Parallel Programming Workshop

Brought to you by

Le Yan, Bhupendar Thakar, Jim Lupo, Alex Pacheco
Registration

- Please make sure you're signed in.
- Won't need a computer this morning
  - unless you need a calculator to add integers
Plan Of Action

- **Monday Morning**
  - Parallel Concepts

- **Monday Afternoon**
  - Define an example problem
  - Work with serial code
  - Develop pseudo programming / parallel description

- **Tuesday**
  - Intro To MPI
  - Map psuedo program to MPI calls
  - Template parallel example

- **Wednesday**
  - Performance analysis
Concepts

- Throw out some terminology
- Manual exercises to identify concepts
- Capture concepts on the board
- Relate them to program analysis when appropriate.
Parallel Programming Models

Based on the problem type, there may be one method of parallel programming preferred over another:

- Distributed Memory
- Shared Memory
- Hybrid

Often dictated by the architecture of a specific machine, but any method possible on any machine.
Exercise 1

• The data set: 5 numbers on a card.
• Desired analysis: summation
• Any volunteer?
• Anyone want to play time keeper?

Conceptually, what process was followed?
Ex 1 Outcomes

- A task – some unit of work, here summing 1 card.
  - If the work is being done on a computer it could be handled in:
    - program / process
    - thread

- Communication in the form of input and output. May be:
  - Implicit, as in a serial program.
  - Explicit, as we'll see eventually in parallel programs.

- Just basic stuff – work to be done and data to be worked on.
Exercise 2

- Make life a little harder: 4 cards, 5 numbers each.
- Someone want to play time keeper?
Ex. 2 Outcomes

• Break down the activity involved.
• Anything different?
• How many tasks?
• How do the input and output compare with Ex 1?
Alternate Approach 1

- Again, 4 cards to sum.
- Tell me when you are done with a card, and I'll give you the next card.
- Time keeper alert.
Alternate Outcomes

- Example of “master / slave” task distribution.
- Overhead – extra work required to handle the coordination.
  - More communication for coordination.
  - More data movement – same amount of data, but note additional activity to handle 1 card at a time – communication latency.
Exercise 3

- Try it again, only with two volunteer adders.
1st Alternate Approach

- Hand out 1 card at a time
2nd Alternate Approach

- “Broadcast” 4 cards to each adder.
- Adder 1 does first two cards.
- Adder 2 does last two cards.
Exercise 3 Outcomes

• Introduce new terminology:
  • Number of adders – the *size* of the adder pool.
  • The ID of an adder – the *rank* of an adder.

• Adders must be able to identify tasks.
  • How to determine even/odd (card set on each card).
  • How to stop looking for tasks when all are consumed.
Concept Summary

- What programming models?
- Types of parallelism?
- Planning requirements?
Example of *sharing memory* simply because you **all** can see **all** the data.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>13</td>
<td>78</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>49</td>
<td>60</td>
<td>138</td>
<td>34</td>
<td>79</td>
</tr>
<tr>
<td>3</td>
<td>59</td>
<td>108</td>
<td>139</td>
<td>188</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>137</td>
<td>50</td>
<td>4</td>
<td>167</td>
<td>189</td>
</tr>
<tr>
<td>5</td>
<td>83</td>
<td>136</td>
<td>215</td>
<td>26</td>
<td>140</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>187</td>
<td>77</td>
<td>216</td>
<td>51</td>
</tr>
</tbody>
</table>
Ex 4 Outcomes

- Benefits?
- Difficulties?
Concept Summary

- Shared memory lets all processors see all data.
- Shared Memory Model is growing in popularity as more cores per node become available, and new devices such as GPUs become common place.
- Hybrid or Heterogeneous models are becoming important as the needed to combine Shared and Distributed models increase.
# Parallel Thinking

- What kind of questions do you need to consider when approaching a new problems?

<table>
<thead>
<tr>
<th>Task</th>
<th>Overhead</th>
<th>Distribution</th>
<th>Load Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (Workers)</td>
<td>Rank (Position)</td>
<td>Scalability</td>
<td>Synchronize</td>
</tr>
<tr>
<td>Correctness</td>
<td>Overlap</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Break
The Laplace Heat Equation

• For a real problem, consider how to go about solving the Laplace Heat Equation in 2-D. Idea is to determine the temperature at any point on a surface, given the temperature at the boundaries:
Formal Solution

The solution must satisfy:

$$\nabla^2 \phi = 0$$

with the application of Dirichlet boundary conditions (constant values around edge of region.)
The Serial Solution

Subdivide the surface into a mesh of points.

Apply the following 5-point stencil iteratively until the temperature stops changing (new temp approximates old temp):

\[ T_{i,j}^{n+1} = 0.25 \left( T_{i-1,j}^n + T_{i+1,j}^n + T_{i,j-1}^n + T_{i,j+1}^n \right) \]
Serial Program

• Grab a copy of the program named:
  /work/jalupo/laplace_solver_serial.f90

• Open with “less” or “vi” so you can follow along.

• Anyone have trouble reading Fortran?

• Anyone not know how to compile and run a Fortran program?
Main Components

- **program laplace_main** – program main line.
- **subroutine laplace** – the actual solver. It also allocates memory to hold the 2-D mesh based on the requested rows and columns.
- **subroutine initialize** – sets the internal temperatures to 0.
- **subroutine set_bcs** – sets up the boundary conditions.
Compiling Fortran

• Here is a quick summary of how to compile and run this particular program (assumes default environment):
  
  $ ifort -o laplace laplace_solver_serial.f90
  $ ./laplace

• You should see the following line of text on your screen:
  
  Usage: laplace nrows ncols niter iprint relerr

Now try executing the program with some real numbers:

  $ ./laplace 100 200 3000 300 0.001
Decomposition

- Assuming 2 processors, let's divide the surface in half.

What overhead do we have to consider adding to make this give the same answer?
Ghost Cells

Process 1

Process 2
Overhead

• Breaking up the problem so multiple processes can work on it introduces *overhead*:
  • Logic must be added so each process knows which part of the mesh it is expected to work on. This directly impacts how the code will start up.
  • Communication must be added so data from adjoining regions can be properly updated.
  • Code must be added so the final results can be communicated. This directly impacts how the code will report results and terminate.

• A serial program is not the same as a parallel program running on 1 processor!
Domination

- Clearly, if you increase the number of processes working on this problem, the amount of communication required increases.
- With a few processes, this problem exhibits the property of being *compute dominated*.
- When the number of processes approach the number of mesh points, it becomes *communication dominated*.
- All parallel programs exhibit one form or the other depending on the problem specifics.
LUNCH
Parallel Thinking

- What kind of questions do you need to consider when approaching a new problems?

<table>
<thead>
<tr>
<th>Task</th>
<th>Overhead</th>
<th>Distribution</th>
<th>Load Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (Workers)</td>
<td>Rank (Position)</td>
<td>Scalability</td>
<td>Synchronize</td>
</tr>
<tr>
<td>Correctness</td>
<td>Overlap</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>