QPACE: Quantum Chromodynamics Parallel Computing on the Cell Broadband Engine

Application-driven computers for Lattice Gauge Theory simulations have often been based on system-on-chip designs, but the development costs can be prohibitive for academic project budgets. An alternative approach uses compute nodes based on a commercial processor tightly coupled to a custom-designed network processor. Preliminary analysis shows that this solution offers good performance, but it also entails several challenges, including those arising from the processor's multicore structure and from implementing the network processor on a field-programmable gate array.

1521-9615/08/$25.00 © 2008 IEEE
Copublished by the IEEE CS and the AIP

Gottfried Goldrian, Thomas Huth, Benjamin Krill, Jack Lauritsen, and Heiko Schick
IBM Research and Development Lab, Böblingen, Germany
Ibrahim Ouda
IBM Systems and Technology Group, Rochester, Minnesota
Simon Heybrock, Dieter Hierl, Thilo Maurer, Nils Meyer, Andreas Schäfer, Stefan Solbrig, Thomas Streuer, and Tilo Wettig
University of Regensburg, Germany
Dirk Pleiter, Karl-Heinz Sulanke, and Frank Winter
Deutsches Elektronen Synchrotron, Zeuthen, Germany
Hubert Simma
University of Milano-Bicocca, Italy
Sebastiano Fabio Schifano and Raffaele Tripiccione
University of Ferrara, Italy
Andrea Nobile
European Center for Theoretical Studies, Trento, Italy
Matthias Drochner and Thomas Lippert
Research Center Jülich, Germany
Zoltan Fodor
University of Wuppertal, Germany

Quantum chromodynamics (QCD) is a well-established theoretical framework to describe the properties and interactions of quarks and gluons, which are the building blocks of protons, neutrons, and other particles. In some physically interesting dynamical regions, researchers can study QCD perturbatively—that is, they can work out physical observables as a systematic expansion in the strong coupling constant. In other (often more interesting) regions, such an expansion isn't possible, so researchers must find other approaches.

The most systematic and widely used nonperturbative approach is lattice QCD (LQCD), a discretized, computer-friendly version of the theory Kenneth Wilson proposed more than 30 years ago.[1] In this framework, QCD is reformulated as a statistical mechanics problem that we can study using Monte Carlo techniques. Many physicists have performed LQCD Monte Carlo simulations over the years and have developed efficient simulation algorithms and sophisticated analysis techniques. Since the early 1980s, LQCD researchers have pioneered the use of massively parallel computers in large scientific applications, using virtually all available computing systems including traditional main-

frames, large PC clusters, and high-performance systems.

We're interested in parallel architectures on which LQCD codes scale to thousands of nodes and on which we can achieve good price-performance and power-performance ratios. In this respect, highly competitive systems are QCD On a Chip (QCDOC)[2] and apeNEXT,[3] two custom-designed LQCD architectures that have been operating since 2004 and 2005, respectively, and IBM's BlueGene[4] series. These architectures are based on system-on-chip designs. In modern technologies, however, VLSI chip development costs are so high that an SoC approach is no longer possible for academic projects.

To address this problem, our QCD Parallel Computing on the Cell (QPACE) project uses a compute node based on IBM's PowerXCell 8i multicore processor and tightly couples it to a custom-designed network processor in which we connect each node to its nearest neighbors in a 3D toroidal mesh. To facilitate the network processor's implementation, we use a Xilinx Virtex-5 field-programmable gate array (FPGA). In both price and power performance, our approach is competitive with BlueGene and has a much lower development cost. Here, we describe our performance analysis and hardware benchmarks for the PowerXCell 8i processor, which show that it's a powerful option for LQCD. We then describe our architecture in detail and how we're addressing the various challenges it poses.

Lattice Gauge Theory Computing

LQCD simulations are among scientific computing's grand challenges. Today, a group active in high-end LQCD simulations must have access to computing resources on the order of tens of sustained Tflops-years.

Accurately describing LQCD theory and algorithms is beyond this article's scope. LQCD simulations have an algorithmic structure exhibiting high parallelism and several features that make it easy to exploit much of this parallelism. LQCD replaces continuous 4D space–time with a discrete and finite lattice (containing $N = L^4$ sites for a linear size L). All compute-intensive tasks involve repeated execution of just one basic step: the product of the lattice Dirac operator and a quark field ψ. In LQCD, a quark field ψ_x^{ia} is defined at lattice sites $x = 1, \ldots, N$ and carries color indices $a = 1, \ldots, 3$ and spinor indices $i = 1, \ldots, 4$. Thus, ψ is a vector with $12N$ complex entries.

As Equation 1 shows, the hopping term of the Wilson Dirac operator D_{h} acts on ψ as follows (other discretization schemes exist, but the implementation details are all similar):

$$\begin{aligned}\psi'_x &= D_{\mathrm{h}}\psi_x \\ &= \sum_{\mu=1}^{4}\left\{\begin{array}{l} U_{x,\mu}\left(1+\gamma_\mu\right)\psi_{x+\hat{\mu}} \\ +\, U^{\dagger}_{x-\hat{\mu},\mu}\left(1-\gamma_\mu\right)\psi_{x-\hat{\mu}}\end{array}\right\}.\end{aligned} \quad (1)$$

Here, μ labels the four space–time directions, and the $U_{x,\mu}$ are the SU(3) gauge matrices associated with the links between nearest-neighbor lattice sites. The gauge matrices are themselves dynamical degrees of freedom and carry color indices (each of the $4N$ different $U_{x,\mu}$ has 3×3 complex entries). The γ_μ are the (constant) Dirac matrices, carrying spinor indices. Today, state-of-the-art simulations use lattices with a linear size L of at least 32 sites.

Roughly speaking, the ideal LQCD simulation engine is a system that can keep the previously defined degrees of freedom in local storage and repeatedly apply the Dirac operator with very high efficiency. Explicit parallelism is straightforward. As Equation 1 shows, D_{h} couples only nearest-neighbor lattice sites, which suggests an obvious parallel structure for an LQCD computer: a d-dimensional grid of processing nodes. Each node contains local data storage and a compute engine. We need only to map the physical lattice onto the processor grid as regular tiles (if $d < 4$, we fully map $4 - d$ dimensions of the physical lattice onto each processing node).

In the conceptually simple case of a 4D hypercubic grid of p^4 processors, each processing node would store and process a 4D subset of $(L/p)^4$ lattice sites. Each processing node handles its sublattice using local data. The nodes also access data corresponding to lattice sites that are just outside the sublattice's surface and are stored on nearest-neighbor processors. Processing proceeds in parallel for two reasons: there are no direct data dependencies for lattice sites that aren't nearest neighbor, and the same operations sequence runs on all processing nodes in lockstep mode.

Parallel performance will be linear in the number of nodes, as long as

- the programmer can evenly partition the lattice on the processor grid, and
- the interconnection bandwidth is large enough to sustain each node's performance.

We meet the first constraint easily for up to thousands of processors while the latter is less trivial

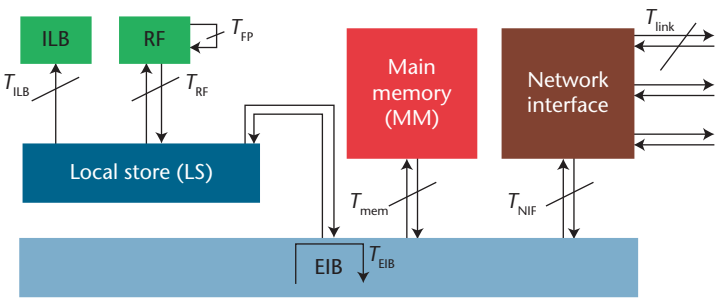

Figure 1. Dataflow paths for a single synergistic processor element and corresponding execution times (T_i). The model assumes a communication network with a 3D torus topology in which 12 links (six inbound and six outbound) simultaneously transfer data.

because the required inter-node bandwidth increases linearly with p (we provide figures on bandwidth requirements later).

Conceptually, this case favors an implementation in which the most powerful processor available provides the total desired processing power and thus reduces processor count and node-to-node bandwidth requirements. IBM's Cell/B.E. (www.ibm.com/developerworks/power/cell) is therefore an obvious choice.

The PowerXCell 8i Processor's Performance

The Cell/B.E. processor contains one PowerPC processor element and eight synergistic processor elements.[5] Each SPE runs a single thread and has its own 256 Kbytes on-chip local store (LS) memory, which is accessible by direct memory access or local load/store operations to and from 128 general-purpose 128-bit registers. An SPE can execute two instructions per cycle, performing up to eight single-precision floating-point operations. Thus, the total single-precision peak performance of all eight SPEs on a Cell/B.E. is 204.8 Gflops at 3.2 GHz.

The PowerXCell 8i is an enhanced version of the Cell/B.E., with the same single-precision performance and a peak double-precision performance of 102.4 Gflops with IEEE-compliant rounding. It has an on-chip memory controller supporting a memory bandwidth of 25.6 Gbytes/s and a configurable Rambus FlexIO I/O interface that supports coherent and noncoherent protocols with a total bidirectional bandwidth of 25.6 Gbytes/s. Internally, all processor units are connected to the coherent element interconnect bus (EIB) by direct memory access (DMA) controllers.

Although it was developed for the PlayStation 3, the Cell/B.E. is obviously attractive for scientific applications as well.[6–8] This is even truer for the PowerXCell 8i. As we reported at the Lattice 2007 Conference,[8] we investigated this processor's performance extending a recently proposed theoretical model.[9] We now summarize our findings.

Performance Model

We consider two classes of hardware devices:

- storage devices, such as registers or LSs, that store data and instructions and are characterized by their storage size and
- processing devices that act on data or transfer data/instructions from one storage device to another. Such devices can be floating-point (FP) units, DMA controllers, or buses and are characterized by their bandwidths β_i and start-up latencies λ_i.

We can divide an algorithm implemented on a specific architecture into microtasks performed by our model's processing devices. The execution time T_i of each task i is estimated by a linear ansatz,

$$T_i \simeq I_i / \beta_i + O(\lambda_i), \tag{2}$$

where I_i is the amount of processed data. In the following, we assume that all tasks can be run concurrently at maximal throughput and that suitable scheduling can hide all dependencies and latencies. Given this, the total execution time is

$$T_{exe} \simeq \max_i T_i . \tag{3}$$

If T_{peak} is the minimal achievable compute time for an application's FP operations in an "ideal implementation," we define the FP efficiency ε_{FP} as $\varepsilon_{FP} = T_{peak}/T_{exe}$.

Figure 1 shows the dataflow paths and associated execution times T_i that enter our analysis:

- floating-point operations, T_{FP};
- load/store operations between register file (RF) and LS, T_{RF};
- off-chip memory access, T_{mem};
- internal communications between SPEs on the same processor, T_{int};
- external communications between different processors, $T_{ext} = \max(T_{NIF}, T_{link})$; and
- transfers via the EIB (memory access, internal, and external communications), T_{EIB}.

We consider a communication network with a 3D torus topology in which the six inbound and six outbound links can simultaneously transfer data, each using a link bandwidth of 1 Gbyte/s. We also assume a balanced network interface providing a bandwidth of 6 Gbytes/s simultaneously in each direction between the Cell/B.E. and the network. (Current FPGA capabilities strongly constrain both the torus network's dimensionality and the bandwidth—specifically, the number of high-speed serial transceivers and the total pin count.) We take all other hardware parameters β_i from the Cell/B.E. manuals (www.ibm.com/developerworks/power/cell). We now describe the results of our performance analysis for different optimizations of the main LQCD kernel.

Lattice QCD Kernel

Computing Equation 1 on a single lattice site amounts to 1,320 (double-precision) FP instructions (not counting sign flips and complex conjugation) and thus yields a T_{peak} of 330 cycles per site. However, executing Equation 1 requires at least 840 multiply-add operations and T_{FP} = 420 cycles per lattice site. Thus, any implementation of Equation 1 on an SPE can't perform better than 78 percent of peak.

The lattice data layout greatly influences the cost of load/store operation for the operands of Equation 1 ($9 \times 12 + 8 \times 9$ complex numbers), as well as the time spent on remote communications. We assign to each Cell/B.E. a local lattice with $V_{Cell} = L_1 \times L_2 \times L_3 \times L_4$ sites and arrange the eight SPEs logically as $s_1 \times s_2 \times s_3 \times s_4 = 8$. A single SPE thus holds a subvolume of $V_{SPE} = (L_1/s_1) \times (L_2/s_2) \times (L_3/s_3) \times (L_4/s_4) = V_{Cell}/8$ sites. On average, each SPE has A_{int} neighboring sites on other SPEs within a Cell/B.E. and A_{ext} neighboring sites outside a Cell/B.E. We investigated two data layout strategies: In the first, all data are stored in the SPEs' on-chip LS; in the second, the data are stored in off-chip main memory (MM).

Data in on-chip memory (LS). In this case, we require that all data for a repeated compute task are held in the SPEs' LS. The LS must also hold a minimal program kernel, the runtime environment, and intermediate results. Therefore, the storage requirements strongly constrain the local lattice volumes V_{SPE} and V_{Cell}.

A spinor field ψ_x needs 24 real words—or 192 bytes in DP—per site, while a gauge field $U_{x,\mu}$ needs 18 words (144 bytes) per link. If we assume that a solver requires storage of roughly 400 words/site (for eight spinors and 12 gauge links, for example), a single SPE's subvolume is restricted to about V_{SPE} = 79 sites. (We assume here a minimal code size of 4 Kbytes for the Dirac kernel; a more realistic assumption of 32 Kbytes for the solver code and runtime environment decreases our estimate to V_{SPE} = 70.) In a 3D network, the fourth lattice dimension must be distributed locally within the same Cell/B.E. across the SPEs (logically arranged as a $1^3 \times 8$ grid). L_4 is then a global lattice extension and could be as large as L_4 = 64. This yields a very asymmetric local lattice, with $V_{Cell} = 2^3 \times 64$ and $V_{SPE} = 2^3 \times 8$. (When distributed over 4,096 nodes, this gives a global lattice size of $32^3 \times 64$.)

Data in off-chip memory (MM). When we store all data in the MM, there are no a-priori restrictions on V_{Cell}. However, we want to avoid redundant loads of Equation 1's operands from MM into LS when sweeping through the lattice. To accomplish this and also permit concurrent computation and data transfers (to/from MM or remote SPEs), we consider a multiple buffering scheme.

Multiple buffering schemes alternate several buffers to either process or load/store data. This permits concurrent computation and data transfer at the price of additional storage (here, in the LS). One way to implement such a scheme is to compute Equation 1's hopping term on a 3D slice of the local lattice and then move the slice along the fourth direction. Each SPE processes all sites along the fourth direction, and the SPEs are logically arranged as a $2^3 \times 1$ grid both to minimize internal SPE communications and to balance external ones. To have all of Equation 1's operands available in the LS, we must be able to keep in the LS the U- and ψ-fields associated with all sites of *three* 3D slices at the same time. This optimization requirement again constrains the local lattice size, now to $V_{Cell} \approx 800 \times L_4$ sites.

Table 1 shows the predicted microtask execution times for the two data layouts and reasonable local lattice size choices. In the LS case, the theoretical efficiency of roughly 27 percent is limited by the communication bandwidth ($T_{exe} \approx T_{ext}$). This is also the limiting factor for the MM case's smallest local lattice; for larger local lattices, the memory bandwidth is the limiting factor ($T_{exe} \approx T_{mem}$).

We have applied our performance model to various linear algebra kernels and verified it by hardware benchmarks. It's not strictly necessary to benchmark the full implementation of a representative QCD kernel such as Equation 1 because, in all relevant cases, T_{FP} is far from

Table 1. Theoretical time estimates T_i (in units of 1,000 clock cycles) for some microtasks needed to compute Equation 1.*

Data in on-chip local store		Data in off-chip main memory			
V_{Cell}	$2^3 \times 64$	$L_1 \times L_2 \times L_3$	8^3	4^3	2^3
A_{int}	16	A_{int} / L_4	48	12	3
A_{ext}	192	A_{ext} / L_4	48	12	3
T_{peak}	21	T_{peak} / L_4	21	2.6	0.33
T_{FP}	27	T_{FP} / L_4	27	3.4	0.42
T_{RF}	12	T_{RF} / L_4	12	1.5	0.19
T_{mem}	—	T_{mem} / L_4	**61**	**7.7**	0.96
T_{int}	2	T_{int} / L_4	5	1.2	0.29
T_{ext}	**79**	T_{ext} / L_4	20	4.9	**1.23**
T_{EIB}	20	T_{EIB} / L_4	40	6.1	1.06
ε_{FP}	27%	ε_{FP}	34%	34%	27%

*Boldface indicates performance bottlenecks.

being the limiting factor. Instead, we've performed hardware benchmarks with the same memory access pattern as in Equation 1, using the above-mentioned multiple buffering scheme for the MM case. We found that execution times were at most 20 percent higher than the theoretical predictions for T_{mem}. (The freely available full-system simulator is not useful in this respect because it doesn't model memory transfers accurately.) Other researchers have reported on benchmarks that use a lattice layout similar to the LS case but that keep only the gauge fields in LS, while spinor fields are accessed in MM.[10] They found efficiencies above $\varepsilon_{FP} = 20$ percent.

Local Store DMA Transfers

Because DMA transfer speeds determine T_{mem}, T_{int}, and T_{ext}, it's crucial that we optimize them to exploit the Cell/B.E. performance. Our analysis of detailed micro-benchmarks for LS-to-LS transfers shows that Equation 2's linear model doesn't accurately describe execution time for DMA operations with arbitrary size I and arbitrary address alignment. We therefore refined our model to account for data transfer fragmentation and the buffers' source and destination addresses (A_s and A_d, respectively):

$$T_{\mathrm{DMA}}(I, A_s, A_d) = \lambda^0 + \lambda^a \cdot N_a(I, A_s, A_d) + N_b(I, A_s) \cdot \frac{128 \text{ bytes}}{\beta}. \quad (4)$$

Our hardware benchmarks, fitted to Equation 4, indicate that each LS-to-LS DMA transfer has a (zero-size transfer) latency of $\lambda^0 \approx 200$ cycles. The DMA controllers fragment all transfers into N_b 128-byte blocks aligned at LS lines (and corresponding to single EIB transactions). When $\delta A = A_s - A_d$ is a multiple of 128 bytes, the source LS lines can be directly mapped onto the destination lines. Then, we have $N_a = 0$, and the effective bandwidth $\beta_{eff} = I/(T_{DMA} - \lambda^0)$ is the approximate peak value. Otherwise, if the alignments don't match (δA isn't a multiple of 128), we encounter an additional latency of $\lambda^a \approx 16$ cycles for each transferred 128-byte block, reducing β_{eff} by roughly a factor of two.

Figure 2 shows how clearly these effects are observed in our benchmarks and how accurately Equation 4 describes them.

Discussion

Our performance model and hardware benchmarks identified the PowerXCell 8i processor as a promising option for LQCD. We expect that a sustained performance above 20 percent is possible on large machines. Parallel systems with O(2000) PowerXCell 8i processors add up to approximately 200 Tflops (DP peak), which corresponds to roughly 50 Tflops sustained for typical LQCD applications. As we discussed earlier, a simple nearest-neighbor d-dimensional interconnection among these processors is all we need to support our algorithms' data exchange patterns. This simple structure allows for a fast and cost-effective design and construction of our forthcoming QCD-oriented number cruncher.

The QPACE Project

QPACE is a collaborative development effort among several academic institutions and the IBM development lab in Böblingen, Germany, to design, implement, and deploy a next genera-

tion of massively parallel and scalable computer architectures optimized for LQCD. Our project's primary goal is to make a vast amount of computing power available for LQCD research. On the technical side, our goals are to

- use commodity processors, tightly interconnected by a custom network;
- leverage the potential of FPGAs for network implementation; and
- aim for an unprecedentedly small ratio of power consumption versus floating-point performance.

In the following, we describe the key elements of the QPACE architecture.

Node Card

The main building blocks of QPACE are the node cards. These processing nodes, which run independently of each other, include two main components:

- a PowerXCell 8i processor, which provides the computing power; and
- a network processor (NWP), which implements a dedicated interface to connect the processor to a 3D high-speed torus network used for communications between the nodes and to an Ethernet network for I/O.

We keep additional logic—to boot and control the machine—to the bare minimum. Furthermore, the node card contains 4 Gbytes of private memory, which is sufficient for all the data structures (and auxiliary variables) of present-day local lattice sizes.

We implemented the NWP using an FPGA (Xilinx Virtex-5 LX110T), which lets us develop and test logic reasonably fast and keep development costs low. However, the devices themselves tend to be expensive (in our case, Xilinx's support of QPACE makes this issue less critical). The NWP's main task is to route data between the Cell/B.E. processor, the torus network links, and the Ethernet I/O interface. A torus network link's bandwidth is approximately 1 Gbyte/s each for transmit and receive. In balance with the overall bandwidth of the six torus links attached to each NWP, the interface between the NWP and the Cell/B.E. processor has a bandwidth of 6 Gbytes/s. (Because existing southbridges don't provide this bandwidth to the Cell/B.E., a commodity network solution is ruled out.)

Unlike other Cell/B.E.-based parallel machines,

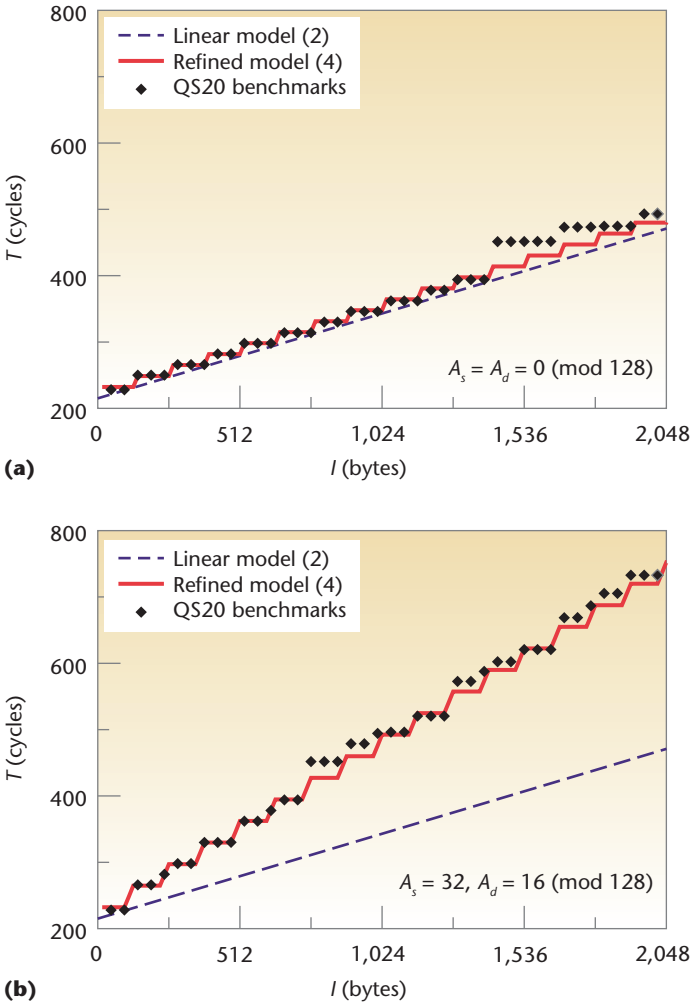

Figure 2. LS-to-LS DMA transfers on an IBM QS20 system. Execution time is measured as a function of the transfer size with (a) aligned and (b) misaligned source and destination addresses. The dashed and solid lines correspond to the theoretical predictions of Equations 2 and 4, respectively.

in QPACE, node-to-node communications proceed directly from the LS of a processor's SPE to the LS of a nearest-neighbor processor's SPE. For communication, we don't move the data through the MM, and thus we reduce the performance-critical data traffic through the MM interface. Instead, we route them, via the EIB, from an LS directly to the Cell/B.E. processor's I/O interface. The PowerPC processor element isn't needed to control such communications. We expect the LS-to-LS copy operations latency to be on the order of 1 μs.

Communication Network

To start a communication, the sending SPE must

initiate a DMA data transfer from its LS to a buffer attached to any link module. Once the data arrives in the buffer, the NWP will move the data across the network without the processor's intervention. On the other end, the receiving device must post a receive request to trigger the DMA data transfer from the NWP's receive buffer to the final destination.

For the torus network links, we implement a lightweight protocol, which handles automatic data resend in case of CRC errors. The physical layer uses well-tested and cost-efficient commercial hardware components, which let us move the most timing-critical logic out of the FPGA. Specifically, we use the 10 Gbits/s XAUI transceiver PMC Sierra PM8358, which provides redundant link interfaces that we can use to select among several torus network topologies (as we describe in more detail later).

Implementing the link to the Cell/B.E. processor's I/O interface (a Rambus FlexIO bus) is much more challenging. At this point, we've connected an NWP prototype and a Cell/B.E. processor at a speed of 3 Gbytes/s per link and direction by using special features available in the Xilinx Virtex-5 FPGAs RocketIO transceivers. (So far, we've tested only a single 8-bit link; we'll use two links in the final design.)

Overall Machine Structure

We attach the node cards to a backplane through which we route all network signals. One backplane hosts 32 node cards and two root cards, each of which controls 16 node cards via an Ethernet-accessible microprocessor. One QPACE cabinet can accommodate eight backplanes, or 256 node cards. Each cabinet therefore has a peak double-precision performance of roughly 25 Tflops.

On the backplane, subsets of nodes are interconnected in a one-dimensional ring topology. If we select the primary or redundant XAUI links, we can select a ring size of 2, 4, or 8 nodes. Likewise, we can configure the number of nodes in the second dimension, in which the nodes are connected by a combination of backplane connections and cables. In the third dimension, we use cables. It's also possible to operate a large QPACE system of N cabinets as a single partition with $2 \cdot N \times 16 \times 8$ nodes.

I/O Network

We implement input and output operations using a Gigabit Ethernet tree network. Each node card is a tree endpoint connected to one of six cabinet-level switches, each of which has a variable number of Gigabit Ethernet uplinks, depending on bandwidth requirements. When we deploy QPACE, we expect typical lattice sizes of $48^3 \times 96$, which require a gauge field configuration of roughly 6 Gbytes. The available I/O bandwidth should thus allow us to read or write the database in $O(10)$ seconds.

Power and Cooling

We expect a single node card's power consumption to be less than 150 watts. A single QPACE cabinet will therefore consume roughly 35 kilowatts, which translates into a power efficiency of approximately 1.5 watts/Gflops. We're developing a liquid cooling system to reach the planned packaging density.

QPACE Software

To operate the QPACE nodes, we'll use Linux running on the PowerPC processor element. As on most other processor platforms, we can't start the operating system directly after system start—instead, we first use a host firmware to initialize the hardware. The QPACE firmware will be based on Slimline Open Firmware (www-128.ibm.com/developerworks/power/pa-slof); system startup is controlled by the root card microprocessor.

Efficiently implementing applications on the Cell/B.E. processor is more difficult compared to standard processors. To optimize large application codes on a Cell/B.E. processor, programmers face several challenges—they must, for example, carefully choose the data layout to maximize memory-interface utilization. Optimizing on-chip memory use to minimize external memory accesses is mandatory. The program's overall performance also depends on how much code can be parallelized on-chip.

We apply two strategies to relieve the programmers' porting burdens. First, in typical LQCD applications, almost all cycles are spent in a few kernel routines (such as Equation 1). We'll therefore provide highly optimized assembly implementations for such kernels, possibly also making use of an assembly generator. Second, we'll leverage the work of the USQCD (US Quantum Chromodynamics) collaboration (www.usqcd.org/software.html), which has pioneered efforts to define and implement software layers to hide hardware details. Such a framework will let programmers build LQCD applications in a portable way on top of these software layers.

With QPACE, our goal is to implement the QCD message-passing (QMP) API, as well as (at least parts of) the QCD data-parallel (QDP) API,

which includes operations on distributed data objects. QMP comprises all communication operations required for LQCD applications. It relies on the fact that these applications typically have regular and repetitive communication patterns, with data being sent between adjacent nodes in a torus grid (and therefore we don't need an API as general as MPI). We'll implement QMP on top of a few low-level communication primitives that might, for example, trigger data transmission via one particular link, initiate the receive operation, and allow the program to wait for completion of communications.

QPACE is an innovative approach to use a commodity multicore processor together with a custom network for building a scalable massively parallel machine for LQCD simulations. Directly connecting a custom network to a high-end commercial processor's I/O interface is a significant technological challenge. For QPACE, we were able to achieve this using an FPGA, but it might be highly nontrivial for other (multicore) processors. It also remains to be seen whether FPGAs will be able to cope with increasing bandwidth requirements in future developments.

The QPACE project's ambitious goal is to complete hardware development by the end of 2008 and to begin manufacturing and deploying larger systems beginning in 2009. We expect the machines to be fully available for lattice QCD research by mid-2009.

Acknowledgments

QPACE is funded by the Deutsche Forschungsgemeinschaft (DFG) through the SFB/TR-55 framework and by IBM. We gratefully acknowledge important contributions to QPACE by Eurotech (Italy) and Knürr (Germany).

References

1. K.G. Wilson, "Confinement of Quarks," *Physical Rev. D*, vol. 10, no. 8, 1974, pp. 2445–2459.
2. P.A. Boyle et al., "Overview of the QCDSP and QCDOC Computers," *IBM J. Research & Development*, vol. 49, nos. 2-3, 2005, pp. 351–366.
3. F. Belletti et al., "Computing for LQCD: apeNEXT," *Computing in Science & Eng.*, vol. 8, no. 1, 2006, pp. 18–29.
4. A. Gara et al., "Overview of the Blue Gene/L System Architecture," *IBM J. Research & Development*, vol. 49, nos. 2-3, 2005, pp. 195–212.
5. H.P. Hofstee et al., "Cell Broadband Engine Technology and Systems," *IBM J. Research & Development*, vol. 51, no. 5, 2007, pp. 501–502.
6. S. Williams et al., "The Potential of the Cell Processor for Scientific Computing," *Proc. 3rd Conf. Computing Frontiers*, ACM Press, 2006, pp. 9–20.
7. A. Nakamura, "Development of QCD-Code on a Cell Machine," *Proc. Int'l Symp. Lattice Field Theory* (LAT 07), Proc. of Science, 2007; http://pos.sissa.it//archive/conferences/042/040/LATTICE%202007_040.pdf.
8. F. Belletti et al., "QCD on the Cell Broadband Engine," *Proc. Int'l Symp. Lattice Field Theory* (LAT 07), Proc. of Science, 2007; http://pos.sissa.it//archive/conferences/042/039/LATTICE%202007_039.pdf.
9. G. Bilardi et al., "The Potential of On-Chip Multiprocessing for QCD Machines," *Proc. Int'l Conf. High-Performance Computing*, LNCS 3769, Springer Verlag, 2005, pp. 386–397.
10. J. Spray, J. Hill, and A. Trew, "Performance of a Lattice Quantum Chromodynamics Kernel on the Cell Processor," 2008; http://arxiv.org/abs/0804.3654.

Gottfried Goldrian *is a Distinguished Engineer in the IBM Lab in Böblingen, Germany. Contact him at goldrian@ibm.de.*

Thomas Huth *is a developer at the IBM Lab in Böblingen. Contact him at thuth@de.ibm.com.*

Benjamin Krill *is a research engineer at the IBM Lab in Böblingen. Contact him at krill@codiert.org.*

Jack Lauritsen *is a developer at the IBM Lab in Böblingen. Contact him at laurits@de.ibm.com.*

Heiko Schick *is a developer at the IBM Lab in Böblingen. Contact him at schickhj@de.ibm.com.*

Ibrahim Ouda *is a senior engineer at the IBM Lab in Rochester, Minnesota. Contact him at ouda@us.ibm.com.*

Simon Heybrock *is an undergraduate student of physics at the University of Regensburg, Germany. Contact him at simon.heybrock@physik.uni-regensburg.de.*

Dieter Hierl *is a research associate at the University of Regensburg, Germany. Contact him at dieter.hierl@physik.uni-regensburg.de.*

Thilo Maurer *is a PhD student in physics at the University of Regensburg, Germany. Contact him at thilo.maurer@physik.uni-regensburg.de.*

Nils Meyer *is a PhD student in physics at the University of Regensburg, Germany. Contact him at nils.meyer@physik.uni-regensburg.de.*

Andreas Schäfer *is a professor of physics at the University of Regensburg, Germany. Contact him at andreas.schaefer@physik.uni-regensburg.de.*

Stefan Solbrig *is a research associate at the Univer-*

sity of Regensburg, Germany. Contact him at stefan.solbrig@physik.uni-regensburg.de.

***Thomas Streuer** is a postdoctoral researcher at the University of Regensburg, Germany. Contact him at thomas.streuer@desy.de.*

***Tilo Wettig** is a professor of physics at the University of Regensburg, Germany. Contact him at tilo.wettig@physik.uni-regensburg.de.*

***Dirk Pleiter** is a research scientist at Deutsches Elektronen Synchrotron (DESY), Zeuthen, Germany. Contact him at dirk.pleiter@desy.de.*

***Karl-Heinz Sulanke** is an electronics engineer at Deutsches Elektronen Synchrotron (DESY), Zeuthen, Germany. Contact him at karl-heinz.sulanke@desy.de.*

***Frank Winter** is a PhD student in physics at Deutsches Elektronen Synchrotron (DESY), Zeuthen, Germany. Contact him at frank.winter@desy.de.*

***Hubert Simma** is a research scientist at Deutsches Elektronen Synchrotron (DESY), Zeuthen, Germany, and a guest lecturer at the University of Milan. Contact him at hubert.simma@desy.de.*

***Sebastiano Fabio Schifano** is a research associate in computer science at the University of Ferrara, Italy. Contact him at schifano@fe.infn.it.*

***Raffaele Tripiccione** is a professor of physics at the University of Ferrara, Italy. Contact him at tripiccione@fe.infn.it.*

***Andrea Nobile** is a PhD student in physics at the European Center for Theoretical Studies, Trento, and the University of Milano-Bicocca, Italy. Contact him at andrea.nobile@mib.infn.it.*

***Matthias Drochner** is a research scientist at the Research Center Jülich, Germany. Contact him at M.Drochner@fz-juelich.de.*

***Thomas Lippert** is the director of the Institute for Advanced Simulation at the Research Centre Jülich, head of the Jülich Supercomputing Centre, and a professor of physics at the University of Wuppertal, Germany. Contact him at th.lippert@fz-juelich.de.*

***Zoltan Fodor** is a professor of physics at the University of Wuppertal, Germany. Contact him at fodor@bodri.elte.hu.*