Algorithmic Challenges of Exascale Computing

Kathy Yelick
Associate Laboratory Director for Computing Sciences and Acting NERSC Center Director
Lawrence Berkeley National Laboratory

EECS Professor, UC Berkeley
“New Algorithm Improves Performance and Accuracy on Extreme-Scale Computing Systems. On modern computer architectures, communication between processors takes longer than the performance of a floating point arithmetic operation by a given processor. ASCR researchers have developed a new method, derived from commonly used linear algebra methods, to minimize communications between processors and the memory hierarchy, by reformulating the communication patterns specified within the algorithm. This method has been implemented in the TRILINOS framework, a highly-regarded suite of software, which provides functionality for researchers around the world to solve large scale, complex multi-physics problems.”

Energy Cost Challenge for Computing Facilities

At ~$1M per MW, energy costs are substantial

- 1 petaflop in 2010 uses 3 MW
- 1 exaflop in 2018 possible in 200 MW with “usual” scaling
- 1 exaflop in 2018 at 20 MW is DOE target
Measuring Efficiency

• Race-to-Halt generally minimized energy use

• For Scientific Computing centers, the metric should be science output per Watt….
  – *NERSC in 2010 ran at 450 publications per MW-year*
  – But that number drops with each new machine

• Next best: application performance per Watt
  – Newest, largest machine is best
  – Lower energy and cost per core
New Processor Designs are Needed to Save Energy

Cell phone processor
(0.1 Watt, 4 Gflop/s)

Server processor
(100 Watts, 50 Gflop/s)

• Server processors have been designed for performance, not energy
  – Graphics processors are 10-100x more efficient
  – Embedded processors are 100-1000x
  – Need manycore chips with thousands of cores
The Amdahl Case for Heterogeneity

A Chip with up to 256 “thin” cores and “fat” core that uses some of the some of the thin core area

Heterogeneity Analysis by: Mark Hill, U. Wisc
New Processors Means New Software

- Exascale will have chips with thousands of tiny processor cores, and a few large ones
- Architecture is an open question:
  - sea of embedded cores with heavyweight “service” nodes
  - Lightweight cores are accelerators to CPUs
Memory Capacity is Not Keeping Pace

Technology trends against a constant or increasing memory per core

- Memory density is doubling every three years; processor logic is every two
- Storage costs (dollars/Mbyte) are dropping gradually compared to logic costs

Evolution of memory density

Cost of Computation vs. Memory

Source: David Turek, IBM

Question: Can you double concurrency without doubling memory?
Why avoid communication?

- Running time of an algorithm is sum of 3 terms:
  - \# flops \* time\_per\_flop
  - \# words moved \/ bandwidth
  - \# messages \* latency

- `Time_per_flop \ll 1 / bandwidth \ll latency`

- Gaps growing exponentially with time [FOSC]

<table>
<thead>
<tr>
<th>Annual improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time_per_flop</strong></td>
</tr>
<tr>
<td>59%</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- And these are hard to change:
  - “Latency is physics, bandwidth is money”
Bandwidth (to Memory and Remote Nodes) is an Energy Hog

![Graph showing energy consumption for different communication types: Intranode/SMP, Intranode/MPI, On-chip / CMP communication. The graph compares current (2018) and future (now) energy consumption.]
Value of Local Store Memory

- Unit stride as important as cache hits on hardware with prefetch
  - Don’t cut unit stride when tiling
- Software controlled memory gives more control ("scrathpad")
  - May also be more for new level of memory between DRAM and disk

Joint work with Shoaib Kamil, Lenny Oliker, John Shalf, Kaushik Datta
Exascale will have chips with thousands of tiny processor cores, and a few large ones.

Architecture is an open question:
- sea of embedded cores with heavyweight “service” nodes
- Lightweight cores are accelerators to CPUs

Low power memory and storage technology are key.
Why Avoid Synchronization?

• Processors do not run at the same speed
  – Never did, due to caches
  – Power / temperature management makes this worse

HPC can’t turn this off
  – Power swings of 50% on systems
  – At $3M/MW of capital costs, don’t want 50% headroom
Errors Can Turn into Performance Problems

• Fault resilience introduces inhomogeneity in execution rates (*error correction is not instantaneous*)

![Diagram showing runtime vs processor number with three lines representing different systems: Jaguar/Catamount XT4 (green), Jaguar/Catamount XT3 (blue), and Franklin/CNL XT4 (red). The graph shows fluctuations in runtime with processor number, indicating performance issues related to error resilience.](Slide source: John Shalf)
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 open, but likely heterogeneous
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) Resiliency critical at large scale (in time or processors)
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., 1000 1-PF machines
Algorithms to Optimize for Communication
Avoiding Communication in Iterative Solvers

• Consider Sparse Iterative Methods for $Ax=b$
  – Krylov Subspace Methods: GMRES, CG,…
  – Can we lower the communication costs?
    • Latency of communication, i.e., reduce # messages by computing multiple reductions at once
    • Bandwidth to memory hierarchy, i.e., compute $Ax, A^2x, \ldots A^kx$ with one read of $A$

• Solve time dominated by:
  – Sparse matrix-vector multiple (SPMV)
    • Which even on one processor is dominated by “communication” time to read the matrix
  – Global collectives (reductions)
    • Global latency-limited
Autotuning Gets Kernel Performance Near Optimal

- Roofline model captures bandwidth and computation limits
- Autotuning gets kernels near the roof

Work by Williams, Oliker, Shalf, Madduri, Kamil, Im, Ethier,…
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

- Idea: pick up part of \(A\) and \(x\) that fit in fast memory, compute each of \(k\) products

- Example: A tridiagonal, \(n=32\), \(k=3\)

- Works for any “well-partitioned” \(A\)
Communication Avoiding Kernels: The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A\cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)
- **Sequential Algorithm**

**Example:** A tridiagonal, \(n=32, k=3\)
- Saves bandwidth (one read of \(A\&x\) for \(k\) steps)
- Saves latency (number of independent read events)
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace k iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)

**Parallel Algorithm**

- Example: A tridiagonal, \(n=32, k=3\)
- Each processor communicates once with neighbors
Communication Avoiding Kernels:
The Matrix Powers Kernel: \([Ax, A^2x, \ldots, A^kx]\)

- Replace \(k\) iterations of \(y = A \cdot x\) with \([Ax, A^2x, \ldots, A^kx]\)
- **Parallel Algorithm**

  ![Diagram showing a parallel algorithm for matrix powers kernel.](attachment:parallel_algorithm.png)

- Example: A tridiagonal, \(n=32, k=3\)
- Each processor works on (overlapping) trapezoid
- Saves latency (# of messages); Not bandwidth
- But adds redundant computation
Matrix Powers Kernel on a General Matrix

- Saves communication for “well partitioned” matrices
  - Serial: $O(1)$ moves of data moves vs. $O(k)$
  - Parallel: $O(\log p)$ messages vs. $O(k \log p)$

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

For implicit memory management (caches) uses a TSP algorithm for layout
Bigger Kernel ($A^k x$) Runs at Faster Speed than Simpler ($Ax$)

Speedups on Intel Clovertown (8 core)
Minimizing Communication of GMRES to solve $Ax=b$

- **GMRES:** find $x$ in span{$b, Ab, ..., A^k b$} minimizing $|| Ax-b ||_2$

Standard GMRES
for $i=1$ to $k$
  
  $w = A \cdot v(i-1)$ \hspace{1cm} ... \hspace{1cm} SpMV
  
  MGS($w, v(0),...,v(i-1)$)
  
  update $v(i)$, H
endfor

solve LSQ problem with H

Communication-avoiding GMRES
$W = [v, Av, A^2v, ..., A^k v]$ 

$[Q,R] = \text{TSQR}(W)$

... “Tall Skinny QR”

build $H$ from $R$

solve LSQ problem with $H$

Sequential case: #words moved decreases by a factor of $k$

Parallel case: #messages decreases by a factor of $k$

- **Oops –** $W$ from power method, precision lost!
TSQR: An Architecture-Dependent Algorithm

Parallel: \[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{bmatrix} \rightarrow \begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix} \rightarrow \begin{bmatrix} R_{02} \end{bmatrix} \]

Sequential: \[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \end{bmatrix} \rightarrow \begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix} \rightarrow \begin{bmatrix} R_{02} \\ R_{03} \end{bmatrix} \]

Dual Core: \[ W = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{bmatrix} \rightarrow \begin{bmatrix} R_{00} \\ R_{10} \\ R_{20} \end{bmatrix} \rightarrow \begin{bmatrix} R_{01} \\ R_{11} \end{bmatrix} \rightarrow \begin{bmatrix} R_{02} \\ R_{03} \end{bmatrix} \]

Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?
Can choose reduction tree dynamically

Work by Laura Grigori, Jim Demmel, Mark Hoemmen, Julien Langou
TSQR Performance Results

• Parallel
  – Intel Clovertown
    – Up to 8x speedup (8 core, dual socket, 10M x 10)
  – Pentium III cluster, Dolphin Interconnect, MPICH
    • Up to 6.7x speedup (16 procs, 100K x 200)
  – BlueGene/L
    • Up to 4x speedup (32 procs, 1M x 50)
  – Grid – 4x on 4 cities (Dongarra et al)
  – Cloud – early result – up and running using Mesos

• Sequential
  – Out-of-Core on PowerPC laptop
    • As little as 2x slowdown vs (predicted) infinite DRAM
    • LAPACK with virtual memory never finished

Data from Grey Ballard, Mark Hoemmen, Laura Grigori, Julien Langou, Jack Dongarra, Michael Anderson
Matrix Powers Kernel (and TSQR) in GMRES

![Graph showing relative norm of residual over iteration count for different methods: Original GMRES, CA-GMRES (Monomial basis), CA-GMRES (Newton basis).]
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

Relative runtime, for best \(k,t\) with floor restarting length \(k\) for: matrix powers, kernel, TSQR, Block Gram-Schmidt, Small dense operations, Sparse matrix-vector product, Modified Gram-Schmidt.
CA-Krylov Methods Summary and Future Work

• The Communication-Avoidance works
  – Provably optimal
  – Faster in practice

• Ongoing work for “hard” matrices
  – Partition poorly (high surface to volume)
  – Idea: separate out dense rows (HSS matrices)
  – [Erin Carson, Nick Knight, Jim Demmel]
Beyond Domain Decomposition: 2.5D Matrix Multiply

- Conventional “2D algorithms” use $P^{1/2} \times P^{1/2}$ mesh and minimal memory
- New “2.5D algorithms” use $(P/c)^{1/2} \times (P/c)^{1/2} \times c^{1/2}$ mesh and c-fold memory
  - Matmul sends $c^{1/2}$ times fewer words – lower bound
  - Matmul sends $c^{3/2}$ times fewer messages – lower bound

2.5D MM on BG/P (n=65,536)

Perfect Strong Scaling

Lesson: Never waste fast memory

Word by Edgar Solomonik and Jim Demmel
Communication Avoidance in Multigrid: Method of Local Corrections (MLC) for Poisson’s Equation

\[ \Delta \varphi = \rho, \quad \varphi(x) = \int_{\Omega} G(x - y) \rho(y) \, dy, \quad x \in \Omega \]

Real analytic, with rapidly convergent Taylor expansion

MLC uses domain decomposition, plus a noniterative form of multigrid, to represent the solution in a way that minimizes global communication and increases computational intensity.

- Local domains fit in cache; solves computed using FFT and a simplified version of FMM. For \(16^3 - 32^3\), this yields \(~5\) flops/byte.
- The (global) coarse problem is small. Communication comparable to single relaxation step.
- Weak scaling: 95% efficiency up to 1024 processors.

Work by Phil Colella et al
Avoiding Synchronization
Avoid Synchronization from Applications

Computations as DAGs

View parallel executions as the directed acyclic graph of the computation

Cholesky 4 x 4

QR 4 x 4

Slide source: Jack Dongarra
Avoiding Synchronization in Communication

- Two-sided message passing (e.g., MPI) requires matching a send with a receive to identify memory address to put data
  - Wildly popular in HPC, but cumbersome in some applications
  - Couples data transfer with synchronization
- Using global address space decouples synchronization
  - Pay for what you need!
  - Note: Global Addressing ≠ Cache Coherent Shared memory

Joint work with Dan Bonachea, Paul Hargrove, Rajesh Nishtala and rest of UPC group
Event Driven LU in UPC

- Assignment of work is static; schedule is dynamic
- Ordering needs to be imposed on the schedule
  - Critical path operation: Panel Factorization
- General issue: dynamic scheduling in partitioned memory
  - Can deadlock in memory allocation
  - “memory constrained” lookahead
DAG Scheduling Outperforms Bulk-Synchronous Style

PLASMA on shared memory

UPC on partitioned memory

UPC LU factorization code adds cooperative (non-preemptive) threads for latency hiding

- New problem in partitioned memory: allocator deadlock
- Can run on of memory locally due to unlucky execution order

PLASMA by Dongarra et al; UPC LU joint with Parray Husbands
Irregular vs. Regular Parallelism

• Computations with known task graphs can be mapped to resources in an offline manner (before computation starts)
  – Regular graph: By a compiler (static) or runtime (semi-static)
  – Irregular graphs: By a DAG scheduler
  – No need for online scheduling

• If graphs are not known ahead of time (structure, task costs, communication costs), then dynamic scheduling is needed
  – Task stealing / task sharing
  – Demonstrated on shared memory

• Conclusion: If your task graph is dynamic, the runtime needs to be, but what if it static?
Load Balancing with Locality

• Locality is important:
  – When memory hierarchies are deep
  – When computational intensity is low

• Most (all?) successful examples of locality-important applications/machines use static scheduling
  – Unless they have an irregular/dynamic task graph

• Two extremes are well-studied
  – Dynamic parallelism without locality
  – Static parallelism (with threads = processors) with locality

• Dynamic scheduling and locality control don’t mix
  – Locality control can cause non-optimal task schedule, which can blow up memory use (breadth vs. depth first traversal)
  – Can run out of memory locally when you don’t globally
Use Two Programming Models to Match Machine

Hybrid Programming is key to saving memory (2011) and sometimes improves performance.
Getting the Best of Each

**PGAS for locality and convenience**

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

*Partitioned:* data is designated as local or global

- Affinity control
- Scalability
- Never say “receive”

**Debate is around the control model:**

- Dynamic thread creation vs. SPMD
- Hierarchical SPMD compromise
  - “Think parallel” and group as needed
Hierarchical SPMD in Titanium

• Thread teams may execute distinct tasks

\[
\text{partition}(T) \{ \\
\{ \text{model\_fluid}(); \} \\
\{ \text{model\_muscles}(); \} \\
\{ \text{model\_electrical}(); \} \\
\}
\]

• Hierarchy for machine / tasks
  – Nearby: access shared data
  – Far away: copy data

• Advantages:
  – Provable pointer
  – Mixed data / task style
  – Lexical scope prevents some deadlocks
Hierarchical Programming Model: Phalanx

- Invoke functions on set of cores and set of memories
- Hierarchy of memories
  - Can query to get (some) aspects of the hierarchical structures
- Functionally homogeneous cores (on Echelon)
  - Can query to get (performance) properties of cores
- Hierarchy of thread blocks
  - May be aligned with hardware based on queries

Echelon ProgSys Team: Michael Garland, Alex Aiken, Brad Chamberlain, Mary Hall, Greg Titus, Kathy Yelick
Stepping Back

• Communication avoidance as old at tiling
• Communication optimality as old as Hong/Kung

What’s new?
– Raising the level of abstraction at which we optimize
– BLAS2 → BLAS3 → LU or SPMV/DOT → Krylov
– Changing numerics in non-trivial ways
– Rethinking methods to models

• Communication and synchronization avoidance
• Software engineering: breaking abstraction
• Compilers: inter-procedural optimizations
Exascale Views

• “Exascale” is about continuing growth in computing performance for science
  – Energy efficiency is key
  – Job size is irrelevant

• Success means:
  – Influencing market: HPC, technical computing, clouds, general purpose
  – Getting more science from data and computing

• Failure means:
  – few big machines for a few big applications

• Not all computing problems are exascale, but they should all be exascale-technology aware
Thank You!