7-1-2019

Harnessing billions of tasks for a scalable portable hydrodynamic simulation of the merger of two stars

Thomas Heller
Friedrich-Alexander-Universität Erlangen-Nürnberg

Bryce Adelstein Lelbach
ÅF AB

Kevin A. Huck
University of Oregon

John Biddiscombe
ÅF AB

Patricia Grubel
ÅF AB

See next page for additional authors

Recommended Citation

Heller, T., Lelbach, B., Huck, K., Biddiscombe, J., Grubel, P., Koniges, A., Kretz, M., Marcello, D., Pfander, D., Serio, A., Frank, J., Clayton, G., Pflüger, D., Eder, D., & Kaiser, H. (2019). Harnessing billions of tasks for a scalable portable hydrodynamic simulation of the merger of two stars. *International Journal of High Performance Computing Applications, 33* (4), 699-715. https://doi.org/10.1177/1094342018819744

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

Research Article

The International Journal of High Performance Computing Applications
2019, Vol. 33(4) 699–715

DOI: 10.1177/1094342018819744
journals.sagepub.com/home/hpc
SAGE

Harnessing billions of tasks for a scalable portable hydrodynamic simulation of the merger of two stars

Thomas Heller[1,2], Bryce Adelstein Lelbach[2,3], Kevin A Huck[4], John Biddiscombe[2,5], Patricia Grubel[2,6], Alice E Koniges[7], Matthias Kretz[8], Dominic Marcello[2,9], David Pfander[10], Adrian Serio[2,9], Juhan Frank[9], Geoffrey C Clayton[9], Dirk Pflüger[10], David Eder[11] and Hartmut Kaiser[2,9]

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 National Energy Research Scientific Computing Center'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 toward 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 article, 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 trade-offs between energy efficiency, resilience, and performance.

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

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Bavaria, Germany
[2] The STE||AR Group
[3] NVIDIA, Santa Clara, CA, USA
[4] University of Oregon, Eugene, OR, USA
[5] Swiss National Supercomputing Centre, Lugano, Switzerland
[6] Los Alamos National Laboratory, NM, USA
[7] Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[8] GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, Germany
[9] Louisiana State University, Baton Rouge, LA, USA
[10] University of Stuttgart, Baden-Württemberg, Germany
[11] Lawrence Livermore National Laboratory, Livermore, CA, USA

Corresponding author:
Alice E Koniges, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720-8150, USA.
Email: aekoniges@lbl.gov

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 (Željko 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 1000 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 Section 2, we describe the stellar merger problem we wish to solve. In Section 3, we briefly describe OctoTiger. In Section 4, we lay out the HPX model and discuss Futurization, and in Section 5, we describe how OctoTiger uses futurization. We executed this model on National Energy Research Scientific Computing Center's (NERSC) Cori machine, described in Section 6. The results of node-level and full-system scaling are detailed in Section 7. In terms of portability, we note that the same source code running at scale on Cori can also be compiled and run on an Apple Laptop. In Section 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 1a supernova (see Figure 1) (Dan et al., 2011, 2012; Guillochon et al., 2010; Katz et al., 2016; Pakmor et al., 2011; Zingale et al., 2009). White dwarf mergers that are not massive enough to cause a Type 1a supernova to leave an observable remnant can be studied by continuing the model past the point of merger (Dan et al., 2014; Montiel et al., 2015; Raskin et al., 2012; Schwab et al., 2012; Staff 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 presents the early stages of the formation of the

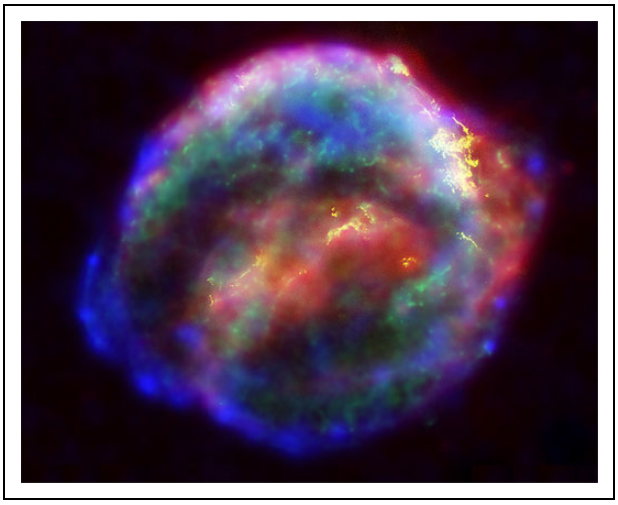

Figure 1. A false-color composite image of the supernova remnant nebula from SN 1604 (Sankrit and Blair, 2004). Visible to the naked eye, Kepler's star was brighter at its peak than any other star in the night sky, with an apparent magnitude of −2.5. It was visible during the day for over 3 weeks. While discovered in Kepler's time, new studies are trying to resolve the question of whether or not DWD mergers cause Type 1a supernova. DWD: double white dwarf.

mass-transfer stream and accretion disk in a binary star where the primary star or accretor (on the left) is five 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 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 min and 40 min. Periodic instabilities in the disc can result in dwarf nova. The buildup of helium in the disc can periodically detonate as a subluminous supernova known as a Type ".1a" (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 (Kadam et al., 2017; Katz et al., 2016) to address these problems. The simulations described in this article use up to 14 levels of refinement (LoRs), to simulate more than four magnitude difference in resolution of space across the

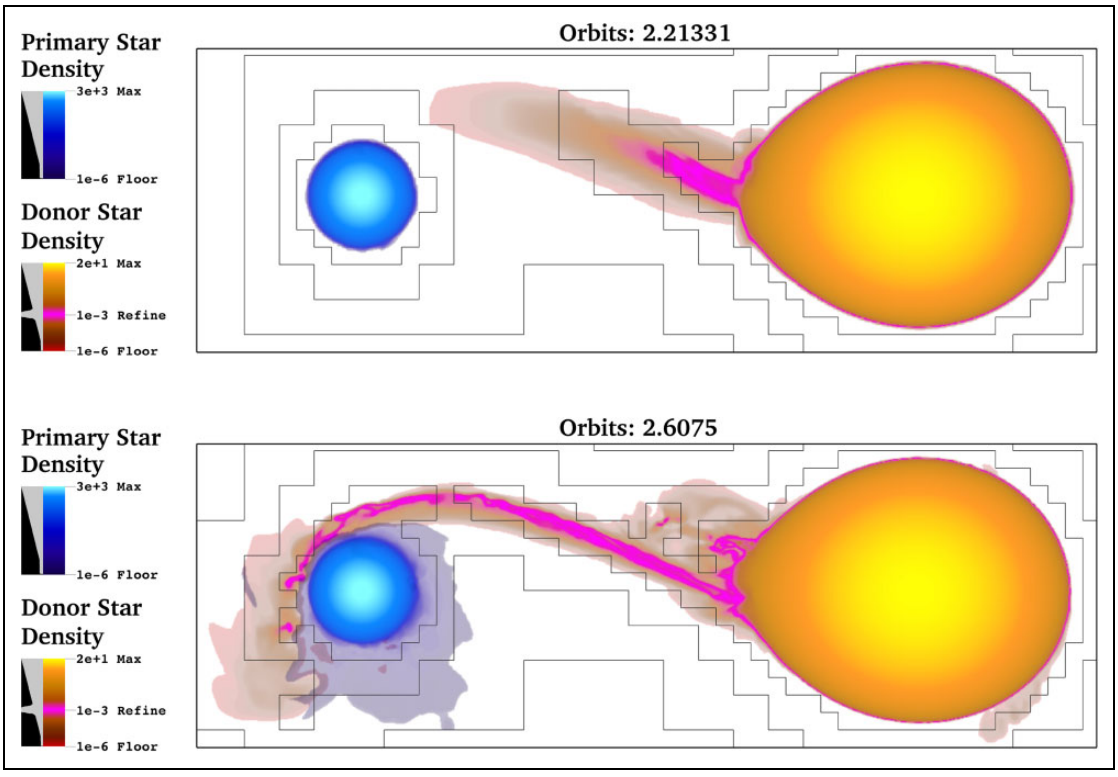

Figure 2. The early stages of mass transfer in a binary star system. The accreting star is five times more massive than the donor star.

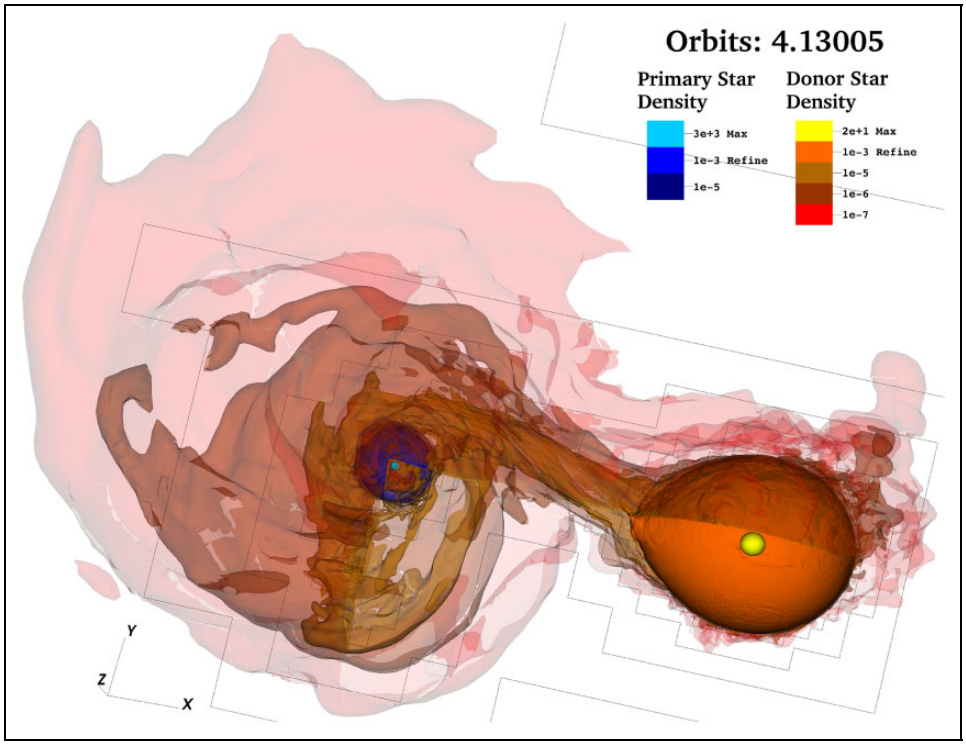

Figure 3. A 3-D contour plot of the system in Figure 2 after an accretion disc begins to form. 3-D: three-dimensional.

computational domain. Multiple coupled physics solvers with very different performance characteristics and algorithmic complexities are combined 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 article, we detail programming model updates to create *OctoTiger*, a three-dimensional (3-D) 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 by Byerly et al. (2014), D'Souza et al. (2006), Kadam et al. (2016), Lelbach et al. (2013), Lindblom et al. (2001), Motl et al. (2007, 2017), Ott et al. (2005), and 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 by 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, 2017; Marcello et al., 2016), 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 initio simulations spanning thousands of orbits, such as stable mass transfer binaries (see Figure 3).

The computational domain in OctoTiger is based on a 3-D octree AMR structure. Each node in the structure is an $N \times N \times N$ Cartesian subgrid (in this work, $N = 8$) and may be further refined into eight child nodes, each containing its own $N \times N \times N$ subgrid 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 are based on density. These refinement criteria in the current simulation are checked every 15 time steps to see whether refinement or coarsening is required, based on the minimal number of time steps required for a feature of the flow to propagate between two cells. After each refinement/coarsening step, the subgrids 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 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, 2016; Kaiser et al., 2014, 2015; 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 futures, channels, 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.

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). 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).

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 (Anderson et al., 2013; Dekate et al., 2012; Huck et al., 2015; USDOE, 2012; Wheeler et al., 2008). 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 (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 toward 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 100 K cores with 60% parallel efficiency, and ChaNGa has demonstrated scalability up to 500 K 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 article is to demonstrate the viability of the HPX programming model through a portable 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.

`Futures` are monadic data structures, for example, 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 `futures` 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).

Rewriting synchronous blocking code as a chain of wait-free continuations can be achieved through the straightforward futurization technique, as depicted in Table 1. The futurization technique allows a delay of direct (sequential) execution 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 (Bierman et al., 2012; Syme et al., 2011). 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), and executors describe where execution will occur (ex: a GPU executor, a thread pool executor, a remote procedure call (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 Section *5.1.*).

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

Table 1. *The Futurization Technique.*[a,b]

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());`

[a]This table shows sequential C++ constructs (left column) and the corresponding constructs after applying a transformation called *Futurization* (center and right columns)

[b]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, and v are variables of those types; and f, g, and h are functions.

Listing 1. Natural recursive tree traversal.

```
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 2. Futurized 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());
}
```

that require a global barrier are (1) rebalancing the 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 presented in listing 2. Instead of direct recursion, we recurse asynchronously. The computation for a given tree node, representing a subgrid in Octo-Tiger, 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`.

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 make asynchronous 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 is implemented as four distinct futurized tree traversals:

1. Check whether 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 subgrid 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. `Channels` 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. `Channels` 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, for example, after either a consumer or producer requests the epoch. `Channels` transparently buffer values on the fly when needed, allowing producers to proceed ahead of consumers—avoiding needless waiting. Our asynchronous solvers use `channels` to allow computation to proceed as far as possible and overlap communication and computation. After the solve for one substep is complete, a tree node will retrieve `futures` from its `channels` and attach continuations to the futures (via `when_all`) that will compute the next substep. The continuation will only begin executing when the necessary data for that substep have been sent to the `channels`.

A simplified example of a `channel`-based asynchronous ghost zone exchange on a two-dimensional mesh partitioned into four subgrids is shown in Figure 4. This technique allows computation to proceed as far as possible without needless waiting. First, all subgrids begin computing the first substep (red). Subgrids A, B, and D finish their computations and send ghost zone data to their neighbor's `channels`. Then, A, B, and D combine the dependencies of the next substep with `when_all` and attach a continuation that computes the next substep to the resulting `future`. C is still computing the first substep, so it has not sent the first substep ghost zone data to A or D, and the second substep 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 substep has all the data it needs, so that computation is executed (blue). When it is completed, a third substep continuation is created and the second substep ghost zone data are sent from B to A and D's `channels`, where it is effectively buffered. In OctoTiger, the dependencies are more complicated, due to 3-D 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 substep 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 substep, 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 LoR, 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:

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 are 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 toward potential standardization in the C++ programming language (Kretz, 2015b, 2015c, 2016, 2017). 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

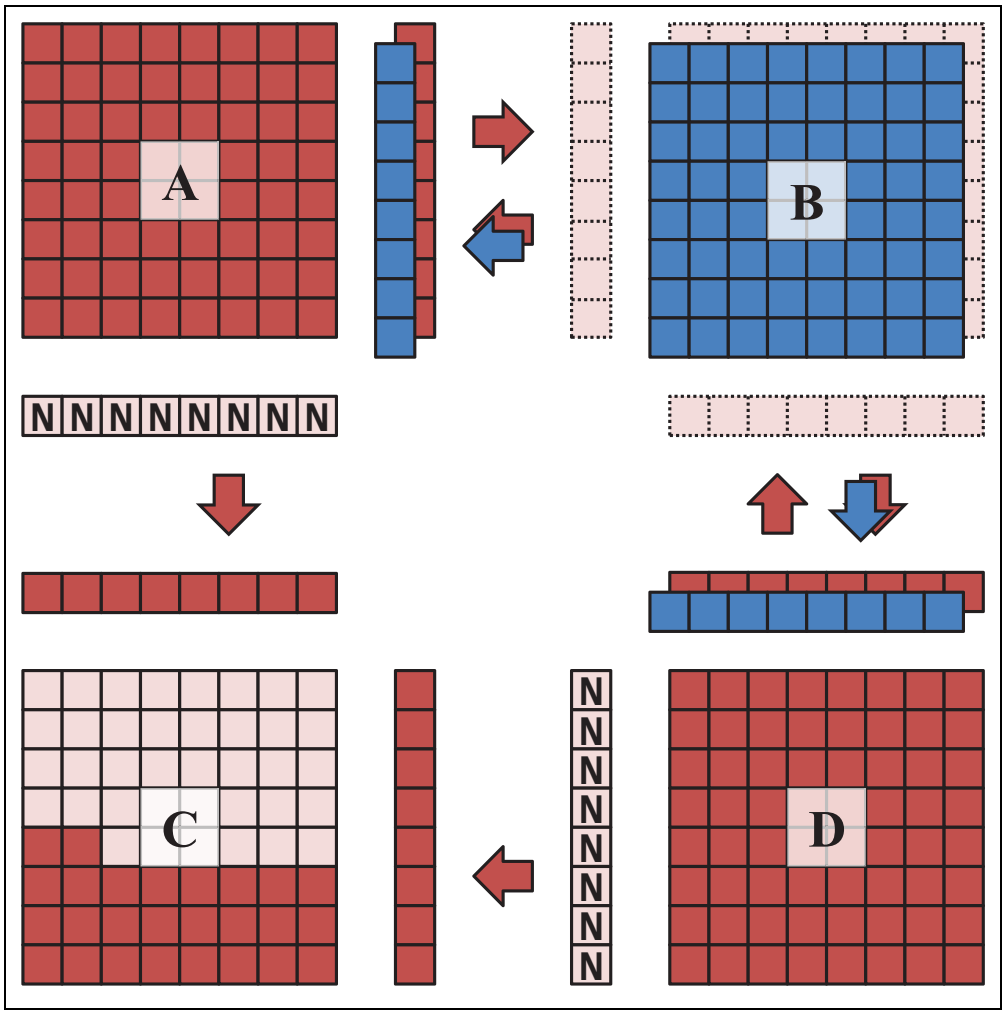

Figure 4. Depiction of ghost zone exchange patterns.

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 file system. Power and energy data are 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

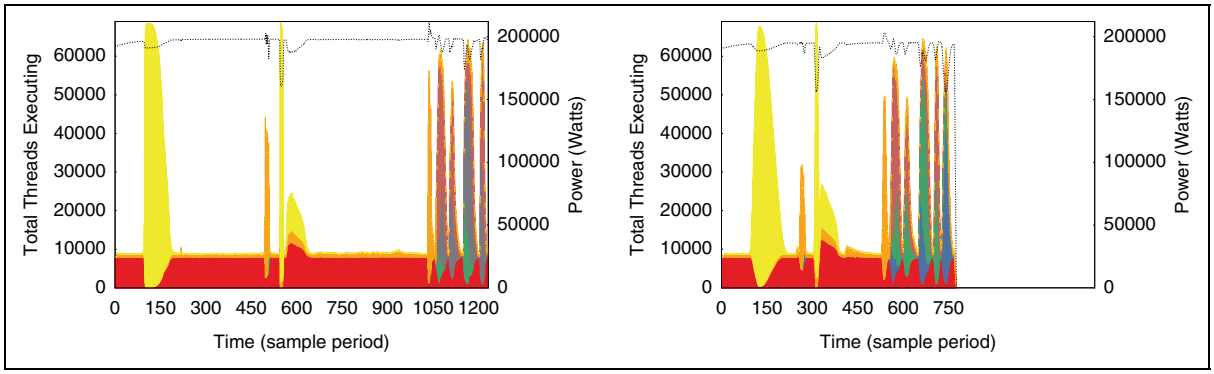

Figure 5. Concurrency views using APEX of OctoTiger running on 1024 nodes of Cori. Higher values indicate better system utilization. The figure on the left shows a profile of OctoTiger prior to Futurization. There are two long periods of serial execution: (1) after the checkpoint load and grid creation 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 (i.e. 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. APEX: autonomous performance environment for eXascale.

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.

Table 2. Software versions used in our experiments.

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)

6. Experimental setup

The primary system used for the experiments in this article is a Cray XC40 installation at the NERSC located in Berkeley, California, USA (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 9688 nodes; each node has a single Xeon Phi 7250 processor with 68 cores and 272 hyperthreads (HTs) (Intel, 2017c). Therefore, there are 658,784 cores and up to 2.6 million threads overall. The whole system has a peak double precision floating point performance of approximately 29.1 PFLOP/s is an aggregated main memory capacity of approximately 1.1 PB. Cori uses a Cray Aries interconnect and has a global bandwidth of approximately 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 16 GB 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 three different NUMA clustering modes: (1) all-to-all, (2) quad, and (3) sub-NUMA-clustering. 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 2 . 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]). Timings are reported for the major phases of the simulation, including initialization, grid creation, regridding, and the solvers.

Table 3. Number of tree nodes (subgrids) for example data set after initialization for different maximum LoRs.[a]

LoRs	Number of tree nodes	File size (GB)
7	1641	0.1
8	4361	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

LoR: level of refinement.
[a]This includes the size of the file needed to be used for initializing the computation.

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 LoRs (see Table 3). The effectiveness of the Futurization of OctoTiger was observed by profiling the application with the APEX toolkit (see Figure 5). The results presented include input/output (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 HTs per core. This experiment shows that using two HTs per core gives the best overall performance, with a parallel efficiency of approximately 87%. This is approximately 1.3× speedup over the results for one 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 subgrids 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 LoRs. Figure 7 provides an overview of the results.

Because we are strong scaling, the speedup naturally flattens off after the number of subgrids per core drops below a certain value. In our experiments, this was happening at 100 subgrids per core. The overall speedup from 10 LoR at 16 compute nodes (approximately 44 subgrids per core) to 14 LoR at 9640 compute nodes (approximately 171 subgrids per core) is approximately 342×, corresponding to a parallel efficiency of 56.8%. Since the number of subgrids 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.

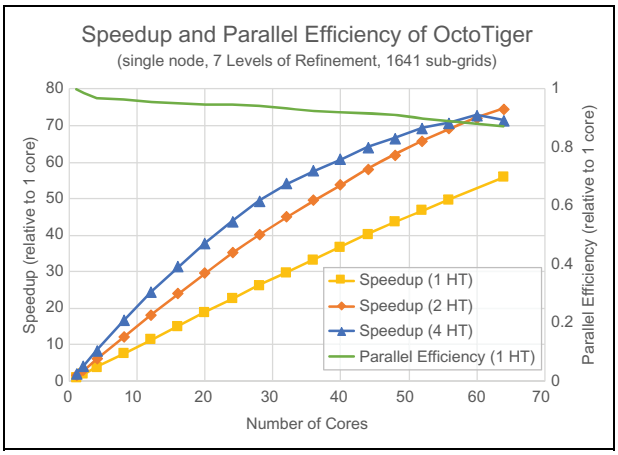

Figure 6. The speedup and parallel efficiency of OctoTiger strong scaling experiments on a single KNL node with different numbers of HTs per core is shown above. The two HTs per core case perform best. It achieves a speedup of approximately 55.9× when scaling from 1 to 64 cores, corresponding to a parallel efficiency of approximately 87%. This is approximately 1.3× speedup over the results for one HT per core. KNL: Knight's Landing; HT: hyperthread.

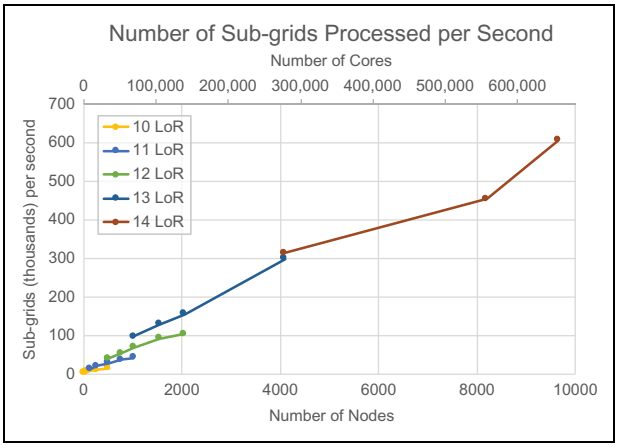

Figure 7. The results of OctoTiger strong scaling runs up to 655,520 cores on Cori for different problem sizes—for example, different maximum LoRs. This graph indicates close to perfect scalability for each problem size, with a clear improvement of performance from one LoR to the next because of increased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

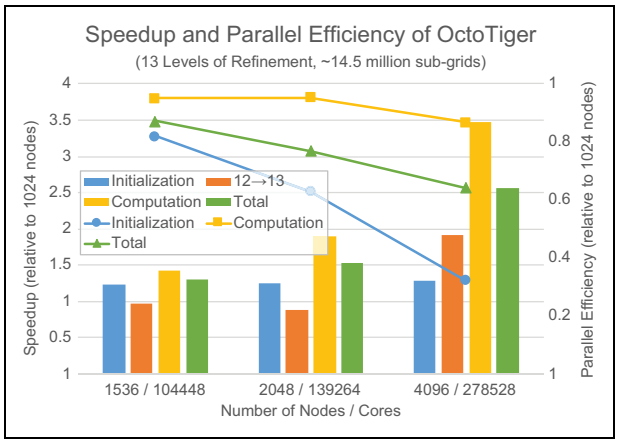

Figure 8. Speedup (bars, left axis) and parallel efficiency (lines, right axis) of OctoTiger strong scaling are a problem with 13 LoR up to 1024 KL nodes/69,632 cores on Cori. The graphs show separate speedup and parallel efficiency results for three application stages (initialization, initial regridding from the 12 LoR restart file to 13 LoR, and the actual computation). The computational phase achieves a speedup of approximately 3.46× when scaling from 1024 nodes to 4096 nodes, corresponding to a parallel efficiency of approximately 87%. LoR: level of refinement; KNL: Knight's Landing.

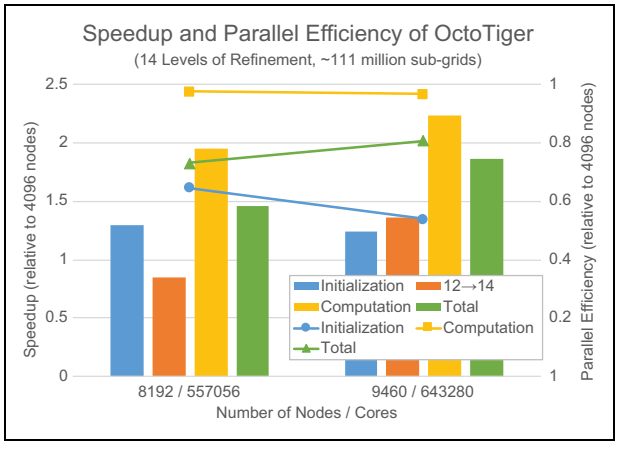

Figure 9. Speedup (bars, left axis) and parallel efficiency (lines, right axis) of OctoTiger strong scaling are a problem with 14 LoR up to 4096 KL nodes/278,528 cores on Cori. The graphs show separate speedup and parallel efficiency results for three application stages (initialization, initial regridding from the 12 LoR restart file to 14 LoR, and the actual computation). The computational phase reaches a speedup of approximately 2.24× when scaling from 4096 nodes to 9460 nodes corresponding to a parallel efficiency of approximately 96.8%. LoR: level of refinement; KNL: Knight's Landing.

These figures show that when executing the 13 LoR and 14 LoR problems, the futurized tree traversal, described in Section 5, is able to scale up to the full Cori system—for example, 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 4096 nodes to 9640 nodes). The sustained aggregated bandwidth for loading the initialization file was at 1.4 GB/s with reading concurrently from 2048 nodes while performing initialization. The data were 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 4096 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 subgrids per core drops to approximately 51 at 4096 nodes, causing the parallel efficiency in this case to be reduced to approximately 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 article is to demonstrate the capability of 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 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 approximately 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. In 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.

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.

Acknowledgements

The authors 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 HPC 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. Further, the authors 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. The authors 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.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the US Department of Energy (DoE) Office of Science's (SC) Advanced Scientific Computing Research (ASCR) program (under contract DE-AC02-05CH11231) using resources of the NERSC. Support for this work was also provided through the X-Stack program by the US DoE SC ASCR program under contracts DE-SC0008638 and DE-SC0008714. This work was also supported by the European Union's Horizon 2020 research and innovation program 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 US DoE under 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.

References

Anderson M, Brodowicz M, Kaiser H, et al. (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. Available at: 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)*, pp. 293–302. DOI: 10.1145/75277.75303. ISBN 0-89791-294-2.

Bauer M, Treichler S, Slaughter E, et al. (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)*, Salt Lake City, USA, 10–16 November 2012, Art. id 66. ISBN 978-1-4673-0805-2. DOI: 10.1109/SC.2012.71.

Bierman G, Russo C, Mainland G, et al. (2012) Pause 'N' Play: formalizing Asynchronous C#. In: *Proceedings of the European conference on object-oriented programming (ECOOP)*, Beijing, China, 11–16 June 2012, pp. 233–257. DOI: 10.1007/978-3-642-31057-7 12. ISBN 978-3-642-31056-0.

Boost (2017) Boost C++ Libraries 1.63.0 source and binary distributions. Available at: https://www.boost.org/users/history/version_1_63_0.html (accessed 25 January 2019). Available under the Boost Software License 1.0 (a BSD-style open source license).

Byerly ZD, Lelbach BA, Tohline JE, et al. (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. Available at: http://adsabs.harvard.edu/abs/2014ApJS.212 . . . 23B (accessed 25 January 2019).

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). Available at: https://wg21.link/N3337 (accessed 25 January 2019), 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). Available at: https://wg21.link/N4659 (accessed 25 January 2019) 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). Available at: https://wg21.link/N4663 (accessed 25 January 2019), 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.

Charles P, Grothoff C, Saraswat V, et al. (2005) X10: an object-oriented approach to non-uniform cluster computing. *ACM SIGPLAN Notices* 40(10): 519–538. DOI: 10.1145/1103845.1094852.

Dan M, Rosswog S, Brüggen M, et al. (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. Available at: http://adsabs.harvard.edu/abs/2014MNRAS.438 . . . 14D (accessed 25 January 2019).

Dan M, Rosswog S, Guillochon J, et al. (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. Available at: http://adsabs.harvard.edu/abs/2011ApJ . . . 737 . . . 89D (accessed 25 January 2019).

Dan M, Rosswog S, Guillochon J, et al. (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. Available at: http://adsabs.harvard.edu/abs/2012MNRAS.422.2417D (accessed 25 January 2019).

Dehnen W (2000) A very fast and momentum-conserving tree code. *Astrophysical Journal, Letters (ApJL)* 536(1): L39–L42. DOI: 10.1086/312724. Available at: http://adsabs.harvard.edu/abs/2000ApJ . . . 536%20L.39D (accessed 25 January 2019).

Dekate C, Anderson M, Brodowicz M, et al. (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. Available at: https://arxiv.org/abs/1109.5190 (accessed 25 January 2019).

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.

deSupinski BR, Olivier SL, Terboven C, et al. (2017) Editors: scaling OpenMP for exascale performance and portability. In: *13th International workshop on OpenMP*, IWOMP 2017, Stony Brook, NY, USA, 20–22 September 2017. SpingerLink: Lecture Notes in Computer Science.

Doerfer D, Deslippe J, Williams S, et al. (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*. Available at: https://crd.lbl.gov/assets/Uploads/ixpug16-roofline.pdf (accessed 25 January 2019).

Dongarra J, London K, Moore S, et al. (2001) Using PAPI for hardware performance monitoring on Linux systems. In: *Proceedings of the international conference on Linux clusters: the HPC revolution*. Available at: www.netlib.org/utk/people/JackDongarra/PAPERS/papi-linux.pdf (accessed 25 January 2019).

D'Souza MCR, Motl PM, Tohline JE, et al. (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. Available at: http://adsabs.harvard.edu/abs/2006ApJ . . . 643.381D (accessed 25 January 2019).

Eschweiler D, Wagner M, Geimer M, et al. (2012) Open trace format 2: the next generation of scalable trace formats and support libraries. *Advances in Parallel Computing* 22: 481–490. DOI: 10.3233/978-1-61499-041-3-481. Available at: https://goo.gl/LVoPi5 (accessed 25 January 2019).

GNU (2017) GNU Compiler Collection 6.3.0 source distributions. Available at: https://ftp.gnu.org/gnu/gcc/gcc-6.3.0/. Available under the GNU General Public License version 3 (accessed 25 January 2019).

Guillochon J, Dan M, Ramirez-Ruiz E, et al. (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. Available at: http://adsabs.harvard.edu/abs/2010ApJ . . . 709%20L.64G (accessed 25 January 2019).

He Y, Cook B, Deslippe J, 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, et al. (2016) Closing the performance gap with modern C++. In: *Proceedings of the international conference on high performance computing workshops (ISC Workshops)*, Workshop on Exascale Multi/Many Core Computing Systems (EMuCoCoS). ISBN 978-3-319-46079-6, pp. 18–31. DOI: 10.1007/978-3-319-46079-6 2. Available at: https://stellar.cct.lsu.edu/pubs/closing_perf_gap_isc_2016.pdf (accessed 25 January 2019).

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. Available at: https://stellar.cct.lsu.edu/pubs/isc2012.pdf (accessed 25 January 2019).

Heller T, Kaiser H, Schäfer A, et al. (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. Available at: https://stellar.cct.lsu.edu/pubs/scala13.pdf (accessed 25 January 2019).

Hoberock J, Garland M, Kohlhoff C, et al. (2017) P0443R2: A unified executors proposal for C++. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings*. Available at: https://wg21.link/P0443R2 (accessed 25 January 2019).

Huck K, Porterfield A, Chaimov N, et al. (2015) An autonomic performance environment for exascale. *Supercomputing Frontiers and Innovations* 2(3): 49–66. DOI: 10.14529/jsfi150305.

Intel (2017a) Intel cilk plus. Available at: https://software.intel.com/en-us/intel-cilk-plus (accessed 25 January 2019).

Intel (2017b) Intel SPMD Program Compiler (ISPC). Available at: https://ispc.github.io/ (accessed 25 January 2019).

Intel (2017c) Intel Xeon Phi Processor 7250 (16 GB, 1.40 GHz, 68 core) Specifications. Available at: https://ark.intel.com/products/94035/Intel-Xeon-Phi-Processor-7250-16GB-1_40-GHz-68-core (accessed 25 January 2019).

Jemalloc (2017) Jemalloc GitHub repository, 4.5.0 tag. Available at: https://github.com/jemalloc/jemalloc/tree/4.5.0. Available under the 2-Clause BSD License (accessed 25 January 2019).

Kadam K, Clayton GC, Motl PM, et al. (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. Available at: http://adsabs.harvard.edu/abs/2017AAS . . . 2294%203314%20K (accessed 25 January 2019).

Kadam K, Motl PM, Frank J, et al. (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. Available at: http://adsabs.harvard.edu/abs/2016MNRAS.462.2237%20K (accessed 25 January 2019).

Kaiser H, Heller T, Bourgeois D, et al. (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)*. pp. 29–37. DOI: 10.1145/2832241.2832244. ISBN 978-1-4503-3996-4. Available at: https://stellar.cct.lsu.edu/pubs/executors_espm2_2015.pdf (accessed 25 January 2019).

Kaiser H, Heller T, Lelbach BA, et al. (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. Available at: https://stellar.cct.lsu.edu/pubs/pgas14.pdf (accessed 25 January 2019).

Katz MP, Zingale M, Calder AC, et al. (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. Available at: http://adsabs.harvard.edu/abs/2016ApJ . . . 819 . . . 94%20K (accessed 25 January 2019).

Kazakova A (2015) C/C++ facts we learned before going ahead with CLion. Technical report, Jetbrains. Available at: https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/ (accessed 25 January 2019).

Kevin Huck (2017) APEX performance monitoring framework GitHub repository, commit 58214cf. Available at: https://github.com/khuck/xpress-apex/commit/58214cfba5ce6ddb2682713329687c56625c580e (accessed 25 January 2019). Available under the Boost Software License 1.0 (a BSD-style open source license).

Knüpfer A, Brunst H, Doleschal J, et al. (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*, pp. 139–155. DOI: 10.1007/978-3-540-68564-7 9. ISBN 978-3-540-68561-6 Available at: https://link.springer.com/chapter/10.1007/978-3-540-68564-7_9#citeas.

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. Available at: http://publikationen.ub.uni-frankfurt.de/frontdoor/index/index/docId/38415 (accessed 25 January 2019).

Kretz M (2015b) N4395: SIMD Types: ABI Considerations. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings*. Available at: https://wg21.link/N4395 (accessed 25 January 2019).

Kretz M (2015c) N4454: SIMD Types Example: Matrix Multiplication. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings*. Available at: https://wg21.link/N4454 (accessed 25 January 2019).

Kretz M (2016) P0350R0: Integrating datapar with Parallel Algorithms and Executors. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings*. Available at: https://wg21.link/P0350R0 (accessed 25 January 2019).

Kretz M (2017) P0214R3: Data-Parallel Vector Types & Operations. *ISO/IEC JTC1/SC22/WG21 (the C++ Standards Committee) Mailings*. Available at: https://wg21.link/P0214R3 (accessed 25 January 2019).

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.

Kumar R, Tullsen DM, Ranganathan P, et al. (2004) Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In: *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pp. 64–75. DOI: 10.1109/ISCA.2004.1310764. ISBN 0-7695-2143-6. Available at: https://ieeexplore.ieee.org/document/1310764 (accessed 25 January 2019).

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.

Lelbach BA, Byerly ZD, Marcello DC, et al. (2013) Octopus: A scalable AMR toolkit for astrophysics. In: *Scientific Computing Around Louisiana (SCALA)*. Available at: http://stellar.cct.lsu.edu/pubs/SCALA2013_lelbach.pdf (accessed 25 January 2019).

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. Available at: http://adsabs.harvard.edu/abs/2001PhRvL.86.1152%20L (accessed 25 January 2019).

Linux Kernel Organization I (2017) Linux power capping framework. Available at: https://www.kernel.org/doc/Documenta

tion/power/powercap/powercap.txt (accessed 25 January 2019).

Marcello DC (2017) A very fast and angular momentum conserving tree code. *The Astronomical Journal* 154(3): 92.

Marcello DC, Kadam K, Clayton GC, et al. (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. Proceedings of Science*. Available at: http://apcs2016.iaps.inaf.it (accessed 25 January 2019).

Martin SJ and Kappel M (2014) Cray XC30 power monitoring and management. In: *Proceedings of the Cray user group conference*. Available at: https://cug.org/proceedings/cug2014_proceedings/includes/files/pap130.pdf (accessed 25 January 2019).

Menon H, Wesolowski L, Zheng G, et al. (2015) Adaptive techniques for clustered N-body cosmological simulations. *Computational Astrophysics and Cosmology* 2, art. id 1. DOI: 10.1186/s40668-015-0007-9. Available at: http://adsabs.harvard.edu/abs/2015ComAC . . . 2 . . . .1%20M (accessed 25 January 2019).

Montiel EJ, Clayton GC, Marcello DC, et al. (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. Available at: http://adsabs.harvard.edu/abs/2015AJ . . . .150 . . . 14%20M (accessed 25 January 2019).

Motl PM, Frank J, Staff J, et al. (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. Available at: http://adsabs.harvard.edu/abs/2017ApJS.229 . . . 27%20M (accessed 25 January 2019).

Motl PM, Frank J, Tohline JE, et al. (2007) The stability of double white dwarf binaries undergoing direct-impact accretion. *Astrophysical Journal (ApJ)* 670(2): 1314–1325. DOI: 10.1086/522076. Available at: http://adsabs.harvard.edu/abs/2007ApJ . . . 670.1314%20M (accessed 25 January 2019).

NERSC (2017a) Cray MPICH 7.4.4 documentation for NERSC Cori. Available at: https://www.nersc.gov/users/computational-systems/cori/programming/compiling-codes-on-cori/ (accessed 25 January 2019).

NERSC (2017b) National Energy Research Scientific Computing Center (NERSC) Cori System Details. Available at: http://www.nersc.gov/users/computational-systems/cori/configuration/ (accessed 25 January 2019).

Open MPI (2017) hwloc 1.11.6 source and binary distributions. Available at: https://www.open-mpi.org/software/hwloc/v1.11/ (accessed 25 January 2019). Available under the 3-Clause BSD License.

Ott CD, Ou S, Tohline JE, et al. (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. Available at: http://adsabs.harvard.edu/abs/2005ApJ . . . 625%20L.119O (accessed 25 January 2019).

Pakmor R, Hachinger S, Röpke FK, et al. (2011) Violent mergers of nearly equal-mass white dwarf as progenitors of subluminous type 1a supernovae. *Astronomy and Astrophysics* 528, art. id A117. DOI: 10.1051/0004-6361/201015653. Available at: http://adsabs.harvard.edu/abs/2011A&A . . . 528A.117P (accessed 25 January 2019).

Raskin C, Scannapieco E, Fryer C, et al. (2012) Remnants of binary white dwarf mergers. *Astrophysical Journal (ApJ)* 746(1, art. id 62). DOI: 10.1088/0004-637X/746/1/62. Available at: http://adsabs.harvard.edu/abs/2012ApJ . . . 746 . . . 62%20R (accessed 25 January 2019).

Sankrit R and Blair W (2004) X-ray, Optical & Infrared Composite (CXO/HST/SST) of Kepler's Supernova Remnant. Technical report, NASA/ESA/JHU. Available at: http://chandra.harvard.edu/photo/printgallery/2004 (accessed 25 January 2019).

Schaller M, Gonnet P, Chalk ABG, et al. (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. Available at: https://arxiv.org/abs/1606.02738 (accessed 25 January 2019).

Schwab J, Shen KJ, Quataert E, et al. (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. Available at: http://adsabs.harvard.edu/abs/2012MNRAS.427.190%20S (accessed 25 January 2019).

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.

Sodani A (2015) Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. In: *Hot Chips Symposium*. Available at: https://goo.gl/a6haUm (accessed 25 January 2019).

Staff JE, Menon A, Herwig F, et al. (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. Available at: http://adsabs.harvard.edu/abs/2012ApJ . . . 757 . . . 76%20S (accessed 25 January 2019).

STE||AR Group (2017a) HPX GitHub repository. Available at: https://github.com.STEllAR-GROUP/hpx (accessed 25 January 2019). Available under the Boost Software License 1.0 (a BSD-style open source license/.

STE||AR Group (2017b) HPX GitHub repository, commit 19bd11a. Available at: https://github.com/STEllAR-GROUP/hpx/commit/19bd11a521f878580316f7f4c7754298b7b45563 (accessed 25 January 2019). Available under the Boost Software License 1.0 (a BSD-style open source license).

STE||AR Group (2017c) Octopus AMR Framework GitHub repository. Available at: https://github.com/STEllAR-GROUP/octopus (accessed 25 January 2019). Available under the Boost Software License 1.0 (a BSD-style open source license).

STE||AR Group (2017d) OctoTiger AMR Framework GitHub repository. Available at: https://github.com/STEllAR-GROUP/octotiger (accessed 25 January 2019). 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. Available at: https://github.com/STEllAR-GROUP/octotiger/commit/0b6cd60d0405be700f191f03e2a011f7503b7af1 (accessed 25 January 2019). 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)*, Austin, USA, 24–25 January 2011, pp. 175–189. Berlin, Heidelberg: Springer-Verlag. DOI: 10.1007/978-3-642-18378-2 15. ISBN 978-3-642-18377-5.

USDOE (2012) *X-Stack: Programming challenges, runtime systems, and tools (DoE-FOA-0000619)*. Technical report, US Department of Energy Office of Science. Available at: https://science.energy.gov/∼/media/grants/pdf/foas/2012/SC_FOA_0000619.pdf (accessed 25 January 2019).

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. Available at: https://ieeexplore.ieee.org/abstract/document/4536359

XPRESS APEX (2017) APEX performance monitoring framework GitHub repository. Available at: https://github.com/khuck/xpress-apex (accessed 25 January 2019). Available under the Boost Software License 1.0 (a BSD-style open source license).

Zingale M, Almgren AS, Bell JB, et al. (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. Available at: http://adsabs.harvard.edu/abs/2009ApJ . . . 704.196Z (accessed 25 January 2019).

Željko I, Axelrod TS, Brandt WN, et al. (2008) Large synoptic survey telescope: From science drivers to reference design. *Serbian Astronomical Journal* (176): 1–13. DOI: 10.2298/SAJ0876001I. Available at: http://adsabs.harvard.edu/abs/2008SerAJ.176 . . . .1I (accessed 25 January 2019).

Author biographies

Thomas Heller is a researcher at the Friedrich-Alexander University in Erlangen. He works at the Computer Science Chair for Computer Architecture where he is pursuing his PhD. His research deals with mapping abstract formulations of algorithms developed with the help of a C++ EDSL onto arbitrary heterogeneous multi- and many-core architectures while achieving optimal performance. He received the NERSC Award for Innovative Use of High Performance Computing for his contributions to the HPX project by utilizing the full CORI Supercomputer with the OctoTiger application.

Bryce Adelstein Lelbach has spent nearly a decade developing libraries in C++. He is the chair of the C++ committee's Library Evolution Incubator study group. He is passionate about C++ community development. He is an organizer for the C++Now and CppCon conferences as well as the Bay Area C++ user group. He works at NVIDIA, where he is the team lead for CUDA Thrust, the CUDA C++ core library. He is one of the initial developers of the HPX parallel runtime system. He also helped start the LLVMLinux initiative and has occasionally contributed to the Boost C++ libraries. On the C++ committee, he has personally worked on the C++17 parallel algorithms, executors, futures, senders/receivers, and multidimensional arrays.

Kevin A Huck is a research faculty and computer scientist at the University of Oregon. He received a BS degree in computer science (1995) from the University of Cincinnati and was awarded his master's (2004) and PhD (2009) degrees in computer and information science from the University of Oregon. Previously, he has worked in various private industry efforts and as a postdoc at the Barcelona Supercomputing Center (2009–2011). He works primarily in the area of large-scale parallel performance measurement, analysis, and visualization. He is the creator and primary developer of APEX, PerfExplorer, and TAUdb.

John Biddiscombe holds a BEng in electronic engineering (1989) and a PhD in software enginieering (2017) from Warwick University. He worked at the Rutherford Appleton Laboratory on Digital Signal Processing methods for altimetry and remote-sensing radar, radio propagation around three-dimensional (3-D) objects, and visualization of 3-D data. He has worked at the Swiss National Supercomputing Centre as a visualization scientist and computational scientist since 2004.

Patricia Grubel is a postdoctoral research associate in computer, computational, and statistical sciences at Los Alamos National Laboratory. She has a PhD in electrical and computer engineering from New Mexico State University. She is involved in profiling scientific applications and task-based runtime systems on new architectures and implementing systems for scientific workflows on high-performance computing and cloud platforms.

Alice E Koniges is currently a computer scientist for the University of Hawaii at the Maui High Performance Computing Center. She is skilled in a variety of high-performance computing (HPC) fields including machine learning, adaptive mesh refinement, linear solvers, and physics applications. She was previously the Berkeley Lab Principal Investigator on several HPC initiatives including

the Exascale Programming Environment and System Software project and a modeling effort for Extreme Ultraviolet Chip Fabrication. She received her PhD in mathematical astrophysics from Princeton University where she was the first woman to graduate in the Program in Applied and Computational Mathematics. She also spent a number of years at the Lawrence Livermore National Laboratory where she served as Head of Institutional Computing and led several industrial Cooperative Research and Development Agreements with Industry to advance HPC applications. She has published over 100 technical papers, one book, and has approximately 1500 citations on her work.

Matthias Kretz is a senior software engineer in the Department of Scientific Computing at the GSI Helmholtz Centre for Heavy Ion Research. Prior to his time at GSI, he received his PhD degree in computer science from Goethe University Frankfurt as well as a Diploma in physics from Heidelberg University. His research interests lie in high-performance computing and API design, with a special focus on developing scalable, intuitive, and efficient abstractions for data parallelism, enabling better utilization of available computing resources.

Dominic Marcello is a research scientist at the Center for Computation and Technology at Louisiana State University. He received his PhD degree in physics from Louisiana State University. His research focuses on modeling stellar mergers and other stellar events using high-performance computing.

David Pfander is a PhD student at the University of Stuttgart. He is interested in performance engineering and the design of algorithms for heterogeneous architectures, particularly numerical algorithms. His research focuses on auto-tuning and how to use it to achieve performance portability.

Adrian Serio is a scientific program coordinator at Louisiana State University. Beginning in 2013, he has been involved in the development of HPX and other associated research projects at the university.

Juhan Frank is a professor in the Department of Physics and Astronomy at Louisiana State University who graduated in physics at the University of Buenos Aires, Argentina, and earned a PhD degree in astrophysics from Cambridge, England. His research interests include theory of mass transfer and merger phenomena in binary stars and binary evolution.

Geoffrey C Clayton is the Ball Family Distinguished Professor of Physics and Astronomy at Louisiana State University. His research interests include hydrodynamics simulations of stellar mergers involving white-dwarf binaries.

Dirk Pflüger is a professor at the Institute for Parallel and Distributed Systems, University of Stuttgart. He holds a PhD degree in computer science from the Technical University of Munich and master's degrees in computer science from the University of Stuttgart and in information technology from the University of Sydney. He heads the project EXAHD of the German priority program Software for Exascale Computing (SPPEXA). His research interests include performance optimization on heterogeneous hardware and hierarchical numerical algorithms and load balancing for HPC.

David Eder has a PhD degree in astrophysics from Princeton University. He is a recognized leader in high-performance computing applications giving tutorials at international venues such as the IEEE SC series. He received an Alexander von Humboldt Award to expand his research on X-ray lasers at the Max Planck Institute for Quantum Optics in Garching, Germany. As Group Leader for National Ignition Facility Modeling at Lawrence Livermore National Laboratory, he received numerous program awards including one for developing disposable debris shields. He has a strong background in parallel programming models including HPX, MPI, OpenMP, OpenCl, and others. He has over 100 publications and approximately 4000 citations. He is currently at University of Hawaii–Maui High Performance Computing Center with a focus on quantum computing.

Hartmut Kaiser is an adjunct professor of computer science at Louisiana State University. At the same time, he holds the position of a senior scientist at the Center for Computation and Technology (LSU). He received his doctorate from the Technical University of Chemnitz (Germany) in 1988. He is currently leading the STE||AR Group at LSU, a research group consisting of researchers and students interested in improving performance and scalability of high-performance computing applications. There he is working on the practical design and implementation of HPX—the standard library for parallelism and concurrency and its use in various high-performance applications. He is the author of several C++ libraries. He has contributed to Boost libraries that are in use by thousands of developers worldwide. He is also a voting member of ISO C++ Standards Committee.