Load Balancing and Patch-Based Parallel Adaptive Mesh Refinement for Tsunami Simulation on Heterogeneous Platforms using Xeon Phi Coprocessors

PASC 2017

Chaulio Ferreira*, Michael Bader
Technical University of Munich

June 26th, 2017

* Supported by CNPq/Brazil
sam(oa)$^2$ framework

- sam(oa)$^2$ - Space-filling curves and Adaptive Meshes for Oceanic And Other Applications
- A parallel framework for solving 2D PDEs
- Dynamically adaptive triangular meshes
- Data storage and traversals based on the Sierpinski curve
- Hybrid MPI+OpenMP parallelization also based on the curve
Tsunami simulations – Tohoku 2011
Salomon supercomputer

- Hosted by the IT4Innovations National Supercomputing Centre, Czech Republic
- Nodes with a dual-socket Haswell system and two Xeon Phi coprocessors
Part 1: Native mode
Intel® Xeon Phi™ coprocessors

<table>
<thead>
<tr>
<th>Intel® Xeon Phi™</th>
<th>7120P (KNC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores (max threads)</td>
<td>61 (244)</td>
</tr>
<tr>
<td>Clock rate</td>
<td>1.24 GHz</td>
</tr>
<tr>
<td>SIMD vector length</td>
<td>512-bit</td>
</tr>
<tr>
<td>Peak perform. (DP)</td>
<td>1.21 TFlops/s</td>
</tr>
<tr>
<td>Peak bandwidth</td>
<td>352 GB/s</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
</tr>
</tbody>
</table>

• It is not trivial to use vectorization in our complex and dynamic mesh.
• Proposed solution: add a static refinement layer where it is possible to add vectorization.
## Intel® Xeon Phi™ coprocessors

<table>
<thead>
<tr>
<th>Intel® Xeon Phi™ 7120P (KNC)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores (max threads)</td>
<td>61 (244) → Many-core parallelization ✔</td>
</tr>
<tr>
<td>Clock rate</td>
<td>1.24 GHz</td>
</tr>
<tr>
<td>SIMD vector length</td>
<td>512-bit</td>
</tr>
<tr>
<td>Peak perform. (DP)</td>
<td>1.21 TFlops/s</td>
</tr>
<tr>
<td>Peak bandwidth</td>
<td>352 GB/s</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
</tr>
</tbody>
</table>
### Intel® Xeon Phi™ coprocessors

<table>
<thead>
<tr>
<th>Intel® Xeon Phi™ 7120P (KNC)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores (max threads)</td>
<td>61 (244) → Many-core parallelization ✓</td>
</tr>
<tr>
<td>Clock rate</td>
<td>1.24 GHz</td>
</tr>
<tr>
<td>SIMD vector length</td>
<td>512-bit → Vectorization x</td>
</tr>
<tr>
<td>Peak perform. (DP)</td>
<td>1.21 TFlops/s</td>
</tr>
<tr>
<td>Peak bandwidth</td>
<td>352 GB/s</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
</tr>
</tbody>
</table>

It is not trivial to use vectorization in our complex and dynamic mesh. Proposed solution: add a static refinement layer where it is possible to add vectorization.
Intel® Xeon Phi™ coprocessors

<table>
<thead>
<tr>
<th>Intel® Xeon Phi™ 7120P (KNC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores (max threads)</td>
</tr>
<tr>
<td>Clock rate</td>
</tr>
<tr>
<td>SIMD vector length</td>
</tr>
<tr>
<td>Peak perform. (DP)</td>
</tr>
<tr>
<td>Peak bandwidth</td>
</tr>
<tr>
<td>Memory</td>
</tr>
</tbody>
</table>

- It is not trivial to use vectorization in our complex and dynamic mesh.
Intel® Xeon Phi™ coprocessors

<table>
<thead>
<tr>
<th>Intel® Xeon Phi™ 7120P (KNC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores (max threads)</td>
</tr>
<tr>
<td>Clock rate</td>
</tr>
<tr>
<td>SIMD vector length</td>
</tr>
<tr>
<td>Peak perform. (DP)</td>
</tr>
<tr>
<td>Peak bandwidth</td>
</tr>
<tr>
<td>Memory</td>
</tr>
</tbody>
</table>

- It is not trivial to use vectorization in our complex and dynamic mesh.
- Proposed solution: add a static refinement layer where it is possible to add vectorization.
Replacing cells with regular patches

- Each cell becomes a patch of regularly refined cells
Replacing cells with regular patches

- Example mesh - 32 cells
Replacing cells with regular patches

- Example mesh - 32 cells (8 patches with $2^2$ cells)
Applying vectorization to the patches

- All edges in a patch can be processed using SIMD vector instructions.
  
  (i) Cell data is copied to temporary arrays that represent the edges.

![Diagram of cells and edges](image)

(h, hu, hv, b)
Applying vectorization to the patches

- All edges in a patch can be processed using SIMD vector instructions.
  (i) Cell data is copied to temporary arrays that represent the edges
  (ii) Then all edges are processed by a vectorized solver

(i)

(ii)

(iii) Finally, the computed updates are used to update the cell data

C. Ferreira et al. | Load Balancing and Patch-Based pAMR for Tsunami Simulation on Xeon Phi coprocessors | PASC17 | 27.06.2017 | 10
Applying vectorization to the patches

- All edges in a patch can be processed using SIMD vector instructions.
  (i) Cell data is copied to temporary arrays that represent the edges
  (ii) Then all edges are processed by a vectorized solver
  (iii) Finally, the computed updates are used to update the cell data
Native mode: Performance results

- Using two Haswells vs. a single Xeon Phi:

![Performance Chart]

- Cell updates/sec
- Performance
- Haswells
- Xeon Phi

C. Ferreira et al. | Load Balancing and Patch-Based pAMR for Tsunami Simulation on Xeon Phi coprocessors | PASC17 | 27.06.2017 | 11
Native mode: Performance results

- However, using large patches increases the number of cells in the mesh.
Native mode: Performance results

- Time to solution is a better metric in this case.
Native mode: Performance results

- Reduction of the mesh complexity contributes greatly to the speedups:

Simulation components

<table>
<thead>
<tr>
<th>Machine/Patch size</th>
<th>Avg. thread time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haswells/1</td>
<td>200</td>
</tr>
<tr>
<td>Haswells/8²</td>
<td>100</td>
</tr>
<tr>
<td>Phi/1</td>
<td>300</td>
</tr>
<tr>
<td>Phi/8²</td>
<td>100</td>
</tr>
</tbody>
</table>

- Time step
- Adaptation
- Distances
- Neighbors
- Conformity
Part 2: Symmetric mode
Symmetric mode: Initial results

Performance

Cell updates/sec

10^6

Haswells 1 Phi 2 Phis Full node
First problem: slow MPI communication with Phis

![Intel® MPI benchmark graph]

- **Host-Host**: Blue dots
- **Host-Phi**: Red squares
- **Phi-Phi**: Green triangles
Second problem: homogeneous load balancing

Phi 1  Haswell 1  Haswell 2  Phi 2

C. Ferreira et al. | Load Balancing and Patch-Based pAMR for Tsunami Simulation on Xeon Phi coprocessors | PASC17 | 27.06.2017 | 18
Second problem: homogeneous load balancing

Phi 1  Haswell 1  Haswell 2  Phi 2

25%  25%  25%  25%

25%  25%  25%  25%
Second problem: homogeneous load balancing
Heterogeneous load balancing

- So, we changed our load balancing implementation to also allow heterogeneous distributions.
Heterogeneous load balancing

- So, we changed our load balancing implementation to also allow heterogeneous distributions.
- But how much load should we give to the Haswells and how much to the Phis?

Experimentally, we found that for this specific system and for this specific simulation, the best performance is achieved by giving 18% to each Haswell and 32% to each Phi.

But what about other systems? And other simulations?

⇒ We also implemented a "auto-tuning" approach, where the simulation code iteratively chooses the distribution based on statistics from previous time steps.

⇒ For that we defined the efficiency of each processor as:

$$Eff = \frac{load}{time}.$$
Heterogeneous load balancing

- So, we changed our load balancing implementation to also allow heterogeneous distributions.
- But how much load should we give to the Haswells and how much to the Phis?
- Experimentally, we found that for this specific system and for this specific simulation, the best performance is achieved by giving 18% to each Haswell and 32% to each Phi.
Heterogeneous load balancing

- So, we changed our load balancing implementation to also allow heterogeneous distributions.
- But how much load should we give to the Haswells and how much to the Phis?
- Experimentally, we found that for this specific system and for this specific simulation, the best performance is achieved by giving 18% to each Haswell and 32% to each Phi.
- But what about other systems? And other simulations?
Heterogeneous load balancing

- So, we changed our load balancing implementation to also allow heterogeneous distributions.
- But how much load should we give to the Haswells and how much to the Phis?
- Experimentally, we found that for this specific system and for this specific simulation, the best performance is achieved by giving 18% to each Haswell and 32% to each Phi.

- But what about other systems? And other simulations?
  - We also implemented a “auto-tuning” approach, where the simulation code iteratively chooses the distribution based on statistics from previous time steps.
  - For that we defined the efficiency of each processor as: \( \text{Eff} = \frac{\text{load}}{\text{time}} \).
Symmetric mode: Single node

Performance

Cell updates/sec

Haswells  1 Phi  2 Phis  Homog.  Manual  Auto

Performance improvements:
+26%  +20%
Symmetric mode: Multiple nodes (weak scaling)

Performance per node

Nodes

Cell updates/sec/node

10^6

120

100

90

80

70

60

50

40

30

20

10

Only Haswells

Only Phis

Symm./Homog.

Symm./Manual

Symm./Automatic

C. Ferreira et al. | Load Balancing and Patch-Based pAMR for Tsunami Simulation on Xeon Phi coprocessors | PASC17 | 27.06.2017
Final remarks

- Patches allow vectorization and also reduce the complexity of adaptive meshes.
- When choosing the patch size, a trade-off between the increases in performance and in the mesh size must be found.
Final remarks

- Patches allow vectorization and also reduce the complexity of adaptive meshes.
- When choosing the patch size, a trade-off between the increases in performance and in the mesh size must be found.
- Symmetric mode can be faster than other modes when heterogeneous load balancing is used.
- But its benefits are restricted to a small number of nodes, because MPI communication with the Xeon Phis is considerably slow.
Final remarks

- Patches allow vectorization and also reduce the complexity of adaptive meshes.
- When choosing the patch size, a trade-off between the increases in performance and in the mesh size must be found.
- Symmetric mode can be faster than other modes when heterogeneous load balancing is used.
- But its benefits are restricted to a small number of nodes, because MPI communication with the Xeon Phis is considerably slow.
- Heterogeneous HPC systems containing accelerator-type devices are becoming increasingly common.
- Thus, many HPC applications will require heterogeneous load balancing for keeping up with the modern hardwares.
References


Acknowledgements

IT4Innovations
national supercomputing center

CNPq