Real Time Hybrid Simulation on a Parallel Machine

Gregory Bunting, Payton Lindsay
Overview

- Introduction to RTHS
- RTHS Example
- My requirements for Parallel Code
- Structure of RTHS Code
- Parallel Transformations
- Execution time and Speedups
- Conclusions
Real Time Hybrid Simulation

- Full scale tests can be expensive
- Hybrid Simulation: Physical + Numerical
- Real Time Hybrid Simulation:
  - Accurate dynamics
  - Typically performed at 1024 Hz
  - Constraints on numerical model size
Real Time Hybrid Simulation
RTHS Example

- Ghannoum and Moehle (2012) UC Berkeley
RTHS Example

- 19 stirrups @ 3 1/2" c/c (typical all beams)
- 8 #2 long. bars, 5/8" dia. wire ties (typical all floors)
- 8 ties @ 1 1/4" c/c
- 13 ties equally spaced
- 8 ties @ 1 1/4" c/c
- 6" 5 ties @ 4" c/c
- 6" 10 ties @ 4" c/c
- 18" 5 ties @ 4" c/c

• Ghannoum and Moehle (2012) UC Berkeley
RTHS Example

- Saoumo et al 2013 UC Boulder
My Requirements for Parallel Code

- Need to meet 1024 Hz requirement
- Size of numerical model is unknown – the bigger numerical model we can handle, the more interesting tests we can run
- Will eventually run on real time machine
- Will run as part of a physical test
  - Shake Table
  - Actuators
  - Accelerometers
Structure of RTHS Code

- Earthquake input (El Centro, 60s)
  - Gains to adjust units
- Numerical Substructure
  - State Space or FEA Representation
- Compensation / Control
- Actuator Dynamics
- Randomly Generated Noise Values
- Noise Filter
- Physical Substructure (Pure Stiffness)
Structure of Code

- Preprocessing – building of state space matrices – generally ignored because this can be done before Real Time section starts
- Large Loop – Time stepping
  - Not parallelizable (Need previous results)
  - Must be solved in less than .976 milliseconds
  - Bulk of computation will be in NS
- Postprocessing – graphing results, etc. generally ignored because this can be done after Real Time Section
Structure of Code
Parallel Transformations

- Parallel for loops and loop collapsing—just on NS solve
- Parallel for loops on other solves had high overhead
- Parallel sections – splitting NS from the rest of the time stepping code and adding a unit delay
  - High overhead for this problem
  - May use if we have more complicated control techniques or other calculations to do
Timing Data

275 Dof Problem on Kepler

Execution Time (milliseconds)

<table>
<thead>
<tr>
<th>Core Configuration</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>1.00</td>
</tr>
<tr>
<td>1 Core</td>
<td>0.50</td>
</tr>
<tr>
<td>2 Cores</td>
<td>0.25</td>
</tr>
<tr>
<td>4 Cores</td>
<td>0.13</td>
</tr>
<tr>
<td>8 Cores</td>
<td>0.07</td>
</tr>
<tr>
<td>16 Cores</td>
<td>0.04</td>
</tr>
<tr>
<td>24 Cores</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Timing Data

275 Dof Problem on Andromeda

Execution Time (milliseconds)

<table>
<thead>
<tr>
<th>Serial</th>
<th>1 Core</th>
<th>2 Cores</th>
<th>4 Cores</th>
<th>8 Cores</th>
<th>16 Cores</th>
<th>24 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.0</td>
<td>2.0</td>
<td>1.5</td>
<td>1.0</td>
<td>0.5</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Red line represents a threshold.
Timing Data

275 Dof Problem on Hephaestus

Execution Time (milliseconds)

Serial | 1 Core | 2 Cores | 4 Cores | 8 Cores | 16 Cores | 24 Cores

0 | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 | 3.5
Timing Data

800 Dof Problem on Kepler

Execution Time (milliseconds)

<table>
<thead>
<tr>
<th>Serial</th>
<th>1 Core</th>
<th>2 Cores</th>
<th>4 Cores</th>
<th>8 Cores</th>
<th>16 Cores</th>
<th>24 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Timing Data

800 Dof Problem on Andromeda

Execution Time (milliseconds)

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>1 Core</th>
<th>2 Cores</th>
<th>4 Cores</th>
<th>8 Cores</th>
<th>16 Cores</th>
<th>24 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (ms)</td>
<td>25</td>
<td>35</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
Timing Data

800 Dof Problem on Hephaestus

Execution Time (milliseconds)

Serial   1 Core   2 Cores   4 Cores   8 Cores   16 Cores   24 Cores
Speedups

- Kepler 800 Dof Problem
- Kepler 275 Dof Problem
- Andromeda 800 Dof Problem
- Andromeda 275 Dof Problem
- Hephaestus 800 Dof Problem
- Hephaestus 275 Dof Problem

Graph showing speedups over a range of values.
Conclusions

- Parallel Code Meets our requirements – increases the size of the NS that we are able to solve
- This code will be used to run a RTHS in the future
- Even though our speedups are not great, it works well for our application
Future Work

- Implement code in RT framework on a RT kernel
- Add ability to interact with sensors and control devices
- Possibly use parallel sections if we have another large computation part
- Run a RTHS with this setup