Performance Modeling of a Dependency-Driven Mini-App for Climate

Bryce Lelbach (blelbach@cct.lsu.edu), Hartmut Kaiser (Louisiana State University)
Hans Johansen (LBNL, CRD, hjohansen@lbl.gov)

STE||AR
stellar.cct.lsu.edu

Motivation

Complex science applications often have very complex structures that weave together numerical and computational software with multi-scale physics components.

As a motivating example, LBNL is developing an adaptive atmospheric dynamical core ("dycore") in the Chombo framework (http://chombo.lbl.gov) for high-accuracy global climate simulations.

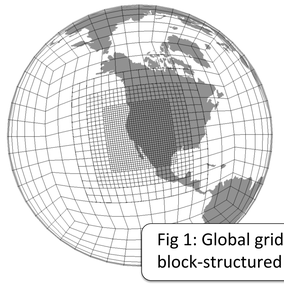


Fig 1: Global grid with block-structured AMR.

Chombo is a high-order massively scalable finite-volume framework for robust and accurate solution of partial differential equations in arbitrary geometry. Chombo also provides a block-adaptive adaptive mesh refinement (AMR) capability to allow feature isolation and tracking. For Climate applications, this could include weather prediction

Scientific applications include:

- Greater resolution of dynamic features (squall lines, atmospheric rivers)
- Regional weather phenomena (tropical cyclones, coastal climate)
- Grid refinement studies for new physics (cloud parameterizations)
- Evaluation of time discretization errors from 1D operator splitting ("column physics")

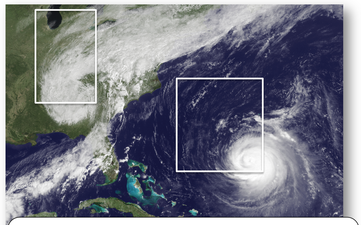


Fig 2: Tropical cyclones Lee and Katia, 6 September 2011, with tracking trajectories.

Research Challenge

Adaptive mesh refinement has been successfully employed on a suite of 2D and 3D climate-related test problems to verify accuracy and robustness of the algorithm. However, the complexity of the algorithm makes it difficult to create an idealized model for parallel scaling. Instead, we express the model as dependencies in a representative mini-app.

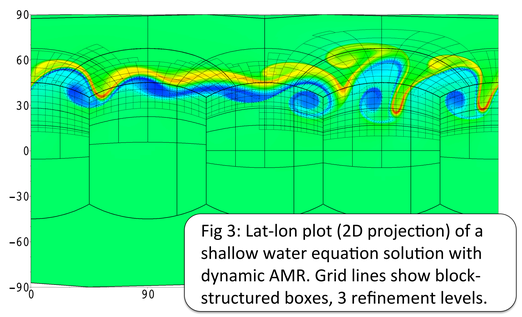


Fig 3: Lat-lon plot (2D projection) of a shallow water equation solution with dynamic AMR. Grid lines show block-structured boxes, 3 refinement levels.

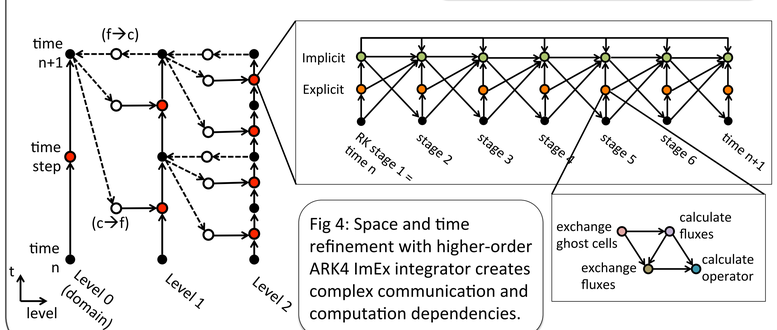


Fig 4: Space and time refinement with higher-order ARK4 ImEx integrator creates complex communication and computation dependencies.

HPX: Imperative vs. Functional

HPX is a parallel programming framework designed to facilitate asynchronous (functional) programming on many-core and traditional architectures. HPX is an alternative to MPI+OpenMP (imperative), and provides an implementation of ISO C++ concurrency in the C++11 standard.

HPX uses dependencies ("futures") for expressing distributed communication between and within nodes, using millions of lightweight tasks per hardware core. On most architectures, HPX can schedule tasks with sub-microsecond overheads, enabling fine-grained parallelism.

HPX's programming model:

- **"Future":** proxy for a dependency that is needed by another task.
- **Composable:** futures can express control flows and work queues through their dependencies.
- **Dynamic Load Balancing:** Work is scheduled as dependencies complete, with a work-stealing thread scheduler.
- **Unified APIs**: Remote communication and local in-memory work use same object-oriented API for dependencies.

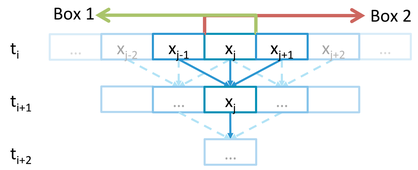


Fig 5: Difference stencils include "ghost cell" regions on neighboring boxes. HPX decomposes dependencies into swarms of tasks (dependencies, computation, and asynchronous communication)

Climate Mini-App Description

The Climate mini-app is a parameter-driven application that serves as a proxy for computation and communication patterns in the AMR Climate code:

- Uses explicit and vertical implicit block-structured discretizations
- Explicit operator is compute or memory-bound depending on its ghost cells, stencil footprint
- Vertical implicit operator is compute-bound by 1D non-linear solves, no ghost cells
- ARK4 time integration expresses complex implicit-explicit coupling dependencies
- AMR time refinement creates another layer of dependencies

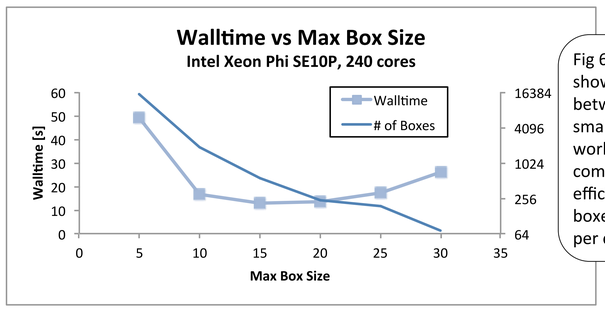


Fig 6: This graph shows the tradeoff between more, smaller boxes (less work per box, more communication) and efficiency (larger boxes, more work per communication)

Speedup on many-core Xeon Phi

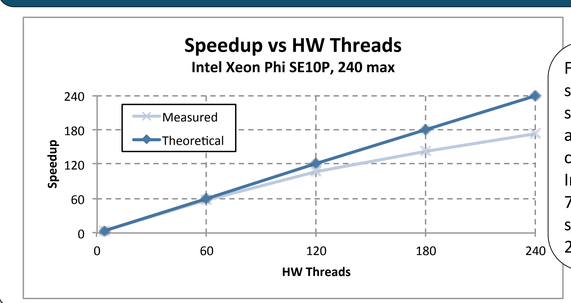


Fig 7: The HPX task scheduler has very low, stable overheads. Using asynchronous data communication and Intel's MKL, we achieve 72% of theoretical speedup on Xeon Phi's 240 hardware threads.

Results for Intel Ivybridge

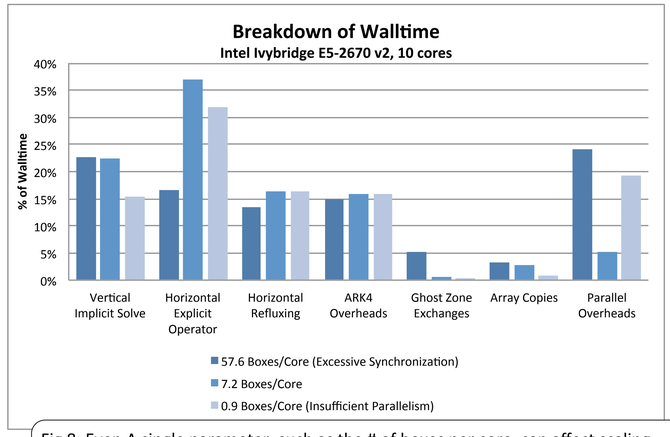


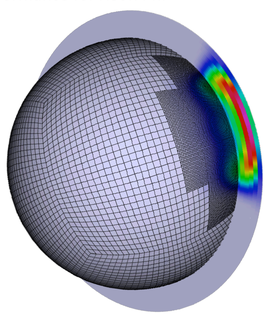
Fig 8: Even A single parameter, such as the # of boxes per core, can affect scaling dramatically. This shows the balance between parallel overheads (~5% indicates good parallelism for the middle case) and useful work (numerical calculations).

Future Research Directions

Future research topics will focus on predictive application performance on Xeon Phi:

- Include space-time AMR dependencies, and higher-order FLOP-intensive stencils
- Model different domain decomposition strategies (vertical, polyhedral, etc.)
- Evolve to representative kernels for the full Climate app performance behavior, including calls to complex "physics" routines
- Compare flat MPI, MPI+OpenMP, and the HPX version on Xeon Phi
- Use HPX's built-in performance monitoring to optimize performance for Xeon Phi

Take-away: Unlike traditional processors, Xeon Phi is an in-order execution, many-core architecture. Each Xeon Phi core has 4 hw threads (240 total), and every cycle the processor context switches to a new thread (barrel threading). Each Phi core is slower but enables more hardware thread-parallelism – if an application can take advantage of it. The ring-bus topology and a distributed hash table manage global caches, so core-cache locality is complex – creating programming complexity. But higher memory bandwidth and SIMD vector instructions improve performance (e.g. Intel MKL for LAPACK).



For More Information

Contact Bryce or Hans via email, or follow this QR code for more:

With support from Intel Parallel Computing Center @ NERSC, a DOE Science Facility