PROCEEDINGS OF SCIENCE

# The BlueGene/Q Supercomputer

**P A Boyle***
*University of Edinburgh*
*E-mail:* paboyle@ph.ed.ac.uk

I introduce the BlueGene/Q supercomputer design, with an emphasis on features that enhance QCD performance, discuss the architectures performance for optimised QCD codes and compare these characteristics to other leading architectures.

*The 30 International Symposium on Lattice Field Theory - Lattice 2012,*
*June 24-29, 2012*
*Cairns, Australia*

*Speaker.

 

## 1. QCD computer design challenge

The floating point operation count in the Monte Carlo evaluation of the QCD path integral is dominated by the solution of discretised Euclidean Dirac equation. The solution of this equation is a classic sparse matrix inversion problem, for which one uses one of a number of iterative Krylov methods. The naive dimension of the (sparse) matrix is of $O([10^9]^2)$, and hundreds of thousands of inversions must be performed in a serially dependent chain to importance sample the QCD path integral. The precision with which we can solve QCD numerically is computationally limited, and the development of faster computers is of great interest to the field.

Many processors are in principal faster than one, providing we can arrange for them to coordinate work effectively. QCD is easily parallelised with a geometrical decomposition spreading $L^4$ space time points across multiple $N^4$ processing nodes, each containing a subvolume of size $l^4$. For simplicity we consider only symmetrical decomposition problems. For many of lattice actions, such as Wilson-clover, overlap and domain wall fermions or twisted mass fermions, the Krylov inversion is dominated by the repeated application of the Wilson operator:

$$D_W(M) = M + 4 - \frac{1}{2} D_{\rm hop},$$

where

$$D_{\rm hop} = (1-\gamma_\mu) U_\mu(x) \delta_{x+\mu,y} + (1+\gamma_\mu) U^\dagger_\mu(y) \delta_{x-\mu,y}.$$

| | |
|---|---|
| $t_{fpu}$ | performing the arithmetic for all sites within a node [$l^4 \times 1320$ flops] |
| $t_{comm}$ | communicating the surface [$l^3 \times 12 \times 16$ words] |
| $t_{mem}$ | loading the fermion field from memory to cache [$l^4 \times 2 \times 24$ words] |
| $t_{cache}$ | using cache resident fermion field $2N_d$ times [$l^4\times$] |

**Table 1:** Components of the application of the Wilson operator include both computation and data motion. These different tasks normally map directly to distinct subsystems in of a computer node. A good implementation will ensure these take place concurrently, and we naturally only consider good implementations. Here we discount the memory overhead of the gauge field because in the computationally demanding 5d overlap and domain wall fermion approaches the gauge field can be reapplied $N_5$ times (indeed may be reused in the highest and fastest level of cache). We assume that the loop order will maximise cache reuse, and therefore count only *compulsory* memory traffic: assuming the inverter working set does not fit in cache, there remains $2 \times N_d$ reuse in sparse matrix since each fermion field enters the stencil of all its neighbours. We note that including even-odd preconditioning does not change this analysis if we use $2L \times L^3$

In Table 1 we model the application of the Wilson operator as consisting of a number of tasks, which include both computation and data motion. These can in principle be overlapped, the longest of these will determine the time $t_{Wilson}$, $t_{Wilson} = \mathrm{Max}\{t_{comm}, t_{fpu}, t_{memory}, t_{cache}\}$. In a balanced design the network, cache and memory bandwidths ($B_N$,$B_M$,$B_C$) should be such that $t_{cache}, t_{comm}, t_{memory} \approx t_{fpu}$. We see that scalability limited when $t_{comm}$ is largest, and this implies a minimum sensible local volume $L_{min}$. For example, if the computation is otherwise cache bandwidth limited we obtain the bound:

$$t_{comm} \leq t_{cache} \qquad \Longleftrightarrow \qquad \frac{192 L^3}{B_N} \leq \frac{216 L^4}{B_C} \qquad \Rightarrow L_{min} \sim \frac{B_C}{B_N}$$

Thus, the Wilson operator scalability ends up being determined by the ratio of network bandwidth to cache & memory bandwidth [1]. This $L_{min}$ in turn sets a limit on the *maximum* performance on a given total problem size $L$:

$$\text{Performance} \sim \frac{1320 \times N^4}{t_{comm}} = \frac{1320 \times N^4 B_N^4}{192 \times B_C^3},$$

and we see with a simple calculation that the maximum performance and scalability are proportional to the *fourth power* of network bandwidth.

It is therefore important for *scalable* computing requirements, such as the generation of gauge configurations, that the network bandwidth be of similar order to the local memory bandwidth. One may view the entire message passing supercomputer as a processing system; the network is simply one (admittedly cumbersome and explicitly programmed) part of the global memory system. In a scalable processing system the cost of a local data reference and a remote data reference should be similar, and this requires a 1:1 ratio of bandwidths. This point is made clear if we consider $64^4$ on an nominal 100Gflop/s node, with various bandwidth ratios $B_C/B_N$ in Table 2. One rapidly concludes that it is important to integrate network controller in the memory system so that $B_N \sim O(B_C)$.

| $\frac{B_C}{B_N}$ | Number of nodes | Max System Performance |
|---|---|---|
| 32 | 16 | 1.6Tflop/s |
| 16 | 256 | 25.6 Tflop/s |
| 8 | 4096 | 409.6 Tflop/s |
| 4 | 64k | 6 Pflop/s |
| 2 | 1M | 96 Pflop/s |

**Table 2:** Considering a nominal 100Gflop/s node, may ask what ultimate scalability one can achieve as a function of the ratio of cache to network bandwidth.

## 2. IBM BlueGene/Q Architecture

The IBM BlueGene/Q architecture[1, 2, 3], figure 1, is based on a special purpose 16+1+1 core PowerPC 64bit architecture computer chip that integrates an MPI network interface, 32 MB L2 cache, and DDR3 memory controller in a single chip. This chip proves 204.8GFlop/s peak floating point performance, and is combined with 16GB DRAM compute nodes.

| GFlop/s | L1 GB/s | L2 GB/s | DDR GB/s | Torus GB/s |
|---|---|---|---|---|
| 204.8 | 820GB/s | 563 | 42.7 | 40 |

**Table 3:** The BlueGene/Q compute chip integrates huge cache, huge MPI bandwidth ($\equiv$ O(10) Infiniband cards) within modest area and power budget, leading to a scalable and energy efficient architecture.

Thirty two compute nodes are integrated in a nodeboard, and 16 of these nodeboards are coupled through midplanes (from front and back). The BlueGene/Q network uses a combination

[1] Either cache or main memory bandwidth usually limits througput the absence of communication overhead

of optical and copper interconnect. Copper is used within a midplane (512 node half rack), The density is important as it allows a great fraction of communications links to remain relatively inexpensive copper. Optical transceivers and cables interconnect links crossing between midplanes. The interconnect is a high bandwidth 5d toroidal network providing a system that is highly scalable and power efficient. The rack has a high power density and water cooling is required[2].

The compute nodes run a lean custom compute node kernel. The standard GNU libraries and compilers are available, for application code, but the custom kernel ensures operating scheduling and virtual memory handling cannot cause the avoidable application slow down. A subset of nodes have links to air cooled I/O nodes, which though essentially identical to compute nodes, run Linux. Any node can route system call requests to an I/O nodes, thus presenting a Linux like environment and these in turn forward disk accesses to an external disk system across an infiniband network.

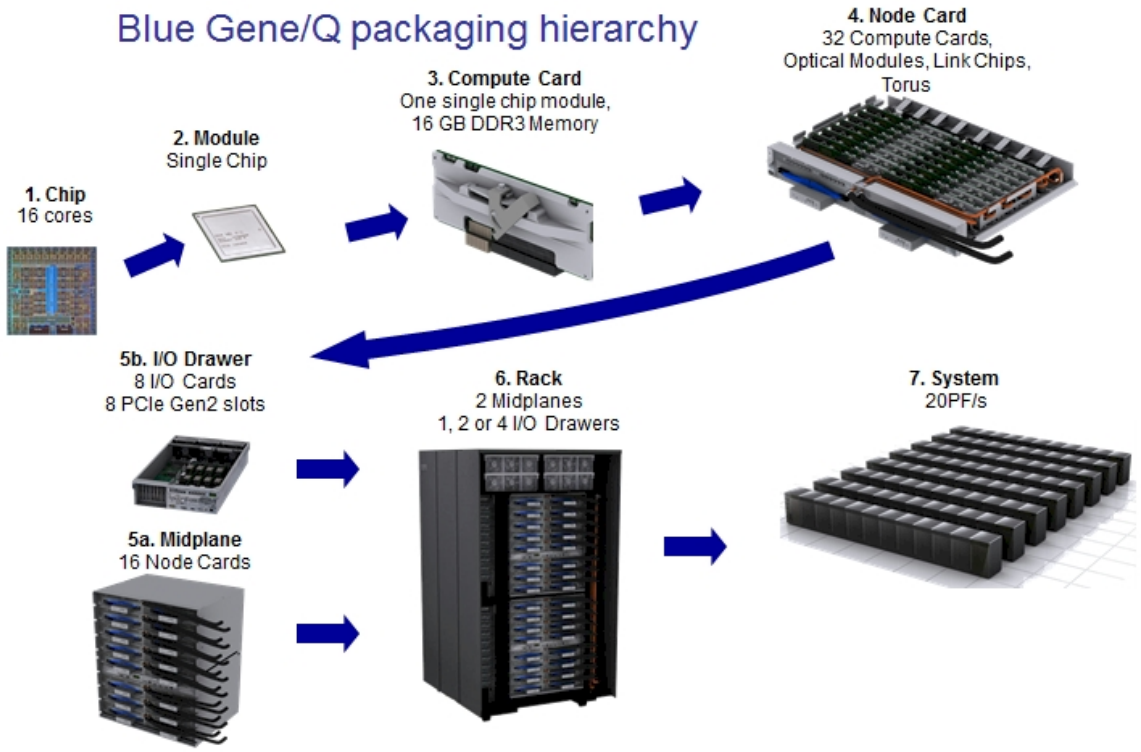


**Figure 1:** BlueGene/Q system design

Scientists from Edinburgh University and Columbia University collaborated with IBM throughout the development, and designed the performance differentiating L1p prefetch engine component. This gave a unique opportunity to tune the architecture for its most important scientific application: the application of the Wilson Dirac operator.

### 2.1 Compute cores

The die, Figure 3 covers 360mm$^2$ in a 45nm process and the chip consumes only 55W at 1.6GHz clock speed. 16 cores are used for application processes, while a 17th is dedicated to

[2] 100KW per 1024 node and 209.7 TFlop/s rack corresponds roughly to the maximum power output of a turbocharged Audi A4, dischared in a volume the size of a US fridge and filled with delicate electronics.

**Figure 2:** The 1.26Pflop/s Edinburgh BlueGene/Q DiRAC system, comprising six racks.

operating system and interrupt handling functions. An unused 18th core is physically present for yield purposes, and may be swapped by fuse blowing. The cores issue instructions from each processing thread in order, but will concurrently fetch and operate on independent instructions from up to four threads. Most computer code spends much of its time waiting for data from memory. The process of pushing this data to and from memory can be thought of akin to juggling with a single ball. Serial dependencies in a single instruction stream stop progress until a complete traversal to and from memory is complete, just as a juggler cannot make the next throw until he has caught the last. The PowerPC A2 core increases throughput by processing multiple independent threads concurrently, just as a juggler with four balls maintains greater throughput than a juggler with only one ball. By replicating only some aspects of the instruction fetch hardware and register files four times, Figure 4, the design avoids expensive and power hungry out-of-order dependency tracking support, while rather more easily finding independent instructions with which to fill processor pipelines, as instructions from different threads are necessarily independent.

The BlueGene/Q QPX floating point unit includes 4 wide double precision single instruction multiple data (SIMD) instructions. Floating point throughput is substantially increased with only replication of the floating point arithmetic pipelines, widening of floating point register files and data paths. In particular, instruction issue rates, control logic or core counts are not increased and the die area impact for this substantial benefit is limited. For scientific array processing problems, SIMD is usable with some effort. The vector loads and stores for consecutive elements must be contiguous in memory, and effective use may entail changing the layout of arrays to ensure adjacent elements of can always be used together. The QPX allows up to eight floating point operations per clock cycle in a fused multiply add format ($z = ax + y$), but only four floating floating point operations if multiplies and adds are independently processed ($z = x + y$, or $z = x * y$). Due to the patterns of adds and multiplies in the Wilson operator, for example, this constraint sets an upper

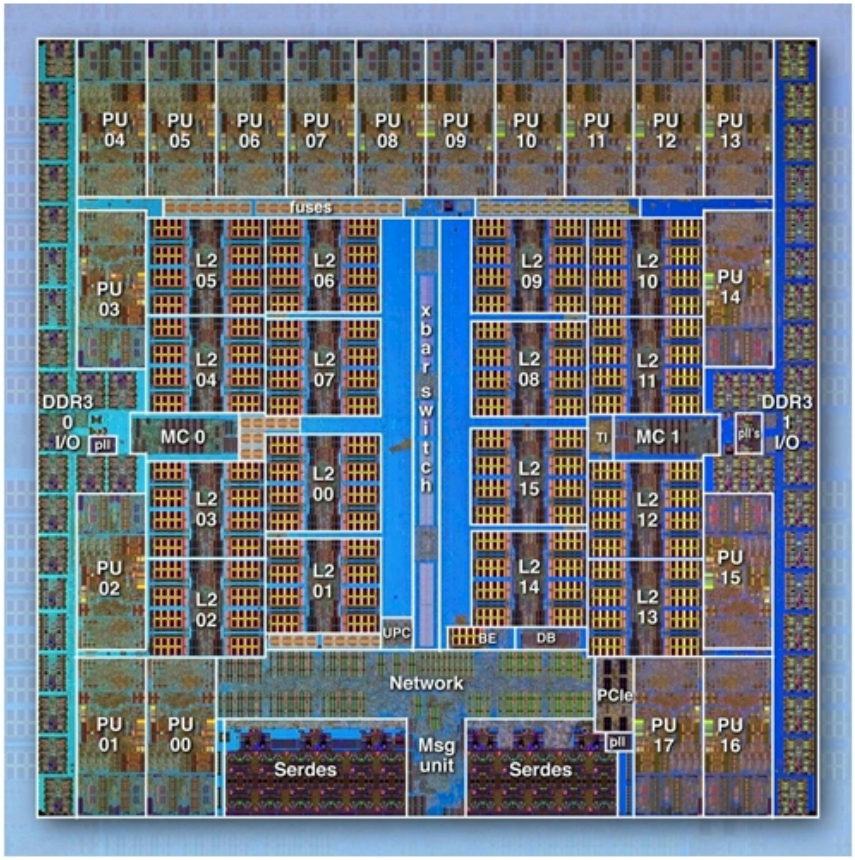


**Figure 3:** The BlueGene/Q compute chip die photo

bound on performance at 78% of peak, even if instructions were issued on every cycle. The floating point unit can operate as twin complex arithmetic pipelines, with efficient support for forming all required real and imaginary products. The 16 compute cores issuing 8 floating point operations per cycle each at 1.6GHz leads to the nominal 204.8 Gflop/s peak performance.

### 2.2 Memory subsystem

With up to 64 threads on each compute node, fine grained threading performance is important. BlueGene/Q provides significant hardware assistance for interthread synchronisation and uses a scalable coherency scheme. The memory system in a many core chip is the key to performance.

**L1 cache** Each core contains an 8 way associative[3] 16KB L1 data cache and 16KB L1 instruction cache, with 64 byte L1 line size. The L1 cache is a write through cache, giving the advantage that parity errors can be recovered from by simply invalidating the line and refetching. The L1 cache has a number of features that enable detailed control by user programmes. A range of cache touch and flush instructions for different levels of cache are supported, and user mode locking of

[3]The associativity indicates the number of possible locations in which a given memory address may be stored. When $N$-way associativity limits are introduced, certain address bits select which group of $N$-cache-locations will store this address. The effective cache capacity can be reduced for certain memory access patterns where these address bits collide, but a faster cycle time for searching the cache is enabled.

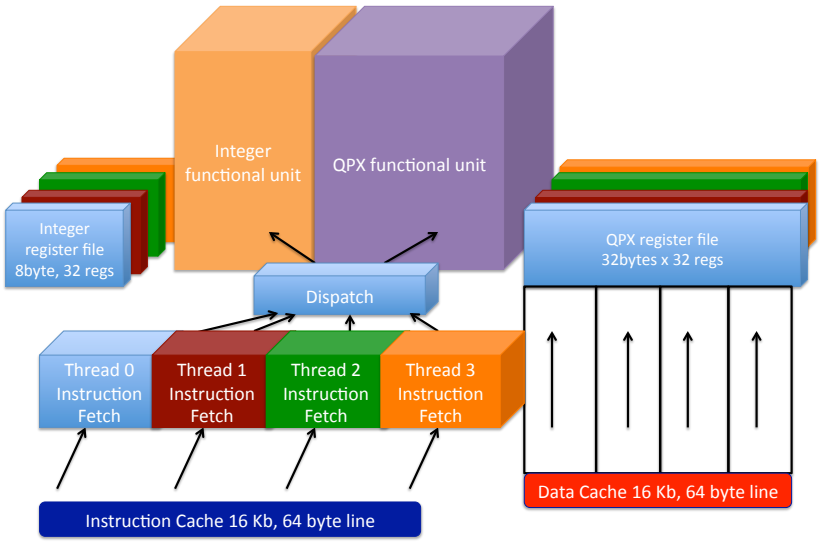


**Figure 4:** The A2 core replicates instruction fetch hardware and register files four times, while the area and power hungry execution units are not replicated. Higher pipeline utilisation is possible with little additional cost. The floating point units contain four multiply-add pipelines which operate on four SIMD vector operands in parallel. Only the floating point arithmetic pipelines, datapaths are widened to support SIMD making this a cost effective way to increase floating point performance.

individual cachelines is supported at both the L1 and L2 level. This enables critical data to be retained, even when streaming large amounts of use once data through the cache.

The A2 core has a load miss queue, allowing up to eight concurrent off-core loads to distinct 64 byte L1 cachelines to be in flight concurrently. [4]

**L1p prefetch engine:** The core interfaces to the memorys system through a 4KB L1p prefetch engine, figure 5, designed by Edinburgh and Columbia Universities. This unit drops the cycle frequency from 1600MHz to 800MHz, and provides 32 prefetch buffers of 128 bytes each, and a large write combining capacity to coalesce consecutive writes into single switch transactions. The L1p prefetch cache is fully associative, and a variable prefetch depth of between 1 and 8 lines used. It is designed to prefetch over a range of sequential access streams, from one thread and one stream, to four threads and four streams. Hardware therefore manages the prefetch depth for between one and sixteen streams. Several modes of operation are supported:

i) Fixed prefetch depth: a known stream count can be optimally prefetched

ii) Adaptive prefetch depth: by allowing bandwdith starved streams to steal depth from least recently used streams the optimal pattern will be learned from the memory reference stream.

[4] A second load to the same cacheline will trigger a pipeline flush. As the SIMD vector size is 32bytes, careful sequencing of loads to distinct cachelines is required to avoid instruction flushes and keep load miss queue filled.

iii) Fixed size demand fetch: Where short sequences of contiguous data are required without streaming, between 1 and 8 lines can be programmed to demand fetched into the L1p in response to a single miss in the L1[5]. This mode increases the effective L1 line size, adding performance where there is spatial locality of access, without performing stream prefetch.

iv) Perfect prefetch: for repetitive, iterative code a sequence of L1 misses can be recorded in the first iteration and replayed through the L1p in subsequent iterations. This can be useful for prefetching non-sequential memory patterns, such as arise in general sparse matrix problems, where there is no way to predict the future memory addresses.

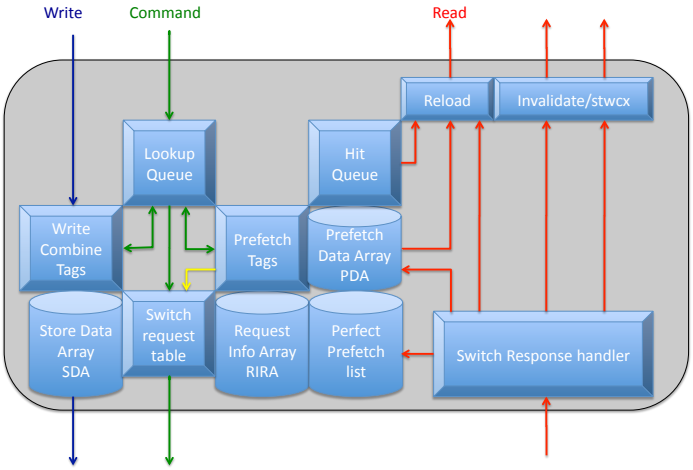


**Figure 5:** The level 1 prefetcher attached to each core monitors memory access patterns. It fills 4KB of prefetch buffers by anticipating the demands of the core, and gathers up adjacent stores from the cores to coalesce them into a single switch transaction. It must remain coherent with all other cores by participating in the invalidation scheme. As it must deal with a wide range (1-16) of concurrent streams with only 32 prefetch buffers, the stream prefetch depth varies adaptively from 128bytes to 1024bytes in response to the observed pattern of accesses. A perfect prefetch mode can record arbitrary access patterns and is useful for iterative code.

**Crossbar switch:** A crossbar switch operates at 800MHz and links the computational cores with sixteen independent L2 slices. The write bus from each core is 128 data bits wide, and the read bus 256 bits wide. The peak bandwidth per core is thus 12.8+25.6 GB/s and across 16 computational core this is a peak 204.8+409.6 GB/s for read and write to L2. Reads and writes compete for command issue slots across the switch, so simultaneously saturating both directions is not possible and the delivered bandwidth is somewhat lower than the sum.

**L2 cache:** The 32 MB L2 cache is globally shared among the cores in a node, with the high density enabled by the use of embedded DRAM which uses only one transistor and a deep trench capacitor per bit-cell. While DRAM has access penalties compared to SRAM, the capac-

[5]In plain lattice-English the author has ensured we can programme the size of a spinor into the memory system!

ity enhanced high bandwidth cache is a great asset for scientific code, and also increases energy efficiency since a reference .

The memory system uses a back-invalidate architecture. When a write is received from one core, messages are sent to other cores to invalidate their copy of lines and ensure a consistent view of memory by different threads after synchronisation. This is in many respects superior to the more common modified-exclusive-shared-invalid line state schemes: a single cacheline can be written to concurrently by many threads without the serialisation created by the need for exclusive ownership. This is particularly important in many core compute nodes where fine grained threading is required.

The L2 also provides hardware atomic operations on any memory address. This avoids, for example, costly load-locked/store-conditional synchronisation to access a shared barrier variable, and accelerates synchronisation substantially, figure 6. The overhead of synchronising all 64 threads is reduced from around 14000 cycles using traditional instructions to around 1000 cycles. As the operation is supported for any memory address, the hardware resources for this acceleration are limited only by the memory capacity.

The L2 interfaces to 16GB external DDR3-1333 memory through dual 128bit wide interfaces. Of the 42.6GB/s peak bandwidth, around 27GB/s is achievable with streaming code. Naturally, the system would be an even better design with greater bandwidth, however external pins on a chip are limited, and a similar bandwidth is given to the interconnect network giving a scalable machine. The large cache capacity plays a big role in application performance and optimising for cache reuse is important.

The L2 also contains the first ever hardware implementation of transactional memory. This is a promising technique that promises to replace serialisation of parallel hardware through locking, with a "try-catch-retry" sequence. Where collision is infrequent lock acquisition overhead is eliminated. The utility for general scientific code is not yet clear.

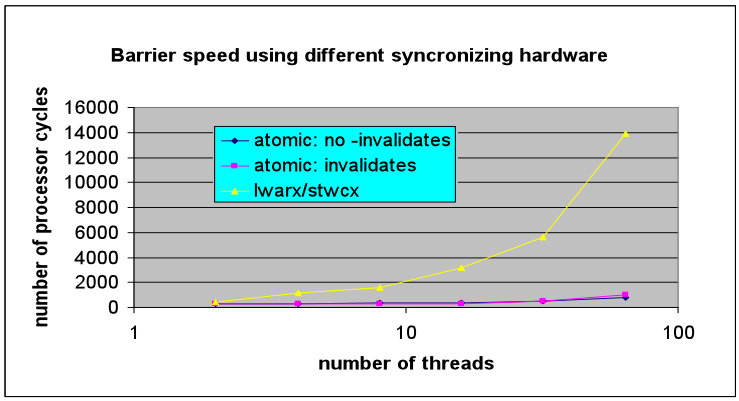


**Figure 6:** With up to 64 threads on each compute node, fine grained threading performance is important. BlueGene/Q provides hardware assistance for interthread synchronisation with hardware atomic operations on any memory address. This avoids, for example, costly load-locked/store-conditional synchronisation to access a shared barrier variable, and accelerates synchronisation substantially.

### 2.3 Network

BlueGene/Q integrates a 5d torus interconnect router and message unit on the compute chip. Each network link has a 2GB/s send + 2GB/s receive peak bandwidth, for an aggregate 40GB/s interconnect bandwidth coming from each compute card. Increasingly multi-gigabit electrical signalling bitrates are converging with same transceivers driving with different protocol technologies. More wires deliver more bandwidth. The large row of black connectors on the BlueGene/Q compute card, figure 1, carry 11 high speed interconnect channels, each comparable to an infiniband link, and give the design excellent network bandwidth and hence scalability.

Accompanying the network is an advanced message unit which provides hardware DMA resources enabling zero copy DMA, direct from user space memory pages, both between nodes and also within the node. From 1 to 64 MPI processes per node are available as a run time option, and OpenMP and Pthreads threading models are supported internally. Thus any combination of pure MPI parallelism also hybrid MPI/threading parallelism is available. The hardware acceleration resources are sufficient to accelerate 64 MPI processes per node if used in this mode. Collectives such as reductions (global summation) and All-to-All operations are supported in hardware and for flexible subsets of a partition.

The message unit has multiple ports on the on chip memory switch, and therefore gains direct access to the high bandwidth L2 cache. This tight integration enables high delivered bandwidth in halo exchange, displayed as a function of memory footprint in figure 7.

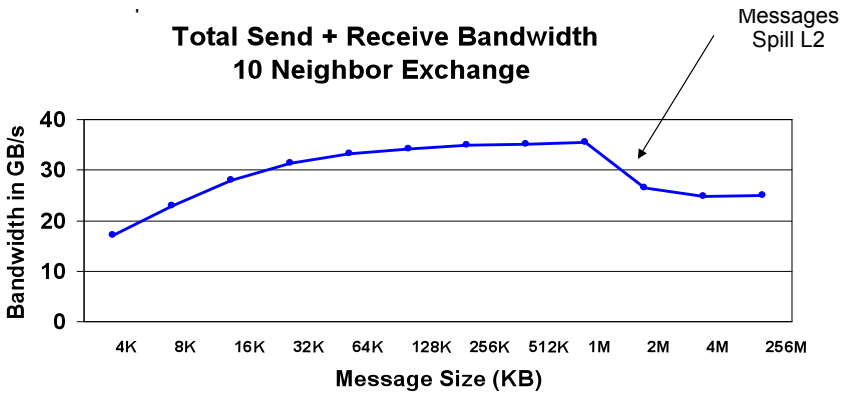


**Figure 7:** Sustained network bandwidth as a function of message size sending and receiving concurrently in 5 dimensions. The memory footprint is thus twenty times the message size, and the performance can be seen to fall from the 35MB/s delivered, to the external memory sustained bandwidth of 27GB/s when the footprint exceeds the 32MB L2 cache capacity. This shows that the interconnect bandwidth exceeds the memory bandwidth, and shows the benefit of integrating the interconnect as high as possible in the memory system.

## 3. Performance, optimisation and codesign

The optimal code for BlueGene/Q must provide both SIMD data parallelism and thread level parallelism, and use the memory heirarchy efficiently. The utility of QCD to the design process

was as a diagnostic to indicate that, after best practice had been followed to achieve optimal code, high performance could indeed be obtained.

### 3.1 QPX SIMD optimisation and layout transformation

SIMD instructions are a cost efficient way to increase floating point throughput. Extracting the best performance from any modern design is always a challenge [10, 13, 14, 15, 18, 16, 11, 19, 20, 17], and this increases with vector length. A vector length four requires substantially more effort to use than earlier BlueGene designs because one is constrained to perform the same operations on elements of data that are consecutive in memory.

The author maintains the BAGEL package[**?**, 9] as a domain specific compiler for QCD. This has been enhanced with the introduction of a scheme that will support arbitrary vector length efficiently. Tailoring the BAGEL domain specific compiler with a detailed pipeline model of the architecture was of particular use in performance verification. The approach taken involved a global data layout transformation that guaranteed to expose data parallelism through pairs of complex operations.

The key observation in selecting the global code transformation was that for the subset of SPMD applications that perform the *same* sequence of operations on each node, *any* level $N$ of SIMD parallelism can be created by having each physical node perform the work of $N$ logical nodes. Data layout on the physical node should be transformed so that adjacent memory words contain data from distinct logical nodes and can be operated upon in a data parallel fashion.

For a concrete example, using *C* language storage ordering, consider the following transformation of two copies of an array representing the work of two logical nodes. Each logical node contains `Nxyzt` space-time sites, and `Nsc` spin or color indices, the transformation that allows SIMD operation upon both logical nodes in parallel can be loosely described as:

`complex Array[2][Nxyzt][Nsc]` $\rightarrow$ `complex Array[Nxyzt][Nsc][2].`

The sequence of scalar operations to act on a single logical node with a data array `complex Array[Nxyzt][Nsc]` is identical to the sequence of SIMD operations carried out in parallel on two logical nodes after the layout transformation. Each scalar addition turns into a SIMD addition and so on. However, the only changes are promoting scalar operations to SIMD operations and widening the complex data word size from 16 bytes to 32 bytes. This can be generalised to arbitrary width SIMD, and is a promising approach for several future architectures. Broadly, the class of layout transformation is similar to those enabled by the data parallel layout directives present in High Performance Fortran. After applying this transformation, the communication between (complex) SIMD lanes is restricted to only those portions of the code that communicate between (logical) nodes, and instruction overhead wasted in permuting data into the SIMD lanes is therefore limited to only the halo exchange, which involves a small subset of the data.

### 3.2 Memory system optimisation

Given the ratio between L2 cache and main memory bandwidth is greater than ten, it is of primary importance to select a loop ordering maximising reuse at the second level of cache. BAGEL is primarily designed to cover various 5d approaches to chiral fermions, and we consider the application of the five dimensional dirac operator. Here the gauge field can be reused $L_s$ times, and each

source fermion operand can be reused up to 10 times as it contributes to the result on each of the neighbouring sites.

The L1 is a write through cache. This means that every store, even to data carefully placed in the L1 cache, results in a store reflected all the way back to the L2. The accumulation of some expression in memory resident variables is therefore inadvisable as every step in the accumulation results in store traffic across the switch. In place, all accumulation variables were found to be best held in registers, and this determined the optimal loop ordering, for a single node, as follows:

```
for(four_dimensions){
  Lock 8 gauge links in L1
  for(fifth_dimension){
    for(mu){
      Load neighbour spinor to registers
      Spin project
      Load gauge link from L1 to registers
      SU(3) multiply
      Accumulate result in registers
    }
  }
  Unlock 8 gauge links in L1
}
```

Several features were introduced to the L1p to enable the most efficient QCD implementation. Given the loop ordering above, we generate a series of short contiguous bursts fetching the 10 (8 in 4d) neigbouring spinors. A mode was introduced, discussed as case (iii) in section2.2, in which we can programme this spinor size to ensure only required data is fetched by L1p.

A sequence of short burts would normally lead to ineffective prefetching. Fortunately, the code is deterministic: the addresses of each new burst can be known hundreds of cycles in advance when coded through offset lookup tables. An additional mode control in the L1p reinterprets a L2 specific cache touch to preplace a stream in the L1p, figure 8. In this mode the L1p fetches a programmable amount of data and holds this until needed by the core. Data, or a new stream, can thus be cheaply preplaced in the L1p hundreds of cycles in advance awaiting a demand fetch from the computation code.

The $L_s$ fold reuse of the gauge field $U_\mu$ is possible as they are applied to $L_s$ fermion fields. The innermost loop ordering, constrained by arguments above, requires that the same 8 gauge links must be refetched from memory multiple times. Arranging that these refetches always come from L1 is clearly the optimal solution. This is not easy, however, as we typically load around 1536 bytes of gauge fields $U_\mu$, and 24576 bytes of fermions for each 4d site. With 4 threads per core contending for use of the L1, the reuse is prevented by evictions of this precious gauge field data by the large volume of use-once fermion data. The PowerPC A2 core has several features to enable distinction of transient from reused data. The most attractive of these is user mode L1 line locking, which provides fine grained and user level control. An L1p mode was introduced to enable fast unlocking of lines, and locking gauge fields in the L1 gave a 15% speed up for $D_W$.

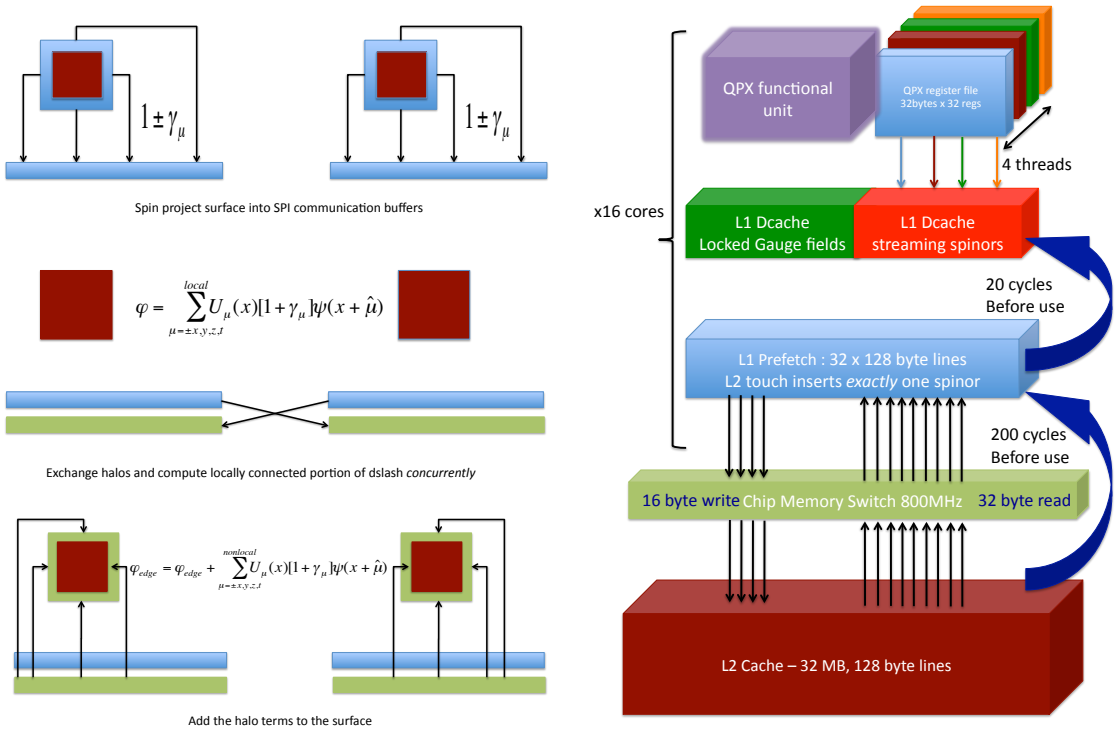


**Figure 8:** Left: The BAGEL implementation of multinode fermion operator. First spin project and starts communication of boundary fields. Overlapped with communication, calculate all contributions to each result site connected to interior fields. Complete communication and the exterior contributions on the local surface. Right: Scheduling the memory system effectively. Data motion can be scheduled by the careful assembler programmer between different levels in the memory system. Around 200 cycles before use, Bagel schedules dcbtl2 instructions to move an entire spinor into the L1p as a holding station. Around 20 cycles before use the spinor is streamed though the L1 cache into QPX registers. Gauge fields are held locked in the L1 cache to be reapplied $L_s$ times without eviction. The L1p provides prefetch hints to the L2 to ensure prefetching arises between the main memory and the L2.

### 3.3 Threading and Network optimisation

The implementation for multiple nodes selected is sketched in figure 8. BAGEL has adopted a model of using a single MPI process per node, and 64 threads. This has two benefits: avoiding unnecessary memory traffic to copy MPI packets internal to a node, and maximising the packet sizes for face communication between nodes in order to ensure we saturate bandwidth. Threads are created with the duration of, say, the entire linear solver (minimising fork-join) overhead, and IBM System Programming Interface (SPI) library calls are used synchronise these long lived threads and divide each element of work (for example a linear combination) between threads, with distinct threads producing disjoint subsections of the result vector.

The BlueGene/Q partitions provide a physical 5d torus network. QCD is normally best treated with message parallelism in either three or four dimensions. It is important to ensure that nearest neighbours in the lower dimensional simulation mesh (e.g. an MPI cartesian communicator) are

nearest neighbours in the physical network. Otherwise, the average packet travels some number of hops, each link must carry the face communications of multiple pairs of nodes. Further, when partitioned not all dimensions are guaranteed to be torii, and the absent wrapping link can cause a loss of performance. The problem of mapping nearest neighbour in a high dimensional torus, reduces to embedding a lower dimensional toroidal manifold that is space filling. A similar trick was introduced in QCDOC[4, 5, 6], and has also been adopted by the K-computer [7]. Figure 9 shows some examples that solve this problem in three dimensions. BAGEL uses SPI calls to fix communication buffer memory pages and determine virtual physical translations so that it can programme the BlueGene/Q DMA communication hardware directly. Both halo exchange and global reductions are accelerated in this way.

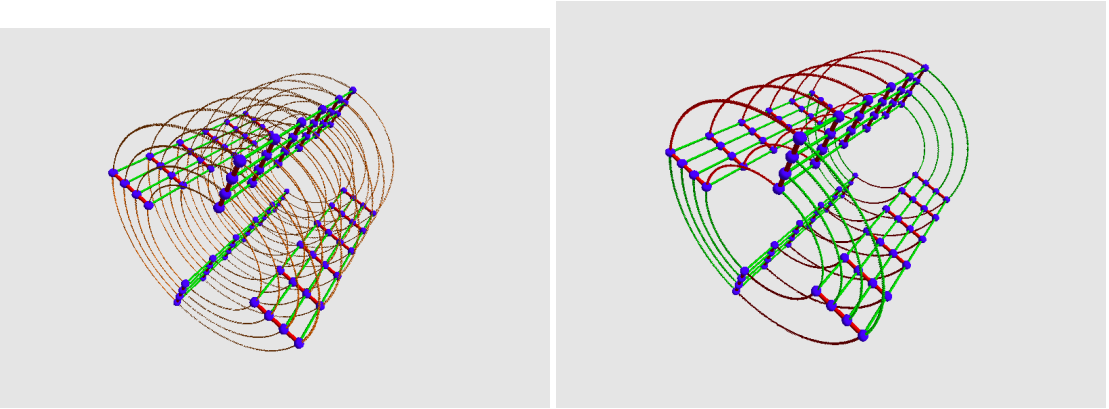

**Figure 9:** Here some solutions used by the author to fill a $4 \times 4 \times 6$ (radial-mesh, azimuthal-torus, axial-mesh). Left plot: the unmodified 3d mesh-torus-mesh displaying all physical links. Right plot: the links used to embed a $8 \times 12$ application torus (red,green) in this 3 dimensional space. Code implementing these mappings has been provided to the maintainers of QMP.

### 3.4 Performance

Very high sustained performance is possible on individual kernels figure 10 (left). In particular the Wilson kernel can obtain around 90 Gflop/s. The difference between single precsion and double precision is minimal when the working set fits in the L2 cache, due to the high bandwidth, but becomes large when the working set grows larger than the 32MB cache. Single precision figures are appropriate as defect correction mixed precision is used by RBC-UKQCD in production RHMC code. In multi-node code, Figure 10 (right) we introduce communication overhead. In particular, a double pass over the result vector is used fetching this twice from eithe L2 cache or main memory depending on the volume. Both the memory and network bandwidth limits impact the performance in double precision for large and small volumes repspectively. However, a good sweetspot can be seen where working set fits in cache, and the surface to volume ratio remains favourable.

Figure 11 displays the multinode performance of Bagel Domain Wall conjugate solver code on BlueGene/Q, and though somewhat slower due to additional linear algebra and reduction overhead, performance as high as 68Gflop/s is sustained. Weak scaling (with a fixed local volume of $8^4 \times 16$) is shown all the way up to very high sustained performance of 6.18 Pflop/s on a $128^3 \times 192 \times 16$

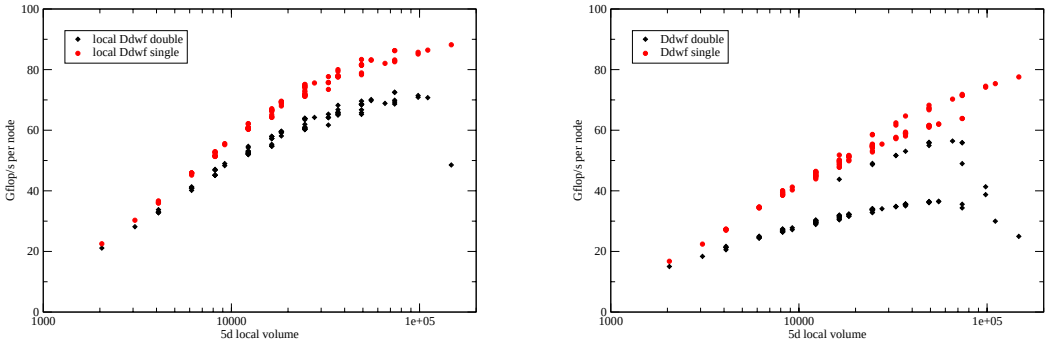


**Figure 10:** Left: single node application of $D_W$ to $L_s$ fermion fields. This is a key component of chiral fermion simulations, and contains around 90% of the floating point operations. L2 spillage occurs for 5d volumes greater than around $10^5$ sites. Right: multi-node application of $D_W$ to $L_s$ fermion fields. In double precision the communication impact of different aspect ratios for the local volume is visible.

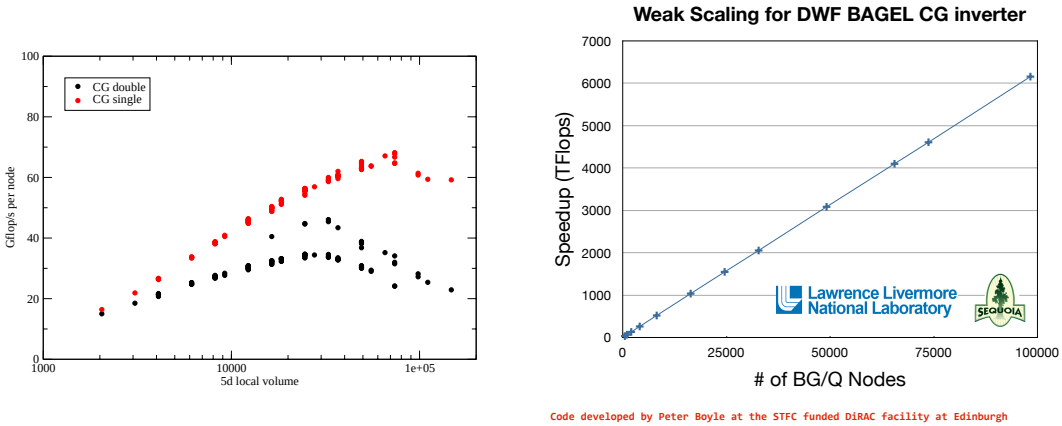


**Figure 11:** Scaling performance of Bagel Domain Wall conjugate solver code on BlueGene/Q. This includes all linear algebra and reduction overhead. Left hand plot displays performance per node as a function of the local volume. Performance as high as 68 Gflop/s is delivered. Right hand plot displays measured weak scaling (with a fixed local volume of $8^4 \times 16$) all the way up to very high sustained performance of 6.18 Pflop/s on a $128^3 \times 192 \times 16$ volume on the full Sequoia system at LLNL (with thanks to Michael Buchoff, LLNL).

global volume on the full Sequoia system at LLNL (with thanks to Michael Buchoff, LLNL). Unfortunately the Edinburgh machine is a little smaller (Figure 2).

### 3.5 Bagel support

BAGEL for BlueGene/Q supports the Wilson, Wilson twisted mass, Domain wall (5d prec, 4d prec). New support includes a broad class of 5d overlap actions:

$$\{Cayley, ContinuedFraction, PartialFraction\} \otimes \{Zolotarev, Tanh\} \otimes \{H_T, H_W\}$$

Clover support is in progress (Karthee Sivalingam). Bagel includes solvers using the conjugate gradient algorithm with multishift and mixed precision solvers variants. and additional algorithms are easy to implement [9]. Fermion force terms for the Wilson, overlap and domain wall actions have also been developed. The software is released under the GPL.

## 4. Competitive position

Broadly three models appear relevant over the next few years, as shown in table 4

| Model | environment | examples | |
|---|---|---|---|
| Homogenous<br>coherent many-core | MPI $\otimes$ OpenMP $\otimes$ SIMD | K-computer<br>BlueGene/Q | single instruction set<br>single language |
| Heterogenous<br>incoherent accelerators | MPI $\otimes$ CUDA<br>MPI $\otimes$ OpenCL<br>MPI $\otimes$ OpenAcc | GPU clusters<br>Cray XK7 | multiple instruction set<br>multiple language |
| Heterogenous<br>coherent many-core<br>accelerators | MPI $\otimes$ OpenMP $\otimes$ SIMD | Intel MIC clusters<br>SGI Altix+MIC | multiple instruction set<br>single language |

**Table 4:** Current and emergent programming models display a woeful lack of simplicity. One might expect some reversal of this process in future as the division between host and accelerator has no underlying advantage.

BlueGene/Q was the most power efficient supercomputer architecture in 2010 and 2011 (Green500), and the Sequioa system was the fastest computer in the world in June 2012. In November 2012 four of the top ten, and fifteen of the fastest 100 computers are BlueGene/Q installations. Since it is water cooled and has significant hosting requirements there is a minimum sensible system size, perhaps around 410 Tflop/s, below which installations may not make sense, but above which it should be very competitive.

Of the competing systems, the K-computer is most similar in design philosophy. By maintaining homogeneity, a single instruction set and language may be used for all aspects of the computer programme, and it is likely that the accessibility of these systems to a broad range of scientific codes is enhanced.

The introduction GPU accelerated Cray systems with good interconnect bandwidth subdivides GPU accelerator systems into those with capability class networks, such as the Titan installation, and those based on Infiniband will have lower levels of scalability. The many-named KnightsCorner/ManyIntegratedCore/XeonPhi device manufactured by Intel brings an interesting simplification of programming model to the accelerator class. Systems based on accelerator offload cards, communicating through a bus interface such as PCIe, are at a fundamental disadvantage for certain applications: this bus interface can become the bottleneck in the analysis of section 1 and limit scalability[12]. Integration of a communication network into the very high speed GPU memory system would be desirable step in future.

Depending on the system size, capital budget, and the whims of vendor bidding any of these systems can certainly be rational choices. Purchasing decisions should be made on delivered price/performance for real QCD benchmark code under competitive tender, bearing in mind that programming models may matter if required to run a wide range of code.

In future, it will cheaper to increase floating point performance and memory bandwidth to increase interconnect bandwidth. Through silicon vias enable 3D integration of logic and memory chips; this promises to increase memory bandwidths and reduce energy substantially by shortening "wires", while interconnect performance will remain expensive to increase.

Domain decomposition [21, 22, 24, 23] has been demonstrated to reduce communication overhead for the Wilson action. For five dimensional chiral fermions, the linear system has so far been most efficiently solved with the conjugate gradient algorithm. Domain decomposition for Cayley form 5d chiral fermions may be an import research activity for future machines. The analysis of section 1 was performed under the assumption of a conjugate gradient (or similar) solver, and all benchmarks in this work achieved scalability with Conjugate Gradient, demonstrating BlueGene/Q as a true capability machine.

## 5. Acknowledgements

PAB thanks the BlueGene/Q design team, and particularly thanks the other L1p design team members Norman Christ, Changhoan Kim, Alan Gara, Martin Ohmacht, and Krishnan Sugavanam. PAB thanks Michael Buchoff, Chris Schroeder and Pavlos Vranas for benchmarking the Sequoia system. PAB was supported in part by STFC grants ST/J000329/1, ST/K005804/1, ST/K000411/1 and ST/H008845/1.

## References

[1] R.A. Haring, M. Ohmacht, T. Fox, M. Gschwind, D.L. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M.A. Blumrich, R. Wisniewski, A. Gara, G.L. Chiu, P.A. Boyle, N. Chist, and C. Kim, "The IBM Blue Gene/Q Compute Chip", IEEE Micro, 2012, pp.48-60.

[2] "Blue Gene/Q: by co-design", The Blue Gene Team, Comput Sci Res Dev, 1865-2034, doi:10.1007/s00450-012-0215-3. Long author list.

[3] "Codesign of the BlueGene/Q Level 1 Prefetch Engine with QCD" P. Boyle, N. Christ, and C. Kim, accepted by the IBM Journal for Research and Development. *in print 2012*.

[4] P. Boyle, D. Chen, N. Christ, M. Clark, S. D. Cohen, C. Cristian, Z. Dong and A. Gara *et al.*, "The QCDOC project," Nucl. Phys. Proc. Suppl. **140**, 169 (2005).

[5] P. A. Boyle *et al.* [QCDOC Collaboration], "The QCDOC supercomputer: Hardware, software, and performance," eConf C **0303241**, THIT003 (2003) [eConf C **0303241**, THIT002 (2003)] [eConf C **0303241**, THIT001 (2003)] [hep-lat/0306023].

[6] Boyle, P. et al., "Overview of the QCDSP and QCDOC computers", IBM JRD 492 2/3 (2004) 351.

[7] Yuichiro Ajima, Shinji Sumimoto, Toshiyuki Shimizu, "Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers," Computer, vol. 42, no. 11, pp. 36-40, Nov. 2009, doi:10.1109/MC.2009.370

[8] P. A. Boyle, "The BAGEL assembler generation library," Comput. Phys. Commun. **180** (2009) 2739.

[9] Qi Liu has contributed an EigCG implementation. Rudy Arthur has contributed polynomial filtered implicitly restarted Arnoldi and Lanczos eigensolvers, BiCG and MCR solvers. Karthee Sivalingam has developed a clover term.

[10] M. Luscher, Nucl. Phys. Proc. Suppl. **106**, 21 (2002) [hep-lat/0110007].

[11] R. Babich, M. A. Clark and B. Joo, arXiv:1011.0024 [hep-lat].

[12] http://www.olcf.ornl.gov/wp-content/training/ascc_2012/thursday/ACSS12-Balint-Joo_s.pdf

[13] C. McClendon, "Optimized Lattice QCD Kernels for a Pentium 4 Cluster", Jlab preprint, JLAB-THY-01-29

[14] A. Pochinsky, Nucl. Phys. Proc. Suppl. **140**, 859 (2005).

[15] A. Pochinsky, J. Phys. Conf. Ser. **46**, 157 (2006).

[16] A. Pochinsky, PoS LATTICE **2008**, 040 (2008).

[17] T. Boku, K. -I. Ishikawa, Y. Kuramashi, K. Minami, Y. Nakamura, F. Shoji, D. Takahashi and M. Terai *et al.*, arXiv:1210.7398 [hep-lat].

[18] J. Doi, PoS LAT **2007**, 032 (2007).

[19] K. Jansen and C. Urbach, Comput. Phys. Commun. **180**, 2717 (2009) [arXiv:0905.3331 [hep-lat]].

[20] S. Krieg and T. Lippert,
[22, 24, 23]

[21] Y. Saad, "Iterative methods for sparse linear systems", SIAM 2003.

[22] M. Luscher, Comput. Phys. Commun. **156**, 209 (2004) [hep-lat/0310048].

[23] L. Del Debbio, L. Giusti, M. Luscher, R. Petronzio and N. Tantalo, JHEP **0702**, 082 (2007) [hep-lat/0701009].

[24] M. Luscher, Comput. Phys. Commun. **165**, 199 (2005) [hep-lat/0409106].