Software Developments for Lattice QCD Calculations on Heterogenous Machines

Steven Gottlieb
Indiana University

Lattice QCD Computational Science Workshop
ORNL
April 29-30, 2013

Outline

- ✦ Background on the MILC Code
- ✦ Evolution of the MILC Code
  - Data structures and methods
  - MILC code
- ✦ Challenges of Exascale
- ✦ Challenges of Acceleration
- ✦ Personnel/Programming Expertise

Background on the MILC Code

- MILC collaboration is over 20 years old.
- Code has been evolving as hardware and software change.
- Original code based on ‘site’ structure that contains all the physical variables defined at each lattice site, including the gluon variables that originate at the site.
- As cache lines got wider, this became inefficient.
- Field major structure: single physical variable from successive sites are stored in succession
  - This rearrangement does not suffice for GPUs or Xeon Phi
- Major bottleneck in code is getting the data to the processor in a form it can use. (Prefetching)

MILC Code II

- Original MILC code developed before MPI was ubiquitous.
  - application specific gather and scatter operations
  - application specific global sums
  - native communication calls hidden from user
  - communication layer chosen at compilation time by selection of a single file containing all the communication routines:
    - com_mpi.c
    - com_vanilla.a
    - com_qmp.c ...
- Looping over sites done through macros, e.g.,
  - FORALLSITES(i,s) {....
  - avoids four dimensional indexing

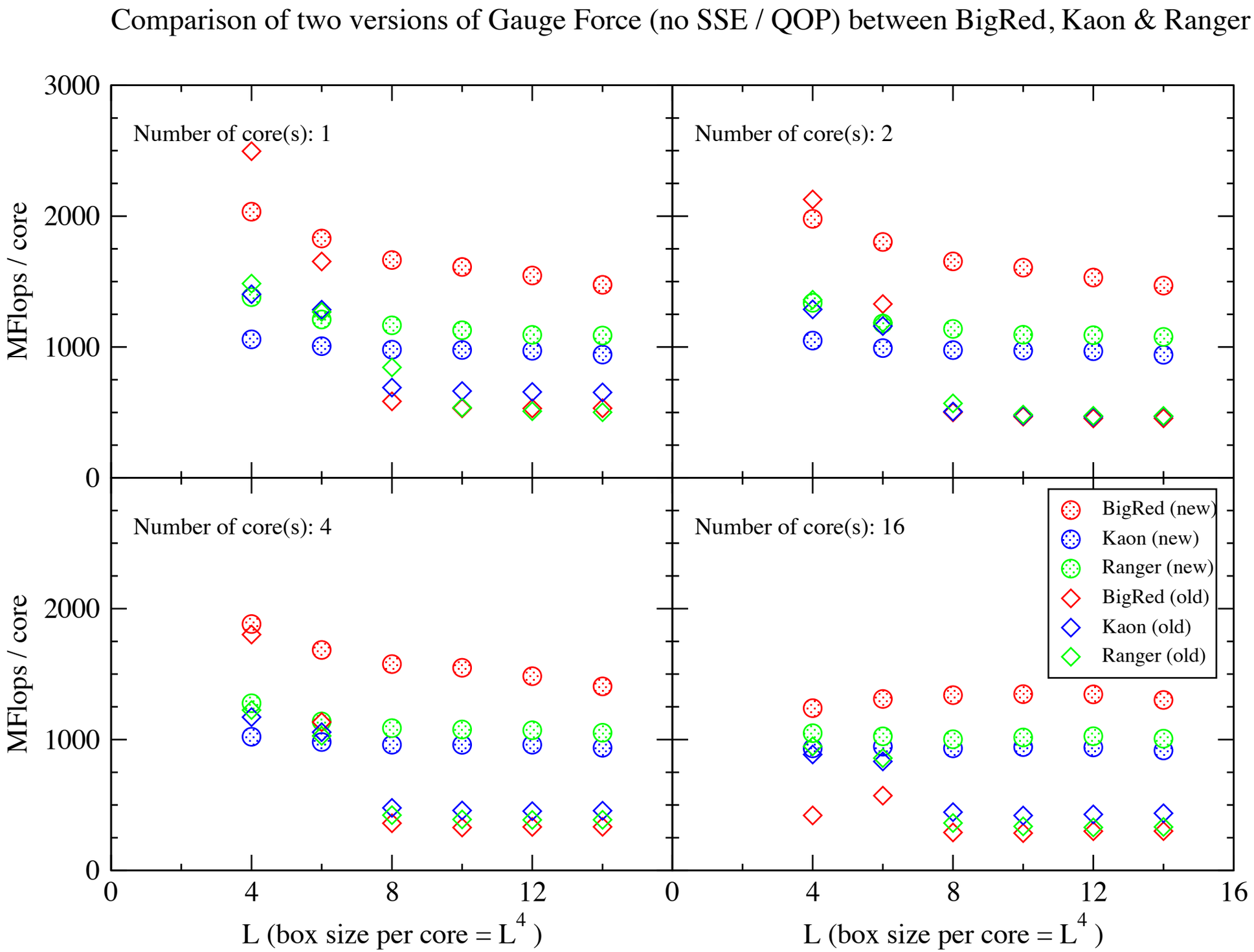
MILC Code III

- ✦ Neighbors accessed through pointers
- ✦ Basic operations on SU(3) matrices and vectors done through a library.
- ✦ Easy to optimize library for a particular CPU, e.g., one supporting SSE.
  - Application code looks the same, just link to the right library
- ✦ USQCD SciDAC software: MILC code can call the optimized routines.
- ✦ New algorithms/methods have been developed as needed, e.g.,
  - R-algorithm $\Longrightarrow$RHMD/RHMC
  - staggered$\Longrightarrow$asqtad$\Longrightarrow$HISQ

MILC Code IV

- ✦ Some work on reducing communication:
- ✦ Gauge force calculation requires frequent gathers from neighbors
  - augment the original MILC paradigm to pregather all off node values required for the entire routine rather than incremental gather of intermediate values
  - then the entire calculation can be done with on-node data
  - also much more cache friendly since then do all the work for a site before moving to next site
  - benchmarks of S. Basak's code on next page
- ✦ Domain decomposition for solvers (see DeTar, Clark,...)

Gauge Force Comparison



MILC Code V

- Can see more details in MILC entry in Encyclopedia of Parallel Computing, Ed. by D. Padua (Springer)
- Code is freely available at http://www.physics.utah.edu/~detar/milc/
- Has been used for benchmarking for supercomputer acquisitions by NSF & DOE
- Part of acceptance test for Blue Waters
- SPEC floating point and MPI benchmarks

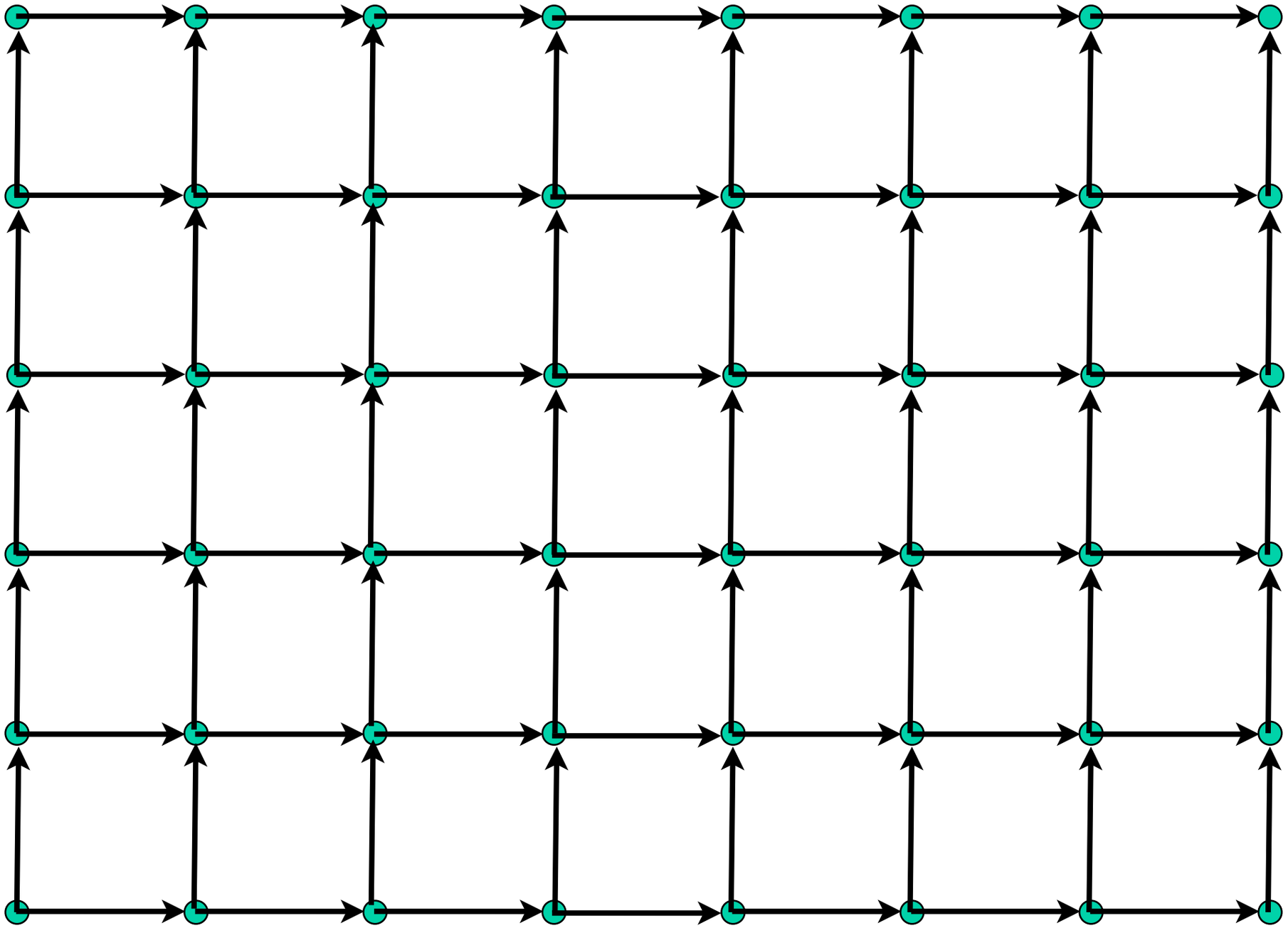
Data Structures

- Gluon field is an element of group SU(3), i.e., a 3×3 unitary matrix. However, 3×3 complex matrices (non-unitary) are also needed.
  - With compressed storage only 2 rows of unitary matrix are stored and third row is reconstructed. Reduces memory BW.
- Quark field has 'colors.' Different lattice formulations are used with various numbers of spin degrees of freedom.
  - Wilson or Clover quarks have 4 spinor components.
  - Kogut-Susskind or staggered quarks have only 1.
  - Domain wall quarks have 4 spinor components and an extra 5-th dimension

Data Structures

- Gluons live on the directed ‘links’ joining lattice sites.
  - At each point of the space-time grid there are 4 independent unitary matrices describing the gluon field.
- Unitary matrix: 3×3=9 complex numbers = 18 reals
  - float is 4 bytes, double is 8
  - with compressed storage, only 12 reals

- Quarks live on the lattice sites.
- Belong to fundamental representation of SU(3)
- Wilson quark: 4×3=12 complex numbers = 24 reals
- Staggered quark: 3 complex numbers = 6 reals



Green circles represent quarks; arrows gauge fields. This is just a 2-dimensional lattice (grid). Space time requires 4 dimensions.

Computational Intensity

- Getting data to processor is key
  - 320 GB/s peak Xeon Phi memory speed; 250 GB/s Nvidia K20X
- Single precision flop
  - 8 bytes input; 4 bytes output
- SU(3) matrix times vector
  - 36 multiplies + 30 adds = 66 flops
  - (18+6)×4=96 bytes input, 24 bytes output
  - 1.45 bytes/flop input; 0.36 bytes/flop output
- Naive staggered Dslash
  - 8*66+7*6=570 flops
  - 768 bytes input, 24 bytes output
  - 1.35 bytes/flop input; 0.04 bytes/flop output

Computational Intensity II

- With 320 GB/s bandwidth to main memory, and 1.35 bytes/flop computational intensity, maximum performance is 237 GF/s.
  - with 250 GB/s, maximum performance is 185 GF/s.
- Can cache reuse reduce required bandwidth?
- Can data compression of SU(3) matrices improve performance?
  - only two rows of SU(3) matrix must be stored as third row can be reconstructed
  - does not work for fat links

Challenges of Exascale

- How will we express parallelism at the level of 100K, 1 million or 1 billion ‘threads’?
  - a $256^3 \times 512$ configuration has 8.6 billion grid points
- How many levels of parallelism will be required?
  - MPI
  - OpenMP/pthreads
  - GPU/vectorization
  - something new
- Will the software be smart enough to do this for us?
  - See next slide from Jeffrey Vetter at 2011 Keeneland tutorial.

Echelon: Extreme-scale Compute Hierarchies with Efficient Locality-Optimized Nodes

Main Objectives

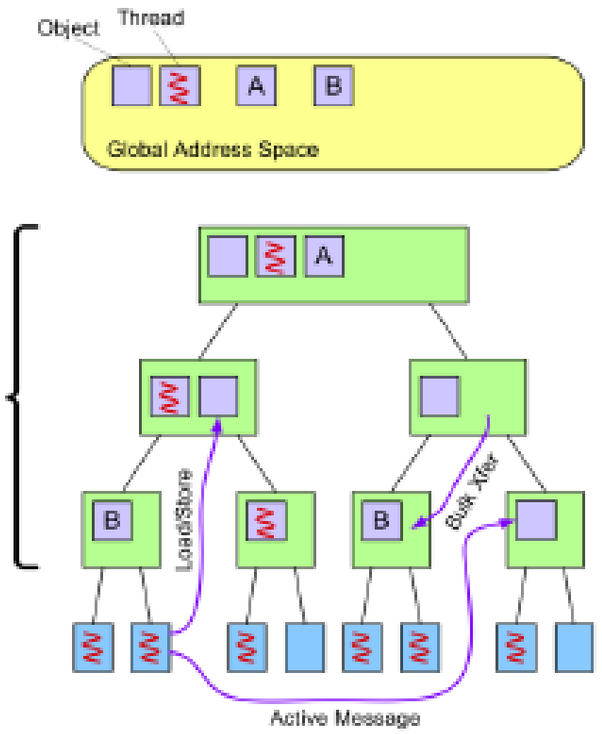
- Two orders of magnitude increase in application execution energy efficiency over today's CPU systems.
- Improve programmer productivity so that the time required to write a parallel program achieving a large fraction of peak efficiency is comparable to the time required to write a serial program today.
- Strong scaling for many applications to tens of millions of threads in UHPC system.
- High application mean-time to interrupt (AMTTI) with low overhead; matched to application needs.
- Machines resilient to attack; enables reliable software.

Key Innovations

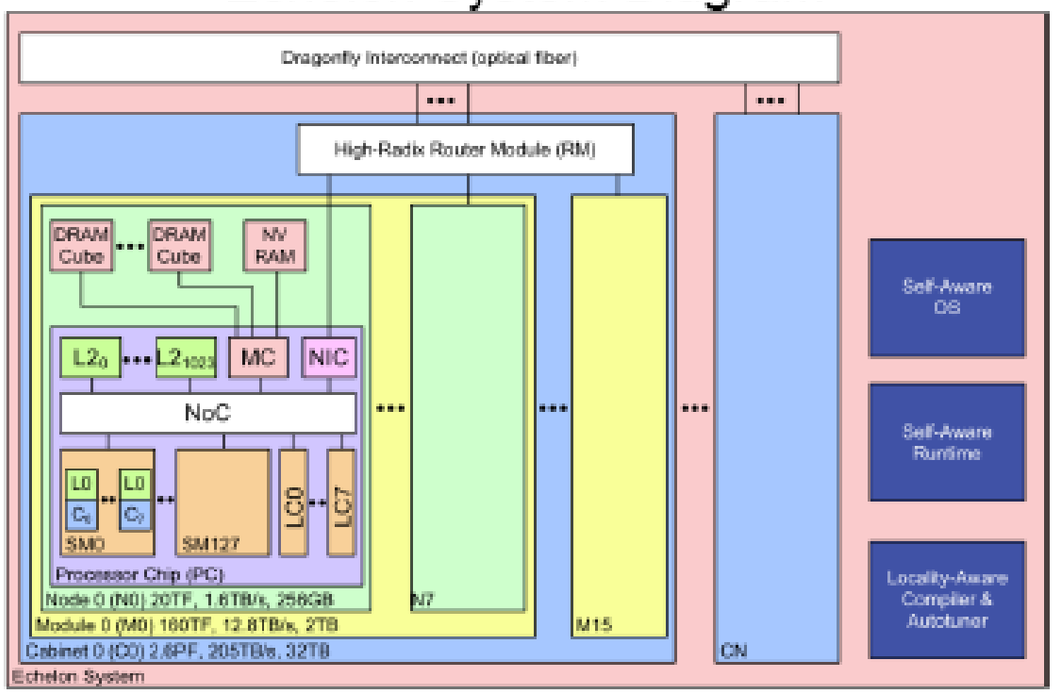
- Programming systems that express concurrency/locality abstractly; autotuning for hardware mapping.
- Self-aware runtime reacts to changes in environment, workload (load-balance), fault states.
- Fine-grained, energy-optimized, multithreaded throughput cores + latency-optimized cores.
- Software selective memory hierarchy configuration; selective coherence for non-critical data.
- HW/SW cooperative resilience for energy- and performance-efficient fault protection.
- Guarded pointers for memory safety.
- Low-power, high speed communication circuits.

Echelon Execution Model

- Programmability: global address space, abstract memory hierarchy, autotuning; runtime task placement/scheduling.
- Efficiency: Active messages, bulk transfer.
- Dependability: software selective redundancy, hardware accelerated guarded pointers.



Echelon System Diagram



Georgia Tech Stanford UC-Berkeley UPenn UT-Austin U. Utah Tennessee Lockheed Martin

Echelon: Extreme-scale Compute Hierarchies with Efficient Locality-Optimized Nodes

Main Objectives

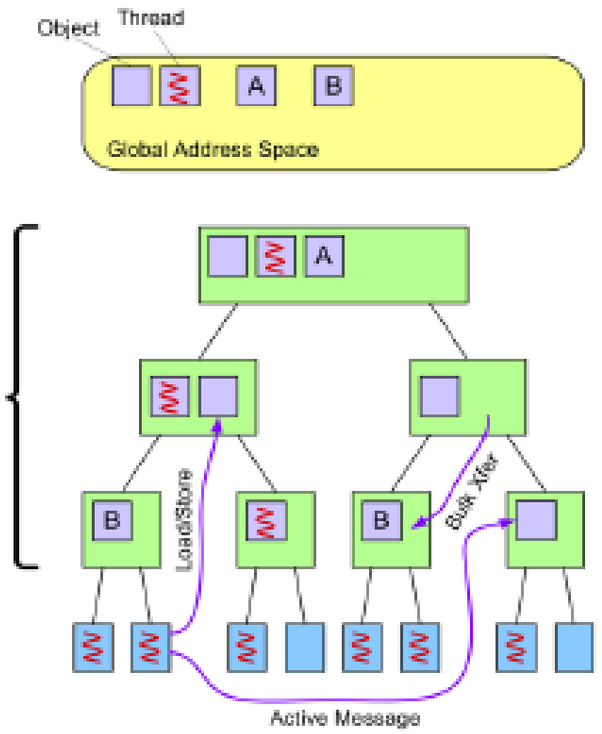
- Two orders of magnitude increase in application execution energy efficiency over today's CPU systems.
- Improve programmer productivity so that the time required to write a parallel program achieving a large fraction of peak efficiency is comparable to the time required to write a serial program today.
- Strong scaling for many applications to tens of millions of threads in UHPC system.
- High application mean-time to interrupt (AMTTI) with low overhead; matched to application needs.
- Machines resilient to attack; enables reliable software.

Key Innovations

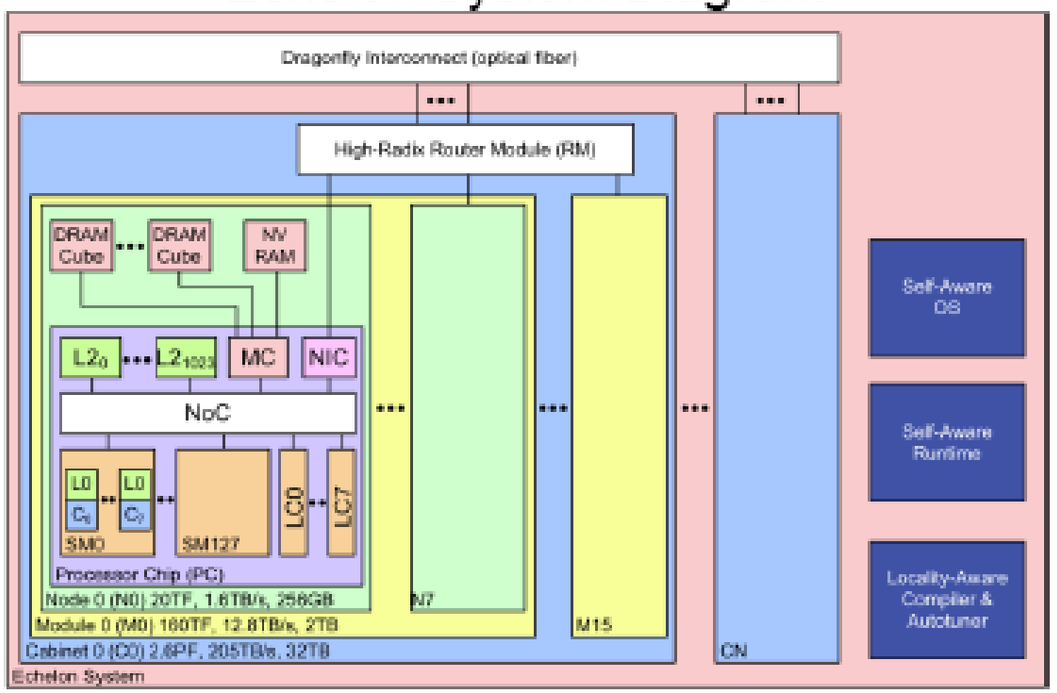
- Programming systems that express concurrency/locality abstractly; autotuning for hardware mapping.
- Self-aware runtime reacts to changes in environment, workload (load-balance), fault states.
- Fine-grained, energy-optimized, multithreaded throughput cores + latency-optimized cores.
- Software selective memory hierarchy configuration; selective coherence for non-critical data.
- HW/SW cooperative resilience for energy- and performance-efficient fault protection.
- Guarded pointers for memory safety.
- Low-power, high speed communication circuits.

Echelon Execution Model

- Programmability: global address space, abstract memory hierarchy, autotuning; runtime task placement/scheduling.
- Efficiency: Active messages, bulk transfer.
- Dependability: software selective redundancy, hardware accelerated guarded pointers.



Echelon System Diagram



Georgia Tech Stanford UC-Berkeley UPenn UT-Austin U. Utah Tennessee Lockheed Martin

Echelon: Extreme-scale Compute Hierarchies with Efficient Locality-Optimized Nodes

Main Objectives

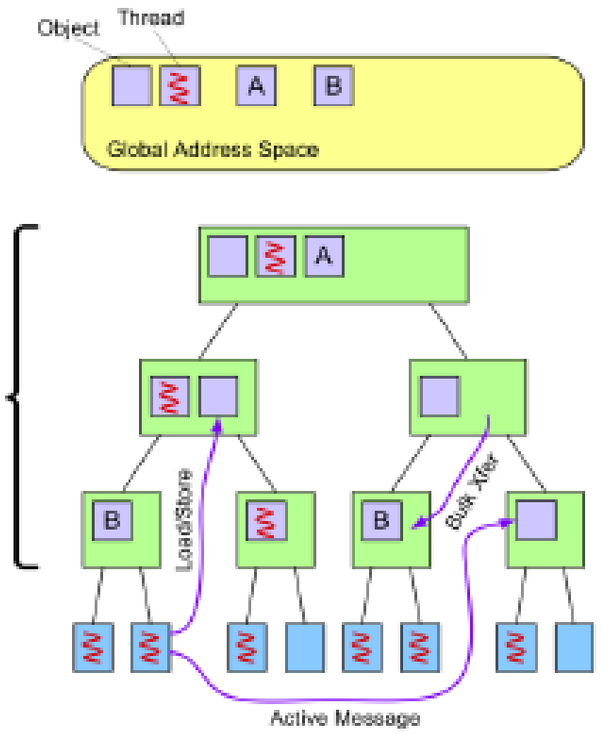
- Two orders of magnitude increase in application execution energy efficiency over today's CPU systems.
- Improve programmer productivity so that the time required to write a parallel program achieving a large fraction of peak efficiency is comparable to the time required to write a serial program today.
- Strong scaling for many applications to tens of millions of threads in UHPC system.
- High application mean-time to interrupt (AMTTI) with low overhead; matched to application needs.
- Machines resilient to attack; enables reliable software.

Key Innovations

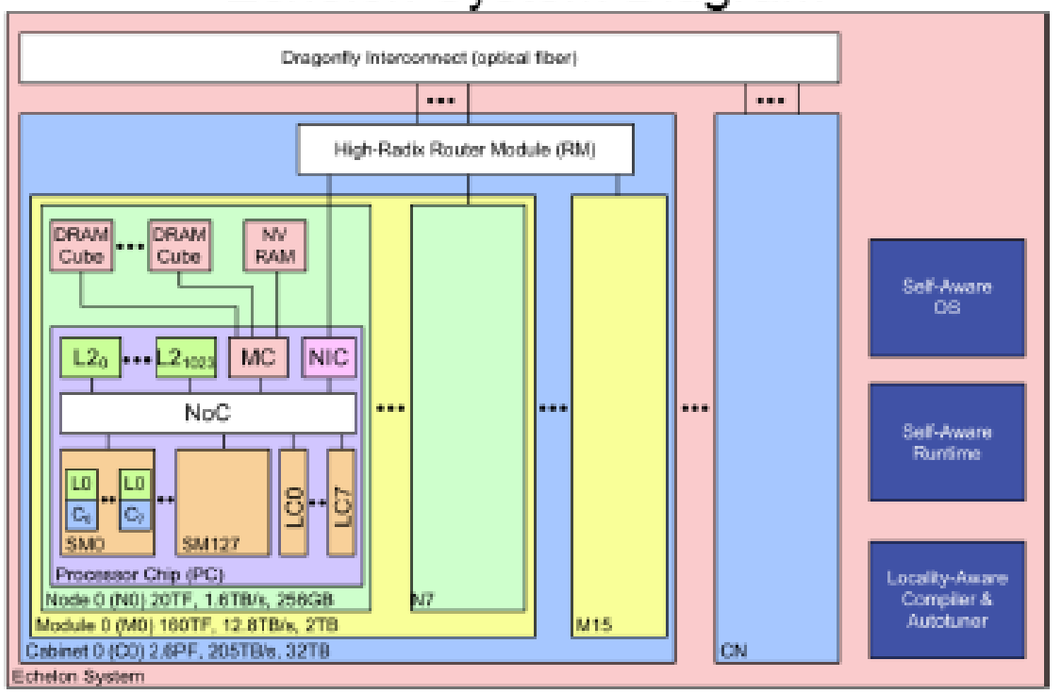
- Programming systems that express concurrency/locality abstractly; autotuning for hardware mapping.
- Self-aware runtime reacts to changes in environment, workload (load-balance), fault states.
- Fine-grained, energy-optimized, multithreaded throughput cores + latency-optimized cores.
- Software selective memory hierarchy configuration; selective coherence for non-critical data.
- HW/SW cooperative resilience for energy- and performance-efficient fault protection.
- Guarded pointers for memory safety.
- Low-power, high speed communication circuits.

Echelon Execution Model

- Programmability: global address space, abstract memory hierarchy, autotuning; runtime task placement/scheduling.
- Efficiency: Active messages, bulk transfer.
- Dependability: software selective redundancy, hardware accelerated guarded pointers.



Echelon System Diagram



Georgia Tech Stanford UC-Berkeley UPenn UT-Austin U. Utah Tennessee Lockheed Martin

Some Comments from B. Dally

- **HPCwire**: What do you see as the biggest challenges to reaching exascale?
- **Dally**: Energy efficiency and programmability are the two biggest challenges.
- For energy, we will need to improve from where we are with the NVIDIA-Kepler-based Titan machine at Oak Ridge National Laboratory in Tennessee, which is about 2GFLOPS/Watt (500pJ/FLOP) to 50GFLOPS/Watt (20pJ/FLOP), a 25x improvement in efficiency while at the same time increasing scale - which tends to reduce efficiency. Of this 25x improvement we expect to get only a factor of 2x to 4x from improved semiconductor process technology.
- As I described before, we are optimistic that we can meet this challenge through a number of research advances in circuits, architecture and software.

Dally (continued)

- Making it easy to program a machine that requires 10 billion threads to use at full capacity is also a challenge. While a backward compatible path will be provided to allow existing MPI codes to run, MPI plus C++ or Fortran is not a productive programming environment for a machine of this scale. We need to move toward higher-level programming models where the programmer describes the algorithm with all available parallelism and locality exposed, and tools automate much of the process of efficiently mapping and tuning the program to a particular target machine.
- A number of research projects are underway to develop more productive programming systems - and most importantly the tools that will permit automated mapping and tuning.
- Changing a large code base, however, is a very slow process, so we need to start moving on this now. As with energy efficiency, progress will be slowed without government funding.

Computing in Science & Engineering

- November/December issue co-edited by Thomas Sterling and me is devoted to exascale computing
  - Hardware: Drogge & Shalf
  - Programming Model: Gropp & Snir
  - Applications: Harrison & Heroux
  - System challenges: Beckman & Sterling
  - Tools: Dongarra et al.
- At an editorial board meeting on Friday, Barry Schneider (NSF) said people keep asking him how are they supposed to prepare their codes for the next computers
  - We can't completely rewrite our codes every three or four years.
  - Doug Post (DOD) reports that codes can last for 20 years and can be written for portability. (I want to hear the details.)

Other Exascale Issues

- ✦ Reliability/Resiliance
- ✦ Jitter from OS or interference of other jobs
  - already seeing this on current machines
- ✦ Performance modeling
  - We are very flop oriented; however, we are not the full employment program for floating point processors. Faster solution with lower flop rate is better. Goal is to get the physics done and get off the computer.
  - Increasingly expressing performance in terms of memory bandwidth.
  - Maybe we need an energy performance model.
  - What is the real cost: hardware, electricity?

Challenges of Acceleration

- We are pretty eager to chase the next new thing...
- First there was the Cell BE
  - I worked on that with NCSA
  - no long term future (Roadrunner at LANL)
- GPU work started with CUDA
  - details in previous talk and on next slide
- Xeon Phi
  - graduate student working on BEACON project
  - this port is much less developed than GPU
  - next talk by Balint Joò covers work with Intel
- What is the next new thing, and will what we have already learned be of any value?

GPU porting effort

- ✦ GPU effort started at Boston University in 2008
- ✦ Effort is ongoing:
  - Not all of the code has been ported
  - New algorithms required continued development (GPU algorithms can require implementation on CPU side)
  - New models, e.g., Kepler
  - Key developers have gone to NVIDIA and Google

- ✦ Of course, Xeon Phi development is comparatively in its infancy

Xeon Phi

- The processor formerly known as MIC (Many Integrated Cores)
- Currently, ≈60 cores on a single chip
- Most of the floating point power comes from a vector floating point unit that can do 16 single precision or 8 double precision ops using 512-bit registers
- This is not your father's vector processor. (Note my father was named Seymour, but not Cray.)
  - There is no gather-scatter or indirection for the floating point unit, i.e., it is strictly SIMD, so operands must all be aligned in succession.
- Peak speed: ≈ 1 Tflop/s

Approaches to Porting

- A plethora of parallel programming possibilities
  - native vs. offload
  - MPI
  - OpenMP
  - hybrid MPI/OpenMP
  - hybrid host cpu/Xeon Phi

- In any case, to make good use of Phi, one must vectorize the code which requires a complete rewrite.
- This is not a chip on which an evolutionary approach is likely to succeed.

Vectorization

- Without vectorizing the code on Phi, we are giving up a large factor (8 to 16) in peak performance.
- Balint Joò at Jefferson Lab working in close conjunction with Intel to port Wilson quark code to Phi
  - Layout is key to vectorization (as it was for GPUs)
- MILC code (elements for single site stored together):
  - typedef struct { fcomplex e[3][3]; } fsu3_matrix;
  - typedef struct { fcomplex c[3]; } fsu3_vector;
- Phi code (vec or soa sites combined):
  - typedef float SU3MatrixBlock[8][3][3][2][vec];
  - typedef float SpinorBlock[3][2][soa];
- vec is vector length, soa is structure of arrays

Architectural Balance

- Current accelerators don't seem to have the architectural balance we are used to.
  - Much more floating point power, but off accelerator bandwidth is often worse than for the cpu/node
  - This can work if the problem size is large enough since required bandwidth usually scales like 1/L (L^4 local grid)
  - But being able to reduce local size L by a factor of 2, lets use use $2^4 = 16$ as much hardware, with great increase in performance.
- When multiple accelerators are placed in a node this problem can be exacerbated.
  - communication avoiding algorithms need more attention

Personnel

- ✦ All this porting requires expert personnel and assistance
  - ORNL's experts can be a great help to us
  - We need training for graduate students and postdocs
  - We need to be informed of future developments
  - Have signed many NDAs where nothing much was disclosed
- ✦ Concerned about career paths
  - Many of the MILC developers are getting old
  - Several of the young people who have development experience are not yet in stable jobs
  - GPU developers, such as Clark and Babich have left for NVIDIA
  - Shi left for Google
  - For the Xeon Phi work, Intel staff is actively engaged (but I don't think they will leave Intel for our world)

Conclusion

- I have been a proud user of ORNL since the Grand Challenge Days of the Intel Paragon.
- Recently, Jaguar, Kraken, Keeneland, BEACON have all been of great use for lattice QCD.
- However, the road ahead may prove more challenging and even closer engagement will be required so that we can accomplish our scientific goals and demonstrate performance, efficiency and scaling to which we are used.
- I look forward to working with ORNL experts to rise to the challenge of exascale computing.