Lattice QCD: Commercial vs. Home-grown Parallel Computers

Clive F. Baillie

Caltech Concurrent Computation Program,
California Institute of Technology,
Pasadena, CA 91125, USA

Abstract

Numerical simulations of Lattice QCD have been performed on practically every computer, since its inception almost twenty years ago. Lattice QCD is an ideal problem for parallel machines as it can be easily domain decomposed. In fact, the urge to simulate QCD has led to the development of several home-grown parallel "QCD machines", in particular the Caltech Cosmic Cube, the Columbia Machine, IBM's GF11, APE in Rome and the Fermilab Machine. These machines were built because, at the time, there were no commercial parallel computers fast enough. Today however the situation has changed with the advent of computers like the Connection Machine 2 and the Ncube 2. Herein, I shall explain why Lattice QCD is such a parallel problem and compare two large-scale simulations of it – one on the commercial Connection Machine and the other on the latest Caltech/JPL hypercube.

1. Introduction

Quantum Chromo-dynamics (QCD) simulations are consuming vast amounts of computer time these days, and promise to do so for at least the foreseeable future. The background for these calculations is two decades of great progress in our understanding of the basic particles and forces. Over time, the particle physics community has developed an elegant and satisfying theory which is believed to describe all the particles and forces which can be produced in today's high energy accelerators. The basic components of the so-called "Standard Model" are matter particles (quarks and leptons), and the forces through which they interact (electromagnetic, weak and strong). The electromagnetic force is the most familiar, and also the first to be understood in detail. The weak force is less familiar, but manifests itself in processes such as nuclear beta-decay, for example. This piece of the Standard Model is now called the electroweak sector. The third part of the Standard Model is the QCD, the theory of the strong force, which binds quarks together into "hadrons", such as protons, neutrons, pions, and a host of other particles. The strong force is also responsible for the fact that protons and neutrons bind together to form the atomic nucleus. Currently we know of five types of quark (referred to as "flavors"): up, down, strange, charm and bottom; and expect at least one more (top) to show up soon. In addition to having a "flavor", quarks can carry one of three possible charges known as "color" (this has nothing to do with color in the macroscopic world!), hence Quantum *Chromo*-dynamics. The strong "color" force is mediated by particles called gluons, just as photons mediate light in electromagnetism. Unlike photons, though, gluons themselves carry a "color" charge and therefore interact with one another. This means that QCD is an extremely nonlinear theory which cannot be solved analytically. Hence we resort to numerical simulations.

2. Lattice QCD

To put QCD on a computer we proceed as follows. The four-dimensional space-time continuum is replaced by a four-dimensional hypercubic periodic lattice, of size $N = N_s \times N_s \times N_s \times N_t$ with the quarks living on the sites and the gluons living on the links of the lattice. N_s is the spatial and N_t is the temporal extent of the lattice. The gluons are represented by 3×3 complex $SU(3)$ matrices associated with each link in the lattice. This link matrix describes how the "color" of a quark changes as it moves from one site to the next. The action functional for the purely gluonic part of QCD is

$$S_G = \beta \sum_P (1 - \frac{1}{3} ReTrU_P), \qquad (1)$$

0-8186-2113-3/90/0000/0397$01.00 © 1990 IEEE

where

$$U_P = U_{i,\mu} U_{i+\mu,\nu} U^{\dagger}_{i+\nu,\mu} U^{\dagger}_{i,\nu} \qquad (2)$$

is the product of link matrices around an elementary square or plaquette on the lattice – see Figure 1. Essentially all of the time in QCD simulations of gluons only is spent multiplying these $SU(3)$ matrices together. The code for this, shown in the Appendix, reveals that its main component is the $a \times b + c$ kernel (which most supercomputers can do very efficiently). The partition function for full lattice QCD including quarks is

$$Z = \int D\psi D\bar{\psi} DU \exp(-S_G - \bar{\psi}(\not{D} + m)\psi), \quad (3)$$

where $\not{D} + m$ is a large sparse matrix the size of the lattice squared. Unfortunately, since the quark variables ψ are anticommuting Grassmann numbers, there is no simple representation for them on the computer. Instead they must be integrated out, leaving a highly non-local fermion determinant:

$$Z = \int DU \det(\not{D} + m) \exp(-S_G), \qquad (4)$$

This is the basic integral one wants to evaluate numerically.

Note that the lattice is a mathematical construct used to solve the theory—at the end of the day, the lattice spacing a must be taken to zero to get back to the continuum limit. The lattice spacing itself does not show up explicitly in the partition function Z above. Instead the parameter $\beta = 6/g^2$, which plays the role of an inverse temperature, ends up controlling the lattice spacing $a(\beta)$. To take the continuum limit $a \to 0$ of lattice QCD one tunes $g \to 0$, or $\beta \to \infty$. Typical values used in simulations these days range from $\beta = 5.3$ to $\beta = 6.0$. This corresponds to $a \approx .1$ Fermi $= 10^{-16}$ meter. Thus at current values of β a lattice with $N_s = 20$ will correspond to a physical box about 2 Fermi on an edge, which is large enough to hold one proton without crushing it too much in the finite volume. Still the spacing $a = .1$ Fermi is not fine enough that we are close to the continuum limit. One can estimate that we still need to shrink the lattice spacing by something like a factor of 4, leading to an increase of a factor 4^4 in the number of points in the lattice in order to keep the box the same physical volume.

The biggest stumbling block preventing a large increase in the number of lattice points is the presence of the determinant $det(\not{D} + m)$ in the partition function. Physically, this determinant arises from closed quark loops. The simplest way to proceed is to ignore these quark loops and work in the so-called "quenched" or "pure gauge" approximation. The quenched approximation assumes that the whole effect of quarks on gluons can be absorbed in a redefinition of the gluon interaction strength. Operationally, one generates gluon field configurations using only the pure gauge part of the action, and then computes the observables of interest in those backgrounds. For some quantities this may be a reasonable approximation. It is certainly orders of magnitude cheaper, and for this reason, most all simulations to date have been done using it. To investigate the fully realistic theory, though, one has to go beyond the quenched approximation and tackle the fermion determinant.

There have been many proposals for dealing with the determinant. The first algorithms tried to compute the change in the determinant when a single link variable was updated. This turned out to be prohibitively expensive. Today, the preferred approach is the so-called "Hybrid Monte Carlo" algorithm [1] . The basic idea is to invent some dynamics for the variables in the system in order to evolve the whole system forward in (simulation) time and then do a Metropolis accept/reject for the entire evolution on the basis of the total energy change. The great advantage is that the whole system is updated at one fell swoop. The disadvantage is that if the dynamics is not correct then the acceptance will be very small. Fortunately (and this one of very few fortuitous happenings where fermions are concerned) good dynamics can be found: the Hybrid algorithm [2] . This is a neat combination of the deterministic microcanonical method [3] and the stochastic Langevin method [4] . which yields a quickly-evolving, ergodic algorithm for both gauge fields and fermions. The computational kernel of this algorithm is the repeated solution of systems of equations of the form

$$(\not{D} + m)\phi = \eta, \qquad (5)$$

where ϕ and η are vectors which live on the sites of the lattice. To solve these equations one typically uses conjugate gradient or one of its cousins, since the fermion matrix $(\not{D} + m)$ is sparse. For more details, see [5] . Such iterative matrix algorithms have as their basic component the $a \times b + c$ kernel, so again computers which do this efficiently will run QCD both with and without fermions well.

However one generates the gauge configurations U, using the quenched approximation or not, one then has to compute the observables of interest. For observables involving quarks one runs into expressions like $\langle \psi(x)\bar{\psi}(y) \rangle$ involving pairs of quark fields at different points. Again because of the Grassmann

nature of fermions fields, one has to express this quantity as

$$\langle \psi(x)\bar{\psi}(y)\rangle = (\not{D} + m)^{-1}_{xy}. \qquad (6)$$

And again one computes as many columns of the inverse as needed by solving systems equations like (5) above. For simulations of full QCD with quark loops, this phase of the calculation is a small overhead, while for quenched calculations it is the dominant part. So whether quenched or not, most of the computer time is spent in applying conjugate gradient to solve large systems of linear equations.

3. Home-grown QCD Machines

Today the biggest resources of computer time for research are the conventional supercomputers at the NSF and DOE centers. The centers are continually expanding their support for lattice gauge theory, but it may not be long before they are overtaken by several dedicated efforts involving concurrent computers. It is a revealing fact that the development of most high performance parallel computers—the Caltech Cosmic Cube, the Columbia Machine, IBM's GF11, APE in Rome, the Fermilab Machine—was actually motivated by the desire to simulate lattice QCD.

Geoffrey Fox and Chuck Seitz at Caltech built the first hypercube computer, the Cosmic Cube or Mark I, in 1983 [6] . It had 64 nodes, each of which was an Intel 8086/87 microprocessor with 128 KB of memory, giving a total of about 2 Mflops (measured for QCD). This was quickly upgraded to the Mark II hypercube with faster chips, twice the memory per node and twice the number of nodes in 1984 [7] . Now QCD is running at 600 Mflops sustained on the latest Caltech hypercube: the 128-node Mark IIIfp (built by JPL) [8] . Each node of the Mark IIIfp hypercube contains two Motorola 68020 microprocessors, one for communication and the other for calculation, with the latter supplemented by one 68881 coprocessor and a 32-bit Weitek floating point processor.

Norman Christ and Tony Terrano at Columbia built their first parallel computer for doing lattice QCD calculations in 1984 [9] . It had 16 nodes, each of which was an Intel 80286/87 microprocessor plus a TRW 22-bit floating point processor with 1 MB of memory, giving a total peak performance of 256 Mflops. This was improved in 1987 using Weitek rather than TRW chips so that 64 nodes give 1 Gflops peak [10] . Very recently, Columbia have finished building their third machine: a 256-node 16 Gflops lattice QCD computer [11] .

Don Weingarten at IBM has been building the GF11 since 1984—it is expected he will start running in production in 1990 [12] . The GF11 is an SIMD machine comprising 576 Weitek floating point processors, each performing at 20 Mflops to give the total 11 Gflops peak implied by the name.

The APE (Array Processor with Emulator) computer is basically a collection of 3081/E processors (which were developed by CERN and SLAC for use in high energy experimental physics) with Weitek floating point processors attached [13] . However, these floating point processors are attached in a special way—each node has four multipliers and four adders in order to optimize complex $a \times b + c$ calculations, which form the major component of all lattice QCD programs. This means that each node has a peak performance of 64 Mflops. The first, small machine—Apetto—was completed in 1986 and had 4 nodes yielding a peak performance of 256 Mflops. Currently, they have a second generation of this with 1 Gflops peak from 16 nodes. By 1992, the APE collaboration hopes to have completed the 100 Gflops 4096-node "Apecento" [14] .

Not to be outdone, Fermilab is also using its high energy experimental physics emulators in constructing a lattice QCD machine for 1991 with 256 of them arranged as a 2^5 hypercube of crates, with 8 nodes communicating through a crossbar in each crate [15] . Altogether they expect to get 5 Gflops peak performance.

These performance figures are summarized in Table 1. The "real" performances are the actual performances obtained on QCD codes; in Figure 2 we plot these as a function of the year the QCD machines started to produce physics results. The surprising fact is that the rate of increase is very close to exponential, yielding a factor of ten every two years! On the same plot we show our estimate of the computer power needed to redo this year's quenched calculations on a 128^4 lattice. This estimate is also a function of time, due to algorithm improvements. Extrapolating both lines, we see the outlook for lattice QCD is rather bright. Reasonable results for the "harder" physical observables should be available within the quenched approximation in the mid-90's. With the same computer power we will be able to redo today's quenched calculations using dynamical fermions (but still on today's size of lattice). This will tell us how reliable the quenched approximation is. Finally, results for the full theory with dynamical fermions on a 128^4 lattice should follow early in the next century (!), when computers are two or three orders of magnitude more powerful again.

Table 1
Peak and real performances in Mflops
of "homebrew" QCD machines

Computer	Year	Peak	Real
Caltech I	1983	3	2
Caltech II	1984	9	6
Caltech III	1989	2000	600
Columbia I	1984	256	20
Columbia II	1987	1000	200
Columbia III	1990	16000	6000
IBM GF11	1990	11000	10000*
APE I	1986	256	20
APE II	1988	1000	200
APE III	1992	100000	20000*
Fermilab	1991	5000	1200*

* All real times are measured except these predicted ones.

With this brief review in hand, we now turn to a comparison of QCD running on one home-grown computer – the Caltech/JPL Mark IIIfp hypercube – with the commercially available TMC Connection Machine 2.

4. QCD on the Caltech/JPL Mark IIIfp

Decomposing QCD onto a d-dimensional hypercube distributed memory computer (with 2^d nodes) is particularly simple. One takes the $N = M2^d$ lattice and splits it up into 2^d sublattices, each containing M sites, one of which is placed in each node. Due to the locality of the action, eq. (2), it is possible to assign the sublattices so that each node needs only to communicate with others to which it is directly connected in hardware. As a result of this fact the characteristic timescale of the communication, t_{comm}, is minimal and corresponds to roughly the time taken to transfer a single $SU(3)$ matrix from one node to its neighbor. Conversely we can characterize the calculational part of the algorithm by a timescale, t_{calc}, which is roughly the time taken to multiply together two $SU(3)$ matrices. For all hypercubes built *without* floating point accelerator chips $t_{comm} << t_{calc}$ and hence QCD simulations are extremely "efficient", where efficiency is defined by the relation

$$\epsilon = \frac{T_1}{kT_k}, \qquad (7))$$

where T_k is the time taken for k processors to perform the given calculation. Typically such calculations have efficiencies in the range $\epsilon \geq .90$ which means they are ideally suited to this type of computation since doubling the number of processors approximately halves the total computational time required for solution. However, as we shall see, the picture changes dramatically when fast floating point chips are used; then $t_{comm} \simeq t_{calc}$ and one must take some care in coding to obtain maximum performance.

QCD simulations have been done on all the Caltech hypercubes; the most recent being a high statistics, large lattice study of the string tension in pure gauge QCD on the Mark IIIfp [16] . For this the 128-node hypercube performs at 0.6 Gflops. As each node runs at 6 Mflops this corresponds to a speedup of 100, and hence an efficiency of 78%. These figures are for the most highly optimized code. The original version of the code written in C ran on the Motorola chips at 0.085 Mflops and on the Weitek chips at 1.3 Mflops. The communication time, which is roughly the same for both, is less than a 2% overhead for the former but nearly 30% for the latter. When the computationally intensive parts of the calculation are written in assembly code for the Weitek this overhead becomes almost 50%. This 0.9 msec of communication, shown in lines 2 and 3 in Table 2, is dominated by the hardware/software message startup overhead (latency), because for the Mark IIIfp the node to node communication time, t_{comm}, is given by

$$t_{comm} \simeq (150 + 2 * W) \quad \mu\text{sec},$$

where W is the number of words transmitted. To speed up the communication we update all even (or odd) links (8 in our case) in each node, allowing us to transfer 8 matrix products at a time, instead of just sending one in each message. This reduces the 0.9 msec by a factor of

$$\frac{8 * (150 + 18 * 2)}{150 + 8 * 18 * 2} = 3.4$$

to 0.26 msec. On all hypercubes with fast floating point chips – and on most hypercubes without for less computationally intensive codes – such vectorization of communication is often important. In Figure 3, the speedups for many different total lattice sizes are shown. For the largest lattice size, the speedup is 100 on the 128-node. The speedup is almost linear in number of nodes. As the total lattice volume increases, the speedup increases, because the ratio of calculation/communication increases. For more information on this performance analysis, see [17] .

5. QCD on the TMC Connection Machine 2

The Connection Machine is also very well suited for large-scale simulations of QCD. The CM-2 is a distributed-memory, Single-Instruction Multiple-Data (SIMD) massively-parallel processor comprising up to 65536 (64K) processors [18] . Each processor consists of an arithmetic-logic unit (ALU), 8 or 32 Kbytes of random-access memory (RAM) and a router interface to perform communications among the processors. There are sixteen processors and a router per custom VLSI chip, with the chips being interconnected as a twelve-dimensional hypercube. Communications among processors within a chip work essentially like a cross-bar interconnect. The router can do general communications but we require only local ones for QCD so we use the fast nearest-neighbor communication software called NEWS. The processors deal with one bit at a time, therefore the ALU can compute any two boolean functions as output from three inputs, and all data paths are 1-bit wide. In the current version of the Connection Machine (the CM-2) groups of 32 processors (two chips) share a 32-bit (or 64-bit) Weitek floating point chip, and a transposer chip which changes 32 bits stored bit-serially within 32 processors into 32 32-bit words for the Weitek, and vice versa.

The high-level languages on the CM, such as *Lisp and CM-Fortran, compile into an assembly language called Paris (Parallel Instruction Set). Paris regards the 64K bit-serial processors as the fundamental units in the machine, and so well represents the global aspects of the hardware. However, floating point computations are not very efficient in the Paris model. This is because in Paris 32-bit floating point numbers are stored "field-wise", that is, successive bits of the word are stored at successive memory locations of each processors memory. However, 32 processors share one Weitek chip which deals with words stored "slice-wise", that is, stored across the processors, one bit in each. Therefore to do a floating point operation, Paris loads in the field-wise operands, transposes them slice-wise for the Weitek (using the transposer chip), does the operation and transposes the slice-wise result back to field-wise for memory storage. Moreover, every operation in Paris is an atomic process, that is, two operands are brought from memory and one result is stored back to memory so no use is made of the Weitek registers for intermediate results. Hence, to improve the performance of the Weiteks, a new assembly language called CMIS (CM Instruction Set) has been written, which models the local architectural features much better. In fact, CMIS ignores the bit-serial processors and thinks of the machine in terms of the Weitek chips. Thus data can be stored slice-wise, eliminating all the transposing back and forth. CMIS allows effective use of the Weitek registers, creating a memory hierarchy, which combined with the internal buses of the Weiteks offers increased bandwidth for data motion.

Currently, the Connection Machine is the most powerful commercial QCD machine available: the "Los Alamos collaboration" is running full QCD at a sustained rate of almost 2 Gflops on a 64K CM-2 [19] . As was the case for the Mark IIIfp hypercube, in order to obtain this performance one must resort to writing assembly code for the Weitek chips *and* for the communication. Our original code, written entirely in *Lisp, achieved around 1 Gflops. As shown in Table 3, this code spends 34% doing communication. When we rewrote the most computationally intensive part in the assembly language CMIS, this rose to 54%. In order to obtain maximum performance we are now rewriting the communication part of our code to make use of "multi-wire NEWS" which will allow us to communicate in all 8 directions on the lattice simultaneously thereby reducing the communication time by a factor of 8 and speeding up the code by another factor of 2.

6. Conclusions

It is interesting to note that when the various groups began building their "homebrew" QCD machines it was clear that they would out-perform all commercial (traditional) supercomputers; however, now that commercial parallel supercomputers have come of age [20] the situation is not so obvious.

On the original versions of both commercial and home-grown parallel computers (without fast floating point chips) one could get good performance from one's favorite high-level language. Now, however, as most of these machines do have fast floating point hardware, one must resort to lover-level assembly programming to obtain maximum performance. Having done just that, we are running QCD at 0.6 Gflops on the Caltech/JPL Mark IIIfp hypercube and at 1.65 Gflops on the TMC Connection Machine 2.

References

[1] S. Duane, A.D. Kennedy, B.J. Pendleton and D. Roweth, Hybrid Monte Carlo, *Phys. Lett.* **B195**, 216 (1987).

[2] S. Duane, Stochastic Quantization versus the Microcanonical Ensemble: Getting the best of both worlds, *Nucl. Phys.* **B257**, 652 (1985).

[3] D. Callaway and A. Rahman, Lattice Gauge Theory in the Microcanonical Ensemble, *Phys. Rev.* **D28**, 1506 (1983); J. Polonyi and H.W. Wyld, Microcanonical Simulation of Fermionic Systems, *Phys. Rev. Lett.* **51**, 2257 (1983).

[4] G. Parisi and Y. Wu, Perturbation Theory without Gauge Fixing, *Sci. Sin.* **24**, 483 (1981); G.G. Batrouni, G.R. Katz, A.S. Kronfeld, G.P. Lepage, B. Svetitsky and K.G. Wilson, Langevin Simulations of Lattice Field Theories, *Phys. Rev.* **D32**, 2736 (1985).

[5] R. Gupta, G.W. Kilcup and S.R. Sharpe, Tuning the Hybrid Monte Carlo Algorithm, *Phys. Rev.* **D38**, 1278 (1988).

[6] C. L. Seitz, The Cosmic Cube, *Comm. of the ACM* **28**, 22 (1985).

[7] J. Tuazon *et al*, The Caltech/JPL Mark II hypercube concurrent processor, In *IEEE 1985 Conference on Parallel Processing*, St. Charles, Illinois, August 1985.

[8] J. C. Peterson *et al*, The Mark III hypercube ensemble concurrent computer, In *IEEE 1985 Conference on Parallel Processing*, St. Charles, Illinois, August 1985.

[9] N. H. Christ and A. E. Terrano, A Very Fast Parallel Processor, *IEEE Trans. on Computers* **C-33**, 344 (1984).

[10] F. Butler, Status of the Columbia Parallel Processors, *Nucl. Phys.* **B (Proc. Suppl.) 9**, 557 (1989).

[11] N. H. Christ, Status of the Columbia 256-node Parallel Supercomputer, In *Proc. of the Int. Symp. Lattice 89*, Capri, September 1989, to appear in *Nucl. Phys.* **B (Proc. Suppl.)** (1990).

[12] J. Beetem, M. Denneau and D. Weingarten, The GF11 Supercomputer, In *IEEE Proc. of the 12th Annual Int. Symp. on Computer Architecture*, IEEE Computer Society, Washington D.C., 1985; D. Weingarten, Progress Report on the GF11 Project, In *Proc. of the Int. Symp. Lattice 89*, Capri, September 1989, to appear in *Nucl. Phys.* **B (Proc. Suppl.)** (1990).

[13] M. Albanese *et al*, The APE Computer: An Array Processor Optimized for Lattice Gauge Theory Simulations, *Comp. Phys. Commun.* **45**, 345 (1987).

[14] R. Tripiccione, Machines for Theoretical Physics, In *Proc. of the Int. Symp. Lattice 89*, Capri, September 1989, to appear in *Nucl. Phys.* **B (Proc. Suppl.)** (1990).

[15] T. Nash *et al*, The Fermilab Advanced Computer Program Multi-Processor Project, In *Proc. Conf. Computing in High Energy Physics*, Amsterdam, June 1985 (North-Holland, Amsterdam, 1986); M. Fischler, Hardware and System Software Requirements for Algorithm Development, In *Proc. of the Int. Symp. Lattice 89*, Capri, September 1989, to appear in *Nucl. Phys.* **B (Proc. Suppl.)** (1990).

[16] H.-Q. Ding, C. F. Baillie and G. C. Fox, Calculation of the heavy quark potential at large separation on a hypercube parallel computer, to appear in Phys. Rev. D. (1990).

[17] H.-Q. Ding, The Mark IIIfp Hypercube: Performance of a QCD code, Caltech preprint C3P-799, submitted to *Comp. Phys. Commun.* (1990).

[18] W. D. Hillis, *The Connection Machine* (MIT Press, Cambridge, MA, 1985); Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary*, TMC Technical Report HA87-4, Cambridge, MA, 1987.

[19] C. F. Baillie, R. G. Brickner, R. Gupta and S. L. Johnsson, QCD with dynamical fermions on the Connection Machine, In *Proc. of Supercomputing 89*, Reno, Nevada (ACM Press, New York, 1989).

[20] G. C. Fox, Parallel Computing Comes of Age: Supercomputer Level Parallel Computations at Caltech, *Concurrency: Practice and Experience* **1**, 1 (1989).

Table 2
Link update time (msec) on Mark IIIfp node
for various levels of programming

Programming level	Calc. time	Comm. time	Total time	Mflops
Motorola MC68020/68881 in C	52	0.86	53	0.085
Weitek XL all in C	2.25	0.90	3.15	1.4
Weitek XL parts in Assembly	0.94	0.90	1.84	2.4
Weitek XL Assembly, vec. comm.	0.94	0.26	1.20	3.8
Weitek XL Assembly, no comm.	0.94	0.0	0.94	4.8

Table 3
Fermion update time (sec) on 64K Connection Machine
for various levels of programming

Programming level	Calc. time	Comm. time	Total time	Mflops
All in *Lisp	8.7	4.5	13.2	900
Inner loop in CMIS	3.3	3.9	7.2	1650
Multi-wire CMIS†	< 3.3	0.5	< 3.8	> 3100

† projected numbers

Fig. 1.
Illustration of plaquette calculation

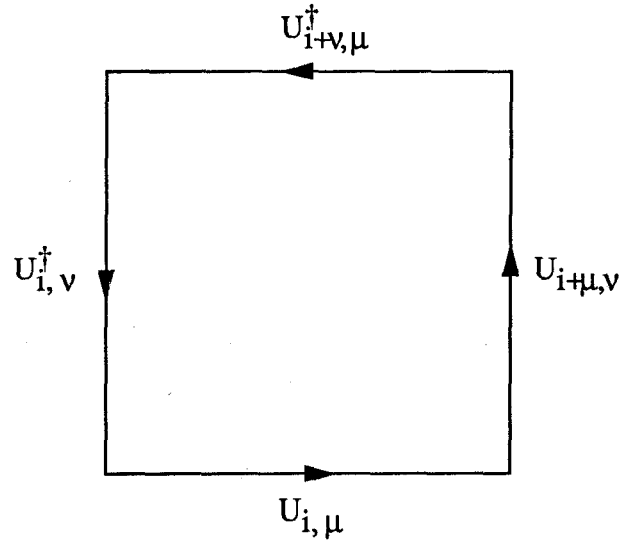

Fig. 2. Computational power of QCD machines

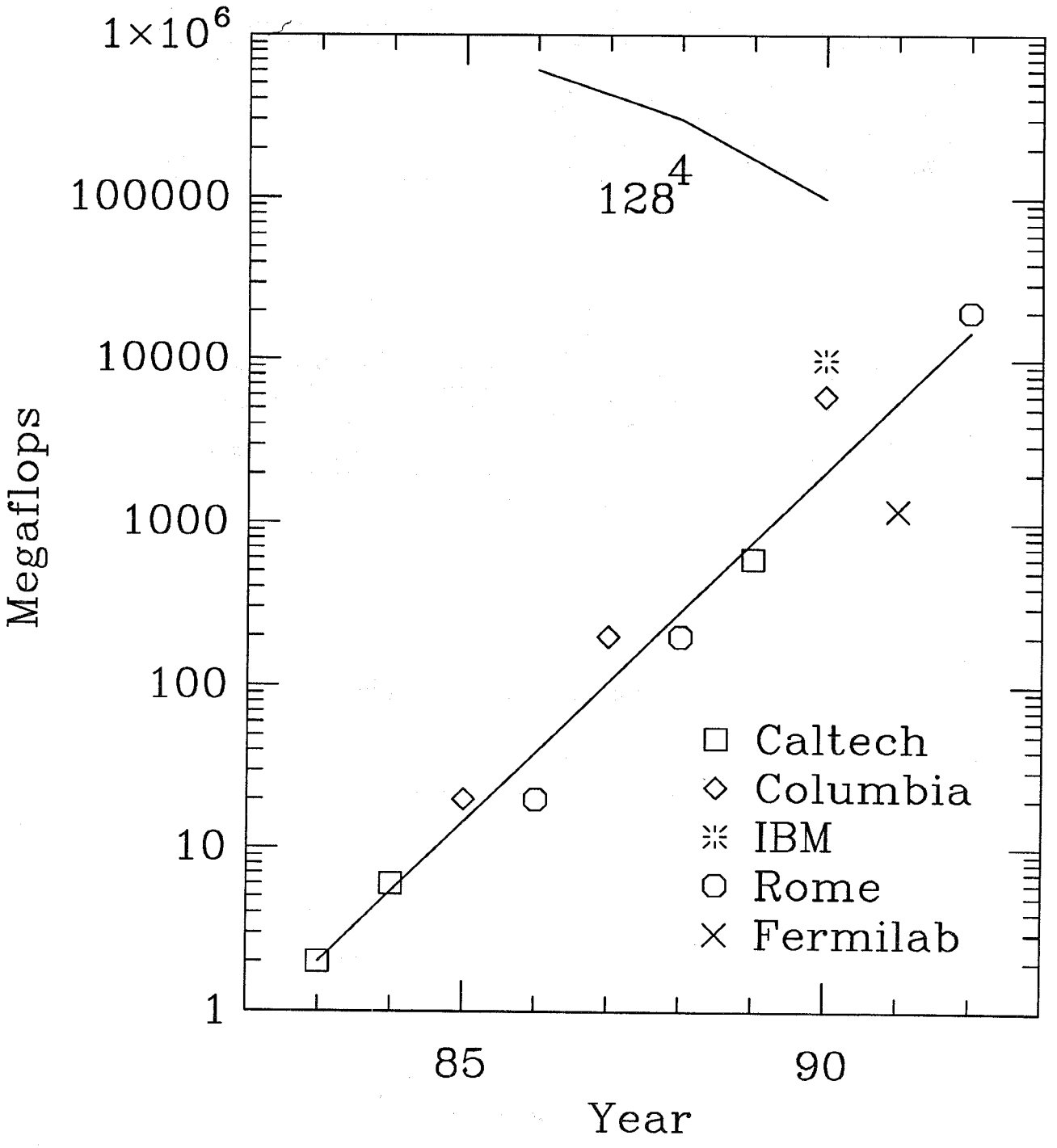

Fig. 3. Speedup of QCD code on Mark IIIfp