Fermi National Accelerator Laboratory

FERMILAB-Conf-89/37

The Fermilab Lattice Supercomputer Project*

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

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

February 1989

*Talk presented by M. Fischler and G. Hockney at "Lattice Gauge Theory '88," Fermilab, Batavia, Illinois, September 22-25, 1988.

Operated by Universities Research Association Inc. under contract with the United States Department of Energy

THE FERMILAB LATTICE SUPERCOMPUTER PROJECT*

Mark FISCHLER, R. ATAC, A. COOK, J. DEPPE, I. GAINES, D. HUSBY,
T. NASH, T. PHAM, and T. ZMUDA
Advanced Computer Program
Fermi National Accelerator Laboratory, Batavia, IL 60510 USA†

George HOCKNEY, E. EICHTEN, P. MACKENZIE, H. B. THACKER,
and D. TOUSSAINT‡
Theoretical Physics Group
Fermi National Accelerator Laboratory, Batavia, IL 60510 USA†

ABSTRACT

The ACPMAPS system is a highly cost effective, local memory MIMD computer targeted at algorithm development and production running for gauge theory on the lattice. The machine consists of a compound hypercube of crates, each of which is a full crossbar switch containing several processors. The processing nodes are single board array processors based on the Weitek XL chip set, each with a peak power of 20 MFLOPS and supported by 8 MBytes of data memory. The system currently being assembled has a peak power of 5 GFLOPS, delivering performance at approximately $250/MFLOP. The system is programmable in C and Fortran. An underpinning of software routines (CANOPY) provides an easy and natural way of coding lattice problems, such that the details of parallelism, and communication and system architecture are transparent to the user. CANOPY can easily be ported to any single CPU or MIMD system which supports C, and allows the coding of typical applications with very little effort.

INTRODUCTION

There is a clear need in the lattice gauge community for an accessible (meaning easily programmed and inexpensive) supercomputer capable of doing large scale computations. Such a machine is especially important for lattice algorithm development where there may yet be orders of magnitude to be gained in improvement of methods. Such an algorithm investigation machine requires several features: It must be powerful enough to try things out on the large lattices where the physics becomes relevant, yet still remain accessible to a broad community of physicists. It should be flexible, both in the nature of interprocessor communications, and in the availability of multiple instruction streams (MIMD). There need

---

* Talk presented by M. Fischler and G. Hockney at "Lattice GaugeTheory '88", Fermilab, September 22-25, 1988

† Fermilab is operated by Universities Research Association, Inc. under contract with the U. S. Department of Energy

‡ Permanent Address: Dept. of Physics, University of Arizona

to be large I/O facilities for archiving intermediate results. The most important feature, however, is that it must be easy to create and modify programs to run on the system.

The ACPMAPS machine, running CANOPY software, addresses both these needs. The system is among the most powerful dedicated lattice gauge machines, with 5 GFLOPS of peak power, 2 GBytes of memory, and 64 Gbyte tape volumes (these figures can be expanded by another order of magnitude with no redesign). It is an MIMD machine, with transparent global communication. The processors are programmable in C and Fortran; the CANOPY software allows the user to be ignorant of the details of the hardware architecture, programming with concepts which are natural to grid-oriented problems.

For example, it took one man-week to code and debug a Kennedy-Pendleton Monte-Carlo configuration generator, which runs with a link update time of .63 msec/link per node. This timing compares favorably with results on systems with similar node power[1,2,3]. Because the coding is modular and not tricky, effort can be focused on things that will boost performance without undue worry about "breaking a working program". Modifications to the code may take from seconds (for instance, changing lattice size or boundary conditions, or size of the machine) to hours (for a completely new heat-bath method or inserting over-relaxation steps). Thus the time between conceiving a new idea and obtaining early physical results, which can currently be months of coding and running, is reduced to days. This more closely matches the time scale on which one comes up with new ideas to try.

ACPMAPS

ACPMAPS (Advanced Computer Program Multiple Array Processor System) is composed of two major components. The backbone of the system is a set of 16 by 16 programmable crossbar switches, implemented as the backplanes of crates: The Bus Switch Backplanes (BSB). Occupying slots of these BSB crates are the Floating Point Array Processor (FPAP) nodes. In addition, there are three minor components: Branchbus Switch Interface Boards (BSIB) connecting BSB switches, a host running Unix with connections to the outside world, and a high-volume distributed tape I/O system. The system is, by design, reconfigurable and expandable; we are currently running on a small 2-crate development system, while a 32-crate, 256-node production system is being assembled.

The BSB is a Eurocard format crate with a sophisticated backplane. (Figure 1) The BSB backplane consists of a 16-port crossbar switch. The key components of this board are 13 TI SN74AS8840 4-channel 16-port crossbar chips, and a PROM containing routing information. The routing information allows the construction of machines composed of many BSB crates. A device in any crate can communicate transparently with one in any other crate, simply by requesting a channel to the appropriate device number. Each channel of communication can proceed at 20 MBytes/sec, and since eight such connections can exist on one crate backplane at a time, the aggregate communications bandwidth is 160 MByte/sec. The time required to reconfigure a switch is roughly half a microsecond, and reconfiguring

does not affect communications along any channel other than the ones being opened or closed. A system may comprise up to 2048 devices.

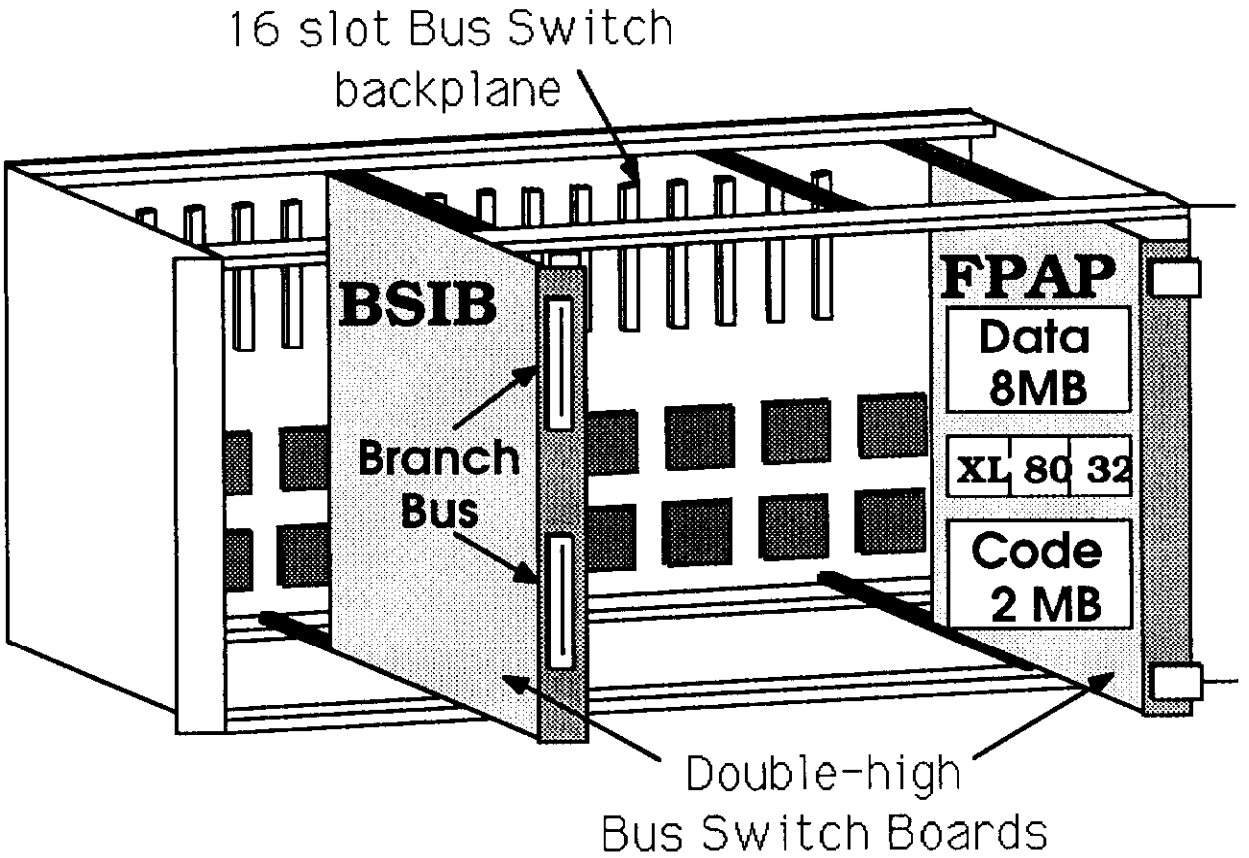

FIGURE 1
The BSB crate, shown with one FPAP node and one BSIB crate interface module. A typical system configuration includes at least 8 FPAP's and up to 8 BSIB's per crate. The 256 node system will comprise 32 BSB crates.

The FPAP is a single board computer based on the Weitek XL-8032 chip set which plugs into the BSB. This chip set includes an integer processor/sequencer which runs at 10 MIPS, and a floating point chip with multiply/accumulate architecture running at 20 MFLOPS peak. There are 32 integer and 32 floating point registers. The memory system is "Harvard architecture": 8 Megabytes of data memory and a separate 2 Megabyte instruction memory. (Thus a 256 node system can easily do physics on a 32^4 lattice.) The memory is fast page mode DRAM; unless a 4K page boundary is missed, one 32-bit word can be accessed every 100 nsec cycle. This architecture is rather well suited to the computations needed in lattice gauge; for example, a routine to do a single SU3 multiplication runs at 15 MFLOPS including calling, starting, pipeline and ending overheads. C and Fortran compilers exist for this chip set, although for optimal performance some crucial kernel subroutines may need to be written in assembly language. Typical applications run at about 6-8 MFLOPS per node.

The tape I/O subsystem consists of helical scan 8mm tape devices (Exabyte tape drives). The 256-node system being assembled will have 32 such drives, controllable

from any node as well as the host. The aggregate bandwidth to tape is 8 Mbytes/sec, while the size of a 32-tape "volume" is 64 Gigabytes -- sufficient to store many large configurations and propagators.

This production system is configured as a 2^5 hypercube of BSB crates, communicating by ACP Branchbus cables to BSIB modules in slots in the crates. Each BSB contains eight FPAP modules; the machine can be thought of as a compound hypercube of nodes, with each point of the hypercube being a crossbar of 8 FPAP's. Alternatively, the system may be viewed as a true 2^5 hypercube of 160 MFLOP, 64 MByte multiprocessor crates, with 20 MByte/sec intercrate bandwidth. The intercrate communications bandwidth is about twice what is needed for the most communication intensive parts of lattice gauge algorithms, even assuming perfect computational efficiency. All ACPMAPS modules will be made commercially available, allowing a system to be configured to the needs of the user.

In order to enhace the availability of such systems, a relatively small packaged "turnkey" ACPMAPS system is being designed. This one-crate system will deliver the power of 1-3 Cray XMP's for about $100,000 -- it is suitable for use as a "physics department supercomputer". The packaged system includes one BSB crate with fifteen FPAP nodes and two Exabyte tape drives. This will allow small groups to acquire a powerful platform for the study of algorithms, capable of running all but the most extensive production programs. Of course, code for this machine can be ported unchanged to larger ACPMAPS systems.

CANOPY

The CANOPY environment is software designed for development of grid-oriented algorithms. To the user, it appears as a set of subroutines which take care of the grid topology and structures in a natural way. On multiprocessor systems, CANOPY distributes grid tasks and handles internode communication transparently. The same programs work on any number of processors without the user needing to consider these details.

The three key concepts in CANOPY are: the **grid** -- a collection of sites with defined connectivity specifying which sites are "neighbors" in which directions; the **field** -- a set of data structures, one copy of which "belongs" to each site in a grid; and the **task** -- a routine to be run for each site, manipulating the fields associated with that site. CANOPY automatically manages parallelism by distributing sites over processors.

CANOPY programs are divided into three pieces: 1) the definitions section, where grids, fields and sets of sites are defined; 2) the main program or control section, which is an ordinary single-thread program which calls routines in 3) the task routines, which are executed in a parallel fashion and are where most of the actual work is done.

The definitions section is actually the first part of the main or control program, where subroutines defining grids, fields over the grids, sets of sites, grid

connectivities, link fields, and paths in the grid are defined. Two important routines in this group are:

```
lattice_id=define_periodic_grid (Ndims, sizes)
```

(sizes *is an array of length* Ndims *containing the extent in each dimension)* and

```
field_id = define_site_field (lattice_id, size)
```

After all the defining subroutine calls have been made, a call to a special complete_definitions subroutine is made. This subroutine sets up the internal CANOPY structures needed and automatically decides how to distribute the problem in a parallel environment.

The control section does the small amount of work that cannot be associated with a set of sites on the lattice (such as inputting parameters and printing results). The control program also starts and coordinates the parallel tasks by calling

```
do_task (task_subroutine,
         set_of_sites,
         list _of_arguments_for_subroutine)
```

which will cause task_subroutine to be executed for each member of the set of sites in a distributed way.

The task routines, which do most of the work, call CANOPY routines to access the field and connectivity information in a natural way. The site for which the task routine is currently running is called the HOME site. Typically, CANOPY calls are made to access fields from the HOME site and neighbors to the HOME site identified by their direction. Some typical routines are:

field_pointer_at_dir (field_id, direction)
Return a pointer to the copy of the specified field data. If the site is assigned to a remote CPU, the data is transferred to a temporary local memory location, and a pointer to that location is returned.

put_field (field_id, site_address, *source)
Copy the data pointed to by source *to a specified field at the site.*

In addition, specialized routines have been created for random number generation and common math operations. Random number generation poses unique problems on multiprocessor systems since it is desirable to be able to reproduce results, even when the job is run on different hardware configurations. With a single subroutine change CANOPY can switch between stream-per-site random number generation, which produces the same result when run on any number of nodes, and stream-per-node random number generation, which is more efficient.

Mathematical kernel routines optimized for the particular hardware being used are provided in libraries. These routines have the same interface on all machines and provide functions such as SU3 multiply and accumulate, quark propagation

through gauge link fields, and dot product routines. These routines are not, strictly speaking, part of CANOPY, but are important for the development and testing of efficient algorithms.

An important consequence of using CANOPY is that the style of coding is guided into being structured and modular. The benefits of this range from more readable code, through easier code modifications and debugging, to the ability to confidently optimize critical sections of the code. Although these coding practices will be beneficial in any context, here there is a particular gain because the concepts of CANOPY structuring are a good match to the natural concepts in the problem. Thus it is usually quicker to bring up an application in the structured fashion using this framework, then it would be to do it in an ad hoc manner. The overhead associated with CANOPY ranges from 0 to 15% in typical cases; this is more than made up for by the reduced debugging time and the ability to do modular optimizations.

CANOPY is written in C, and is easily portable to any single-CPU or MIMD multi-processing system with support for UNIX calls. Thus one can try out routines on small lattices on a much less powerful system, and migrate to the production machine without changing any code. In fact, one can run CANOPY in a multi-process fashion even on single-CPU systems, to verify that the program is not dependant on the single stream nature of the computer. To date, the software has been ported to the ACPMAPS system, and to an Ultrix Microvax, a MIPS M500 system, a Sun workstation and an IBM PC running Turbo C.

SUMMARY

The ACPMAPS system is a highly cost effective large scale machine designed for lattice gauge calculations and algorithm development. A 5 GFLOP system is currently being assembled, at a cost of about one million dollars. The CANOPY software provides an easy and natural means of programming grid-oriented problems for such a system, substantially reducing the interval between the conception of an idea and early physics results.

References

1. Norman Christ, in: Field Theory on the Lattice, ed. A. Billoire et al., Nucl. Phys. B (proceedings supplement) **4**, 3 (1988).

2. Enzo Marinari, in: Field Theory on the Lattice, ed. A. Billoire et al., Nucl. Phys. B (proceedings supplement) **4**, 241 (1988).

3. A review of special purpose QCD machines, is presented by N. Christ, in Proceedings of Lattice Gauge Theory '88", to be published by Nucl. Phys B.