Designing Linear Algebra for Multicore

D. Bindel

Courant Institute for Mathematical Sciences
New York University

Manycore Reading Group, 8 Mar 2008
Consider Cholesky factorization: \( A = LL^T \).

- Useful for solving linear SPD systems
- Straightforward algorithm, \( n^2/2 \) data, \( n^3/3 \) floating point ops
- Given as a programming exercise for my scientific computing class
- Also: compare to LAPACK / MATLAB / Octave
Timing for three versions of Cholesky

GFlop/s for Cholesky codes

- Naive code
- LAPACK code (vecLib)
- Octave code

$GFlop/s$ for Cholesky codes

$n$
Timing for naive Cholesky

GFlop/s for naive Cholesky code

$GFlop/s$ vs $n$
Can one rationally focus on parallelism and claim that the serial performance is a secondary issue?
Naive code is limited by memory latency (almost constant cache misses)

On this machine, about $50 \times$ difference between L1 hit and main memory access.

LAPACK uses blocking – most of the computation is shifted onto level 3 BLAS operations (i.e. matrix-matrix multiplies) which are tuned for cache efficiency.

There are projects for automatically tuning such computational kernels (FFTW, FLAME, ATLAS, OSKI, etc).
Algorithmic tradeoffs

- Often have *mathematically* equivalent reorganizations of LA algorithms (i.e. same formulas, different arrangement)
  - *Not* equivalent with respect to cache!
  - *Not* equivalent with respect to communication!
  - *Not* equivalent with respect to approximate arithmetic!

- Also have arrangements that compute the same thing through different formulations.
Algorithm strategies

For *direct* (dense or sparse direct) linear algebra:
- Blocking to avoid memory overhead
- Redundant computations to avoid communication
- Randomization to avoid need for pivoting
- Iterative refinement to clean up fast-but-sloppy computations

For *iterative* linear algebra:
- Trading time to precondition against iteration count
- Redundant computations to avoid communication
Shameless theft!

“Lesser artists borrow. Great artists steal.”
– Picasso, Dali, Stravinsky?

Demmel: “Avoiding Communication in Linear Algebra.”
Programmer’s task

Find an algorithm that

1. Yields the right answer.
2. Uses structure to minimize computation.
3. Is organized to minimize memory / communication burdens.
How languages can help

For novices:
- Make it easy to build on work of experts!

For experts:
- Make it “easy” to write cache-aware algorithms.
- Make it possible to tune those algorithms.
- Make it possible to express tradeoffs (e.g. scoped optimizations like “license associativity”).

Unfortunately, line between “novice” and “expert” is unclear.