Harnessing Billions of Tasks for a Scalable Portable Hydrodynamic Simulation of the Merger of Two Stars

The International Journal of High Performance Computing Applications
XX(X):2–30

DOI: 10.1177/ToBeAssigned
www.sagepub.com/

Thomas Heller[FAU*] , Bryce Adelstein Lelbach[NV*] , Kevin A. Huck[UO] , John Biddiscombe[CSCS*] , Patricia Grubel[LANL*] , Alice E. Koniges[LBNL] , Matthias Kretz[GSI] , Dominic Marcello[LSU*] , David Pfander[US] , Adrian Serio[LSU*] , Juhan Frank[LSU] , Geoffrey C. Clayton[LSU] , Dirk Pflüger[US] , David Eder[LLNL] , and Hartmut Kaiser[LSU*]

thom.heller@gmail.com, brycelelbach@gmail.com, khuck@cs.uoregon.edu, biddisco@cscs.ch, pagrubel@lanl.gov, aekoniges@lbl.gov, m.kretz@gsi.de, dmarce504@gmail.com, david.pfander@ipvs.uni-stuttgart.de, aserio@cct.lsu.edu, frank@phys.lsu.edu, gclayton@fenway.phys.lsu.edu, dirk.pflueger@ipvs.uni-stuttgart.de, davidceder@gmail.com, hkaiser@cct.lsu.edu

[FAU]Friedrich-Alexander-Universität Erlangen-Nürnberg, [NV]NVIDIA , [LBNL]Lawrence Berkeley National Laboratory, [UO]University of Oregon, [CSCS]Swiss National Supercomputing Centre, [LANL]Los Alamos National Laboratory, [GSI]GSI Helmholtzzentrum für Schwerionenforschung, [US]University of Stuttgart, [LLNL]Lawrence Livermore National Laboratory, [LSU]Louisiana State University

[*]The STE||AR Group,stellar-group.org

Prepared using sagej.cls

Abstract
We present a highly scalable demonstration of a portable Asynchronous Many Task programming model and runtime system applied to a grid-based Adaptive Mesh Refinement hydrodynamic simulation of a Double White Dwarf merger with 14 Levels of Refinement that spans 17 orders of magnitude in astrophysical densities. The code uses the portable C++ Parallel Programming Model that is embodied in the HPX library and being incorporated into the ISO C++ Standard. The model represents a significant shift from existing bulk synchronous parallel programming models under consideration for exascale systems. Through the use of the Futurization technique, seemingly sequential code is transformed into wait-free asynchronous tasks. We demonstrate the potential of our model by showing results from strong scaling runs on NERSC's Cori system (658,784 Intel Knight's Landing cores) that achieve a parallel efficiency of ~96.8% using billions of asynchronous tasks.

Keywords
parallel runtime, binary star merger, asynchronous tasks, HPX, C++

1. Introduction

As High Performance Computing (HPC) moves towards increasingly diverse architectures and larger processor-counts, the HPC community is considering a range of new models. Newly evolving manycore and heterogeneous architectures present many programming challenges suggesting that a move beyond traditional HPC programming models may provide substantial benefits. In this paper we describe an approach to portable application programming at scale with a novel runtime and programming model. We demonstrate our approach with a real application on modern hardware. Our programming model approach emphasizes the following attributes:

- *Scalability:* Enable applications to strongly scale to exascale systems.
- *Programmability:* Reduce the burden we are placing on HPC programmers.
- *Performance Portability:* Eliminate or significantly minimize requirements for porting to future platforms.
- *Energy Efficiency:* Maximally exploit dynamic energy saving opportunities, leveraging the tradeoffs between energy efficiency, resilience, and performance.

In this paper we show how several of these objectives can be realized with a scalable simulation in modern time-domain astrophysics.

Advanced time evolving astrophysical simulations are only now becoming possible with new hardware and programming models. In particular, we show how our programming model approach enables the simulation of transient events caused by stellar mergers, which are spectacular, interesting, and fundamental

phenomena in our Universe. Their study pays dividends in the broadest possible ways, leading to insights into both astrophysics and fundamental physics. Additionally, until recently, merger events have been rarely observed, because they are short lived and can happen anywhere in the sky. Early in the next decade, the behemoth Large Synoptic Survey Telescope (LSST) (Željko Ivezić et al. 2008) study will begin. With a combination of a large telescope and a large field of view, this study will be 40 to 1,000 times more powerful than any survey done before. It will finally present us with a real chance to detect traditionally elusive events like Double White Dwarf (DWD) mergers in great numbers (~1 million new transients per night). Numerical simulations can help identify the signatures of the most interesting mergers within this very high volume of transients.

In §2 we describe the stellar merger problem we wish to solve. In §3 we briefly describe OctoTiger. In §4 we lay out the HPX model and discuss futurization, and in §5 we describe how OctoTiger uses futurization. We executed this model on NERSC's Cori machine, described in §6. The results of node-level and full-system scaling are detailed in §7. In terms of portability, we note that the same source code running at scale on Cori also can be compiled and run on an Apple Laptop. In §8 we describe how our experiment demonstrates the ability of HPX to realize scalability, programmability, and performance portability.

2. The Physical Problem

Interacting binary star systems are potential progenitors of a wide array of astrophysical phenomena. As a result, they have received attention from across the astrophysical community. Some groups have modeled mergers of DWDs as progenitors of Type Ia Supernovae (see Figure 1) (Katz et al. 2016; Pakmor et al. 2011; Zingale et al. 2009; Dan et al. 2011, 2012; Guillochon et al. 2010). White dwarf mergers that are not massive enough to cause a Type Ia Supernovae to leave an observable remnant, can be studied by continuing the model past the point of merger (Montiel et al. 2015; Dan et al. 2014; Staff et al. 2012; Schwab et al. 2012; Raskin et al. 2012). The beauty of a merger is that a vast range of fundamental physical phenomena with countless connections to all areas of astrophysics are packed into a small volume and a short time. However, this information can only be decoded with a suitable set of observations and numerical simulations.

Practically, DWD simulations feature tremendous variances in scale between the simulated physical entities. Figure 2 depicts the early stages of the formation of the mass-transfer stream and accretion disk in a binary star where the primary star or accretor (on the left) is ~5 times more massive than the donor star. Some binaries merge and others, such as this one, are expected to be stable to mass transfer. If the accretor is compact enough, the accretion stream will miss the donor on the first pass and form a disc. Figure 3 shows the same system after an accretion disc begins to form. These binary systems, known as AM Canum Venaticorum (AM CVn) systems, exist in a state of mass transfer for millions of years as the donor slowly transfers matter to the

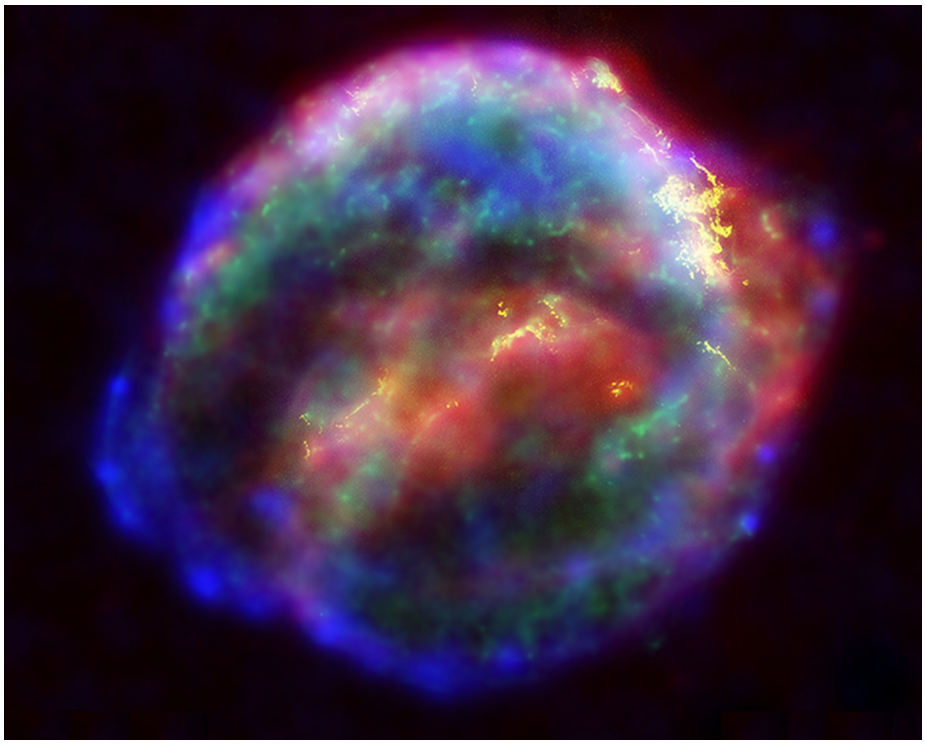

Figure 1. A false-color composite image of the Supernova remnant nebula from SN 1604 (Sankrit and Blair 2004). Visible to the naked eye, Kepler's Star was brighter at its peak than any other star in the night sky, with an apparent magnitude of -2.5. It was visible during the day for over three weeks. While discovered in Kepler's time, new studies are trying to resolve the question of whether or not Double White Dwarf (DWD) mergers cause Type Ia Supernovae.

accretor and the orbital separation widens. When mass transfer first ensues, they have orbital periods of only a few minutes. The mass transfer causes the orbit to separate, and the AM CVns we observe typically have periods between 10 and 40 minutes. Periodic instabilities in the disc can result in dwarf nova. The buildup of Helium in the disc can periodically detonate as a sub-luminous Supernova known as a Type ".Ia" (point one a) Supernova. Stable mass transfer cases require thousands of orbits to simulate and thus necessitate both extreme computation scales and the conservation of angular momentum. In a simulation, we find that a dynamically adapting grid to capture many orders of magnitude of scales and accurate angular momentum conservation is required to properly evolve and distinguish these different outcomes. For example, it is necessary to resolve the grid around the accretion stream with high resolution. Thus, many groups have turned to Adaptive Mesh Refinement (AMR) methods (Katz et al. 2016; Kadam et al. 2017) to address these problems. The simulations described in this paper use up to 14 Levels of Refinement (LoR), to simulate more than 4

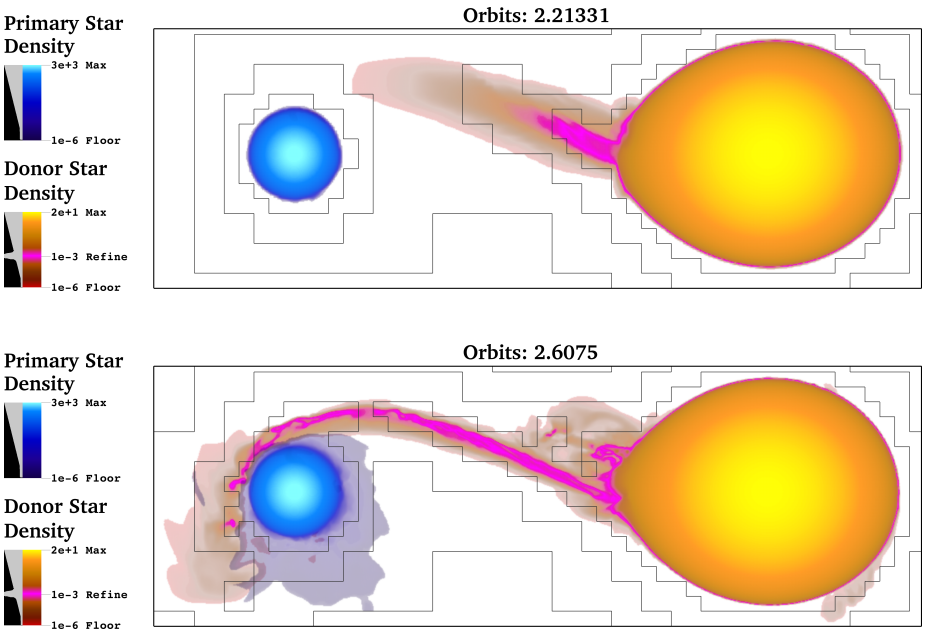

Figure 2. The early stages of mass transfer in a binary star system. The accreting star is five times more massive than the donor star.

magnitudes difference in resolution of space across the computational domain. Multiple coupled physics solvers with very different performance characteristics and algorithmic complexities are combined in order to adequately simulate the behavior of DWD systems. Rapidly changing physical quantities with steep gradients require the mesh to dynamically change in resolution. Consequently, the resulting application is highly irregular and frequent load balancing over the course of a simulation is required.

3. The OctoTiger Simulation Package

In this paper, we detail programming model updates to create *OctoTiger*, a 3D finite-volume octree AMR hydrodynamics code with Newtonian gravity (Marcello et al. 2016; STE||AR Group 2017d). It is a successor to previous codes described in (Lindblom et al. 2001; Ott et al. 2005; D'Souza et al. 2006; Motl et al. 2007; Kadam et al. 2016; Motl et al. 2017; Byerly et al. 2014; Lelbach et al. 2013; STE||AR Group 2017c). In OctoTiger, the astrophysical fluid is modeled using the inviscid Euler equations. These are solved using a finite-volume central scheme (Kurganov and Tadmor 2000). Species within the fluid are evolved as passive scalars. We use an angular momentum conserving hydrodynamic solver, based on the methods described in Despres and Labourasse (Desprésa and Labourasse 2015). The gravitational potential and force are computed using a modified version of the *Fast Multipole*

Method (FMM) (Dehnen 2000), that conserves linear momentum to machine precision. With our extension to the method (Marcello et al. 2016; Marcello 2017), OctoTiger's FMM solver also conserves angular momentum to machine precision. The code's capability to conserve angular momentum at scale is novel and facilitates long-running ab inito simulations spanning thousands of orbits, such as stable mass transfer binaries (see Figure 3).

The computational domain in OctoTiger is based on a three-dimensional octree AMR structure. Each node in the structure is an $N \times N \times N$ Cartesian sub-grid (in this work, $N = 8$), and may be further refined into eight child nodes, each containing its own $N \times N \times N$ sub-grid with twice the resolution of the parent. The AMR structure is *properly nested*, meaning that there is no more than one jump in refinement level across adjacent leaf nodes. The AMR refinement criteria is based on density. This refinement criteria in the current simulation is checked every 15 timesteps to see if refinement or coarsening is required, based on the minimal number of timesteps required for a feature of the flow to propagate between two cells. After each refinement / coarsening step, the sub-grids need to be redistributed, introducing a need for dynamic load balancing. The images in Figure 2 show how the AMR grid is dynamically refined as the simulation progresses in order to properly reflect density changes in the mass-transfer stream. In the top image, the accretion stream has not formed yet, and the region between the stars is not fully refined as it does not fall below the refinement criteria density of 1e-3. In the bottom image, the accretion stream, now fully formed and refined, has just missed the primary star. Over time, the material from the stream will slowly form an accretion disk around the primary.

4. Programming Model Considerations

We developed the OctoTiger application framework (STE||AR Group 2017d) in ISO C++11 using HPX (Heller et al. 2012, 2013; Kaiser et al. 2014, 2015; Heller et al. 2016; STE||AR Group 2017a). HPX is a C++ standard library for distributed and parallel programming built on top of an Asynchronous Many Task (AMT) runtime system. Such AMT runtimes may provide a means for helping programming models to fully exploit available parallelism on complex emerging HPC architectures. The HPX methodology described here includes the following essential components:

- An ISO C++ standard conforming API that enables wait-free asynchronous parallel programming, including `future`s, `channel`s, and other asynchronization primitives.
- An Active Global Address Space (AGAS) that supports load balancing via object migration.
- An active-message networking layer that ships functions to the objects they operate on.
- A work-stealing lightweight task scheduler that enables finer-grained parallelization and synchronization.

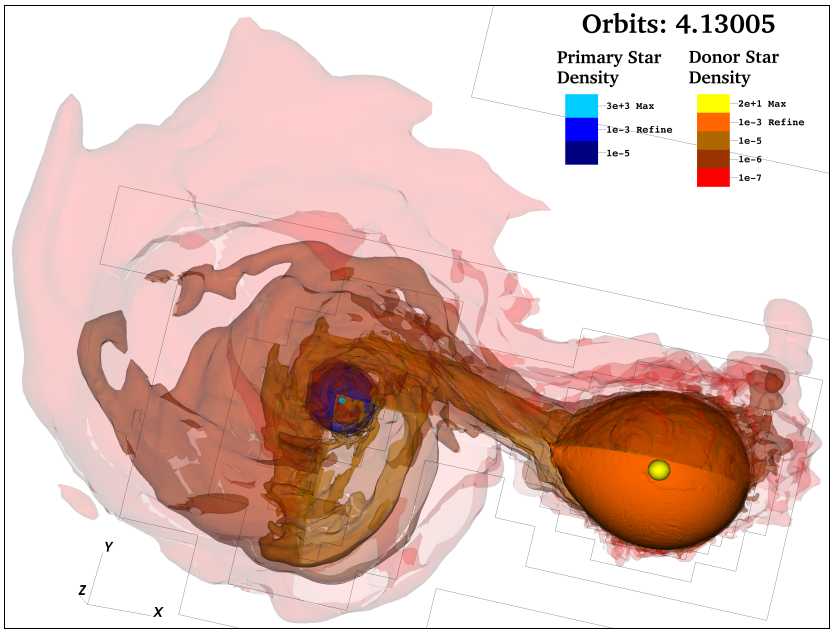

Figure 3. A three-dimensional contour plot of the system in Figure 2 after an accretion disc begins to form.

The design features of HPX allow application developers to naturally use key parallelization and optimization techniques, such as overlapping communication and computation, decentralizing control flow, oversubscribing execution resources, and sending work to data instead of data to work. Using *Futurization*, developers can express complex dataflow execution trees that generate billions of HPX tasks that are scheduled to execute only when their dependencies are satisfied (see Section *4.2.1 Futurization*). Additionally, HPX also provides a performance counter and adaptive tuning framework that allows users to access performance data, such as processor utilization, task overheads, and network throughput (see Section *5.2 Performance Counters and APEX*).

4.1 Background

Parallel programming models are emerging to meet the challenges of manycore, heterogeneous, and exascale architectures. Models call on the AMT methodology to efficiently avoid artificial barriers and overlap communication with computation to hide unavoidable latencies (USDOE 2012; Wheeler et al. 2008; Huck et al. 2015; Anderson et al. 2013; Dekate et al. 2012). While the concepts of AMT systems are emerging in many modern programming models, some of the more advanced competing implementations of new programming models with similar concepts include Charm++ (Kumar et al. 2004), Intel Cilk Plus (Intel 2017a), OpenMP with tasking (deSupinski et al. 2017),

Chapel (Chamberlain et al. 2007), Intel SPMD Program Compiler (ISPC) (Intel 2017b), X10 (Charles et al. 2005), and Legion (Bauer et al. 2012).

Parallelism is expressed and presented to the user in different ways in each of these solutions, but a trend towards asynchronous programming is evident. While the majority of the task based programming models focus on dealing with node level parallelism, HPX presents a single model to the programmer that supports both local and remote execution. This concept assures a uniform programming model that is architecture independent.

Many applications must overcome the scaling limitations imposed by current programming practices by embracing an entirely new way of coordinating parallel execution. Fortunately, this does not mean that we must abandon all of our legacy code. HPX can use MPI as a highly efficient portable communication platform and at the same time serve as a back-end for OpenMP, OpenCL, or even Chapel while maintaining or even improving execution times. This opens a migration path for legacy codes to a new programming model that allows old and new code to coexist in the same application.

Within the astrophysics community, two recent and important contributions that use asynchronous tasking and AMR are SWIFT (Schaller et al. 2016) and ChaNGa (Menon et al. 2015). SWIFT scales up to ~100K cores with ~60% parallel efficiency, and ChaNGa has demonstrated scalability up to ~500K cores with ~93.4% parallel efficiency. SWIFT code uses asynchronous tasking to blend computation and communication, in a similar manner to HPX, but its parallel runtime is application-specific and not designed for reuse. ChaNGa uses Charm++ (Kumar et al. 2004) and is tied to the Charm++ compiler and runtime. In contrast to both, HPX is a library-only generic solution that implements standardized C++ APIs and can be applied to existing C++ code with minimal modifications.

4.2 Fundamental Properties of the HPX Model

One goal of this paper is to demonstrate the viability of the HPX programming model through a portable and standards conforming application and to demonstrate that application at scale. OctoTiger fully embraces the C++ Parallel Programming Model, including additional constructs that are incrementally being adopted into the ISO C++ Standard. The programming model views the entire supercomputer as a single C++ abstract machine. A set of tasks operates on a set of C++ objects distributed across the system. These objects interact via asynchronous function calls; a function call to an object on a remote node is sent as an *active message* to that node. A powerful and composable primitive, the `future` object, is used to represent and manage asynchronous execution and dataflow.

A crucial property of this model is the *semantic equivalence* between local and remote operations. This provides a unified approach to vector-, core-, and node-level parallelism based on proven generic algorithms and data structures in the ISO C++ Standard today. The programming model is intuitive and enables performance portability across a broad spectrum of the landscape of

increasingly diverse supercomputing hardware. Results of using the HPX model on additional architectures will be presented in a future publication.

4.2.1 Futurization The fundamental asynchrony primitive in the C++ Parallel Programming Model is the `future`. A `future` consists of a state (ready or not ready), a value, and a set of continuation functions. `future` objects represent values that have not yet been computed.

`future`s are monadic data structures, e.g. they can be composed together and used to chain operations. An operation `g` can be attached as a continuation to a `future f`; the result of the continuation-attaching operation (`f.then(g)`) is another `future` that represents the computation of `g`. Multiple `future`s can be joined together into a single `future`, and multiple continuations can be attached to a single `future`, allowing the construction of arbitrary dataflow graphs.

The continuation-attaching operation is very powerful because it enables Continuation Passing Style (CPS) (Appel and Jim 1989) asynchrony. CPS ensures that tasks that have data dependencies do not start executing until all their dependencies are satisfied. This approach has been shown to reduce unnecessary waiting and avoid latency (Syme et al. 2011).

Sequential	**Futurized**	**Futurized with Coroutines**
`t`	`future(t)`	*Same as* ***Futurized****.*
`T f(){...}`	`future<T> f(){...}`	*Same as* ***Futurized****.*
`T t = f();`	`future<T> t = f();`	*Same as* ***Futurized****.*
`U g(T t){...}`	`future<U> g(T t){...}`	*Same as* ***Futurized****.*
`U u = g(t);`	`future<U> u = t.then(g);`	`future<U> u = g(co_await t);`
`U u = g(f());`	`future<U> u = f().then(g);`	`future<U> u = g(co_await f());`
`V h(T t,U u){...}`	`future<V> h(T t,U u){...}`	*Same as* ***Futurized****.*
`V v = h(t,u);`	`future<V> v = when_all(t,u).then(h);`	`future<V> v = h(co_await t,co_await u);`
`V v = h(f(),g());`	`future<V> v = when_all(f(),g()).then(h);`	`future<V> v = h(co_await f(),co_await g());`

Table 1. The Futurization Technique: This table shows sequential C++ constructs (left column) and the corresponding constructs after applying a transformation called *Futurization* (center and right columns). This technique transforms sequential code into wait-free asynchronous tasks with explicit data dependencies. The right column uses the ISO C++ Coroutines Technical Specification (C++ Standards Committee 2017b), that provides language-level support for this transformation. `T`, `U` and `V` are types, `t`, `u`, `v` are variables of those types, and `f`, `g` and `h` are functions.

Rewriting synchronous blocking code as a chain of wait-free continuations can be achieved through the straightforward Futurization technique, depicted in Table 1. The Futurization technique allows a delay of direct (sequential) execution in order to avoid synchronization. The futurized code no longer directly calculates results, but instead generates a dynamic execution tree

representing the inherent data dependencies of the original algorithm. The execution of this tree can now be parallelized by the underlying AMT runtime, yielding the same result as the original code. Some programming languages, such as F# and C#, have introduced powerful language facilities that simplify this transformation (Syme et al. 2011; Bierman et al. 2012). Such a feature, based on `await` in C#, has recently been introduced to the C++ programming language via an ISO Technical Specification (C++ Standards Committee 2017b) and will further simplify the process of futurizing existing code.

Futures are created by *control structures*, generic routines parameterized on execution semantics, such as C++11 `std::async` C++ Standards Committee (2011, §futures.async) or the C++17 parallel algorithms library C++ Standards Committee (2017a, §algorithms.parallel). *Execution policies* describe the constraints and parameters of execution (ex: `std::par` allows parallelization, while `std::par_unseq` allows parallelization and interleaving C++ Standards Committee (2017a, §algorithms.parallel.overloads)), and *executors* describe where execution will occur (ex: a GPU executor, a thread pool executor, an RPC executor). An upcoming ISO Technical Specification (Hoberock et al. 2017) will add executors to the C++ language. Vector data types and execution policies can be used to facilitate vector parallelism (see *5.1 Vectorization*).

5. Methodology

We use HPX Futurization in OctoTiger to eliminate global barriers and thus reduce barriers to optimal parallel performance as much as possible. The only phases of OctoTiger that require a global barrier are 1) rebalancing the Adaptive Mesh Refinement (AMR) tree after refinement/coarsening and 2) computing and distributing the maximum allowed time-step size.

We implement AMR operations in a natural way that is generic and can be easily parallelized: traverse the tree recursively, apply a transformation (ex: refinement/coarsening, migrate children, performing a time-step) to each tree node, and return a recursively reduced result.

The sequential version of this algorithm is shown in Listing 1. If a specific node is refined, that is if it contains children, we recurse in a depth-first pattern. The computation for the current node is combined with the results returned by the traversal from the children, leading to a natural way to gather results and propagate values from the leaves to the root. The downside to this approach is that a stack overflow might occur when executing trees of great depth, due to excessive recursion.

By applying the Futurization techniques, the formulation of tree traversal can be parallelized. The futurized traversal algorithm is depicted in Listing 2. Instead of direct recursion, we recurse asynchronously. The computation for a given tree node, representing a sub-grid in OctoTiger, is overlapped with the children computations. When communication with possibly remote tree node objects takes place, it is transparently hidden by other ongoing computations. This futurized tree traversal is the basic parallel pattern implemented in OctoTiger, including the most expensive operations: `regrid` and `step`.

```
T tree_node::traverse() {
  if (is_refined) {
    // 8 for children, 1 for this node.
    array<T, 9> r;
    for (int i = 0; i < 8: ++i)
      r[i] = children[i].traverse();
    r[8] = compute_result();
    return combine_results(r);
  }
  else return compute_result();
}
```

Listing 1: Natural recursive tree traversal.

```
future<T> traverse() {
  if (is_refined) {
    // 8 for children, 1 for this node.
    array<future<T>, 9> r;
    for (int i = 0; i < 8: ++i)
      r[i] = async(traverse, children[i]);
    r[8] = compute_result();
    return when_all(r).then(combine_results);
  }
  else return future(compute_result());
}
```

Listing 2: Futurized recursive tree traversal.

The extension of this approach to a distributed memory application using AGAS is straightforward. We simply register the tree node objects with AGAS, and store Globally Unique Identifiers (GUIDs) in the `children` array instead of storing the objects themselves. The GUIDs can be thought of as global pointers. A GUID can be passed as the second argument to `async` (after the function to be invoked, before the arguments to the function) to an make asynchronous Remote Procedure Call (RPC). The children may reside on the same node as the parent, or they may be distributed over various different compute nodes; the algorithm works in both cases, because of the *semantic equivalence* of local and remote interfaces in our model.

The `regrid` phase performs refinement/coarsening of the entire AMR octree. It's implemented as four distinct futurized tree traversals:

1) Check if each node in the octree needs to be refined or coarsened, and if so, mark it for refinement and ensure the correct refinement levels of its neighbors (e.g. *proper nesting*).

2) Count the total number of tree nodes and the number of tree nodes in each child via recursive reduction.
3) Move tree nodes that are being rebalanced. A space filling curve is used to determine the distribution. Migration is initiated by parent tree nodes.
4) Update references to neighboring tree nodes. Old references may be outdated due to the preceding steps.

The `step` phase implements the hydrodynamics and gravity solvers. Each tree node has its own N^3 sub-grid containing the evolved variables for the solvers. The futurized fluid solver, gravity solver and time-step size propagation differ from the other futurized tree traversals. The hydrodynamics and gravity solvers depend on ghost zone data (zones of data that are shared between processors, usually on the boarder of grids separated for parallel computation) from neighboring regions and the dynamically-computed time-step size from a reduction on the previous time-step's entire tree.

To avoid introducing needless waiting during these solves and exchanges, OctoTiger uses a `channel` to propagate results between neighboring regions. `channel`s are primitives that represent a series of futures that will be produced asynchronously. A consumer can request a `future` for a particular *epoch* (e.g. an integer) from a `channel` and a producer can set a value for an epoch. `channel`s allow producers and consumers to agree on a location (the `channel`) where they will communicate repeatedly.

The associated state and storage for each epoch's `future` is created lazily on demand, e.g. after either a consumer or producer requests the epoch. `channel`s transparently buffer values on the fly when needed, allowing producers to proceed ahead of consumers – avoiding needless waiting. Our asynchronous solvers use `channel`s to allow computation to proceed as far as possible and overlap communication and computation. After the solve for one sub-step is complete, a tree node will retrieve `future`s from its channels and attach continuations to the futures (via `when_all`) that will compute the next sub-step. The continuation will only begin executing when the necessary data for that sub-step has been sent to the channels.

A simplified example of a `channel`-based asynchronous ghost zone exchange on a 2D mesh partitioned into 4 sub-grids is shown in Figure 4. This technique allows computation to proceed as far as possible without needless waiting. First, all sub-grids begin computing the first sub-step (red). Sub-grids A, B, and D finish their computations and send ghost zone data to their neighbor's `channel`s. Then, A, B, and D combine the dependencies of the next sub-step with `when_all` and attach a continuation that computes the next sub-step to the resulting `future`. C is still computing the first sub-step, so it has not sent the first sub-step ghost zone data to A or D, and the second sub-step continuation for A and D does not start yet (C, however, has received ghost zone data from A and D). B's continuation for the second sub-step has all the data it needs, so that computation is executed (blue). When it is completed, a third sub-step continuation is created and the second sub-step ghost zone data is sent from B to A and D's `channel`s, where it is effectively buffered. In OctoTiger, the

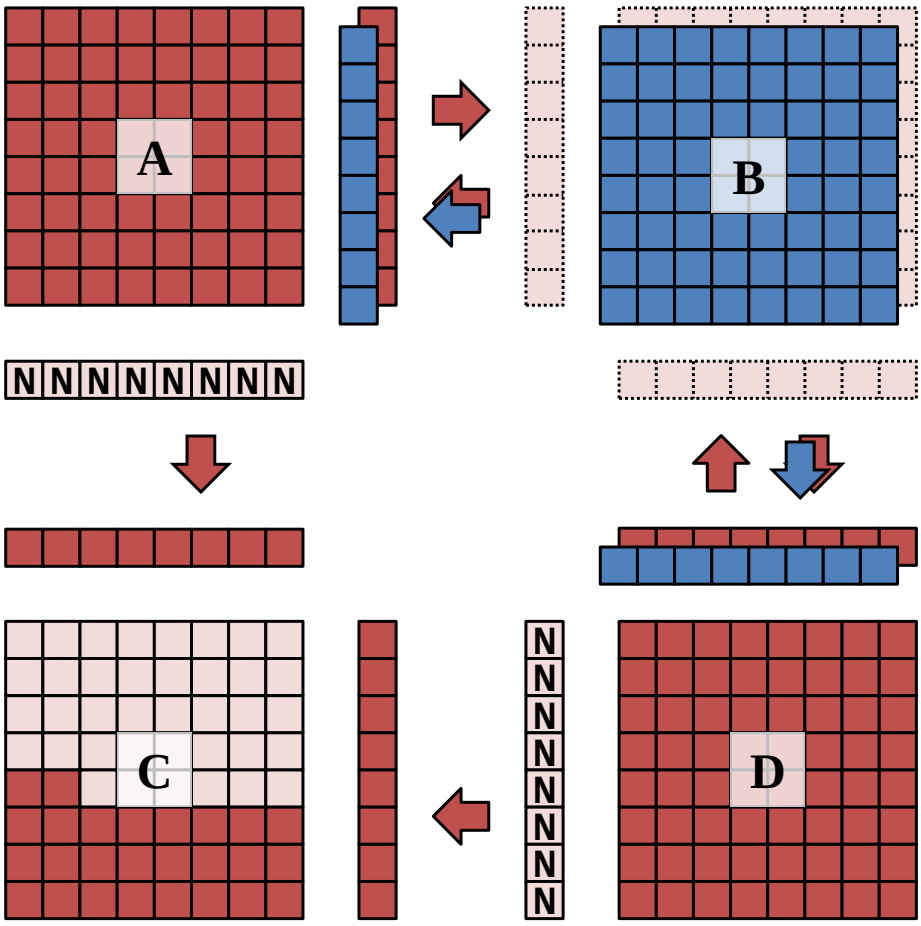

Figure 4. Depiction of ghost zone exchange patterns.

dependencies are more complicated, due to 3D geometry, coarse-fine boundaries, and other communications such as flux corrections from children tree nodes to parent tree nodes.

To advance the hydrodynamics variables for each cell, a single sub-step in time requires knowledge of the variables in the neighboring three cells on each side and in each dimension. These *ghost zones* are updated after every sub-step, with each tree node sending the required data from its interior (non-ghost zone) cells to its neighboring tree nodes. When a leaf node has no neighboring tree node at the same level of refinement, the ghost zones are interpolated from the neighboring tree nodes of its parent.

Like the hydrodynamics solver, the FMM solver also requires data from neighboring tree nodes on the same level. It also requires data from its child and parent tree nodes. Each tree node executes three steps for the FMM algorithm:

Levels of Refinement	# of Tree Nodes	File Size (GB)
7	1,641	0.1
8	4,361	0.3
9	36,201	0.8
10	47,721	3.5
11	278,921	21
12	1,934,025	140
13	14,412,841	N/A
14	111,806,409	N/A

Table 2. Number of tree nodes (sub-grids) for example dataset after initialization for different maximum Levels of Refinement. This Includes the size of the file needed to be used for initializing the computation.

1) Compute multipoles by combining multipoles from its children tree nodes, and communicate multipoles to its parent tree node and the relevant subsets to its neighboring tree nodes.
2) Compute the Taylor expansion of the gravitational interaction between multipoles, including those from neighboring tree nodes on the same refinement level.
3) Add its own Taylor expansion to the Taylor expansion of the gravitational potential from its parent tree node, and communicate the total Taylor expansions to its children tree nodes.

The FMM solver requires four cells of ghost zone data from its neighboring tree nodes. Because of the large amount of neighboring data required, rather than storing ghost zone cells for the FMM solver, the data from neighboring cells is discarded once the relevant interactions are computed.

5.1 Vectorization

The combination OctoTiger with HPX uses the Vc (Kretz 2015a; Kretz and Lindenstruth 2011) library for the portable expression vector parallelism. This library is advancing towards potential standardization in the C++ programming language (Kretz 2017, 2016, 2015b,c). The Vc library enables explicit and portable vectorization through vector data types (`datapar<T>`) that store a target-architecture specific number of elements. `datapar` has arithmetic operator and math function overloads that simultaneously apply element-wise. These data types mimic the semantics of the built-in arithmetic types of the C++ language to a large extent, making algorithm vectorization almost as easy as a simple type replacement.

5.2 Performance Counters and APEX

Within our HPX implementation is a performance counter framework with a uniform interface for extracting arbitrary information including performance metrics, queue lengths, execution times of crucial functions, memory footprint,

network utilization, or any other aspect of application or runtime system behavior. The counters are accessible through AGAS and are available at the thread, process, or system level. The counters can be queried periodically, on demand, or at the end of execution. They are output either to the terminal or to a file in a variety of formats.

The HPX implementation is also integrated with the Autonomous Performance Environment for eXascale (APEX) (Huck et al. 2015; XPRESS APEX 2017). The APEX environment provides a lightweight performance measurement and control library specifically designed for use in HPX and has been extended to other runtime systems. It is integrated into the task scheduler within HPX, providing direct measurement of every task scheduled by the runtime. Unlike other parallel performance measurement libraries, APEX includes support for direct timing of tasks that yield and resume, even if resumption happens on a different OS thread than the one that yielded the task. The aforementioned HPX counters are also stored in APEX along with hardware and OS utilization/status counters.

Optionally, APEX can be integrated with TAU (Shende and Malony 2006) for detailed profiling, sampling and tracing, and PAPI (Dongarra et al. 2001) for hardware counter and GPGPU support. Alternatively, APEX can utilize the OTF2 (Eschweiler et al. 2012) library for full event tracing for analysis and visualization in Vampir (Knüpfer et al. 2008). The APEX environment gathers system health and utilization information through available user-space OS methods such as the `/proc` virtual filesystem. Power and energy data is available through the Cray PM Counters (Martin and Kappel 2014) or Linux Power Capping Framework (Linux Kernel Organization 2017) interface.

On its own, APEX maintains an internal performance state that is asynchronously updated by the HPX runtime as a scheduled task. Measurement overheads are typically less than 2-3%. The APEX measurement artifacts include process-level profiles, concurrency and scatter plot charts, and formatted text output. During this research, APEX was used to help identify, resolve, and verify a serialization bottleneck during the tree formation and rebalancing steps of the regridding phase of OctoTiger, as shown in Figure 5.

6. Experimental Setup

The primary system used for the experiments in this paper is a Cray XC40 installation at the National Energy Research Scientific Computing Center (NERSC) located in Berkeley, California. (He et al. 2018) known as Cori. Significant dedicated time on the Cori machine enabled the accurate scaling measurements presented here. Cori consists of two phases, Phase 1 consisting of an Intel Haswell partition and Phase 2 consisting of an Intel Knight's Landing (KNL) partition. For the following system description and results, we limit our discussion to Phase 2. The Phase 2 system consists of 9,688 nodes; each node has a single Xeon Phi 7250 processor with 68 cores and 272 hyper-threads (HTs) (Intel 2017c). Therefore, there are 658,784 cores and up ~2.6 million threads overall. The whole system has a peak double precision floating point

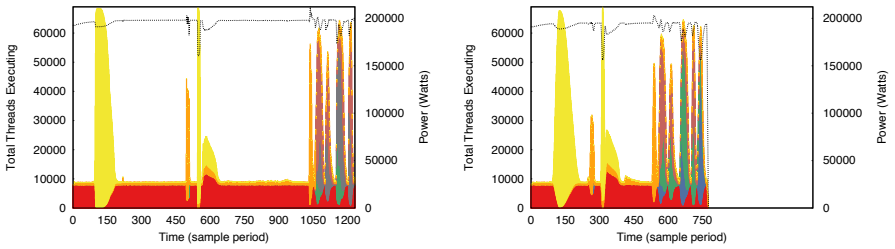

Figure 5. Concurrency views using APEX of OctoTiger running on 1,024 nodes of Cori. Higher values indicate better system utilization. The figure on the left shows a profile of OctoTiger prior to Futurization. There are two long periods of serial execution: 1) after the checkpoint load and grid creation stage (first yellow bulge) and 2) after the first gravity solve (center), followed by two time-steps. The figure on the right shows the same sequence of events after applying the Futurization methodology to make the regridding algorithm (that is used during checkpoint loading) more scalable. Each color represents an HPX task type, and the aggregate power consumption (Watts) across nodes is visible as a dashed black line. The axis scales are equivalent.

performance of ~29.1 PFLOP/s and an aggregated main memory capacity of ~1.1 PB. Cori uses a Cray Aries interconnect and has a global bandwidth of ~45 TB/s.

The Intel KNL Xeon Phi 7250 processor supports the AVX512 instruction set and has two 512-bit fused-multiple-add vector units per core. Each Xeon Phi 7250 is capable of up to 3 TFLOP/s (double precision). The KNL cores are derived from the Intel Atom architecture that has lower scalar performance compared to contemporary Intel Xeon processors. (Doerfer et al. 2016) In addition to 96 GB of DDR4 for main memory, the Xeon Phi 7250 has 16GB high bandwidth memory (MCDRAM). The bandwidth of the MCDRAM is ~460 GB/s. The MCDRAM can be configured as a direct mapped cache or as explicitly-programmable memory. (Doerfer et al. 2016) The KNL processor can be configured in 3 different NUMA clustering modes: 1) All-to-All, 2) Quad, and 3) Sub-Numa-Clustering (SNC). Each mode determines the configuration of the entries in distributed hash table among tiles for virtual address translation. The latencies for memory access are highest in the first case and lowest in the third (Sodani 2015).

After a first initial evaluation of the achievable performance, the single-node performance did not vary significantly between the different clustering and high bandwidth memory modes. Thus we used the Quad mode (NERSC's default) and configured the MCDRAM as a cache.

In our experiments, we used the software listed in Table 3. Experiments were also performed with other compilers, however the computations were significantly slower, so we chose to use GCC. To measure execution times, we used `std::chrono::steady_clock`, a monotonically increasing system clock with nanosecond resolution C++ Standards Committee (2011, §time.clock.steady).

Compiler	GCC 6.3.0 (GNU 2017)
MPI	Cray MPICH 7.4.4(NERSC 2017a)
Jemalloc	4.5.0(jemalloc 2017)
Boost	1.63(Boost 2017)
hwloc	1.11.6(Open MPI 2017)
HPX	19bd11a(STE||AR Group 2017b)
APEX	58214cf(Kevin Huck 2017)
OctoTiger	0b6cd60(STE||AR Group 2017e)

Table 3. Software versions used in our experiments.

Timings are reported for the major phases of the simulation, including initialization, grid creation, regridding and the solvers.

7. Results

The following section presents results from strong scaling experiments with OctoTiger on Cori (NERSC 2017b) and demonstrates the scalability of our approach. The problem size was controlled by setting the maximum Levels of Refinement (LoR) (see Table 2). The effectiveness of the Futurization of OctoTiger was observed by profiling the application with the APEX toolkit (see Figure 5). The results presented include I/O in the initialization phase and plain computation time for the remaining part.

7.1 Node-Level Scaling

First, we look at single node scalability to determine a suitable number of cores and processing elements per core for performing the distributed memory full-system scaling experiments.

Figure 6 shows the scalability of an OctoTiger 7 LoR 10 time-step simulation on a single KNL node with different numbers of hyper-threads (HTs) per core. This experiment shows that using 2 HTs per core gives the best overall performance, with a parallel efficiency of ~87%. This is a ~1.3× speedup over the results for 1 HT per core.

This demonstrates the applicability of the futurization technique on manycore system like the KNL. By having the system oversubscribed with ~24 tree nodes per core, we were able to exploit the on-node parallelism efficiently. Increasing the number of sub-grids per core even further does not show significant improvements with respect to scalability. Reducing this number however, leads to a drop in performance.

7.2 Full-System Scaling

To assess scalability and distributed performance of our futurized application, we did strong scaling runs for different LoR. Figure 7 provides an overview of the results.

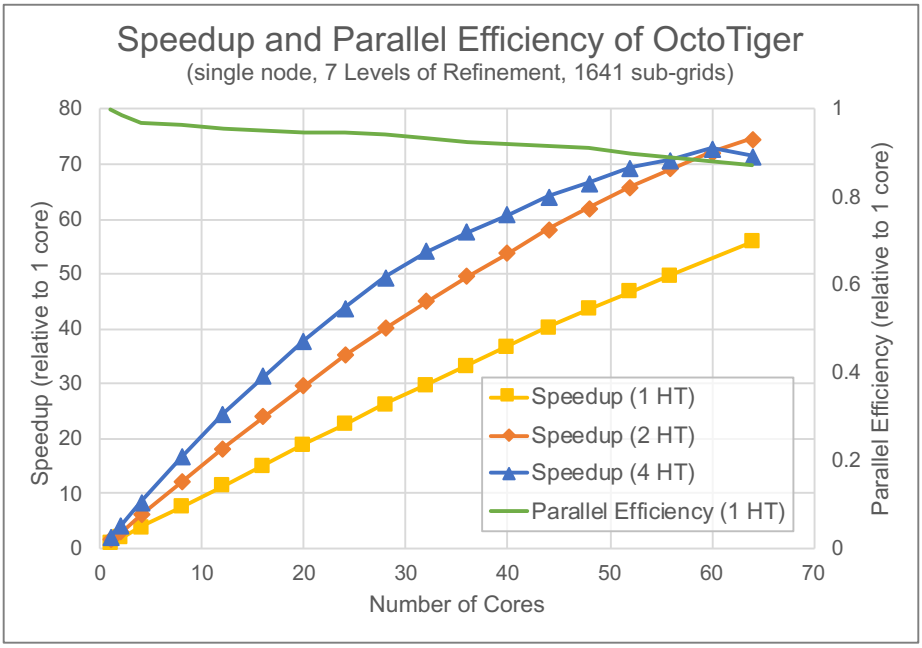

Figure 6. The speedup and parallel efficiency of OctoTiger strong scaling experiments on a single KNL node with different numbers of hyper-threads (HTs) per core is shown above. The 2 HTs per core case performs best. It achieves a speedup of ~55.9× when scaling from 1 to 64 cores, corresponding to a parallel efficiency of ~87%. This is a ~1.3× speedup over the results for 1 HT per core.

Because we are strong scaling, the speedup naturally flattens off after the number of sub-grids per core drops below a certain value. In our experiments, this was happening at ~100 sub-grids per core. The overall speedup from 10 LoR at 16 compute nodes (~44 sub-grids per core) to 14 LoR at 9,640 compute nodes (~171 sub-grids per core) is ~342×, corresponding to a parallel efficiency of ~56.8%. Since the number of sub-grids per core is not the same, the comparison does not fully reflect the scalability of OctoTiger. However, it can be clearly observed that excellent scalability can be achieved by providing sufficient work to be executed by each core.

To further demonstrate the scalability of OctoTiger and show the effects of futurization at scale, Figures 8 and 9 provide a breakdown of performance of the different stages of the application.

These figures show that when executing the 13 LoR and 14 LoR problems, the futurized tree traversal, described in §5, is able to scale up to the full Cori system – e.g. 655,520 cores. While the initialization of the problem, which includes loading the initial octree from disk, is hampered by I/O limitations, the actual computation exhibits a parallel efficiency of 96.8% (obtained by the strong scaling of the 14 LoR problem from 4,096 nodes to 9,640 nodes). The sustained aggregated bandwidth for loading the initialization file was at 1.4 GB/s with

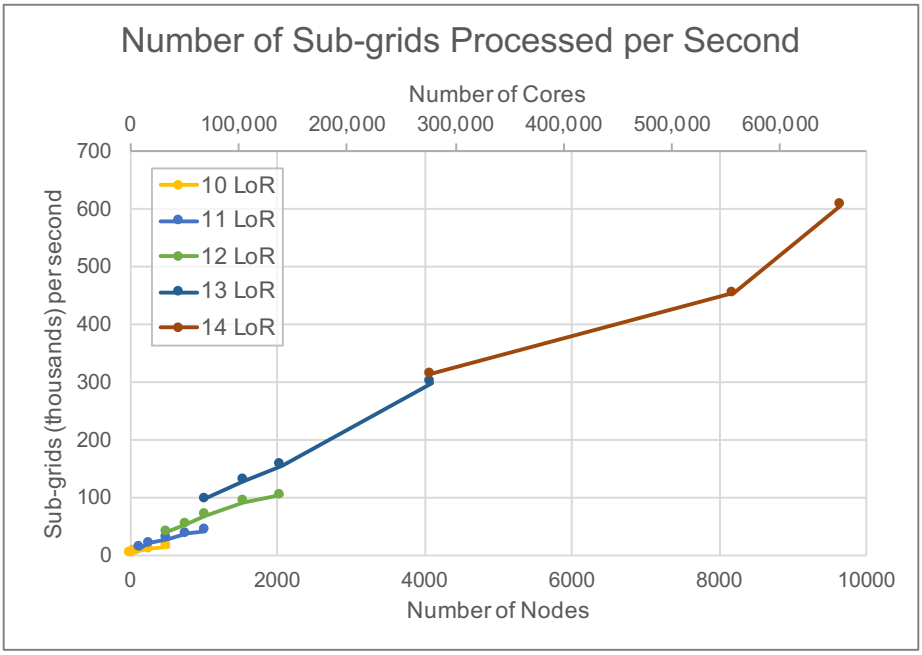

Figure 7. The results of OctoTiger strong scaling runs up to 655,520 cores on Cori for different problem sizes – e.g. different maximum LoR. This graph indicates close to perfect scalability for each problem size, with a clear improvement of performance from one LoR to the next because of increased latency hiding and parallel work due to higher oversubscription (higher number of sub-grids per core). The larger problem sizes cannot be run on a smaller number of cores because they will exceed memory capacity.

reading concurrently from 2048 nodes while performing initialization. The data was read from the Burst Buffer, spread across 50 Burst Buffer nodes. This only uses a fraction of the available bandwidth. Future work will improve this with proper support for parallel I/O. We started the scaling experiment for the 14 LoR problem at 4,096 nodes as it does not fit in memory on a smaller number of compute nodes. On the other hand, the scaling of the 13 LoR problem is limited by the lack of work, as the number of sub-grids per core drops to ~51 at 4,096 nodes, causing the parallel efficiency in this case to be reduced to ~87%.

8. Conclusions and Implications

The OctoTiger simulations presented here show the onset of a stable mass transfer stream and the initial few orbits during which an accretion disk forms with excellent detail. Additional orbits are needed for the disk to fully develop and reach a quasi-steady state. Ultimately the disk will transport angular momentum outward and tidal forces will return this angular momentum to the orbit. The stability and final fate of the binary do depend on this further evolution, but the intent of this paper is to demonstrate the capability of

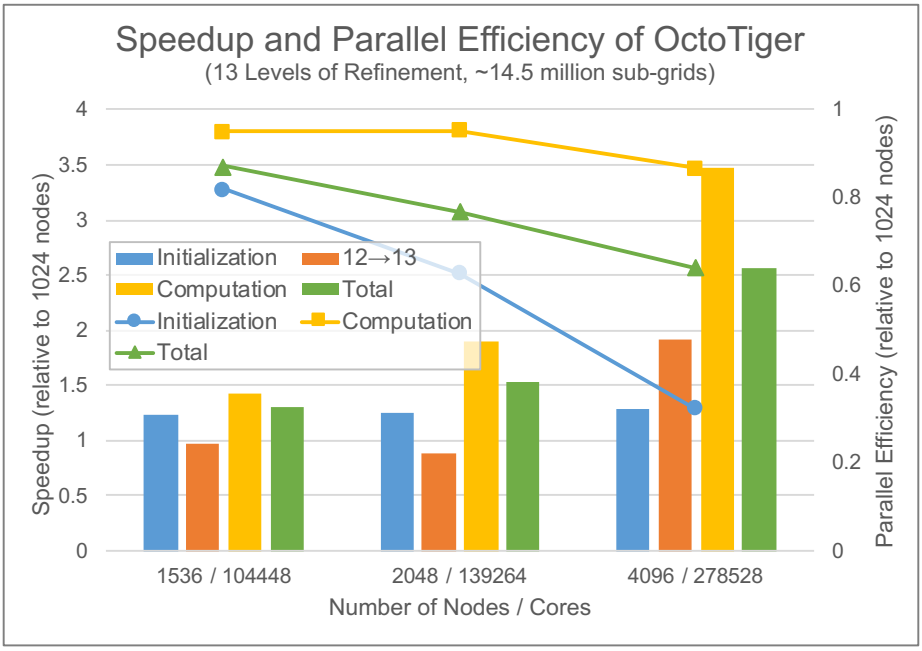

Figure 8. Speedup (bars, left axis) and parallel efficiency (lines, right axis) of OctoTiger strong scaling a problem with 13 LoR up to 1,024 Knights Landing nodes / 69,632 cores on Cori. The graphs show separate speedup and parallel efficiency results for three application stages (initialization, initial regridding from the 12 LoR restart file to 13 LoR, and the actual computation). The computational phase achieves a speedup of ~3.46× when scaling from 1,024 to 4,096 nodes, corresponding to a parallel efficiency of ~87%.

OctoTiger and HPX. Notably, futurized HPX applications can strong scale out to hundreds of thousands of cores. As for application portability, we note that the same source code we ran at scale on Cori we also ran on an Apple Laptop using HPX and compiled with gcc.

The OctoTiger advances show the potential to substantially reduce the time-to-solution for high fidelity DWD merger simulations. Experiments that previously took weeks or months of computational effort can now be performed in a matter of days on petascale systems like Cori. This scalability will make it possible to simulate evolutions over more orbits and allow us to make these simulations more realistic by including more compute-intensive physical effects such as radiative transfer, light curve generation, radiation hydrodynamics and nuclear reactions. Code development along these lines is already underway.

This work also has several implications for parallel programming and future architectures. The AMT runtime systems are a powerful and viable addition to the current set of prevalent parallel programming models. Our work demonstrates that it is not only possible to utilize these emerging tools to perform on the largest scales, but also that it might even be desirable to leverage the latency hiding, finer-grained parallelism and natural support for

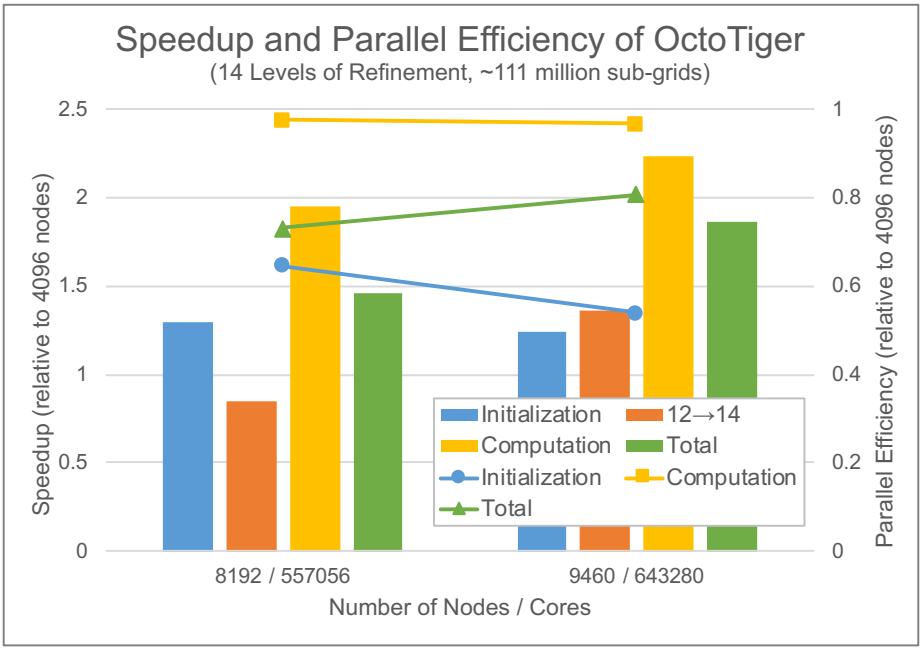

Figure 9. Speedup (bars, left axis) and parallel efficiency (lines, right axis) of OctoTiger strong scaling a problem with 14 LoR up to 4,096 Knights Landing nodes / 278,528 cores on Cori. The graphs show separate speedup and parallel efficiency results for three application stages (initialization, initial regridding from the 12 LoR restart file to 14 LoR, and the actual computation). The computational phase reaches a speedup of ~2.24× when scaling from 4,096 to 9,460 nodes corresponding to a parallel efficiency of ~96.8%.

heterogeneity that the AMT model exposes. As more and more applications choose to utilize this model, future hardware architectures will be encouraged to better support the needs of AMTs by adding features such as faster user-space context switching, task queues, and global address space facilities.

The standard C++ parallelism, concurrency, and memory models are easily programmable, portable and performant. Further additions to the ISO C++ Standard will allow the world's ~4.4 million C++ programmers (Kazakova 2015) to write applications that generate billions of tasks across hundreds of thousands of heterogeneous cores within a hierarchical and heterogeneous memory space. Leveraging concepts such as `futures`, executors, and generic algorithms, the C++ Parallel Programming Model enables developers to focus attention on the application logic instead of on managing vectors, threads and network connections. Users are presented with a coherent approach to vector-, core- and node-level parallelism that both simplifies reading and writing parallel code and increases performance portability. This programming model allows the expression of all forms of parallelism at a high level of abstraction with minimal cost.

In order to be sustainable and maintainable, scientific applications must not only perform well but must also perform portably, a feat that will become difficult as hardware becomes more diverse, heterogeneous, and hierarchical. High-level abstractions will become pivotal. We believe our work lays a foundation and a potential vision for developing future performance portable applications.

Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center (NERSC), that is supported by the U.S. Department of Energy (DoE) Office of Science's (SC) Advanced Scientific Computing Research (ASCR) program (contract DE-AC02-05CH11231). We would like to thank Brandon Cook, Dave Paul, Stephen Leak, Doug Jacobson, and Jack Deslippe for NERSC technical support. Additionally the Talapas HPC cluster and the High Performance Computing Research Core Facility at the University of Oregon were used in the course of this research. Support for this work was also provided through the X-Stack program funded by the U.S. DoE SC ASCR program under contracts DE-SC0008638, DE-SC0008714, and DE-AC02-05CH11231. This work has received funding from the European Union's Horizon 2020 research and innovation programm under grant agreement 671603 (AllScale). The development of the OctoTiger code is supported through the National Science Foundation award 1240655 (STAR). This research was partially supported under U.S. DoE contract DE-AC02-05CH11231 for Lawrence Berkeley National Laboratory that is operated by the University of California, contract DE-AC52-06NA25396 for Los Alamos National Laboratory that is operated by Los Alamos National Security, LLC (LA-UR-17-31311), and contract DE-AC52-07NA27344 for Lawrence Livermore National Laboratory operated by Lawrence Livermore National Security, LLC. Further, we would like to acknowledge the Center for Computation and Technology at Louisiana State University, the Department of Computer Science 3 - Computer Architecture at the University of Erlangen Nuremberg, and the Institute for Parallel and Distributed Systems at the University of Stuttgart. Comments on the paper content by Hans Johansen and Tim Mattson are greatly appreciated. We further acknowledge the following individuals for various contributions: Agustín (K-Ballo) Bergé, Kundan Kadam, Joel E. Tohline, Nigel Tan, Dietmar Fey, Ram Ramanujam, Nick Chaimov, and Allen D. Malony.

References

Anderson M, Brodowicz M, Kaiser H, Lelbach BA and Sterling T (2013) Tabulated Equations of State with a Many-tasking Execution Model. In: *Proceedings of the IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW)*, Workshop on Large-Scale Parallel Processing (LSPP). ISBN 978-1-4799-1372-5, pp. 1691–1699. DOI:10.1109/IPDPSW.2013.162. `https://stellar.cct.lsu.edu/pubs/tabulated_eos.pdf`.

Appel AW and Jim T (1989) Continuation-Passing, Closure-Passing Style. In: *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ISBN 0-89791-294-2, pp. 293–302. DOI: 10.1145/75277.75303. `https://dx.doi.org/10.1145/75277.75303`.

Bauer M, Treichler S, Slaughter E and Aiken A (2012) Legion: Expressing Locality and Independence with Logical Regions. In: *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, art. id 66. ISBN 978-1-4673-0805-2. DOI:10.1109/SC.2012.71. `https://dx.doi.org/10.1109/SC.2012.71`.

Bierman G, Russo C, Mainland G, Meijer E and Torgersen M (2012) Pause 'N' Play: Formalizing Asynchronous C#. In: *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*. ISBN 978-3-642-31056-0, pp. 233–257. DOI:10.1007/978-3-642-31057-7_12. `https://dx.doi.org/10.1007/978-3-642-31057-7_12`.

Boost (2017) Boost C++ Libraries 1.63.0 source and binary distributions. `{https://www.boost.org/users/history/version_1_63_0.html}`. Available under the Boost Software License 1.0 (a BSD-style open source license).

Byerly ZD, Lelbach BA, Tohline JE and Marcello DC (2014) A Hybrid Advection Scheme for Conserving Angular Momentum on a Refined Cartesian Mesh. *Astrophysical Journal, Supplement (ApJS)* 212(2, art. id 23). DOI:10.1088/0067-0049/212/2/23. `http://adsabs.harvard.edu/abs/2014ApJS..212...23B`.

C++ Standards Committee (2011) ISO/IEC 14882:2011, Standard for Programming Language C++ (C++11). Technical report, ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee). `https://wg21.link/N3337`, last publicly available draft.

C++ Standards Committee (2017a) ISO/IEC DIS 14882, Working Draft, Standard for Programming Language C++ (C++17). Technical report, ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee). `https://wg21.link/N4659`, last publicly available draft.

C++ Standards Committee (2017b) ISO/IEC TS 22277, Programming Languages – C++ Extensions for Coroutines. Technical report, ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee). `https://wg21.link/N4663`, last publicly available draft.

Chamberlain BL, Callahan D and Zima HP (2007) Parallel Programmability and the Chapel Language. *International Journal of High Performance Computing Applications (IJHPCA)* 21(3): 291–312. DOI:10.1177/1094342007078442. `https://dx.doi.org/10.1177/1094342007078442`.

Charles P, Grothoff C, Saraswat V, Donawa C, Kielstra A, Ebcioglu K, von Praun C and Sarkar V (2005) X10: An Object-Oriented Approach to Non-Uniform Cluster Computing. *ACM SIGPLAN Notices* 40(10): 519–538. DOI:10.1145/1103845.1094852. https://dx.doi.org/10.1145/1103845.1094852.

Dan M, Rosswog S, Brüggen M and Podsiadlowski P (2014) The Structure and Fate of White Dwarf Merger Remnants. *Monthly Notices of the Royal Astronomical Society (MNRAS)* 438(1): 14–34. DOI:10.1093/mnras/stt1766. http://adsabs.harvard.edu/abs/2014MNRAS.438...14D.

Dan M, Rosswog S, Guillochon J and Ramirez-Ruiz E (2011) Prelude to A Double Degenerate Merger: The Onset of Mass Transfer and Its Impact on Gravitational Waves and Surface Detonations. *Astrophysical Journal (ApJ)* 737(2, art. id 89). DOI:10.1088/0004-637X/737/2/89. http://adsabs.harvard.edu/abs/2011ApJ...737...89D.

Dan M, Rosswog S, Guillochon J and Ramirez-Ruiz E (2012) How the Merger of Two White Dwarfs Depends on Their Mass Ratio: Orbital Stability and Detonations at Contact. *Monthly Notices of the Royal Astronomical Society (MNRAS)* 422(3): 2417–2428. DOI:10.1111/j.1365-2966.2012.20794.x. http://adsabs.harvard.edu/abs/2012MNRAS.422.2417D.

Dehnen W (2000) A Very Fast and Momentum-conserving Tree Code. *Astrophysical Journal, Letters (ApJL)* 536(1): L39–L42. DOI:10.1086/312724. http://adsabs.harvard.edu/abs/2000ApJ...536L..39D.

Dekate C, Anderson M, Brodowicz M, Kaiser H, Lelbach BA and Sterling T (2012) Improving the Scalability of Parallel Nbody Applications with an Event Driven Constraint Based Execution Model. *International Journal of High Performance Computing Applications (IJHPCA)* 26(3): 319–332. DOI:10.1177/1094342012440585. https://arxiv.org/abs/1109.5190.

Desprésa B and Labourasse E (2015) Angular Momentum Preserving Cell-Centered Lagrangian and Eulerian Schemes on Arbitrary Grids. *Journal of Computational Physics* 290: 28–54. DOI:10.1016/j.jcp.2015.02.032. https://dx.doi.org/10.1016/j.jcp.2015.02.032.

deSupinski BR, Olivier SL, Terboven C, Chapman BM and Mullerr MS (2017) Editors: Scaling OpenMP for Exascale Performance and Portability - 13th International Workshop on OpenMP, IWOMP 2017, Stony Brook, NY, USA, Proceedings. *SpingerLink: Lecture Notes in Computer Science* .

Doerfer D, Deslippe J, Williams S, Oliker L, Cook B, Kurth T, Lobet M, Malas T, Vay JL and Vincenti H (2016) Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor. In: *Proceedings of the Intel Xeon Phi User Group Workshop Annual US Meeting*. https://crd.lbl.gov/assets/Uploads/ixpug16-roofline.pdf.

Dongarra J, London K, Moore S, Mucci P and Terpstra D (2001) Using PAPI for Hardware Performance Monitoring on Linux Systems. In: *Proceedings of the International Conference on Linux Clusters: The HPC Revolution*. www.netlib.org/utk/people/JackDongarra/PAPERS/papi-linux.pdf.

D'Souza MCR, Motl PM, Tohline JE and Frank J (2006) Numerical Simulations of the Onset and Stability of Dynamical Mass Transfer in Binaries. *Astrophysical*

Journal (ApJ) 643(1): 381–401. DOI:10.1086/500384. http://adsabs.harvard.edu/abs/2006ApJ...643..381D.

Eschweiler D, Wagner M, Geimer M, Knüpfer A, Nagel WE and Wolf F (2012) Open Trace Format 2: The Next Generation of Scalable Trace Formats and Support Libraries. In: *Advances in Parallel Computing*, volume 22. pp. 481–490. DOI: 10.3233/978-1-61499-041-3-481. https://goo.gl/LVoPi5.

GNU (2017) GNU Compiler Collection 6.3.0 source distributions. https://ftp.gnu.org/gnu/gcc/gcc-6.3.0/. Available under the GNU General Public License version 3.

Guillochon J, Dan M, Ramirez-Ruiz E and Rosswog S (2010) Surface Detonations in Double Degenerate Binary Systems Triggered by Accretion Stream Instabilities. *Astrophysical Journal, Letters (ApJL)* 709(1): L64–L69. DOI:10.1088/2041-8205/709/1/L64. http://adsabs.harvard.edu/abs/2010ApJ...709L..64G.

He Y, Cook B, Deslippe J, Friesen B, Gerber R, Hartman-Baker R, Koniges A, Kurth T, Leak S, Yang WS et al. (2018) Preparing nersc users for cori, a cray xc40 system with intel many integrated cores. *Concurrency and Computation: Practice and Experience* 30(1).

Heller T, Kaiser H, Diehl P, Fey D and Schweitzer MA (2016) Closing the Performance Gap with Modern C++. In: *High Performance Computing, Proceedings of the International Conference on High Performance Computing Workshops (ISC Workshops)*, Workshop on Exascale Multi/Many Core Computing Systems (E-MuCoCoS). ISBN 978-3-319-46079-6, pp. 18–31. DOI:10.1007/978-3-319-46079-6_2. https://stellar.cct.lsu.edu/pubs/closing_perf_gap_isc_2016.pdf.

Heller T, Kaiser H and Iglberger K (2012) Application of the ParalleX Execution Model to Stencil-Based Problems. *Computer Science - Research and Development* 28(2-3): 253–261. DOI:10.1007/s00450-012-0217-1. https://stellar.cct.lsu.edu/pubs/isc2012.pdf.

Heller T, Kaiser H, Schäfer A and Fey D (2013) Using HPX and LibGeoDecomp for Scaling HPC Applications on Heterogeneous Supercomputers. In: *Proceedings of the ACM/IEEE Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA, SC Workshop)*, art. id 1. ISBN 978-1-4503-2508-0. DOI: 10.1145/2530268.2530269. https://stellar.cct.lsu.edu/pubs/scala13.pdf.

Hoberock J, Garland M, Kohlhoff C, Mysen C, Edwards HC and Brown G (2017) P0443R2: A Unified Executors Proposal for C++. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings* https://wg21.link/P0443R2.

Huck K, Porterfield A, Chaimov N, Kaiser H, Malony A, Sterling T and Fowler R (2015) An Autonomic Performance Environment for Exascale. *Supercomputing Frontiers and Innovations* 2(3): 49–66. DOI:10.14529/jsfi150305. https://dx.doi.org/10.14529/jsfi150305.

Intel (2017a) Intel Cilk Plus. https://software.intel.com/en-us/intel-cilk-plus.

Intel (2017b) Intel SPMD Program Compiler (ISPC). https://ispc.github.io/.

Intel (2017c) Intel Xeon Phi Processor 7250 (16GB, 1.40 GHz, 68 core) Specifications. https://ark.intel.com/products/94035/Intel-Xeon-Phi-Processor-7250-16GB-1_40-GHz-68-core.

jemalloc (2017) jemalloc GitHub repository, 4.5.0 tag. https://github.com/jemalloc/jemalloc/tree/4.5.0. Available under the 2-Clause BSD License.

Kadam K, Clayton GC, Motl PM, Marcello DC and Frank J (2017) Numerical Simulations of Close and Contact Binary Systems Having Bipolytropic Equation of State. In: *Proceedings of the American Astronomical Society (AAS)*, meeting 229, art. id 433.14. http://adsabs.harvard.edu/abs/2017AAS...22943314K.

Kadam K, Motl PM, Frank J, Clayton GC and Marcello DC (2016) A Numerical Method for Generating Rapidly Rotating Bipolytropic Structures in Equilibrium. *Monthly Notices of the Royal Astronomical Society (MNRAS)* 462(2): 2237–2245. DOI:10.1093/mnras/stw1814. http://adsabs.harvard.edu/abs/2016MNRAS.462.2237K.

Kaiser H, Heller T, Bourgeois D and Fey D (2015) Higher-level Parallelization for Local and Distributed Asynchronous Task-based Programming. In: *Proceedings of the ACM/IEEE International Workshop on Extreme Scale Programming Models and Middleware (ESPM, SC Workshop)*. ISBN 978-1-4503-3996-4, pp. 29–37. DOI:10.1145/2832241.2832244. https://stellar.cct.lsu.edu/pubs/executors_espm2_2015.pdf.

Kaiser H, Heller T, Lelbach BA, Serio A and Fey D (2014) HPX: A Task Based Programming Model in a Global Address Space. In: *Proceedings of the International Conference on Partitioned Global Address Space Programming Models (PGAS)*, art. id 6. ISBN 978-1-4503-3247-7. DOI:10.1145/2676870.2676883. https://stellar.cct.lsu.edu/pubs/pgas14.pdf.

Katz MP, Zingale M, Calder AC, Swesty FD, Almgren AS and Zhang W (2016) White Dwarf Mergers on Adaptive Meshes. I. Methodology and Code Verification. *Astrophysical Journal (ApJ)* 819(2, art. id 94). DOI:10.3847/0004-637X/819/2/94. http://adsabs.harvard.edu/abs/2016ApJ...819...94K.

Kazakova A (2015) C/C++ Facts We Learned Before Going Ahead with CLion. Technical report, Jetbrains. https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/.

Kevin Huck (2017) APEX Performance Monitoring Framework GitHub repository, commit 58214cf. https://github.com/khuck/xpress-apex/commit/58214cfba5ce6ddb2682713329687c56625c580e. Available under the Boost Software License 1.0 (a BSD-style open source license).

Knüpfer A, Brunst H, Doleschal J, Jurenz M, Lieber M, Mickler H, Müller MS and Nagel WE (2008) The Vampir Performance Analysis Tool-Set. In: *Tools for High Performance Computing: Proceedings of the International Workshop on Parallel Tools for High Performance Computing*. ISBN 978-3-540-68561-6, pp. 139–155. DOI:10.1007/978-3-540-68564-7_9. https://dx.doi.org/10.1007/978-3-540-68564-7_9.

Kretz M (2015a) *Extending C++ for Explicit Data-Parallel Programming via SIMD Vector Types*. PhD Thesis, Goethe University Frankfurt. DOI:10.13140/RG.2.1.2355.4323. http://publikationen.ub.uni-frankfurt.de/frontdoor/index/index/docId/38415.

Kretz M (2015b) N4395: SIMD Types: ABI Considerations. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings* https://wg21.

link/N4395.

Kretz M (2015c) N4454: SIMD Types Example: Matrix Multiplication. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings* https://wg21.link/N4454.

Kretz M (2016) P0350R0: Integrating datapar with Parallel Algorithms and Executors. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings* https://wg21.link/P0350R0.

Kretz M (2017) P0214R3: Data-Parallel Vector Types & Operations. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings* https://wg21.link/P0214R3.

Kretz M and Lindenstruth V (2011) Vc: A C++ Library for Explicit Vectorization. *Software: Practice and Experience* 42(11): 1409–1430. DOI:10.1002/spe.1149. https://dx.doi.org/10.1002/spe.1149.

Kumar R, Tullsen DM, Ranganathan P, Jouppi NP and Farkas KI (2004) Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In: *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*. ISBN 0-7695-2143-6, pp. 64–75. DOI:10.1109/ISCA.2004.1310764. https://dx.doi.org/10.1109/ISCA.2004.1310764.

Kurganov A and Tadmor E (2000) New High-Resolution Central Schemes for Nonlinear Conservation Laws and Convection-Diffusion Equations. *Journal of Computational Physics* 160(1): 241–282. DOI:10.1006/jcph.2000.6459. https://dx.doi.org/10.1006/jcph.2000.6459.

Lelbach BA, Byerly ZD, Marcello DC, Clayton GC and Kaiser H (2013) Octopus: A Scalable AMR Toolkit for Astrophysics. In: *Scientific Computing Around Louisiana (SCALA)*. http://stellar.cct.lsu.edu/pubs/SCALA2013_lelbach.pdf.

Lindblom L, Tohline JE and Vallisneri M (2001) Nonlinear Evolution of the r-Modes in Neutron Stars. *Physical Review Letters (PRL)* 86(7): 1152–1155. DOI:10.1103/PhysRevLett.86.1152. http://adsabs.harvard.edu/abs/2001PhRvL..86.1152L.

Linux Kernel Organization I (2017) Linux Power Capping Framework. https://www.kernel.org/doc/Documentation/power/powercap/powercap.txt.

Marcello DC (2017) A Very Fast and Angular Momentum Conserving Tree Code. *accepted for publication by the American Astronomical Society (AAS) Journals* http://adsabs.harvard.edu/abs/2017arXiv170606989M.

Marcello DC, Kadam K, Clayton GC, Frank J, Kaiser H and Motl PM (2016) Introducing Octo-tiger/HPX: Simulating Interacting Binaries with Adaptive Mesh Refinement and the Fast Multipole Method. In: *Proceedings of the International Conference on Accretion Processes in Cosmic Sources*. http://apcs2016.iaps.inaf.it.

Martin SJ and Kappel M (2014) Cray XC30 Power Monitoring and Management. In: *Proceedings of the Cray User Group Conference*. https://cug.org/proceedings/cug2014_proceedings/includes/files/pap130.pdf.

Menon H, Wesolowski L, Zheng G, Jetley P, Kale L, Quinn T and Governato F (2015) Adaptive Techniques for Clustered N-body Cosmological Simulations.

Computational Astrophysics and Cosmology 2, art. id 1. DOI:10.1186/s40668-015-0007-9. http://adsabs.harvard.edu/abs/2015ComAC...2....1M.

Montiel EJ, Clayton GC, Marcello DC and Lockman FJ (2015) What Is the Shell Around R Coronae Borealis? *Astronomical Journal (AJ)* 150(1, art. id 14). DOI:10.1088/0004-6256/150/1/14. http://adsabs.harvard.edu/abs/2015AJ....150...14M.

Motl PM, Frank J, Staff J, Clayton GC, Fryer CL, Even W, Diehl S and Tohline JE (2017) A Comparison of Grid-based and SPH Binary Mass-transfer and Merger Simulations. *Astrophysical Journal, Supplement (ApJS)* 229(2, art. id 27). DOI:10.3847/1538-4365/aa5bde. http://adsabs.harvard.edu/abs/2017ApJS..229...27M.

Motl PM, Frank J, Tohline JE and D'Souza MCR (2007) The Stability of Double White Dwarf Binaries Undergoing Direct-Impact Accretion. *Astrophysical Journal (ApJ)* 670(2): 1314–1325. DOI:10.1086/522076. http://adsabs.harvard.edu/abs/2007ApJ...670.1314M.

NERSC (2017a) Cray MPICH 7.4.4 documentation for NERSC Cori. https://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/.

NERSC (2017b) National Energy Research Scientific Computing Center (NERSC) Cori System Details. http://www.nersc.gov/users/computational-systems/cori/configuration/.

Open MPI (2017) hwloc 1.11.6 source and binary distributions. https://www.open-mpi.org/software/hwloc/v1.11/. Available under the 3-Clause BSD License.

Ott CD, Ou S, Tohline JE and Burrows A (2005) One-armed Spiral Instability in a Low-T/—W— Postbounce Supernova Core. *Astrophysical Journal, Letters (ApJL)* 625(2): L119–L122. DOI:10.1086/431305. http://adsabs.harvard.edu/abs/2005ApJ...625L.119O.

Pakmor R, Hachinger S, Röpke FK and Hillebrandt W (2011) Violent Mergers of Nearly Equal-Mass White Dwarf as Progenitors of Subluminous Type Ia Supernovae. *Astronomy & Astrophysics* 528, art. id A117. DOI:10.1051/0004-6361/201015653. http://adsabs.harvard.edu/abs/2011A&A...528A.117P.

Raskin C, Scannapieco E, Fryer C, Rockefeller G and Timmes FX (2012) Remnants of Binary White Dwarf Mergers. *Astrophysical Journal (ApJ)* 746(1, art. id 62). DOI:10.1088/0004-637X/746/1/62. http://adsabs.harvard.edu/abs/2012ApJ...746...62R.

Sankrit R and Blair W (2004) X-ray, Optical & Infrared Composite (CXO/HST/SST) of Kepler's Supernova Remnant. Technical report, NASA/ESA/JHU. http://chandra.harvard.edu/photo/printgallery/2004.

Schaller M, Gonnet P, Chalk ABG and Draper PW (2016) SWIFT: Using Task-Based Parallelism, Fully Asynchronous Communication, and Graph Partition-Based Domain Decomposition for Strong Scaling on More Than 100,000 Cores. In: *Proceedings of the ACM Platform for Advanced Scientific Computing Conference (PASC)*, art. id 2. ISBN 978-1-4503-4126-4. DOI:10.1145/2929908.292991. https://arxiv.org/abs/1606.02738.

Schwab J, Shen KJ, Quataert E, Dan M and Rosswog S (2012) The Viscous Evolution of White Dwarf Merger Remnants. *Monthly Notices of the Royal Astronomical Society (MNRAS)* 427(1): 190–203. DOI:10.1111/j.1365-2966.2012.21993.x. http://adsabs.harvard.edu/abs/2012MNRAS.427..190S.

Shende S and Malony A (2006) The TAU Parallel Performance System. *International Journal of High Performance Computing Applications (IJHPCA)* 20(2): 287–311. DOI:10.1177/1094342006064482. https://dx.doi.org/10.1177/1094342006064482.

Sodani A (2015) Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. In: *Hot Chips Symposium*. https://goo.gl/a6haUm.

Staff JE, Menon A, Herwig F, Even W, Fryer CL, Motl PM, Geballe T, Pignatari M, Clayton GC and Tohline JE (2012) Do R Coronae Borealis Stars Form from Double White Dwarf Mergers? *Astrophysical Journal (ApJ)* 757(1, art id. 76). DOI:10.1088/0004-637X/757/1/76. http://adsabs.harvard.edu/abs/2012ApJ...757...76S.

STE||AR Group (2017a) HPX GitHub repository. https://github.com/STEllAR-GROUP/hpx. Available under the Boost Software License 1.0 (a BSD-style open source license).

STE||AR Group (2017b) HPX GitHub repository, commit 19bd11a. https://github.com/STEllAR-GROUP/hpx/commit/19bd11a521f878580316f7f4c7754298b7b45563. Available under the Boost Software License 1.0 (a BSD-style open source license).

STE||AR Group (2017c) Octopus AMR Framework GitHub repository. https://github.com/STEllAR-GROUP/octopus. Available under the Boost Software License 1.0 (a BSD-style open source license).

STE||AR Group (2017d) OctoTiger AMR Framework GitHub repository. https://github.com/STEllAR-GROUP/octotiger. Available under the Boost Software License 1.0 (a BSD-style open source license).

STE||AR Group (2017e) Octotiger AMR Framework GitHub repository, commit 0b6cd60. https://github.com/STEllAR-GROUP/octotiger/commit/0b6cd60d0405be700f191f03e2a011f7503b7af1. Available under the Boost Software License 1.0 (a BSD-style open source license).

Syme D, Petricek T and Lomov D (2011) The F# Asynchronous Programming Model. In: *Proceedings of the International Conference on Practical Aspects of Declarative Languages (PADL)*. ISBN 978-3-642-18377-5, pp. 175–189. DOI:10.1007/978-3-642-18378-2_15. https://dx.doi.org/10.1007/978-3-642-18378-2_15.

USDOE (2012) X-Stack: Programming Challenges, Runtime Systems, and Tools (DoE-FOA-0000619). Technical report, US Department of Energy Office of Science. https://science.energy.gov/~/media/grants/pdf/foas/2012/SC_FOA_0000619.pdf.

Wheeler K, Murphy R and Thain D (2008) Qthreads: An API for Programming with Millions of Lightweight Threads. In: *Proceedings of the IEEE International Symposium on Parallel Distributed Processing, Workshops and PhD Forum (IPDPSW)*, Workshop on Multithreaded Architectures and Applications (MTAAP). ISBN 978-1-4244-1693-6. DOI:10.1109/IPDPS.2008.4536359. https:

//dx.doi.org/10.1109/IPDPS.2008.4536359.

XPRESS APEX (2017) APEX Performance Monitoring Framework GitHub repository. https://github.com/khuck/xpress-apex. Available under the Boost Software License 1.0 (a BSD-style open source license).

Zingale M, Almgren AS, Bell JB, Nonaka A and Woosley SE (2009) Low Mach Number Modeling of Type IA Supernovae. IV. White Dwarf Convection. *Astrophysical Journal (ApJ)* 704(1): 196–210. DOI:10.1088/0004-637X/704/1/196. http://adsabs.harvard.edu/abs/2009ApJ...704..196Z.

Željko Ivezić, Axelrod TS, Brandt WN, Burke DL, Claver CF, Connolly AJ, Cook KH, Gee P, Gilmore DK, Jacoby SH, Jones RL, Kahn SM, Kantor JP, Krabbendam VL, Lupton RH, Monet DG, Pinto PA, Saha A, Schalk TL, Schneider DP, Strauss MA, Stubbs CW, Sweeney DW, Szalay A, Thaler JJ, Tyson JA and the LSST Collaboration (2008) Large Synoptic Survey Telescope: From Science Drivers To Reference Design. *Serbian Astronomical Journal* (176): 1–13. DOI: 10.2298/SAJ0876001I. http://adsabs.harvard.edu/abs/2008SerAJ.176....1I.