UCRL-JRNL-229921

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Massively Parallel QCD

Ron Soltz, Pavlos Vranas, Matthias Blumrich, Dong Chen, Alan Gara, Mark Giampap, Philip Heidelberger, Valentina Salapura, James Sexton

April 15, 2007

IBM Journal of Research and Development

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Massively Parallel QCD

Pavlos Vranas, Matthias Blumrich, Dong Chen, Alan Gara,
Mark Giampapa, Philip Heidelberger, Valentina Salapura, James C. Sexton
IBM T.J. Watson Research Center
Yorktown Heights, New York

Ron Soltz
Lawrence Livermore National Laboratory
Livermore, California

Gyan Bhanot
Rutgers University
Piscataway, New Jersey

Abstract

The theory of the strong nuclear force, Quantum Chromodynamics (QCD), can be numerically simulated from first principles on massively-parallel supercomputers using the method of Lattice Gauge Theory. We describe the special programming requirements of lattice QCD (LQCD) as well as the optimal supercomputer hardware architectures that it suggests. We demonstrate these methods on the BlueGene massively-parallel supercomputer and argue that LQCD and the BlueGene architecture are a natural match. This can be traced to the simple fact that LQCD is a regular lattice discretization of space into lattice sites while the BlueGene supercomputer is a discretization of space into compute nodes, and that both are constrained by requirements of locality. This simple relation is both technologically important and theoretically intriguing. The main result of this paper is the speedup of LQCD using up to 131,072 CPUs on the largest BlueGene/L supercomputer. The speedup is perfect with sustained performance of about 20% of peak. This corresponds to a maximum of 70.5 sustained TFlop/s. At these speeds LQCD and BlueGene are poised to produce the next generation of strong interaction physics theoretical results.

1. Introduction

Perhaps the best introduction to the theory of Quantum Chromodynamics (QCD) was given by Frank Wilczek. He described QCD as "…our most perfect physical theory." [1]. Wilczek was a co-recipient of the 2004 Nobel Prize in Physics for the discovery of the properties of QCD. In [1] he describes the properties of QCD that make it worthy of such a qualification: it embodies deep and beautiful principles (it is a relativistic quantum field theory), it provides algorithms to answer questions (one such algorithm is the subject of this paper), it has a wide scope (from nuclear physics to the genesis of the cosmos), it contains a wealth of phenomena (asymptotic freedom, confinement and many others), it has few parameters (and is therefore simple to describe), it is true (has been verified experimentally), and it lacks flaws (it is fully described by its definition).

QCD is the theory of sub-nuclear physics. All nuclear particles are made of elementary particles called quarks and gluons. The gluons mediate the strong nuclear force that binds the quarks together to form stable nuclear particles. The strong nuclear force is one of the four known physical forces, the others being electromagnetism, weak nuclear, and gravity. The strong nuclear force is also responsible for the interactions of nuclear particles and is therefore the basic ingredient of nuclear physics.

Nuclear matter constitutes about 90% of the visible universe and makes all things that surround us. But it was not always like this. It is believed that until about 10 ③sec after the big-bang, nuclear matter did not exist. The universe was so hot that quarks and gluons were in a plasma state, the quark-gluon-plasma. After 10 ③sec the temperature dropped below 2 trillion degrees Kelvin and the quark-gluon-plasma underwent a phase transition to stable nuclear matter. Currently at the Brookhaven National Laboratory, an enormously powerful accelerator collides heavy nuclei, in particular gold, at speeds near the speed of light. The Relativistic Heavy Ion Collider (RHIC) produces collisions so powerful as to re-create, if only for a brief moment, the conditions for the formation of the quark-gluon-plasma. Strong evidence suggests that RHIC has been successful in re-creating this state of matter that has not existed in our universe since 10 ③sec after its birth.

One of the most staggering properties of the theory is the behavior of its force. Quarks inside nuclear particles behave almost as if they were free. They experience very little of the strong nuclear force. This property is called asymptotic freedom. However, if one tries to "pull" a quark out of a nuclear particle, the force rapidly becomes extremely strong. A flux tube of gluons forms and forbids the quark from escaping. This property is called confinement. Nobody has ever observed a single isolated quark. It is remarkable that both of these dramatically opposite properties are described by a single theory. Furthermore, the theory of QCD is extremely simple in its mathematical description; it is described by a one line mathematical formula.

Many physical quantities can be calculated analytically for the case where the force is weak by using expansions around the zero-force point. However, the calculation of physical quantities becomes extremely difficult when the force is strong. Few analytical calculations are possible. This would have been a serious problem if it were not for the discovery of lattice gauge theory [2][3]. This, allows us to calculate physical quantities, such as the masses of nuclear particles or the characteristics of the thermal phase transition, at strong force by using computer simulations. Lattice gauge theory for QCD (LQCD) is described in Section 2.

Even so, it turns out that the computing requirement is enormous. As a result, LQCD has always required the largest supercomputers available to be able to make progress. In Section 3 we describe the special programming requirements of LQCD as well as the optimal supercomputer hardware architectures that it suggests. We demonstrate these methods on the BlueGene massively-parallel supercomputer, and argue that LQCD and the BlueGene architecture is a natural match. This can be traced to their curiously common properties. In Section 4 we present the main result of this paper: the sustained performance of LQCD on BG/L scaled to 131,072 CPUs, scaling with perfect speedup to 70.5 Tflop/s with sustained performance of about 20% of peak [4][5]. In Section 5 we give our conclusions.

For an introduction to quantum field theory and QCD, the reader is referred to the books [6] and [7]. For an introduction to lattice gauge theory and lattice QCD the reader is referred to the books [8], [9], and [10].

2. **Lattice QCD**

In this section we give a brief overview of the lattice gauge theory method [2][3] that allows QCD to be simulated on a computer.

QCD is defined in the continuous four-dimensional space-time. The quarks and gluons are described by fields over space-time. Fields are complex functions of the space-time coordinates and, loosely speaking, indicate the probability of a particle's existence at each coordinate. That probability is a complicated function of the fields. Specific local and global symmetries constrain these functions.

Since space-time is continuous, one would need an infinite amount of numbers to describe a field even in a finite region. But a computer is a finite machine with finite memory and computing capability. How then is it possible to simulate QCD?

The first step is to make space-time discrete by replacing it with a four-dimensional lattice. Typically the lattice is taken to be hypercubic. Because of the confinement property of QCD, only a small region of space that contains the nuclear particles is needed. Therefore, the lattice is finite and periodic/anti-periodic boundary conditions are typically implemented. The sites of the lattice are connected by links, where the distance along a link is called lattice spacing "a".

This rather brute-force approach could have destroyed the symmetry properties of the theory beyond recovery. In fact this is not a trivial matter at all. It turns out that by defining the quark fields on the lattice sites, while defining the gluon fields (also called gauge fields) on the lattice links, one of the most important symmetries of the theory is preserved. Local gauge invariance is exact.

Obviously the rotational and translational symmetries of continuous space-time are destroyed, among some other important symmetries. However, these symmetries are recovered as the lattice spacing "a" is decreased towards the zero lattice spacing limit. By repeating the calculation on lattices with more lattice points but smaller lattice spacing, one can extrapolate to the zero lattice spacing limit.

Therefore, the quark and gluon fields can be defined on a finite set of points. In fact, 24 real numbers per lattice site are needed for each quark field, while 18 real numbers per lattice link are needed for the gauge field. In a typical QCD simulation on the lattice, the computer generates these sets of numbers, called field configurations, with a probability that is calculated based on the formula of QCD. This calculation is complicated and computationally intensive, but it is possible. From each field configuration one can then calculate a wealth of physically interesting quantities such as energy, mass, etc.. Average values of these quantities are calculated for the full set of field configurations generated by the computer. The method is very similar to what is used to simulate statistical mechanics systems. The equivalent of the Boltzman weight is present here as well, and it dictates the probability with which field configurations are generated. In particular, molecular dynamics techniques are employed to generate new field configurations from previous ones.

Although it is possible to calculate many physical quantities using numerical simulations, there is a class of them that still remains beyond reach. For example, equilibration processes, or finite density physics, involve a severe "sign" problem (it is actually a complex phase problem) that prohibits use of these techniques. Research efforts have been active for many years now to address these issues. Thankfully, a large class of problems does not suffer from these difficulties. Nuclear physics calculations, calculations of the critical phenomena of the QCD thermal transition, calculations that relate to the physics of the current theory of elementary particle physics (the standard model), as well as of theories beyond the standard model are currently being simulated on the largest supercomputers. But this is not without effort, for lattice QCD requires enormous computational resources.

3. **BlueGene/L and LQCD**

In this section we discuss LQCD on massively-parallel supercomputers and in particular, LQCD on BlueGene/L.

It may seem strange that a physical theory in the frontier of science would have anything at all to do with a machine designed by engineers with strict timetables and guidelines in the frontier of technology. Theoretical physics carries a tradition of "pure thinking" and of analytical calculation where computers are barely needed. Conversely, the computing industry is defined by a most rapid development schedule of ever faster machines that must follow Moore's law, and where there is not time for theories especially of some distant and unrelated subject.

But it is not strange. As described in the previous section, the strong force regime of QCD would be inaccessible to theoretical calculations if it were not for the largest supercomputers available. In fact, if lattice theoretical physicists had all their wishes come true it would make today's supercomputers look desperately slow. The thirst for computing speed is almost unquenchable. That this is so and also that it is possible for lattice QCD to naturally absorb these vast amounts of computing is very interesting. As we will present in the following section the weak scaling of QCD on BG/L - a massively-parallel supercomputer - is perfect. And the need for ever finer lattices that occupy ever larger volumes indicates that very large lattices are of interest. This implies that QCD can use almost any large size parallel supercomputer that any current and near feature technologies can build. Petaflop-scale machines are anxiously awaited. This brings up the thought that perhaps there is a much deeper reason for all this than simple complexity. We will come back to this at the end of the section with some thoughts.

From the supercomputing engineering perspective, QCD has proven to be of great value for many reasons. To understand this let us briefly describe the QCD code and, in particular, its implementation on the BG/L supercomputer.

It turns out that in most QCD implementations about 90% of the compute cycles are expended inside a small routine (about 1,000 lines of code) called the QCD kernel or D-slash. This kernel calculates the dynamics of the quarks and their interaction with the gluons. Obviously, optimizing D-slash very well is of great importance. The basic operation that involves D-slash is:

$$\Psi(x) = \sum_{y} \not{D}(x,y)\Psi(y) \qquad (1)$$

where 𝓮𝓻(x) is the quark field at the space-time coordinate x and $\not{D}(x,y)$ is the D-slash operator. This is a sparse matrix with indices x, y. Most elements are zero except when x and y are nearest neighbors. Because the D-slash operator is so sparse, it is not stored in memory and its action is calculated operationally. D-slash is given by the following equation:

$$\not{D}(x,y) = \frac{1}{2}\sum_{\mu=1}^{4}[\\ U_\mu(x)(1+\gamma_\mu)\delta(x+\mu,y) + \qquad (2) \\ U^+_\mu(x-\mu)(1-\gamma_\mu)\delta(x-\mu,y)]$$

In the above equation, the sum over ③ is a sum over the four space time directions. The gluon field residing on a link that originates at location x and is along the ③ direction is a 3x3 complex matrix (18 real numbers) represented by U$_{③}$(x). The gluon field caries an internal index, called color charge, that can take 3 values. The ♑$_{③}$ matrices are 4x4 complex matrices that act on another internal index, called spin, carried by the quark field. The function ♋(a,b) is one if a=b and zero otherwise. These functions implement the nearest-neighbor feature of the operator. It should be noted that the terms $\frac{1}{2}(1\pm\gamma_\mu)$ are projection operators and reduce the 24 component quark field (also refered to as full spinor below) into four 12-component intermediate fields (also referred to as half spinors below). One standard way to efficiently implement equation (1) so that it allows for possible overlap of computations and communications is as follows:

1) Using the four projection operators (1+♑$_{③}$)/2 (③=1,2,3,4)
spin project 𝓮𝓻 into four temporary half spinors ✐$^{f}_{③}$ for all local lattice sites, and store them to memory.
2) Begin sending/receiving each ✐$^{f}_{③}$ that is on the surface of the local lattice to/from the nearest neighbor nodes along the negative direction ③. Each half spinor consists of 12 numbers. In double precision this is 96 bytes that need to be communicated for each site on the negative direction surfaces.
3) Using each projection operator (1-♑$_{③}$)/2 (③=1,2,3,4) spin project 𝓮𝓻 and multiply the result with U$^{+}_{③}$ in order to form four half spinors ✐$^{b}_{③}$ for all local lattice sites, and store them to memory.
4) Begin sending/receiving each ✐$^{b}_{③}$ that is on the surface of the local lattice to/from the nearest neighbor nodes along the positive direction ③. Each half spinor consists of 12 numbers. In double precision this is 96 bytes that need to be communicated for each site on the positive direction surfaces.
5) Wait for the ✐$^{f}_{③}$ communication to complete. Typically this involves polling a network register.
6) Now that all needed half spinors ✐$^{f}_{③}$ are on the node, multiply each of them with U$_{③}$ and convert them to full spinors. Add all four full spinors for each site and store the resulting full spinor to memory.
7) Wait for the ✐$^{b}_{③}$ communication to complete. Typically this involves polling a network register.
8) Now that all ✐$^{b}_{③}$ are on the node, convert each of them into a full spinor and for each site add them together. For each site add the result to the full spinor of step (6) after loading it from memory. This produces the resulting full spinor for each site.

Notice that in the above steps, the ✐ fields are not sequential in memory. The U fields are sequential for the first set of four terms and for the second set of four terms but not between the two sets. Also, the loop over lattice sites is over a four-dimensional lattice. As a result memory accesses from the linear memory are typically sequential only in the internal indices, as indicated above, which are therefore very short. Memory accesses per site consist of 24 numbers for the full spinors, 12 numbers for the half-spinors and 4x18=72 numbers for the U field in all four links originating from the same site. Furthermore, the communications involve very short size messages. The half spinors that are communicated reside on the surfaces of the four-dimensional lattice and typically can not be grouped together into a large message. As a result, each half spinor is communicated individually. These are short messages of only 96 bytes each. The communications and memory accesses cannot be rearranged because they are in-between computations. The computations themselves are short, involving only a few operations. For example, the multiplication of the gluon matrix U with a half spinor involves 72 multiply-add operations which execute in just

36 cycles in a double floating point unit. Therefore, the above code involves very "bursty" memory accesses, communications and calculations and, as a result, is very sensitive to memory, communication network and floating point unit latencies. Surprisingly, the above code indicates a wealth of architectural requirements in order to achieve maximum performance.

Since QCD is defined in a nearest-neighbor lattice of space-time points, it is naturally mapped on a lattice of compute nodes connected with nearest-neighbor physical links. However there are some implementations of QCD that require local but more distant than nearest-neighbor communications. This implies that a strict nearest-neighbor network would be limiting and leads to the requirement for a more general network.

The above code allows for almost maximal overlap between computations, communications and memory accesses. Given the sensitivity to latencies, a machine that could overlap all three of these activities would offer a substantial performance advantage. This indicates that the following hardware features are desirable for QCD:
a) Load/store accesses in "parallel" with computations and communications.
b) Sophisticated memory pre-fetching that allows block-strided accesses.
c) Communications that can overlap with computations and memory accesses. This implies a DMA driven network.

Finally, it should be mentioned that equation (1) is the inner-most part of a Conjugate Gradient (CG) inverter. This requires two global sum reductions per iteration. As a result, fast global sum reduction capability is important, suggesting that a good part of it should be hardware supported.

Although BG/L is a general purpose supercomputer, not designed for optimal QCD performance, many of the above features are present in its hardware. Here is a short description of the BG/L hardware. The reader is referred to [11] for a full description.

The BG/L supercomputer is a massively-parallel machine comprised of compute nodes that are inter-connected via nearest-neighbor links arranged in a three-dimensional torus topology. Each node is an IBM ASIC containing two IBM PowerPC 440 CPU cores. Each core has a custom double multiply-add unit capable of performing up to four floating-point operations per cycle. Therefore each node can execute up to eight floating-point operations per cycle. Each core has a 32 KByte L1 data cache memory, but the two L1 memories are not coherent. Each is fed by a small, multi-stream, sequential prefetcher (L2) which, in turn accesses a shared, on-chip, 4 MByte L3 cache memory. The L3 accesses external DRAM via an on-chip DDR2 controller. The ASIC contains a sophisticated, packet-based virtual cut-through router. This allows any node to send packets to any other node without intermediate CPU intervention. Packets that arrive at a node are kept if they are destined for that node or are routed to the appropriate output links in order to reach their final destinations in an optimal way. The network router is accessed from either CPU core by writing/reading packets into hardware addresses that correspond to SRAM-based FIFOs inside the router. A second, independent collective network is also on the ASIC and provides fast reduction operations such as global sums. Two such ASICs (nodes) are assembled on a small circuit board that also contains the external DRAM (typically 1 GByte for both nodes). More functionality is present in the ASIC, but it does not directly relate to the purposes of this paper.

The 440 core has a separate load/store pipeline and can have up to three outstanding load instructions. This allows memory access and computations to overlap. However, the torus communication network does not allow for overlap of computations and communications because the CPU has to prepare the hardware packets and copy them between memory and the torus FIFOs. Because of this, the earlier code description should be modified by consolidating step (2) with step (4) and step (7) with step (5).

Given the above, it is clear that equation (1) must be carefully coded in a way that is "molded" to the BG/L hardware in order to achieve high sustained performance. This is particularly hard since the sensitivity to latencies is amplified by the large computing capability of the hardware (8 Flops per CPU cycle). In order to be able to take full advantage of the hardware, we wrote our code in inline assembly. The main features of our code are:
i) All floating-point operations use the double multiply-add instructions by pairing all additions with multiplications in sets of two wherever possible. The complex numbers used by QCD make this natural.
ii) All computations are arranged to avoid pipeline conflicts. These have to do with register access rules.
iii) The storage order of the quark and gluon fields is chosen to maximize the size of sequential accesses.
iv) Load and store operations are arranged to take advantage of the cache hierarchy and the three outstanding load instructions capability of the 440 CPU.
v) Since load and store operations can proceed in parallel with floating-point computations, we overlapped memory accesses with computations wherever possible to reduce memory access latencies.
vi) Since each CG iteration requires two global sums over the entire machine, we used fast reduction over the global collective network.
vii) BG/L does not have a network DMA engine and, as mentioned earlier, the CPUs are responsible for loading/unloading data from the network, reading and storing them to memory, as well as preparing and attaching the hardware packet headers. Since the transfers that need to complete between calculations are very short, we are very careful not to introduce any unnecessary latencies. To this end, we developed a very fast communications layer directly on the torus hardware. This layer takes advantage of the nearest-neighbor nature of the communication and dispenses with control related communications. In addition, because the communication pattern is ``persistent'', the packet headers are only calculated once at the beginning of the program. Furthermore, all communications involve direct transfers from/to memory without intermediate copying. Also, although QCD requires a four-dimensional lattice, while BG/L has a three-dimensional lattice of nodes, there is a natural way to map QCD onto BG/L. The two CPU cores in each node can serve as a "fourth" dimension. The system software has a virtual node mode of operation where each core is assigned its own memory footprint, And half the torus FIFOs can be assigned to each core. In this sense each core is a virtual node.

Communication between cores is possible via a commonly mapped area of memory. We carefully overlap the necessary memory copy time with the time it takes for the network packets to be fully received.

As was mentioned earlier, D-slash is responsible for 90% of the consumed cycles. The remaining 10% are spent by the bulk of the QCD code. This code is tens of thousands of lines long and is written in a high-level language. It encodes both the physics of QCD as well as ingenious algorithms. These codes are written by groups of theoretical physicists and have been developed over many years. It is interesting that the full QCD code stack involves two extremes: a short kernel written in assembly together with a large amount of code written in a high-level language. In our work, we programmed the D-slash kernel but used the C++ code base of the CPS physics system that originated at Columbia University (Columbia Physics System) [12].

From the above discussion it should be clear that although BG/L is a general purpose supercomputer, it is also a natural match for QCD. But now one can also see that QCD can offer more to computer engineering than just architectural guidelines. In fact, it can and has been used during all the steps involved in the development of a supercomputer.

If one wishes to design hardware that will perform well for QCD, the design will have to be simple and modular in order to be able to serve all the concurrent and competing demands. In particular, tradeoff decisions that affect latency can be based on the QCD very low latency performance requirements. For example, this may affect the depths of various pipelines.

During software design, application related libraries as well as communications layers are developed. Again, QCD performance requirements indicate that a very simple, low-latency communications layer molded directly on the hardware is important. Such a layer would be useful for any QCD-type application where the communication pattern is fixed and small amounts of data (KByte size) are communicated at one time. This is in contrast to the general purpose "heavier" type of communication layer, such as MPI. Furthermore, given the importance of low-latency memory access, specialized library functions can be developed for commonly used operations such as the ones found in QCD.

During verification, the QCD kernel D-slash can serve as a valuable tool. It can be used to expose "bugs" that may otherwise be unreachable. This is because QCD uses the hardware at high efficiencies as well as at high overlap. For example, the floating point unit can be operating at full performance while the network is transferring data at high bandwidth and the memory hierarchy is moving data at high performance. This type of situation applies pressure on the hardware from competing as well as concurrent demands. Furthermore, it is the full kernel of a real application and it is therefore of direct importance. Applications tend to be one (but certainly not the only one) of the best verification tools. There are examples where bugs "escaped" full verification suites only to appear during execution of some application. The problem is that applications tend to be large and therefore not suitable for hardware simulators. This is not the case for the small-sized QCD kernel which can execute in only a few thousand cycles.

During full system validation QCD can serve as a unique tool for fault isolation. One can program all nodes to perform identical operations on identical data sets (for example, by setting the random number generator seed to be the same on all nodes). This is possible because the communications are nearest-neighbor, their pattern is fixed for all nodes, and the application is strictly SIMD. All nodes will send and receive the same data from their neighbors. At certain intervals one can check that all nodes have the same value for some intermediate number (for example the on-node energy of the gluon field). If a node differs, then the fault is isolated in the neighborhood of that node and corresponding links.

Finally, and very importantly, the QCD kernel can serve as a powerful performance evaluation tool. This can be done as a paper study even before the computer development begins. Because the QCD demands are well defined by equation (1) these studies can be reliable. Equally important, the performance of D-slash can be measured at every stage of the development, from verification to a fully built system performance evaluation.

These considerations are not just theoretical ponderings. Many of them have been part of the development of several supercomputers, including IBM's line of BlueGene systems.

In closing this section, let us comment on the cryptic remark we made at the beginning of the section, namely that the need of lattice theoretical physics for massively-parallel computing speed and the ability to use it may not be accidental. One clue is that the four-dimensional local interaction properties of QCD and other similar theories allows for a natural map onto the grid of a massively-parallel supercomputer. To be sure, the size of the BG/L machine is on the order of 10 meters while the size of a nuclear particle is about 10^{-15} meters. That is 16 orders of magnitude difference. But this gap could, in principle, close if Moore's law were to hold down to such small distances. The point is that in principle there is no obstacle to bringing the computation close to the size of what is being simulated, especially when considering quantum computation. This is certainly a theoretical curiosity. However, some physicists have arrived at similar conclusions from quite a different direction. It seems as if nature can, after all, be described as a massively-parallel computer computing all the phenomena we observe. The idea of a cosmic computer has been the stuff of science fiction, but it may turn out to be a most convenient description of nature that can unlock our understanding of some yet unsolved mysteries. That massively-parallel supercomputers could have something to do with such ideas is an added bonus.

4. Performance

In this section we present the performance results of our code on BG/L.

The strong scaling properties of our kernel were measured early on [13]. Our method is not the usual one since we simply kept the number of nodes fixed (to two nodes, four-cores) while we decreased the local problem size. This is akin to strong scaling which keeps the global size fixed while increasing the number of nodes and, thereby, decreasing the local problem size. The results are given in Table 1.

%of peak	**2^4**	**$4x2^3$**	**4^4**	**$8x\ 4^3$**	**8^2x4^2**	**$16\ x\ 4^3$**
D-slash no comms	31.5	28.2	25.9	27.1	27.1	27.8
D-slash	12.6	15.4	15.6	19.5	19.7	20.3
CG Inverter	13.1	15.3	15.4	18.7	18.8	19.0

Table 1: Sustained performance for various local lattice sizes, akin to strong scaling.

As can be seen, the smallest local lattice (2^4) without communications achieves 31.5% of peak. This high performance is largely due to the fact that the data mostly fits into the L1 cache resulting in fast memory accesses. However, such a small local lattice has a large surface to volume ratio and therefore, a large number of communications per volume are necessary. Because communications cannot be overlapped with computations on BG/L, the communication cost is additive and the performance drops down dramatically to 12.6% when communications are included. For the larger $16x4^3$ local lattice, the performance without communications is less (27.8%), but the surface to volume ratio is smaller, so the cost of adding communications is less severe, dropping performance to 20.3%. This interplay between memory access and communications is interesting in itself.

Nevertheless, QCD is typically used as a weak scaling application. The nearest-neighbor nature of the communications as well as the existence of a fast global sum collective network in BG/L give perfect speedup as the number of compute cores is increased. We were able to increase the number of cores all the way up to the maximum number present in the fastest supercomputer (as of the date of this writing), the BG/L 64-rack system at LLNL. This result [4] is the culmination of our efforts, as well as of the considerations described in this paper. It appears here, for the first time in print, in Figure 1, which shows a maximum of 70.5 TFlop/s sustained on 131,072 CPUs. The local lattice size is 4x4x4x16, resulting in a.maximum global size of 128x128x256x32 since the grid of compute nodes of the full machine is 32x32x64x2. The sustained percent of peak in this figure is 19.3% for the D-slash kernel and 18.7% for the full Conjugate Gradient inverter, which includes the global sum reductions.

5. Conclusions

In this article we gave a general description of the physics of QCD and discussed how massively-parallel supercomputers are a natural match for this application. QCD and supercomputing has a long history. The reader may be interested to know that one of the most popular theoretical physicists and a Nobel laureate, Richard Feynman, was involved in the development of the Connection Machine 2. In fact, he actually coded QCD for that machine [14].

Furthermore, we discussed how QCD can help in the development of massively-parallel supercomputers from architecture to final system performance evaluation. Indeed, this has been a component of several supercomputer development efforts, including the BlueGene series of machines.

Finally, we presented the culmination of our efforts in Figure 1, showing a perfect speedup of QCD up to 131,072 CPU cores and 70.5 sustained TFlop/s. This result was obtained with the fastest supercomputer as of the time of this writing, the 64 rack BG/L system at LLNL.

We hope to have communicated the close ties of QCD with supercomputing since these ties can serve both fields well in the very interesting and challenging immediate future, where new technologies make it possible for dizzying computing speeds, and new physics experiments generate new mysteries for lattice QCD to solve.

Acknowledgements

We would like to thank Dr. George Chiu of IBM Research for his help and support. We would also like to thank IBM Research and the IBM BlueGene/L team for their support. We are grateful to the IBM Watson BG/L supercomputing center and to Lawrence Livermore National Laboratory for allowing us access to these most precious resources. We would like to thank the QCDOC collaboration for providing us with the CPS software. Ron Soltz acknowledges the Department of Energy for supporting his research.

References

1. F. Wilczek, “What QCD Tells Us About Nature -- and Why We Should Listen” In *Nucear Physics A 663 3-20 (2000), proceedings of PANIC 99 conference, http://xxx.lanl.gov/ps/hep-ph/9907340*
2. K.G. Wilson, “Confinement of Quarks”, In *Physical Review D 10, 2445 (1974).*
3. K.G. Wilson, “Quarks And Strings On A Lattice”, In *New Phenomena in Subnuclear Physics, ed. A. Zichichi (Plenum Press, New York), Part A, 69, (1975).*
4. P. Vranas, G. Bhanot, M. Blumrich, D. Chen, A. Gara, M. Giampapa, P. Heidelberger, V. Salapura, J.C. Sexton, and R. Soltz, “2006 Gordon Bell Special Achievement Award”, In *Proceedings of Supercomputing 2006, Tampa Florida.*
5. P. Vranas, G. Bhanot, M. Blumrich, D. Chen, A. Gara, P. Heidelberger, V. Salapura, J.C. Sexton, “The BlueGene/L Supercomputer and Quantum Chromo Dynamics”, In *Proceedings of Supercomputing 2006, Tampa, FL (2006), http://sc06.supercomputing.org/schedule/pdf/gb110.pdf*
6. Ta-Pei Cheng and Ling-Fong Li, “Gauge theory and elementary particle physics”*, Oxford University Press (1984).*

7. M.E. Peskin and D.V. Schroeder, "An introduction to Quantum Field Theory", *Perseus Books Publishing (1995).*
8. M. Creutz, "Quark, gluons and lattices", *Cambridge University Press (1983).*
9. I. Monvay and G. Munster, "Quantum Fields on a Lattice", *Cambridge University Press (1994).*
10. J. Kogut, "Milestones In Lattice Gauge Theory", *Kluwer Academic Pub. (2004).*
11. The BlueGene/L, In *IBM Journal of Research and Development, Vol 49, 2/3 (2005).*
12. CPS: *Columbia Physics System, http://www.epcc.ed.ac.uk/~ukqcd/cps*
13. G. Bhanot, D. Chen, A. Gara, J. Sexton, and P. Vranas, "QCD on the BlueGene/L Supercomputer", In *Nuclear Physics Proceedings Supplement 140, 823-825 (2005), http://xxx.lanl.gov/ps/hep-lat/0409042*
14. W. Daniel Hillis, "Richard Feynman and the connection machine", In *http://www.kurzweilai.net/* articles/*art0504.html?printable=1*

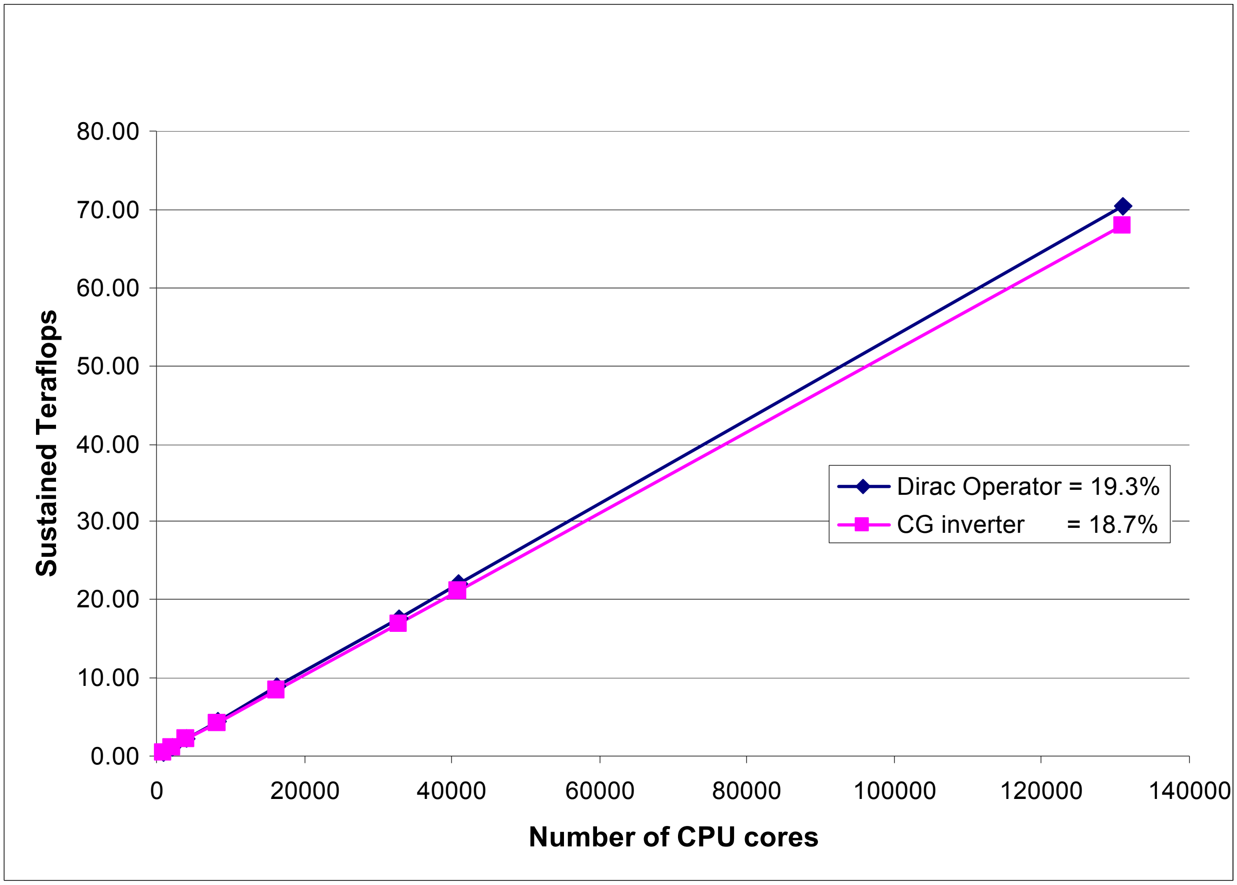

Figure 1: The QCD Dirac operator (D-slash) and Conjugate Gradient inverter speedup on the BG/L supercomputer as the number of CPU cores is increased up to the full machine size, 131,072. The highest sustained speed in this graph is 70.5 TFlop/s [4]. The total lattice has size 128x128x256x32, while the CPU cores form a grid with size 32x32x64x2. Therefore the local lattice on a CPU has size 4x4x4x16.