Overview of the QCDSP and QCDOC computers

P. A. Boyle
D. Chen
N. H. Christ
M. A. Clark
S. D. Cohen
C. Cristian
Z. Dong
A. Gara
B. Joó
C. Jung
C. Kim
L. A. Levkova
X. Liao
G. Liu
R. D. Mawhinney
S. Ohta
K. Petrov
T. Wettig
A. Yamaguchi

The QCDSP and QCDOC computers are two generations of multithousand-node multidimensional mesh-based computers designed to study quantum chromodynamics (QCD), the theory of the strong nuclear force. QCDSP (QCD on digital signal processors), a four-dimensional mesh machine, was completed in 1998; in that year, it won the Gordon Bell Prize in the price/performance category. Two large installations—of 8,192 and 12,288 nodes, with a combined peak speed of one teraflops—have been in operation since. QCD-on-a-chip (QCDOC) utilizes a six-dimensional mesh and compute nodes fabricated with IBM system-on-a-chip technology. It offers a tenfold improvement in price/performance. Currently, 100-node versions are operating, and there are plans to build three 12,288-node, 10-teraflops machines. In this paper, we describe the architecture of both the QCDSP and QCDOC machines, the operating systems employed, the user software environment, and the performance of our application—lattice QCD.

Introduction

In this paper we discuss two massively parallel computers that were designed and constructed for efficient, cost-effective calculations of physical systems subject to the strong nuclear force. While simulations of the strong nuclear force may seem a very esoteric goal, the techniques and underlying mathematics are common to a broad class of problems and are particularly common among exceedingly demanding problems in high-performance computing. Two numerical methods, whose use in our computations is detailed below, are the Metropolis algorithm for importance–sampling a large dimensional integral (more than approximately 10^7 dimensions for QCD) and Krylov space inversion of a large sparse matrix.

The strong nuclear force affects the protons and nucleons of atomic nuclei, the up and down quarks, which are constituents of protons and nucleons, and the four additional short-lived quarks (for a total of six quarks) that are known from experiment to exist. The theory of the strong nuclear force, known as quantum chromodynamics (QCD), is an elegant generalization of the theory of electromagnetism, and is accurately described by a simple Hamiltonian that can easily be written in closed form. Just as the photon mediates the electromagnetic interaction between electrically charged particles, QCD includes a similar mediating particle, called the gluon.

A fundamental difference between QCD and electromagnetism is that the gluon interacts with itself as well as with the quarks. (In electromagnetism, photons in a vacuum interact only with one another very weakly.) For low-energy systems, such as the proton itself, the quark–gluon interaction and the gluon self-interaction are very strong. This makes the equations of QCD highly nonlinear, and in this regime they are not amenable to analytic calculations. (At very high energies, analytic calculations can be performed reasonably accurately, and these are an important part of our belief that QCD is the correct theory of the strong interactions.)

Thus, to study QCD at lower energies—to understand the proton mass, decays of particles made of short-lived quarks, the effects of heating protons to high temperatures, and many other phenomena—we are led to numerical techniques that can deal with this nonlinearity. While QCD can be cast as an infinite number of coupled

0018-8646/05/$5.00 © 2005 IBM

nonlinear differential equations, it is more easily handled numerically by Monte Carlo integration. To start out, the integration is over all values of quark and gluon fields in space–time, with each configuration of quarks and gluons weighted by its classical action. This is the well-known Feynman path integral formulation of a quantum-mechanical system.

To evaluate the Feynman path integral, we discretize space and time with a four-dimensional (4D) Cartesian grid. This grid, or lattice, gives the study of QCD with this technique its name, *lattice QCD* (LQCD). Only a small number of input parameters are needed: the quark masses and the strength of the coupling constant. When the grid spacing is made sufficiently small, QCD simulations should yield precise answers for a wide variety of physical phenomena. Because of its completeness and simplicity, and the precision of its numerical formulation, QCD is an attractive target for large-scale simulation and has received much attention as a *grand challenge* problem in scientific computing.

The characteristics of the problem mean that the minimal design parameters for an ideal QCD machine can be somewhat restrictive compared with those of the mythical general-purpose parallel machine:

- The problem naturally has a Cartesian structure, allowing for a simple nearest-neighbor mesh network.
- The only common non-nearest-neighbor communication is global summation.
- Both communication and memory access patterns are deterministic and amenable to both software and hardware prefetching.

Exploiting these key simplifications to obtain an advantage in both price and performance has been the *raison d'etre* for a number of specialized machines built in the U.S., Italy, and Japan, which we briefly discuss in the next section.

Computer architectures for QCD

All known algorithms for performing the importance–sampling of the Feynman path integral for QCD rely on solving the linear differential equation for a quark propagating in a given gluon background field (the Dirac equation). Discretizing this Dirac equation on a space–time grid produces a matrix equation to solve in which the matrix is very sparse and generally connects only nearest-neighbor points on a 4D grid. A Krylov solver is generally used to solve this equation and, since this dominates the calculational effort, designs for QCD machines concentrate on this part of the problem.

The nearest-neighbor structure of the Dirac equation makes computers with a regular mesh network of processing nodes an obvious choice. The mesh is generally chosen to be periodic, with the last processor node in a given direction connected to the first in that direction. The linearity of the Dirac equation means that all processing nodes are faced with the same amount of computation, resulting in no need for adaptive mesh approaches or other load-balancing techniques—an obvious simplification for both the hardware and software. (QCD is a very nonlinear system. Once the Dirac equation is solved, its result feeds back into the system to produce a new Dirac equation to solve. This feedback gives the system its strong nonlinearity.) Since standard techniques, such as conjugate gradient, for solving linear equations require global operations, a mesh machine must have the capability to do fast global sums and broadcasts to all nodes.

Given a mesh network architecture, the ability to solve the Dirac equation efficiently leads to requirements on the capabilities for the processing nodes and the network bandwidth and latency. The bandwidth required between processor and memory is one floating-point number per three floating-point operations; relatively little memory per processor is required. For computers with about 10K processors, a few megabytes of fast local memory allows lattice QCD problems with a total size of $32^3 \times 64$ to be done using only local memory. A larger amount of slower memory (approximately 100 MB per processing node) provides ample space for storing intermediate results from calculations. Interprocessor communication requirements can be as high as one floating-point number transferred per ten local floating-point operations. Additionally, the Dirac equation solution requires the exchanges of many small messages, so low latency (currently submicrosecond) in the mesh network is vital. Since QCD calculations require few input parameters and generate modest output data, the input/output (I/O) requirements are modest.

Some of the earliest QCD machines were designed and built at Columbia University in the 1980s. The network on these early machines was a 2D periodic mesh with the memory on each node also mapped into the address space of the four neighboring nodes, a method that has also been employed in all of the generations of Italian array processor experiment (APE) machines [1]. Machines of hundreds of nodes were built, and their general statistics are given in **Table 1**. These machines achieved good performance through processing nodes made up of a microprocessor and an external floating-point unit (FPU), but this configuration was not well supported by software, relying instead on handwritten and optimized microcode. Similar machines were developed by groups in Japan, Fermilab Italy, and IBM Research. Reviews discussing these early machines can be consulted for further details [1–3].

Table 1 A list of the machines built for QCD by Columbia University (2D and 4D mesh machines) and the 6D mesh QCDOC machine built by researchers from Columbia University, IBM Research, the RBRC, and the UKQCD collaboration. The US LQCD QCDOC is funded by the U.S. Department of Energy for use by LQCD physicists in the U.S.

Network	*Name*	*Date*	*Processor/FPU* (precision)	*Nodes*	*Speed* (Gflops)	*Memory* (GB)
2D mesh	16-node	1985	286/TRW (22)	16	0.25	0.016
	64-node	1987	286/Weitek (32)	64	1.0	0.128
	256-node	1989	286/Weitek (64)	256	16.0	0.5
4D mesh	CU[1] QCDSP	1998	TI DSP (32)	8,192	400	16
	RBRC[2] QCDSP	1998	TI DSP (32)	12,288	600	24
6D mesh	RBRC QCDOC	2004	PPC440 (64)	12,288	10,000	1,570
	UKQCD[3] QCDOC	2004	PPC440 (64)	12,288	10,000	1,570
	US LGT[4] QCDOC	2005	PPC440 (64)	12,288	10,000	1,570

[1]Columbia University; [2]RIKEN–BNL Research Center; [3]A large group of QCD physicists from the United Kingdom; [4]U.S. lattice gauge theory.

As latencies become longer relative to cycle time, this remote memory-addressing technique either becomes inefficient or requires large off-node reference queues to hide the latency and avoid stalling the central processing unit (CPU). As will be seen, in the later QCDSP (QCD on digital signal processors) and QCDOC (QCD-on-a-chip) machines the semantic for accessing the data of a neighboring node has been decoupled from the CPU. Programmable direct memory access (DMA) engines and an asynchronous message-passing library are used to give a simple but efficient overlapping of communication and computation with complete hiding of the internode communication latency.

The remainder of this paper discusses the QCDSP and QCDOC computers. The APE project in Italy has continued to develop computers for LQCD [4], and commodity clusters are also being used for lattice QCD, as reviewed in [5].

The QCDSP computer

For the early QCD computers, a processing node occupied a 100-square-inch to 200-square-inch printed circuit board. With the increase in levels of integration that became available in the 1990s, processing nodes, including memory, of 10 square inches to 20 square inches became possible without relying on expensive packaging technologies. With chips of lower electrical power consumption available and longer mean times between failures, machines of thousands of nodes could be imagined. The QCDSP machines designed and constructed between 1993 and 1998 represent an important example of this style of computer. As shown in Table 1, QCDSP machines of 8,192 and 12,288 nodes were built. Further details about QCDSP can be found in [6, 7].

While the earliest 2D mesh machines provided important results for LQCD, much more computing power was needed. One factor determining the computing power needed is the total size of the lattice on which the problem is formulated. Early calculations were done on lattices with a total size of $16^3 \times 32$ points, but various approximations were employed or the masses of quarks were kept unrealistically (from a physical point of view) large. For smaller, physically realistic quark masses, the calculations could not be done. Thus, a difficult kind of scaling was required for progress in LQCD; much more processing power was needed on a problem with a fixed number of lattice points. By increasing the dimensionality of the processor mesh from 2 to 4, the number of processors could be dramatically increased without a concurrent increase in the linear size of the smallest lattice that could be tackled. Of course, each processor node in this 4D mesh also had increased computing power, following the general industry trends.

The basic processing element of QCDSP is a Texas Instruments TMS320C31 digital signal processor (DSP), a 32-bit processor running at 50 MHz, consuming approximately 1 W and having a peak speed of 50 megaflops. At the time it was selected, it was not the fastest processor, but it did seem ideally suited for QCD applications because of its low electrical power and relatively high speed. In addition, the software tools available for it (a C and C++ compiler, for example) made QCDSP easier to program than earlier QCD machines. Each processing node also contained 2 MB of 60-ns dynamic random access memory (DRAM).

The additional component of the QCDSP processing node is a custom application-specific integrated circuit (ASIC), called the *node gate array* (NGA), designed by the Columbia Lattice QCD Group. The NGA includes a

prefetching buffer to supply the DSP with single-cycle access to DRAM provided that the pattern of memory fetches is regular, which is the case for the Dirac equation in QCD. With this prefetching buffer, the DSP and 60-ns DRAM make a solid processing node, since the DSP is not memory-bandwidth-limited during the solution of the Dirac equation.

The other major part of the NGA is the serial communications unit (SCU) that drives the eight nearest-neighbor connections in a 4D mesh. The SCU contains DMA engines to access local memory for data that is being sent or received, allowing simultaneous communication with eight neighboring nodes as well as concurrent local floating-point calculations. The nearest-neighbor communications are bit–serial, since any wider bus between nodes was not possible with the commodity cable and connector densities then available. The latency for memory-to-memory transfers for neighboring nodes was around 1 μs. In addition, the SCU contains special hardware to perform global sums without involving the local processor, yielding a fast global sum.

The development of QCDSP and the 400-gigaflops, 8,192-processor machine at Columbia was supported by the U.S. Department of Energy (DOE), with the 600-gigaflops, 12,288-processor machine at the RIKEN–BNL Research Center (RBRC), funded by the RIKEN Laboratory in Japan. The machines achieve a maximum performance of 30% of peak speed for QCD and won the Gordon Bell prize at the SC98 High Performance Networking and Computing Conference. The machines do not have any fault-tolerant capabilities, so any hardware failures must be promptly repaired. Most of the time, partitions of 2,048 and 4,096 nodes have been used, although we have been running one 4,608-node partition for almost two years. These partitions are created by physically recabling the machine, a task that requires a few hours. The partition sizes have been determined to best suit the physics calculations we wish to perform.

The applications software for QCDSP is written primarily in C++, with optimized kernels in assembly for the solution of the Dirac equation. The compiler is a single-node compiler, with data transfer between nearest-neighbor nodes done via our own message-passing interface, which was designed to minimize software latency. Each node runs a real-time kernel we developed that provides user code with standard C and C++ runtime support, although there is support for only a single application process running. A host workstation runs manager software we call the *qdaemon*, which coordinates the thousands of nodes on QCDSP by handling booting, program loading, I/O, and hardware failure reporting.

A final important part of the QCDSP architecture is the compact size achieved by having low electrical power consumption. This improves the reliability, since the compact size places more connections inside printed circuit boards. The low power means that QCDSP costs around $50,000 a year to run, including air-conditioning costs.

QCDOC hardware

QCDSP represented a major step forward in QCD machines. It demonstrated that machines of many thousands of nodes could be operated reliably, that a mesh network of bit–serial connections could provide low latency and fast global sums, and that good performance for QCD per dollar spent could be achieved by balancing local floating-point speed, communications speed, and electrical power consumption. However, QCD is a sufficiently complex problem for it to be desirable to have many more orders of magnitude computing power than QCDSP provides. An obvious path for improvement was to enhance the basic QCDSP architecture with components reflecting much greater transistor density and clock speeds. IBM system-on-a-chip (SoC) technology appeared to be a good match for our design goals.

The success of QCDSP also made a larger group of researchers interested in using this computer architecture for QCD. Design work for the QCDOC computer began with support from the U.S. DOE, the RIKEN Laboratory in Japan, and a large group of QCD physicists from the United Kingdom, known as the UKQCD. This large group of collaborators provided more human and financial resources to the design process than had been available for QCDSP.

QCDOC processing node

In the fall of 1999, we began the design of the custom QCDOC ASIC using IBM SoC technology. **Figure 1** is a block-level schematic of the resulting ASIC. This ASIC contains an embedded IBM PowerPC* 440 (PPC440) processor, an attached 64-bit IEEE FPU, 4 MB of memory, communications controllers for 24 simultaneous bit–serial communications links, two Ethernet controllers, and a controller for external double-data-rate synchronous DRAM (DDR SDRAM). Thus, the QCDOC ASIC combines all of the features of the QCDSP node, in addition to many more, on a single chip, and it provides 20 times the performance at about twice the cost. We next describe some of the features of this QCDOC ASIC. Further details are available in [8–10].

Prefetching embedded DRAM controller

A critical custom element of the QCDOC ASIC is the high-speed interface to the embedded DRAM, designed by members of our collaboration at the IBM Thomas J. Watson Research Center and their colleagues. This interface provides three bidirectional ports to the embedded DRAM: one for the processor local bus (PLB),

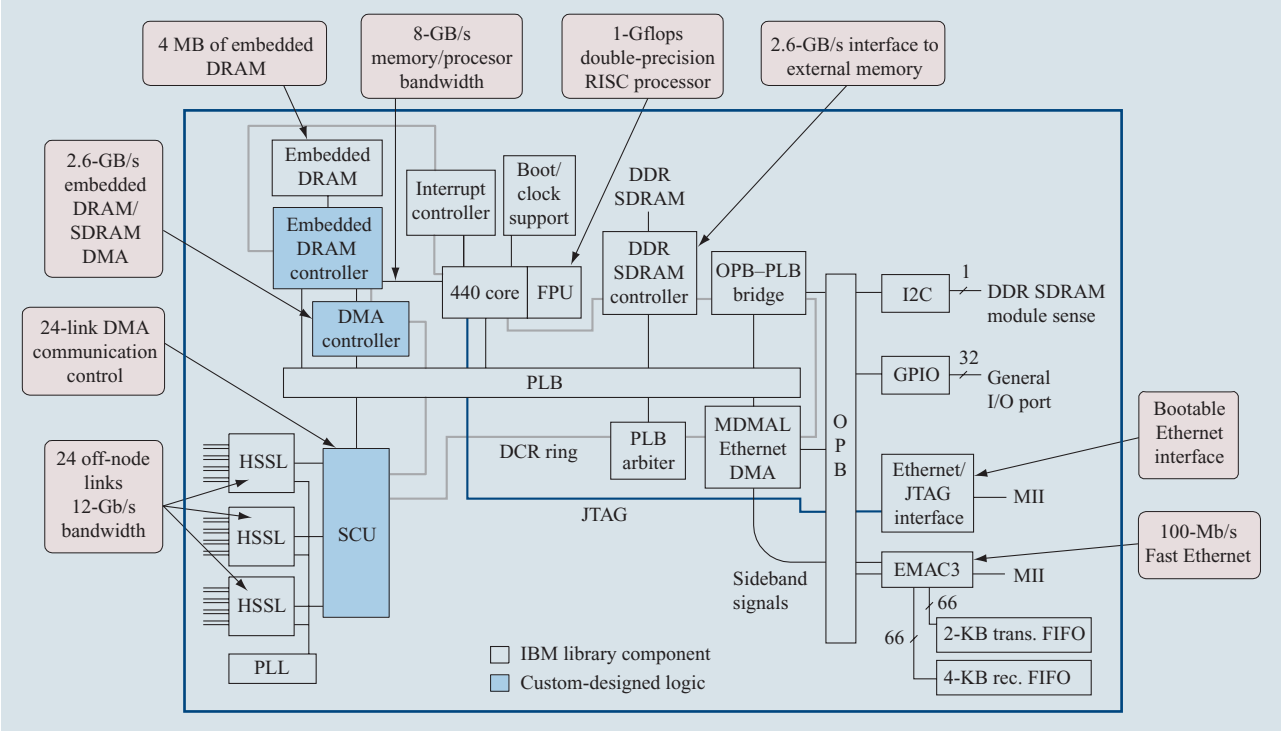

Figure 1

The QCDOC ASIC designed with IBM SoC technology is a complete processor node on a single QCDOC chip. The blue components in the figure were specially designed for QCDOC, while the remaining components are standard IBM macros. (GPIO = general-purpose input/output; DCR = device control register; HSSL = high-speed serial link; I2C = Inter-Integrated Circuit Serial Bus Interface; MII = Media Independent Interface; OPB = on-chip peripheral bus; PLL = phase-locked loop.)

one to a DMA unit used to move blocks of data between embedded DRAM and the external DDR SDRAM, and the third and most important, to the PPC440 core itself. Each port accesses memory through a 1,024-bit-wide path (actually 1,152, including the error correction control bits) and provides an efficient matching between this slower memory and the other, faster units on the chip. Each port has two pairs of 1,024-bit read buffers and two 1,024-bit write buffers. Each pair of read buffers arbitrates independently for embedded DRAM access and will prefetch up to 1,024 bits in advance of the actual read request. The presence of two such prefetching units permits two separate data streams to be read efficiently at the same time. (For QCD, this is often the gauge link matrices and the conjugate gradient solution vector.)

Write data is held in the two write buffers until a subsequent write or overlapping read request forces that write data to be flushed into embedded DRAM. Full coherency is provided between these buffers across all three ports. This architecture has proven to be very efficient, easily saturating the bandwidth of the three devices that it supports. This is most important for the PPC440 processor, where the embedded DRAM controller is connected directly to the PPC440 data bus and operates at the full processor speed, not the usual 3:1 ratio. For many of our calculations, the variables needed to solve the Dirac equation can be completely contained within this on-chip memory, so this high-bandwidth, low-latency connection to memory can accelerate the entire calculation.

Serial communications unit

The second major custom-designed part of the ASIC is the serial communications unit (SCU). This unit controls the sending and receiving of serial data on the 12 output and 12 input channels that connect a given node to its 12 nearest neighbors in the 6D mesh. To a large extent, the transfers in these 12 directions are independent and are controlled by 12 independent subunits. Serial packets exchanged between neighboring nodes are of three general types: 64-bit data packets, 64-bit supervisor packets, and 8-bit partition interrupt packets. Each of these packet types begins with an additional 8-bit control byte with error correction control (ECC) parity and control information. We describe the features of each of these three types of communication in turn.

Data packets

The receive unit has three 64-bit buffers that are initially empty, and it acknowledges each received data packet when that buffer has been unloaded by sending an 8-bit acknowledge (ack) packet back to the sender. Thus, the send unit can safely transmit three 64-bit packets before receiving a returned ack. This time exceeds the latency in this send-ack cycle and permits full bandwidth transmission to occur. Data to be transmitted in a particular direction is pulled from memory (either embedded DRAM or external DDR SDRAM) by a DMA control unit, dedicated to that direction, in a *block-strided* pattern. A block-strided pattern is determined by the start address, block length, number of blocks, and block separation. The instruction specifying this block-strided move is stored in a 128-bit word. Sixteen such DMA instruction words can be stored for each send and each receive direction, and their execution can be chained, permitting autonomous communication of rather complex data storage patterns. For each of these 12 send and 12 receive units, data to be transferred to or from memory is stored in eight 128-bit buffers, permitting the slowly transmitted serial data to be efficiently written to memory using burst PLB transfers.

Supervisor packets

These are two 64-bit words written by the processor directly into an SCU register and interleaved, with high priority, among outgoing data packets. When received, the corresponding PPC440 processor is interrupted and the receipt acknowledged when the 128-bit word has been read (implying that the next supervisor packet can be sent). This permits other software processes to communicate between nodes without interfering with an ongoing data transfer.

Partition interrupt packets

An important motivation for the 6D architecture of QCDOC is to provide convenient partitioning of the machine. For the earlier 4D QCDSP machine, a change in the size of the lattice on which the problem was formulated often required that the actual cables providing the communications had to be reconfigured. As discussed below, the extra two dimensions provided in the QCDOC 6D mesh permit a variety of 4D problems to be folded into those six dimensions simultaneously. However, to fully realize such a partitioning of the machine, we must provide interrupt functionality that is also partition-specific. This is most economically done using the 6D communications network, permitting the same software partitioning that routes data and supervisor packets within a given partition to similarly route interrupts.

The 8-bit interrupt packets provide this capability, supporting eight separate interrupts that can be received only by processors within the same partition as that of the node sending the interrupt. The interrupt packets are interpreted within the framework of a relatively slow (a few hundred kHz) on-node counter roughly synchronized among all the nodes in a given partition. The interrupts are then generated and interpreted in a pipelined fashion. During cycle n determined by this slow counter, each node assembles an 8-bit interrupt word I_n from a memory-mapped register. Set bits cause an interrupt, cleared bits do not. During this same nth cycle, each SCU transmits the OR of the interrupt word I_{n-1} and any incoming interrupt packets received during this cycle.

A new transmission is initiated only when a logically different interrupt packet is received. Such transmissions are made to all neighboring nodes within the partition. The final 8-bit interrupt word being transmitted during the nth cycle is presented to the interrupt controller of the PPC440 at the start of cycle $n + 1$. By making the slow clock period appropriately long and providing guard times near the end and beginning of the cycle, we can ensure that this system is functionally equivalent to the more standard hardwired interrupt systems often implemented in a parallel machine.

Error checking and recovery

Considerable error checking is provided within the SCU. The individual packets are transmitted according to a protocol that permits automatic recovery from any single-bit error. This requires that the 6-bit control code contain an ECC-style redundancy and that the remaining bits—data, supervisor, or interrupt—include parity. If a receiving unit detects a bad data or supervisor packet, an error acknowledgment is sent, and the sender then retransmits the corresponding packet. If an erroneous interrupt packet is received, it is ignored. The interrupt packets are not acknowledged. In addition to this packet-level error correction, a 64-bit checksum is kept for all sent and received data packets. These checksums are read, compared, and cleared infrequently, typically at the start and end of a job, and provide a guarantee that no incorrect internode data has been used.

Global sums and broadcasts

The final SCU design topic is the hardware support provided for global sums and broadcasts. Such global operations can be seriously inefficient on a mesh machine if too much time is required for each node to receive and then resend data that must ultimately be distributed to every node in the machine. To avoid the potentially large overhead associated with such receive and resend steps, the SCU contains limited logic that links the otherwise independent 12 send and 12 receive units. In "global" mode, incoming data from a particular direction is immediately sent out in as many as eleven other directions

as well as stored in memory according to the correctly active DMA instruction for the receiving direction. The logic supporting this one-to-many transmission is doubled so that global sum data can be passed in both directions of a specific machine dimension simultaneously. This effectively reduces the linear size of the machine in each direction by a factor of 2.

This global logic permits a broadcast to be performed over the entire machine as a single operation, with the one-to-many transmissions configured to pipe the data from the broadcasting node to all of the nodes of the machine. A global sum is performed as a series of partial sums in increasingly many dimensions. For example, as a first step for a 4D partition, local data would be transmitted in the x-direction among each of the N_x processors sharing common y, z, and t coordinates. This can be done in $N_x/2$ hops. Next, each node would add the available N_x numbers and start the same process in the y-direction. After four such processes composed respectively of $N_x/2$, $N_y/2$, $N_z/2$, and $N_t/2$ steps, each processor has the desired global sum. Of course, the mapping between the four dimensions in this partition and the actual 4D subvolume of the 6D machine may be quite contorted, with no effect on the programming or performance of these two types of global operation.

Summary

The SCUs provide both nearest-neighbor and global operations for QCDOC. The total nearest-neighbor bandwidth, utilizing simultaneous sends and receives, is 1.3 GB/s at 500 MHz. The memory-to-memory latency for a single 64-bit word transfer between nearest neighbors is approximately 600 ns. Since the 24 nearest-neighbor links can run concurrently, the effective latency can be much smaller.

Ethernet and JTAG interface

The last custom component of the QCDOC ASIC design that we highlight here is the special Ethernet controller that receives and sends User Datagram Protocol (UDP) packets containing JTAG (IEEE Standard 1149.1 developed by the Joint Task Action Group) code [11]. Incoming packets are automatically transcribed as JTAG sequences that are fed into the PPC440, while the resulting JTAG bitstream produced coming from the PPC440 is re-encoded in UDP packets and returned to the sender. This device is designed to work from power-on and is used to boot each node, removing the need for a boot programmable read-only memory (PROM). This unit was originally designed by Peter Hochschild and Richard Swetz at the IBM Thomas J. Watson Research Center and was available as Verilog** code that had been used to create a successful field-programmable gate array (FPGA) for another IBM project. A design team from IBM Rochester translated this Verilog design into the Very high-speed integrated circuit Hardware Description Language (VHDL) form that we integrated into the QCDOC ASIC.

QCDOC ASIC design methodology

The QCDOC ASIC was manufactured for Columbia University by the IBM Microelectronics Division under a standard commercial contract. As described above, much of the logic design and initial timing for some of the components of the QCDOC ASIC were performed by other groups within IBM. However, our collaboration, centered at Columbia, took final responsibility for the design. We worked with an IBM customer support engineering team in Raleigh, including Beth Danford, Harry Linzer, and Doan Trinh Nguyen. They provided the initial design setup, advised us on the style of logic design appropriate for the IBM ASIC design process, and performed the final physical design. We used Synopsys design tools for almost all of our work, including final gate-level simulations with physical timing parameters. The static timing specification was done using EinsTimer*, with considerable help from the IBM Raleigh team. For most of the IBM components, the entire timing specification was carried out at Raleigh. In fact, for a complex component such as the DDR SDRAM controller, since an intimate knowledge of the component is needed, it would have been very difficult to have had this done at Columbia. We believe that this process worked very well, with the IBM team providing excellent and timely support.

QCDOC networks

Three networks make up the QCDOC architecture. The nearest-neighbor mesh network, described above, is responsible for the high performance for the QCD application. We adopted a 6D mesh network, since this allows us to partition the machine, in software, into many smaller machines of lower dimensionality. (In fact, there has even been a recent theoretical advance in discretizing the Dirac operator, which naturally leads to a 5D operator, so we may use a 5D submachine for QCD as well as 4D ones.) The dimensionality was limited to 6, since higher dimensionalities would increase the number of onboard wires and inter-crate cables beyond practical limits. The characteristics of this 6D network are described in the discussion of the serial communications unit above.

The second QCDOC network is a standard 100-Mb Ethernet. There are two Ethernet connections to each ASIC. One connection is to the custom ASIC element, which allows JTAG control of the processor via the Ethernet, described above. The other is to a standard 100-Mb Ethernet controller, an available IBM macro.

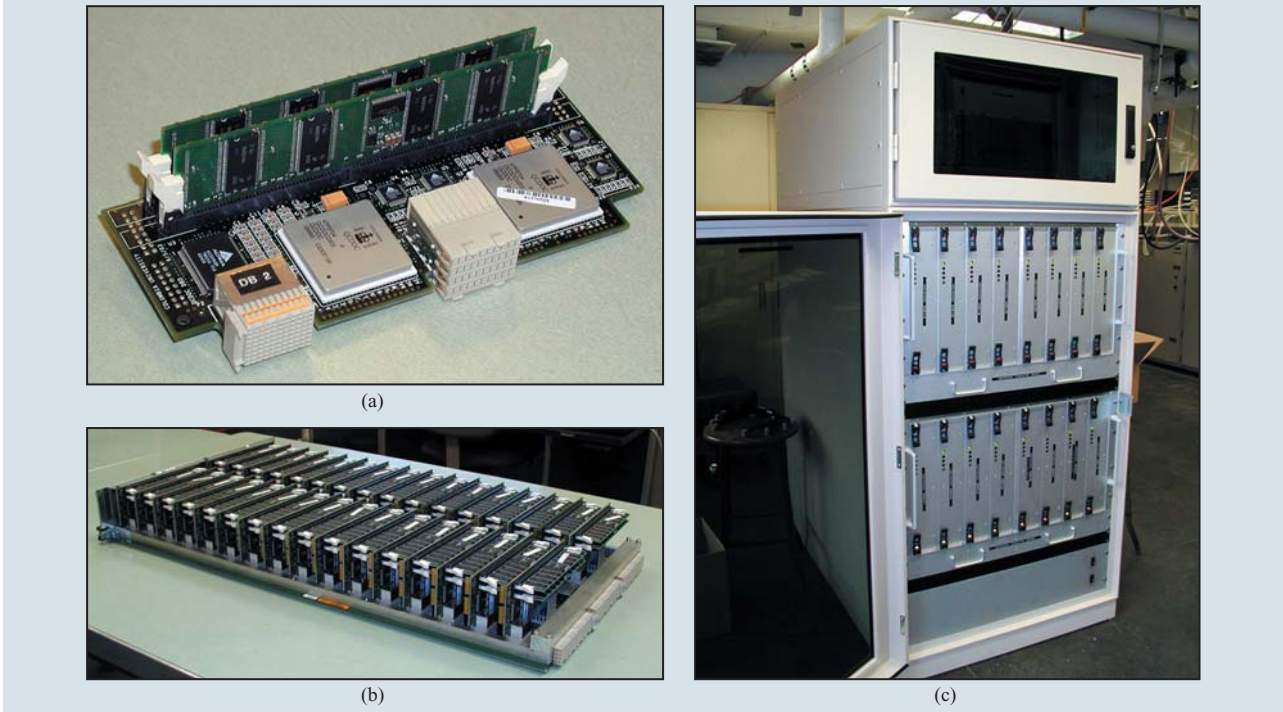

Figure 2

(a) A QCDOC daughterboard holds two nodes and their associated dual inline memory modules, or DIMMs (512-MB DIMMs shown). (b) A QCDOC motherboard holds 64 nodes as a 2^6 hypercube. (c) A QCDOC water-cooled rack fully populated with 16 motherboards mounted in two identical eight-motherboard crates.

The Ethernet/JTAG connection is used to boot each node and to run diagnostics, and for access via the IBM RISCWatch debugger. After booting, the standard 100-Mb Ethernet is used for general I/O from each node. Currently, nodes can mount external disks via standard network file system (NFS) protocols, and a large number of external disks are planned for the machine. These will be used primarily for temporary storage of intermediate results.

The third network is a simple three-wire global interrupt system that permits any processor to interrupt all others. This is used for initial synchronization at power-on, but is supplanted by the partition interrupts when the machine is operating as a collection of independent partitions.

QCDOC integration

As a university-based project, we must approach the electrical and mechanical design in a very careful fashion to minimize the number of personal computer (PC) boards that must be designed and maintained, and to provide a high degree of modularity and simplicity. In these design aspects, we closely followed the approach taken in the construction of the QCDSP machines.

At the lowest level of the fabrication hierarchy is a daughterboard: a 3-in. × 6.5-in., 18-layer PC board that holds two nodes, an associated dual inline memory module (DIMM) socket for each node, four Ethernet PHY chips, and one five-port repeater chip to provide Ethernet connectivity. **Figure 2(a)** shows such a daughterboard. The use of standard DIMM packaging for the memory permits a last-minute choice of memory size, which is, given the volatility of the memory market, a great convenience.

At the next level, we mount 32 of these daughterboards in a motherboard, as shown in **Figure 2(b)**. This is a 14.5-in. × 27-in., 18-layer board that joins the 64 nodes in the 32 daughterboards into a 2^6 hypercube. Six of the 12 faces of this hypercube are wired to the corresponding face of the same orientation. The other six faces are carried to the edge of the motherboard with high-speed traces. Thus, by connecting $N_0 \times N_1 \times N_2$ motherboards, these six faces can be joined to those on neighboring boards, creating a $2N_0 \times 2N_1 \times 2N_2 \times 2 \times 2 \times 2$ hypertorus. A large fraction of the motherboard wiring simply provides the matched 100-Ω traces needed for these network connections. In addition, support chips for the 100-Mb Ethernet as well as Ethernet and processor

clock distribution are present on the motherboard. The motherboard is supplied with 48 V power, which is converted to the required 1.8 V, 2.5 V, and 3.3 V by a row of ten quarter-brick power converters that runs down the middle of the two rows comprising 16 daughterboards, which are visible in Figure 2(b).

Four motherboards are inserted in a backplane, and two of these backplanes, holding eight motherboards, are mounted in a crate. Each four-motherboard backplane contains clock generation and fan-out circuitry that can be interconnected with cables to realize the specific clock tree needed for a given machine. The 0.210-in. thickness of the backplane is sufficient to permit the connectors into which the motherboard is inserted on one side of the board and those to which the cables attach on the other side of the board to press-fit into the same holes. Thus, there is no high-speed wiring on the backplane.

Two QCDOC crates are housed in a water-cooled rack, as shown in **Figure 2(c)**. These racks are the standard form-factor for production QCDOC machines and are manufactured for us by Elma USA (Fremont, CA). There are water-cooled radiators below each crate, and the cooling air circulates within the rack. The racks can be stacked two cabinets high, permitting a high-density, compact installation of 2,048 QCDOC computing nodes with a footprint of about 12 square feet. The input ac power draw is approximately 50 A at 208 V.

The needed high-speed serial connections between the motherboards are provided by 24 20-pair differential cables that attach through the backplane to each motherboard. Four of these cables join each pair of motherboards connected in a given dimension. Two of these cables carry the signals propagating in one dimension, and two carry the signals propagating in the opposite direction. Only 32 of the 40 signals are required to connect a 32-site face of the 2^6 hypercube of nodes on a motherboard. Six of the remaining 16 wires are used for the global interrupts, and the remaining ten are not used.

In summary, the entire QCDOC computer requires the design and fabrication of three distinct PC boards. Only three cable configurations are needed: 50-Ω clock distribution cables, 20-pair, 100-Ω differential cables, and standard RJ45-connected Ethernet cables. The quality of the cables, connectors, and board traces allows our low-voltage differential signaling (LVDS) 500-MHz serial communications to propagate essentially undistorted from the driver on the sending QCDOC ASIC through five PC boards and three pairs of connectors. Our electrical design was carefully modeled in advance by colleagues of our IBM collaborators, an exercise that has proven to be very accurate.

The first QCDOC ASICs were delivered in June of 2003 and have been extensively tested for many months. To date, no flaws have been found in the design, and the large machines will be built with this first design. As of October 2004, about 8,000 daughter cards have been produced, and 12,000 are expected to be finished by the end of fall. More than 200 motherboards have already been manufactured, along with 14 water-cooled racks. These components will make up two large machines, one to be installed in the RIKEN BNL Research Center at the Brookhaven National Laboratory, and the second at the University of Edinburgh. **Figure 3** shows a nearly completed 12,288-node QCDOC machine being assembled at Brookhaven National Laboratory.

Figure 3

A 12,288-node QCDOC machine under construction at Brookhaven National Laboratory in October 2004.

QCDOC software

The QCDOC software environment consists of three broad areas. First is the portion of the operating system software that runs on the host front end to boot, manage, and interface with the machine. Second is the kernel that runs on each node; it cooperates with the host software in the management of the machine and loading of user code, and also provides the underlying implementation of the user environment.

Finally, the user environment provides the programmer-visible portion of the software. In QCDOC, much effort has been taken to keep this user-visible layer as standard as possible without compromising performance. The user environment includes an open-source *libc* that is compliant with the Portable Operating System Interface Standard (POSIX). Two standard compile chains, *gcc* and *xlc*, are used. Only the messaging and memory model are nonstandard and reflect constraints of the lean hardware design.

Host software

The front end for QCDOC is a standard UNIX** symmetric multiple processor (SMP) server. The software

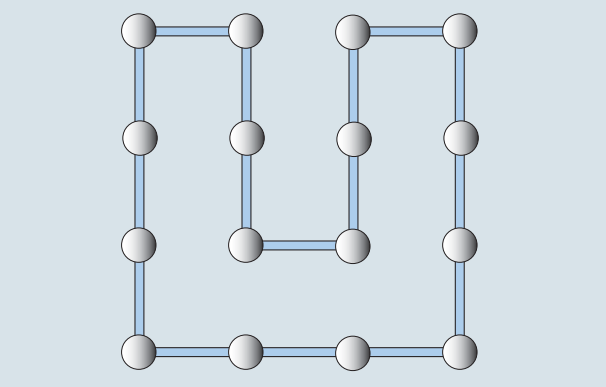

Figure 4

Example of partitioning a physical 4 × 4 machine into a single 1D partition of 16 nodes.

runs on AIX*, Solaris**, and Linux**, and for large QCDOC machines, IBM pSeries* servers are used. The host software consists of a multithreaded qdaemon. A modified user shell called the *qcsh* is used to pass QCDOC-specific commands to the qdaemon. The first job of the qdaemon is to boot a QCDOC back end in a timely manner. A boot kernel containing initialization, self test, and Ethernet software is downloaded into the instruction cache of the PPC440 processor on each node using the JTAG interface. The run kernel is then bootstrapped into working memory, after which the high-speed communications network is initialized and applications are run.

Both kernels require approximately 100 reliable UDP packets per node, or approximately one million packets for a large machine. With multithreaded software and multiple gigabit links on an SMP, the download time can be kept very modest on even a large machine without resorting to less strongly checked broadcast or more complex logarithmic boot schemes.

Communication between the qdaemon and the run kernels is carried out using the Remote Procedure Call Protocol (RPC) and greatly aids the maintainability and portability of our operating system. The use of RPC as a uniform transport mechanism is particularly attractive, since it is required by our NFS client to access both the front end and distributed disks. The conventional *standard out* and *standard error* from application programs are routed through the qdaemon to the user shell, and post-exit diagnostics of error counters and checksums on the machine are carried out after every job by the qdaemon.

The qdaemon manages multiple user connections simultaneously and supports dividing a large machine into multiple partitions, on each of which a single application may be run. The partitions are, in general, 6D subslices of the machine and may be remapped internally to present a 1D to 6D application-visible torus. The remapping is carried out by iteratively folding together two (not necessarily periodic) machine dimensions to produce a new single-application dimension that is periodic. When all six dimensions are remapped down to a 1-torus, the single-application dimension meanders around following a 6D space-filling curve. The case in which a 4×4, 2D machine is remapped into a 16-node, 1D partition is illustrated in **Figure 4**. In contrast to the Message Passing Interface (MPI) protocol, this remapping is done prior to running a job, thereby ensuring that a nearest neighbor in the Cartesian application torus is always physically a nearest neighbor on the real communications mesh.

Node kernels

We have chosen to write a very lean runtime executive, or run kernel, rather than use a standard UNIX-like operating system or even a commercial real-time operating system for QCDOC. This enables us to enhance the robustness of the machine and to eliminate a number of software-related overheads that occur on massively parallel processors (MPPs) based on UNIX-like nodes, such as scheduler impact and translation lookaside buffer (TLB) miss overhead, and to facilitate zero-copy direct memory access (DMA) on simple hardware.

Both QCDOC and QCDSP are much larger machines than typical MPPs. Reliability and uptime are more easily ensured when the boot process and device drivers are completely controlled. During boot, no component in the system is used without at least some form of modest test and corresponding reporting on error. This test set can be refined and augmented as new failure modes are discovered. Additional self testing and reporting are easily added to our homegrown run kernel.

QCD applications involve the need to invert sparse matrices exceedingly quickly. On QCDOC, the sparse matrix multiply can take as little as 20 μs, which means that the problem is very tightly coupled on the scale of a typical scheduling quantum, or indeed interrupt handling. When an application is running on tens of thousands of nodes, the impact of many different single nodes switching execution stream for a scheduling quantum can be very large.

The principal jobs of the run kernel are to support program loading, drive hardware devices, and service system calls and interrupts. Devices managed include the Ethernet subsystem via a custom implementation of the sockets UDP interface, the internode communication network via the SCU message-passing application programming interface that directly represents the rich features of the DMA engines, and the various ancillary on-chip components.

The kernel maintains a process table for two threads (kernel and application). However, it does not actively schedule, but only switches at job initiation and termination, ensuring that the compute nodes are dedicated solely to computation.

The memory management unit of the PPC440 is used to protect memory and enhance the robustness and user-friendliness of the machine, but not to translate. Since virtual and physical addresses are identical, the use of zero-copy DMA in the communications is trivial. Large pages enable the complete elimination of TLB miss overhead, since the entire memory can be mapped in the 64-entry TLB. As discussed below, memory management unit qualifiers are used to mark memory regions as transient, supporting the streaming nature of high-performance computing code.

Programming environment

Much effort has been made to keep the environment as standard as possible when this is consistent with our performance requirements. The standard C and C++ compilers (g++ and x1C) are supported as cross-compilers for the QCDOC nodes.

To provide a standard programming environment, we have reused the open-source, POSIX-compliant, C runtime library from Cygnus. This library requires a custom implementation of the *libgloss* (also known as GNU low-level operating system support). Libgloss is provided by our homegrown user space support, typically by a system call into the run kernel.

Internode communication is provided by messaging libraries extending the standard C support. At the lowest layer, SCU calls provide application access to DMA commands that send to the receive FIFOs (queues in which access becomes available according to the first-in first-out rule) on each of the 12 neighbors of a node and to DMA commands that receive from the receive FIFOs of a node. The calls are asynchronous, allowing communication and computation to be easily overlapped. Fast global reduction over arbitrary hyperplanes of the machine is provided.

On top of the SCU layer, there is a standard interface—the QCD message passing (QMP) interface—developed by the LQCD research community. This library provides fast nearest-neighbor messaging calls and more general communications patterns via software routing. MPI is not supported, since the additional generality is not needed for QCD, and leaner libraries can run with higher performance.

Memory model

The presence of level 1 (L1) instruction and data caches in the PPC440 is a marked improvement over QCDSP, where only an instruction cache was available. This allows QCDOC to achieve better performance for general programs that are not carefully written for the architecture. However, for the highest performance on QCDOC, the programmer must be aware of and use the fast on-chip memory judiciously.

The multiported directly addressable on-chip memory system on QCDOC somewhat complicates the programming model. Since the L1 cache of the PPC440 does maintain coherence, direct memory accesses by the SCU must be actively made coherent with the caches by software intervention. To avoid excessive overhead from cache flushes, a strategy has been developed to maintain coherency at low latency while retaining the spatial locality benefits of the cache.

A *qalloc* allocator has been implemented to manage multiple heaps, with a flags argument specifying the memory qualifiers. One flag to qalloc returns "communicable" memory that is marked transient in the PPC440 L1 cache. (Two ways of the PPC440 cache are set as transient; the remaining 62 ways are set as normal.) This communicable memory can be efficiently flushed by using as little as 32 cache-touch instructions. This flush is done by the communications calls.

The allocator always returns cache-aligned pointers, which means that well-formed messages cannot partially overlap the basic coherency unit in a dangerous way. The choice of two cache ways for the transient region well matches the two read prefetch streams supported by the prefetching embedded DRAM controller described earlier. The typical third write stream (e.g., AXPY, or $y \mathrel{+}= a \cdot x$ for vectors x and y and a scalar quantity a) performs well using the store-gathering feature of the PPC440 core.

Even for local floating-point operations, the use of transient memory prevents cache pollution and stack eviction. We also produce more deterministic cache-miss patterns, reducing the overhead from prefetching embedded DRAM controller misses and giving a significant speedup when the working set exceeds the PPC440 32-KB cache.

Application software

A broad set of QCD application code has been ported to QCDOC, facilitated by the standard runtime environment. This includes all of the major LQCD simulation code suites of the U.S. and U.K. research communities. Since virtually all QCD simulation codes have been written in a message-passing style for many years, a basic port to QCDOC is straightforward. In addition, the QMP library being developed by the U.S. lattice community is supported on QCDOC.

For maximum performance, libraries of optimized QCD kernels have been written in assembly language and are available to the research community. Since

Table 2 Performance on QCDOC of linear algebra operations for some common fermion types, shown as a percentage of peak speed. The Wilson and ASQTAD optimized results were run at 450 MHz (900 Mflops/node peak speed). All other results from 128-node machines were run at 420 MHz on newly built QCDOC nodes, which are not yet stable at 450 MHz. These benchmarks represent our current status; further improvements, of approximately 5% of peak speed, may be achieved.

Fermion	*Sites/node*	*Application*		
		Simulator D (%)	*128-node QCDOC*	
			D (%)	CG (%)
Wilson	2^4	47	44	32
	4^4		43	38
	6^4		44	39
	8^4		29	
ASQTAD	2^4		22	19
	4^4	43	42	40
	6^4		37	36
	8^4		29	28
Clover	2^4			31
	4^4			47
DWF	$2^4 \times 4$			32
	$4^4 \times 4$			42

the dominant computational load is in the Krylov space solvers used to solve the Dirac equation, much effort has gone into optimizing the Dirac operator. There are a number of different discretizations of the Dirac operator in current use, and these are known as different fermion types in LQCD. **Table 2** shows the performance on QCDOC for some of the common fermion types in use today: Wilson fermions, a-squared tadpole (ASQTAD) staggered fermions, clover fermions, and domain wall fermions (DWF). The performance is given for the gate-level simulations done during the ASIC design and on 128-node QCDOC machines. All calculations are done in double precision, and the results represent our status to date. There are additional improvements that can be made in some cases that could increase the performance by an additional 5% of peak. The column labeled D shows the performance achieved for the application of the Dirac operator to a vector. The column labeled CG shows the performance achieved for a full conjugate gradient solution to the problem $Dx = y$, given y. This is slightly below the Dirac operator performance due to the smaller number for operations per memory access in vector inner products than in matrix $\times$ vector calculations.

The different Dirac operators and volumes in Table 2 represent noticeably different challenges for achieving high performance. The ASQTAD case for small volumes provides a challenge because of the small amount of local computation relative to internode communications bandwidth. For Wilson fermions, in addition to utilizing internode bandwidth efficiently, the need to project from a four-component spinor to a two-component one, and reconstruct, presents a challenge to utilize the memory bandwidth between the PPC440 and the embedded DRAM to avoid processor stalls. For some of the 8^4 volumes, the calculation no longer fits in embedded DRAM, and the performance drops as the off-chip memory is used.

For machines with larger numbers of processors, only the global sum required in vector inner products increases with the processor number. The global operations capabilities of the SCU discussed above show that the time required for a global sum on a 4D lattice with the same size in each direction grows as $V^{1/4}$ as the volume increases. Machines with a larger number of processors should have very similar performance results, except for the smallest volume, 2^4, where a slight drop in performance is expected.

Without resorting to writing assembly code, we have also optimized other parts of our C and C++ QCD codes on QCDOC. As an example, the fermion force term used in the importance–sampling code achieved a speed of 20% of peak with standard function inlining and a reuse of communications channels allocated in software. Compared with QCDSP, generic C and C++ code on QCDOC performs markedly better, minimizing the parts of our application code that must be handled in assembly.

Conclusions

QCDSP has provided a working example of a computer on the scale of 10K nodes that has reliably performed calculations in QCD for a number of years. QCDOC is an effort to extend this architecture to a higher level of integration, performance, and ease of use. Only one version of the QCDOC ASIC has been manufactured, and it has performed very well, running for weeks in 128-, 512-, and 1,024-node QCDOC machines with very small hardware error rates. About 15 1,024-node QCDOC machines have been assembled. Some are in final debugging, and a few are running physics successfully. Given our experience with moving to large QCDSP machines and the regular repetitive character of the QCDOC nodes and network, we expect larger QCDOC machines to function reliably.

At this stage of the QCDOC project, there are still some significant unknowns that will affect the final price/performance of the machine. We are currently running our 1,024-node machines at 420 MHz and have run

smaller machines reliably at 450 MHz. We will continue to work for the highest possible clock speed. With respect to our cost/performance goal, some unknowns remain, including the final clock speed that can be used for a machine with thousands of processors and the cost of some of the components that are only now being procured in quantity. However, barring unusual difficulties, we expect to achieve our original design goal of a cost/performance of $1 per sustained megaflops for a QCD computer of the 10-teraflops scale.

Acknowledgments

We would like to thank the U.S. Department of Energy, the RIKEN–BNL Research Center, and the Particle Physics and Astronomy Research Council (PPARC) of the U.K. for funding the design and construction of the machines described in this paper.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Cadence Design Systems, Inc., The Open Group, Sun Microsystems, Inc., or Linus Torvalds in the United States, other countries, or both.

References

1. R. Tripiccione, "Dedicated Computers for LGT," *Nucl. Phys. B (Proc. Suppl.)* **17**, 137–145 (September 1990).
2. N. H. Christ, "QCD Machines," *Nucl. Phys. B (Proc. Suppl.)* **9**, 549–556 (June 1989).
3. D. H. Weingarten, "Parallel QCD Machines," *Nucl. Phys. B (Proc. Suppl.)* **26**, 126–136 (1992).
4. R. Ammendola, F. Bodin, P. Boucaud, N. Cabibbo, F. Di Carlo, R. De Pietri, F. Di Renzo, W. Errico, A. Fucci, M. Guagnelli, H. Kaldass, A. Lonardo, S. de Lucad, J. Micheli, V. Morenas, O. Pene, R. Petronzio, F. Palombi, D. Pleiter, N. Paschedag, F. Rapuano, P. De Riso, A. Salamon, G. Salina, L. Sartori, F. Schifano, H. Simma, R. Tripiccione, and P. Vicini, "Status of the apeNEXT Project," *Nucl. Phys. B (Proc. Suppl.)* **119**, 1038–1040 (May 2003).
5. T. Lippert, "Recent Developments and Perspectives of Machines for Lattice QCD," *Nucl. Phys. B (Proc. Suppl.)* **129/130**, 88–101 (March 2004).
6. I. Arsenin, D. Chen, N. H. Christ, R. Edwards, A. Gara, S. Hansen, A. Kennedy, R. D. Mawhinney, J. Parsons, and J. Sexton, "A 0.5 Teraflops Machine Optimized for Lattice QCD," *Nucl. Phys. B (Proc. Suppl.)* **34**, 820–822 (April 1994).
7. R. D. Mawhinney, "The 1 Teraflops QCDSP Computer," *Parallel Comput.* **25**, No. 10/11, 1281–1296 (September 1999).
8. D. Chen, N. H. Christ, C. Cristian, Z. Dong, A. Gara, K. Garg, B. Joo, C. Kim, L. Levkova, X. Liao, R. D. Mawhinney, S. Ohta, and T. Wettig, "QCDOC: A 10-Teraflops Scale Computer for Lattice QCD," *Nucl. Phys. B (Proc. Suppl.)* **94**, No. 1/3, 825–832 (March 2001).
9. P. A. Boyle, C. Jung, and T. Wettig, "The QCDOC Supercomputer: Hardware, Software, and Performance," *Proceedings of the Conference on Computers in High Energy Physics (CHEP03)*, June 2003, pp. 1–11.
10. P. A. Boyle, D. Chen, N. H. Christ, M. Clark, S. D. Cohen, C. Cristian, Z. Dong, A. Gara, B. Joo, C. Jung, C. Kim, L. Levkova, X. Liao, G. Liu, R. D. Mawhinney, S. Ohta, K. Petrov, T. Wettig, and A. Yamaguchi, "Hardware and Software Status of QCDOC," *Proceedings of the 21st International Symposium on Lattice Field Theory*, 2003, pp. 838–843.
11. K. A. Perrine, D. R. Jones, P. Hochschild, and R. A. Swetz, "Interactive Parallel Visualization Framework for Distributed Data," *Proceedings of the SPIE Conference on Visualization and Data Analysis*, 2002, pp. 196–206.

Received April 21, 2004; accepted for publication June 18, 2004; Internet publication April 7, 2005

Peter A. Boyle *School of Physics, University of Edinburgh, Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK (pab@physics.columbia.edu)*. Dr. Boyle received his Ph.D. degree in lattice field theory from the University of Edinburgh. He has been researching lattice quantum chromodynamics (QCD) since 1994, and has used a variety of massively parallel processor (MPP), cluster, and vector high-performance computing platforms. Dr. Boyle is the author of the cross-platform domain-specific compiler for high-performance QCD assembler kernels; he has spent the last four years working on testing, operating systems, and hardware debug on the QCDOC project at Columbia University.

Dong Chen *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (chendong@us.ibm.com)*. Dr. Chen is a Research Staff Member in the Exploratory Server Systems Department. He received his B.S. degree in physics from Peking University in 1990, and M.A., M.Phil., and Ph.D. degrees in theoretical physics from Columbia University in 1991, 1992, and 1996, respectively. He continued as a postdoctoral researcher at the Massachusetts Institute of Technology from 1996 to 1998. In 1999 he joined the IBM Server Group, where he worked on optimizing applications for IBM RS/6000* SP* systems. In 2000 he moved to the IBM Thomas J. Watson Research Center, where he has been working on many areas of the Blue Gene/L supercomputer and collaborating on the QCDOC project. Dr. Chen is an author or coauthor of more than 30 technical journal papers.

Norman H. Christ *Department of Physics, Columbia University, 538 West 120th Street, New York, New York 10027 (nhc@phys.columbia.edu)*. Professor Christ received a B.A. degree and a Ph.D. degree in physics from Columbia University in 1965 and 1966, respectively. He is currently a Professor of Physics at Columbia University. His major focus of research is the study of nonperturbative phenomena in quantum chromodynamics (QCD), addressing frontier questions in particle and nuclear physics. His research is performed using powerful, custom-built massively parallel computers and the techniques of lattice gauge theory. Professor Christ has played a leading role in the design and construction of some of the most powerful QCD-targeted computers since the early 1980s.

Michael A. Clark *School of Physics, University of Edinburgh, Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK (mike@ph.ed.ac.uk)*. Mr. Clark is a graduate student in the School of Physics at the University of Edinburgh. He received his M.S. degree in physics from the University of Edinburgh in 2001. He was involved in the development of the QCDOC supercomputer at Columbia University in 2003 and 2004.

Saul D. Cohen *Department of Physics, Columbia University, 538 West 120th Street, New York, New York 10027 (sdcohen@phys.columbia.edu)*. Mr. Cohen graduated from the University of Maryland at College Park in 2001 with degrees in physics, astronomy, mathematics, and computer science. He is currently pursuing a Ph.D. degree in physics at Columbia University with the Lattice Gauge Theory Group.

Calin Cristian *Citigroup Quantitative Equity Trading, US Equity Derivatives, 390 Greenwich Street, 3rd Floor, New York, New York 10013 (calin.cristian@citigroup.com)*. Dr. Cristian is a Quantitative Analyst in the Quantitative Equity Trading Group at Citigroup. He received his B.S. degree in physics from the University of Bucharest in 1996, his M.S. degree from Boston University in 1998, and his Ph.D. degree from Columbia University in 2002. Dr. Cristian joined Citigroup in 2003.

Zhihua Dong *Department of Physics, Columbia University, 538 West 120th Street, New York, New York 10027 (dong@phys.columbia.edu)*. Dr. Dong is currently a Research Staff Member at Columbia University. He received his Ph.D. from Columbia in 1993, and since 1999 has been involved in system administration for the front-end host computers for QCD on digital signal processors (QCDSP). Dr. Dong has participated in the development of the QCDOC supercomputer since 2000.

Alan Gara *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (alangara@us.ibm.com)*. Dr. Gara is a Research Staff Member at the IBM Thomas J. Watson Research Center. He received his Ph.D. degree in physics from the University of Wisconsin at Madison in 1986. In 1998 Dr. Gara received the Gordon Bell Award for the QCDSP supercomputer in the most cost-effective category. He is the chief architect of the Blue Gene/L supercomputer. Dr. Gara also led the design and verification of the Blue Gene/L compute ASIC as well as the bring-up of the Blue Gene/L prototype system.

Bálint Joó *School of Physics, University of Edinburgh, Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK (bj@ph.ed.ac.uk)*. Dr. Joó received a first-class honors degree in computer science and physics and a Ph.D. degree in theoretical particle physics, both from the University of Edinburgh, in 1996 and 2000, respectively. He completed postdoctoral research at the University of Kentucky and Columbia University, and is currently at the University of Edinburgh. He has been a member of the Columbia QCDOC design team since mid-2000. Dr. Joó has been actively involved with the U.S. Department of Energy Scientific Discovery through Advanced Computing (SciDAC) software effort, playing a major part in the QDP++ and Chroma software projects since 2002.

Chulwoo Jung *Brookhaven National Laboratory, P.O. Box 5000, Upton, New York 11973 (chulwoo@phys.columbia.edu)*. Dr. Jung, a native of South Korea, received a Ph.D. degree in physics from Columbia University in 1998. He spent two years as a postdoctoral Research Scientist at the University of Maryland at College Park. Dr. Jung has been a part of the QCDOC collaboration since 2001, when he joined Columbia University and Brookhaven National Laboratory as a Research Staff Member.

Changhoan Kim *Department of Physics, Columbia University, 538 West 120th Street, New York, New York 10027 (chateau@phys.columbia.edu)*. Dr. Kim received his B.S. degree in physics from Seoul National University and his Ph.D. degree in theoretical physics from Columbia University in 2004. He has participated in the QCDOC project since 2000.

Ludmila A. Levkova *Indiana University, 117 West Swain Hall, Bloomington, Indiana 47401 (llevkova@indiana.edu)*. Dr. Levkova is a Postdoctoral Fellow at Indiana University. She received an M.S. degree in theoretical physics from Sofia

University, Bulgaria, in 1997, and a Ph.D. degree in theoretical physics from Columbia University in 2004. Dr. Levkova was involved with the QCDOC project in 2000 and 2001 during her graduate studies at Columbia University.

Xiaodong Liao *UBS Investment Bank, 677 Washington Boulevard, Stamford, Connecticut 06902 (sheldon.liao@ubs.com)*. Dr. Liao is currently a Research Analyst and Associate Director in the Program Trading Group at UBS Investment Bank. He received his B.S. degree in physics from Fudan University, Shanghai, China, and his Ph.D. degree in physics from Columbia University.

Guofeng Liu *Gluon Capital LLC, 377 Broadway, 10th Floor, New York, New York 10013 (gfliu@physics.columbia.edu)*. Dr. Liu received a B.S. degree in physics from Beijing University in 1997 and a Ph.D. degree in physics from Columbia University in 2003. He was the primary author of the QCDSP parallel file system and was actively involved in the design of the QCDOC ASIC. Dr. Liu is currently the chief architect of Gluon Capital LLC, specializing in computerized automated equity trading.

Robert D. Mawhinney *Department of Physics, Columbia University, 538 West 120th Street, New York, New York 10027 (rdm@physics.columbia.edu)*. Professor Mawhinney is an Associate Professor of Physics at Columbia University. He received his B.S degree in physics from the University of South Florida in 1980 and his Ph.D. degree in physics from Harvard University in 1987. He held postdoctoral positions at the University of Pittsburgh and Columbia University, joining the Columbia faculty in 1992. Professor Mawhinney's physics research is focused on the study of QCD using these large parallel computers. He has been involved in the hardware and software design of both QCDSP and QCDOC.

Shigemi Ohta *Institute of Particle and Nuclear Studies, The High Energy Accelerator Research Organization, Tsukuba, Ibaraki 305-0801, Japan (shigemi.ohta@kek.jp)*. Dr. Ohta is a member of the Theory Division of the Institute of Particle and Nuclear Studies in the High Energy Accelerator Research Organization (KEK) of Japan. He has been working in numerical lattice QCD since 1985. From 1987 to 1990 he did postdoctoral research under Professor Norman H. Christ at Columbia University. In the early 1990s he was involved with RIKEN and KEK procurements of supercomputers. Dr. Ohta has recently been involved with the 600-Gflops QCDSP computer at the RIKEN–Brookhaven National Laboratory (BNL) Research Center, Japan, and with the development of the QCDOC computer.

Konstantin Petrov *Brookhaven National Laboratory, P.O. Box 5000, Upton, New York 11973 (petrov@bnl.gov)*. Dr. Petrov received his Ph.D. degree in the field of theoretical and computational particle physics from Bielefeld University, Germany, in 2002. He subsequently joined the QCDOC development team as a Research Associate in the Nuclear Theory Group at the Brookhaven National Laboratory. His research interests include large-scale lattice simulations of finite-temperature quantum chromodynamics. Dr. Petrov is also involved in the U.S Department of Energy SciDAC project for lattice QCD.

Tilo Wettig *Institute for Theoretical Physics, University of Regensburg, 93040 Regensburg, Germany (tilo.wettig@physik.uni-regensburg.de)*. Dr. Wettig is a Professor of Physics at the University of Regensburg, Germany. His previous appointments have included the following: Associate Professor at Yale University, Fellow of the RIKEN–BNL Research Center, Japan, and postdoctoral positions in Germany at the Technical University of Munich and the Max-Planck-Institute in Heidelberg. Dr. Wettig received a Ph.D. degree in theoretical physics from Stony Brook University in 1994.

Azusa Yamaguchi *Department of Physics and Astronomy, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK (azusa@physics.columbia.edu)*. Dr. Yamaguchi received her Ph.D. degree from Ochanomizu University, Japan. She subsequently worked as a postdoctoral researcher in the Computer Science Laboratory at the RIKEN Laboratory in Tokyo, and she was a Research Staff Member at the Particle Physics Laboratory at Ochanomizu University. Dr. Yamaguchi joined the QCDOC project at Columbia University as a postdoctoral researcher. She designed the digital logic for the internode communication in the QCDOC ASIC and the low-level link protocol, flow control, and retransmission logic for the QCDOC serial communications unit (SCU). She is currently a Research Fellow with the Particle Physics Theory Group of the University of Glasgow. Dr. Yamaguchi has been awarded a European Union Marie Curie Fellowship to study the nucleon dipole moment using QCDOC.