Lecture 7

Analytic Performance Modeling

Announcements

• Section and lecture will be swapped on Thursday and Friday
• There will be no lecture on 4/29, instead…
• A make up lecture on Friday 5/7 in place of section
• A2 due today, A3 posted later today
Today’s lecture

- Model of performance for stencil methods
- The curse of dimensionality

Modeling Parallel Performance

- The model has two parts
  - Local computation
  - Communication
- For the stencil method we can ignore the convergence test (collective communication)
- Communication overheads are due to ghost cell updates only
- Let’s start with the 1D ODE solver and then move to higher dimensional spaces
Model assumptions and definitions: 1D case

- $T(1,n) =$ running time of the **best serial algorithm** on a problem of size $n$
- $T(P,n) =$ running time on $P$ processors
- $T_\gamma(P,n) =$ **grind time**
  - Time to perform a single mesh update
  - Helps us normalize with respect to problem size
  - $T_\gamma(P,n0) = T(P,n)/(n \cdot \text{Niter})$

Local Computation time

- $T(1,n) = n \cdot T_\gamma \cdot \text{Niter}$, where $T_\gamma$ is the cost of performing an update
  \[ u_i = \frac{(u_{i+1} + u_{i-1} + h^2 f_i)}{2} \]
- On Valkyrie
  \[ T_\gamma \approx 20\beta \]
- Datum are 8-byte double precision numbers, message passing time = $\alpha + 8\beta N$
More on the Performance model

- Make the naïve assumption that $T(1,n)$ is independent of $n$
- $T_{\text{comm}} = \text{local communication for ghost cells}$
- $T(P,N) = T(1,N/p) + T_{\text{comm}}$
- $= 16N \beta + 2(\alpha + 8\beta)$
  $\approx 16N \beta$

The curse of dimensionality

- As we move to higher dimensional spaces
  - There are alternative partitionings of the problem (processor geometries)
  - Communication involves higher dimensional arrays
  - The relative fraction of communication increases for a fixed number of unknowns $N^D$
- In 1D
  - There is only one possible processor geometry
  - Each process communicates at most 2 points
- In 2D
  - There are 1D and 2D geometries
  - Each process communicates a set of 1D arrays
- In $D$ dimensions
  - $D$ different sets of geometries
  - Each process communicates a set of $(D-1)$ dimensional arrays
Model assumptions and definitions: 2D case

- \( T(1,(m,n)) \) = 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 \( P \)

Decomposing the data

- There are several ways to subdivide data over \( P \) processors
- The processor geometry
  - \( p \times q \), for integers \( p,q \) such that \( P = p \times q \)
- Geometries can be 1- or 2-dimensional
Local Computation time

- $T(1,(m,n)) = mn T_\gamma N_{\text{Iter}}$, where $T_\gamma$ is the cost of performing an update

\[
    u_{i,j} = (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{j,i-1} + h^2 f_j)/4
\]

More on the Performance model

- Assume $T_\gamma(1,(m,n))$
  - is independent of $(m,n)$

- $T(P,(N,N)) = T(1,(N/p,N/q)) + T_{\text{local}}^{\text{comm}}$
Communication performance for 1D

- 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

- 1-D decomposition
  \[ N^2 \beta + 2(\alpha + 8\beta N) \]

- 2-D decomposition
  \[ N^2 \beta + 4(\alpha + 8\beta N/\sqrt{P}) \]

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 that \( P \geq 2 \) we have
  \[ N < (\sqrt{P}/(\sqrt{P} - 2))(\alpha/8\beta) \]
Applying the model

• Assuming that $P \geq 2$ we have
  \[ N < \left( \sqrt{P}/(\sqrt{P} - 2) \right) \frac{\alpha}{8\beta} \]

• Consider a machine with
  \[ \alpha = 24 \text{ us} \]
  \[ \beta = 1/(390 \text{ MB/sec}) \]
  \[ N < 1170 \left( \sqrt{P}/(\sqrt{P} - 2) \right) \]

• For $P = 16$, when $N < 2340$, strips are preferable

Parallel speedup and efficiency

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

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

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

• What is the efficiency for \(N=64\), \(P=8\)?
  0.29

• Let’s plot \(E_P\) as a function of \(N\), varying \(P\) as a parameter

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

Parallel speedup and efficiency
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
High surface to volume ratio

1 unit of work
4 units of communication

Reducing the surface to volume ratio

16 units of work
16 units of communication
Surface to volume ratios in higher dimensions

- In 2D: \( \frac{4N}{N^2} = \frac{4}{N} \)
- In 3D: \( \frac{6N^2}{N^3} = \frac{6}{N} \)

Refinements to the performance model

- We ignored some details
- The grind time is sensitive to the aspect ratio of the local grid
Other refinements to the performance model

- Transmitting data that are not contiguous in memory is more expensive than transmitting contiguous data
  - This effect is usually more pronounced in 3D
  - Why?
- We have to pack non-contiguous data into a buffer before transmitting, or use MPI datatypes

Stride increases as we enlarge the array

The stride in 2D = distance between elements in the same column but in successive rows

Strides are generally larger in 3D
### Some results in 3D on an IBM SP2 with 16 processors

<table>
<thead>
<tr>
<th>Geometry</th>
<th>MF/s</th>
<th>Time</th>
<th>Comm</th>
<th>MF/s</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16x 1x1</td>
<td>510</td>
<td>15.80</td>
<td>2.41</td>
<td>601</td>
<td>13.39</td>
</tr>
<tr>
<td>1x16x 1</td>
<td>655</td>
<td>12.29</td>
<td>2.43</td>
<td>816</td>
<td>9.87</td>
</tr>
<tr>
<td><strong>1x 1x 16</strong></td>
<td><strong>547</strong></td>
<td><strong>14.72</strong></td>
<td><strong>8.33</strong></td>
<td><strong>1260</strong></td>
<td><strong>6.39</strong></td>
</tr>
<tr>
<td>4x 4x 1</td>
<td>611</td>
<td>13.18</td>
<td></td>
<td>656</td>
<td>12.27</td>
</tr>
<tr>
<td>4x 1x 4</td>
<td>674</td>
<td>11.94</td>
<td>2.71</td>
<td>872</td>
<td>9.23</td>
</tr>
<tr>
<td>1x 4x 4</td>
<td>621</td>
<td>12.96</td>
<td>2.77</td>
<td>790</td>
<td>10.19</td>
</tr>
<tr>
<td>8x 2x 1</td>
<td>554</td>
<td>14.54</td>
<td>1.43</td>
<td>614</td>
<td>13.11</td>
</tr>
<tr>
<td>8x 1x 2</td>
<td>619</td>
<td>13.00</td>
<td>1.21</td>
<td>683</td>
<td>11.79</td>
</tr>
<tr>
<td>4x 2x 2</td>
<td>629</td>
<td>12.81</td>
<td>1.10</td>
<td>688</td>
<td>11.71</td>
</tr>
<tr>
<td>2x 4x 2</td>
<td>629</td>
<td>12.81</td>
<td>1.07</td>
<td>686</td>
<td>11.74</td>
</tr>
<tr>
<td>2x 2x 4</td>
<td>671</td>
<td>12.00</td>
<td>2.48</td>
<td>846</td>
<td>9.52</td>
</tr>
<tr>
<td>2x 8x 1</td>
<td>669</td>
<td>12.03</td>
<td>1.39</td>
<td>757</td>
<td>10.64</td>
</tr>
<tr>
<td>2x 1x 8</td>
<td>617</td>
<td>13.06</td>
<td>4.40</td>
<td>931</td>
<td>8.65</td>
</tr>
<tr>
<td>1x 2x 8</td>
<td>560</td>
<td>14.37</td>
<td>4.43</td>
<td>810</td>
<td>9.95</td>
</tr>
<tr>
<td><strong>1x 8x 2</strong></td>
<td><strong>693</strong></td>
<td><strong>11.63</strong></td>
<td><strong>1.84</strong></td>
<td><strong>823</strong></td>
<td><strong>9.79</strong></td>
</tr>
</tbody>
</table>

Try this out yourself

- Code on valkyrie
  ```
  ~/../public/examples/redblack3D
  ```

- Explanation of the code linked into the last lecture’s web page