How HPC Hardware and Software are Evolving Towards Exascale

Kathy Yelick
Associate Laboratory Director and NERSC Director
Lawrence Berkeley National Laboratory

EECS Professor, UC Berkeley
NERSC Overview

NERSC represents science needs
- Over 4000 users, 500 projects, 500 code instances
- Over 1,600 publications in 2009
- Time is used by university researchers (65%), DOE Labs (25%) and others

Petaflop Hopper system, late 2010
- High application performance
- Nodes: 2 12-core AMD processors
- Low latency Gemini interconnect
How and When to Move Users

Want to avoid two paradigm disruptions on road to Exa-scale
Exascale is about Energy Efficient Computing

At $1M per MW, energy costs are substantial

- 1 petaflop in 2010 will use 3 MW
- 1 exaflop in 2018 at 200 MW with “usual” scaling
- 1 exaflop in 2018 at 20 MW is target
Challenges to Exascale

1) System power is the primary constraint
2) Concurrency (1000x today)
3) Memory bandwidth and capacity are not keeping pace
4) Processor architecture is an open question
5) Programming model heroic compilers will not hide this
6) Algorithms need to minimize data movement, not flops
7) I/O bandwidth unlikely to keep pace with machine speed
8) Reliability and resiliency will be critical at this scale
9) Bisection bandwidth limited by cost and energy

Unlike the last 20 years most of these (1-7) are equally important across scales, e.g., 100 10-PF machines
Anticipating and Influencing the Future

Hardware Design
Moore’s Law Continues, but Only with Added Concurrency

Past: 1000x performance increase was 40x clock speed and 25x concurrency

Future: All in added concurrency, include new on-chip concurrency
Where does the Energy (and Time) Go?

Counting flops is irrelevant, only data movement matters.
Case for Lightweight Core and Heterogeneity

Lightweight (thin) cores improve energy efficiency

Ubiquitous programming model of today (MPI) will not work within a processor chip
Technology trends against a constant or increasing memory per core

- Memory density is doubling every three years; processor logic is every two
- Memory costs are dropping gradually compared to logic costs

Question: Can you double concurrency without doubling memory?
Are GPUs the Future?
(Includes Cell and GPU)

1.7x speedup versus optimized Nehalem (C2050 w/ECC)


Cache-based
- Gainestown
- Barcelona
- Victoria Falls

Local store-based
- Cell Blade
- GTX280
- GTX280-Host
The Roofline Performance Model

**Generic Machine**

- The flat room is determined by arithmetic peak and instruction mix.
- The sloped part of the roof is determined by peak DRAM bandwidth (STREAM).
- X-axis is the computational intensity of your computation.

Diagram shows calibrated achievable peak performance with various factors:
- Peak BW
- W/out SW prefetch
- W/out NUMA
- W/out SIMD
- W/out ILP
- Mul / add imbalance
- Peak DP

Y-axis: Attainable Gflop/s
X-axis: Actual flop:byte ratio
Relative Performance Expectations

Fermi & Nehalem Roofline

- Double-precision peak
- 6.7x peak
- 2.2x Measured BW
- 1.7x 7-point Stencil

Attainable Gflop/s vs. Arithmetic Intensity
Relative Performance Across Kernels

Xeon X5550 (Nehalem)

NVIDIA C2050 (Fermi)
What Heterogeneity Means to Me

• Case for heterogeneity
  – Many small cores are needed for energy efficiency and power density; could have their own PC or use a wide SIMD
  – Need one fat core (at least) for running the OS

• Local store, explicitly managed memory hierarchy
  – More efficient (get only what you need) and simpler to implement in hardware

• Co-Processor interface between CPU and Accelerator
  – Market: GPUs are separate chips for specific domains
  – Control: Why are the minority CPUs in charge?
  – Communication: The bus is a significant bottleneck.
  – Do we really have to do this? Isn’t parallel programming hard enough
Open Problems in Software

• Goal: performance through parallelism
• Locality is equally important
• Heroic compilers unlikely solution:
• Need better programming models that:
  – Abstract machine variations
  – Provide for control over what is important
• Data movement ("communication") dominates running time and power
What’s Wrong with Flat MPI?

• We can run 1 MPI process per core
  – This works now for Quad-Core on Franklin
• How long will it continue working?
• What is the problem?
  – Latency: some copying required by semantics
  – Memory utilization: partitioning data for separate address space requires some replication
    • How big is your per core subgrid? At 10x10x10, over 1/2 of the points are surface points, probably replicated
  – Memory bandwidth: extra state means extra bandwidth
  – Weak scaling will not save us -- not enough memory per core
  – Heterogeneity: MPI per CUDA thread-block?
• This means a “new” model for most NERSC users
Conclusions so far:

- Mixed OpenMP/MPI saves significant memory
- Running time impact varies with application
- 1 MPI process per socket is often good

Run on Hopper next:

- 12 vs 6 cores per socket
- Gemini vs. Seastar

= OpenMP thread parallelism
Partitioned Global Address Space Languages

**Global address space**: thread may directly read/write remote data

**Partitioned**: data is designated as local or global

- No less scalable than message passing
- Permits sharing, unlike message passing
- One-sided communication: never say “receive”
- Affinity control for shared and distributed memory
Autotuners To Handle Machine Complexity

- **OSKI**: Optimized Sparse Kernel Interface
- Optimized for: size, machine, and matrix structure
- *Functional portability* from C (except for Cell/GPUs)
  - *Performance portability* from install time search and model evaluation at runtime
  - Later tuning, less opaque interface

See theses from Im, Vuduc, Williams, and Jain
Consider Sparse Iterative Methods

- Nearest neighbor communication on a mesh
- Dominated by time to read matrix (edges) from DRAM
- And (small) communication and global synchronization events at each step

Can we lower data movement costs?

- Take $k$ steps “at once” with one matrix read from DRAM and one communication phase
  - Parallel implementation
    $O(\log p)$ messages vs. $O(k \log p)$
  - Serial implementation
    $O(1)$ moves of data moves vs. $O(k)$

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin
Communication-Avoiding Algorithms

• Sparse Iterative (Krylov Subpace) Methods
  – Nearest neighbor communication on a mesh
  – Dominated by time to read matrix (edges) from DRAM
  – And (small) communication and global synchronization events at each step

• Can we lower data movement costs?
  – Take $k$ steps with one matrix read from DRAM and one communication phase
    • Serial: $O(1)$ moves of data moves vs. $O(k)$
    • Parallel: $O(\log p)$ messages vs. $O(k \log p)$

• Can we make communication provably optimal?
  – Communication both to DRAM and between cores
  – Minimize independent accesses (‘latency’) 
  – Minimize data volume (‘bandwidth’)

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin
Bigger Kernel ($A^k x$) Runs at Faster Speed than Simpler ($Ax$)

Speedups on Intel Clovertown (8 core)

Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick
"Monomial" basis $[Ax,\ldots,A^kx]$ fails to converge.

A different polynomial basis does converge.
Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown

Runtime per kernel, relative to CA-GMRES(k,t), for all test matrices, using 8 threads and restart length 60
Communication-Avoiding Dense Linear Algebra

• Well known why BLAS3 beats BLAS1/2: Minimizes communication = data movement
  – Attains lower bound \( \Omega \left( \frac{n^3}{\text{cache_size}^{1/2}} \right) \) words moved in sequential case; parallel case analogous

• Same lower bound applies to all linear algebra
  – BLAS, LU, Cholesky, QR, eig, svd, compositions…
  – Sequential or parallel
  – Dense or sparse (\( n^3 \Rightarrow \#\text{flops in lower bound} \))

• Conventional algs (Sca/LAPACK) do much more

• We have new algorithms that meet lower bounds
  – Good speed ups in prototypes (including on cloud)
  – Lots more algorithms, implementations to develop

  – See talk in the MIT Math Dept, December 13th
General Lessons

• Early intervention with hardware designs
• Optimize for what is important: energy $\rightarrow$ data movement
• Anticipating and changing the future
  – Influence hardware designs
  – Use languages that reflect abstract machine
  – Write code generators / autotuners
  – Redesign algorithms to avoid communication
• These problems are essential for computing performance in general