Quick review of earlier lecture

• What do you call
  • A program written in PyGAS, a Global Address Space language based on Python…
  • That uses a Monte Carlo simulation algorithm to approximate π …
  • That has a race condition, so that it gives you a different funny answer every time you run it?

Monte - π - thon

Outline

• History and motivation
  • What is dense linear algebra?
  • Why minimize communication?
  • Lower bound on communication
• Structure of the Dense Linear Algebra motif
  • What does \( A \backslash b \) do?
• Parallel Matrix-matrix multiplication
  • Attaining the lower bound
• Other Parallel Algorithms (next lecture)
Motifs

The Motifs (formerly “Dwarfs”) from “The Berkeley View” (Asanovic et al.)

Motifs form key computational patterns

A brief history of (Dense) Linear Algebra software (1/7)

• In the beginning was the do-loop…
  • Libraries like EI/SPACK (for eigenvalue problems)
  • Then the BLAS (1) were invented (1973-1977)
    • Standard library of 15 operations (mostly) on vectors
      • “AXPY” (y = α·x + y), dot product, scale (x = α·x), etc
      • Up to 4 versions of each (S/D/C/Z), 46 routines, 3300 LOC
  • Goals
    • Common “pattern” to ease programming, readability
    • Robustness, via careful coding (avoiding over/underflow)
    • Portability + Efficiency via machine specific implementations
  • Why BLAS 1? They do O(n^3) ops on O(n^3) data
  • Used in libraries like LINPACK (for linear systems)
    • Source of the name “LINPACK Benchmark” (not the code!)
A brief history of (Dense) Linear Algebra software (2/7)

• But the BLAS-1 weren’t enough
  • Consider AXPY (\(y = \alpha \cdot x + y\)): 2n flops on 3n read/writes
  • Computational intensity = \((2n)/(3n) = 2/3\)
  • Too low to run near peak speed (read/write dominates)
  • Hard to vectorize (“SIMDize”) on supercomputers of the day (1980s)

• So the BLAS-2 were invented (1984-1986)
  • Standard library of 25 operations (mostly) on matrix/vector pairs
    • “GEMV”: \(y = \alpha \cdot A \cdot x + \beta \cdot y\), \(x = T^{-1} \cdot x\)
    • Up to 4 versions of each (S/D/C/Z), 66 routines, 18K LOC
    • Why BLAS 2? They do \(O(n^2)\) ops on \(O(n^2)\) data
    • So computational intensity still just \(\approx (2n^2)/(n^2) = 2\)
      • OK for vector machines, but not for machine with caches

02/26/2015 CS267 Lecture 12

A brief history of (Dense) Linear Algebra software (3/7)

• The next step: BLAS-3 (1987-1988)
  • Standard library of 9 operations (mostly) on matrix/matrix pairs
    • “GEMM”: \(C = \alpha \cdot A \cdot B + \beta \cdot C\), \(B = T^{-1} \cdot B\)
    • Up to 4 versions of each (S/D/C/Z), 30 routines, 10K LOC
  • Why BLAS 3? They do \(O(n^3)\) ops on \(O(n^2)\) data
  • So computational intensity \((2n^3)/(4n^2) = n/2\) – big at last!
    • Good for machines with caches, other mem. hierarchy levels

• How much BLAS1/2/3 code so far (all at www.netlib.org/blas)
  • Source: 142 routines, 31K LOC, Testing: 28K LOC
    • Reference (unoptimized) implementation only
    • Ex: 3 nested loops for GEMM
    • Lots more optimized code (eg Homework 1)
    • Motivates “automatic tuning” of the BLAS
  • Part of standard math libraries (eg AMD ACML, Intel MKL)

02/26/2015 CS267 Lecture 12

A brief history of (Dense) Linear Algebra software (4/7)

• LAPACK – “Linear Algebra PACKage” - uses BLAS-3 (1989 – now)
  • Ex: Obvious way to express Gaussian Elimination (GE) is adding multiples of one row to other rows – BLAS-1
    • How do we reorganize GE to use BLAS-3? (details later)
  • Contents of LAPACK (summary)
    • Algorithms that are (nearly) 100% BLAS 3
      – Linear Systems: solve \(Ax=b\) for \(x\)
    • Least Squares: choose \(x\) to minimize \(||Ax-b||_2||\)
    • Algorithms that are only \(\approx 50\%\) BLAS 3
      – Eigenproblems: Find \(\lambda\) and \(x\) where \(Ax = \lambda x\)
    • Singular Value Decomposition (SVD)
    • Generalized problems (eg \(Ax = \lambda Bx\))
    • Error bounds for everything
    • Lots of variants depending on \(A\)’s structure (banded, \(A=A^\top\), etc)

• How much code? (Release 3.5.0, Nov 2013) (www.netlib.org/lapack)
  • Source: 1740 routines, 704K LOC, Testing: 1096 routines, 467K LOC
  • Ongoing development (at UCB and elsewhere) (class projects!)
    • Next planned release June 2015

02/26/2015 CS267 Lecture 12
A brief history of (Dense) Linear Algebra software (5/7)

- Is LAPACK parallel?
  - Only if the BLAS are parallel (possible in shared memory)
- ScaLAPACK – “Scalable LAPACK” (1995 – now)
  - For distributed memory – uses MPI
  - More complex data structures, algorithms than LAPACK
    - Only (small) subset of LAPACK’s functionality available
    - Details later (class projects!)
- All at www.netlib.org/scalapack

Success Stories for Sca/LAPACK (6/7)

- Widely used
  - Adopted by Mathworks, Cray, Fujitsu, HP, IBM, IMSL, Intel, NAG, NEC, SGI, …
  - 7.5M webhits/year @ Netlib (incl. CLAPACK, LAPACK95)
- New Science discovered through the solution of dense matrix systems
  - Nature article on the flat universe used ScaLAPACK
  - Other articles in Physics Review B that also use it
  - 1998 Gordon Bell Prize
  - www.nersc.gov/news/reports/newNERSResults050703.pdf

A brief future look at (Dense) Linear Algebra software (7/7)

- PLASMA, DPLASMA and MAGMA (now)
  - Ongoing extensions to Multicore/GPU/Heterogeneous
  - Can one software infrastructure accommodate all algorithms and platforms of current (future) interest?
    - How much code generation and tuning can we automate?
  - Details later (Class projects!) (icl.cs.utk.edu/(d)plasma,magma)
- Other related projects
  - Elemental (libelemental.org)
    - Distributed memory dense linear algebra
    - “Balance ease of use and high performance”
  - FLAME (z.cs.utexas.edu/wiki/flame.wiki/FrontPage)
    - Formal Linear Algebra Method Environment
    - Attempt to automate code generation across multiple platforms
  - BLAST Forum (www.netlib.org/blas/blast-forum)
    - Attempt to extend BLAS, add new functions, extra-precision, …

Back to basics:

Why avoiding communication is important (1/3)

Algorithms have two costs:
1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).
Why avoiding communication is important (2/3)

- Running time of an algorithm is sum of 3 terms:
  - $\text{flops} \times \text{time\_per\_flop}$
  - $\frac{\text{words moved}}{\text{bandwidth}}$
  - $\text{messages} \times \text{latency}$
- Time\_per\_flop $\ll \frac{1}{\text{bandwidth}} \ll \text{latency}$
- Gaps growing exponentially with time

<table>
<thead>
<tr>
<th>Time_per_flop</th>
<th>Bandwidth</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>59% DRAM</td>
<td>26%</td>
<td>15%</td>
</tr>
<tr>
<td>23% Network</td>
<td>5%</td>
<td></td>
</tr>
</tbody>
</table>

- Minimize communication to save time

Why Minimize Communication? (3/3)

Minimize communication to save energy

Goal:
Organize Linear Algebra to Avoid Communication

- Between all memory hierarchy levels
  - L1 $\leftrightarrow$ L2 $\leftrightarrow$ DRAM $\leftrightarrow$ network, etc
- Not just *hiding* communication (overlap with arithmetic)
  - Speedup $\leq 2x$
- Arbitrary speedups/energy savings possible
- Later: Same goal for other computational patterns
  - Lots of open problems

Source: John Shalf, LBL
Review: Blocked Matrix Multiply

• Blocked Matmul C = A · B breaks A, B and C into blocks with dimensions that depend on cache size
  ... Break A^{n x n}, B^{n x n}, C^{n x n} into b x b blocks labeled A(i,j), etc
  ... b chosen so 3 b x b blocks fit in cache
  for i = 1 to n/b, for j=1 to n/b, for k=1 to n/b
  C(i,j) = C(i,j) + A(i,k) · B(k,j)
  ... b x b matmul, 4b^2 reads/writes

• When b=1, get "naïve" algorithm, want b larger ...

• \((n/b)^3 \cdot 4b^2 = 4n^3/b\) reads/writes altogether
  • Minimized when \(3b^2 = \) cache size = \(M\), yielding \(O(n^3/M^{1/2})\) reads/writes

• What if we had more levels of memory? (L1, L2, cache etc)?
  • Would need 3 more nested loops per level
  • Recursive (cache-oblivious algorithm) also possible

Communication Lower Bounds: Prior Work on Matmul

• Assume \(n^3\) algorithm (i.e. not Strassen-like)
  • Sequential case, with fast memory of size \(M\)
    • Lower bound on \#words moved to/from slow memory = \(\Omega \left( n^3 / M^{1/2} \right) \) [Hong, Kung, 81]
    • Attained using blocked or cache-oblivious algorithms

• Parallel case on \(P\) processors:
  • Let \(M\) be memory per processor; assume load balanced
    • Lower bound on \#words moved
      = \(\Omega \left( n^3 / (p \cdot M^{1/2}) \right) \) [Irony, Tiskin, Toledo, 04]
    • If \(M = 3n^2/p\) (one copy of each matrix), then
      lower bound = \(\Omega \left( n^2 / p^{1/2} \right) \)
      • Attained by SUMMA, Cannon’s algorithm

New lower bound for all “direct” linear algebra

Let \(M\) = “fast” memory size per processor
  = cache size (sequential case) or \(O(n^2/p)\) (parallel case)
  \#flops = number of flops done per processor

  \#words_moved per processor = \(\Omega(\#flops / M^{1/2})\)
  \#messages_sent per processor = \(\Omega(\#flops / M^{3/2})\)

  • Holds for
    • Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, ...
    • Some whole programs (sequences of these operations, no matter how they are interleaved, eg computing \(A^n\))
    • Dense and sparse matrices (where \#flops \(<< n^3\))
    • Sequential and parallel algorithms
    • Some graph-theoretic algorithms (eg Floyd-Warshall)
    • Generalizations later (Strassen-like algorithms, loops accessing arrays)

SIAM Linear Algebra Prize, 2012

02/26/2015 CS267 Lecture 12 21

New lower bound for all “direct” linear algebra

Let \(M\) = “fast” memory size per processor
  = cache size (sequential case) or \(O(n^2/p)\) (parallel case)
  \#flops = number of flops done per processor

  \#words_moved per processor = \(\Omega(\#flops / M^{1/2})\)
  \#messages_sent per processor = \(\Omega(\#flops / M^{3/2})\)

  • Sequential case, dense \(n \times n\) matrices, so \(O(n^3)\) flops
    • \#words_moved = \(\Omega(n^3/M^{1/2})\)
    • \#messages_sent = \(\Omega(n^3/M^{3/2})\)
  • Parallel case, dense \(n \times n\) matrices
    • Load balanced, so \(O(n^3/p)\) flops processor
    • One copy of data, load balanced, so \(M = O(n^2/p)\) per processor
      • \#words_moved = \(\Omega(n^2/p^{1/2})\)
      • \#messages_sent = \(\Omega( p^{1/2} )\)

02/26/2015 CS267 Lecture 12 22

SIAM Linear Algebra Prize, 2012
Can we attain these lower bounds?

- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Mostly not yet, work in progress
- If not, are there other algorithms that do?
  - Yes

Goals for algorithms:

- Minimize \#words\_moved
- Minimize \#messages\_sent
  - Need new data structures
- Minimize for multiple memory hierarchy levels
  - Cache-oblivious algorithms would be simplest
- Fewest flops when matrix fits in fastest memory
  - Cache-oblivious algorithms don’t always attain this

Attainable for nearly all dense linear algebra

- Just a few prototype implementations so far (class projects!)
- Only a few sparse algorithms so far (e.g., Cholesky)

Outline

- History and motivation
  - What is dense linear algebra?
  - Why minimize communication?
  - Lower bound on communication
- Structure of the Dense Linear Algebra motif
  - What does \( A \backslash b \) do?
  - Parallel Matrix-matrix multiplication
  - Attaining the lower bound
  - Proof of the lower bound (if time)
  - Other Parallel Algorithms (next lecture)

For all linear algebra problems:

Ex: LAPACK Table of Contents

- Linear Systems
- Least Squares
  - Overdetermined, underdetermined
  - Unconstrained, constrained, weighted
- Eigenvalues and vectors of Symmetric Matrices
  - Standard (\( Ax = \lambda x \)), Generalized (\( Ax=\lambda Bx \))
- Eigenvalues and vectors of Unsymmetric matrices
  - Eigenvalues, Schur form, eigenvectors, invariant subspaces
  - Standard, Generalized
- Singular Values and vectors (SVD)
  - Standard, Generalized
- Level of detail
  - Simple Driver
  - Expert Drivers with error bounds, extra-precision, other options
  - Lower level routines ("apply certain kind of orthogonal transformation")
What does $A\backslash b$ do? What could it do?

Ex: LAPACK Table of Contents

- BD – bidiagonal
- GB – general banded
- GE – general
- GG – general, pair
- GT – tridiagonal
- HB – Hermitian banded
- HE – Hermitian
- HG – upper Hessenberg, pair
- HP – Hermitian, packed
- HS – upper Hessenberg
- OR – (real) orthogonal
- OP – (real) orthogonal, packed
- PB – positive definite, banded
- PO – positive definite
- PP – positive definite, packed
- PT – positive definite, tridiagonal
- SB – symmetric, banded
- SP – symmetric, packed
- ST – symmetric, tridiagonal
- SY – symmetric
- TB – triangular, banded
- TG – triangular, pair
- TP – triangular, packed
- TR – triangular
- TZ – trapezoidal
- UN – unitary
- UP – unitary packed
What does A\b do? What could it do?

Ex: LAPACK Table of Contents

- BD – bidiagonal
- GB – general banded
- GE – general
- GG – general, pair
- GT – tridiagonal
- HB – Hermitian banded
- HE – Hermitian
- HG – upper Hessenberg, pair
- HP – Hermitian, packed
- HS – upper Hessenberg
- OR – (real) orthogonal
- OP – (real) orthogonal, packed
- PB – positive definite, banded
- PO – positive definite
- PP – positive definite, packed
- PT – positive definite, tridiagonal
- SB – symmetric, banded
- SP – symmetric, packed
- ST – symmetric, tridiagonal
- SY – symmetric
- TB – triangular, banded
- TG – triangular, pair
- TP – triangular, packed
- TR – triangular
- TZ – trapezoidal
- UN – unitary
- UP – unitary packed

Organizing Linear Algebra – in books

www.netlib.org/lapack

www.netlib.org/scalapack

www.netlib.org/templates

www.cs.utk.edu/~dongarra/etemplates

gams.nist.gov

Different Parallel Data Layouts for Matrices (not all!)

1) 1D Column Blocked Layout
2) 1D Column Cyclic Layout

4) Row versions of the previous layouts

3) 1D Column Block Cyclic Layout

5) 2D Row and Column Blocked Layout
6) 2D Row and Column Block Cyclic Layout

Outline

- History and motivation
  - What is dense linear algebra?
  - Why minimize communication?
  - Lower bound on communication
- Structure of the Dense Linear Algebra motif
  - What does A\b do?
- Parallel Matrix-matrix multiplication
  - Attaining the lower bound
- Other Parallel Algorithms (next lecture)
**Parallel Matrix-Vector Product**

- Compute \( y = y + A^*x \), where \( A \) is a dense matrix
- **Layout:**
  - 1D row blocked
  - \( A(i) \) refers to the \( n \) by \( n/p \) block row that processor \( i \) owns,
- \( x(i) \) and \( y(i) \) similarly refer to segments of \( x, y \) owned by \( i \)
- **Algorithm:**
  - Foreach processor \( i \)
  - Broadcast \( x(i) \)
  - Compute \( y(i) = A(i)*x \)
- Algorithm uses the formula
  \[
  y(i) = y(i) + A(i)^*x = y(i) + \sum_j A(i,j)^*x(j)
  \]

**Matrix-Vector Product \( y = y + A^*x \)**

- A column layout of the matrix eliminates the broadcast of \( x \)
- But adds a reduction to update the destination \( y \)
- A 2D blocked layout uses a broadcast and reduction, both on a subset of processors
  - \( \sqrt{p} \) for square processor grid

**Parallel Matrix Multiply**

- Computing \( C = C + A^*B \)
- Using basic algorithm: \( 2*n^3 \) Flops
- Variables are:
  - Data layout: 1D? 2D? Other?
  - Topology of machine: Ring? Torus?
  - Scheduling communication
- Use of performance models for algorithm design
  - Message Time = "latency" + \#words * time-per-word
    \[
    = \alpha + n^\beta
    \]
- Efficiency (in any model):
  - serial time / (\( p \) * parallel time)
  - perfect (linear) speedup \( \leftrightarrow \) efficiency = 1

**Matrix Multiply with 1D Column Layout**

- Assume matrices are \( n \times n \) and \( n \) is divisible by \( p \)
- \( A(i) \) refers to the \( n \) by \( n/p \) block column that processor \( i \) owns (similarly for \( B(i) \) and \( C(i) \))
- \( B(i,j) \) is the \( n/p \) by \( n/p \) subblock of \( B(i) \)
  - in rows \( j*n/p \) through \( (j+1)*n/p - 1 \)
- Algorithm uses the formula
  \[
  C(i) = C(i) + A^*B(i) = C(i) + \sum j A(j)^*B(j,i)
  \]
Matrix Multiply: 1D Layout on Bus or Ring

- Algorithm uses the formula
  \[ C(i) = C(i) + A*B(i) = C(i) + \sum_j A(j)*B(j,i) \]

- First consider a bus-connected machine without broadcast: only one pair of processors can communicate at a time (ethernet)

- Second consider a machine with processors on a ring: all processors may communicate with nearest neighbors simultaneously

MatMul: 1D layout on Bus without Broadcast

Naïve algorithm:
\[
C(\text{myproc}) = C(\text{myproc}) + A(\text{myproc})*B(\text{myproc}, \text{myproc})
\]
for i = 0 to p-1
  for j = 0 to p-1 except i
    if (myproc == i) send A(i) to processor j
    if (myproc == j)
      receive A(i) from processor i
    C(\text{myproc}) = C(\text{myproc}) + A(i)*B(i,\text{myproc})
  barrier

Cost of inner loop:
- computation: \(2*n^3/p^2\)
- communication: \(\alpha + \beta*n^2/p\)

Only 1 pair of processors (i and j) are active on any iteration, and of those, only i is doing computation
=> the algorithm is almost entirely serial

Running time:
\[
= (p*(p-1) + 1)*\text{computation} + p*(p-1)*\text{communication}
= 2*n^3 + p^2*\alpha + p*n^2*\beta
\]

This is worse than the serial time and grows with p.

Matmul for 1D layout on a Processor Ring

- Pairs of adjacent processors can communicate simultaneously

Naïve MatMul (continued)

Cost of inner loop:
- computation: \(2*n^3/(n/p)^2 = 2*n^3/p^2\)
- communication: \(\alpha + \beta*n^2/p\) \(\ldots\) approximately

Only 1 pair of processors (i and j) are active on any iteration, and of those, only i is doing computation
\=> the algorithm is almost entirely serial

Running time:
\[
= (p*(p-1) + 1)*\text{computation} + p*(p-1)*\text{communication}
= 2*n^3 + p^2*\alpha + p*n^2*\beta
\]

This is worse than the serial time and grows with p.

Matmul for 1D layout on a Processor Ring

- Pairs of adjacent processors can communicate simultaneously
Matmul for 1D layout on a Processor Ring

- Time of inner loop = $2(\alpha + \beta n^2/p) + 2n^2(n/p)^2$
- Total Time = $2n^2(n/p)^2 + (p-1) \cdot \text{Time of inner loop}$
- Total Time = $2n^2(n/p)^2 + (p-1) \cdot (\alpha + \beta n^2/p) + 2n^2(n/p)^2$
- (Nearly) Optimal for 1D layout on Ring or Bus, even with Broadcast:
  - Perfect speedup for arithmetic
  - A(myproc) must move to each other processor, costs at least $(p-1) \cdot \text{cost of sending } n^2(n/p)$ words
- Parallel Efficiency = $2n^3 / (p \cdot \text{Total Time})$
  $= 1/(1 + \alpha \cdot p/(2n^2) + \beta \cdot p/(2n))$
  $= 1/(1 + O(p/n))$
- Grows to 1 as $n/p$ increases (or $\alpha$ and $\beta$ shrink)
- But far from communication lower bound

Need to try 2D Matrix layout

Summary of Parallel Matrix Multiply

- SUMMA
  - Scalable Universal Matrix Multiply Algorithm
  - Attains communication lower bounds (within log $p$)
- Cannon
  - Historically first, attains lower bounds
  - More assumptions
    - A and B square
    - $P$ a perfect square
- 2.5D SUMMA
  - Uses more memory to communicate even less
  - Parallel Strassen
    - Attains different, even lower bounds

SUMMA Algorithm

- SUMMA = Scalable Universal Matrix Multiply
- Presentation from van de Geijn and Watts
  - www.netlib.org/lapack/lawns/lawn96.ps
  - Similar ideas appeared many times
- Used in practice in PBLAS = Parallel BLAS
  - www.netlib.org/lapack/lawns/lawn100.ps
SUMMA uses Outer Product form of MatMul

- C = A*B means C(i,j) = Σk A(i,k)*B(k,j)

- Column-wise outer product:
  \[ C = A*B \]
  \[ = Σk A(:,k)*B(k,:) = Σk (k-th col of A)\*(k-th row of B) \]

- Block column-wise outer product (block size = 4 for illustration)
  \[ C = A*B = A(:,1:4)*B(1:4,:) + A(:,5:8)*B(5:8,:) + \ldots \]
  \[ = Σk (k-th block of 4 cols of A)*\]
  \[ (k-th block of 4 rows of B) \]

SUMMA – n x n matmul on P^{1/2} x P^{1/2} grid

- C[i,j] is n/P^{1/2} x n/P^{1/2} submatrix of C on processor P_{ij}
- A[i,k] is n/P^{1/2} x b submatrix of A
- B[k,j] is b x n/P^{1/2} submatrix of B
- C[i,j] = C[i,j] + Σk A[i,k]*B[k,j]
  - summation over submatrices
- Need not be square processor grid

SUMMA Costs

For k=0 to n-1
  for all i = 1 to P^{1/2}
    owner of A[i,k] broadcasts it to whole processor row (using binary tree)
    \[ \#words = log P^{1/2}*b*n/P^{1/2}, \]
    \[ \#messages = log P^{1/2} \]
    for all j = 1 to P^{1/2}
      owner of B[k,j] broadcasts it to whole processor column (using bin. tree)
      \[ \#words = \#messages \]
      \[ \#flops = 2n^2*b/P \]

For k=0 to n/b-1
  for all i = 1 to P^{1/2}
    owner of A[i,k] broadcasts it to whole processor row (using binary tree)
  for all j = 1 to P^{1/2}
    owner of B[k,j] broadcasts it to whole processor column (using bin. tree)
    \[ \#words = \#messages \]
    Receive A[i,k] into Acol
    Receive B[k,j] into Brow
    \[ C_{myproc} = C_{myproc} + Acol*Brow \]
    \[ \#flops = 2n^2/b/P \]

- Total #words = log P * n^2/P^{1/2}
- Within factor of log P of lower bound
- (more complicated implementation removes log P factor)
- Total #messages = log P * n/b
- Choose b close to maximum, n/P^{1/2}, to approach lower bound P^{1/2}
- Total #flops = 2n^3/P
**PDGEMM = PBLAS routine for matrix multiply**

**Observations:**
- For fixed N, as P increases, Mflops increases, but less than 100% efficiency.
- For fixed P, as N increases, Mflops (efficiency) rises.

**DGEMM = BLAS routine for matrix multiply**

**Maximum speed for PDGEMM = # Procs * speed of DGEMM**

**Observations (same as above):**
- Efficiency always at least 48%.
- For fixed N, as P increases, efficiency drops.
- For fixed P, as N increases, efficiency increases.

---

**2.5D Matrix Multiplication**

- Assume can fit cn²/P data per processor, c > 1.
- Processors form (P/c)² x (P/c)² x c grid.

**Example:** P = 32, c = 2

---

**Can we do better?**

- Lower bound assumed 1 copy of data: M = O(n²/P) per proc.
- What if matrix small enough to fit c>1 copies, so M = cn²/P?
  - #wordsMoved = Ω(#flops / M¹/²) = Ω(n² / (c¹/²P¹/²))
  - #messages = Ω(#flops / M³/²) = Ω(P¹/² /c³/²)
- Can we attain new lower bound?
  - Special case: “3D Matmul”: c = P¹/³
    - Bernstein 89, Agarwal, Chandra, Snir 90, Aggarwal 95
    - Processors arranged in P¹/³ x P¹/³ x P¹/³ grid
    - Processor (i,j,k) performs C(i,j) = C(i,j) + A(i,k)*B(k,j), where each submatrix is n/P¹/³ x n/P¹/³
- Not always that much memory available...
2.5D Matmul on IBM BG/P, n=64K

- As $P$ increases, available memory grows $\Rightarrow$ $c$ increases proportionally to $P$
- #flops, #words\_moved, #messages per proc all decrease proportionally to $P$
- #words\_moved $= \Omega(\text{flops} / \sqrt{M}) = \Omega(n^2 / (c^{1/2}P^{1/2}))$
- #messages $= \Omega(\text{flops} / M^{1/2}) = \Omega(P^{1/2}c^{1/2})$
- Perfect strong scaling! But only up to $c = P^{1/3}$

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 $c$ $\Rightarrow$ total memory increases by a factor of $c$
- Notation for timing model:
  - $Y, \beta, \alpha =$ secs per flop, per word\_moved, per message of size $m$
  - $T(P) = n^3 / (cP) \left[ \gamma + \beta / \sqrt{M} + \alpha / (mM^{1/2}) \right]$
- Notation for energy model:
  - $Y_\epsilon, \beta_\epsilon, \alpha_\epsilon =$ joules for same operations
  - $\delta_\epsilon =$ joules per sec for leakage, etc.
  - $E(P) = cP \left[ n^3 / (cP) \left[ \gamma + \beta_\epsilon / \sqrt{M} + \alpha_\epsilon / (mM^{1/2}) \right] + \delta_\epsilon MT(P) + \epsilon_\epsilon T(P) \right]$
- $c$ cannot increase forever; $c \leq P^{1/3}$ (3D algorithm)
- Corresponds to lower bound on #messages hitting 1
- Perfect scaling extends to Strassen’s matmul, direct N-body, …
- “Perfect Strong Scaling Using No Additional Energy”
- “Strong Scaling of Matmul and Memory-Indep. Comm. Lower Bounds”
- Both at bebop.cs.berkeley.edu
**Classical Matmul**

- Complexity of classical Matmul
- Flops: $O(n^3/p)$
- Communication lower bound on #words:
  $\Omega((n^3/p)/M^{1/2}) = \Omega((n/M^{1/2})^3/p)$
- Communication lower bound on #messages:
  $\Omega((n^3/p)/M^{3/2}) = \Omega((n/M^{1/2})^3/p)$
- All attainable as $M$ increases past $O(n^2/p)$, up to a limit:
  can increase $M$ by factor up to $p^{1/3}$
  #words as low as $\Omega(n/p^{2/3})$

---

**Strong scaling of Matmul on Hopper (n=94080)**

G. Ballard, D., O. Holtz, B. Lipshitz, O. Schwartz

---

**ScaLAPACK Parallel Library**

ScaLAPACK SOFTWARE HIERARCHY

- For each processor that does $G$ flops with fast memory of size $M$
  $\#\text{words}_\text{moved} = \Omega(G/M^{1/2})$
- Extension: for any program that “smells like”
  - Nested loops …
  - That access arrays …
  - Where array subscripts are linear functions of loop indices
    - Ex: $A(i,j), B(3^i-4*k^5), i-j, 2*k, ...,$ …
  - There is a constant $s$ such that
    $\#\text{words}_\text{moved} = \Omega(G/M^{s-1})$
- $s$ comes from recent generalization of Loomis-Whitney ($s=3/2$)
- Ex: linear algebra, n-body, database join, …
- Lots of open questions: deriving $s$, optimal algorithms …

---

**Extensions of Lower Bound and Optimal Algorithms**

- For each processor that does $G$ flops with fast memory of size $M$
  $\#\text{words}_\text{moved} = \Omega(G/M^{1/2})$
- Extension: for any program that “smells like”
  - Nested loops …
  - That access arrays …
  - Where array subscripts are linear functions of loop indices
    - Ex: $A(i,j), B(3^i-4*k^5), i-j, 2*k, ...,$ …
  - There is a constant $s$ such that
    $\#\text{words}_\text{moved} = \Omega(G/M^{s-1})$
- $s$ comes from recent generalization of Loomis-Whitney ($s=3/2$)
- Ex: linear algebra, n-body, database join, …
- Lots of open questions: deriving $s$, optimal algorithms …

---