Programming Model Challenges for Managing Massive Concurrency

Kathy Yelick

National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory and
EECS Department, University of California, Berkeley
Software Issues at Scale

• Power concerns will dominates all others;
  – Concurrency is the most significant knob we have: lower clock, increase parallelism
  – Power density and facility energy
    • TCO has always been a factor in NERSC procurements, and while LBNL electrical rates are very low, about 6.5cents/kwh, this term in TCO is growing

• Summary Issues for Software
  – 1EF system: Billion-way concurrency, O(1K) cores per chip
  – 1 PF system: millions of threads and O(1K) cores per chip
  – The memory capacity/core ratio may drop significantly
  – Faults will become more prevalent
  – Flops are cheap relative to data movement
Massive Concurrency

- Processor architecture in chaos
  - Programming model research and development is harder than ever
- “Core” is probably the wrong word
- How many threads (program counters) per functional unit (say floating point)
  - 1 as in traditional microprocessors?
  - >1 with multithreading (e.g., Sun, Cray,…)
  - <1 as with vectors / SIMD (energy efficient)
- This is not just an HPC concern if we’re going to continue to leverage commodity processors
  - May encourage heterogeneity (see Berkeley View)
Efficiency of software is also critical, so:

How can we waste an Exaflop machine?
Rule #1: Ignore Little’s Law

\[
\text{Required concurrency} = \text{Latency} \times \text{Bandwidth}
\]

Ignoring this is a good way to waste expensive memory bandwidth
void stencil3d(double A[], double B[], int nx, int ny, int nz) {
    for all grid indices in x-dim {
        for all grid indices in y-dim {
            for all grid indices in z-dim {
                B[center] = S0* A[center] +
            }
        }
    }
}

• 3D, 7-point, Jacobi iteration on a 256³ grid
• Flop:Byte Ratio:
  – 0.33 (write allocate), 0.5 (Ideal)

Joint work by K. Datta, S. Kamil, S. Williams
J. Shalf, L. Oliker
**Stencil Performance**
*(out-of-the-box code)*

- Expect performance to be between SpMV and LBMHD
- Scalability is universally poor
- Performance is poor
Auto-tuned Stencil Performance
(architecture specific optimizations)

- Cache bypass can significantly improve Barcelona performance.
- DMA, SIMD, and cache blocking were essential on Cell
Rule #2: Use processors engineered for serial applications

Many features of “high performance” processors waste energy: out-of-order execution, speculation, hardware-controlled caches,...
Stencil Results

**Single Precision**

- Performance
  - Total GFlop/s
    - Clovertown: 2.5
    - Barcelona: 8
    - Victoria Falls: 7
    - Cell: 14
    - G80: 14
    - G80/PCIe: 3

- Power Efficiency
  - MFlop/s/Watt
    - Clovertown: 5
    - Barcelona: 15
    - Victoria Falls: 10
    - Cell: 15
    - G80: 20
    - G80/PCIe: 3

**Double Precision**

- Performance
  - Total GFlop/s
    - Clovertown: 45
    - Barcelona: 16
    - Victoria Falls: 12
    - Cell: 34
    - G80: 53
    - G80/PCIe: 3

- Power Efficiency
  - MFlop/s/Watt
    - Clovertown: 5
    - Barcelona: 15
    - Victoria Falls: 10
    - Cell: 20
    - G80: 25
    - G80/PCIe: 3
**Why is the STI Cell So Efficient?**
*(Latency Hiding with Software Controlled Memory)*

- **STriad Benchmark**
  - Measures bandwidth on alternation unit strides runs (Stanza) and jumps
- **Tremendous cost to non-unit stride in traditional architectures**
  - Smarter prefetchers may improve this for some patterns, but can be counter-productive
  - Explains why decreasing cache misses does not always increase performance
Rule #3: Rely on weak scaling

Many SC highlights from past decade have used problems that weakly scale
DRAM component density is only doubling every 3 years

Weak scaling at risk, even for science problems that can weakly scale
Rule #4: Synchronize all data communication events
Sharing and Communication Models: PGAS vs. MPI

- A one-sided put/get message can be handled directly by a network interface with RDMA support
  - Avoid interrupting the CPU or storing data from CPU (preposts)
- A two-sided messages needs to be matched with a receive to identify memory address to put data
  - Offloaded to Network Interface in networks like Quadrics
  - Need to download match tables to interface (from host)

Joint work with D. Bonachea, R. Nishtala, P. Hargrove, and the UPC group
Two-Sided Communication Introduced Synchronization Overhead

Use a programming model in which you can’t utilize bandwidth or “low” latency

8-byte Roundtrip Latency

Flood Bandwidth for 4KB messages
Strongly Scaled 3D FFT on BG/P

![Graph showing performance results for different methods in 3D FFT on BG/P](image)
What’s Wrong with MPI Everywhere

• We can run 1 MPI process per core
  – This works now (for CMPs) and will work for a while

• How long will it continue working?
  – Depends on performance expectations -- more on this later

• 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: success model for the “cluster era;” will not be for the many core era -- not enough memory per core
  – Heterogeneity: MPI per SIMD element or CUDA thread-block?
What about Mixed MPI and Threads?

- Threads: OpenMP, PThreads, …
- Problems
  - Will OpenMP performance scale with the number of cores / chip?
  - More investment in infrastructure than MPI, but can leverage existing technology
  - Do people want two programming models?
  - Doesn’t go far enough
    - Thread context is a large amount of state compared to vectors/streams
    - Op per instruction vs. 64 ops per instruction
PGAS Languages

- **Global address space**: thread may directly read/write remote data
- **Partitioned**: data is designated as local or global

**Implementation issues:**
- Distributed memory: Reading a remote array or structure is explicit, not a cache fill
- Shared memory: Caches are allowed, but not required

- No less scalable than MPI!
- Permits sharing, whereas MPI rules it out!
Sharing and Communication Models: PGAS vs. Threads

• “Shared memory” OpenMP, Threads,…
  – No control over locality
    ⇒ Caching (automatic management of memory hierarchy) is critical
    ⇒ Cache coherent needed (hw or sw)
• PGAS / One-sided Communication
  – Control over locality, explicit movement
    ⇒ Caching is not required; programmer makes local copies and manages their consistency
    ⇒ Need to read/write without bothering remote application (progress thread, DMA)
    ⇒ No cache coherent needed, except between the network interface and procs in a node
• PGAS languages are a good fit to shared memory machines, including multicore
  – Global address space implemented as reads/writes
  – Also may be exploited for processor with explicit local store rather than cache, e.g., Cell, GPUs,…
• Open question in architecture
  – Cache-coherence shared memory
  – Software-controlled local memory (or hybrid)
Rule #5: Add Global Synchronization
Avoid Global Synchronization

- Bulk-synchronous programming has too much synchronization
- Bad for performance
  - Linpack performance
  - On Multicore / SMP (left, Dongarra et al) and distributed memory (right, UPC)
- Also bad direction for fault tolerance
Rule #6: Use Algorithms
Design to minimize Flops

Count data movement, not Flops
Communication is limiting factor for scalability
- Movement of data within memory hierarchy
- Movement of data between cores/nodes

Two aspects of communication
- Latency: number of discrete events (cache misses, messages)
- Bandwidth: volume of data moved

More efficient algorithms tend to run less efficiently
- Algorithm efficiency: E.g., O(n) ops on O(n) data
- Efficiency of computation on machine: % of peak
- At scale, cannot afford to be algorithmically inefficient
- Need to run low compute intensity algorithms well, or flops are wasted

Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyudden
Avoiding Communication in Sparse Linear Algebra

- Take $k$ steps of Krylov subspace method
  - GMRES, CG, Lanczos, Arnoldi
  - Assume matrix “well-partitioned,” with modest surface-to-volume ratio
  - Parallel implementation
    - Conventional: $O(k \log p)$ messages
    - “New”: $O(\log p)$ messages - optimal
  - Serial implementation
    - Conventional: $O(k)$ moves of data from slow to fast memory
    - “New”: $O(1)$ moves of data – optimal

- Can incorporate some preconditioners
  - Hierarchical, semiseparable matrices ...

- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation
Avoiding Communication in Sparse Iterative Solvers

- Consider Sparse Iterative Methods for $Ax=b$
  - Use Krylov Subspace Methods like GMRES, CG
  - Can we lower the communication costs?
    - Latency of communication, i.e., reduce # messages by computing $A^k x$ with one read of remote $x$
    - Bandwidth to memory hierarchy, i.e., compute $A$
- Example: Inner loop is sparse matrix-vector multiply, $Ax$ (=nearest neighbor computation on a graph)
- Partition into cache blocks or by processor, and take multiple steps
- Simple examples, $A$ is matrix of:
  - 2D Mesh has “5 point stencil”
  - 1D mesh mesh “3 point stencil”

Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyudden
Locally Dependent Entries for $[x, Ax]$, A tridiagonal
2 processors

Can be computed without communication
Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin
Locally Dependent Entries for \([x,Ax,A^2x]\), A tridiagonal 2 processors

Can be computed without communication
Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyudden
Locally Dependent Entries for \([x, Ax, ..., A^3x]\), A tridiagonal 2 processors

Can be computed without communication
Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin
Locally Dependent Entries for \([x, Ax, \ldots, A^4x]\), A tridiagonal 2 processors

Can be computed without communication

Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyudden
Latency Avoiding Parallel Kernel for $[x, Ax, A^2x, \ldots, A^kx]$

- Compute **locally dependent entries** needed by neighbors
- Send/shared data to neighbors
- Compute remaining **locally dependent entries**
- Compute **remotely dependent entries**
- Shown for 1D mesh, but can be done for general matrix
  - Traveling salesman problem within for layout

Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyudden
Can use Matrix Power Kernel, but change Algorithms

Work by Demmel and Hoemmen
Performance Results To Date

• Tall-skinny QR (measured)
  • 6.7x on Pentium cluster, 4x on BlueGene
• Square QR (modeled)
  • 22x on petascale machine) 22x
• $A^kx$ kernel
  • (modeled 2D mesh matrix, on petascale)
    15x without overlap, 7x with overlap
  • (measured 2D mesh matrix, on “out of core” system with matrix on disk) 3.2x
Conclusions

• Re-think Programming Models
  – Software to make the most of hardware
    • One-sided communication to avoid synchronization
    • Global address space to increase sharing (re-use) and for productivity

• Re-think software for libraries/applications
  – Write self-tuning applications

• Re-think Algorithms
  – Design for bottlenecks: latency and bandwidth