of the form:

```
do_task( subroutine, set_of_sites, parameter_list)
```

`Do_task` will cause `subroutine` to be executed for each site in `set_of_sites`. Execution of the subroutine takes place in parallel, with each processor processing the sites that belong to it. The subroutine is a standard C or FORTRAN subroutine and is passed the parameters that are specified in `parameter_list`.

Accessing Data

Data access routines in Canopy deal primarily with fields and pointers to fields. Fields can be accessed by specifying a site address and field identifier. A site address can be specified absolutely (coordinates on a lattice), or in some direction or path relative to another site. The two basic data access functions are:

```
field_pointer(field_id, site_address)
/* Return a pointer to the field. If the field is on a */
/* remote node, then first transfer the field to a */
/* temporary local memory location. */

put_field(field_id, site_address, *source)
/* Copy the data pointed to by source to the */
/* addressed field. If the field is on a remote */
/* node, transfer it. */
```

There are many variations on these routines that incorporate various site addressing modes, operate on `link_fields`, and synchronize access to fields.

File System

Canopy supports most UNIX operating system calls and thus can support file and terminal I/O using standard C libraries. UNIX calls are channeled from FPAPs to a host MicroVax/ULTRIX system via a Branchbus connection.

In addition to UNIX I/O, Canopy can store entire fields (i.e. field data from each site on a lattice) on a distributed tape system. The tape system uses multiple Exabyte tape drives connected via Branchbus. Each tape drive has a formatted storage capacity of 2 Gigabytes.

Canopy provides subroutines to create tape sets, create named field files, and store field data.

Other Utilities

Random number generation poses problems that are unique to multiple processor systems. It is desirable to allow each processor to generate its own random numbers, however it is also desirable to be able to reproduce results, even when a job is run on different hardware configurations. Toward that end, Canopy supports the concept of multiple random number streams. For reproducible results, a program can have a separate random number stream for each site.

A library of optimized math routines is provided. The library includes a variety of SU(3) operations as well as transcendental functions.

Advanced users have access to low-level routines and data structures. Documentation is provided to allow programmers direct access to internode communications, lattice distribution, resource allocation, and remote processing functions.

ACPMAPS System Design

The ACPMAPS hardware and software were designed as a balance of several forces: Special purpose versus general purpose, efficiency versus flexibility, burden on the user versus burden on the system programmer.

While a special purpose system can squeeze out the most performance for a particular problem, it is not necessarily easily adapted to solve other problems efficiently. In addition, special purpose hardware places a burden upon the hardware designer and end user: The designer must know many aspects of the problem to be solved while the application programmer must learn hardware specifics including communication details and perhaps even microcode. It also places a burden upon the system programmer to develop tools such as hardware specific assemblers and compilers.

On the other hand, general purpose systems are ideal for efficient development of new software, yet they place a great deal of burden upon the system programmer to develop parallelizing compilers and operating systems. Such systems require more resources than are usually available to non-profit oriented projects.

The ACPMAPS project has taken an approach that is intended to optimize overall productivity. It is specific enough that system software implementation is fairly straightforward. Much of the system software already exists as products supported by the microprocessor manufacturer: compilers, assemblers, linkers, debuggers and simulators. It is general enough to allow a user to bring up applications quickly while minimizing the number of new concepts which must be learned.

One might expect to pay for this middle of the road approach through a loss of performance. The cost, however, is not as great as might be expected. Again, the middle of the road offers some advantages from both worlds. Programs can be developed and tested in a high-level language environment. Then, when a user is satisfied that he has chosen the right algorithm and that it is working, he can concentrate upon optimizing and hand-coding selected parts of his program. For programs that are dominated by compute intensive inner loops, this tactic approaches the performance of full hand-coding but with much less effort.

Results

The validity of the ACPMAPS approach has already been proven to a certain extent. System software development for the project has taken approximately a year, requiring 30 theorist/programmer-months of effort. In that year, the Canopy software has been ported to several systems including VAX/ULTRIX, VAX/VMS, MIPS workstation, IBM PC/AT, and of course, the multi-FPAP prototype system. Two application programs have been brought up on the system with less than 1 month of work. Other application programs are currently in development.

A prototype hardware system consisting of 16 FPAPs, 2 BSB crates, and 4 tape drives has been working since September, 1988. According to benchmark results from two different programs, the system is delivering the equivalent processing power of a Cray X-MP processor. The system is currently being used by theorists for program development.

The parts cost of the prototype system was approximately \$100,000. About 60 percent of that cost is

due to the cost of memory chips which is expected to drop by about 50 percent in the next year.

Development time from concept to prototype was about 21 months. Hardware development required 45 engineer-months and 12 technician-months of work.

Future Development

A production system consisting of 256 FPAPs, 32 BSB crates, and 32 tape drives is currently being constructed and, depending upon the availability of memory, is expected to be finished by mid 1989.

Work is also being done on a "Turnkey" system package. The turnkey system will consist of 15 FPAPs in a single BSB crate with an integrated host processor, disk, and tape drives.

Both systems will become standard ACP products which means that the systems will soon be available from a commercial manufacturer.

Of course, there are plans on the back burner to build bigger and more powerful systems. Branchbus will likely be replaced with faster, smaller optical fibers. Future FPAPs will use faster microprocessors and more memory. The modular design of the ACPMAPS hardware and software can easily accommodate new technology as it becomes available.

References

- [1-4] Papers presented at the 1988 Symposium on Lattice Field Theory, Fermilab, Batavia, Illinois, September 22-25, 1988
- [1] F. Butler, "Present Status of the Columbia Parallel Supercomputer Project",
- [2] E. Remiddi, "APE100: Project for a 100 Gigaflop Supercomputer"
- [3] J. Sexton, "Status of GF11"
- [4] J. Richardson, "The Connection Machine"
- [5] F. Brown and N. Christ, "Parallel Supercomputers for Lattice Gauge Theory", *Science*, vol. 239, pp. 1393-1400, March, 1988
- [6] R. Hance, "The ACP Branchbus and Real Time Applications of the ACP Multiprocessor System", *IEEE Transactions on Nuclear Science*, Vol. NS-34, No.4, pp. 878-883, August, 1987
- [7] R. Atac, "Crossbar Switch Backplane and its Applications", presented at the IEEE 1988 Nuclear Science Symposium, Orlando, FL, Nov. 9-11, 1988
- [8] E. T. Nash, "High Performance Parallel Computers for Science", presented at the Workshop on Computational Atomic and Nuclear Physics at One Gigaflop, Oak Ridge, TN, April 14-16, 1988