Lecture 12

More applications
Announcements

• Remember: no class next week
Typical bandwidth curve (SDSC Triton)

\[ N_{1/2} \approx 20 \text{ KB} \]

\[ \alpha = 3.2 \mu \text{sec} \]

Long Messages: $\beta^{-1} \infty \ n \gg \alpha$
Today’s lecture

• Applications
  ❖ Stencil method
    • Multi-tier programming
    • A communication cost model
  ❖ Cannon’s Matrix Multiplication Algorithm
• Parallel Print Function
Stencil method

• Repeat until converged

\[
\text{for } (i,j) \text{ in } 0: N-1 \times 0: N-1 \\
u'[i,j] = \frac{u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1]}{4} \\
u = u'
\]
Parallel Implementation

- Partition data into parts, assigning each to a unique processor
- Dependences on values found on neighboring processes
- “Overlap” or “ghost” cells hold a copies off-processor values

©2012 Scott B. Baden /CSE 260/ Fall 2012
Ghost cells higher dimensions

- Ghost cells surround each local subproblem
- Non-contiguous data
- Inefficient to communicate individual values
Managing ghost cells

- Post **IReceive** ( ) for all neighbors
- **Send** data to neighbors
- **Wait** for completion
Multi-tier execution (MPI+X)

- Use MPI to move the data on behalf of the nodes
- Multicore parallelism on the node, using shared memory

Repeat until converged

Node Communicates ghost cells

#pragma omp parallel for
Sweep u

\[ u = u' \]
Today’s lecture

• Applications
  ◆ Stencil method
    • Multi-tier programming
    • A Communication cost model
  ◆ Cannon’s Matrix Multiplication Algorithm

• Parallel Print Function
Performance is sensitive to processor geometry

• Aliev- Panfilov method running on triton.sdsc.edu Nehalem Cluster)

• 256 cores, n=2047, t=10 (8932 iterations)

<table>
<thead>
<tr>
<th>Geometry</th>
<th>GFlops</th>
<th>Gflops w/o Commun.</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 x 8</td>
<td>573</td>
<td>660</td>
</tr>
<tr>
<td>8 x 32</td>
<td>572</td>
<td>662</td>
</tr>
<tr>
<td>16 x 16</td>
<td>554</td>
<td>665</td>
</tr>
<tr>
<td>2 x 128</td>
<td>508</td>
<td>658</td>
</tr>
<tr>
<td>4 x 64</td>
<td>503</td>
<td>668</td>
</tr>
<tr>
<td>128 x 2</td>
<td>448</td>
<td>658</td>
</tr>
<tr>
<td>256 x 1</td>
<td>401</td>
<td>638</td>
</tr>
</tbody>
</table>
Modeling the parallel running time

- The model has two parts
  - Local computation
  - Communication
- We may ignore the convergence test (check infrequently)
- Communication overheads are due to ghost cell updates
- 2 kinds of geometries: strips vs boxes
Model assumptions and definitions

- $T(1,(m,n)) =$ running time of the best serial algorithm on a problem of size $m \times n$
- $T(P,(m,n)) =$ running time on $P$ processors
- $T_\gamma(P,(m,n)) =$ grind time on $P$ processors
  - $T_\gamma(P,(m,n)) = T(P,(m,n))/(m \cdot n \cdot Niter)$
  - Ideally $T_\gamma$ is independent of $m$, $n$, and $P$
- Following analysis applies to Laplace’s equation, a special case of Poisson’s Equation with $f=0$

$$u'[i,j] = (u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1])/4$$
Completing the Performance model

• Message passing time \( T(N) = \alpha + 8\beta N \)
  (8-byte double precision numbers)
• For the Blue Horizon IBM SP3 system at the San Diego Supercomputer Ctr (with Power3 CPUs)
  \( T_\gamma \approx 16 \beta \)
• Processor geometry is \( p \times q \)
  - Strips or box–like partitions
• \( T(P,(N,N)) = T(1,(m,n)) + T_{\text{local}}^{\text{comm}} \)
  \[ m = \frac{N}{p}, \quad n = \frac{N}{q} \]
Communication costs for 1D geometries

- P divides N evenly
- N/P > 2
- For horizontal strips, data are contiguous
  \[ T_{\text{comm}} = 2(\alpha + 8\beta N) \]
2D Processor geometry

- Assume $\sqrt{P}$ divides $N$ evenly and $N/\sqrt{P} > 2$
- Ignore the cost of packing message buffers
- $T_{\text{comm}} = 4(\alpha + 8\beta N/\sqrt{P})$
Summing up the performance models

• Substituting \( T_\gamma \approx 16 \beta \)
• 1-D decomposition

\[
(16N^2 \beta / P) + 2(\alpha + 8\beta N)
\]

• 2-D decomposition

\[
(16N^2 \beta / P) + 4(\alpha + 8\beta N / \sqrt{P})
\]
Comparative performance

• Strip decomposition will outperform box decomposition …
resulting in lower communication times …
when $2(\alpha + 8\beta N) < 4(\alpha + 8\beta N/\sqrt{P})$

• Assuming $P \geq 2$: $N < (\sqrt{P}/(\sqrt{P} - 2))(\alpha/8\beta)$

• On SDSC’s IBM SP3 system “Blue Horizon”
  \begin{align*}
  \alpha &= 24 \text{ us} \\
  \beta &= 1/(390 \text{ MB/sec})
  \end{align*}

• $N < 1170 (\sqrt{P}/(\sqrt{P} - 2))$

• For $P = 16$, strips are preferable when $N < 2340$
Parallel speedup and efficiency

• 1-D decomposition
  \[ S_P = \frac{T_1}{T_P} = \frac{16N^2\beta}{(16N^2\beta/P + 2(\alpha+8\beta N))} \]
  \[ E_P = \frac{S_P}{P} = \frac{16N^2\beta}{(16N^2\beta + 2P(\alpha+8\beta N))} \]
  \[ = \frac{1}{1 + (\alpha+8\beta N)P/(8N^2\beta))} \]

• 2-D decomposition
  \[ S_P = \frac{T_1}{T_P} = \frac{16N^2\beta}{(16N^2\beta/P+4(\alpha+8\beta N/\sqrt{P}))} \]
  \[ E_P = \frac{S_P}{P} = \frac{16N^2\beta}{((16N^2\beta)+4(\alpha P+8\beta N\sqrt{P}))} \]
  \[ = \frac{1}{1 + (\alpha P+8\beta N\sqrt{P})/(4N^2 \beta))} \]
Putting these formulas to work

• 1-D decomposition
• Let’s plot $E_P$ as a function of $N$, varying $P$ as a parameter
  
  $E_P = \frac{1}{1 + (\alpha+8\beta N)P/ (8N^2\beta)}$

• Let’s also plot the fraction of time spent communicating
Parallel efficiency

N = 1024

N = 128
Communication fraction

![Graph showing communication fraction with N = 128 and N = 1024]
Surface to volume ratio affects performance

- The *surface to volume ratio* of a geometry is the maximum number of points on the surface (perimeter) over all partitions divided by the volume.
- As we increase $N$ while leaving $P$ fixed, we decrease the surface to volume ratio, which gives us a measure of the relative cost of communication.
- As volume increases, $S/V$ drops.
Surface to volume ratio

1 unit of work
4 units of communication

16 units of work
16 units of communication
The curse of dimensionality

- As we move to higher dimensional spaces, communication becomes relatively more costly
  - In 2D: \( \frac{4N}{N^2} = \frac{4}{N} \)
  - In 3D: \( \frac{6N^2}{N^3} = \frac{6}{N} \)
- Large memory strides
Sensitivity to cache interference

- 3D Jacobi’s method
Today’s lecture

• Applications
  - Stencil method
    - Multi-tier programming
    - A Communication cost model
  - Cannon’s Matrix Multiplication Algorithm

• Parallel Print Function
Parallel print function

- Debug output can be hard to sort out on the screen
- Many messages say the same thing
  
  Process 0 is alive!
  Process 1 is alive!
  ...
  Process 15 is alive!

- Compare with

  Processes[0–15] are alive!

- Parallel print facility
  
  http://www.llnl.gov/CASC/ppf
Summary of capabilities

- Compact format list sets of nodes with common output
  
  PPF_Print(MPI_COMM_WORLD, "Hello world");
  0–3: Hello world

- %N specifier generates process ID information
  
  PPF_Print(MPI_COMM_WORLD, "Message from %N\n");
  Message from 0–3

- Lists of nodes
  
  PPF_Print(MPI_COMM_WORLD, (myrank % 2)
    ? "[%N] Hello from the odd numbered nodes!\n"
    : "[%N] Hello from the even numbered nodes!\n")

  [0,2] Hello from the even numbered nodes!
  [1,3] Hello from the odd numbered nodes!
Practical matters

• Installed in $(PUB)/lib/PPF
• Specify ppf=1 on the “make” line
  • Defined in arch.gnu.generic
• Each module that uses the facility must
  
  \#include “ptools_ppf.h”

• Look in $(PUB)/Examples/MPI/PPF for example programs ppfexample_cpp.C and test_print.c
Today’s lecture

• Applications
   Stencil method
    • Multi-tier programming
    • A Communication cost model
   Cannon’s Matrix Multiplication Algorithm

• Parallel Print Function
Matrix Multiplication

• An important core operation in many numerical algorithms

• Given two *conforming* matrices $A$ and $B$, form the matrix product $A \times B$
  - $A$ is $m \times n$
  - $B$ is $n \times p$

• Operation count: $O(n^3)$ multiply-adds for an $n \times n$ square matrix

• See Demmel

  [www.cs.berkeley.edu/~dennmel/cs267_Spr99/Lectures/Lect02.html](http://www.cs.berkeley.edu/~dennmel/cs267_Spr99/Lectures/Lect02.html)
Simplest Serial Algorithm

“ijk”

\[
\begin{align*}
&\text{for } i := 0 \text{ to } n-1 \\
&\quad \text{for } j := 0 \text{ to } n-1 \\
&\quad \quad \text{for } k := 0 \text{ to } n-1 \\
&\quad \quad \quad C[i,j] += A[i,k] \times B[k,j]
\end{align*}
\]
Parallel matrix multiplication

• Assume \( p \) is a perfect square
• Each processor gets an \( \frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}} \) chunk of data
• Organize processors into rows and columns
• Process rank is an ordered pair of integers
• Assume that we have an efficient serial matrix multiply (dgemm, sgemm)
Canon’s algorithm

- Move data incrementally in $\sqrt{p}$ phases
- Circulate each chunk of data among processors within a row or column
- In effect we are using a ring broadcast algorithm
- Consider iteration $i=1, j=2$

Canon’s algorithm

\[ C_{1,2} = A_{1,0} * B_{0,2} + A_{1,1} * B_{1,2} + A_{1,2} * B_{2,2} \]

- We want \( A_{1,0} \) and \( B_{0,2} \) to reside on the same processor initially

- Shift rows and columns so the next pair of values \( A_{1,1} \) and \( B_{1,2} \) line up

- And so on with \( A_{1,2} \) and \( B_{2,2} \)
Skewing the matrices


- We first skew the matrices so that everything lines up
- Shift each row \( i \) by \( i \) columns to the left using sends and receives
- Communication wraps around
- Do the same for each column
Shift and multiply


- Takes \( \sqrt{p} \) steps
- Circularly shift
  - each row by 1 column to the left
  - each column by 1 row to the left
- Each processor forms the product of the two local matrices adding into the accumulated sum
Cost of Cannon’s Algorithm

forall i=0 to √p -1
    CShift-left A[i; :] by i // T= α+βn²/p
forall j=0 to √p -1
    Cshift-up B[: , j] by j // T= α+βn²/p
for k=0 to √p -1
    forall i=0 to √p -1 and j=0 to √p -1
        C[i,j] += A[i,j]*B[i,j] // T = 2*n³/p³/2
        CShift-leftA[i; :] by 1 // T= α+βn²/p
        Cshift-up B[: , j] by 1 // T= α+βn²/p
end forall
end for

T_P = 2n³/p + 2(α(1+√p) + βn²/(1+√p)/p)
E_P = T_1/(pT_P) = ( 1 + αp³/2/n³ + β√p/n ))⁻¹
    ≈ ( 1 + O(√p/n))⁻¹
E_P → 1 as (n/√p) grows [sqrt of data / processor]