Introduction to the On-line Course:

CS267
Applications of Parallel Computers

www.cs.berkeley.edu/~demmel/cs267_Spr14/

Jim Demmel

EECS & Math Departments

demmel@berkeley.edu

Outline

- 3 free on-line parallel computing courses
  - Offered by UC Berkeley and XSEDE
- CS267 – Applications of Parallel Computers
  - Outline of course
  - Big Idea #1: Common Computational Patterns
  - Big Idea #2: Avoiding Communication
  - Who takes it? What final projects do people do?
- Parallel Shortcourse (CS267 in 3 days)
- CS194 – Engineering Parallel Software (for Performance)
  - Undergraduate version – emphasis on (parallel) software engineering

CS267 – Applications of Parallel Computers

- UC Berkeley 1-semester graduate class (some ugrads too)
- Offered every Spring semester - webcast live
  - Spring 2015: Jan 20 – Apr 30, T Th 9:30-11:00
  - Local students use DOE supercomputers at NERSC
- Archived videos broadcast by XSEDE
  - Carefully edited lectures with in-line quizzes
  - Available as SPOC = Small, Personalized On-line Course to students for credit at their universities, with local instructor to give official grades
  - Free NSF supercomputer accounts to do autograded homework and class projects
  - UC Berkeley teaching assistants help answer questions for remote students
  - Contact Steve Gordon (sgordon@osc.edu) if interested

Motivation and Outline of Course

- Why ~~powerful~~ *all* computers must be parallel processors

  Including your laptops and handhelds
- Large Computational Science and Engineering (CSE) problems require powerful computers

  Commercial problems too
- Why writing (fast) parallel programs is hard

  But things are improving
- Structure of the course

Course summary and goals

- Goal: teach grads and advanced undergrads from diverse departments how to use parallel computers:
  - Efficiently – write programs that run fast
  - Productively – minimizing programming effort
- Basics: computer architectures
  - Memory hierarchies, Multicore, Distributed Memory, GPU, Cloud, …
- Basics: programming languages
  - C + Pthreads, OpenMP, MPI, CUDA, UPC, frameworks …
- Beyond basics: common "patterns" used in all programs that need to run fast:
  - Linear algebra, graph algorithms, structured grids,…
  - How to compose these into larger programs
- Tools for debugging correctness, performance
- Guest lectures: climate modeling, astrophysics, …

5

Which applications *require* parallelism?

Analyzed in detail in "Berkeley View" report

HPC

Structured Grid
Dense Matrix
Sparse Matrix
Spectral (FFT)

N-Body
MapReduce

Unstructured Grid

Which applications *require* parallelism?

Analyzed in detail in "Berkeley View" report

Embed SPEC DB Games ML HPC

Structured Grid
Dense Matrix
Sparse Matrix
Spectral (FFT)

N-Body
MapReduce

Unstructured Grid

Which applications *require* parallelism?

Analyzed in detail in "Berkeley View" report

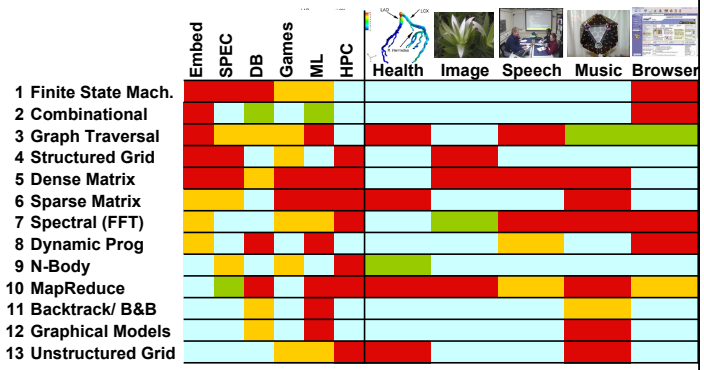
Embed SPEC DB Games ML HPC

1 Finite State Mach.
2 Combinational
3 Graph Traversal
4 Structured Grid
5 Dense Matrix
6 Sparse Matrix
7 Spectral (FFT)
8 Dynamic Prog
9 N-Body
10 MapReduce
11 Backtrack/ B&B
12 Graphical Models
13 Unstructured Grid

Analyzed in detail in "Berkeley View" report www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

What do commercial and CSE applications have in common?

Motif/Dwarf: Common Computational Patterns
(Red Hot → Blue Cool)



For (many of) these patterns, we present

- How they naturally arise in many applications
- How to compose them to implement applications
- How to find good existing implementations, if possible
- Underlying algorithms
  - Usually many to choose from, with various tradeoffs
    - Autotuning: use computer to choose best algorithm
  - Lots of open research questions
  - What makes one algorithm more efficient than another?
    - Most expensive operation is not arithmetic, it is communication, i.e. moving data, between level of a memory hierarchy or between processors over a network.
  - Goal: avoid communication

10

Why avoid communication? (1/2)

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

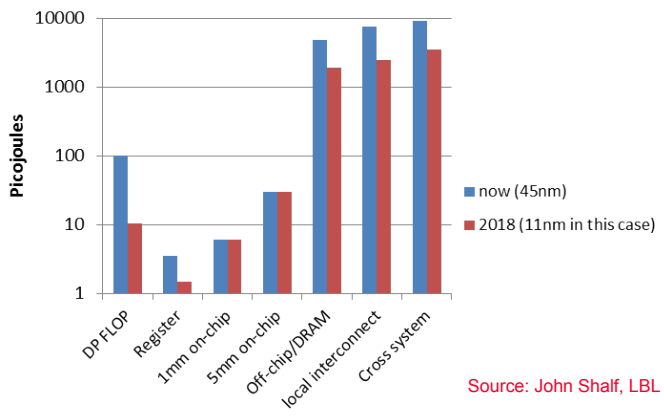
  } communication
- Time_per_flop << 1/ bandwidth << latency
  - Gaps growing exponentially with time [FOSC]

Annual improvements			
Time_per_flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

- Avoid communication to save time

11

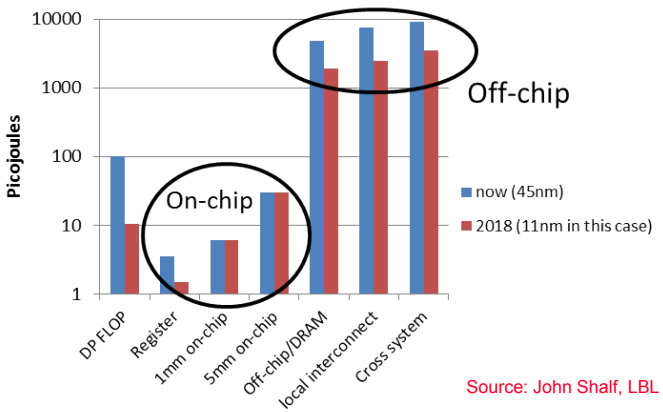
Why Minimize Communication? (2/2)



Source: John Shalf, LBL

Why Minimize Communication? (2/2)

Minimize communication to save energy



Source: John Shalf, LBL

Goals

- Redesign algorithms to *avoid* communication
  - Between all memory hierarchy levels
    - L1 ↔ L2 ↔ DRAM ↔ network, etc
- Attain lower bounds if possible
  - Current algorithms often far from lower bounds
  - Large speedups and energy savings possible

14

President Obama cites Communication-Avoiding Algorithms in the FY 2012 Department of Energy Budget Request to Congress:

"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."

FY 2010 Congressional Budget, Volume 4, FY2010 Accomplishments, Advanced Scientific Computing Research (ASCR), pages 65-67.

CA-GMRES (Hoemmen, Mohiyuddin, Yelick, JD)
"Tall-Skinny" QR (Grigori, Hoemmen, Langou, JD)

Summary of CA Linear Algebra

- "Direct" Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - Mostly not attained by algorithms in standard libraries
  - New algorithms that attain these lower bounds
    - Being added to libraries: Sca/LAPACK, PLASMA, MAGMA
    - Large speed-ups possible
  - Autotuning to find optimal implementation
- Ditto for "Iterative" Linear Algebra

Lower bound for all "n^3-like" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul,

17

Lower bound for all "n^3-like" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent ≥ #words_moved / largest_message_size

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

18

Lower bound for all "n^3-like" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent (per processor) = Ω(#flops (per processor) / $M^{3/2}$)

- Parallel case: assume either load or memory balanced
- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops << n^3)

SIAM SIAG/Linear Algebra Prize, 2012
Ballard, D., Holtz, Schwartz

Can we attain these lower bounds?

- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Often not
- If not, are there other algorithms that do?
  - Yes, for much of dense linear algebra, APSP
  - New algorithms, with new numerical properties, new ways to encode answers, new data structures
  - Not just loop transformations (need those too!)
- Only a few sparse algorithms so far
  - Ex: Matmul of "random" sparse matrices
  - Ex: Sparse Cholesky of matrices with "large" separators
- Lots of work in progress

20

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors
- Minimum Memory per processor = M = $O(n^2 / P)$
- Recall lower bounds:
  #words_moved = $\Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$
  #messages = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$
- Does ScaLAPACK attain these bounds?
  - For #words_moved: mostly, except nonsym. Eigenproblem
  - For #messages: asymptotically worse, except Cholesky
- New algorithms attain all bounds, up to polylog(P) factors
  - Cholesky, LU, QR, Sym. and Nonsym eigenproblems, SVD

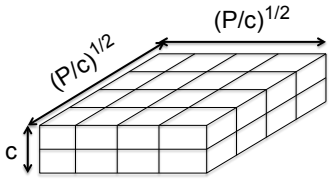
Can we do Better?

Can we do better?

- Aren't we already optimal?
- Why assume M = $O(n^2/p)$, i.e. minimal?
  - Lower bound still true if more memory
  - Can we attain it?

2.5D Matrix Multiplication

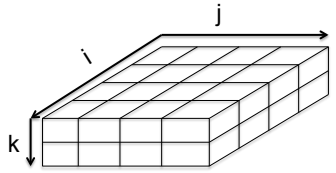
- Assume can fit cn^2/P data per processor, c > 1
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



Example: P = 32, c = 2

2.5D Matrix Multiplication

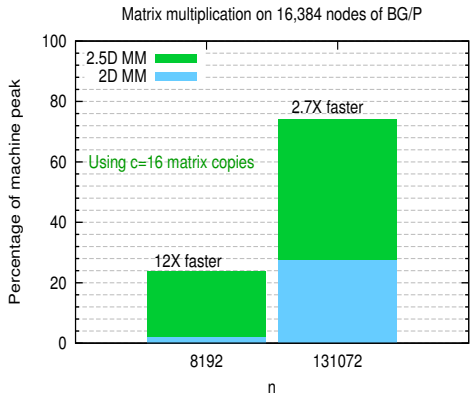
- Assume can fit cn^2/P data per processor, c > 1
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



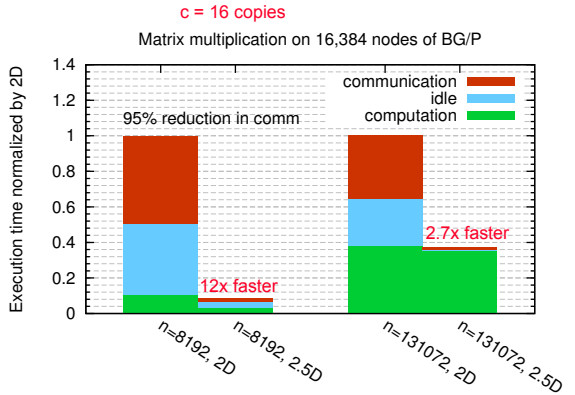
Initially P(i,j,0) owns A(i,j) and B(i,j) each of size $n(c/P)^{1/2}$ x $n(c/P)^{1/2}$

(1) P(i,j,0) broadcasts A(i,j) and B(i,j) to P(i,j,k)
(2) Processors at level k perform 1/c-th of SUMMA, i.e. 1/c-th of Σ_m A(i,m)*B(m,j)
(3) Sum-reduce partial sums Σ_m A(i,m)*B(m,j) along k-axis so P(i,j,0) owns C(i,j)

2.5D Matmul on BG/P, 16K nodes / 64K cores



2.5D Matmul on BG/P, 16K nodes / 64K cores



Distinguished Paper Award, EuroPar'11 (Solomonik, D.)
SC'11 paper by Solomonik, Bhatele, D.

Perfect Strong Scaling – in Time and Energy

- Every time you add a processor, you should use its memory M too
- Start with minimal number of procs: $PM = 3n^2$
- Increase P by a factor of c ➔ total memory increases by a factor of c
- Notation for timing model:
  - γ_T, β_T, α_T = secs per flop, per word_moved, per message of size m
- $T(cP) = n^3/(cP) [\gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})]$
  $= T(P)/c$
- Notation for energy model:
  - γ_E, β_E, α_E = joules for same operations
  - δ_E = joules per word of memory used per sec
  - ε_E = joules per sec for leakage, etc.
- $E(cP) = cP \{ n^3/(cP) [\gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})] + \delta_E MT(cP) + \varepsilon_E T(cP) \}$
  $= E(P)$
- Extends to N-body, Strassen, …
- Can prove lower bounds on needed network (eg 3D torus for matmul)

What are students expected to do?

- On-line students – quizzes embedded in lectures
- HW0 – Describing a Parallel Application
- HW1 – Tuning matrix multiplication
- HW2 – Parallel Particle Simulation
- HW3 – Parallel Knapsack*
- Project – Many possibilities …

28

On-Line Quizzes for XSEDE students

- Simple questions after each 20-30 minute video ensuring students followed speaker and understood concepts
- Around 20 questions per lecture depending on length
- All questions multiple choice with 4 options and students allowed 3 tries (can be varied via moodle options)

29

HW0 – Describing a Parallel Application

- A short description of self and parallel application that student finds interesting
- Should include a short bio, research interests, goals for the class then a description of parallel application
- Helps group students later for class projects
- Examples of the 2012 descriptions can be found at this link:

https://docs.google.com/a/berkeley.edu/spreadsheet/pub?key=0AjRuoFQ8J8BRdFVHNnBNTmhBUXR3ZEIJWExvb21vS0E&single=true&gid=2&output=html

30

Programming Assignments

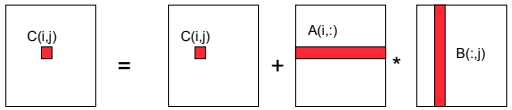
- Each assignment has 'autograder' code given to students so they can run tests and see potential grade they will receive
- Submission of files is done to the moodle website for each assignment with instructions for archiving files or submitting particular compiler directives.
- With help from TAs, local instructors will download files and run submissions and update scores on moodle after runs return
  - For the 2nd and 3rd assignment this usually takes 30+ minutes for scaling studies to finish

31

HW1 – Tuning Matrix Multiply

- Square matrix multiplication
- Simplest code very easy but inefficient

```
for (i=0;i<n;i++)
  for (j=0;j<n;j++)
    for (k=0;j<n;k++)
      C(i,j) = C(i,j) +A(i,k)*B(k,j);
```



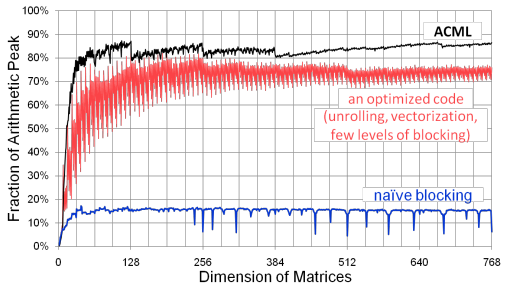
n3 + O(n2) reads/writes altogether

- Give students naïve blocked code

32

HW1 – Tuning Matrix Multiply

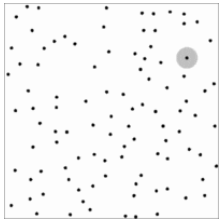
- Memory access and caching
- SIMD code and optimization
- Using Library codes where available



33

HW2 – Parallel Particle Simulation

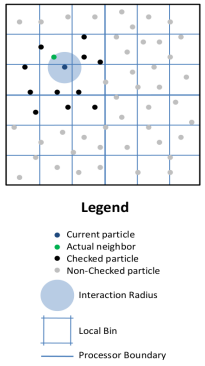
- Simplified particle simulation
- Far field forces null outside interaction radius (makes simple O(n) algorithm possible)
- Give $O(n^2)$ algorithm for serial, OpenMP, MPI and CUDA (2nd part)



34

HW2 – Parallel Particle Simulation

- Introduction to OpenMP, MPI and CUDA (2nd part) calls
- Domain decomposition and minimizing communication
- Locks for bins and avoiding synchronization overheads



35

HW3 – Parallel Knapsack*

- 0-1 Knapsack problem for n items
- Serial, UPC dynamic programming solution given but using inefficient UPC collectives and layout

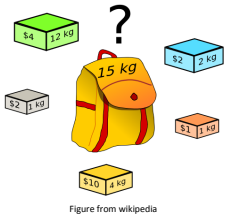


Figure from wikipedia

T(:,0) = 0
T(w,i) = max(T(w,i-1) , T(w-w_i, i-1) + v_i)
T(W_{max}, n) is solution

- Initial UPC version runs slower than serial for most cases due to communication overhead

36

Students completing class at Berkeley in 2014

- 54 enrolled (40 grad, 9 undergrad, 5 other)
- 28 CS or EECS students, rest from

Applied Math
Applied Science & Technology
Astrophysics
BioPhysics
Business Administration
Chemical Engineering
Chemistry
Civil & Environmental Engineering
Math
Mechanical Engineering
Music
Nuclear Engineering
Physics

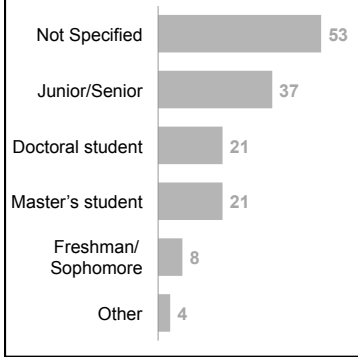
- 6 CS or EECS undergrads, 3 double

37

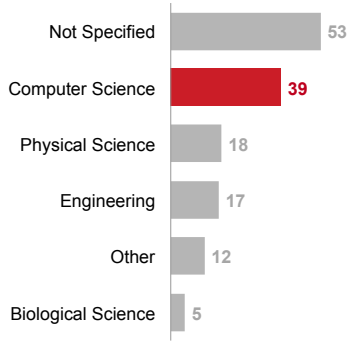
CS267 Spring 2014 XSEDE Student Demographics

144 Students from 19 Universities in
North America, South America and Europe enrolled

Graduate and undergraduate students were near evenly represented



Most students were in Computer Science



CS267 Class Project Suggestions

How to Organize A Project Proposal

- Parallelizing/comparing implementations of an Application
- Parallelizing/comparing implementations of a Kernel
- Building /evaluating a parallel software tool
- Evaluating parallel hardware

40

A few sample CS267 Class Projects

all posters and video presentations at
www.cs.berkeley.edu/~demmel/cs267_Spr09/posters.html

- Content based image recognition
  - "Find me other pictures of the person in this picture"
- Faster molecular dynamics, applied to Alzheimer's Disease
- Better speech recognition through a faster "inference engine"
- Faster algorithms to tolerate errors in new genome sequencers
- Faster simulation of marine zooplankton population
- Sharing cell-phone bandwidth for faster transfers

3/22/12

More Prior Projects

1. High-Throughput, Accurate Image Contour Detection
2. CUDA-based rendering of 3D Minkowski Sums
3. Parallel Particle Filters
4. Scaling Content Based Image Retrieval Systems
5. Towards a parallel implementation of the Growing String Method
6. Optimization of the Poisson Operator in CHOMBO
7. Sparse-Matrix-Vector-Multiplication on GPUs
8. Parallel RI-MP2

42

More Prior Projects

1. Parallel FFTs in 3D: Testing different implementation schemes
2. Replica Exchange Molecular Dynamics (REMD) for Amber's Particle-Mesh Ewald MD (PMEMD)
3. Creating a Scalable HMM based Inference Engine for Large Vocabulary Continuous Speech Recognition
4. Using exponential integrators to solve large stiff problem
5. Clustering overlapping reads without using a reference genome
6. An AggreGATE Network Abstraction for Mobile Devices
7. Parallel implementation of multipole-based Poisson-Boltzmann solver
8. Finite Element Simulation of Nonlinear Elastic Dynamics using CUDA

43

Still more prior projects

1. Parallel Groebner Basis Computation using GASNet
2. Accelerating Mesoscale Molecular Simulation using CUDA and MPI
3. Modeling and simulation of red blood cell light scattering
4. NURBS Evaluation and Rendering
5. Performance Variability in Hadoop's Map Reduce
6. Utilizing Multiple Virtual Machines in Legacy Desktop Applications
7. How Useful are Performance Counters, Really? Profiling Chombo Finite Methods Solver and Parsec Fluids Codes on Nehalem and SiCortex
8. Energy Efficiency of MapReduce
9. Symmetric Eigenvalue Problem: Reduction to Tridiagonal
10. Parallel POPCycle Implementation

44

QUESTIONS?

45