Bagel for BlueGenes

Peter Boyle

May 17, 2012

BlueGene/Q Compute chip

Floating point

- 16 compute cores per chip @ 1600 MHz
- Quad SIMD double/single precision FMAC
  $\Rightarrow$ 8 flops per clock per core $\Rightarrow$ hard to use due to SIMD constraints (quad real, paired complex) $\Rightarrow$ 204.8 Gflop/s peak per node
- 4 hardware threads per core - execution of different threads is interleaved to help keep pipelines full.

Memory

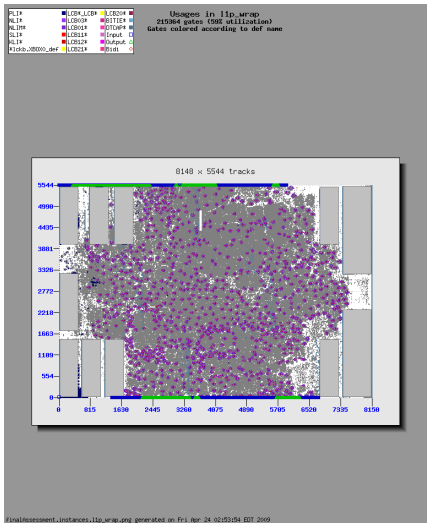
- 16KB L1 cache (shared between 4 threads) for each core (819GB/s across chip)
- 4KB L1p prefetch engine for each core
- 32MB L2 cache (563 GB/s peak read+write)
- 4/8/16 GB DDR (42GB/s peak)

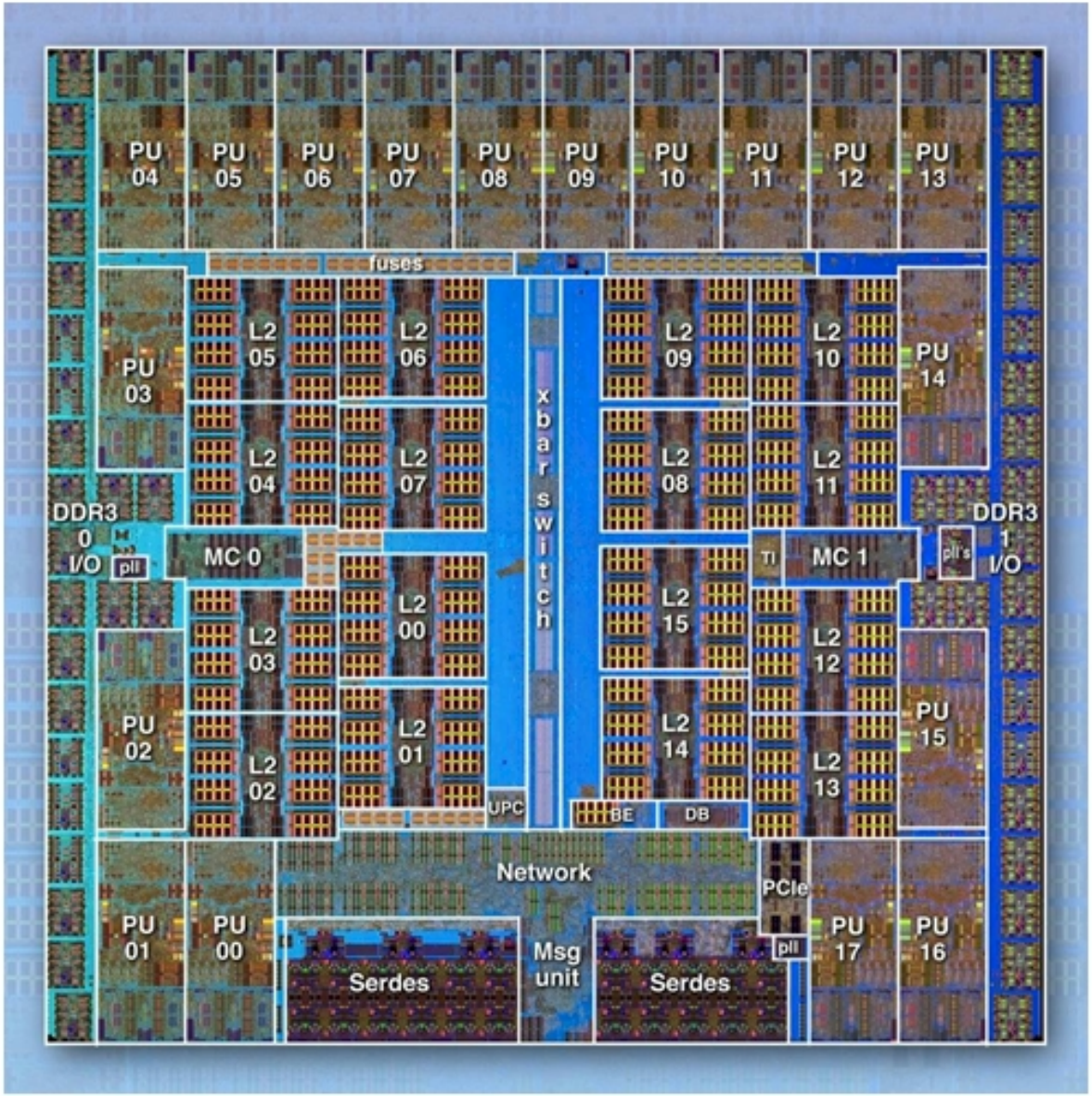
Note bandwidth drop off **Network**

- 5-d torus interconnect
- 2GB/s peak per link $\Rightarrow$ 40GB/s Send+Receive
- Similar partitioning to QCDOC possible – better MPI support needed
- 5th dimension remains local
- Copper within a midplane
- Optical torus globally

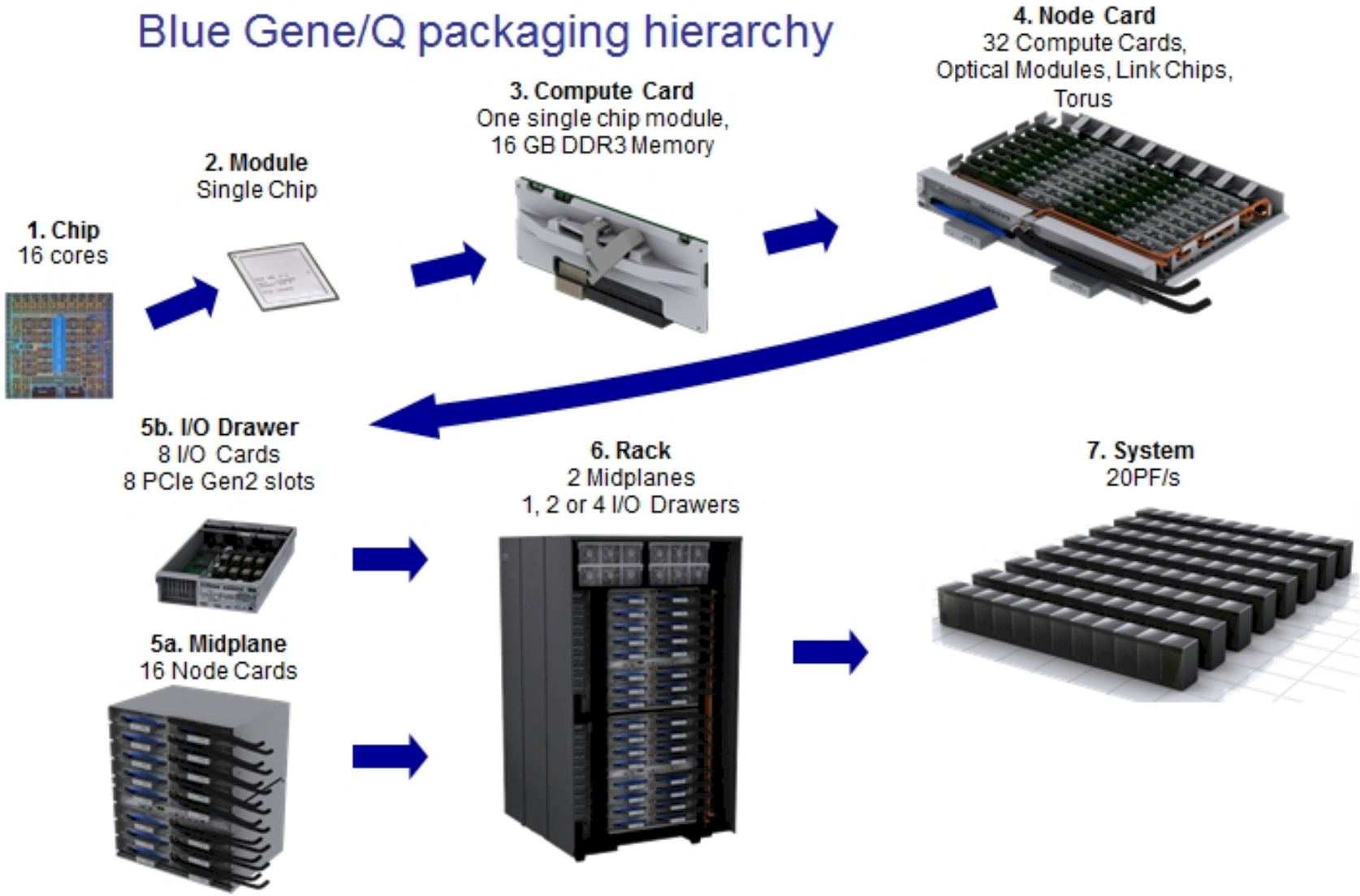
Eyesight test

Can you find this unit in this die photo?





System design



Simple performance analysis

FP/Memory/NetworkDesign balance (GFlop/s : GB/s :GB/s ratio)

- Ideal balance based on counting bytes & flop/s in SU(3)
  200:600:60
- BG/Q (high L2 hit rate) L2 resident
  200:600:40
- BG/Q (low L2 hit rate) DDR resident
  200:40:40

Simple analysis

- Machine is very fast from 32MB L2 cache
- Cache friendly programming is a must.
- 5d Domain Wall QCD reuse rates in sparse matrix can be very high with right loop ordering:
  - Gauge field reused $L_s \sim 16$ times at L1 level
  - Each input spinor reused $2 \times (N_d + 1) = 10$ times at L2 level
  - Each output spinor field takes 1320 flops to compute.
- The memory system has involved a significant *co-design* element from QCD
- Can map QCD efficiently to the 5d torus

There is a real benefit from integrating a network equivalent to 10 PCIe links on chip!

Bagel domain specific compiler

Bagel maintains a virtual RISC-like model, with pipeline scheduler and translation rules for various architectures.
3-7 $\times$ speed up over C++, and up to 65% of peak on key leaf routines
Has been a key component of co-design project with IBM
BFM is a library of physics kernels written to use bagel and supporting many forms of Wilson type lattice fermions.

- Wilson
- Twisted mass
- Domain Wall
- Generalised 5d Cayley / ContinuedFraction / Partial Fraction chiral fermions
- The continued fraction and partial fraction do not *yet* support preconditioning

Optimisation for BlueGene: SIMD

Transform internal data layout to *force* there to be SIMD parallelism

- Geometric decomposition is familiar in QCD: multiple nodes do *exactly* the same thing
- So why the hell do we get confused when $N_c = 3$ is hard to fit into SSE vectors?

Reorder to put a *logical* node index innermost to force SIMD parallelism:
complex Array [2][Nxyzt][Nsc] $\longrightarrow$ Array [Nxyzt][Nsc][2]
Each node loops over Nxyzt local sites, applying operations in spin color indices to *two* logical nodes worth of data.
Wasteful pack/unpack/permute operations managing SIMD lanes are automatically reduced to a single face
Generalises to longer SIMD vectors easily (Intel MIC?)

Optimisation for BlueGene: Memory system

- L2 store bandwidth is $\frac{1}{2}$ Load bandwidth
- Store payload 16 bytes. Load payload 128 bytes. Store transaction rate is high.
- Consequence: much higher penalty for stores than for loads.
- Forces a choice of loop ordering in collecting the terms in D_W

```
loop xyzt
   loop s
    loop mu
       Load neighbor 4-spinor
       spin-project
       if (mu<8)
         load gauge link
         SU3 multiply
       Accumulate result (12 regs)
     end loop mu
     store result
   end loop s
end loop xyzt
```

Pro: suppresses stores
reuses gauge links Ls times (L1)
reuses fermion field 10 times (L2)

Con: only 384 byte sequential accesses

Mitigation:
introduced L1p modes to
i) programme size of a spinor
ii) enable hints prefetch next spinor.
iii) Assist user mode line locking in L1.
Retain gauge link
while read $10 \times L_s$ fermions

Optimisation for BlueGene: Threading and communication

Threads: Use 64 threads per node, in a single MPI task

- Use fast L2 barriers for 1μs synchronisation within node
- Expose largest possible packet size to message passing system
- All threads can work on all tasks, even the collection of a single face.
  Not possible if geometrically associate threads with data.

Comms: SPI optimisation

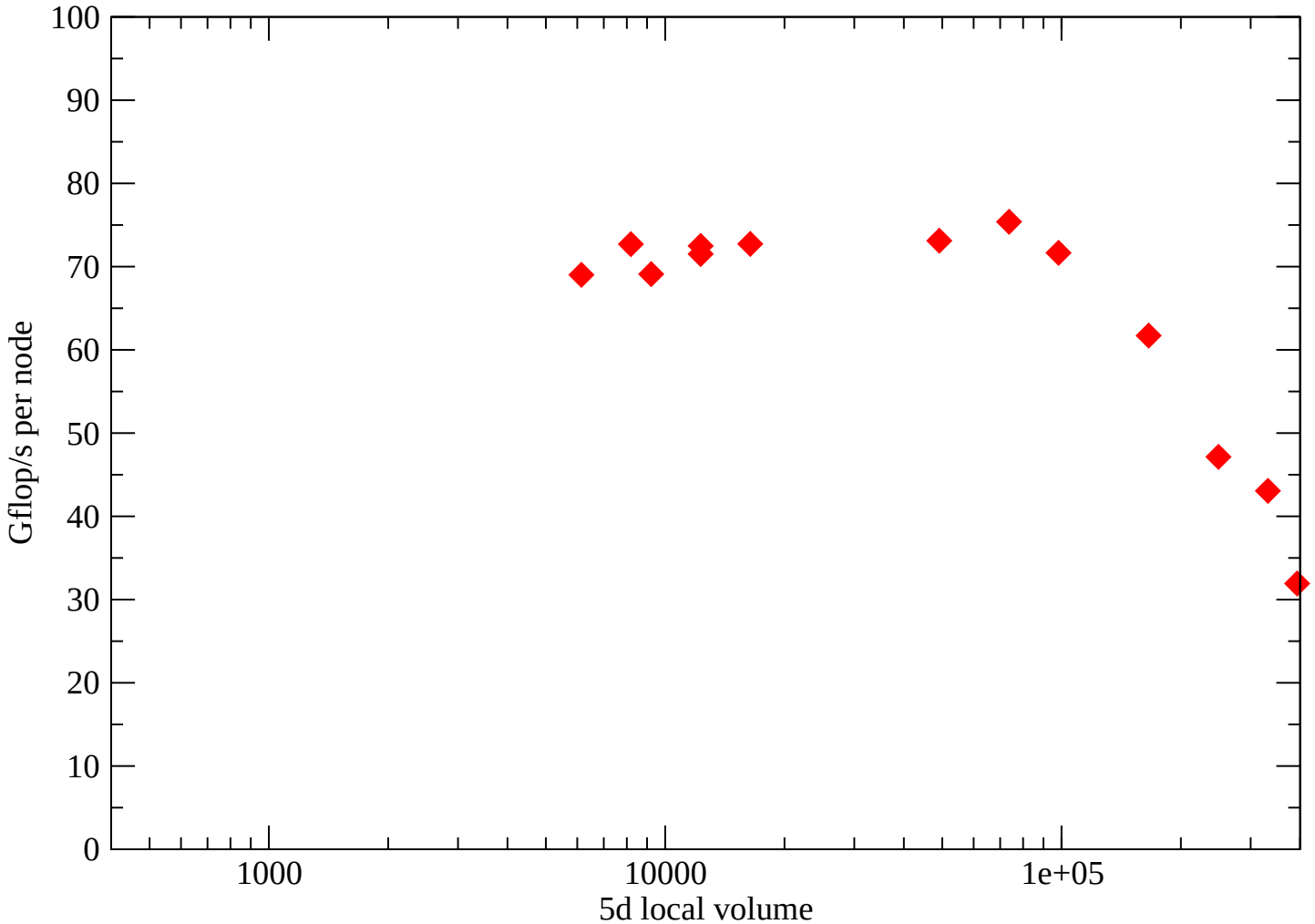
- Perform Dslash halo exchange by programming dma hardware
- Provided patch to QMP to make sure it's MPI and my SPI comms agree on layout
- Efficiently map 3d and 4d application torii to the 5d physical torus so that nearest neigbours ... are
- *-qmp-geom native*, *-qmp-geom native3d*
- Saturates link bandwidth on my communication tests with O(30GB/s).
- *Overlaps communications with computation* unlike MPI

Code output

```
bfmcommspi: Printing all sorts of ugly stuff to convince you this is hard
bfmcommspi: SPI device memory pool 67108864 bytes
bfmcommspi: allocated SPI device memory pool in virtual memory space 0x00000019c8000000 - 0x00000019cc000000
bfmcommspi: called KernelCreateMemoryRegion to pin this in contiguous physical region
bfmcommspi: Memory translation is as follows:
bfmcommspi: SPI device memory Base Physical Address 0000000005000000
bfmcommspi: SPI device memory Base Virtual Address 00000019c5000000
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 0
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 4
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 8
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 12
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 16
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 20
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 24
bfmcommspi: allocating Block Address Translation (BAT) id 0 Group 28
bfmcommspi: preallocating reception counters, buffers, descriptors in this device memory pool
bfmcommspi: Injection FIFO setup
bfmcommspi: Global Interrupt Barriers initialised
bfmcommspi: preallocated section of device heap state for reinitialisation
bfmcommspi: SPI setup is complete
bfmcommspi: End of scary messages
********************************
* QMP BlueGene/Q Native topology
********************************
* Torus: 4 x 4 x 4 x 4 x2
* SMP: 1 tasks per node
* MPI: 512 tasks
* 3d torus: Folding B and E dimensions, A and C/D
* Physical XxYxZxT = 4x4x4x8
* Logical XxYxZxT = 4x4x4x8
********************************
```

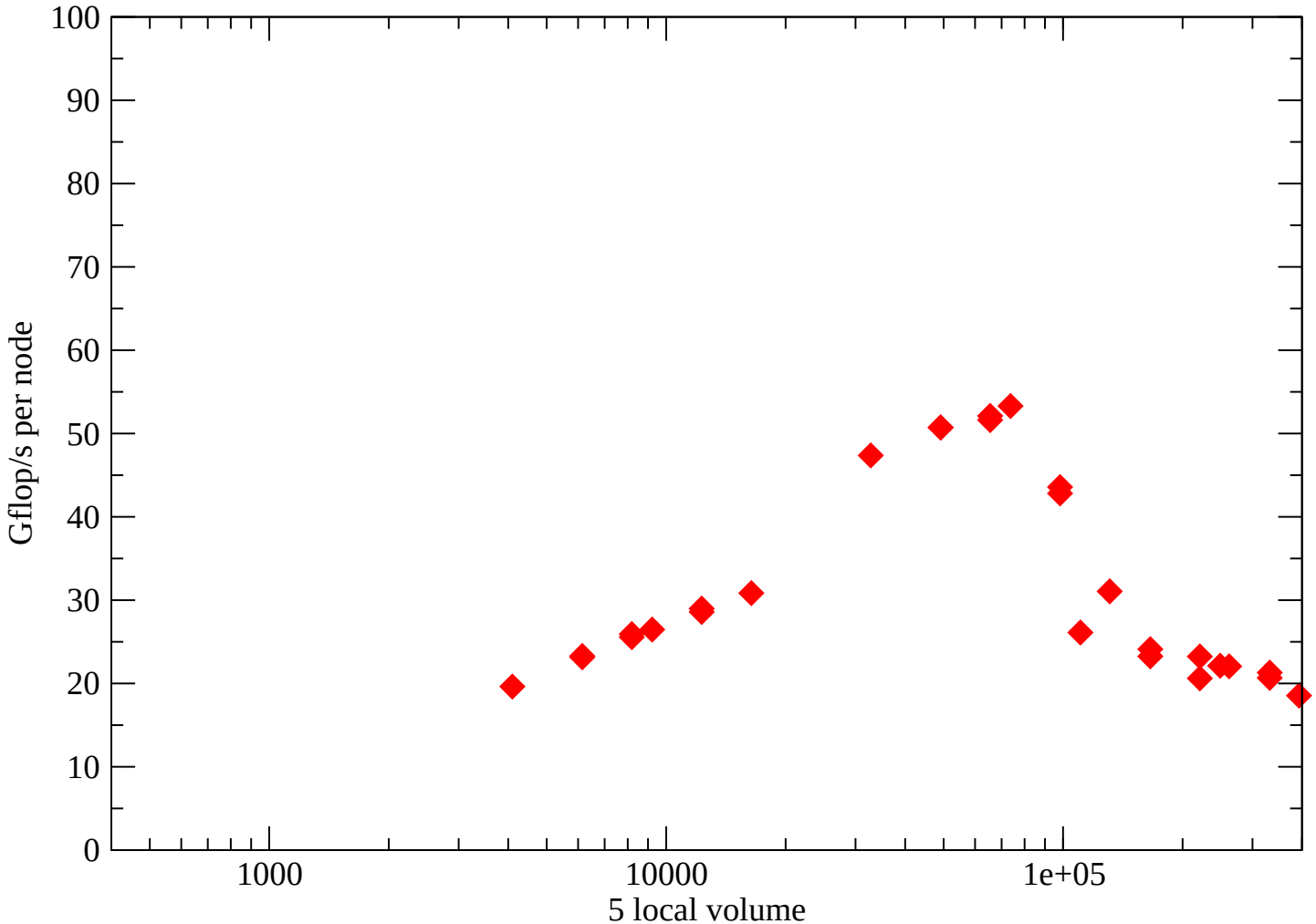
Performance : Double precision

D_W kernel on a single with no communication or linear algebra overhead



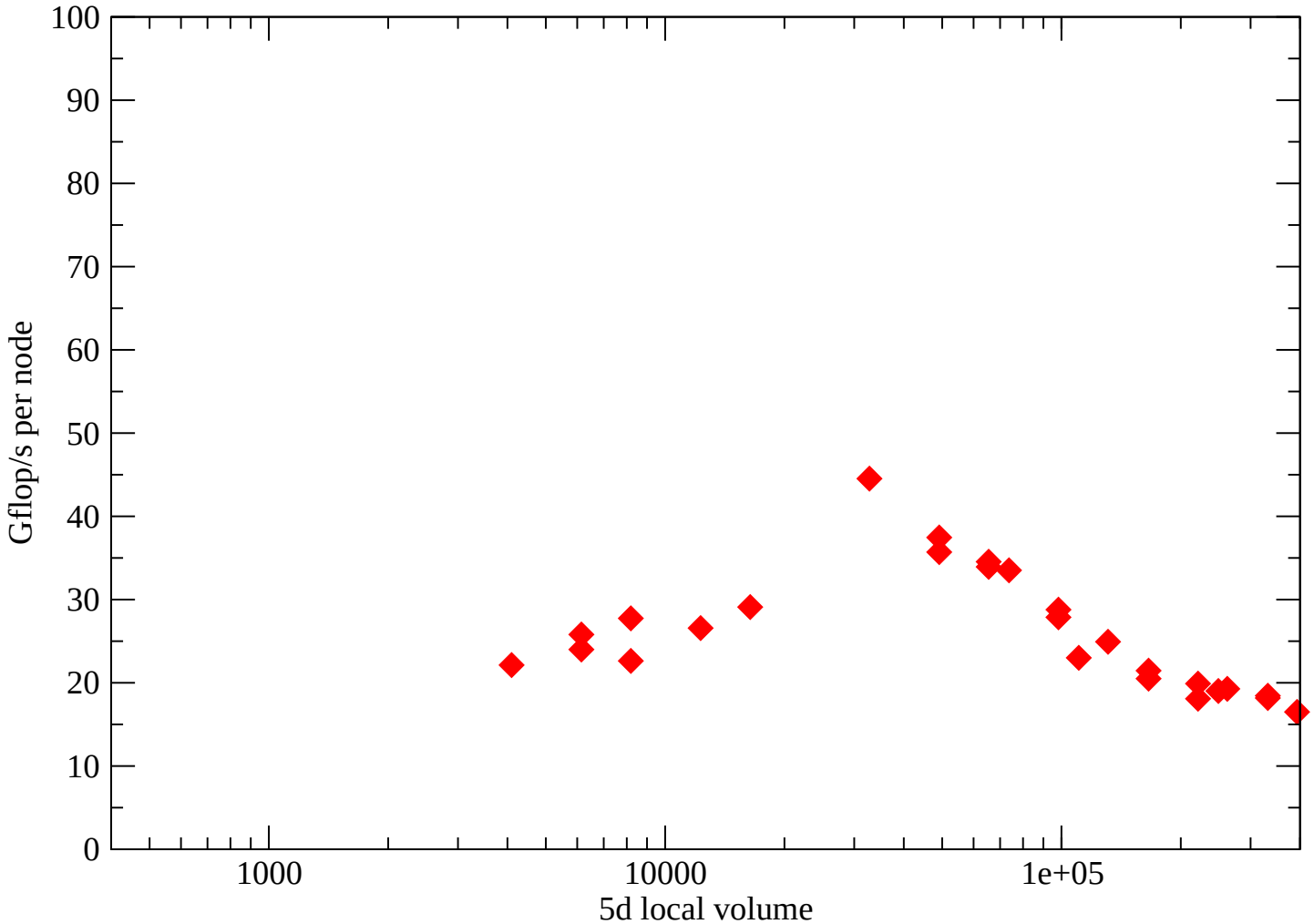
Performance : Double precision

D_W on 512 nodes, spread out in 4d, performance per node vs local volume



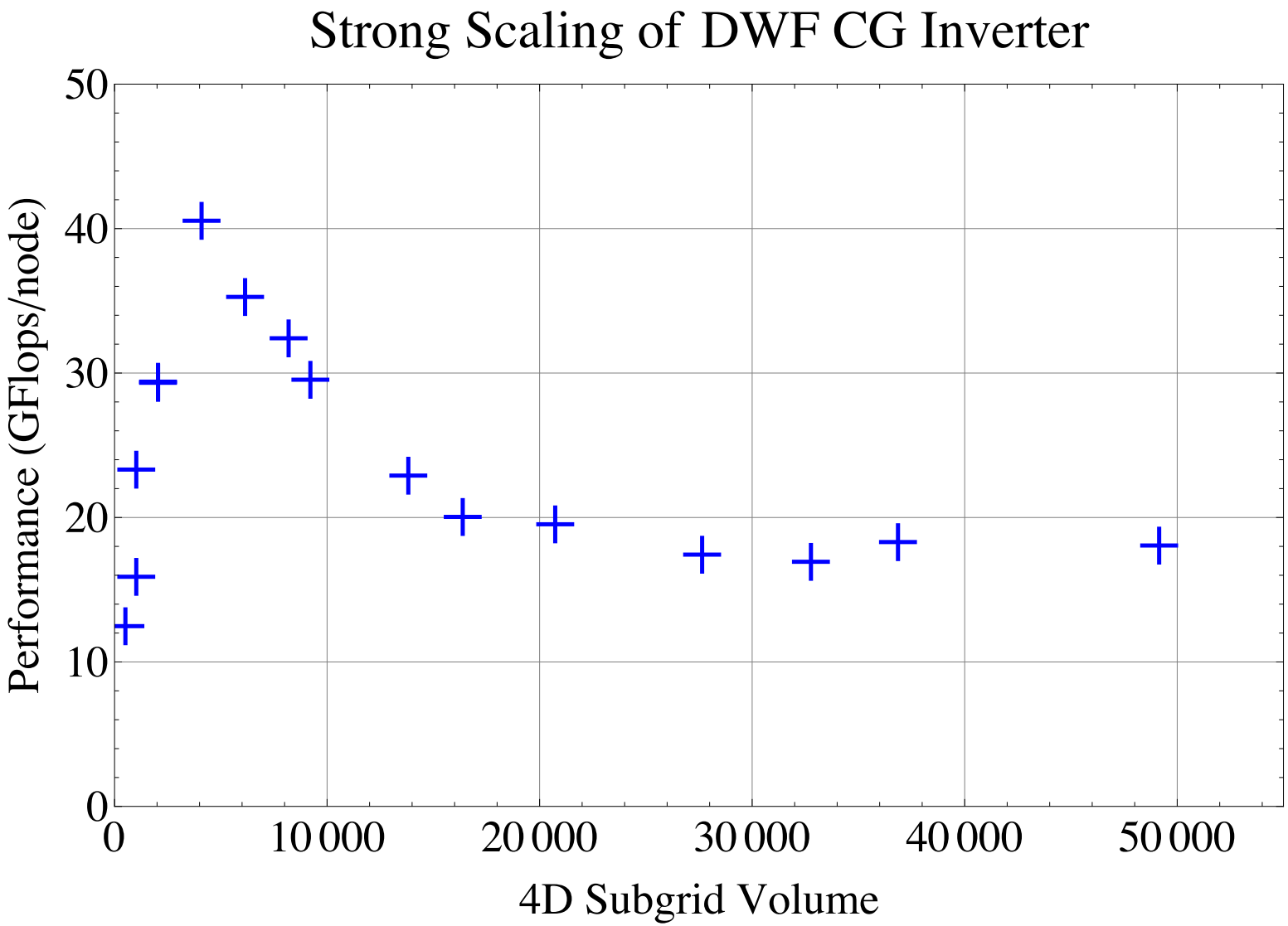
Performance : Double precision

DWF CGNE on 512 nodes, spread out in 4d, performance per node vs local volume



Performance : Double precision

DWF CGNE on 512 nodes, spread out in 4d, performance per node vs local volume



Scaling (thanks to LLNL for running this)

Double precision weak scaling on $8^4 \times 8$ local volume:

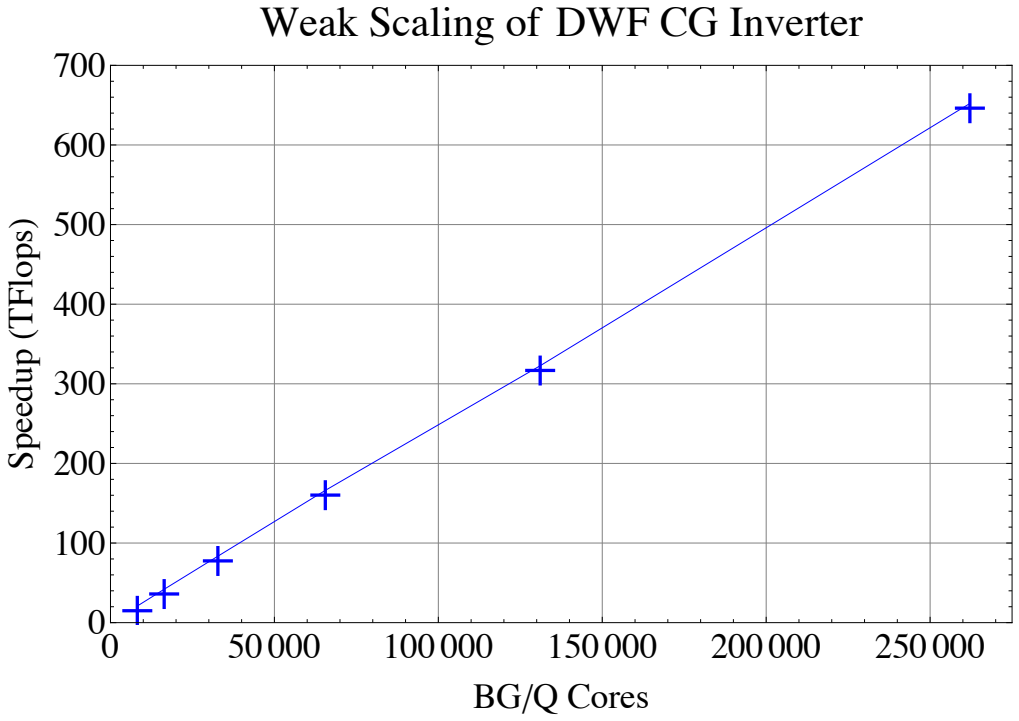


Fig. 2. Weak scaling with 8^4 lattice sites per node on up to 16K nodes of LLNL Sequoia BG/Q (the largest machine available to us as of May 2012). The speedup increases linearly with a rate of 40 Gflops/node.

Expect over 1Pflop/s sustained QCD performance in single precision when they next run my code.

Performance : Summary

Current status (subject to upwards revision)

DWF, CGNE	Single	up to 65Gflop/s per node
DWF, CGNE	Double	up to 45Gflop/s per node
Mobius, CGNE	Single	up to 35Gflop/s per node
Mobius, CGNE	Double	up to 27Gflop/s per node

- Recall QCD has imbalanced multiplies and adds
  Saturating the floating point pipeline would only deliver 160Gflop/s
- The above corresponds to around 28% efficiency in double, and 41% in single for the *whole* DWF conjugate gradient.

This is good... but I hope to do better in future...

Status: regression to Chroma

Unpreconditioned
BFM tests *general-5d-cg-unprec*, *general-5d-munprec* regresses to chroma for:

`HtCayleyZolo, HwCayleyZolo,`
`HmCayleyTanh, HtCayleyTanh, HwCayleyTanh,`
`HwContFracZolo, HwPartFracZolo`

Preconditioned
BFM tests *general-5d-cg-prec*, *general-5d-mprec* regresses to chroma for:

`HtCayleyZolo, HwCayleyZolo, HmCayleyTanh, HtCayleyTanh, HwCayleyTanh`

I am convinced the Chroma preconditioner for Cayley Zolo is a bad choice

I intend to release a library that has efficient implementation of all reasonable 5d chiral fermion approaches + Wilson + TwistedMass + Clover

Calling Example

```
g5dParams parms;
if ( solver == HtCayleyTanh )
Printf(‘‘Testing HtCayleyTanh aka DWF’’);
parms.ShamirCayleyTanh(mq,M5,Ls);
else if ( solver == HmCayleyTanh )
parms.ScaledShamirCayleyTanh(mq,M5,Ls,htscale);
Printf(‘‘Testing HmCayleyTanh Moebius’’);
else if ( solver == HwCayleyTanh )
parms.WilsonCayleyTanh(mq,M5,Ls,hwscale);
Printf(‘‘Testing HwCayleyTanh aka Borici’’);
else if ( solver == HtCayleyZolo )
parms.ShamirCayleyZolo(mq,M5,Ls,shamirlo,shamirhi);
Printf(‘‘Testing HtCayleyZolo’’);
else if ( solver == HwCayleyZolo )
parms.WilsonCayleyZolo(mq,M5,Ls,wilsonlo,wilsonhi);
Printf(‘‘Testing HwCayleyZolo aka Chiu’’);

bfmCayleyCG<double>(sol,src,u,PP,PA,PAc,PJ5,parms,1.0e-12,80000);
```

Outstanding work

- Cayley code has the 5th dim handled in cache unfriendly loop order
  Uses assembler, but more work to do
  (vector ops over whole 4d, 5d matrix loops outer most). (3 days work?)
- Smarter inner single/outer double solver (1 days work?) to deliver single performance
  Especially good for out of cache performance
- Cayley Zolotarev O(10x) more iterations than Cayley Tanh (random gauge $32^3 \times 64$)
  Better preconditioner for Zolotarev/Cayley (2 days work?)

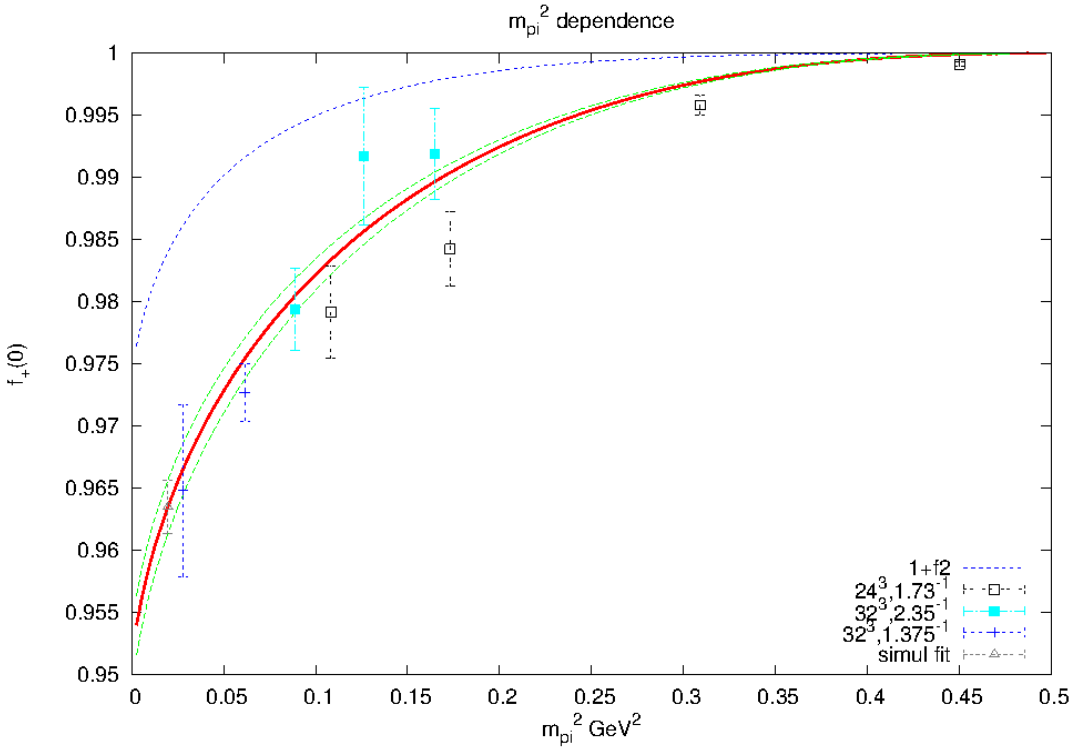
$$\begin{pmatrix} M_{ee} & M_eo \\ M_{oe} & M_oo \end{pmatrix} \rightarrow \begin{pmatrix} M_{ee} & 0 \\ 0 & M_oo \end{pmatrix} \begin{pmatrix} 1 & M_{ee}^{-1} M_eo \\ M_{oo}^{-1} M_{oe} & 1 \end{pmatrix}$$

- Precondition contfrac/part frac (2 days?).
  Unpreconditioned has no particular penalty for Zolatarev vs Tanh
  The Continued Fraction and Partial Fraction matrix forms are very easy to implement force term for H_W kernel; less so for H_T.
- Revisit prefetching strategy on FPGA and eliminate stalls
  Have a single node D_W code that got 107Gflop/s (or 65%) prior to my tweaks for comms

First results for K_{l3} with 180 MeV pion

Very preliminary (Karthee Sivalingam, James Zanotti, Andreas Juttner):

$$f_+(0) = 0.9645(35) \rightarrow f_+(0) = 0.9635(21)$$



Summary

- CPS+Bagel immediately able to run DWF RHMC very efficiently
- CPS+Bagel immediately able to run Mobius/Tanh RHMC pretty efficiently
- Bagel requires better preconditioner of work to run Mobius/Zolotarev
- Bagel requires preconditioning to seriously run Continued and Partial fraction 5d
- CPS requires force term to run Continued and Partial fraction 5d
- CPS requires work for smeared link force terms