BlueGene/Q

Peter Boyle
University of Edinburgh

June 29, 2012

Outline

- The design challenge
- BlueGene/Q[1]
  - Architecture
  - Optimising for BlueGene/Q
  - Performance
- Competitive position

[1] Will perform analysis by subsystem over floating point, memory, and network

Wilson Dirac Operator

Usual Wilson matrix is

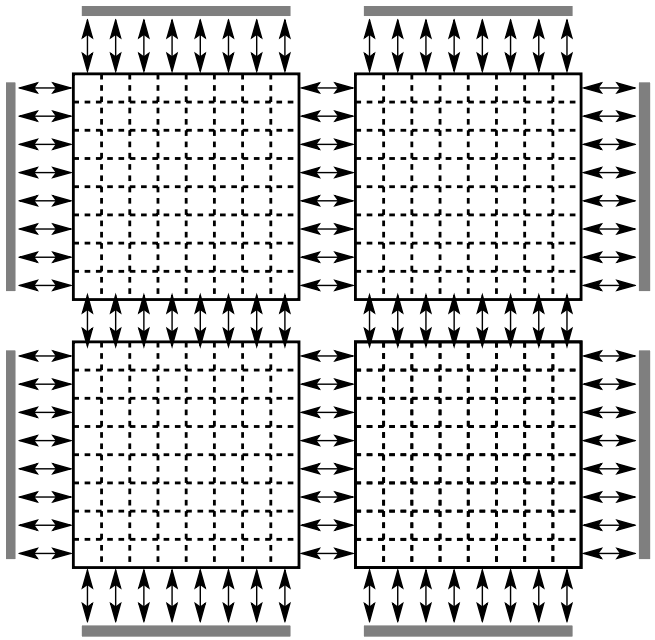
$$D_W(M) = M + 4 - \frac{1}{2} D_{\mathrm{hop}},$$

where

$$D_{\mathrm{hop}} = (1 - \gamma_\mu) U_\mu(x) \delta_{x+\mu,y} + (1 + \gamma_\mu) U_\mu^\dagger(y) \delta_{x-\mu,y} \tag{1}$$

Dirac equation is a classic sparse matrix problem

- Geometrical decomposition on multiple nodes
- Messages representing halos communicated between nodes
- 4d-Torus communications pattern
- Cost $\sim O(L^{11}) - O(L^{13})$
  $\Rightarrow$ strong scaling requirement
- Will focus on 5d overlap/dwf formulations with D_W as building block



(Simplified) Wilson matrix performance analysis

Model time to apply Wilson operator as $t_{Wilson} = Max\{t_{comm}, t_{fpu}, t_{memory}, t_{cache}\}$

Wilson operator D_W

- $2 \times 24L^4$ words to memory
- $9 \times 24L^4$ words to cache [2]
- $16 \times 12L^3$ words bidi comms

FPU

- $1320 \times L^4$ flops: 480 MADDS, 96 MULS, 264 ADDS

Challenge: design network and memory bandwidth so $t_{cache}, t_{comm}, t_{memory} \approx t_{fpu}$

Assumptions

- When coded right these will take place concurrently. The longest will determine time
- loop order will maximise cache reuse; count *compulsory* memory traffic
- Inverter working set does not fit in cache; 8x reuse in sparse matrix
- Ignore gauge field U_μ as high L1 reuse in DWF and overlap fermions
- Note: including preconditioning does not change analysis if we use $2L \times L^3$

[2] "cache" really means the highest level of memory at which reuse can occur. This may be some form of local memory on certain systems.

How fast can a computer go?

$B_N/B_M/B_C$ are Network/Memory/Cache bandwidths (fp words/sec)

- Scalability limited when t_{comm} large $\Rightarrow$ minimum sensible local volume L_{min}

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

- D_W scalability determined by ratio of network bandwidth to cache & memory bandwidth [3]
- Maximum performance on a given total problem size N then determined by L_{min}. e.g.

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

- Maximum performance and scalability fall as *fourth power* of network bandwidth.

[3] or floating point processing rate – whichever is rate limiter – usually bandwidth

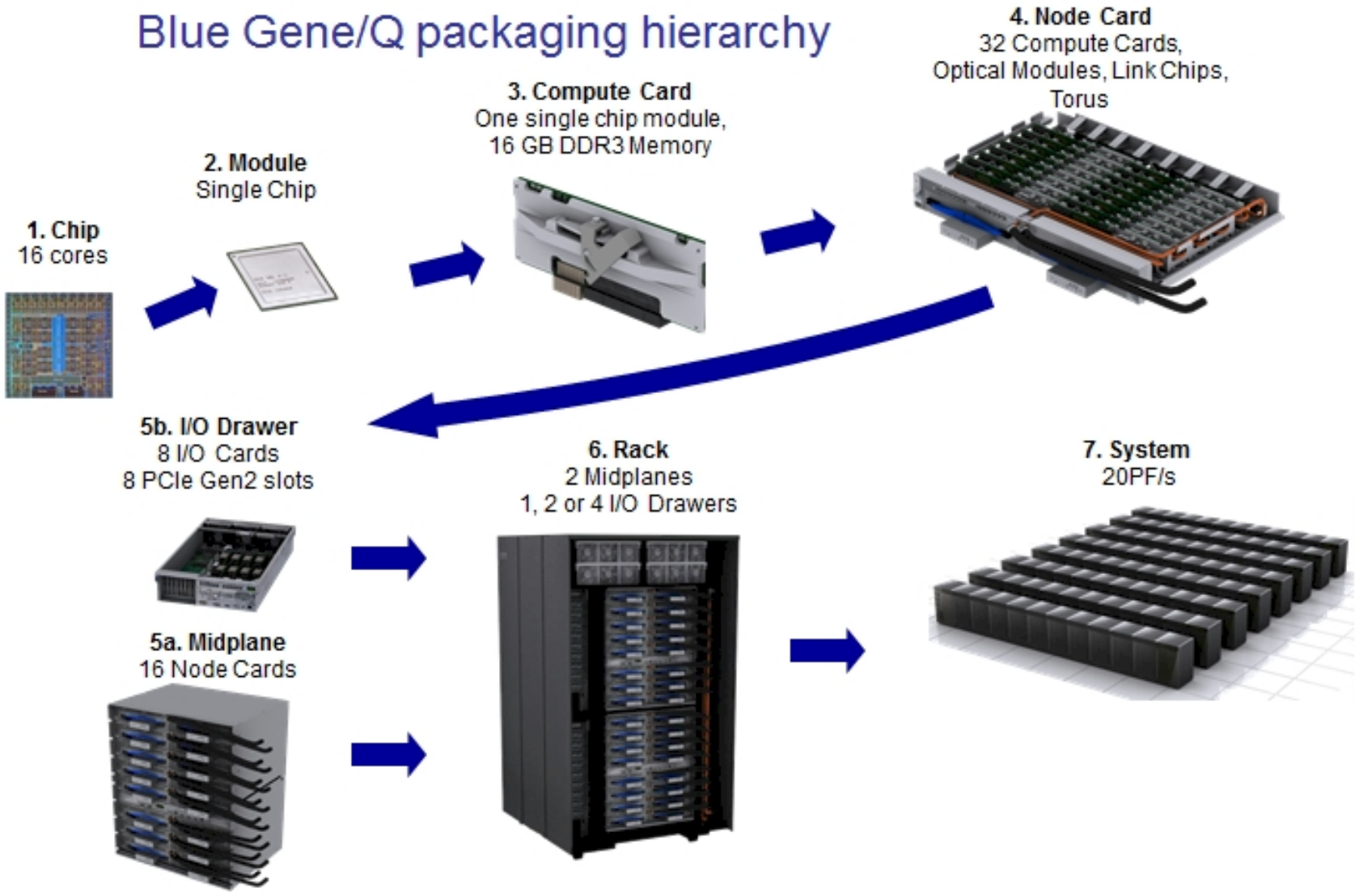
An example

- Consider 64^4 on an nominal 100Gflop/s node

$\frac{B_C}{B_N}$	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

- Conclusion: integrate network controller in the memory system so that $B_N \sim O(B_C)$

Machines: BlueGene/Q



BlueGene/Q overview

- 45nm, 360mm^2, 1.6GHz, 55W
- 16 $\times$ PowerPC 64 bit compute cores (+1 O/S +1 yield)
- 16 KB L1 data cache, 4KB L1p prefetch engine, 32 MB L2 cache
- 16GB DDR3 1333 memory (dual controller : 2 $\times$ 128 bit I/F)
- 4 threads per core, 64 threads per chip
- Quad double precision short vector (SIMD) fpu
  Can operate as twin complex arithmetic fpu
  8 floating point operations per clock cycle *if* $z = ax + y$
  4 floating point operations per clock cycle *if* $z = x + y$
  4 floating point operations per clock cycle *if* $z = x * y$
  Wilson is limited to 78% of peak
- FP/Memory/Network bandwidths

GFlop/s	L1 GB/s	L2 GB/s	DDR GB/s	Torus GB/s
204.8	820GB/s	563(448)	42.7	40

- SoC integrates huge cache, huge MPI bandwidth ($\equiv$ O(10) Mellanox cards) within modest area and power budget
  $\Rightarrow$ scalable *and* power efficient

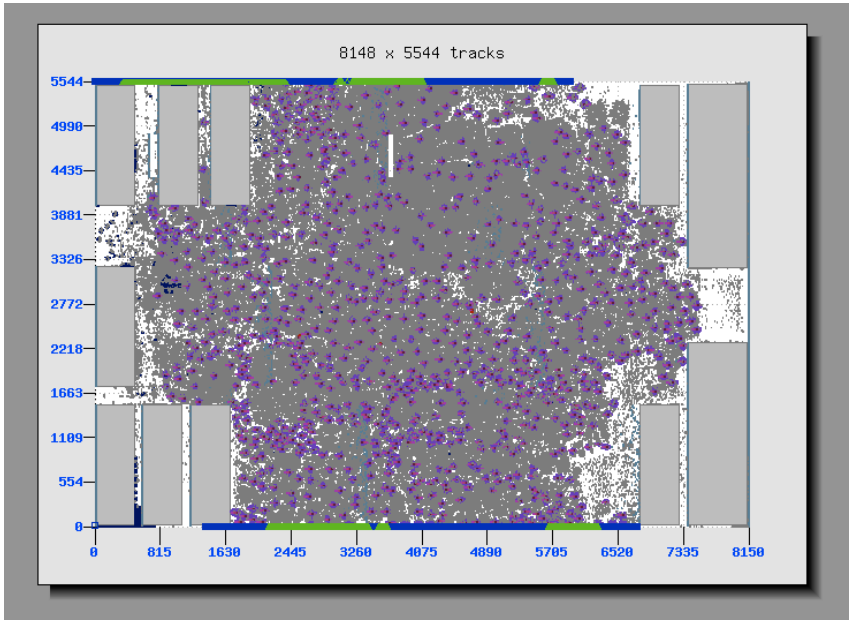
Edinburgh/Columbia/IBM Collaboration

Dec 2007 IBM Research, Edinburgh U., Columbia U. formed a collaboration agreement to jointly develop next generation of BlueGene.

2007-2011 PAB (UoE), Christ (CU), and Changhoan Kim (CU, now IBM) designed adaptive memory prefetch engine (L1P) as contractors.
VHDL logic design, clock tree, test structures, timing closure and placement

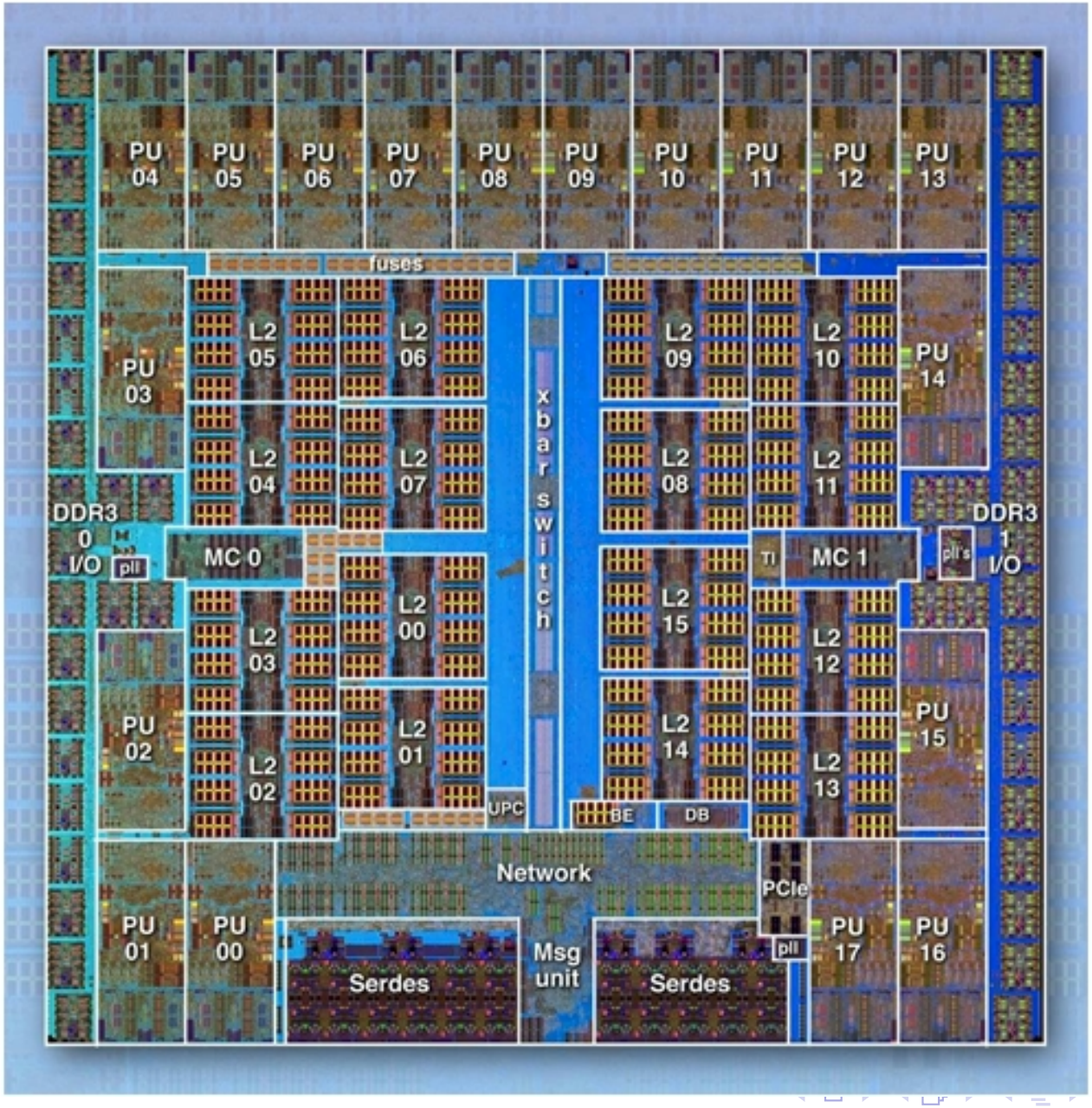
- *QCD assembler and hardware prefetcher jointly developed*
  $\longrightarrow$ The *design* element of codesign is truly important

Can you find L1p in the next slide's die photo?



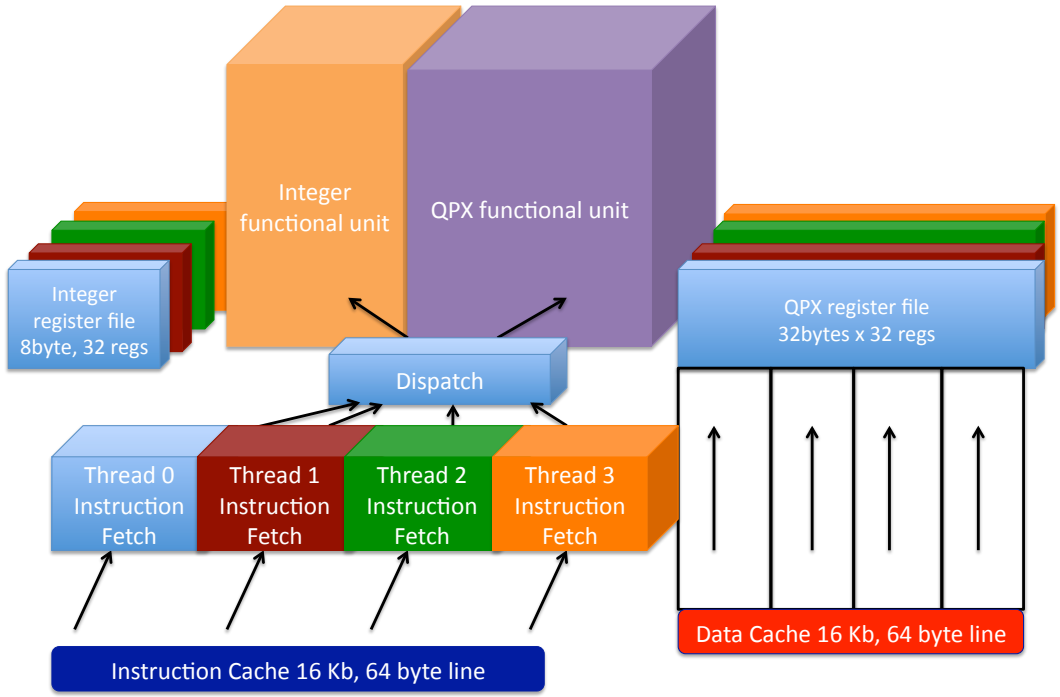
Hint: SRAMS are the rectangular blocks - match the SRAM pattern

BlueGene/Q die photo



BlueGene/Q processor core

- Most processors in the world spend 95% of their time idle stalled on memory
- If fetch *independent* instructions from another thread they can be executed
- Replicate instruction fetch, register files. Share the big functional units



- QPX loads, stores and operates on four consecutive double prec. words in parallel

SIMD optimisation

QPX supports paired complex SIMD operations (quad double)

- Develop BAGEL domain specific compiler for BG/Q QPX support
- Remember why SIMD was *easy* on the Connection Machine!
  - Subdivide node volume into smaller *virtual nodes*
  - Spread virtual nodes across SIMD lanes (these were memory banks in CM5)
  - Modifies data layout to align data parallel operations to SIMD hardware
- Data parallel operation on both virtual nodes is now simple
  - Crossing between SIMD lanes restricted to during cshifts between virtual nodes
  - Code to treat N-virtual nodes is identical to scalar code for one, except datum is N fold bigger

$$\underbrace{(A, B, C, D)}_{\text{virtual subnode}} \quad \underbrace{(E, F, G, H)}_{\text{virtual subnode}} \rightarrow \underbrace{(AE, BF, CG, DH)}_{\text{Packed SIMD}}$$

  - CSHIFT involves a CSHIFT of SIMD, and a permute *only* on the surface

$$(AE, BF, CG, DH) \rightarrow \underbrace{(BF, CG, DH, AE)}_{\text{cshift bulk}} \rightarrow \underbrace{(BF, CG, DH, EA)}_{\text{permute face}}$$

SIMD made easy

- Sequence of operations remains the same as on BG/Q after BAGEL layout transformation
- O(100%) SIMD efficiency

Optimised sequence of operations is *identical* for scalar complex and SIMD operation
BG/L(left, scalar complex) and BG/Q(right vector complex) assembler comparison

```
bt gt,  __lab3
addi.  %r9 , %r13 , 0
__lab3:
fxcxnpma  0 , 30 , 29 , 26
dcbt  %r18,%r9
fxcxnpma  1 , 30 , 22 , 24
stfpdx  9,%r21,%r17
fxcxnpma  2 , 30 , 7 , 23
stfpdx  10,%r22,%r17
fxcxnpma  3 , 30 , 28 , 27
dcbt  %r20,%r9
fxcxnpma  4 , 30 , 21 , 25
stfpdx  11,%r23,%r17
fxcxnpma  5 , 30 , 6 , 31
la   %r16, -1(%r16)
fxpmul  7 , 15 , 0
dcbt  %r22,%r9
fxpmul  6 , 12 , 0
```

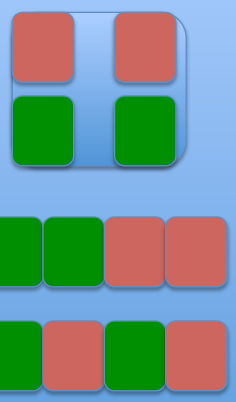
```
        bt gt,  __lab3
         addi  %r9 , %r13 , 0
__lab3:
         qvfxxnpmadd  0 , 29 , 30 , 26
         dcbt    %r18,%r9
         qvfxxnpmadd  1 , 22 , 30 , 24
         qvstfdx  9,%r21,%r17
         qvfxxnpmadd  2 , 7 , 30 , 23
         qvstfdx  10,%r22,%r17
         qvfxxnpmadd  3 , 28 , 30 , 27
         dcbt    %r20,%r9
         qvfxxnpmadd  4 , 21 , 30 , 25
         qvstfdx  11,%r23,%r17
         qvfxxnpmadd  5 , 6 , 30 , 31
         la   %r16,      -1(%r16)
         qvfxmul  7 , 15 , 0
         dcbt    %r22,%r9
         qvfxmul  6 , 12 , 0
```

Path to wider SIMD?

- F90 data parallel compiler with HPF-like distribute extensions controlling *both* SIMD and Thread parallelism could be an exascale killer app
- cmfortran + MPI !

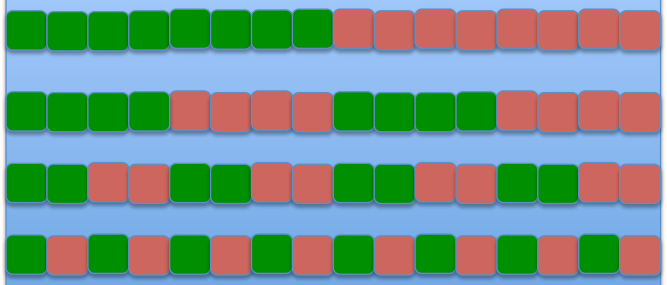
Generalises to wider SIMD

- 2x2 (SSE single, Altivec)



Permute/insert/extract stencils simple

- 2x2x2x2 (MIC)



- Permutation/insert/extract masks required for 16 way SIMD & 2x2x2x2

- 50% efficiency face operations till 16 way SIMD
- 25% efficiency for up to 4^4 = 256 way SIMD

L1p architecture

Interface between core and memory switch

- Handles coherency, synchronisation, write and read traffic
- 1600MHz - 800MHz domain crossing
- MPGZ - Asic design process boundary

Write combine

- Hold recently written data; merge with later writes to same 32byte line
- 20 entry buffer for writes allows core to continue progress

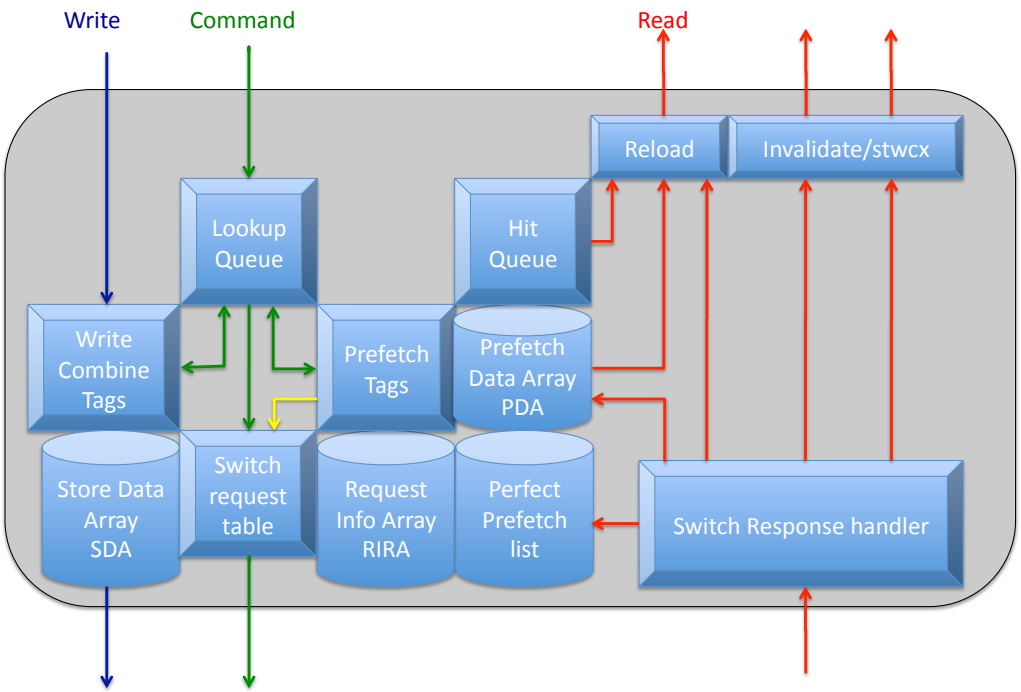
Prefetch cache

- 4KB fully associative prefetch cache: 32 lines, 128b line size
- Coherent & consistent ; invalidates lines if written

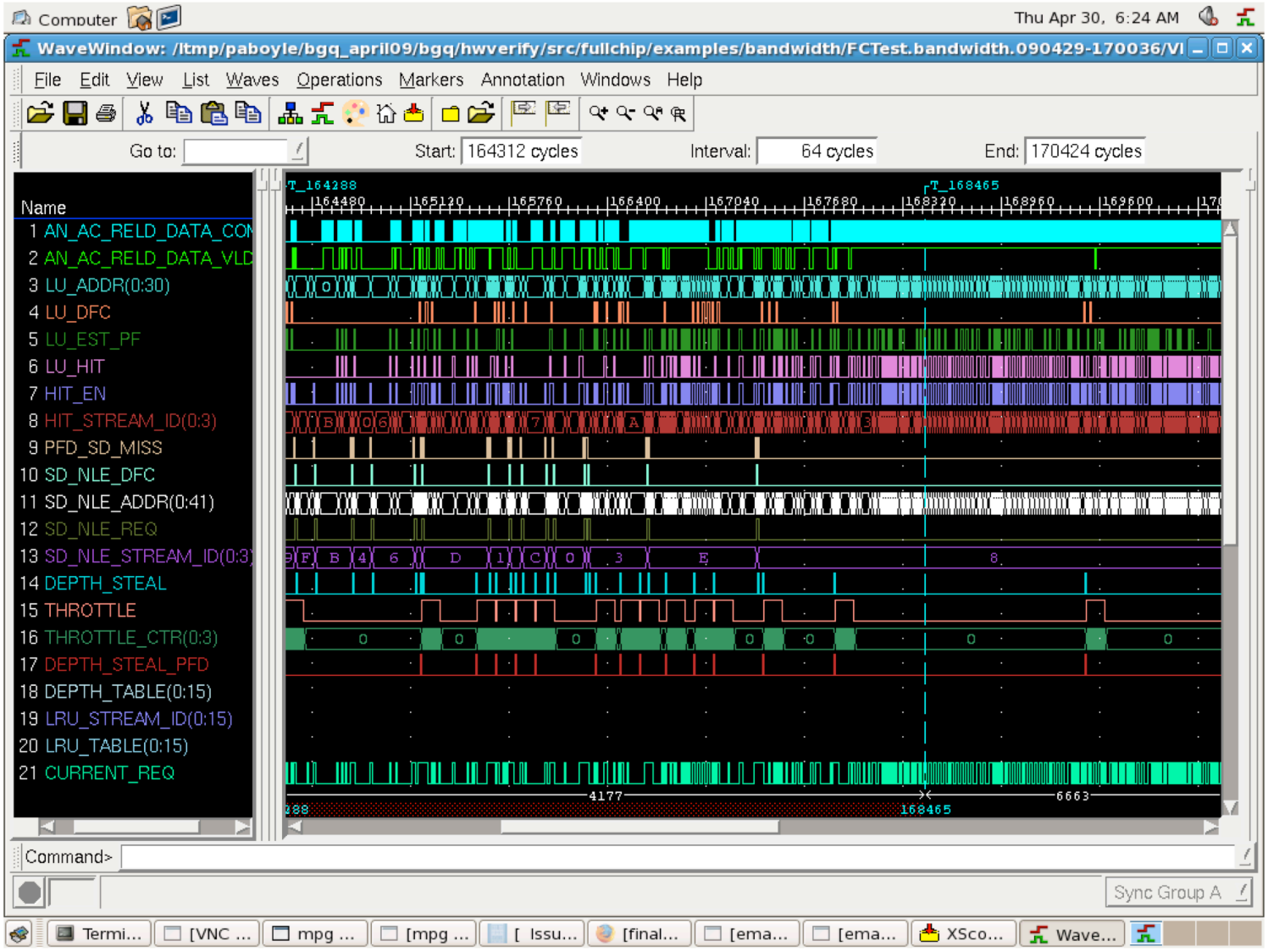
Stream prefetch

- Adaptive length prefetch (configurable)
- Fixed length prefetch (optional)
- 1-16 streams of depth 1-8 each

Perfect prefetch – record access patterns and replay



Adaptive prefetch



Memory optimisation

Loop order determined by

1. maximising reuse
2. avoiding write through traffic

Must accumulate spinors in QPX registers

```
for(x) {
 for(s=0; s< Ls; s++) {
  for ( mu ) {
    psi[s][x] += U[x][mu] (1+gamma[mu]) psi[x+mu]
  }
 }
}
```

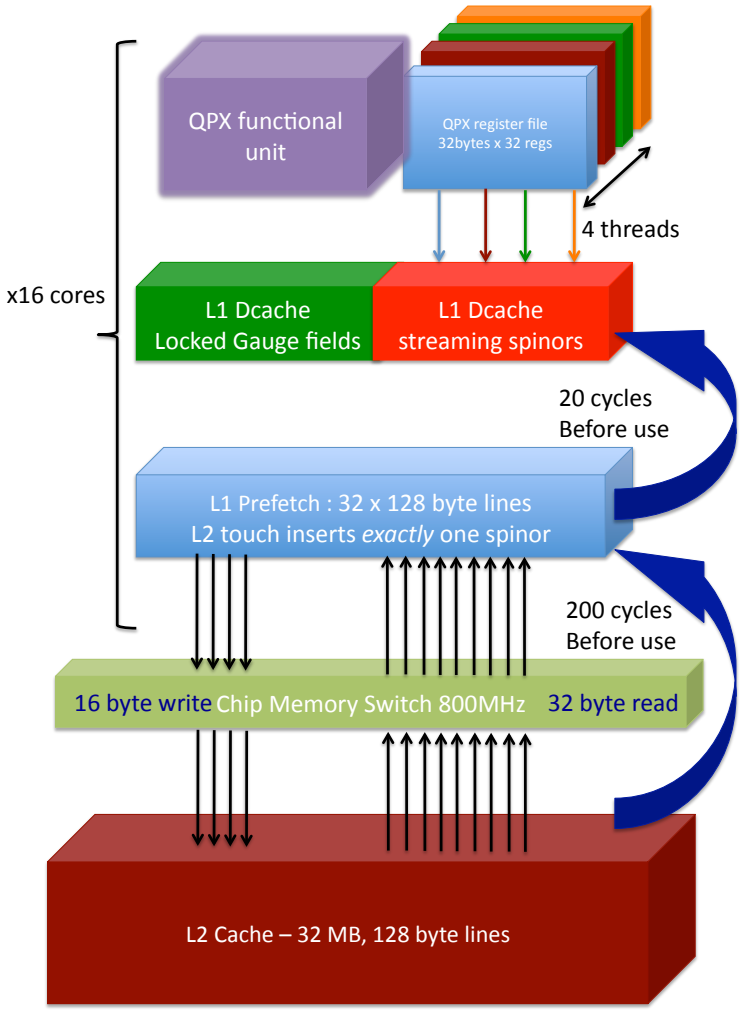
⇒ *short streams* length – one spinor 192/384 bytes
⇒ introduce L1p modes to optimise for this

- Program *predetermined* stream length (1-8 L2 lines)
  Avoids wasting bandwidth fetching beyond the end
  Essentially a programmable line size

Programmer tunes fetch hardware to *exactly* match algorithms spatial locality
Particularly important when there is no temporal locality benefit of cachelines

- Change effect of *dcbtl2* hint to preplace spinor in L1p
- Hide L1 locking from L2 to avoid unlock overhead

Dataflow


QPX functional unit
QPX register file
32bytes x 32 regs
4 threads
x16 cores
L1 Dcache
Locked Gauge fields
L1 Dcache
streaming spinors
20 cycles
Before use
L1 Prefetch : 32 x 128 byte lines
L2 touch inserts *exactly* one spinor
200 cycles
Before use
16 byte write
Chip Memory Switch 800MHz
32 byte read
L2 Cache – 32 MB, 128 byte lines

BlueGene/Q network design

5 dimensional routing torus network: ABCDE

- 2GB/s send + 2GB/s recv per link
- 40GB/s over 5 dimensions, 32GB/s available to 4d QCD partition
- E dimension remains on the the node
- D dimension connects top bottom in rack
- A,B,C,D connections between midplanes

Use up to 3d optical torus between racks

- Copper within midplane
- Optics is expensive in power and money
- High density racks reduce optics overhead by surface to volume ratio
- Physical space is only a secondary issue pushing high density

Message unit resources

- Messaging engines for each thread: able to do local copying as well as remote
- Up to 64 MPI processes per node possible

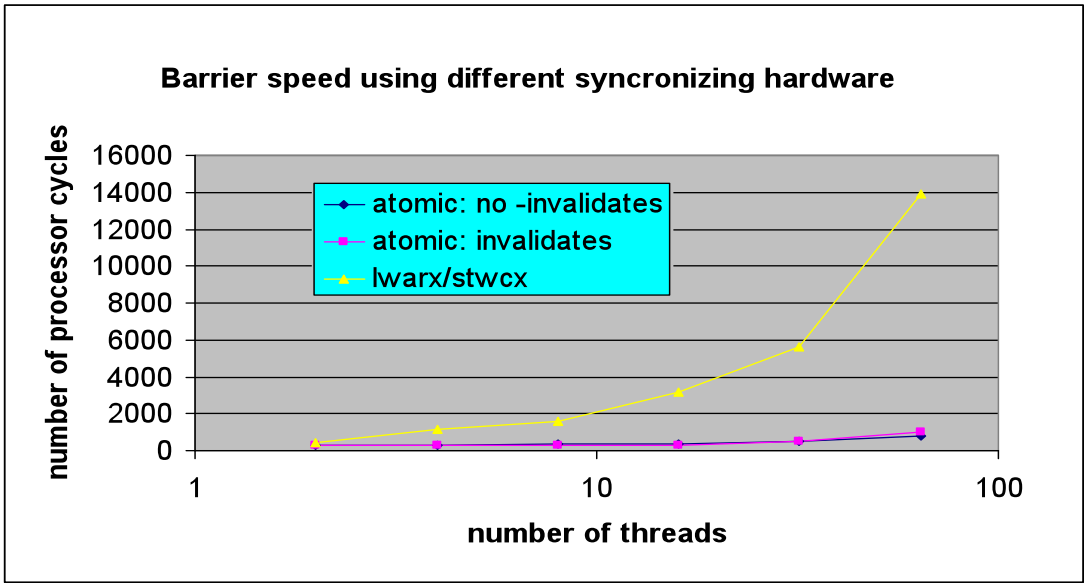
Edinburgh BlueGene/Q prototype (1.26Pflop/s)

Six Edinburgh BG/Q racks
Part of STFC funded DiRAC facility

BG/Q thread acceleration

L2 atomic operations

- High order address bits invoke non-cache atomic operations
  Use to implement *fast* 600ns barriers
  Barriers carefully designed to send *absolute* minimum across memory switch

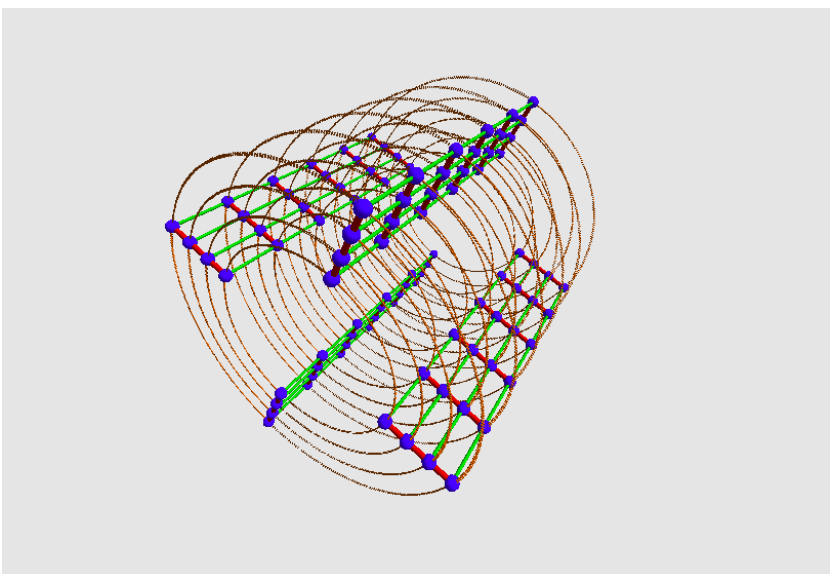


Bagel uses 64 threads and one MPI process per node

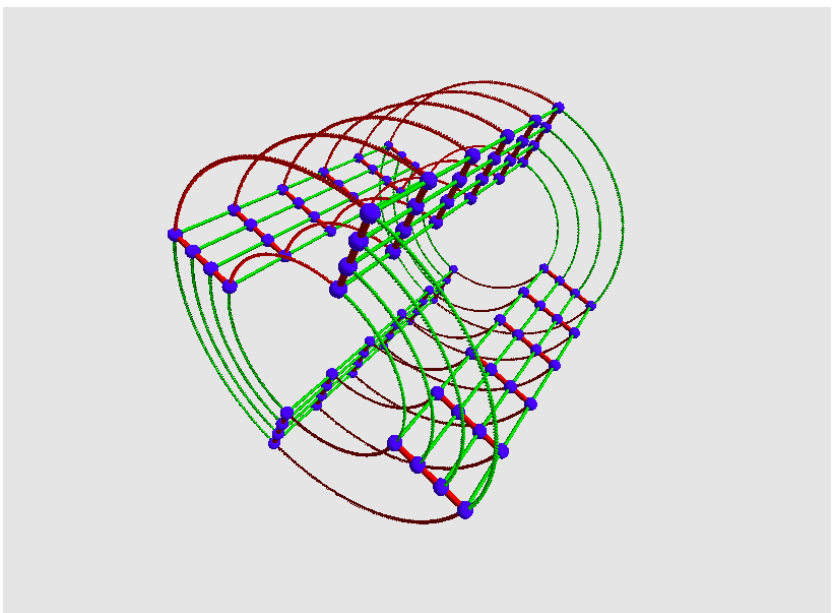
- Long lived threads duration of solver
- Barrier synchronisation
  - minimises fork/join overhead
  - External packet size is maximised giving best MPI bandwidth
  - Internal copying for MPI within node is eliminated
- Use System Programming Interface (SPI) to obtain best performance
- Pin communications buffers in dedicated pinned memory pages etc.

BAGEL Torus mapping

- BlueGene/Q has a 5d physical torus/mesh
  Periodic link not guaranteed on all paritions
- Can *always* ensure 3d torus using improved version of QCDOC dimension folding
  - Fold periodic dimension length 4 into *two* orthogonal mesh directions
  - QMP patch supplied to James Osborne
- Bagel code uses SPI DMA communications for halo exchange
  Coexists *gracefully* with MPI in rest of code
  3x speed up at small volumes

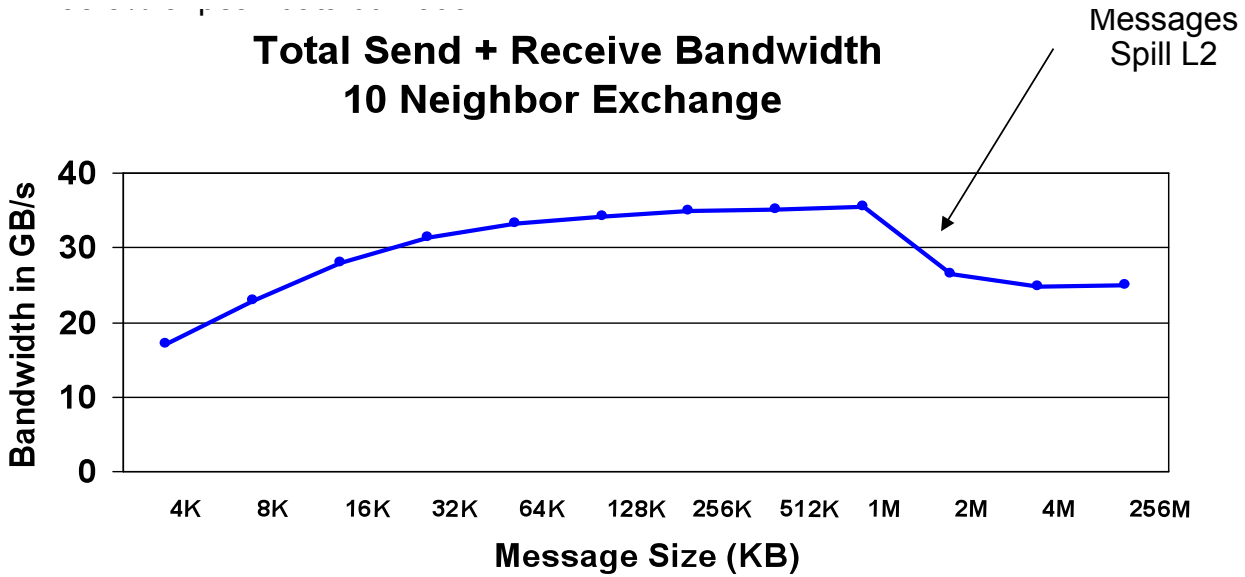


4 (torus, circles) x 4(mesh, line) x 6(mesh, line) physical grid



8 (torus) x 12(torus) logical grid

SPI network performance

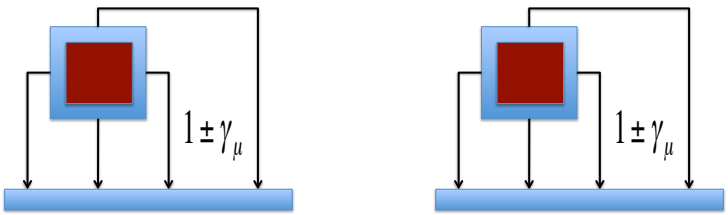


90% link saturation; delivered network bandwidth *exceeds* DDR memory bandwidth
$\Rightarrow$ designed for scaling

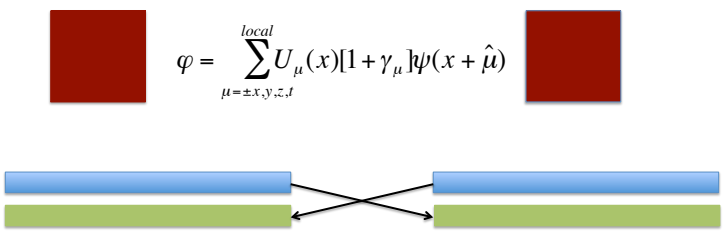
600ns latency available through SPI

$\not\partial$ implementation

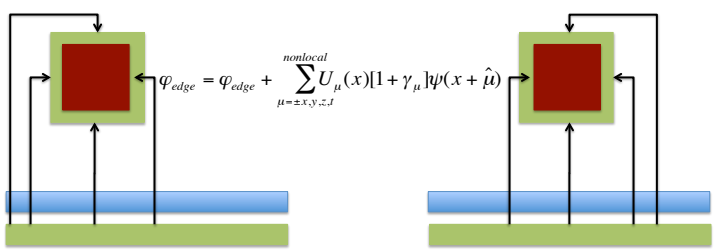
- Single-node, double precision get 110Gflop/s (65% pipeline usage) within L2 cache
- Multi-node cache optimal loop order forces two pass approach to overlap comms & compute (interior/exterior)



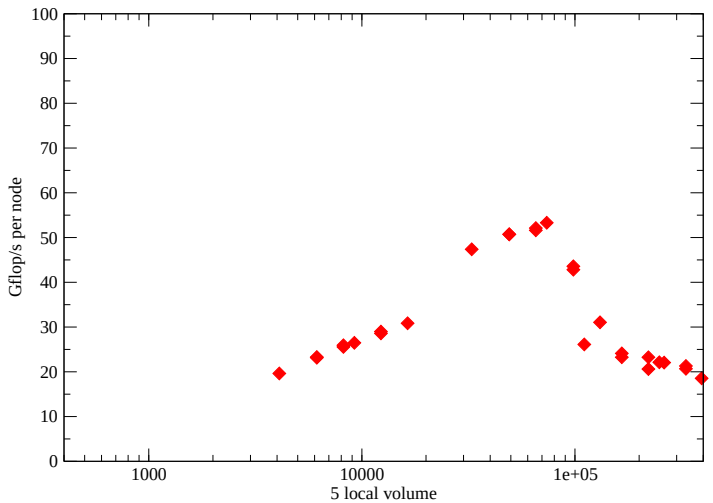
Spin project surface into SPI communication buffers



Exchange halos and compute locally connected portion of dslash *concurrently*



Add the halo terms to the surface



Multi-node double precision DWF dslash performance

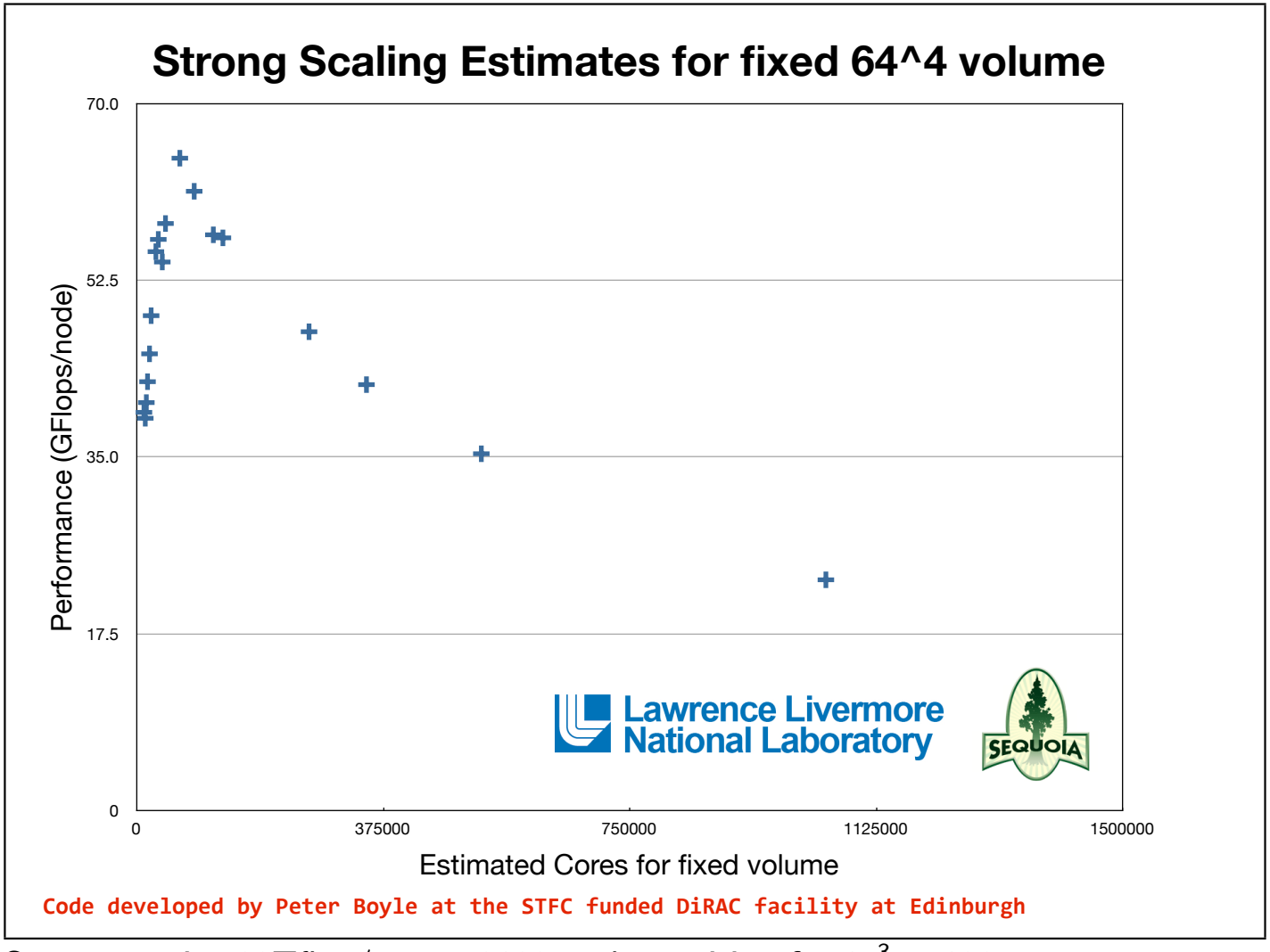
Conjugate gradient optimisation

Minimise load on off-chip memory

- Linear combinations have a sizable performance impact due to memory bandwidth
  Fuse $\mathbf{Z} = \mathbf{Z} + A\mathbf{X} + B\not{D}\mathbf{Y}$ operations into Dslash routine to avoid write-reread cycles
- Alternate CG recurrence expression eliminates serial dependency
  Fuse all linear combinations into a single pass to increase cache reuse
  (Pipelined CG, Strzodka & Goddeke)
- Implement single inner, double outer defect correction
  $\Rightarrow$ correcting single precision defect is a *rapidly* convergent expansion

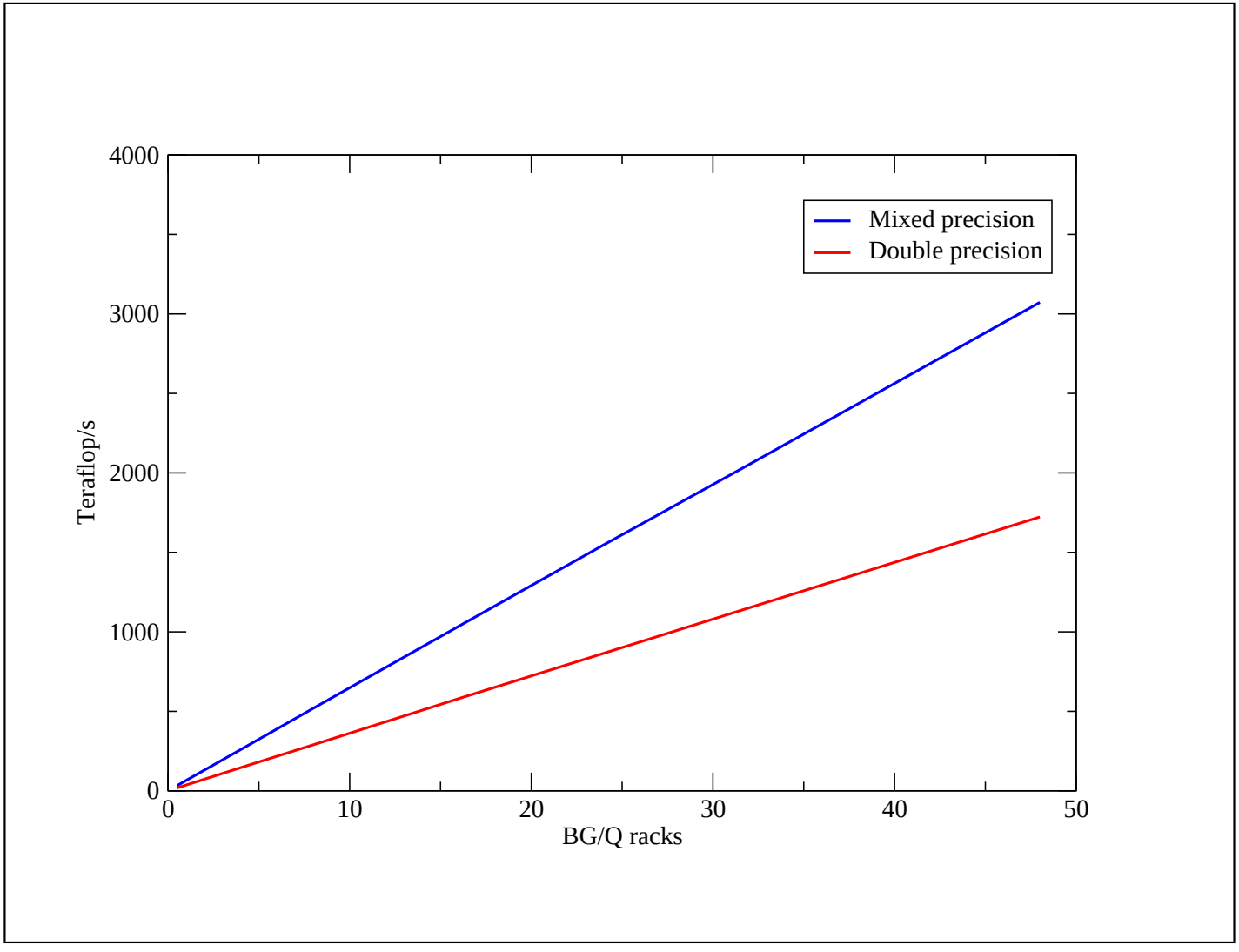
Time for something a little bigger!

DWF mixed precision CG on Sequoia



Sweet spot is 532Tflop/s on a 8192 node partition for $64^3 \times 128$.
Plot thanks to Michael Buchoff, Pavlos Vranas, Joseph Wasem, Christopher Schroeder, Thomas Luu and Ron Soltz at Lawrence Livermore National Laboratory.

DWF CG performance on Sequoia (48 racks, 50% machine)



Weak Scaling on $8^4 \times 16$ local volume

Thanks to Michael Buchoff, Pavlos Vranas, Joseph Wasem, Christopher Schroeder, Thomas Luu and Ron Soltz at Lawrence Livermore National Laboratory.

Record performance

- 3.07 Petaflop/s *sustained* performance on half the Livermore system
- 32% of peak, 41% Floating point pipeline usage over *entire* CG
- Global volume equivalent to $128^3 \times 96 \times 16$
- Expect 6.14 Petaflop/s on useful global volume equivalent to $128^3 \times 192 \times 16$ for full machine
- Smashed the Petaflop/s barrier 14th Jun 2012!

Bagel support

Algorithms

- CG, Multi-shift CG, Mixed precision CG
- Polynomial filtered implicitly restarted Arnoldi/Lanczos (Rudy Arthur)
- New algorithms easy to implement (Qi Liu: EigCG, Arthur: Bicg variants)

Actions

- Wilson
- Wilson twisted mass
- DWF (5d prec)
- DWF (4d prec)
- 5d Overlap : $\{Cayley, ContinuedFraction, PartialFraction\} \otimes \{Zolotarev, Tanh\} \otimes \{H_T, H_W\}$
- Clover in progress (Karthee Sivalingam)

Bagel is GPL software and (beta) available from `www.ph.ed.ac.uk/~paboyle/bagel` (v2.95)
In process of freezing v3.0.0

Competitive position

- Most efficient computer in Green500
- Fastest computer Graph500 memory system benchmark
- Fastest computer in the world top500 June 2012 @ 16.32 Pflop/s Linpack
- 2 systems in top 3 / 6 systems in top 20 / 15 systems in top 100

Planned QCD deployments:

- 1.26 Pflop/s dedicated QCD racks at KEK
- 1.26 Pflop/s dedicated QCD racks at Edinburgh (DiRAC)
- 0.630 Pflop/s dedicated QCD racks at BNL
- 20Pflop/s Livermore shared use
- 10Pflop/s Argonne shared use
- 2.1 Pflop/s Cineca shared use (Fermi
- 1.64 Pflop/s Juelich shared use

$\Rightarrow$ aggregate multi-Pflop/s sustained QCD performance

The competition

Architecture	Cache read BW/size	Memory BW/size	Network BW	$L_{min} \sim \frac{B_C}{B_N}$
BG/Q	410GB/s , 32MB	43GB/s, 16GB	40GB/s (30)	10 (8)
K-computer	??/6MB	64GB/s, 16GB	100GB/s (60)	(4??)
Cray XK6 (twin GPU)	??	354GB/s , 12GB	20 GB/s[4]	18
GPU+infiniband 1:1	??	150GB/s , 6GB	5GB/s	30
GPU+infiniband 4:1	??	600GB/s , 24GB	5GB/s	120

- Educated guess: K-computer looks more scalable but *as far as I can tell* it only provides 3d partitions $\Rightarrow 4^3 \times L$ minimum volume?
- Prediction for Titan (XK6) broadly consistent with results of Clark & Joo for bicgstab They find domain decomposition helps for Wilson/Clover, 1.5x less numerically efficient
- GPUs + IB (1:1) will allow modest scaling on big volumes
- GPUs + IB (4:1) will not scale beyond one node on any reasonable lattice

Broadly two models emerging:

- Coherent many-core nodes: MPI $\otimes$ OpenMP $\otimes$ SIMD
- Incoherent accelerator nodes: MPI $\otimes$ CUDA/OpenCL/OpenAcc
- Intel MIC somewhere in between

[4] CudaMemcpy limited, I assume bidirectional copies

BGQ-4-YOU

Real benefit from integrating 40GB/s network and 32MB cache on a chip

3 Petaflop/s *sustained* performance on half the Livermore system!