Fermi National Accelerator Laboratory

FERMILAB-Conf-125-T
May, 1989

Machines for Lattice Gauge Theory*

Paul B. Mackenzie
Theoretical Physics Group
Fermi National Accelerator Laboratory
Batavia, IL 60510 USA

Abstract

The most promising approach to the solution of the theory of strong interactions is large scale numerical simulation using the techniques of lattice gauge theory. At the present time, computing requirements for convincing calculations of the properties of hadrons exceed the capabilities of even the most powerful commercial supercomputers. This has led to the development of massively parallel computers dedicated to lattice gauge theory. This talk will discuss the computing requirements behind these machines, and general features of the components and architectures of the half dozen major projects now in existence.

1 Introduction

There is overwhelming qualitative evidence that quantum chromodynamics (QCD) is the correct theory of the strong interactions, but quantitative success with the theory has so far been meager. Perturbation theory has had some success in describing some very high energy phenomena, but no completely solid results have been obtained for low energy strong interactions. Other theories may also require nonperturbative treatment. For example, if no light Higgs boson is found, the odds are good that the dynamics which gives mass to the W and Z bosons is strongly interacting.

Wilson has formulated gauge theories on a discrete grid: lattice gauge theory. This has made available new calculational approaches to QCD. At present, the approach with the most promise seems to be large scale numerical simulation of the equations. The theory is solved on a four dimensional space-time grid with finite volume and finite lattice spacing, and then one attempts

*Talk given at the conference "Computing in High Energy Physics", Oxford, England, April 10-14, 1989.

to find the large volume, small lattice spacing limit. This approach is similar in many respects to the use of finite element methods for computer calculation of air flow around an airplane wing or the calculation of stresses in structural analysis. Simulations of two dimensional problems can be done very accurately, sometimes even on medium sized work stations. Existing supercomputers can perform three dimensional simulations well in some but not in all cases. It is not surprising that simulation calculations for four dimensional quantum field theories have to a large extent been too taxing for even the largest existing computers.

This has led to an intense search by particle theorists for ways of obtaining the maximum possible computing power, both in floating point operations per second and in algorithms.

The technological background. Coupled with these large computing demands, there is a window of technological opportunity. Lattice calculations require very large numbers of floating point operations and very large amounts of memory. Fast floating point chips which can deliver 20 Mflops now cost a few hundred dollars, and 1 Mbit memory chips cost $12 to $15. This shows that with an architecture based on a large number of the chips operating in parallel the key components of a 10 Gflop, 1 Gbyte computer cost as little as $300 thousand. It turns out that it is possible to build such a computer for a total of about 3-10 times this figure, depending on how many bells and whistles are desired. (Disks, tape drives, and compilers are considered bells and whistles by the designers of these computers.) This results in roughly a two order of magnitude difference in cost effectiveness compared with a $10 million 1 Gflop Cray XMP. This factor is clearly very significant, even allowing for the fact that commercial products improve a bit while an experimental computer is designed and debugged.

The history of QCD machines. The building of experimental computers to perform specific scientific calculations is not a new phenomenon. The modern era of computing was ushered in during World War II by the ENIAC and EDVAC projects whose original purpose was to use state of the art electronic technology (vacuum tubes) to calculate ballistic trajectories for constructing firing and bombing tables. The von Neuman architecture which arose out of those early projects has become the standard since then: the computer executes a single thread of instructions from a stored program. The availability of large quantities of cheap components has meant that this architecture, on which conventional commercial computers are based, is no longer optimal for many of the problems in which physicists are interested, including lattice problems.

More recently, many particle theorists became interested in large scale computing when Monte Carlo methods were first applied to lattice gauge theory,

around 1980. The first recent machines to be constructed to solve a specific problem in theoretical physics were the machines built at Santa Barbara[1] and Delft.[2] These were aimed specifically at the Ising model. At about the same time, groups at Tsukuba and Caltech were constructing more general purpose parallel machines. They were not aimed primarily at providing production computers for a specific purpose like the more recent lattice gauge theory projects, but instead were aimed at exploring the possibilities of parallel computers. They very effectively demonstrated that a wide variety of interesting scientific problems can be performed on parallel architectures. The first QCD calculations on a parallel machine were done by the Edinburgh group on the commercial machine, the DAP, built by the UK company, ICL. [3] The effective use of this machine to perform important calculations further demonstrated the potential of parallel architectures.

The first machine designed primarily with QCD in mind was the sixteen node machine built at Columbia University, which began producing physics results in 1985. It was the first parallel machine to incorporate fast floating point chips to achieve high cost effectiveness. There are now at least six projects to build machines with large CPU power, large memory, and fast communications. They are at the University of Tsukuba (QCDPAX) [4], Caltech (Mark III) [5], Columbia University [6], IBM (GF 11) [7], Italy (APE) [8], and Fermilab (ACPMAPS) [9]. All of these machines except Caltech's are aimed specifically at QCD. Like the first experimental computers designed during World War II, although they are mostly aimed at specific scientific goals, they are fairly general purpose and suitable for a wide variety of scientific problems.

In 1988, the largest QCD calculations were performed on the 1 Gflop (peak speed) machines of Columbia and the APE collaboration. Machines are now under construction (at Tsukuba, Columbia, IBM, and Fermilab) which have peak speeds ranging from 5 to 15 gigaflops, and memories ranging from .5 to 2 gigabytes. For a nice review of the status of the individual projects, see [10].

2 What kind of computer do we need for lattice QCD?

This section will focus on rough estimates of computing requirements for some of the simplest QCD calculations: the masses of the light hadrons. (These, of course, are only a small part of the many calculations that we ultimately want to do with lattice methods.)

How much memory? The physical volume required for a reasonable simulation can be estimated on physical grounds. From scattering data, we know that the diameters of light hadrons are of the order 1.3 fermi (pion) to 1.6 fermi (proton). This suggests that volumes of L = 2-3 fermi across may be

appropriate for hadron mass calculations. Lüscher[11] has calculated exactly the asymptotic finite volume errors in a periodic box. These are dominated by fake pion loops which go out one side of the box and come back in the other side via the periodic boundary conditions. He finds that these are very small as long as the box is larger than around 3 fermi. Parisi[12] has pointed out that it is reasonable to expect finite volume errors comparable to nuclear binding energies ($\sim$ 14 MeV) when the nucleon density in the periodic volume is comparable to nuclear densities (L $\sim$ 1.8 fermi). A more pessimistic line of handwaving may be produced by considering momentum scales of the quarks inside hadrons. From scattering experiments, we know that quarks have typical transverse momenta of a few hundred MeV. In a finite volume of $L =$ four fermi across, the momentum is quantized in units of $p_{min} = 2\pi/L \sim 300$ MeV. This suggests that volumes of (4-6 fermi)^3 may be necessary to correctly reproduce the dynamics of the quarks inside hadrons. A final piece of information may be obtained from an existing lattice calculation. The temperature of the deconfining transition in pure SU(3) gauge theory was calculated on large lattices in 1985-6, by two groups: a California group[13] using conventional supercomputers, and the Columbia group[14] using a dedicated lattice gauge engine. This was probably the first calculation of a "physical" quantity done with simulation methods for four dimensional nonabelian gauge theory which achieved a solid control of errors, including finite volume and finite lattice spacing errors. Although the calculation applies to a world with no light quarks, and is thus only qualitatively applicable to our world, it still provides a sort of bound on the lattice sizes required for real QCD, since the addition of quarks to the theory can hardly do anything but make the various systematic errors worse. These groups found that finite volume errors were not much worse than their statistics (under 10%) on lattices around 2 fermi. Combining all these arguments, it's quite likely that lattices somewhere between 2 and 6 fermi across are what is needed.

The equally important question of what lattice spacing is required to make errors small is much harder to answer on physical grounds. The T_c calculation referred to above revealed no discernable finite lattice spacing errors when the lattice spacing was less than 0.1 fermi. Including the effects of quarks can only make the errors worse. This suggests lattice spacings of .05 - .1 fermi or less for full QCD. It is possible in principle to increase the allowable lattice spacing by using Wilson's renormalization group methods to reduce finite lattice spacing errors (much as using the slightly more complicated Simpson's rule reduces the discretization errors in a numerical integral compared with the trapezoidal rule). These methods are complicated and little practical progress has been made with them in QCD calculations. At present, it looks as if the estimates for the lattice sizes (and therefore for the memory) required are unlikely to be reduced by improved methods.

We are thus led to guess that lattices with $20^4 - 128^4$ sites are likely to

be required for QCD calculations, with 20^4 being an optimistic minimum, and $32^4 - 64^4$ being more probable guesses. What are the data memory requirements for lattices of this size? The gluons are vector fields (four for each site) of 3x3 complex SU(3) matrices. The quarks have four spins times three colors of complex numbers stored for each site. The more efficient algorithms may use several copies of each type of field. One kilobyte of data memory required per site is a good estimate. The lattice sizes given above thus require 0.16 - 270 gigabytes of data memory, with 1-16 gigabytes being the best guess.

How much CPU power? This uncertainty of perhaps three orders of magnitude in the number of lattice sites required contributes a like factor to the uncertainty in the CPU time required. On top of this there is an additional uncertainty in the CPU time required per site.

The properties of hadrons are calculated in lattice gauge theory by calculating the propagation of quarks through a set of configurations of the gluon fields. On the lattice, gluons are represented by SU(3) matrices defined on the links which connect neighboring sites of the lattice. The probability of choosing a configuration is proportional to $\exp(-S)$, where the action is a function of the gluon fields. If the effects of sea quarks are neglected ("pure gauge theory"), this is a local function of the gluon fields which is usually taken to be proportional to the sum of trace of the products of the SU(3) matrices on the small squares of the lattice (the plaquettes).[15] To produce a new statistically independent configuration, a long series of small changes is made to an existing configuration. The changes to each link are guided by the six sets of three neighboring links in the plaquettes which include it. An operation which is repeated over and over is performing the two matrix multiplications to join the links of each of the six triplets. This operation accounts for 50% to 80% of the CPU time in pure gauge updating algorithms. Twelve complex 3x3 matrix multiplications require 2304 floating point operations. There are four links per site, so this operation takes of order 10^4 operations per site. To produce 100 gauge configurations separated by 1000 sweeps thus requires of order 10^9 floating point operations per site.

When the effects of sea quarks are included ("dynamical fermions"), CPU requirements are much worse. Instead of a local function guiding gauge dynamics, the gauge fields are affected by quarks travelling on arbitrarily long paths around the lattice.

To calculate the effects of these quark paths on the gauge fields, it is necessary to make many sweeps through the lattice with the Dirac operator for each change to the gauge fields. A four component Dirac quark is represented by four complex three vectors at each site. In calculating the Dirac operator at a site, the three vectors at each of the eight neighboring sites are multiplied by the SU(3) matrices linking them to the central site. This requires 2112 operations per site. The number of operations for a calculation is then the

product of 2112 (plus small change), the number of configurations, the number of gauge sweeps separating each configuration, and the number of Dirac sweeps required for each gauge sweep. Results on small lattices suggest estimates of 100, 1000, and 1000 for the last three numbers, yielding a very rough estimate of 10^{11} operations per site for a single calculation.

These numbers are only order of magnitude estimates of the requirements. Furthermore, the number of gauge and Dirac sweeps required is expected to grow with the lattice size in a way which is incompletely understood, but could be large (the problem of "critical slowing down"). On the other hand, algorithms which reduce the CPU time required for QCD with dynamical quarks have made amazing progress over the last eight years. On a 32^4 lattice, the hybrid Monte Carlo algorithm [16] is perhaps 10,000 times faster than the algorithm of Weingarten and Petcher from which it descends. There are many promising ideas which have not yet been fully exploited which may further improve the speed of our algorithms by further large factors. These include Fourier acceleration of the simulation and quark propagator programs, ILU preconditioning of the quark propagator programs, and multi-grid methods. These methods have the potential for cutting the amount of CPU time required by another one to two orders of magnitude, but it is by no means guaranteed that they will achieve their potential.

The range of possible CPU requirements is therefore very broad. We could imagine getting away with as little as 0.1 gigaflop-years of CPU time if 20^4 lattices are adequate and there is more algorithmic progress. (A gigaflop year is $3\text{x}10^{16}$ floating point operations.) Many teraflop-years may be required if algorithmic progress stops dead in its tracks (unlikely in my view) and much larger lattices are required.

How much programmability? The power of the multigigaflop machines now being constructed is about four orders of magnitude greater than that of the Vaxes on which the first QCD calculations were performed ten years ago. Of this factor, about two orders of magnitude are due to the experimental architecture, one order of magnitude to component improvements over the last ten years, and one order of magnitude to more money being spent.

It is impressive that, large as this factor is, a like factor has been achieved in the improvements to algorithms over the same period of time. This strongly suggests that the programmability necessary for rapid investigation and refinement of algorithms is as important as CPU power for QCD machines. This aspect of the machine is, however, impossible to quantify.

3 The design of a computer for QCD

This section will discuss the decisions to be made in designing a massively parallel supercomputer. All the machines designed for QCD since the 16 node

Columbia machine have been similar in many respects. They consist of a large number of cheap nodes. Each node does floating point calculations for the sites in its own part of the lattice using one or several fast floating point chips which multiply and accumulate at peak speeds of 16-64 Mflops per node. Each node is associated with many megabytes of memory for the fields on the sites in its part of the lattice. The machines have used different approaches in controlling the nodes, in communications, and in programming.

Parallelization strategy. From the first, all computers have made use of some types of parallelism, for example the parallel transfer of the bits making up a single number. More recently, high energy experimentalists have made effective use of trivial or "event parallelism"[17]: if a large computing problem consists of a large number of small independent problems, it can be run on an array of small, independent CPUs. This type of parallelism is not appropriate for lattice problems, which require a huge amount of memory in each lattice.

The appropriate way to run a lattice problem on a parallel computer could be called "site parallelism." The sites in a large lattice are apportioned among a large number of nodes. Each node keeps track of the data and does calculations for the sites under its control. It must communicate with other nodes when it needs data from a site not in its local memory.

Arithmetic chips. Most SU(3) lattice calculations are dominated by the calculation of dot products of complex three vectors, or in terms of real numbers, by the operation $c = a_1b_1 + a_2b_2 + a_3b_3 + a_4b_4 + a_5b_5 + a_6b_6$. This suggests that the arithmetic chips on each node be capable of floating point multiply and accumulate as a fundamental operation.

The Caltech and Fermilab projects, which have most emphasized programmability, have used the Weitek XL chip set. It consists of three chips: the Weitek 3132 multiply and add chip, an integer processor, and and instruction sequencer. The chip set as a whole is programmable in Fortran and C. Time critical subroutines are hand coded for maximum efficiency. The Caltech chips are run at 16 Mflops peak speed (8 megahertz), the Fermilab chips at 20 Mflops.

The Columbia machine uses the 64 bit version of the above floating point chip, the Weitek 3364, without the integer processor and sequencer. It is run at a faster clock speed, and has two chips per board instead of one. This gives a peak speed per board of 64 Mflops, but makes it harder to program since all floating point operations must be microcoded. A similar approach is used by the IBM machine, which also employs the strategy of two add and accumulate units per board, with older Weitek chips. Tsukuba is the only group not using Weitek floating point chips in its current machine. It is using a single LSI L64132 per node, which has a peak speed of 32 Mflops and a single stage pipeline.

A strategy which is only slightly different is employed on the APE machine. On each node, the outputs of four multiply chips are piped into four add chips, arranged so as to optimize complex multiply and accumulate. They are planning on using the more standard real multiply and accumulate in their next machine.

These approaches are all very similar, with some trade-off being visible between speed and programmability. A question which is not well understood is the extent to which 32 bit precision can affect efficiency and results.

Memory chips. The relative amounts of money invested in CPU power, memory size, and bandwidth to memory are important optimizations in the design. The amount of CPU power required per site, and therefore per megabyte of data, is hard to estimate even to the nearest order of magnitude. Even if we knew the right ratio of CPU power to memory size for one problem, it would be different for other problems. Therefore, spending comparable amounts on CPU power and memory is the sensible thing to do: even if we work on a problem for which the needed ratio is different by a factor of a thousand one way or the other, we could only have save 50% by designing a machine with exactly the right ratio for that particular problem.

Memories are arranged in hierarchies. The floating points units are fed by a small number of very fast registers. These are in turn fed by somewhat slower main memory. The number of registers ranges from 32 to 128. The XL chips contain 32 registers on the floating point chips themselves. This in not enough to contain two input and one output SU(3) matrices at the same time, so some temporary storage in main memory is required during an SU(3) matrix multiply. The GF11 uses 128 registers and thereby reduces some overhead due to memory transfers.

The chips used for data memory may be high-speed static RAM, which has access times of around one word every 20-50 nsec., dynamic RAM, which is cheaper, four times as dense and has access times of 80-200 nsec., or a hierarchical combination of both.

Control: SIMD and MIMD. The standard architecture of conventional computers is sometimes called SISD (for Single Instruction stream, Single Data stream). There are two basic strategies for the control of a parallel system working on a single problem. In Single Instruction Multiple Data (SIMD) computers, all of the individual processors execute precisely the same instruction on each cycle. In Multiple Instruction Multiple Data (MIMD) computers, the processors act as independent computers.

SIMD machines have the advantage that space for code memory and a local controller for the floating point chips need not be set aside on each board. This reduces the cost and simplifies the design and debugging of SIMD boards. They also have the advantage that since communications are done in

lock step, bottlenecks cannot occur in which many processors want to access the same memory or data path at the same time.

MIMD machines are more flexible. They can handle algorithms such as the heat bath algorithm which are awkward with SIMD machines. They can handle random lattices and the irregular lattices of finite element problems. Possible lattice sizes are not constrained by machine hardware. MIMD machines can be more programmable. Commercial compilers can be used for the basis of the software on each node. The node structure of the hardware can be made invisible to the user.

Both approaches have been used for lattice machines, and at the present time both approaches have their proponents. So far it seems that the communications bottlenecks of MIMD machines are not very severe, and that SIMD machines are adequately fault tolerant and are flexible enough to handle at least the algorithms which have at this point been proven to be effective for QCD. Proponents of SIMD tend to believe that SIMD will deliver more flops per dollar. Proponents of MIMD tend to believe that MIMD will deliver more physics per dollar by being more flexible and programmable.

Figure 1 shows block diagrams of the machines being discussed. The IBM and APE machines are completely SIMD. Microcode instructions for the floating point units are broadcast to the nodes from an external controller: a 3081 Emulator for APE, and a PC RT for IBM. The other four machines have local instruction memory and controllers. The Columbia and Tsukuba machines operate synchronously, all nodes timed by a single clock. The floating point units on each node are controlled by microprocessors: a Motorola 68020 for Tsukuba and an Intel 80286 for Columbia. Programs may be run in MIMD fashion until off-node data needs to be accessed, when they are resynchronized. The Fermilab and and Caltech machines are completely MIMD; computation and communication are completely asynchronous, with the XL chip set on each node acting as an independent computer.

Communications. For many QCD lattice problems, roughly one eighth to one quarter of accesses to memory are for data which is not stored on the node doing the processing. For example, suppose a 16 node computer is working on a 16^4 lattice and one timeslice (data for all the sites having the same time coordinate) is stored on each node. When collecting the nearest neighbor fields, data for the sites in the six neighboring directions having the same time coordinate is already on node, but data for the sites in the $\pm$ time directions must be obtained from another node's memory. Fourier acceleration is an example of a possible algorithmic improvement which is even more communications intensive. Its memory accesses are almost entirely off-node.

This means that the the requirements for bandwidth to off-node memory approach those for bandwidth to on-node memory.

The nodes of the Columbia and Tsukuba machines are arranged in a two

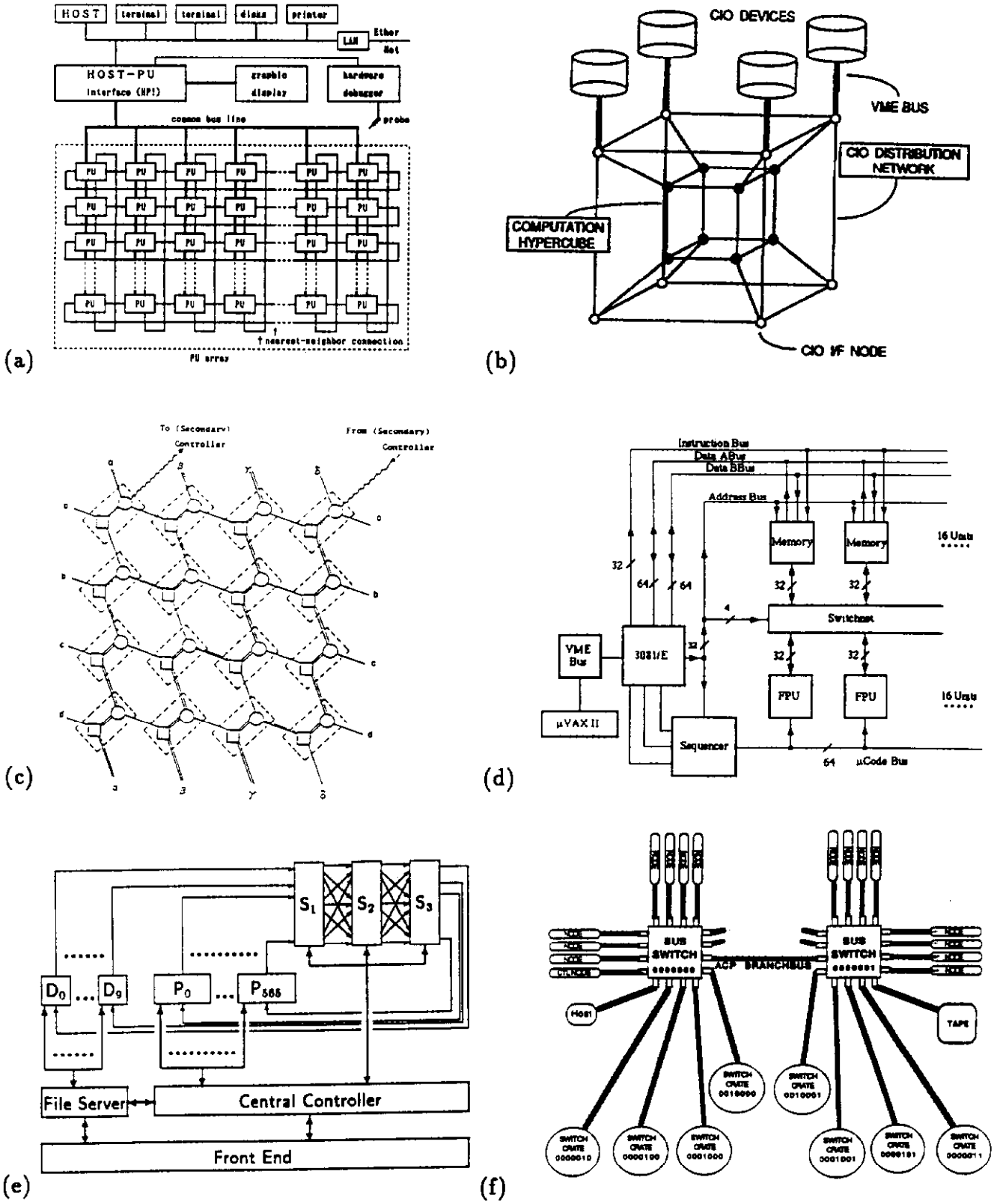

Figure 1: Architectures for lattice machines. a) QCDPAX (University of Tsukuba); b) Mark III (Caltech); c) the Columbia University machine; d) APE; e) GF11 (IBM); f) ACPMAPS (Fermilab).

dimensional grid. The memory of each node is shared with its neighbors. This allows nearest neighbor data to be accessed at the same speed as local data. Fourier transforms would have to be performed by a somewhat more awkward bucket brigade.

IBM's GF11 possesses a powerful, programmable three stage switch which allows any node to access any other node's memory at the same rate of speed at which it accesses its own. The 16 memory boards of APE are arranged in a linear array. One or two timeslices are usually assigned per board. A barrel shifter allows the CPUs to communicate synchronously with their local memories, or with memories a fixed distance in time away. This in principle allows rapid access to arbitrarily nonlocal memory, but in a less flexible way than GF11. This design is difficult to scale to a very large number of nodes, and for their next machine, the APE collaboration is planning to use a three dimensional regular, local grid.

Communication in the Fermilab and Caltech machines is asynchronous. On the Fermilab machine, communication is handled by sixteen port crossbar switchcrates which allow 20 megabyte per second communication per port. Clusters of 8-12 nodes are connected in a single switchcrate; the clusters are connected, for example, in a hypercube. The nodes in the Caltech machine are connected in a 2^n hypercube. Each node can send messages to its neighbors at a rate of 2 Mbytes/sec., with a relatively large latency of 150 μsec. for each transfer. Caltech is developing a powerful "hyperswitch" which is to be much faster and which will transparently handle nonlocal routing of messages.

Programming strategies. The various projects probably differ most noticeably in their ways of writing user programs. Approaches range from hand generated microcode for the floating point part of each user program, to the construction of limited machine specific compilers, to the use of ordinary Fortran and C for the basis of the software.

The floating point parts of all programs for the Columbia machine must be written in microcode by someone who is an expert on the machine architecture. The APE and IBM projects have invested a lot of effort in software to generate microcode. IBM's microcode is generated by calls from C to a limited but powerful set of subroutines. Preliminary indications are that the microcode produced by this software will be very efficient. The microcode for APE is generated from a new compiled language (called APEse) which resembles a limited form of Fortran or C. There is also an extensive software environment including a profiler and a symbolic debugger. APE's published work in the past year has included work on several different topics, all of which has employed new methods and techniques, which is an impressive demonstration of the usability of the software.

The Caltech and Fermilab machines have based their software on the commercial Fortran and C compilers available with the Weitek XL chip set. With

the Caltech machine, subroutines are provided for message passing and synchronization. The Fermilab software defines new data types in C (lattices, sites, fields, etc.) which are fundamental to grid based scientific problems. User programs are written in C augmented by these new data types and subroutines operating on them. System subroutines handling these data types (to access field data, for example) automatically handle the multinode structure of the machine in a way invisible to the user.

4 The future

Commercial parallel machines. In 1985, the Columbia group using their own dedicated machine was barely able to keep up with the California group who used conventional supercomputers, but by the 1988 lattice conference, large scale QCD calculations were dominated by results from the dedicated machines built at Columbia and at Rome. Dedicated machines are now under construction which are ten times more powerful still, which will deliver computing power which can be matched only by hundreds of millions of dollars worth of Cray or Cyber time. The growth path of Cray and Cyber style supercomputers is well established and it seems unlikely that this style of architecture will catch up in cost effectiveness with the massively parallel approach. It is an interesting question whether commercial companies be able to take advantage of the experience gained by the lattice gauge computer builders and create commercially viable machines of this type. This type of architecture is in principle effective not only for theoretical physics problems but for a wide variety of practical grid-based problems of enormous economic importance. The architectures used in lattice machines can easily be commercialized. The obstacles are primarily in software, and perhaps in sociology. A software interface convenient enough to attract a sufficiently large group of commercial users is necessary.

There are at least two commercial parallel computers which have been used for lattice calculations and it will be interesting to see how they fare commercially. The Meiko computing surface is composed of a large array of T800 Transputers, which were developed by the Inmos corporation and partially underwritten by the Common Market supported Esprit project[18]. Transputers contain a microprocessor and a floating point unit capable of several megaflops. The nodes operate in MIMD fashion and are arranged in a two dimensional array. It was originally designed to programmed in the parallel language OCCAM, but Fortran and C compilers have been developed for it.

The Connection Machine II of Thinking Machines resembles the lattice machines described above in having floating point power based on a large number of Weitek chips[19]. Its architecture is quite a bit different from those of the previous machines; its speed is limited almost entirely by bandwidth to

memory. It is programmed in versions of Fortran, C, and LISP with parallel extensions. A machine costing around $5 million was able to run a QCD problem at a sustained (not peak) speed of 2 gigaflops. Although still not as cost effective as the experimental machines, this would make it more than an order of magnitude more cost effective than conventional supercomputers.

The next generation of lattice machines. The next few years will see large increases in the power of these machines arising from improved components, architectural fine tuning, and larger pricetags. The continuing improvement in the performance of chips, which has been responsible for most of the dramatic increases in the power and cost-effectiveness of general purpose computers over the last two or three decades, will make possible more powerful massively parallel computers. The highly programmable Weitek XL chip sets used in the Fermilab and Caltech machines could be replaced by high performance RISC microprocessors. An example is the Intel I860 chip which, we are told, will ultimately have a peak speed of 100 megaflop, or five times the power of the XL chip set, at a similar price (around $1000 or less). Similarly, the fast floating point chips used in other machines ought to double in speed and halve in price in the next few years. Today's one-megabit memory chips will be replaced with four-megabit chips. A factor of around four in cost effectiveness is thus to be expected from evolution of the components over the next few years. Architectural fine tuning may produce improvements in cost effectiveness by factors of perhaps two to four. For example, if raw CPU power rather than memory size or bandwidth to memory is thought to be the limiting factor, the number of floating point chips on most of the existing machines could be increased by a factor of two or more, while increasing the cost per board by a much smaller factor. Finally, the budgets of at least some of the projects are likely to be scaled up by an order of magnitude. The bottom line is that the next generation of machines is likely to have peak speeds in the 0.1 to 1.0 teraflop range, at pricetags of $5-20 million.

The plans for one such project have already been announced. APE100, the successor to APE, will consist of a large array of boards connected in a three dimensional array.[20] Each board will contain 8 nodes with peak speeds of 32 megaflops each, based on a semicustom floating point chip from LSI. The planned peak speed of the machine is 100 gigaflops.

The generation of machines now under construction will have the memory to explore lattice sizes in the range $20^4 - 32^4$. The next generation discussed above will be able to explore the range $40^4 - 64^4$. Since the cost effectiveness of chips improves at the rate of at least an order of magnitude per decade, it is reasonable to expect lattices with sizes up to 128^4 to be investigated during the 1990's, if necessary. Whether these machines will be effectively limited to smaller lattice sizes by CPU requirements depends on the progress of algorithms. I believe that it is realistic to expect algorithmic progress to

continue to keep pace with hardware progress.

After the breakeven point. The foregoing discussion has focussed on computing requirements to perform the simplest QCD calculations: the calculation of the masses of the light hadrons to, let's say, 30 MeV. What will happen to the demand for computing for theoretical physics when this goal is accomplished? It takes no imagination to predict that some people will immediately begin to try to calculate the spectrum to 10 MeV, which will require one to two orders of magnitude more computing power if done by brute force. Since experimenters already know the masses of the light hadrons very well, these calculations serve more to test lattice gauge methods than to learn about physics. Much more important will be the application of lattice gauge theory to obtain new physics information, for example, the hadronic matrix elements necessary to extract the Kobayashi–Maskawa angles from hadronic data. When lattice methods are reliable and well established, the range of useful applications will be for all practical purposes unlimited. At the present time, the theoretical physics community is expending a great deal of time and money on lattice gauge theory with only a modest return. The calculation of the hadron spectrum with good error bars, whether it comes later this year or later this decade, will greatly increase the demand for theoretical computing rather than end it.

Acknowlegments

I would like thank the members in the projects described here for conversations and correspondence.

References

[1] R. B. Pearson, J. L. Richardson, and D. Toussaint, J. Comp. Phys. **51** (1983) 241.

[2] A. Hoogland et al., J. Comp. Phys. **51** (1983) 250.

[3] D. J. Wallace, Phys. Repts. **103** (1985) 191.

[4] Y. Iwasaki et al., Comp. Phys. Comm. **49** (1988) 449.

[5] E. Brooks et al., Phys. Rev. Lett. **52** (1984) 2324. J. C. Peterson et al., in the proceedings of *The 1985 Conference on Supercomputing* (IEEE).

[6] N. H. Christ and A. E. Terrano, IEEE Trans. Comput. **33** (1984) 344; F. Butler, in *Lattice 88*, Proceedings of the 1988 Conference on Lattice Field Theory, edited by A. S. Kronfeld and P. B. Mackenzie, to be published in Nuc. Phys. B (Proc. Suppl.)

[7] J. Beteem, M. Denneau, and D. Weingarten, in *The 12th Annual International Symposium on Computer Architecture*, (IEEE Computer Society Press, Silver Spring, MD, 1985); J. Sexton, in *Lattice 88*, op cit.

[8] P. Bacilieri et al., in *Computing in High Energy Physics*, edited by L. O. Hertzberger and W. Hoogland, (North Holland, Amsterdam, 1986).

[9] P. B. Mackenzie et al., in *Field Theory on the Lattice*, edited by A. Billoire et al., Nuc. Phys. B (Proc. Suppl.) **4** (1988); M. Fischler and G. Hockney et al., in *Lattice 88*, op cit.

[10] N. H. Christ, in *Lattice 88*, op cit.

[11] M. Lüscher, Commun. Math. Phys. **104** 177 (1986).

[12] G. Parisi, Phys. Reports **103** (1984) 203.

[13] S. A. Gottlieb et al., Phys. Rev. Lett. **55** 1958 (1985).

[14] N. H. Christ and A. E. Terrano, Phys. Rev. Lett. **56** 111 (1985).

[15] See the talk by Tony Kennedy in these proceedings.

[16] S. Duane, A. D. Kennedy, B. J. Pendleton and D. Roweth, Phys. Lett. **195B,** 2 (1987).

[17] See, for example, the talk by Tom Nash in these proceedings.

[18] See the talk by David Wallace in these proceedings.

[19] L. W. Tucker and G. G. Robertson, Computer (August 1988) 26.

[20] E. Remiddi, in *Lattice 88*, op cit.