5-1-2021

Performance Measurements within Asynchronous Task-Based Runtime Systems: A Double White Dwarf Merger as an Application

Patrick Diehl
Louisiana State University

Dominic Marcello
Louisiana State University

Parsa Amini
Louisiana State University

Hartmut Kaiser
Louisiana State University

Sagiv Shiber
Louisiana State University

See next page for additional authors

Follow this and additional works at: https://digitalcommons.lsu.edu/physics_astronomy_pubs

Recommended Citation

Authors

Patrick Diehl, Dominic Marcello, Parsa Amini, Hartmut Kaiser, Sagiv Shiber, Geoffrey C. Clayton, Juhan Frank, Gregor Dais, Dirk Pfluger, David Eder, Alice Koniges, and Kevin Huck

Performance Measurements within Asynchronous Task-based Runtime Systems: A Double White Dwarf Merger as an Application

Patrick Diehl[iD]**, Dominic Marcello, Parsa Amini, Hartmut Kaiser**[iD]
Center of Computation and Technology
Louisiana State University
Baton Rouge, LA, U.S.A.
{pdiehl,hkaiser,parsa}@cct.lsu.edu | dmarrce504@gmail.com

Sagiv Shiber[iD]**, Geoffrey C. Clayton**[iD]**, Juhan Frank**
Department of Physics & Astronomy
Louisiana State University
Baton Rouge, LA, U.S.A.
frank@phys.lsu.edu | sshiber1@lsu.edu | gclayton@fenway.phys.lsu.edu

Gregor Daiß, Dirk Pflüger[iD]
IPVS
University of Stuttgart, Stuttgart, Germany
{Dirk.Pflueger,Gregor.Daiss}@ipvs.uni-stuttgart.de

David Eder, Alice Koniges
Maui High Performance Computing Center
University of Hawaii, Maui, HI, U.S.A
{dceder,koniges}@hawaii.edu

Kevin Huck[iD]
OACISS
University of Oregon, Eugene, OR, U.S.A.
khuck@cs.uoregon.edu

June 10, 2021

Abstract

Analyzing performance within asynchronous many-task-based runtime systems is challenging because millions of tasks are launched concurrently. Especially for long-term runs the amount of data collected becomes overwhelming. We study HPX and its performance-counter framework and Autonomic Performance Environment for Exascale to collect performance data and energy consumption. We added HPX application-specific performance counters to the Octo-Tiger full 3D adaptive multigrid code astrophysics application. This enables the combined visualization of physical and performance data to highlight bottlenecks with respect to different solvers. We examine the overhead introduced by these measurements, which is around 1%, with respect to the overall application runtime. We perform a resolution study for four different levels of refinement and analyze the application's performance with respect to adaptive grid refinement. The measurements' overheads are small, enabling the combined use of performance data and physical properties with the goal of improving the code's performance. All runs were obtained on NERSC's Cori, Louisiana Optical Network Infrastructure's QueenBee2, and Indiana University's Big Red 3.

*K*eywords HPX · APEX · PAPI · double white dwarf merger · convergence test

1 Introduction

Over the last decade, the development of asynchronous task-based runtime systems has offered users an alternative to MPI. These systems provide them abstractions to distribute their applications on large systems. However, while these runtime systems can handle millions of concurrent tasks over thousands of compute nodes, collecting information about application performance and framework overhead is a challenge, in particular for massively parallel large-scale simulations.

In this work, we demonstrate how we can gather performance data for massively parallel production runs on NERSC's Cori, Louisiana Optical Network Infrastructure's QueenBee2, and Indiana University's Big Red 3.

We use the asynchronous task-based runtime system HPX and its performance-counter framework, and the extension Automatic Performance for Exascale (APEX) to collect the data. Furthermore, we discuss the overheads encountered and physics results obtained for a double white dwarf merger simulation.

As an application, we choose Octo-Tiger. Octo-Tiger is a code for the study of self-gravitating astrophysical fluids using Newtonian gravity. It is particularly suited for stellar mergers, and we use it to simulate a merger in this paper. Octo-Tiger is based entirely upon HPX, which provides the performance counters to profile our application in a massively parallel setting. Of the many types of stellar interactions, the merger event is the most compelling. The merging process is violent, and can usually be seen in many wavelength regimes as a bright outburst, observable at large distances. Some classes of stars and outbursts, for example type Ia supernovae, derive almost certainly from mergers; for others, such as magnetic white dwarfs, mergers are suspected. None of these phenomena is well understood. For the simulations in this paper, we investigate a stellar merger that may form an R Coronae Borealis (RCB) star.

Previously, it has been shown that Octo-Tiger can scale up to thousands of compute nodes [1]. Further, the performance of individual kernels has been showcased [1], and Octo-Tiger has been used to measure the overhead of certain HPX components before [2]. Octo-Tiger is already used for production runs, making it an ideal candidate for performance and overhead measurements in a real-world scenario.

Here, we demonstrate how to collect and visualize both physics and performance data, including energy consumption. Allowing us to identify bottlenecks and to optimize on the level of algorithms and on a lower code level.

In this paper we perform a resolution study using a full-scale white dwarf merger runs with Octo-Tiger. This gives us the insight for the integration of additional physical properties and refined simulations for better production runs.

From the computational perspective, we collect a multitude of performance measurements about APEX, HPX, and Octo-Tiger itself during the full-scale resolution study and several complementary short runs. This enables us to gain insight into the overheads of the utilized frameworks. These measurements include the overhead of taking the APEX measurements, the amount of energy required for the simulation at different scales, the HPX idle rate, and the HPX AGAS (Active Global Address Space). Some of those measurements we collect during short runs since Octo-Tiger must be run with multiple configurations to gather the data. Other measurements, like the ones for the AGAS, are collected during the actual resolution study. While there had been similar measurements of the AGAS in Amini et. al. [2], these were only for short simulation runs. We extend their results and obtain measurements for long, real-world production runs as well. In contrast to the short runs, long runs exhibit highly dynamic behavior and require frequent adaptive mesh refinement, which poses extra challenges for scalability and efficient parallelization.

Here we present both new simulation results of Octo-Tiger and the overhead measurements of the utilized frameworks.

2 Related Work

2.1 R Coronae Borealis (RCB) merger

When two stars merge into one, they form objects with strange characteristics. The R Coronae Borealis are rare, carbon stars with little hydrogen and dust-induced variability [3]. They are almost impossible to explain through single-star evolution. The observational evidence points to a stellar merger of two white dwarfs (WDs). The RCB stars are close cousins of Type Ia SNe.

WD-WD modeling is dominated by the question of where and when the primary star detonates. But the mergers that produce RCB stars are low mass and the star does not explode. Previous simulations have been important to studies of RCB stars [4].

2.2 Performance measurements and their visualization

Typically, performance data is visualized and represented in the physical and/or logical context of the hardware and/or software resources used in the simulation. Data is organized by processes and threads, and visualized with respect to nodes, network topology and CPU architectures. Huck et al. [5] integrated performance data with simulation output in order to project the performance data into the scientific domain. Using the Scalable Observation System (SOS), performance data was aggregated and queries were executed to extract performance data and generate Visualization Toolkit (VTK) output files [6]. Using a similar approach, fusion simulation performance data was aggregated and exported to VTK files [7].

3 Scientific application: unstable mass transfer in a double white dwarfs system leading to a merger

Approximately half of the double white dwarf systems in our galaxy will merge within a Hubble time. These systems slowly lose their orbital energy by emitting gravitational waves. This drives the two stars closer together, until they start to interact via mass transfer and ultimately merge. Observational evidence suggests the stellar merger scenario for the origin of RCB stars [3]. We simulate the interaction between two low mass white dwarfs that leads to merger.

3.1 Transfer of angular momentum

Mass transfer in systems with mass ratios above a certain threshold results in merger due to tidal disruption of the donor star. As angular momentum is transferred from the orbit to the spin of the accretor, the orbital angular momentum decreases. For certain equations of state, the donor increases in size as it loses mass, leading to runaway mass transfer. The donor star is ripped apart by tidal forces, its remnants falling onto the accretor.

According to previous simulations, the onset of mass transfer and the time the binary system experiences mass transfer before merging are highly sensitive to initial conditions. Motl et. al. [8] showed that the resolution of a simulation affects the initial mass transfer, which in turn determines the length of the mass transfer episode before the merger. The higher the resolution, the longer the duration before a merger. We expect to see the same behavior in our simulations.

The consistent simulation of mass transfer and its feedback on the binary dynamics is a serious challenge for every numerical method. For smoothed particle hydrodynamics (SPH) codes, this is because the transferred matter comes initially from the WD surface which is the poorest resolved region of the star. For finite volume codes, this is due to the resolution-dependent angular momentum conservation. Octo-Tiger is a good candidate in overcoming these challenges because it is a finite-volume code that conserves angular momentum to high precision.

3.2 The stellar model

Our initial model is generated using Octo-Tiger's Self Consistent Field (SCF) code module. The initial masses of the accretor and donor are $0.5M_\odot$ and $0.35M_\odot$ (mass ratio of 0.7), respectively, with an initial separation of $0.047R_\odot$ and an initial period of $111s$. The accretor is composed of a $0.455M_\odot$ core with an even mixture by mass of carbon and oxygen (CO), surrounded by a helium shell of mass $0.045M_\odot$. This helium shell has a role in reducing the presence of ^{16}O in the outer layers, where the nuclear reactions occur. The donor is composed of only helium.

The grid domain runs from $-6.12R_\odot$ to $+6.12R_\odot$ in each dimension. Table 1 shows the finest grid cell size of each maximum level of refinement. We approximate the stars as $n = 3/2$ polytropes and evolve the gas with an ideal gas equation of state.

4 Software framework

4.1 Octo-Tiger

Octo-Tiger is a finite volume, octree based, adaptive mesh refinement (AMR) code that solves the in-compressible Euler equations with self-gravity on a fully three-dimensional rotating Cartesian mesh. It uses a 3rd order method to solve the hydrodynamics and a fast multipole method (FMM) to solve the gravity. The task-based approach provided by HPX is ideally suited for an application such as Octo-Tiger for several reasons. Through futures, HPX provides the ability to set up tasks dependent on data availability. HPX schedules those tasks as the data become available. The global address space of HPX allows Octo-Tiger to easily set up its octree over the computational domain and allows for easy load balancing. Finally, HPX futures do not distinguish between node level and system level parallelism, removing the need to use OpenMP to parallize at the node level. We refer the reader to Marcello, Shiber, et. al. [9] for a more detailed description of Octo-Tiger.

4.2 HPX

The development of Octo-Tiger in ISO C++11 using HPX [10] is shown in by Daißet. al. [1]. 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 emerging HPC architectures. The HPX methodology described here includes the following components:

- An ISO C++ standard conforming API that enables wait-free asynchronous parallel programming, including futures, channels, and other primitives for asynchronous execution.
- An Active Global Address Space (AGAS) that supports load balancing via object migration and enables exposing a uniform API for local and remote execution.
- An active-message networking layer that enables running functions close to the objects they operate on. This also implicitly overlaps computation and communication.
- A work-stealing lightweight task scheduler that enables finer-grained parallelization and synchronization and automatic load balancing across local compute resources.
- APEX, an in-situ profiling and adaptive tuning framework.

Table 1: The four levels of refinements for resolutions utilized.

Levels of Refinement	Initial Sub-grids	Finest grid Cell Size (cm)	Finest Grid Cell Size ($R_\odot$)	Initial Leaf Nodes	AMR Boundaries	Memory (GB)	Time of Disruption	Diff from Next Level
10	1185	1.03×10^8	1.48×10^{-3}	1038	1503	15.6	6.7	6.7
11	4974	5.18×10^7	7.40×10^{-3}	4352	4158	66.6	13.4	2.1
12	8781	2.59×10^7	3.70×10^{-4}	5969	7683	117.7	15.5	1.3
13	25 353	1.26×10^7	1.85×10^{-4}	22 184	11 800	339.8	16.8	–

HPX exposes an asynchronous, standards conforming programming model enabling Futurization, with which developers can express dataflow execution trees that generate billions of HPX tasks that are scheduled to execute only when their dependencies are satisfied. Also, Futurization enables automatic parallelization and load-balancing to emerge. This provides a unified approach to intra- and inter-node parallelism based on proven generic algorithms and data structures available in today's ISO C++ Standard. The programming model is intuitive and enables performance portability across a broad spectrum of diverse HPC hardware.

Additionally, HPX provides a performance counter and adaptive tuning framework that allows users to access performance data, such as core utilization, task overheads, and network throughput; these diagnostic tools were instrumental in scaling Octo-Tiger to the full machine.

4.2.1 AGAS

AGAS, short for Active Global Address Space, is the part of HPX runtime system that provides the means to access objects that live on different compute nodes from other compute nodes. Through this system, accessing a global object is transparent to the code and is done through the same API, whether the object lives on the same node as the code that attempts to access it or not. AGAS is able to hide some I/O and object access latencies through the use of active messages when accessing remote objects.

Since AGAS is an active component of the HPX runtime system and does its work during application execution, it takes a portion of the execution time. While AGAS overheads have been shown to be insignificant in short tests on Piz Daint at CSCS [2], this study provides an opportunity to study its overheads in a long-running application with dynamic adaptivity and observe whether it affects application execution times in an interesting fashion.

4.2.2 Performance counters

HPX performance counters are first class objects, each with a global address mapped to a unique symbolic name, useful for introspection at execution time by the application or the runtime system. The performance counters are used to provide information about how the runtime system or the application is performing. Counter data can help determine system bottlenecks and fine-tune system and application performance.

Listing 1: Example for an application-specific HPX performance counter and its registration

```
 1  // Function to return the application-
         specific
 2  // performance counter
 3  uint64_t leaf_count(bool reset) {
 4   std::lock_guard<hpx::mutex> lock(
         leaf_count_mtx);
 5   if (reset) {
 6   cumulative_leaf_count = 0;
 7   }
 8   return cumulative_leaf_count;
 9  }
10  // Register the performance counter
11  void register_counter() {
12   using namespace hpx::
         performance_counters;
13   install_counter_type(
14   "octotiger/subgrid_leaves",
15   &leaf_count,
16   "total number of subgrid leaves
         processed");
17  }
```

HPX exposes special API functions that allow one to create, manage, and read the counter data [11]. Any performance counters can be accessed remotely or locally using the same API. Moreover, since all counters are accessible through a uniform set of functions, any code consuming counter data can be utilized to access arbitrary system information with minimal effort. HPX also allows the extension of the set of performance counters to include application-specific information (see Listing 1 for an example of all code necessary to do so). An application needs to expose the counter data through a function (line 3) that has to be registered with HPX (line 13) with a unique name (line 14). The application-specific performance counters are then exposed through the same interfaces as the predefined counters and are readily available to APEX and other tools for intrinsic performance analysis.

4.3 APEX

APEX [12] (Autonomic Performance Environment for Exascale) is a performance measurement library for distributed, asynchronous multitasking systems such as HPX. It provides lightweight measurement and high concurrency. To support performance measurement in systems that employ user level threading, APEX uses a dependency chain

4

rather than the call stack to produce traces. APEX supports both synchronous and asynchronous introspection. The synchronous measurement by APEX uses a timer-based instrumentation and event listeners. The API includes events to start, stop, yield, or resume timers and to sample values in the application. In contrast, the asynchronous measurements periodically perform first- and third-person observations of the application, libraries, the hardware and operating system states such as load, energy consumption and resource utilization.

The *policy engine* within APEX provides an API to construct policies that can modify the behavior of the application, execute a function in the runtime or select important runtime and application parameters. Typically, the policies are used to apply optimization strategies but can be implemented as a feedback-and-control system. There are two ways to execute a policy: either explicitly triggered or asynchronously periodic. A triggered policy can be initiated by a specific event within the HPX runtime. In contrast, periodic policies use a periodic timer interrupt which is specified during the policy's registration. All policies are stored in a policy queue and executed in the order they are registered. Defined policies can search for a set of optimum parameters by minimizing a measurement value from APEX, such as the time of a measured region/task or by looking at any other introspection data gathered by APEX.

4.4 PAPI

Both APEX and HPX are integrated with PAPI (The Performance API). PAPI provides a unified, generic interface to CPU hardware performance counters across many hardware platforms and operating systems. PAPI provides portable, abstracted counter names to access the hardware counters without having to customize the measurement to every system available. Recently, PAPI has also added components to measure GPU hardware as well as overall system measurements such as network interfaces, file system performance, sensor data, and energy consumption.

5 Computer Science Results

This section contains the results of the computer science and the domain science aspects for the RCB merger described in Section 3. Table 1 lists the details of the four different level of refinements of the RCB scenario used for the resolution study and the investigation of performance.

5.1 Overhead measurements

This section studies the overhead of HPX's performance counters and APEX/PAPI. On NERSC's Cori system the sub-grids processed per second for the first two levels were obtained up to 1024 nodes. For one node up to 64 nodes four different scenarios were executed: *I* HPX + performance counters and APEX, *II* HPX + APEX, *III* HPX + Performance counters, and *IV* pure HPX. For up to
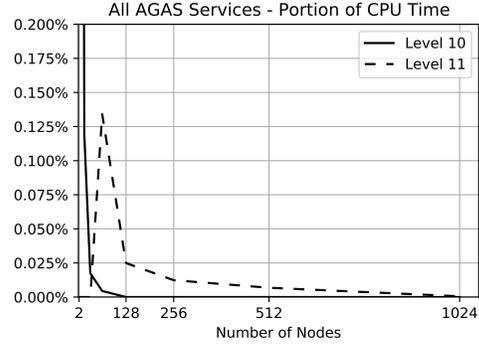

Figure 1: Portion of CPU time spent executing AGAS code. These runs were done on NERSC's Cori.

1024 nodes and all long-term runs the configuration *I* was used for producing the results in the following sections.

For all four configurations, the amount of processed sub-grids per second only differs by approximately 1%, which is acceptably within the expected variability of a typical shared system. In general adding HPX's performance counters and APEX does not affect the sub-grids processed per second and no huge effect on performance can be identified up to 64 nodes. Note that a run using the performance counters and APEX/PAPI is only needed once to identify potential bottle necks in the application by combining HPX specific and application-specific performance counters. We had to choose four small enough levels fitting into our awarded node hours for the resolution study presented in this paper. The scaling flattens out soon, since we had not enough work for the larger amount of nodes.

5.2 Performance counters

5.2.1 HPX counters

Fig 1 shows that the percentage of CPU time that is spent executing AGAS code is insignificant. This is the same behavior has been previously reported [2] for a different scenario on Piz Daint. In this study we conduct a broader study of AGAS overheads by measuring them on four long-term production runs from 45, 177.50, 227.83 hours up to 14 days on QueenBee2. The AGAS overhead (% in parenthesis) was (0.019) 10 , (0.12) 20, (0.17) 40, and (0.17) 64-128 nodes. As with the short-term runs, we see that the AGAS overheads are insignificant.

5.2.2 APEX counters

Our prospective metric to determine the optimal number of nodes is the sub-grids processed per second. However, another metric could be to optimize the energy consumption. Figure 2a shows the total energy consumption as the number of nodes increases and Figure 2b shows the processed sub-grids per kJ for a run on Cori, including initialization and shutdown. We could not perform the energy measurements for the long-term runs, since the Linux
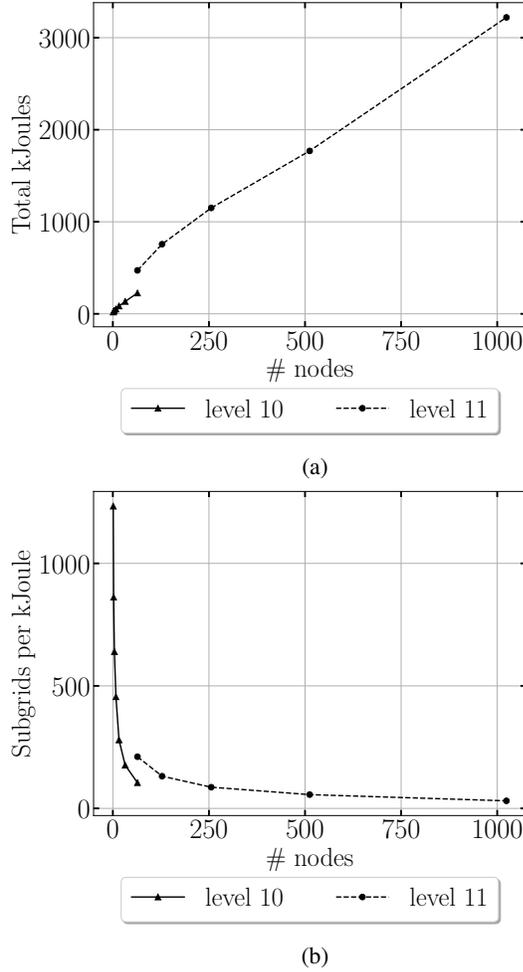
(a)

(b)

Figure 2: Total kJoules consumed for each run on Cori, and the number of sub-grids processed per kJoule for each run on Cori.
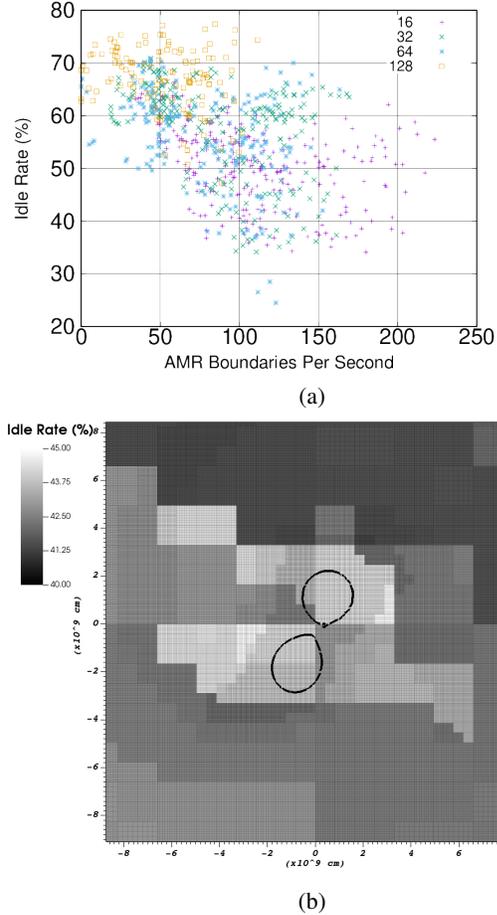

(a)

(b)

Figure 3: Idle rate (%) vs. AMR boundaries / s, sampled for each process every hour during part of the level 12 run (a) and the idle rates projected onto the computational domain (b). Higher idle rate indicates worker starvation due to an insufficient amount of work.

kernel on QueenBee2 was too outdated and this feature requires a kernel version ≥ 3.7.

5.2.3 Octo-Tiger counters

Octo-Tiger has three application-specific performance counters. They count the total numbers of sub-grids, leaf nodes, and AMR boundaries processed. HPX tracks the idle rate, defined as the amount of time an HPX locality has no work ready to schedule, through its own performance counter. We show in Figure 3a the idle rate versus the AMR boundaries processed per second for a short test run of the level 13 problem on Big Red 3 super computer. We ran the test on 16, 32, 64, and 128 nodes. As expected, the runs with higher node counts have a higher idle rate since the amount of sub-grids for each node decays from 3169 to 198. Despite the high idle rates, significant performance improvement in terms of total sub-grids per second is gained even moving from 64 (3493) to 128 (4834) nodes. Going from 16 to 128 nodes ($8\times$) gives a speedup of four

for the processed sub-grids for a small problem size. We had to find a trade-off for running four levels of refinement for scientific results and could not run the optimal problem size with respect to scaling/permanence in this paper.

We also notice a weak dependence between the AMR boundary boundaries per second and the idle rate. Although the number of sub-grids and leaf nodes is distributed evenly between localities, the number of AMR boundaries a given compute node must process varies between localities. Localities with more boundaries to process have lower idle rates due to their higher work load. These results suggest there may be an opportunity for more efficient load balancing by distributing the load based on AMR boundaries as well as sub-grid counts. In the future, another quantity to investigate with performance counters could be the ratio of local to remote AMR boundaries.

Octo-Tiger also includes the idle rates by locality as part of its grid output. This feature can be used to investigate whether different parts of the spatial domain have lower

or higher work loads. Figure 3b shows the idle rates for a zoomed in equatorial slice of the domain for the level 12 run. There is a single density contour, and the AMR grid structure is shown. Since the idle rate is the same for each locality, the plot also reveals locality boundaries. The range of idle rates shown is very narrow, indicating good load balancing for this particular slice.

6 Astrophysical Results

Figure 4 shows the evolution of three parameters of the binary's evolution. The separation is measured between the stars' centers of mass. There are epicyclic variations in the separation, while the overall trend is down. The stars continue to get closer until the donor is tidally disrupted. The donor mass loss rate is shown on a logarithmic scale, revealing how sensitive the mass transfer rate is to resolution. Lower resolution runs transfer mass from the donor at higher rates, resulting in quicker mergers. Initially, the stars are driven into closer contact by removing 1% of the angular momentum of each cell per orbit over a period of two orbits. This is done to mimic the driving that occurs due to gravitational radiation, just on a much shorter time scale.

One prominent effect of resolution is the time at which merger occurs. Because the time-scale for each run is different, this makes grid to grid comparisons difficult. We can, however, investigate global properties as resoluton increases. As shown in Table 1, the time of merger is one global property whose difference between successive resolutions grows smaller as resolution increases. The orbital separation, the orbital angular momentum, and the donor mass loss rates, in Figure 4, also exhibit this behavior.

7 Conclusion and Outlook

The resolution study for the production-level simulation demonstrates that global properties such as the time of merger and mass loss rate Octo-Tiger are resolution dependent, however, their successive differences with increasing resolution grows smaller. This is an important verification of the simulation results. By collecting performance data during these long-running simulations and some short runs, we were further able to determine the overheads of the frameworks utilized by Octo-Tiger, as well as the overhead of gathering all the data using APEX.

We have been able to show that the overhead of collecting performance data did affect the overall runtime performance of Octo-Tiger only slightly: The difference in performance between runs with and without APEX was only around 1%. The measured AGAS overhead during the short runs suggests that on more than 128 compute nodes, less than 0.025% of the total runtime is spent executing the AGAS code. During the resolution study, the measured AGAS overhead was around 0.17%. The discrepancy here is probably due to a different number of compute nodes and due to the increased scenario size and the dynamic

behavior, refining the grid data structures with evolving simulation time.

The performance of the HPX AGAS itself and the low runtime overhead of measuring overheads with HPX and APEX underpin the viability of using the asynchronous task-based framework HPX for real-world applications. For the first time, we have obtained convincing numbers for large, dynamic production-scale runs and scenarios. This extends the proof of applicability of our approach from large but short-running and thus static runs to large long-running dynamic simulations with frequent re-gridding and adaptive mesh refinement.

Furthermore, we have investigated the energy consumption of running Octo-Tiger. This will provide us with an extra metric for further optimization of Octo-Tiger. For large-scale simulations we have the responsibility to not only consider scaling and wall-clock time but to do science with less economical and environmental impact.

The I/O demands of Octo-Tiger at higher levels of refinement dominates the execution time.

8 ACKNOWLEDGMENT

8.1 Supplementary materials

Build scripts[1]; input files[2] and software version; and slurm files[3] are available to reproduce the results.

Copyright notice

References

[1] Gregor Daiß, Parsa Amini, John Biddiscombe, Patrick Diehl, Juhan Frank, Kevin Huck, Hartmut Kaiser, Dominic Marcello, David Pfander, and Dirk Pfüger. From Piz Daint

---

[1] https://github.com/diehlpk/PowerTiger
[2] https://doi.org/10.5281/zenodo.3751820
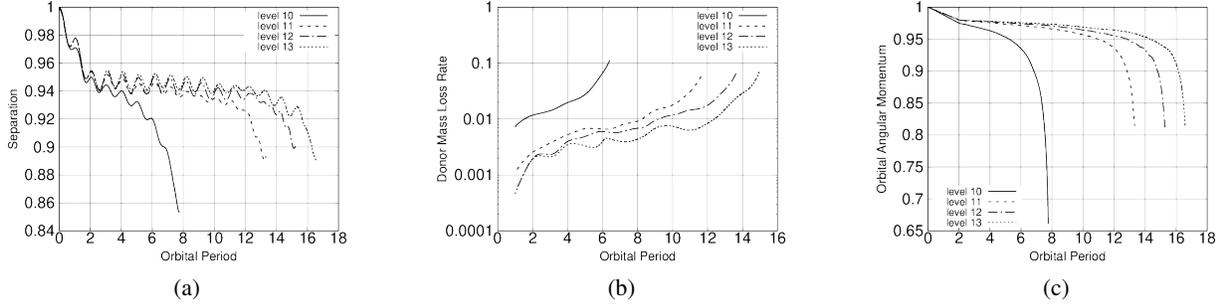[3] https://doi.org/10.5281/zenodo.3753539

Figure 4: The separation between the stars' centers of mass is depicted in (a), normalized to the initial separation. The donor mass loss rate is shown in (b), normalized to one donor mass per initial orbital period. The orbital angular momentum is shown in (c), normalized to the initial orbital angular momentum. The time coordinate for all three plots is shown in units of the initial orbital period.

to the stars: simulation of stellar mergers using high-level abstractions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–37. ACM, 2019.

[2] Parsa Amini and Harmut Kaiser. Assessing the performance impact of using an active global address space in hpx: A case for agas. In *2019 IEEE/ACM Third Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware*, pages 26–33. IEEE, 2019.

[3] Geoffrey C. Clayton. What are the r coronae borealis stars? *Journal of the American Association of Variable Star Observers (JAAVSO)*, 40(1):539, 2012.

[4] Amber Lauer, Emmanouil Chatzopoulos, Geoffrey C. Clayton, Juhan Frank, and Dominic C. Marcello. Evolving R Coronae Borealis stars with MESA. MNRAS, 488(1):438–450, 2019.

[5] Kevin A. Huck, Kristin Potter, Doug W. Jacobsen, Hank Childs, and Allen D. Malony. Linking performance data into scientific visualization tools. In *2014 First Workshop on Visual Performance Analysis*, pages 50–57. IEEE, 2014.

[6] Chad Wood, Matthew Larsen, Alfredo Gimenez, Kevin Huck, Cyrus Harrison, Todd Gamblin, and Allen Malony. Projecting performance data over simulation geometry using sosflow and alpine. In *Programming and Performance Visualization Tools*, pages 201–218. Springer, Cham, 2019.

[7] Jong Youl Choi, Choong-Seock Chang, Julien Dominski, Scott Klasky, Gabriele Merlo, Eric Suchyta, Mark Ainsworth, Bryce Allen, Franck Cappello, Michael Churchill, et al. Coupling exascale multiphysics applications: Methods and lessons learned. In *2018 IEEE 14th International Conference on e-Science*, pages 442–452, 2018.

[8] Patrick M. Motl, Juhan Frank, Jan Staff, Geoffrey C. Clayton, Christopher L. Fryer, Wesley Even, Steven Diehl, and Joel E. Tohline. A comparison of grid-based and sph binary mass-transfer and merger simulations. *ApJS*, 229(2):27, 2017.

[9] Dominic C Marcello, Sagiv Shiber, Orsola De Marco, Juhan Frank, Geoffrey C Clayton, Patrick M Motl, Patrick Diehl, and Hartmut Kaiser. Octo-tiger: A new, 3d hydrodynamic code for stellar mergers that uses hpx parallelisation. *arXiv preprint arXiv:2101.08226*, 2021.

[10] Hartmut Kaiser, Patrick Diehl, Adrian S. Lemoine, Bryce Adelstein Lelbach, Parsa Amini, Agustín Berge, John Biddiscombe, Steven R. Brandt, Nikunj Gupta, Thomas Heller, Kevin Huck, Zahra Khatami, Alireza Kheirkhahan, Auriane Reverdell, Shahrzad Shirzad, Mikael Simberg, Bibek Wagle, Weile Wei, and Tianyi Zhang. Hpx - the c++ standard library for parallelism and concurrency. *Journal of Open Source Software*, 5(53):2352, 2020.

[11] Patricia Grubel, Hartmut Kaiser, Kevin Huck, and Jeanine Cook. Using intrinsic performance counters to assess efficiency in task-based parallel applications. In *IPDPS Workshops*, pages 1692–1701. IEEE, 2016.

[12] Kevin A. Huck, Allan Porterfield, Nick Chaimov, Hartmut Kaiser, Allen D. Malony, Thomas Sterling, and Rob Fowler. An autonomic performance environment for exascale. *Supercomputing frontiers and innovations*, 2(3):49–66, 2015.