

**Fermi National Accelerator Laboratory**

FERMILAB-Conf-89/058

**A Site Oriented Supercomputer for Theoretical Physics:  
The Fermilab Advanced Computer Program  
Multi Array Processor System (ACPMAPS)\***

T. Nash, R. Atac, A. Cook, J. Deppe, M. Fischler, I. Gaines, D. Husby, T. Pham, and T. Zmuda

*Advanced Computer Program*  
Fermi National Accelerator Laboratory  
P.O. Box 500, Batavia, Illinois 60510

E. Eichten, G. Hockney, A. Kronfeld, P. Mackenzie, and H. B. Thacker

*Theoretical Physics Group*  
Fermi National Accelerator Laboratory  
P.O. Box 500, Batavia, Illinois 60510

March 6, 1989

\*Presented by T. Nash at the 4th Hypercubes Concurrent Computers and Applications Conference, Monterey, California, March 6-8, 1989.

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

# A SITE ORIENTED SUPERCOMPUTER FOR THEORETICAL PHYSICS: THE FERMILAB ADVANCED COMPUTER PROGRAM MULTI ARRAY PROCESSOR SYSTEM (ACPMAPS)\*

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

E. Eichten, G. Hockney, A. Kronfeld, P. Mackenzie, and H.B. Thacker  
*Theoretical Physics Group  
Fermi National Accelerator Laboratory  
Batavia, IL 60510*

## Abstract

The ACPMAPS multiprocessor is a highly cost effective, local memory parallel computer with a hypercube or compound hypercube architecture. Communication requires the attention of only the two communicating nodes. The design is aimed at floating point intensive, grid like problems, particularly those with extreme computing requirements. The processing nodes of the system are single board array processors, each with a peak power of 20 Mflops, supported by 8 Mbytes of data and 2 Mbytes of instruction memory. The system currently being assembled has a peak power of 5 Gflops. The nodes are based on the Weitek XL Chip set. The system delivers performance at approximately \$300/Mflop.

The system is programmable in C and Fortran. An underpinning of system routines (CANOPY) provides an easy and natural way of dividing site oriented problems among the many nodes, such that the detailed architecture of the system is transparent to the user. Users program the machine in terms of sites and fields on sites, not in terms of nodes. When intersite communication implies internode communication, CANOPY automatically handles it. CANOPY is now running on a variety of machines in single thread or parallel mode, including the ACPMAPS system, Sun workstations, Motorola 68020 and AT&T 32100 based systems, MIPS RISC processors, VAXes running ULTRIX, and PCs. It can be readily ported to any single CPU or MIMD system which supports C. A library of microcoded kernel mathematical routines has been written, enabling typical lattice gauge applications to run at high efficiency on the Weitek CPU.

The modular architecture may be adjusted to the problem types anticipated. Typically, the system is configured as a compound hypercube, with eight nodes on a crossbar and the crossbars connected as a hypercube. Full hypercube interconnections are also possible. Unlike traditional hypercubes, any node can communicate in a transparent manner at full speed with any other node without involving additional nodes. The individual nodes are in switch crates whose backplanes handle full 16 port crossbar switching at bandwidths of 20 Mbytes/second per connection. Interfacing modules, which allow communication with other crates at the same speed, may be plugged into the switch ports. The switches contain automatic routing information to other switch crates. This connectivity allows distributed I/O to low cost video technology storage devices from each crate to support check pointing of long calculations and archiving of results. A single tape volume can hold as much as 64 gigabytes of data.

A 32 node 640 Mflop system is currently running physics applications. A 256 node machine is being assembled and is expected to be operational by mid 1989. The 256 node computer will cost roughly \$1.5 million (at today's memory prices) and deliver over 5 Gflops (peak). Expansion to 2048 nodes is possible. The primary applications of this system at Fermilab are theoretical particle physics calculations, using lattice gauge techniques. The system is available for commercialization, allowing its application to many other problems. Among the configurations to be made available will be a one crate "department supercomputer", costing about \$100,000, and performing at the level of 1-3 CRAY X-MP's. It is anticipated that many workers with new site oriented codes will find the ACP machine easier to program than vector supercomputers.

---

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

## Introduction

The Advanced Computer Program Multi-Array Processor System (ACPMAPS) is a massively parallel floating point computer designed for lattice gauge theory and other grid-oriented problems. In the last few years, lattice gauge theorists have seen the achievement of two important milestones on the road to first-principles calculations of hadronic properties with reliable estimates of the accuracy of the calculations. First, apparently reliable Monte Carlo calculations of simple quantities in pure gauge theory (without quarks) have begun to appear, starting with the temperature of the deconfining transition in pure gauge theory. In addition to providing confidence that the program of large scale Monte Carlo calculation in four dimensional nonabelian gauge theory is succeeding, these calculations have put estimates of the computing needs for full QCD calculations on a firmer footing. Lattice sizes of  $32^4 - 64^4$ , requiring 1 - 20 gigabytes of data memory seem to be a reasonable guess.

Second, the search for improved algorithms for the most difficult part of QCD calculations, the inclusion of the effects of the sea quarks in hadron calculations, has been very successful<sup>1</sup>. On large lattices, the hybrid Monte Carlo algorithms which are now coming into use are roughly  $10^4$  times faster than the seven year old algorithms from which they descend, and are now "only" about 100 times slower than the analogous algorithms for pure gauge theory. Further improvements using nonlocal techniques such as Fourier acceleration and multigrid methods are quite possible.

The generation of special purpose computing machines now coming on line for lattice gauge theory are around  $10^4$  times as powerful as the VAXes on which the first Monte Carlo calculations of the hadron spectrum were performed in 1981. The combined improvement in hardware and algorithmic calculational power of  $10^8$  since 1981 approaches but does not quite meet the demands of full QCD if the more conservative estimates of calculational needs are correct. The remaining factor must and almost certainly will be met by further improvements in hardware and algorithms.

The Fermilab lattice machine was designed to provide very large amounts of computational power at reasonable cost, without compromising the programmability required for further rapid development. A sixteen node machine (320 megaflops, peak speed) has been constructed and is running physics code. It is a prototype for a 256 node machine (5 gigaflops, peak speed, 2 gigabytes of data memory) which is presently being assembled.

## Architecture

The architecture shares some features with all lattice machines which have been built since the first Columbia machine<sup>2</sup>: it is based on a massively parallel set of nodes each containing fast floating point hardware and locally accessible data memory. Each node typically performs calculations for the subset of lattice sites whose fields are stored in its local memory.

Among the most important differences between our machine and others of its type are programmability in an ordinary high level language, and the operation of the individual nodes totally asynchronously both in computation

and in communication. These resulted from two fundamental design goals for the machine: that it be programmable in a high level language and that the architecture of the machine constrain as little as possible the types of lattice problems which can be done on it.

Programmability in C and Fortran (we are currently using C) was made possible by the Weitek XL chip set. This chip set contains a 20 megaflop (peak speed) floating point unit, an integer processor, and an instruction sequencer. It is programmable as a whole using compilers supplied by Weitek. Each node contains one chip set, eight Mbytes of data memory and two Mbytes of code memory. The memory is 100 nsec page mode dynamic RAM.

The desire to have the architecture constrain possible physics problems as little as possible led to the use of totally asynchronous operation of the nodes (MIMD) and completely transparent nonlocal communication between nodes. MIMD architecture is very flexible: it can handle problems which are awkward or impossible for single instruction, multiple data (SIMD) architectures, such as heat bath and incomplete LU decomposition algorithms and random lattice problems. The allowed sizes and shapes of the lattices are independent of the details of the hardware. The node structure of the machine can be made invisible in the high level code, resulting in improved programmability. MIMD has the potential for communications bottlenecks which cannot occur with SIMD, but these do not seem to be very severe in the codes we have tested so far.

Asynchronous internode communication is made possible by a network of switch crates into which the nodes are plugged. They handle full sixteen port crossbar switching at bandwidths of 20 Mbytes/second per channel. This yields a total bandwidth of 2.56 gigabytes/second for a 256 node machine. The switches allow any node to access the memory of any other node without knowing where the other node is located on the network. With the current switch crate hardware, systems of up to 2048 nodes are possible before this transparent nonlocal communications feature is lost.

## ACPMAPS Hardware

ACPMAPS is designed to provide a hardware platform for development and running of lattice gauge algorithms. Such a system must meet several criteria: It must be powerful enough to try methods 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 to be large-scale I/O facilities for archiving intermediate results. Most crucial is the requirement that the system support a software structure which will facilitate the development of algorithms.

The machine consists of a compound hypercube of crates, each of which is a full crossbar switch containing several processors (Figure 1). 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.

By "compound hypercube" we mean an architecture wherein many multi-processor units (in our case, crates of single board processors) are configured as a hypercube, while

the connectivity within each crate is a superset of hypercube connectivity (here, a full crossbar). In situations where the communications bandwidths in the hypercube connecting the crates is adequate for the power of these multi-processors, this architecture contains the ordinary hypercube topology.

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)<sup>3</sup>. Occupying slots of these switch crates are the Floating Point Array Processor (FPAP) nodes<sup>4</sup> -- the crates are then connected by Branchbus cables, forming a hypercube. Branchbus is a 20 Mbytes/second intercrate cable system previously used by the ACP for experimental high energy physics VME based multi processors<sup>5</sup>. 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 switch crate is a Eurocard format crate with a sophisticated backplane. The BSB crate backplane consists of

a 16-port crossbar switch. The key components of this board are 13 Texas Instruments SN74AS8840 4-channel 16-port crossbar chips, and a Programmable Read Only Memory containing routing information. The programmable 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 -- the PROM determines the path the connection is to take. The intercrate connections can be reconfigured with a change in one PROM chip per crate. Thus the BSB crates allow the topology of the system to be matched to the needs of the applications to be run.

Each channel of communication can proceed at 20 Mbytes/second. Since eight such connections can coexist on one crate backplane, the aggregate communications bandwidth is 160 Mbyte/second. 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.

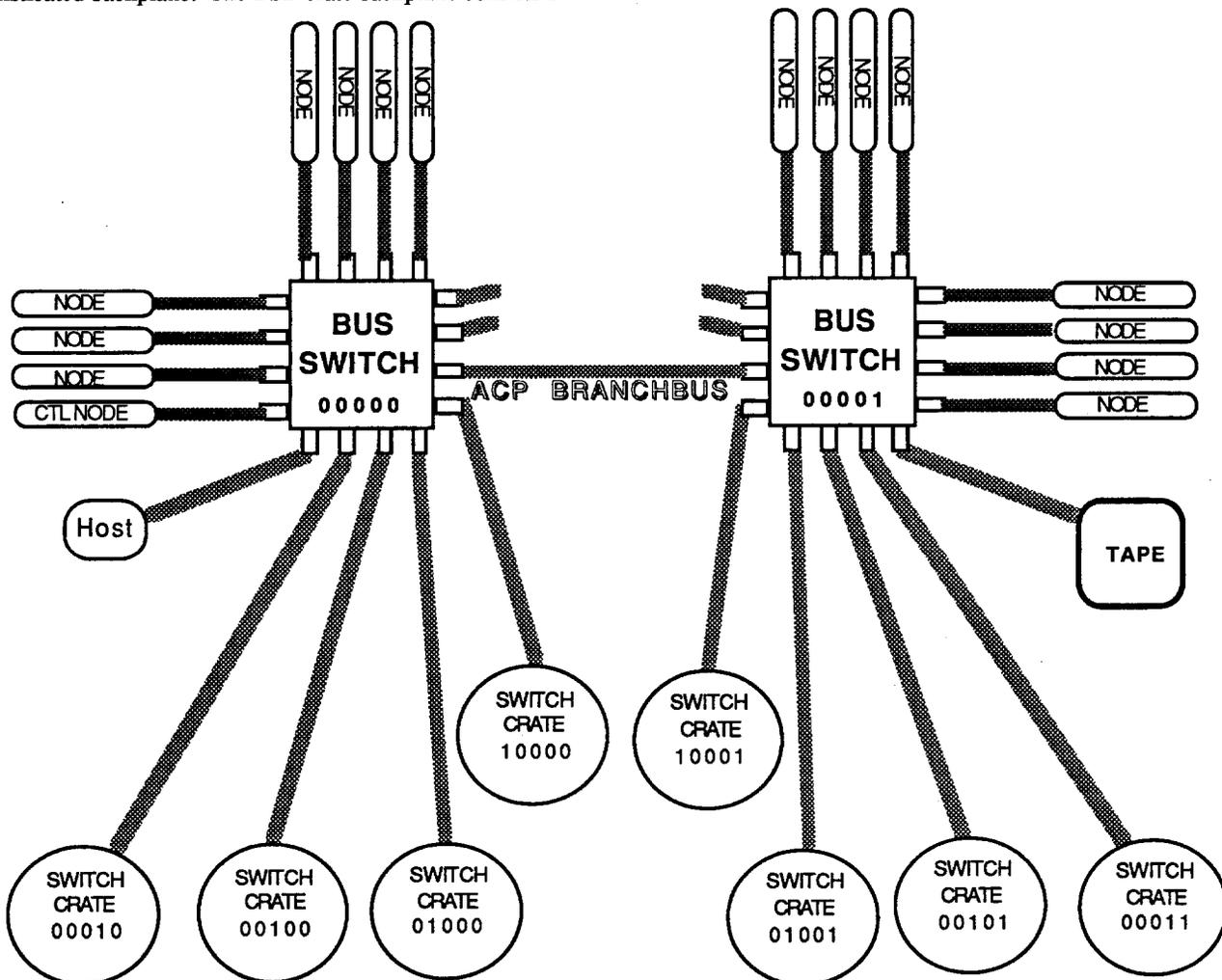


Figure 1: The ACPMAPS system consists of a hypercube of switch crates. Each slot in each crate may contain an FPAP processor node or a BSIB board providing a connection to a neighboring crate or other device. The system being assembled consists of a 2<sup>5</sup> hypercube of 32 switch crates, each containing 8 FPAP nodes.

Using a crate backplane itself to implement a crossbar switch shares the advantages usually associated with a bus crate, such as VME. A board simply plugs into the crate via connectors on the backplane; the processor board does not require extra cables emerging to connect to "neighboring" nodes. The crate can be fully or partially populated. Each module arbitrates for the "bus" and then owns that channel until it relinquishes it — this asynchronous internode communication allows MIMD operation. A separate stand-alone switch device is not necessary.

The BSB switch crate has two key advantages over conventional bus crates. The first results from the crossbar nature of the backplane — since many communications paths may be opened at one time, the effective "bus" bandwidth is up to eight times that available for conventional busses. The other advantage is transparent expansibility — a system of many interconnected switch crates appears to each individual node as a single extended crossbar switch. This simplifies the construction and programming of very large systems.

The FPAP is a single board computer based on the Weitek XL-8032 chip set, which plugs into one slot of the switch crate. 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 Mbytes of data memory and a separate 2 Mbyte 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.

(The cost of a single FPAP board [at today's processor and memory prices] is about \$5,000. The total cost for the 5 peak Gflop, 256-node system [including switch crates, tape drives, and host] is roughly \$1.5 million.)

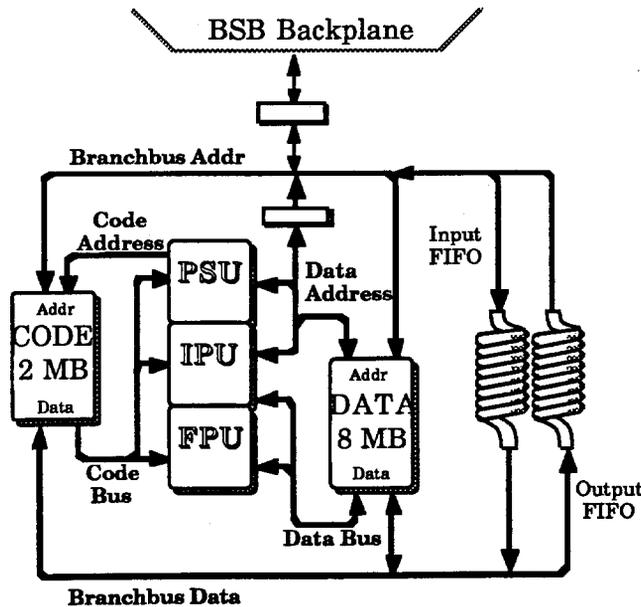


Figure 2: The FPAP consists of the 3-chip XL-8032 CPU, 8 Mbytes of data and 2 Mbytes of code memory, and an interface to the switch crate backplane.

This processor architecture is rather well suited to the computations needed in lattice gauge. For example, a routine to do a single SU(3) 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.

A special socket is being designed to allow substituting the new Weitek XL-3164 double-precision floating point unit for the existing XL-3132 FPU in an FPAP. Thus, a double precision version of the processor will be available if algorithms are found which need extensive 64-bit floating point operations. The modified processor would run at the same speed when doing 32-bit arithmetic, and about half as fast (due to memory access limitations) on double precision.

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.

The 256-node system being constructed is configured as a  $2^5$  hypercube of switch crates, communicating by ACP Branchbus cables to BSIB modules in slots in the crates. Each switch 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. This means that 8 nodes share the 20 Mbyte/sec intercrate communications bandwidth, so the system architecture will include a  $2^8$  hypercube of inter-processor bandwidth of 2.5 Mbytes/sec. (An alternative viewpoint is that the system is a  $2^5$  hypercube with internode bandwidths of 20 Mbytes/sec; here each "compound node" is a multiprocessing unit with total peak power of 160 Mflops.)

For the lattice gauge algorithms envisioned at present (the most severe cases being an FFT step and conjugate gradient propagator calculations), this bandwidth is a factor of two more than is needed for highly optimized codes. At the cost of adding more crates, one can increase the interprocessor bandwidth. An extreme would be 128 crates with 2 processors in each and two connections along each of the seven dimensions; this is a true  $2^8$  hypercube with the full 20 Mbyte/sec bandwidth between nodes.

The BSB crates allow complete flexibility in terms of intercrate connectivity. For smaller systems, one could easily choose to connect every crate to every other one. To minimize the number of crates, one could configure the system as a ring, with 14 processors and two interconnect modules in each crate. The choice of hypercube architecture is a compromise between these extremes, and has particularly nice scaling properties: For a large class of non-local algorithms, including Fast Fourier Transforms, the average computation time needed per site rises logarithmically with the problem size; the average communication need grows the same way in a hypercube, so the algorithm does not become more communications intensive as the problem size varies.

All ACPMAPS modules will be made available for commercialization, allowing a system to be configured to the needs of the user. In order to enhance 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 X-MP'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. Code for this machine can be ported unchanged to larger ACPMAPS systems.

## Software and Programming

The control program, which manages global tasks such as defining lattices and fields and starting global operations on fields, executes on one of the nodes. This "control" node is identical to all of the other nodes except in software. Global (lattice-wide) objects and operations such as global fields and operations on them, might be implemented as arrays and for-loops on a one-CPU machine. These must be handled with extensions to the language when the data and operations are spread over many CPUs. The extensions should reflect as much as possible the concepts of the problems to be solved. Some of the fundamental concepts of lattice problems are:

Objects	"Algebras"	Operations
sites $s$	grid:	$s' = s + d$
directions $d$		$s' = s + p$
paths $p$		$p' = p + d$
unitary matrix $u$	SU(3):	$u_3 = u_1 u_2$
quark $q$		
fields $U, Q$	Dirac equation:	$(D-m)Q = \delta$

In object oriented languages such as C++ which are just beginning to appear, these concepts could be added to the language as new data types and operations. To us, the structure rather than the syntax is most important and we implement them with C's typedef and C subroutines called from the control program. Our software package for doing grid-oriented problems on parallel machines is called CANOPY<sup>6</sup>.

Consider, for example, the following set of statements from a control program.

```
lat1 = periodic_grid (NDIM, latsize);
q = site_field ( lat1, sizeof(quark) );
q1 = site_field ( lat1, sizeof(quark) );
complete_definitions ( );
```

The function `periodic_grid ( )` tells the system that our calculation will be done on a lattice of NDIM dimensions whose sizes are contained in the array `latsize`, and which will be identified by `lat1`. The functions `site_field ( )` tell the system that memory will be required for two fields identified by `q` and `q1`, each with `sizeof(quark)` bytes for each site of `lat1`. The `complete_definitions ( )` function calls routines which assign specific sites to specific nodes, allocate memory in the nodes for the field data and site structures, and set up structures for each site pointing to the memory areas of adjacent sites of the lattice.

The loop over lattice sites in a function which operates on a field `q` with an operator `dslash` and stores the result in another field `q1` is replaced by the statement

```
do_task( dslash_, lat1,
        PASS, q, sizeof(q),
        PASS, q1, sizeof(q1),
        END);
```

The system function `do_task ( )` passes to all the nodes a pointer to the user supplied function `dslash_` and an identifier of a list of sites on which to operate, which may be the entire lattice `lat1` or some previously defined list of sites such as `red_sites`. A system subroutine on the node, invisible to the user, calls `dslash_` for the sites in the set of sites which have been assigned to the node. `do_task` may be used to pass (PASS) to the nodes arguments required by the function (such as the field identifiers `q` and `q1`), or to integrate (INTEGRATE) data returned from the individual nodes.

The site subroutines access and replace data from global fields with system functions such as

```
pq = field_pointer( q, &site1 );
```

They determine whether the desired data is already present in the node's local memory and open a channel to the communications hardware if necessary.

CANOPY is written in C and is easily portable to any single-CPU or MIMD multiprocessing system with support for UNIX calls. Thus, programs can be tried out on small lattices on a workstation, and migrate to the production machine without changing any code. 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. 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 modification and debugging, to the ability to confidently optimize critical sections of the code.

The overhead associated with CANOPY is typically less than 15%. Generally, the improvement in code structure associated with using CANOPY more than outweighs any actual overhead. That is, the modular nature of code, and the fact that computations of addresses of field elements proceed via a package of efficient system routines, normally lead to a speedup in execution. This gain often more than offsets the time spent executing the CANOPY routines themselves.

Although this software framework was designed with lattice gauge theory in mind, CANOPY is applicable to any site-oriented scientific problem. Obvious examples include finite element analysis, partial differential equations, and large scale simulations such as weather computations. Other algorithms can run under this framework as well. For example, inversion of large, dense matrices can be accomplished by treating each row of the matrix as a "site".

## Performance

Three relevant aspects of the performance of the ACPMAPS system have been studied. The performance of a single FPAP node on actual physics programs was measured. (When these codes are brought up on a supercomputer or dedicated QCD machine, the optimization process often involves prodigious efforts. Considerably less effort was required to get good performance results for the ACPMAPS system.) Then the physics programs were run on a medium scale multi-node system, demonstrating that the speedup is

linear and that the communications overheads are small. Finally, a calculation was modified to artificially increase the stress on communications bandwidth — the results here indicate performance limitations stemming from the compound hypercube architecture in the full-scale system will be very small.

A sample algorithm which has been carefully optimized on the ACPMAPS system is a pseudo-heat bath code for generating gauge configurations. The program sweeps through the gauge fields on the lattice (links) and, for each link, determines how it should be updated. This basic heat-bath Monte Carlo has been coded and optimized for almost every lattice gauge machine — a natural measure of performance is the number of link updates accomplished per second.

To bring the algorithm up on ACPMAPS, the code was first written in C. The compiled code of key routines such as SU(3) multiplication was then replaced by hand optimized code in a modular way as the time consuming parts of the code were identified. A library of these hand-coded mathematical kernel modules is maintained. This modular approach is particularly valuable on the ACPMAPS system because the overhead for entering such a routine is minimal, and because the CANOPY makes modularity very natural to implement. Since each module can perform one clear function, and need not be tailored to peculiarities in data structures or parallelism for a particular application, coding optimized modules for this library has not been a difficult task.

The current link update time for this code is 0.60 msec on a single node. Link update times for this algorithm have been published for two other QCD machines<sup>7,8</sup>. (These were programmed in assembly or assembly-like languages, and very carefully optimized). The ACPMAPS performance is roughly 40% faster than these, when normalized to the peak speeds of the machines.

The gain of a few tens of per cent in relative efficiency, while very encouraging, is not the only important point, and will vary from algorithm to algorithm. Most important is that a very high efficiency was obtained using high level programming and modular optimization. This contrasts with the prodigious effort often needed to port an algorithm to a special-purpose machine. Bringing up and optimizing this algorithm on the ACPMAPS system also involved considerably less effort than putting it on a CRAY.

When many MIMD processors work on a site-oriented problem, there can be three sources of deviations from perfect linear speedups: (1) "Tail effects", wherein the last processor to complete a task delays the remaining processors; (2) Communications overhead to obtain data associated with off-node sites; (3) Communications bandwidth limitations. The first of these effects is minimal when there are a large number of sites handled by each processor; this is the normal situation for lattice gauge (and most other) problems. Both of the other effects grow with the surface to volume ratio of the sites handled by each node. So, even though pipeline startup and overhead effects may be negligible, the ACPMAPS system shares the ubiquitous feature that performance scales with the number of processors only if the problem size also increases. Fortunately, that is indeed the case for lattice gauge calculations today: Increased power is

desired precisely to be able to work with larger (finer grained) lattices.

To measure the communications overhead/bandwidth effects, we have run the optimized gauge configuration generators on lattices of fixed sizes (small enough to fit into one or two nodes). Results for very small and medium lattices agree: The processing time is proportional to the volume of sites in a node, plus about .08 times the number of boundary sites. This is illustrated in Figure 3 for a lattice of dimension  $10^4$ . The nearly linear appearance of this figure shows that even for the harsh case of a small fixed lattice size, performance is roughly scaling with number of nodes (the sixteen node system performs somewhat faster than a CRAY X-MP). The efficiency, normalized by running a smaller lattice on a single node, remains above 85% when using many nodes. The magnitude of the communications cost can be understood quantitatively in terms of communications overhead alone. This means that the finite intracrate and intercrate bandwidths cost very little for this problem.

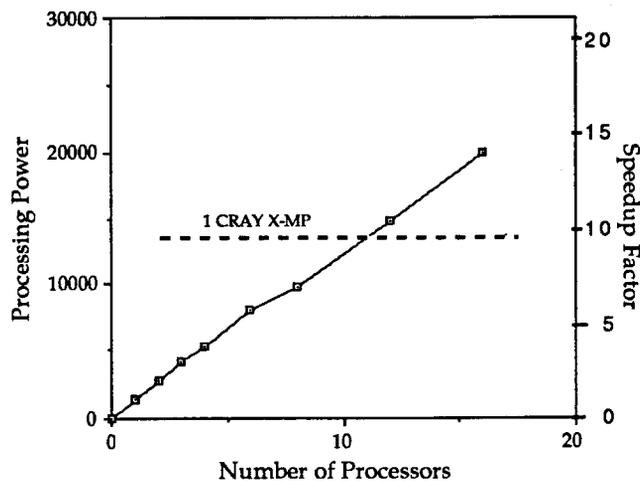


Figure 3: Performance speedup of a multi-node ACPMAPS system, plotted against number of nodes, for a configuration generator on a lattice of fixed size  $10^4$ . The processing power is measured in number of link updates per second. A smaller lattice was run on a single processor to provide a basis for the speedup factor relative to a single node. The performance of the CRAY is based on an elapsed time per link update of 73  $\mu$ sec taken for an optimized program, running on one X-MP CPU.

Since the full scale system has an intercrate hypercube bandwidth which is only as large as the crossbar bandwidth between two nodes within a crate, any communications bandwidth limitations will be dominated by intercrate communication. To focus on this effect, we have measured the performance of two-crate systems with a special distribution of sites among the nodes: Instead of grouping the nodes such that neighboring sites tend to be in the same crate, we intentionally software-configured the lattice such that neighboring processors were in opposite crates. This artificially created intercrate traffic along the single interconnection of up to 8 times the anticipated traffic in an actual system. The results, for a  $16^4$  lattice, are shown in Figure 4.

For relatively unsaturated intercrate bandwidth, one would expect the cost of intercrate communication to scale

with the square of the number of boundaries between neighboring nodes that cross the intercrate connection. (The fraction of time the communications path is unavailable, and the frequency of communications along that path, are each growing with the intercrate communications load. As long as these effects are small, the net cost is the product of them.) This quadratic behavior is exhibited in Figure 4, indicating that our measurements are not being distorted by saturation effects.

For the worst case of sixteen boundaries (no neighboring nodes within the same crate), the performance hit is 15% (17800 link updates per second, compared to 20700 for only two boundaries). Without the artificial increase in intercrate traffic, the hit was 64 times smaller. That effect was unobservable on the performance speedup study (Figure 3).

The most important (in terms of processing needed) and communications intensive problem we see on the horizon is computation of the fermion propagator. This calculation uses Conjugate Gradient or similar algorithms. These algorithms are expected to be about twice as communications intensive as this gauge configuration generator. In that case, the anticipated performance hit due to intercrate communications bandwidth is about 3%. For 256 node system, in particular, it would not be cost effective to decrease the number of processors per crate to increase the per-processor intercrate bandwidth.

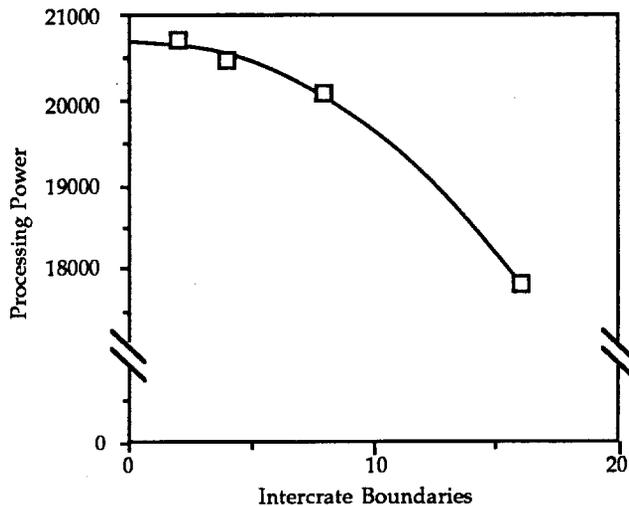


Figure 4: Processing Power running a gauge configuration generator, versus number of boundaries between processors handling neighboring sites which cross the intercrate connection. ( $16^4$  lattices were run on 16 processors in each case.) Processing Power is measured in number of link updates per second. The data agrees well with the parabolic fit shown.

To summarize the performance issues, large lattice gauge problems run on multiple nodes with an 85-90% efficiency relative to the single processor speed. (The prototype system mimics the intercrate bandwidth of larger systems by using

only one Branchbus between crates.) This performance hit can be understood largely in terms of the several-microseconds of overhead involved each time an off-node field access is done (which is a sizable fraction of an SU(3) multiply time). A further degradation due to intercrate bandwidth limitations has been studied, and is estimated to range up to 3%.

## Current Status

The sixteen node prototype machine is built and being used for code development and physics production running. Several programs have been brought up, including algorithms currently in use by theorists for extracting actual physics. The list includes a Kennedy-Pendleton heat bath algorithm with quenched fermions, a calculation using the "Hybrid Molecular Dynamics" method with unquenched Kogut-Susskind fermions, and Conjugate Gradient algorithms to find propagators for Wilson fermions. With only casual optimization efforts, these typically run at speeds within 30% of their optimum. This is encouraging — a wide variety of algorithms can be studied without extensive programming efforts, and without sacrificing much machine performance.

The 256-node production system is being assembled; 16 of the nodes and two of the BSB crates destined for this system have temporarily been incorporated into the prototype to provide enhanced development facilities with a total of 32 nodes.

## References

- [1] Don Weingarten, Monte Carlo Algorithms For QCD, to appear in the Proceedings of the 1988 Symposium on Lattice Field Theory, Fermilab, Sept. 22-25, 1988, to be published in Nuc. Phys. B.
- [2] For a review of special purpose QCD machines, see Norman Christ, QCD Machines, to appear in the Proceedings of the 1988 Symposium on Lattice Field Theory, Fermilab, Sept. 22-25, 1988, to be published in Nuc. Phys. B.
- [3] R. Atac *et al.*, Crossbar Switch Backplane and its Applications, to be published in Proceedings of 1988 IEEE Nuclear Science Symposium.
- [4] D. Husby *et al.*, A Floating Point Engine for Lattice Gauge Calculations, to be published in Proceedings of 1988 IEEE Nuclear Science Symposium.
- [5] I. Gaines, *et al.*, The ACP Multiprocessor System At Fermilab, Computer Physics Communications 45,323 (1987).
- [6] CANOPY Version 2.0, Fermilab Theoretical Physics Groups & Fermilab Advanced Computer Program, December 1988. Internal document, to be released as CANOPY User's Guide.[6] I. Gaines, *et al.*, The ACP Multiprocessor System At Fermilab, Computer Physics Communications 45,323(1987)
- [7] Enzo Marinari, in Field Theory on the Lattice, ed. A. Billoire *et al.*, Nuc. Phys. B (proc. Suppl.) 4,3 (1988).
- [8] Norman Christ, in Field Theory on the Lattice, ed. A. Billoire *et al.*, Nuc. Phys. B (Proc. Suppl.) 4,241 (1988).