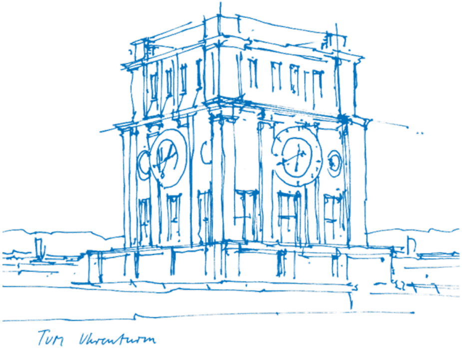
Load Balancing and Patch-Based Parallel Adaptive Mesh Refinement for Tsunami Simulation on Heterogeneous Platforms using Xeon Phi Coprocessors

PASC 2017

Chaulio Ferreira*, Michael Bader
Technical University of Munich

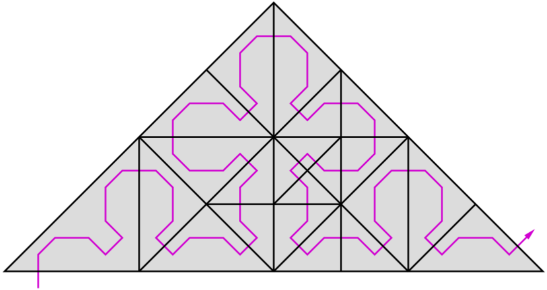
June 26th, 2017



* Supported by CNPq/Brazil

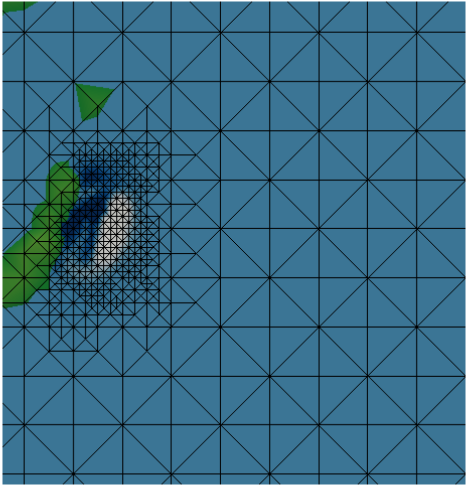
sam(oa)^2 framework

- 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



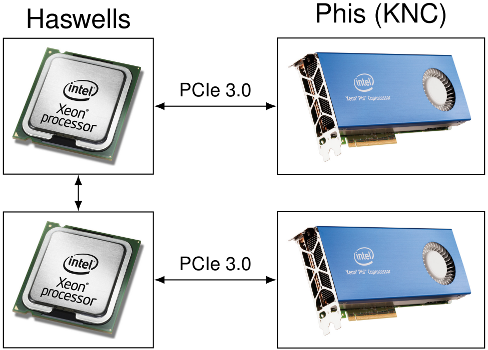
Tsunami simulations – Tohoku 2011

Tsunami simulations – Tohoku 2011

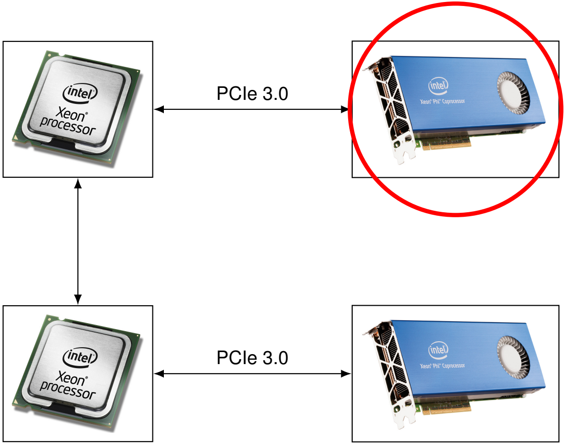


Salomon supercomputer

- 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

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

Intel® Xeon Phi™ coprocessors

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

Intel® Xeon Phi™ coprocessors

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

Intel® Xeon Phi™ coprocessors

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

- It is not trivial to use vectorization in our complex and dynamic mesh.

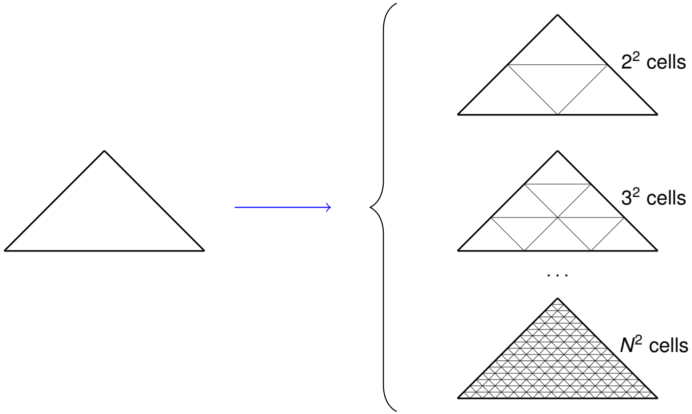
Intel® Xeon Phi™ coprocessors

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

- 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.

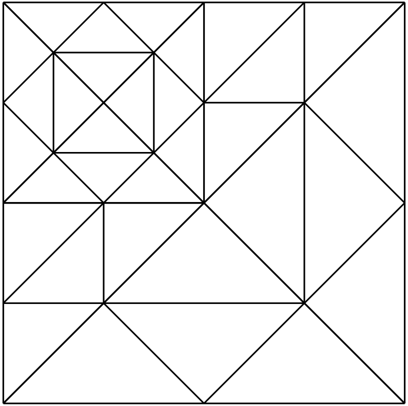
Replacing cells with regular patches

- Each cell becomes a patch of regularly refined cells



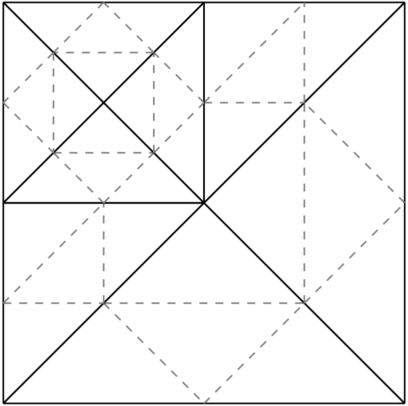
Replacing cells with regular patches

- Example mesh - 32 cells



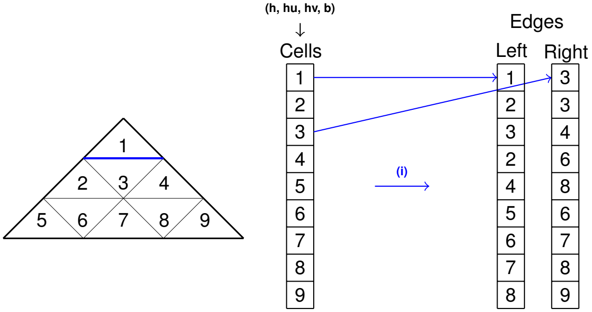
Replacing cells with regular patches

- Example mesh - 32 cells (8 patches with 2^2 cells)



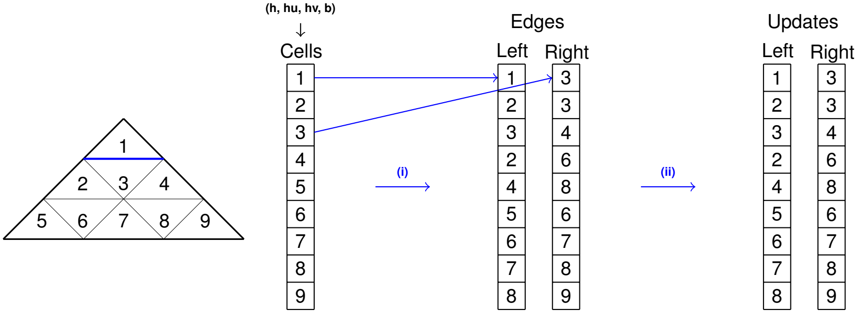
Applying vectorization to the patches

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



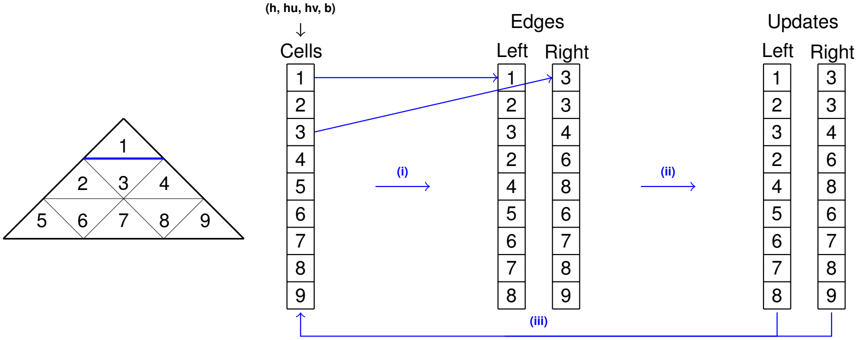
Applying vectorization to the patches

- 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



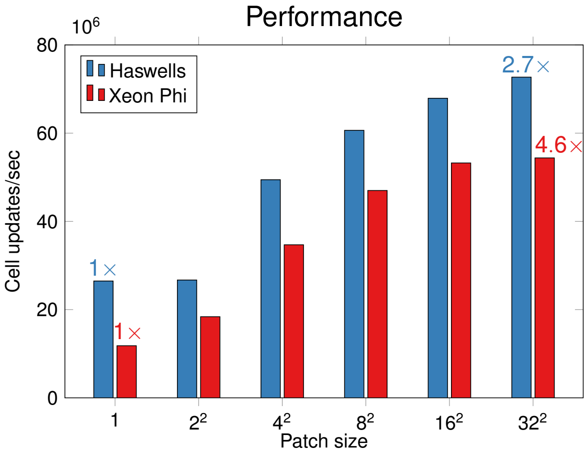
Applying vectorization to the patches

- 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



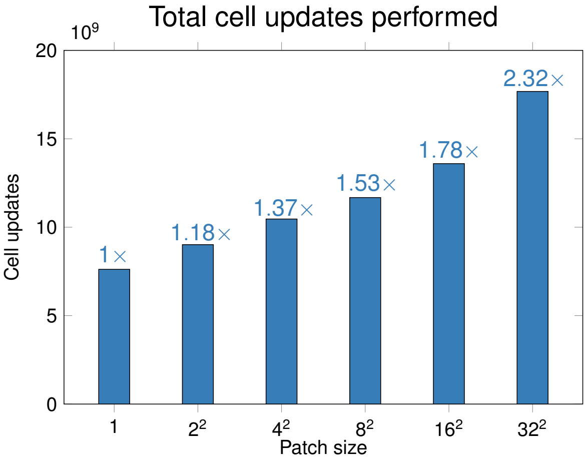
Native mode: Performance results

- Using two Haswells vs. a single Xeon Phi:



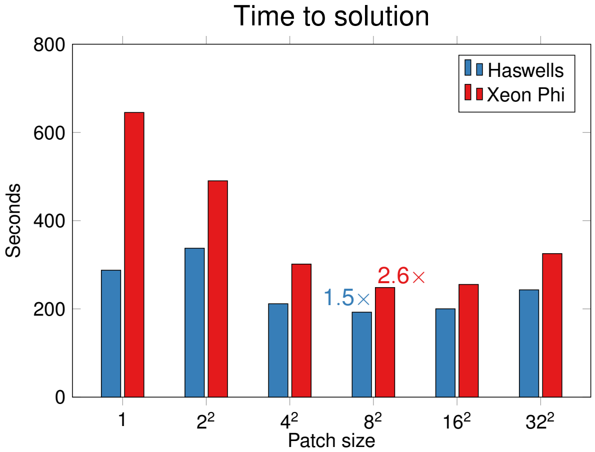
Native mode: Performance results

- However, using large patches increases the number of cells in the mesh.



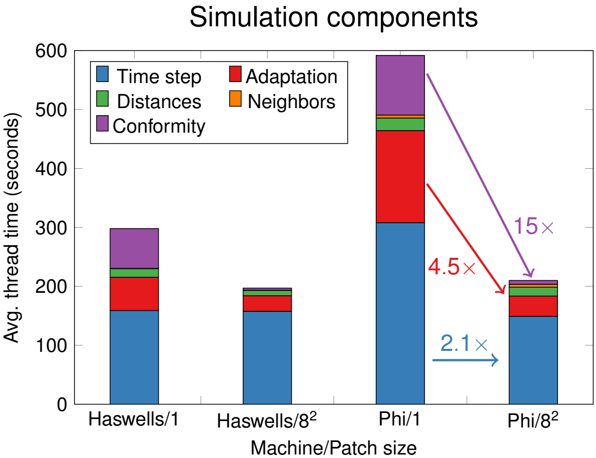
Native mode: Performance results

- Time to solution is a better metric in this case.

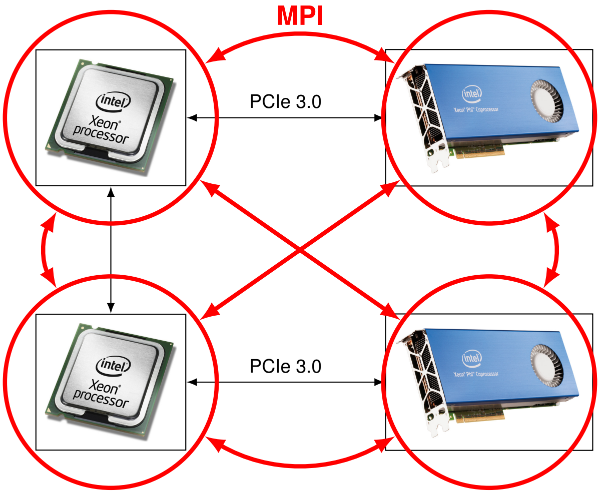


Native mode: Performance results

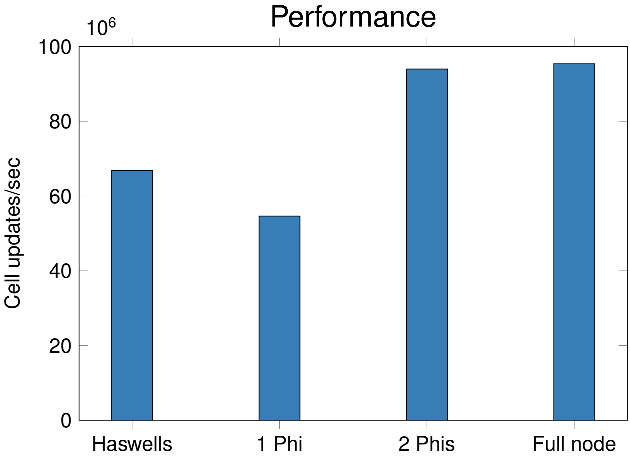
- Reduction of the mesh complexity contributes greatly to the speedups:



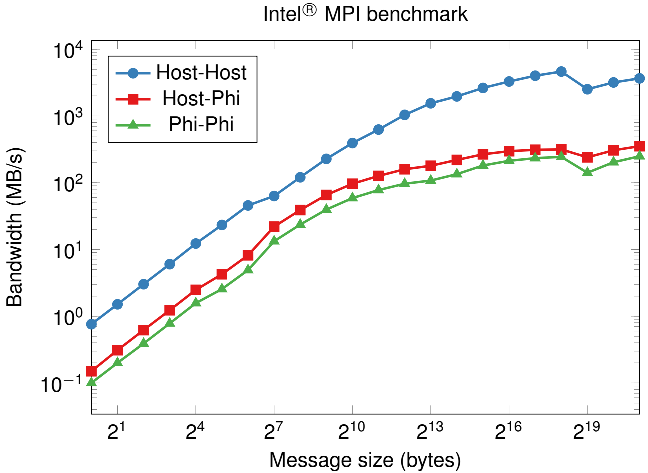
Part 2: Symmetric mode



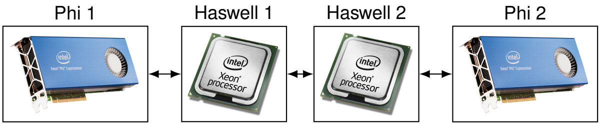
Symmetric mode: Initial results

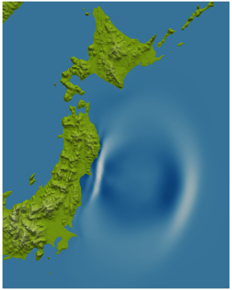


First problem: slow MPI communication with Phis

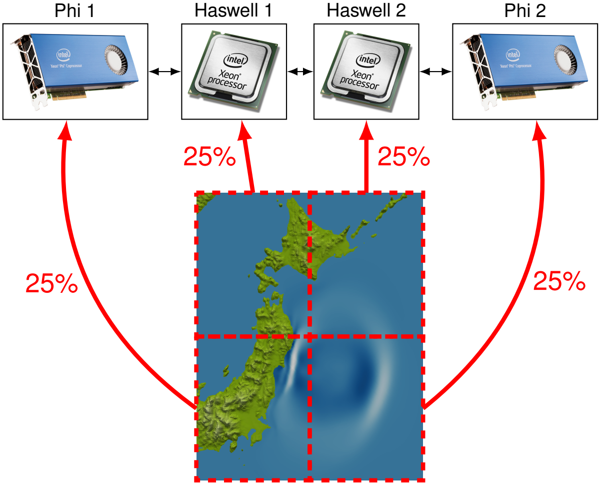


Second problem: homogeneous load balancing

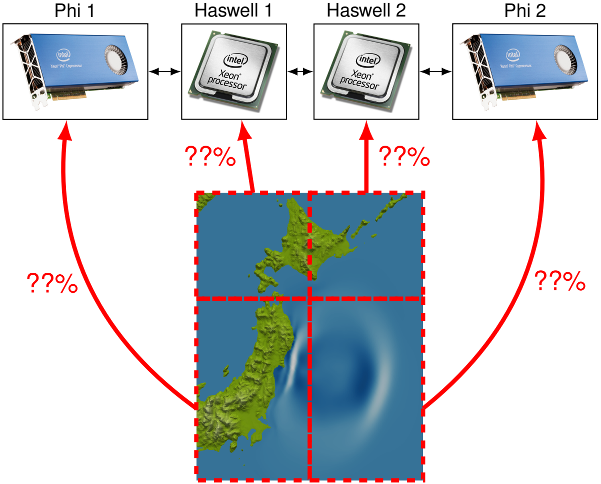




Second problem: homogeneous load balancing



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?

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.

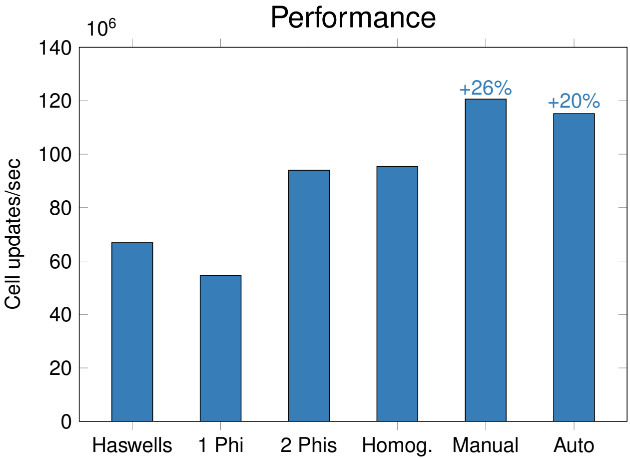
- But what about other systems? And other simulations?

Heterogeneous load balancing

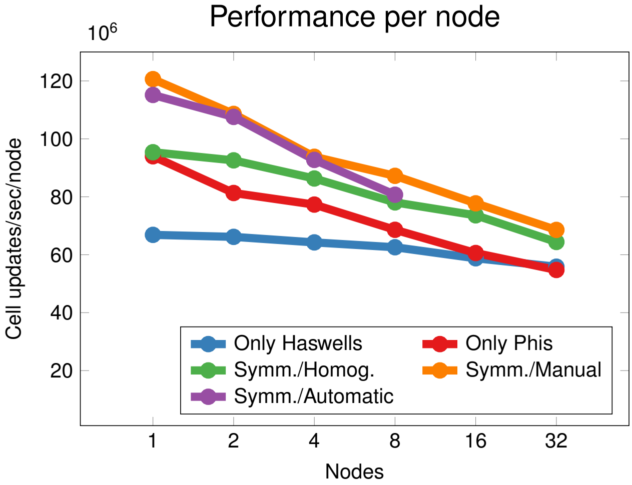
- So, we changed our load balancing implementation to also allow heterogeneous distributions.
- But how much load should we give to the Haswells and how much to the Phis?
- Experimentally, we found that for this specific system and for this specific simulation, the best performance is achieved by giving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

- But what about other systems? And other simulations?
  $\Rightarrow$ We also implemented a “auto-tuning” approach, where the simulation code iteratively chooses the distribution based on statistics from previous time steps.
  $\Rightarrow$ For that we defined the efficiency of each processor as: $Eff = load/time$.

Symmetric mode: Single node



Symmetric mode: Multiple nodes (weak scaling)



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

1. Meister et al. “*Parallel memory-efficient adaptive mesh refinement on structured triangular meshes with billions of grid cells*”. ACM Transactions on Mathematical Software (TOMS), 2016.
2. Ferreira et al. “*Load Balancing and Patch-Based Parallel Adaptive Mesh Refinement for Tsunami Simulation on Heterogeneous Platforms using Xeon Phi Coprocessors*”. Platform for Advanced Scientific Computing (PASC), 2017. (Being published with Open Access)

Acknowledgements

IT4Innovations
national01$#&0
supercomputing
center@#01%101

CNPq