QCDOC: A 10 Teraflops Computer for Tightly-coupled Calculations

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, A. Yamaguchi

July 26, 2004

Abstract

Numerical simulations of the strong nuclear force, known as quantum chromodynamics or QCD, have proven to be a demanding, forefront problem in high-performance computing. In this report, we describe a new computer, QCDOC (QCD On a Chip), designed for optimal price/performance in the study of QCD. QCDOC uses a six-dimensional, low-latency mesh network to connect processing nodes, each of which includes a single custom ASIC, designed by our collaboration and built by IBM, plus DDR SDRAM. Each node has a peak speed of 1 Gigaflops and two 12,288 node, 10+ Teraflops machines are to be completed in the fall of 2004. Currently, a 512 node machine is running, delivering efficiencies as high as 45% of peak on the conjugate gradient solvers that dominate our calculations and a 4096-node machine with a cost of $1.6M is under construction. This should give us a price/performance less than $1 per sustained Megaflops.

1 Introduction

Commodity processors continue to advance in speed and fall in price, and consequently have become a major force in high-performance computing. The need for high-bandwidth connections between processors and local memory is as acute in the single-node desktop as in a multi-thousand node parallel computer. However, the network fabric of a large parallel computer has no analog in the commodity electronics market if the network is to achieve both high bandwidth and low latency. High bandwidth is an obvious requirement, but low latency is also vital if a problem of a fixed size is to be run on a machine with tens of thousands of nodes, since adding more nodes generally increases the ratio of inter-node communication to local floating point operations. Scaling of this type, commonly called hard scaling, is important in calculations where existing resources are insufficient to achieve the accuracy desired for problems already at hand. QCD is one such problem.

QCD simulations use numerical techniques common to many other large scale scientific problems. QCD simulations generally are done on a regular space-time grid of dimension four, with recent improved formulations actually requiring a five-dimensional grid. To study QCD, the Feynman path integral approach is taken, where an importance sampling is done of the phase space of the system. All known importance sampling techniques for QCD involve the solution of a large, sparse linear equation, the Dirac equation. This equation governs the propagation of a quark in a background gluon field and is a generalization of

0-7695-2153-3/04 $20.00 (c) 2004 IEEE

the propagation of an electron in a background photon field. Standard Krylov space solvers work well to produce the solution and dominate the calculational time for QCD simulations.

When naively discretized, the Dirac equation becomes a matrix equation where the matrix connects nearest neighbors in the coordinate space of the physics problem. More sophisticated discretizations, which reduce errors from the finite size of the grid spacing, may require second or third nearest-neighbor communications in the physics problem grid. In either case, the communications requirements are easily met by a computer with a regular Cartesian grid network, provided it is augmented by a fast, global sum.

The iterative steps in Krylov solvers require application of the matrix (the Dirac operator) to various vectors distributed over the entire machine. For hard scaling, as the number of processors increases, the number of local, on-node floating point operations falls and the size of data transfers with the nearest neighbors also decreases. Since many small transfers must be performed in a short period of time, a low-latency network is required. Also, inner-products of vectors over the entire machine are needed in Krylov space solvers. Since in the hard scaling limit, more CPU power is being applied to a problem of fixed size, the fraction of time spent in global operations can become large unless fast global operations are supported.

The considerations above make a mesh network ideal for QCD applications, provided the global operations are also well supported. Another aspect of QCD simulations that makes a mesh network useful is the natural load-balancing in the problem. Although the fundamental equations of QCD are non-linear, the solution of the Dirac equation (a linear equation) requires the same number of floating point operations on each processing node. Thus, no load balancing is needed beyond the initial trivial mapping of the physics coordinate grid to the machine mesh. On a four-dimensional machine, each processor becomes responsible for the local variables associated with a space-time hypercube.

Thus as processing power continues to improve, designing a machine that performs well under hard scaling becomes a matter of producing a network that has a low enough latency and high enough bandwidth that processors do not have to wait for data, even when very small amounts of data are being transferred. Densely packaging components aids in achieving low latency. The same dense packaging also improves reliability and is achievable if the overall electrical power consumption per node is modest. These considerations show why commerical cluster solutions have limitations for QCD, since one cannot achieve the required low-latency communications with commodity hardware currently available.

The above arguments have been well understood for quite some time. An earlier computer, QCDSP (Quantum Chromodynamics on Digital Signal Processors), was designed by some of the authors and it incorporated a low-latency four-dimensional mesh network to realize peak speeds of 1 Teraflops with 20,000 nodes [1, 2]. Two large QCDSP machines were built, an 8,192 node QCDSP at Columbia University, funded by the U.S. Department of Energy (DOE) and a 12,288 node QCDSP at Brookhaven National Laboratory (BNL), funded by the RIKEN lab in Japan and used by researchers at the RIKEN-BNL Research Center (RBRC). The RBRC QCDSP achieved a price performance of \$10/sustained Megaflops and won the Gordon Bell prize in price/performance at SC 98. Another series of QCD machines, the APE computers, have been designed and constructed by a collaboration in Italy [3].

Following on the success of QCDSP, a new generation of massively parallel computers, QCDOC [4], designed to meet these same architectural goals, but with a price performance of \$1 per sustained Megaflops, was begun. In this paper, we will describe the QCDOC architecture, which incorporates a six-dimensional processor mesh, and current progress towards reaching our goal of 10,000+ node QCDOC machines with a speed of 10 Teraflops. Three 12,288 node QCDOC machines will be built: one for the RBRC at BNL and funded by the RIKEN laboratory in Japan; one for the UKQCD collaboration to be located in Edinburgh, Scotland and funded by PPARC; and one for the US Lattice Gauge Theory community to be located at BNL and funded by the U.S. Department of Energy. A separate evolution of the regular mesh architecture of QCDSP is represented by IBM's Blue Gene/L supercomputer, which uses a three-dimensional mesh,

will incorporate up to 64,000 processing nodes and has a peak speed of 360 Teraflops [5]. This machine is targeted at a wide variety of scientific problems, including protein folding, and already has two entries on the top 500 list of supercomputers. Some members of the Blue Gene/L design team are also collaborators on QCDSP and QCDOC, representing an important cross-fertilization between academic and corporate researchers.

2 QCDOC Hardware Overview

Following the general architecture goals outlined in the previous section, the design of QCDOC began in November of 1999. QCDSP had shown that machines of size $\approx 10,000$ nodes could be built and reliably operated by a small number of researchers. The use of processing nodes with modest electrical power consumption was a vital part of this architecture, since this allows for dense packaging, simplified cooling and low time-of-flight for signals between nodes. IBM's system-on-a-chip technology was an ideal match for our design, allowing a single ASIC to be built which contained all hardware functionality for our nodes except for a commodity external DDR SDRAM.

The success of QCDSP made a larger community of QCD researchers interested in working on the development of QCDOC. Design work began with a group of researchers at Columbia University, the RBRC, IBM's T.J. Watson Research Laboratory in Yorktown Heights and rapidly grew to include members of the UKQCD collaboration in the United Kingdom. Our connection with IBM was two-fold: 1) a research design collaboration with significant participation from members of the research staff at Yorktown Heights and 2) an external ASIC customer with IBM's Microelectronics Division. Funding for QCDOC's research and development has come from the U.S. DOE, the RIKEN lab/RBRC and the PPARC in the U.K. Another vital part of this larger collaboration was a larger pool of researchers to contribute to the design.

The largest part of the R&D effort went into the design of the QCDOC ASIC, shown in Figure 1. The unshaded components in the figure are part of IBM's standard system-on-a-chip technology for external ASIC customers. The shaded components were custom designed for QCDOC and were written in VHDL. The entire design was simulated extensively using cycle-level accurate simulators, both at the level of the individual components shown as blocks in the figure, and also as an integrated entity. Particularly during these simulations of the entire ASIC we used programs written in C, C++ and PPC assembler which are common with dominant portions of both our O/S and application suite, and these were executed by the simulated processor during our verification of the design. Many tests were written to specifically create targeted corner cases and error conditions and test important logic which is rarely used. Before releasing the chip to manufacturing, we also performed simulations with back-annotated timing files, which reflected the true wire loadings and propagation delays for the cells as they were actually implemented in the silicon for the chip.

We now turn to a brief discussion of the major systems in the QCDOC ASIC.

2.1 Processor and Memory

The processor in the QCDOC ASIC is an IBM PPC 440, a 32 bit integer unit compliant with IBM's Book-E specifications, and it has a 64 bit, IEEE floating point unit attached. The floating point unit is capable of one multiply and one add per cycle, giving a peak speed of 1 Gflops for a 500 MHz clock speed. The PPC 440 has instruction and data caches of 32 kBytes each.

IBM provides a Processor Local Bus (PLB) for connecting the major components of a system-on-a-chip design. The PPC 440 has read/write access to this bus from the data cache and read access from the instruction cache. For the QCDOC ASIC, we have retained the PLB bus for interconnection of the major

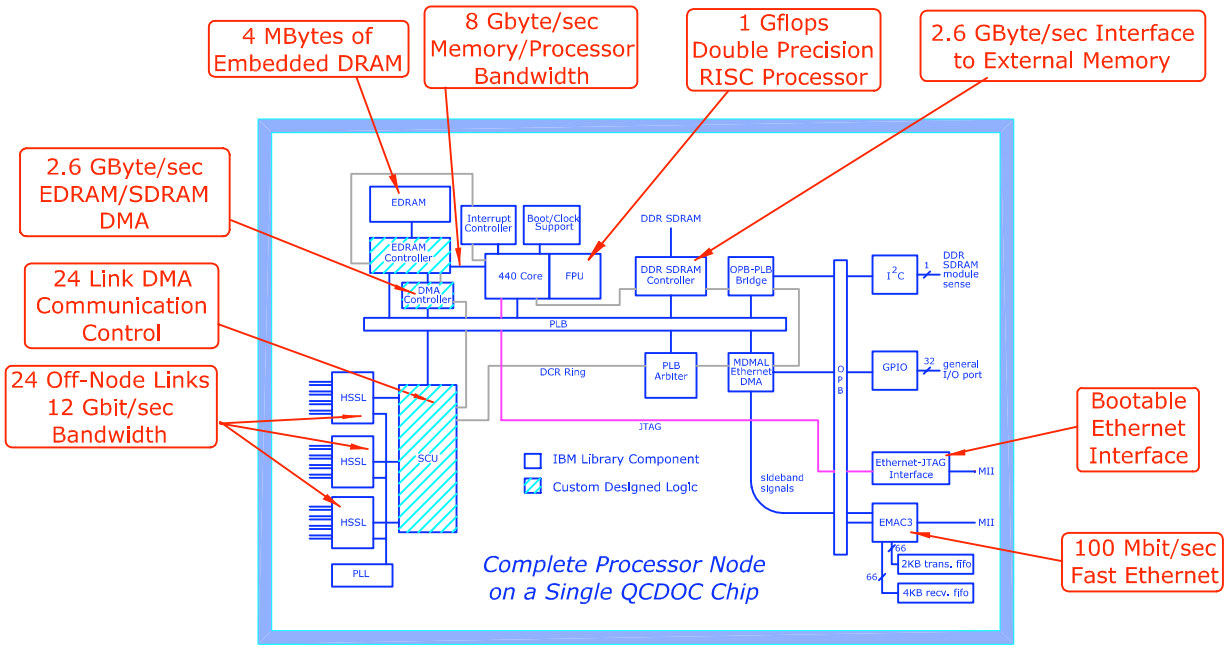

Figure 1: The QCDOC ASIC designed with IBM's system-on-a-chip technology. The shaded components in the figure were designed specially for QCDOC, while the remaining components are standard IBM macros.

subsystems of the ASIC, but have modified the PPC 440's data cache access to the PLB. This modification, designed by members of our collaboration at IBM Yorktown Heights, has the data bus connection first going to a prefetching EDRAM (Embedded DRAM) controller and then, if the access requires it, going out onto the PLB bus.

The prefetching EDRAM controller provides access to the 4 MBytes of on-chip memory. The EDRAM supports memory reads and writes of width 1024 bits, plus additional ECC bits. The EDRAM controller combines these 1024-bit words and provides the data cache connection to the PPC 440 with 128 bit words at the full speed of the processor. (Conventionally the data cache connection would run at 1/3 of the processor speed.) This gives us a maximum bandwidth of 8 GBytes/second between the processor and EDRAM.

To avoid page miss delays in the EDRAM and to utilize this large bandwidth, the EDRAM controller maintains two prefetching streams, each stream fetching from a group of contiguous memory addresses. Thus for an operation involving $a(x) \times b(x)$, where $a(x)$ and $b(x)$ are individually contiguous in memory, the EDRAM controller will fetch data from the EDRAM without suffering excessive page miss overheads.

For memory accesses outside the memory space of the EDRAM, the PPC 440 data cache request is sent to the main PLB bus in the ASIC. Also attached to the PLB bus is a controller for external DDR SDRAM, with a bandwidth of 2.6 BGytes/second. Up to 2 GBytes of memory per node can be used.

2.2 Mesh Communications Network

The QCDOC ASIC contains a Serial Communications Unit (SCU) to drive the six-dimensional nearest-neighbor network that makes our individual processing nodes into a tightly coupled massively parallel machine. While QCD has four- and five-dimensional formulations, we chose to make the mesh network six dimensional, so we can make lower-dimensional partitions of the machine in software, without moving cables. The dimensionality could not be larger than six, due to limitations in the number of cables that can reasonably be connected to our motherboards. There are 12 nearest neighbors in a six-dimensional network

and QCDOC supports concurrent sends and receives to each individual neighbor. Thus the SCU manages 24 independent, uni-directional connections concurrently.

The fundamental physical link in the QCDOC communications network is a bit-serial connection between neighboring nodes in the six-dimensional mesh. This link runs at the same clock speed as the processor, i.e. a target speed of 500 MHz. This physical link is managed by IBM system-on-a-chip components called HSSL (High Speed Serial Link) controllers. When powered on and released from reset, these HSSL controllers transmit a known byte sequence between the sender and receiver on the link, establishing optimal times for sampling the incoming bit stream and determining where the byte boundaries are. Once trained, the HSSL controllers exchange so-called idle bytes when data transmission is not being done.

The HSSL controllers are in turn controlled by the SCU unit of the QCDOC ASIC, which was designed by members of our collaboration. Over a given physical link to a neighboring node, the SCU is responsible for coordinating three types of data transfer:

1. Normal data transfers of 64 bit data words. The SCU's have DMA engines allowing block strided access to local memory. In a normal data transfer, data is fetched from local memory on one node, passes through the SCU on that node, is serialized by the HSSL controller on that node, is received by a neighboring node and placed into its local memory via its SCU DMA engine. The low latency for communications in QCDOC is a consequence of the DMA engine having direct access to the local memory on a node. Data is not copied to a different memory location before it is sent, rather the SCUs are told the address of the starting word of a transfer and the SCU DMA engines handle the data from there. This leads to a memory-to-memory transfer time of about 600 ns for a nearest neighbor transfer.

2. Supervisor data transfers where a single 64 bit word, called a supervisor packet, is sent to a register in a nearest-neighbor's SCU. The arrival of the supervisor packet causes an interrupt to be received by the neighbor's CPU. This supervisor capability provides a mechanism for one processor to send an interrupt to its neighbors.

3. Partition interrupt packets of 8 bits. Since QCDOC can be partitioned into machines of lower dimensionality, a mechanism is needed to send an interrupt to all nodes in a partition. The partition interrupt packets are this mechanism. If a node receives a partition interrupt packet its SCU forwards this packet on to all of its neighbors if the packet contains an interrupt which had not been previously sent. This forwarding is done during a time interval controlled by a relatively slow global clock, which also controls when interrupts are presented to the processor from the SCU. This global clock period is set so that during the transmit window, any node that sets an interrupt will know it has been received by all other nodes before the sampling of the partition interrupt status is done.

Each of these three types of data can be multiplexed onto a physical link to a nearest neighbor node. The supervisor packets take priority over normal data transfers. The type of packet that is being sent is encoded into an 8 bit packet header, with codes determined so that a single bit error will not cause a packet to be misinterpreted. The packet header also contains two parity bits for the data sent and a single bit error causes an automatic resend in hardware. In addition, checksums at each end of the link are kept, so at the conclusion of a calculation, these checksums can be compared. This offers a final confirmation that no erroneous data was exchanged during the course of a calculation.

All data that is sent to a neighbor is acknowledged. For normal data transfers, up to three, 64 bit data words can be sent before an acknowledgement is given. This "three in the air" protocol allows full bandwidth to be achieved between nodes, and amortizes the time for a round-trip handshake to be done.

After the initial training of the HSSL units, the SCU controllers are placed in a mode called idle receive. In this mode, if a neighbor sends data before the receiver has been programmed with information about where the data will be placed in memory, the receiver holds the first three words received in an SCU register and does not issue an acknowledgement. This blocks the sender from sending further data until the receiver's SCU unit is given a destination in memory. This idle receive feature means that there need be no temporal ordering between software issuing a send on one node and a receive on another.

This acknowledgement of every data packet exchanged makes QCDOC self-synchronizing on the individual link level. In a tightly coupled application involving extensive nearest-neighbor communications, if a given node stops communicating with its neighbors, the entire machine will shortly become stalled. Once the initial blocked link resumes its transfers, the whole machine will proceed with the calculation. This link-level handshaking also allows one node to get slightly behind in a uniform operation over the whole machine, say due to a memory refresh. Provided the delay due to the refresh is short enough, the majority of the machine will not see this pause by one node.

Global operations The SCU units also provide a mechanism for fast global sums and broadcasts. In its global mode, the SCU controllers can perform functions involving the 12 incoming and 12 outgoing links on a node. Incoming data from one link can be sent out to any combination of the other 11 links and also stored to local memory. Thus for a global broadcast, one node starts sending a given 64 bit word out on up to 12 of its links. The other nodes receive the word, store it in memory and send it out on some set of their links. The pattern of links is chosen to rapidly span the entire machine.

Since the physical links are bit-serial, there would be a large latency per node passed through in global operations if each word was fully assembled on a node and then sent out again as another 64 bit transfer. By doing this pass-through in the SCUs only 8 bits must be received before they are sent out to neighbors, markedly reducing the latency.

The global functionality of the SCUs is doubled, *i.e.* two disjoint sets of links can be chosen as participants in a global operation. This effectively halves the size of the machine.

For global sums, every node is assumed to have one word to contribute to the sum. To perform a four-dimensional global sum on a four-dimensional machine, consider the x direction first, which is assumed to have N_x processors in it. Every node would place its word in a register used for sending, start a send in the x direction and then receive $N_x - 1$ words from its neighbors. At this point all nodes with the same y, z and t coordinates would have the same data to sum locally. This pattern would then be repeated for the y, z and t directions. This sequence achieves a global sum by having data hop between $N_x + N_y + N_z + N_t - 4$ nodes. Using the doubled functionality of the SCUs global modes, the sum can be reduced to requiring $N_x/2 + N_y/2 + N_z/2 + N_t/2$ hops.

In summary, the SCU units on each QCDOC ASIC handle simultaneous, bi-directional nearest-neighbor communications to 12 other QCDOC ASICs. The total bandwidth is 1.3 GBytes/second at 500 MHz. Memory-to-memory transfer times for nearest neighbors are around 600 ns. Thus for transfers as small as 24, 64 bit words to a neighbor, the latency of 600 ns for the first word is still small compared to the 3.3 μs time for the remaining 23 words. Our 600 ns memory-to-memory latency is to be compared to times of 5-10 μs just to begin a transfer when using standard networks like Ethernet.

2.3 Booting, Diagnostics and I/O

There are many other hardware features of QCDOC, which are described in Reference [6, 7]. Here we will mention those associated with booting, diagnostics and I/O. A schematic diagram of QCDOC, showing its networks, is given in Figure 2. The processing nodes (QCDOC ASIC plus DDR SDRAM DIMM) are the small, open black squares. They are connected by the SCU driven mesh network, shown in red. (Only a

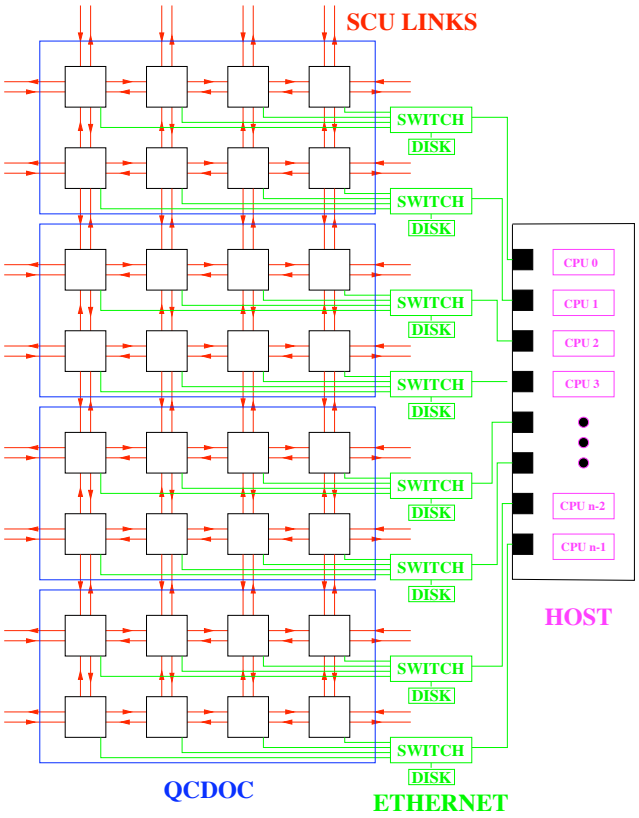

Figure 2: A schematic showing the various networks in QCDOC.

two-dimensional slice of the SCU network can be easily represented.) The larger blue square represents the various processing nodes that are grouped together on a QCDOC motherboard. Each node is connected to a standard Ethernet system, shown in green in the figure. This Ethernet system connects QCDOC to an SMP host and also to external disks.

Each QCDOC ASIC contains two Ethernet connections. One connection drives a standard 100 Mbit Ethernet controller on the ASIC which, with appropriate software programs, responds to conventional Ethernet packets. The second connection receives only UDP Ethernet packets and, in particular, only responds to Ethernet packets which carry Joint Test Action Group (JTAG) commands as their payload. This second Ethernet/Jtag connection uses circuitry designed by Peter Hochschild and Richard Swetz of IBM Yorktown Heights that requires no software to do the UDP packet decoding and manipulate the JTAG controller on the ASIC according to the instructions in the UDP packet.

While physics calculations utilize the high-bandwidth, low-latency network shown in red, booting, diagnostics and I/O are done via the Ethernet network. Since the Ethernet/JTAG controller is ready to receive packets after power on, it is used to load code into the QCDOC ASIC. (There are no PROMs on QCDOC.) We load a boot kernel into the ASIC, do basic hardware testing of the ASIC and then load a run kernel via the 100 Mbit Ethernet connection. At this point, the nodes all have access to the disk resources distributed in the Ethernet network or on the host.

The Ethernet/JTAG controller also gives us a powerful tool for hardware and software debugging. We can use the Ethernet/JTAG controller to provide the physical transport mechanism required for IBM's standard RISCWatch debugger. Thus a user can debug and single step code on a given node. For hardware debugging, this same mechanism offers us an I/O path to monitor and probe a failing node.

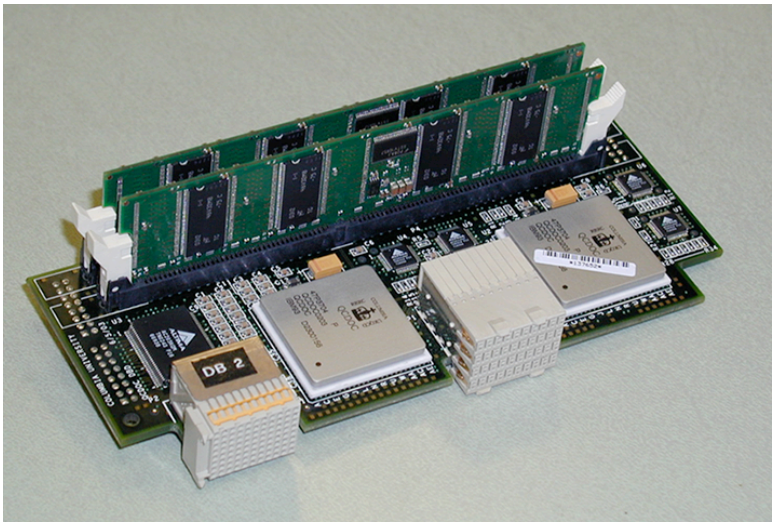

Figure 3: A QCDOC daughter board which holds two nodes and their associated DIMMs (512 Mbyte DIMMs shown).

2.4 Integration

The first QCDOC ASICs were received in June of 2003. They have undergone extensive testing since then and no errors have been found to date. Two ASICs are mounted on a single, 18 layer printed circuit board, which we call a daughterboard, along with a 5 port Ethernet hub and two DDR SDRAM DIMMs. The entire assembly is 3" $\times$ 6.5", is fed by 1.8, 2.5 and 3.3 volt DC power and consumes about 20 Watts for both nodes, including the DRAMs. Figure 3 shows a picture of our daughterboard.

We then plug 32 daughterboards into a motherboard of size 14.5" $\times$ 27". A fully populated motherboard is shown in Figure 4. There is relatively little additional electronics on a motherboard. The motherboard has DC to DC power converters taking the 48 volt supplied to the motherboard and producing the 1.8, 2.5 and 3.3 volts required by the ASICs, DRAMs and other chips. The motherboard also distributes the global clock (around 40 MHz) and has 5 port hubs for distributing the Ethernet. For the high speed signals of the mesh network, the motherboard provides a matched impedance path from the ASIC's, through the motherboards, through external cables, onto another motherboard and to the destination ASIC. No redrive is done for these signals and we find they have very good electrical properties even after going through many meters of external cables.

Eight motherboards are arranged into a single crate, and two crates are placed into a rack. The racks include internal heat exchangers and the first populated rack is shown in Figure 5. When populated, this water-cooled rack gives a peak speed of 1.0 Teraflops and consumes less than 10,000 watts. For our final machines, the water-cooled racks can be stacked, allowing 10,000 nodes to have a footprint of about 60 square feet.

3 Software

There are three broad categories for the software for QCDOC. The first is the software which runs on the host computer, an IBM p-Series SMP, and manages QCDOC from booting through running user applications. The second is the operating system software that executes on the individual nodes of QCDOC. Finally there

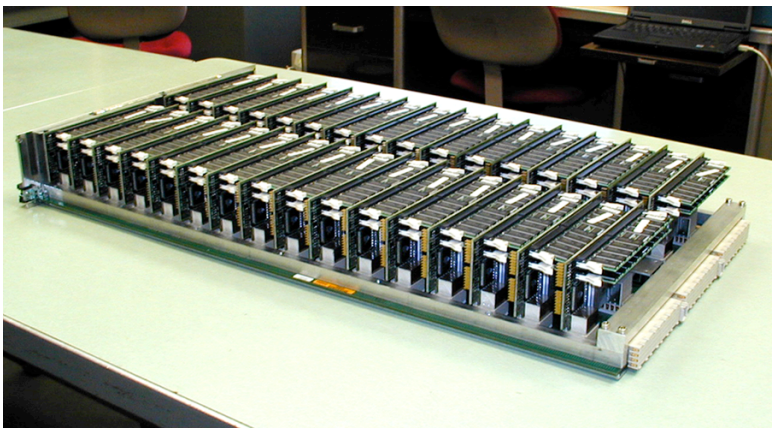

Figure 4: A QCDOC mother board which holds 64 nodes as a 2^6 hypercube.

Figure 5: A QCDOC water-cooled rack which hold 1024 nodes on 16 motherboards mounted in two identical eight-motherboard crates.

is the software that is visible to the user and provides user application support. We will briefly describe these three major categories.

3.1 Host software

Since we are building and maintaining a large parallel computer in QCDOC, we wanted to use an industry standard host computer. QCD does not place large demands on the host; I/O for QCD applications is quite modest for the compute power needed, the data generated is relatively simple in form and jobs run for long periods of time. Thus we sought to write our host-side software to make use of a commercial SMP.

Our primary host software is called the qdaemon. This software is responsible for booting QCDOC, coordinating the initialization of the various networks, keeping track of the status of the nodes (including hardware problems), allocating user partitions of QCDOC, loading and starting execution of applications, and returning application output to the user. The physical connection to QCDOC is via multiple Gigabit Ethernet links. To allow the size of the host machine to be flexible, the qdaemon is heavily threaded software which should take advantage of commercial SMPs. The qdaemon's interactions with QCDOC generally involve the exchange of many small UDP packets, placing some demand on the networking capability of the host.

During the initial boot of QCDOC, each node receives about 100 UDP packets that are handled by the Ethernet/JTAG controller. These packets contain code that is written directly into the instruction cache of the PPC 440. When executed, this code does basic hardware tests of the ASIC and attached DRAM and initializes the standard Ethernet controller. Then the run kernel is loaded down, also taking about 100 UDP packets. The run kernel initializes the SCU controllers and the mesh network, checks the functionality of the partition interrupts and determines the six-dimensional size of the machine. At this point, QCDOC is ready for applications to run. All subsequent communications between the host and nodes uses the RPC protocol.

The command line interface to QCDOC is a modified UNIX tcsh, which we we call the qcsh. The qcsh runs with the UID of the application programmer, gathers commands to send to the qdaemon and manages the returning data stream. A subprocess of the qcsh is also available to the qdaemon, so the qdaemon can request files on the host to be opened and they will have the permissions and protections of the application programmer.

The qdaemon can manage many different partitions of QCDOC and has the capability to carve out lower dimensional partitions of QCDOC from the native six-dimensional mesh. A user requests that the qdaemon remap their partition to a dimensionality between one and six, before program execution begins. Of course, for a large QCDOC machine, many partitions could be created but in practice only a few will be active at any time to avoid underutilization of the available processors.

3.2 Node software

We have chosen to write our own lean, run-time kernel for QCDOC. A complete Linux system on each QCDOC node is easy to do, but comes with many overheads which do not directly help us in our application. In particular, when running our highest performance programs to solve the Dirac equation, our performance would suffer markedly if various nodes were handling internal Linux system housekeeping. There are embedded real-time kernels available, but they do not offer one of the most important features our software must provide - the ability to monitor and report on the hardware status.

Our node run kernels provide essentially two threads - a kernel thread and an application thread. For QCD, we have no reason to multitask on the node level, so the run kernels do not do any scheduling. During

booting, initialization and debugging, the kernel thread is active. Once a user application is started, the thread switches to the application, until a system call is made by the application. The kernel services this request and then returns control to the application thread. Upon program termination, the kernel thread is reinvoked and it checks on hardware status and reports back to the qdaemon and user.

The kernel manages various hardware devices. For the standard Ethernet controller, the kernel uses a custom implementation of the sockets UDP interface. For the SCUs, we have a message passing API that directly reflects the underlying hardware features of our communications unit. The memory management capabilities of the PPC 440 are used to protect memory from unintended access, but not to translate addresses. This allows the use of the SCU DMAs in a zero-copy fashion without the introduction of additional zero-copy hardware features (such as the hardware page table walk in the Quadrics Elan chip or Infiniband). The kernel also includes support for NFS mounting of remote disks, which is already being used by application programs to write directly to the host disk system.

3.3 User software

We have tried to keep the software environment that applications see as standard as possible. The general applications environment involves using a single node compiler and message passing to use the parallel characteristics of QCDOC. At the single node level, both gcc and xlc compilers are supported. For the run-time environment we have used an open-source POSIX-compliant C run-time library from Cygnus. This library requires a custom implementation of libgloss (also known as GNU Low Level OS Support), which we have done using our home-grown user space support (generally a system call to the kernel).

The communications API allows the user to control the settings of the DMA units in the SCUs. In particular, for repetitive transfers over the same link, the SCU's can store DMA instructions internally, so that only a single write (start transfer) is needed to start up to 24 communications. This capability is reflected in our communications API. As noted in the description of the SCU hardware, the temporal ordering of a start send on one node and start receive on a neighbor is not important since the receiver will hold the first three words sent even if the receive transaction has not yet begun. We also have API interfaces to the global sum and broadcast features of the SCU hardware.

4 Performance for QCD

At the time of this writing (July, 2004), we have successfully run our QCD application on 64, 128 and 512 node QCDOC machines for many weeks. Part of our final verification of the correct functioning of the QCDOC ASIC was to run some of our application programs which evolve a QCD system through the phase space of the Feynman path integral. A five day simulation was completed on a 128 node machine in December, 2003 and then redone, with the requirement that the resulting QCD configuration be identical in all bits. This was found to be the case. No hardware errors on the SCU links were reported.

We have recently received the motherboards and daughterboards needed to assemble the 1024 node QCDOC machine shown in Figure 5. All nodes in this machine are cabled together in a single six-dimensional mesh, giving a machine of size $8 \times 4 \times 4 \times 2 \times 2 \times 2$. This 1024 node machine is passing basic tests and is in the process of final debugging. We have two additional racks in our machine room, with nodes expected for them in August. With the addition of a final rack, a 4096 node (4 Teraflops) QCDOC should be assembled at Columbia in August. The parts for the 12,288 node QCDOC for the RBRC and the separate 12,288 node for the UKQCD Collaboration will begin to arrive shortly and the machines will be assembled at BNL this fall. Parts ordering for the US Lattice Gauge Theory machine has begun and it should be completed early next calendar year.

Our current performance figures come from solving the Dirac equation, using a conjugate gradient solver, on a 128 node QCDOC. There are a number of discretizations of the Dirac operator available. We have benchmarks ready for three of them: naive Wilson fermions, ASQTAD staggered fermions and clover improved Wilson fermions. On a 4^4 local volume, we sustain 40%, 38% and 46.5% of peak speed, respectively. These are full double precision calculations and performance for single precision is slightly higher due to the decreased bandwidth to local memory that is needed in this case.

The performance quoted above is achieved with carefully hand-tuned assembly code for the Dirac operator. A newer discretization of the Dirac operator, domain wall fermions, has been heavily used in our QCD simulations on QCDSP. This is a prime target for much of our work with QCDOC. This discretization is naturally five-dimensional, so we are finishing an assembly-coded version of this operator which we expect will surpass the performance of the clover improved Wilson operator mentioned above.

A 4^4 local volume is a reasonable size for machines with a peak speed of 10 Teraflops and translates into a $32^3 \times 64$ lattice size for a 8,192 node machine. For most of the fermion formulations, a 6^4 local volume still fits in our 4 Megabytes of imbedded memory. For still larger volumes, when we must put part of the problem in external DDR DRAM, the performance figures fall to the range of 30% of peak.

Although we have not received all the components for the combined 4096 node QCDOC machines, we can accurately know their cost since they have all been purchased on Columbia University purchase orders. The 2048 daughterbords cost \$1,105,692.67 (this provides 128 Mbytes of off-chip memory per node for one half of the nodes and 256 Mbytes for the other half), the 64 mother boards cost \$180,404.88, the four water cooled cabinets cost \$187,296 and the 768 cables for the mesh network cost \$71,040. Awaiting final accounting, the host computer, Ethernet switches and disks should cost \$64,300 (this provides 6 Tbytes of parallel RAID storage) for a total machine cost of \$1,610,442. The design and prototying costs required to develop these machines were \$2,166,000 which does not include academic salaries. If this cost is prorated over all of the presently funded QCDOC machines, this represents an additional cost of \$99,159 giving a total cost of this 4096-node machine of \$1,709,601.

For our 128 node benchmarks, we ran with a clock speed of 450 MHz and used buffered DDR SDRAMS. For our 512 node machine, as for all subsequent machines, we have moved to unbuffered memory since it is substantially cheaper. This has required further tuning of the memory controller on our ASIC. On our 512 node machines, we have successfully run QCD on the machine for days at 360 MHz, after tuning the memory controller. Recently we have successfully tuned the memory controller for reliable operation of the memory at 420 MHz.

For our current price performance estimates, we have run 128 nodes at a speed of 450 MHz. 512 nodes has performed reliably at 360 MHz and is in the process of being checked at 420 MHz. Using a cost of \$1,709,601 for our 4096 node QCDOC and a 45% efficiency for our Dirac operator, gives a price/performance of \$1.29 per sustained Megaflops for 360 MHz operation, \$1.10 per sustained Megaflops for 420 MHz operation and \$1.03 per sustained Megaflops for 450 MHz operation. For the full size 12,288 machines, the cost per node will be reduced, due to the discount from volume ordering for the larger number of parts. This should put us very close to our targeted \$1 per sustained Megaflops.

5 Summary

QCDOC is a tightly-coupled massively parallel computer designed for optimal price/performance for QCD. Making use of IBM's system-on-a-chip technology, the processing nodes for QCDOC are a single ASIC plus an external SDDR DRAM DIMM. Machines of sizes 64, 128 and 512 nodes have been assembled and tested and have run QCD reliably. The final clock speed for larger machines is not yet determined. A 1024 node

machine is currently being debugged and a 4096 node machine should be assembled by the end of August. Given the performance we are seeing on 128 node QCDOC machines, we expect to achieve our targeted performance of $1 per sustained Megaflops on the large, 12,288 node machines.

6 Acknowledgments

We would like to thank the U.S. Department of Energy, the RIKEN-BNL Research Center and PPARC of the UK for the funding the design and construction of the machines described in this article.

References

[1] I. Arsenin, *et. al.*, *A 0.5 Teraflops Machine Optimized for Lattice QCD*, Nuclear Physics **B34** (Proceedings Supplement) (1994), page 820.

[2] R. Mawhinney, *The 1 Teraflops QCDSP computer*, Parallel Computing **25** (1999) 1281; also http://lanl.arXiv.org, hep-lat/0001033.

[3] R. Ammendola, *et. al.*, *Status of the apeNEXT project*, http://arxiv.org/abs/hep-lat/0211031.

[4] D. Chen, *et. al.*, *QCDOC: A 10-teraflops scale computer for lattice QCD*, Nuclear Physics **B94** (Proceedings Supplement) (2001) page 825; also http://lanl.arXiv.org, hep-lat/0011004.

[5] `http://www.research.ibm.com/bluegene`

[6] P.A. Boyle, C. Jung and T. Wettig, *The QCDOC Supercomputer: hardware, software and performance*, 2003 Conference on Computers in High Energy Physics; http://lanl.arXiv.org, hep-lat/0306023.

[7] P.A. Boyle, *et. al.*, *Hardware and Software Status of QCDOC*, 2003 International Conference on Lattice Field Theory, Tsukuba, Japan; http://lanl.arXiv.org, hep-lat/0306096.