Performance Debugging and Tuning of Flash-X with Data Analysis Tools

Kevin Huck*, Xingfu Wu†, Anshu Dubey†, Antigoni Georgiadou‡, J. Austin Harris‡, Tom Klostermann†, Matthew Trappett*, Klaus Weide†

iD 0000-0001-7064-8417

Email: khuck@cs.uoregon.edu

* University of Oregon, †Argonne National Laboratory, ‡Oak Ridge National Laboratory

Performance Debugging and Tuning of Flash-X with Data Analysis Tools

Kevin Huck [*], Xingfu Wu[†], Anshu Dubey[†], Antigoni Georgiadou[‡], J. Austin Harris[‡], Tom Klostermann[†], Matthew Trappett[*], Klaus Weide[†]

khuck@cs.uoregon.edu

- University of Oregon, †Argonne National Laboratory, ‡Oak Ridge National Laboratory

Executive Summary

- While evaluating new partitioning library in Flash-X…
- Removing “not needed” communication code led to a slowdown in computation
- We developed a set of **Jupyterlab Python** scripts that utilize **Pandas** and **Plotly** to automate the generation of **distribution and correlation visualizations** for better understanding of performance behavior
- In this process, we **discovered and removed/mitigated** two additional performance limiting bottlenecks for performance tuning

UNIVERSITY OF OREGON

Introduction

- Multiphysics simulations and HPC platforms have **many degrees of freedom** leading to high complexity
- Optimization search space is huge, compilers make conservative assumptions (do no harm)
- Modifying code by hand – even changes as simple as removing a function call – can have **negative unintended** (and unexpected) **performance consequences**
- Pulling on the loose thread can be a rabbit hole…and/or lead to other insights (apologies for mixed metaphors)

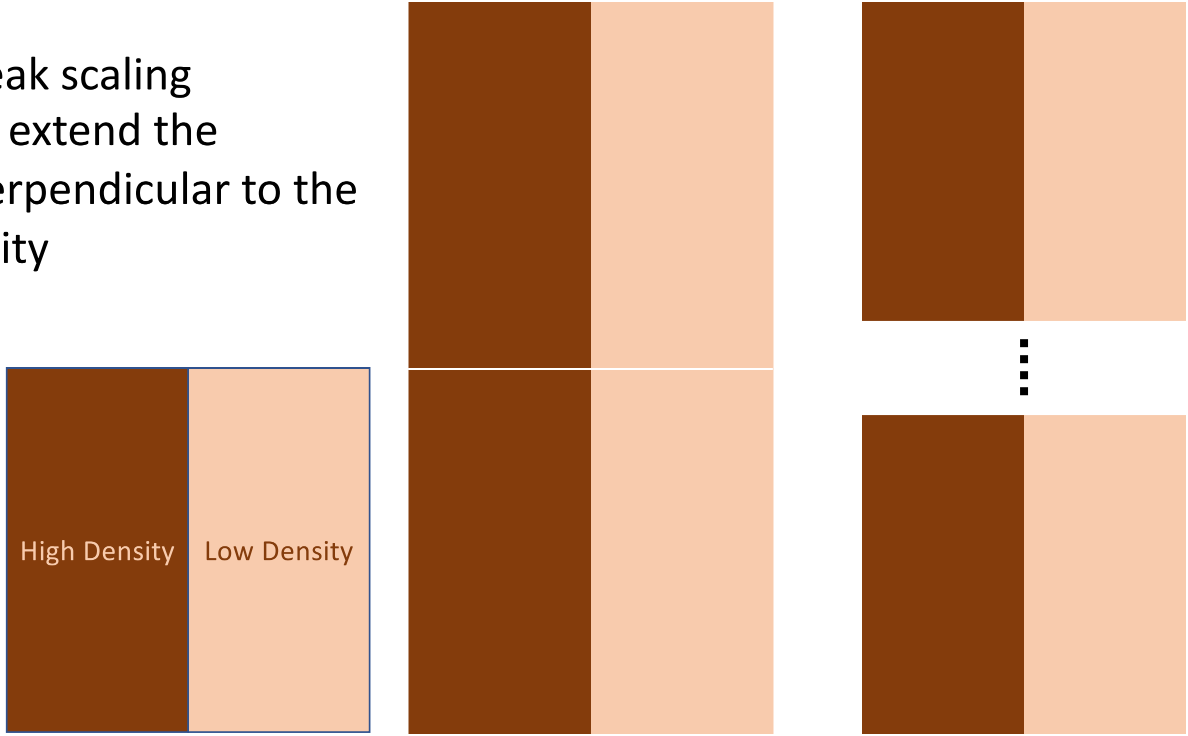
Flash-X Anomaly

- Flash-X: newer descendant of FLASH – a multiphysics, multicomponent code
- Fortran implementation
- Base discretization in Flash-X is Eulerian, with 3 flavors of management for the discretized mesh
  - Uniform
  - Two different AMR methods, Paramesh and (now) AMReX
- Integration of AMReX revealed **vestigial/superfluous communication routine** from Paramesh implementation
- Removing them reduced communication cost, but ***increased*** (unrelated?) **computation cost** by more than 20% in some cases

https://flash-x.org

Sod Shock Tube, Weak Scaling Setup

Perfect weak scaling scenario – extend the domain perpendicular to the discontinuity

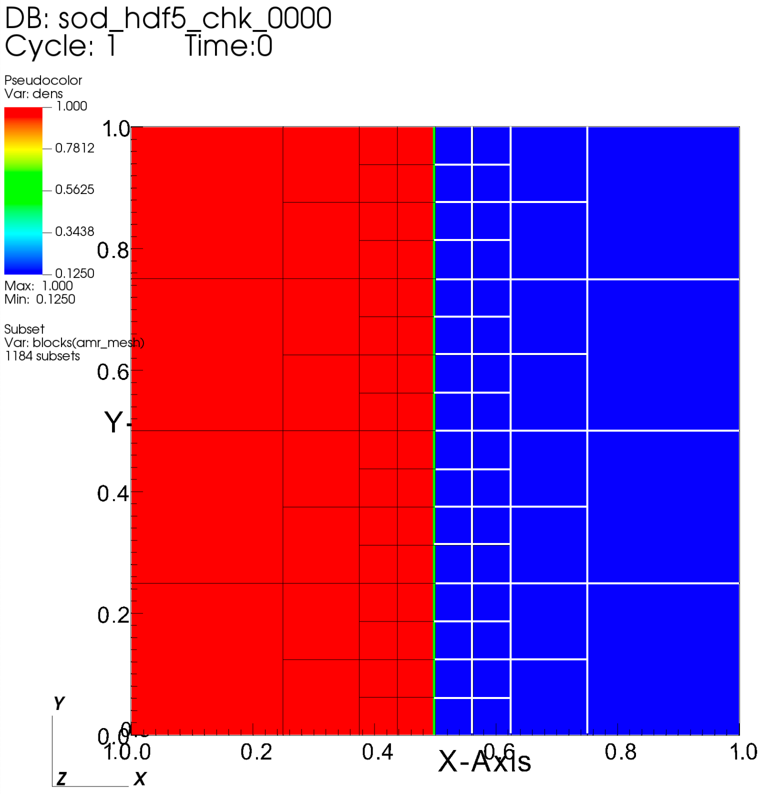


Base Sod setup

Changing setup for twice as many nodes (replicate once)

Changing setup for n times as many nodes (replicate n times)

Flash-X AMR Partitioning Overview



- Paramesh is only AMR support in FLASH, Flash-X added new support for AMReX
- All blocks are identical in #cells along each dimension, and all blocks are organized in an **octree**. (AMReX allows for irregular cells, but not needed in this case)
- Parallelism provided by MPI, load balancing provided by Morton ordering
- Original task: **integrate, evaluate AMReX in Flash-X and compare with Paramesh** (work is still ongoing)

Figure: A 2D slice of the initial 3D mesh for the Sod shock tube problem. Each of the outlined squares represents a block of 16^3 computational cells.

UNIVERSITY OF OREGON

Systems Evaluated

- Summit: IBM® POWER9™ at ORNL, 200 PFLOP/s
  - 4,608 IBM POWER System AC922 nodes
  - Two IBM POWER9 processors, 42 total compute cores and six NVIDIA Volta V100 accelerators, 512 GB of DDR4 per node
- Theta: Cray® XC40 at ANL, 12 PFLOP/s
  - 4,392 Cray XC40 nodes
  - One Intel Phi Knights Landing (KNL) 7230 with 64 compute cores, shared L2 cache of 32 MB (1 MB L2 cache shared by two cores), 16 GB of high-bandwidth in-package memory, 192 GB of DDR4 RAM per node
  - Cache mode used for all experiments

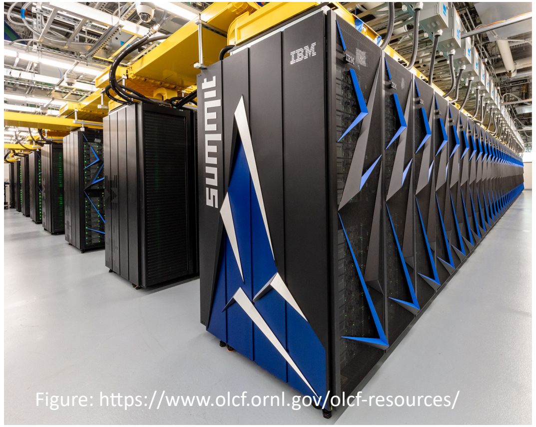


Figure: https://www.olcf.ornl.gov/olcf-resources/

Figure: https://www.alcf.anl.gov/alcf-resources/theta

UNIVERSITY OF OREGON

Performance Anomaly

- Paramesh grid routine: **`mpi_amr_boundary_block_info`**
  - Collects some information about the faces of blocks that coincide with the boundary of the simulation domain, and makes this global information available as a *heap-allocated* array on each rank
  - Called once each time that the AMR grid has changed
  - Allowed specialized handling of boundary conditions…but -
  - Result never used (not used in *current* version of code)
- To reduce global communication overhead and improve performance, it was removed
- Performance *got worse*…?

mpi_amr_boundary_block_info

- **Allocates scratch space** that is some multiple of the number of local blocks that have any of their neighboring blocks on the boundary
- **Collective operations** to share this information globally
- In total, six scratch arrays are allocated, some of which persist through the evolution step
- In brief: relevant portions of the routine are
  1. Allocations
  2. Global collective operations
  3. Memory is freed
- These allocations are sized by the number of blocks, therefore can have odd sizes – each rank can be different size

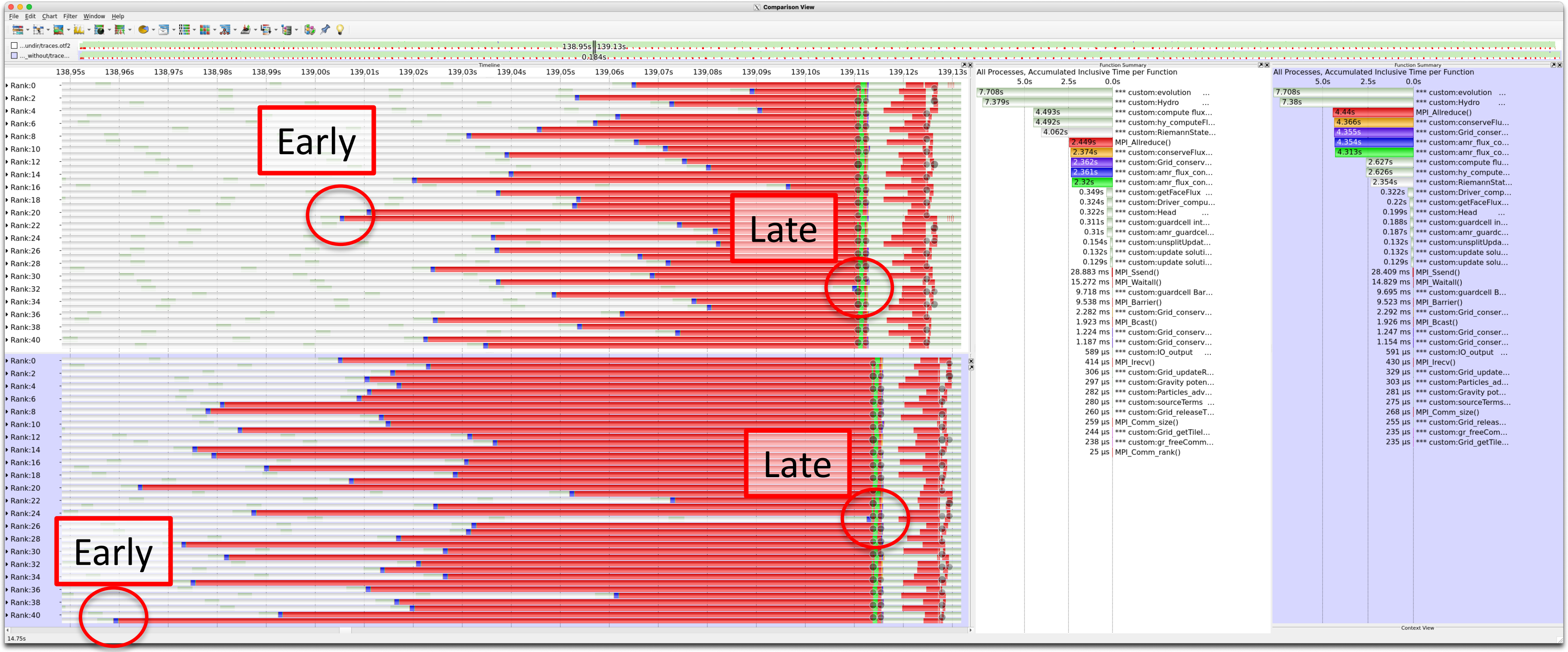
Result From Removing Call

- Performance anomaly first discovered on Theta
- Time spent in communication routines **decreased as expected**, *but*…
- Overall **evolution time (total simulation time) increased** non-trivially, in some instances by as much as 20% or more
- The degradation in performance was observed in the routine that computes Riemann states. This is a **completely local routine**, essentially an expensive stencil calculation, that ***has nothing whatsoever to do with communication***
- Behavior was reproducible not only on the same number of MPI ranks and therefore the same problem, but also **across the entire weak scaling study**
- On Summit the effect is more subtle, but exists
- Behavior **not** due to variability inherent in the platforms or because of workload differences at different times

Debugging Approach – TAU

- TAU Performance System – University of Oregon
- Tuning and Analysis Utilities (28+ year project)
- Integrated performance toolkit:
  - Multi-level performance instrumentation
  - Highly configurable
  - Widely ported performance profiling / tracing system
  - Portable (java, python) visualization / exploration / analysis tools
- Supports all major HPC programming models
  - MPI/SHMEM, OpenMP/ACC, CUDA, HIP, OneAPI, Kokkos...
- Flash-X already integrated with TAU, so logical choice

Trace comparison – TAU data in Vampir



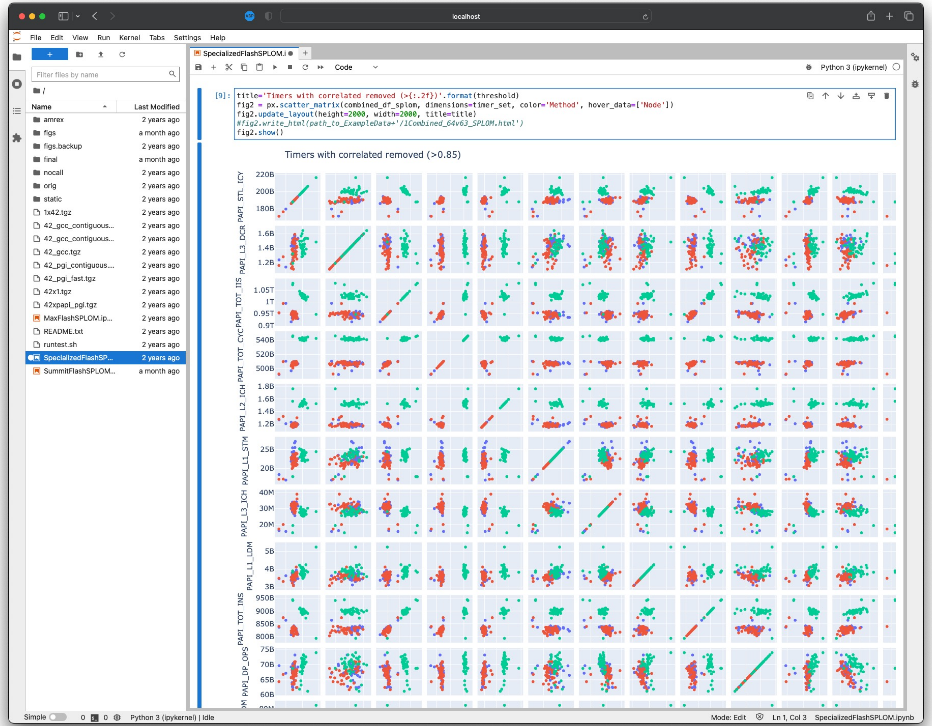
Trace comparison between orig (above timeline, left profile) and nocall (bottom timeline, right profile) Flash-X on Summit, visualized in Vampir. The red regions represent time spent in MPI_Allreduce.

Needed: Analysis Tools

- Need to quickly and easily **visualize the distributions, correlations** of timers/metrics in the performance data
- Vampir, ParaProf, PerfExplorer all have limitations (license limit, capability, scale/age respectively) for runs with 2688+ processes
- **Python** provides nice set of tools to prototype with
  - Pandas
  - Plotly
  - JupyterLab
- TAU has a Python data parser that loads the data into DataFrames
- See https://github.com/Flash-X/SC-22-artifacts for notebooks used in this study

UNIVERSITY OF OREGON

Step 1: Which counters?

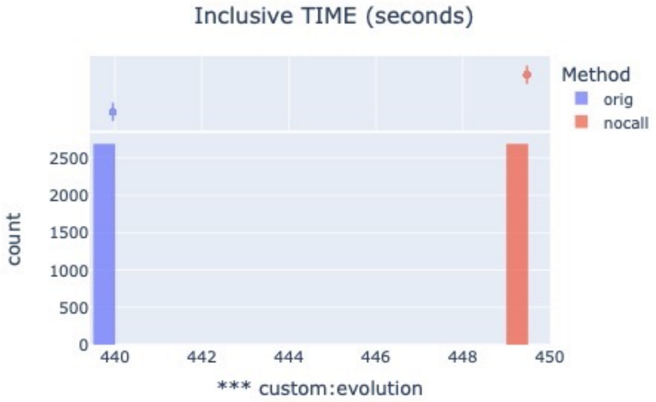


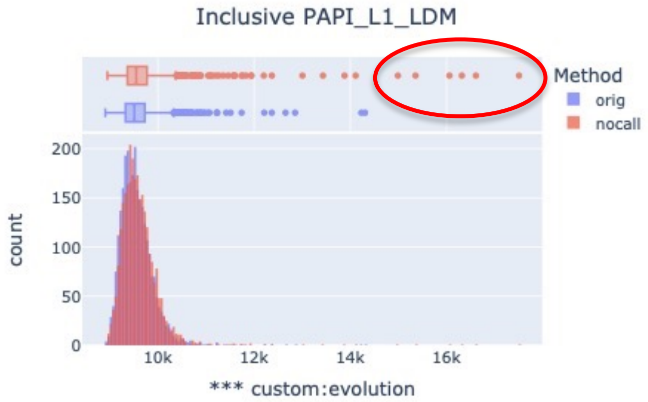
- Collected several PAPI counters from multiple runs
- Which counter(s) are correlated with time (and/or each other)?
- PAPI_L1_DCM, PAPI_RES_STL*
- Also collected PAPI_TOT_INS

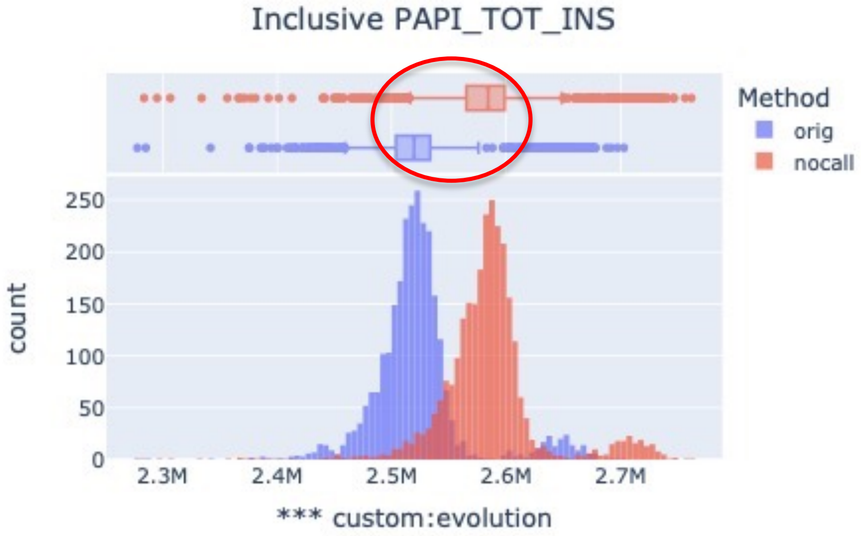
*Only available on Theta

UNIVERSITY OF OREGON

Evolution – High Level Timer

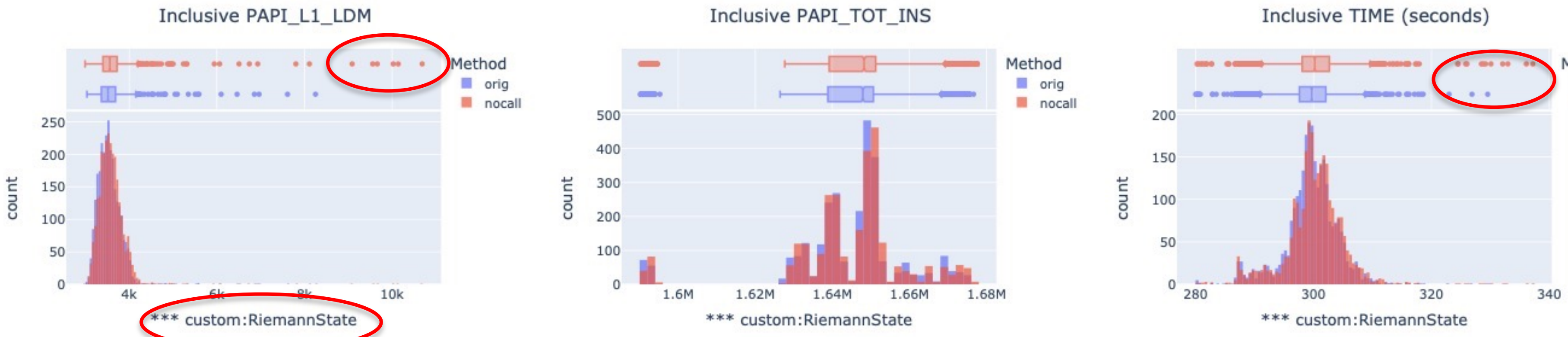




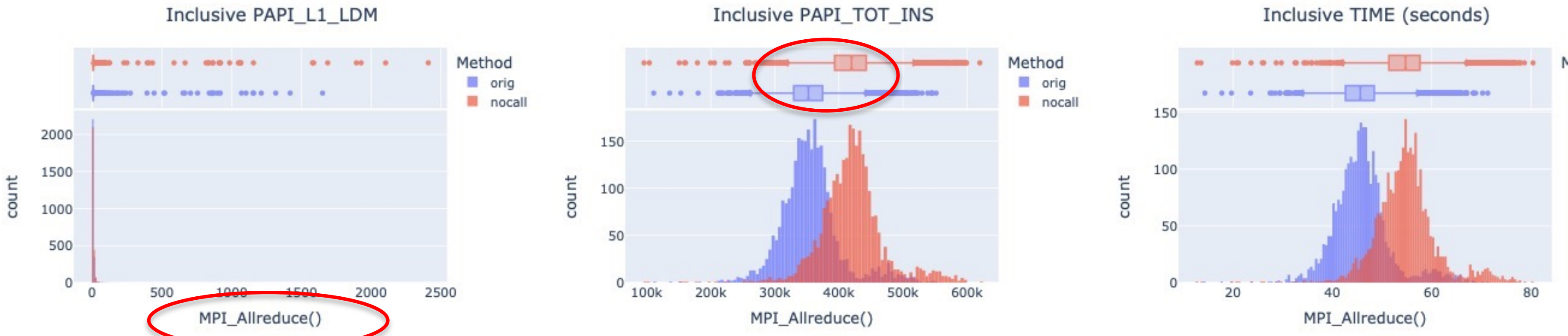


Summit run with 2688 MPI ranks shown. Almost no variability in evolution suggests that MPI collective synchronization is aligning all ranks – we need a lower level timer to tease apart MPI and actual computation. There's also an increase in total instructions – what caused that?

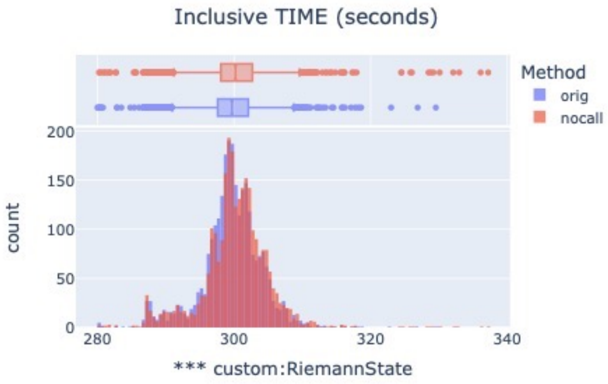
Breaking Down Evolution

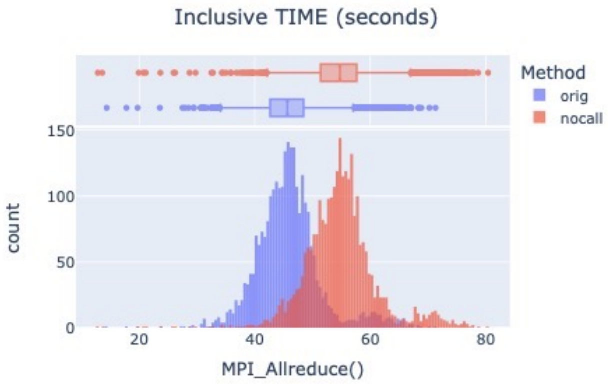


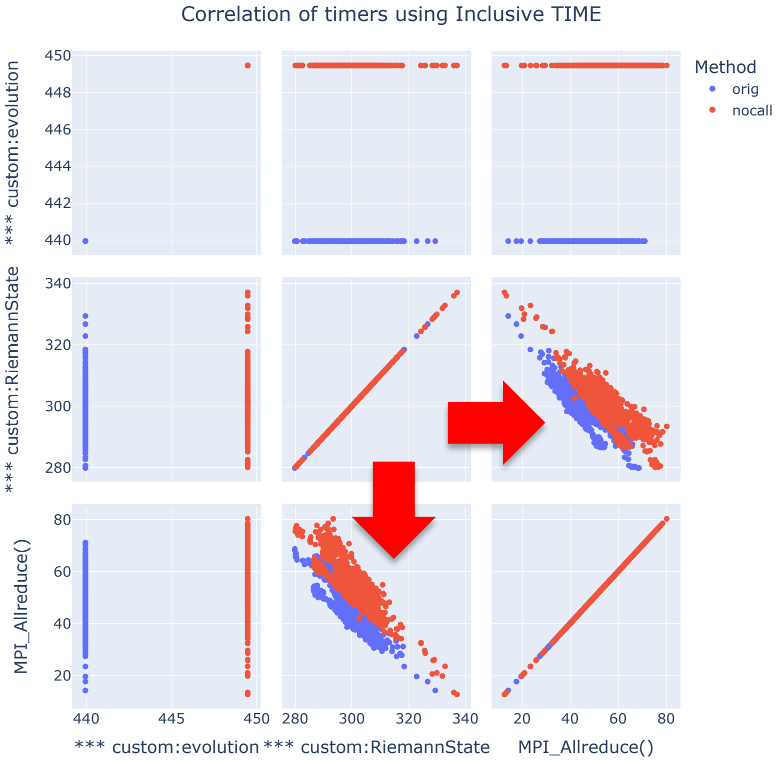
MPI busy waiting at synchronization is cause of increased instructions



Correlation With MPI_Allreduce







Adding Sampling to Dig Deeper

Original

Name	Inclusive TIME ▽	Inclusive PAPI_L1_LDM	Inclusive (PAPI_L1_LDM / PAPI_TOT_INS)
*** custom:hy_getRiemannState.calculating	254.807	2,729,301,359	0.002
[CONTEXT] *** custom:hy_getRiemannState.calculating	251.571	4,132,851,393	0.003
[SAMPLE] __GI___libc_free	68.855	1,083,900,360	0.003
[SAMPLE] __GI___libc_malloc	65.834	1,022,906,177	0.003
[SAMPLE] _int_malloc	29.298	456,492,454	0.003
[SUMMARY] hy_upwindtransverseflux_	24.84	402,237,951	0.003
[SUMMARY] hy_datareconstructnormaldir_mh_	13.62	220,462,336	0.003
[SUMMARY] hy_datareconstonestep_	9.42	146,145,069	0.003
[SAMPLE] hy_slopelimiters_mc_	8.67	133,127,855	0.004
[SUMMARY] pgf90_alloc04a_i8	8.49	139,025,309	0.003
[SUMMARY] hy_getriemannstate_	4.978	76,135,675	0.003
[SAMPLE] __memset_power8	4.95	83,204,722	0.004

No call

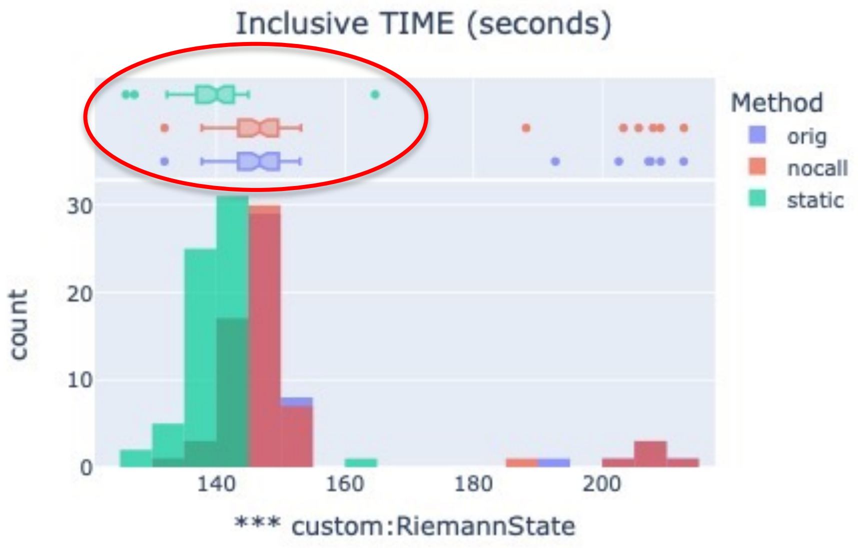
Name	Inclusive TIME ▽	Inclusive PAPI_L1_LDM	Inclusive (PAPI_L1_LDM / PAPI_TOT_INS)
*** custom:hy_getRiemannState.calculating	261.464	5,293,958,279	0.004
[CONTEXT] *** custom:hy_getRiemannState.calculating	257.779	5,990,283,496	0.004
[SAMPLE] __GI___libc_free	67.794	1,577,579,079	0.004
[SAMPLE] __GI___libc_malloc	67.32	1,404,165,584	0.004
[SAMPLE] _int_malloc	29.07	665,616,994	0.004
[SUMMARY] hy_upwindtransverseflux_	27.75	646,351,039	0.004
[SUMMARY] hy_datareconstructnormaldir_mh_	13.5	287,406,289	0.004
[SUMMARY] hy_datareconstonestep_	9.24	196,263,205	0.004
[SUMMARY] pgf90_alloc04a_i8	8.88	177,618,901	0.004
[SAMPLE] hy_slopelimiters_mc_	8.34	181,020,716	0.004
[SUMMARY] hy_eigenvector_	5.28	99,952,763	0.004
[SAMPLE] __memset_power8	5.22	98,019,098	0.004

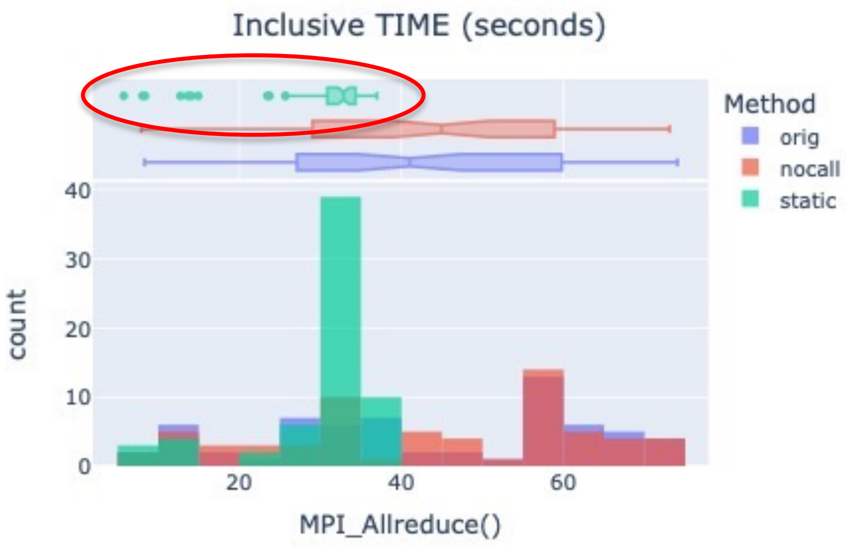
Performance Conclusions

- **High contention for memory subsystem from concurrent processes**
- Main computation is *aggressively* using ALLOCATABLE array variables and ALLOCATE/DEALLOCATE operations
- Manual analysis determined that array variables in the Riemann computation don't change over time – consistent size per process
- Replace them with arrays that are **allocated once and reused/saved** (configuration named "static")

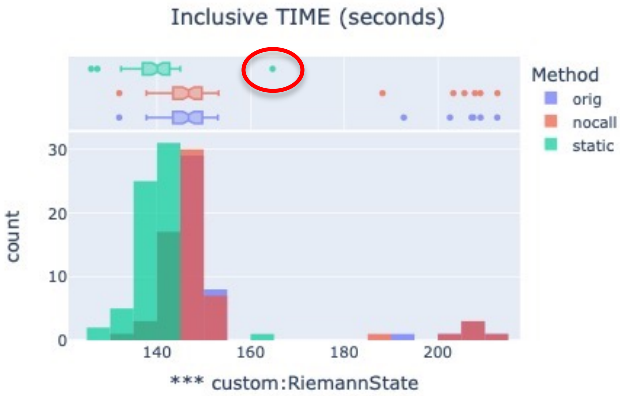
UNIVERSITY OF OREGON

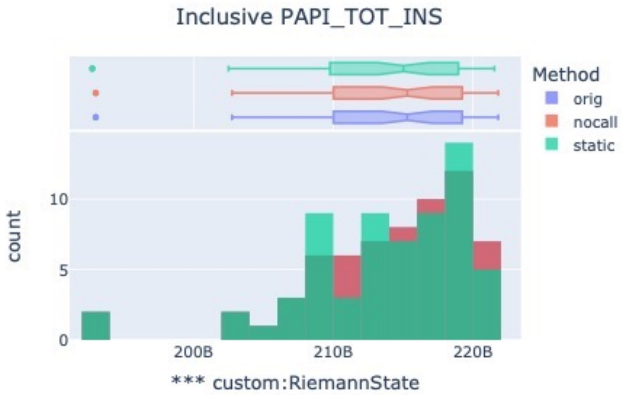
Single Node Performance on Theta

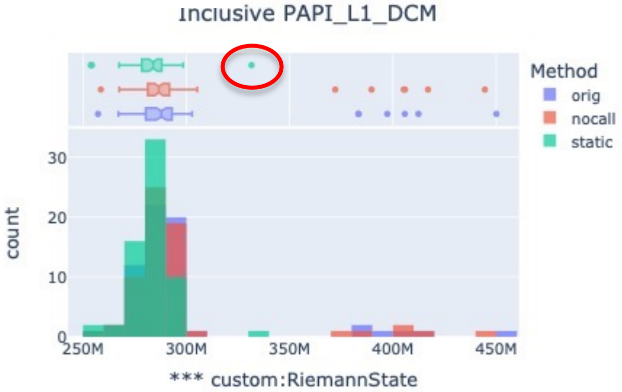


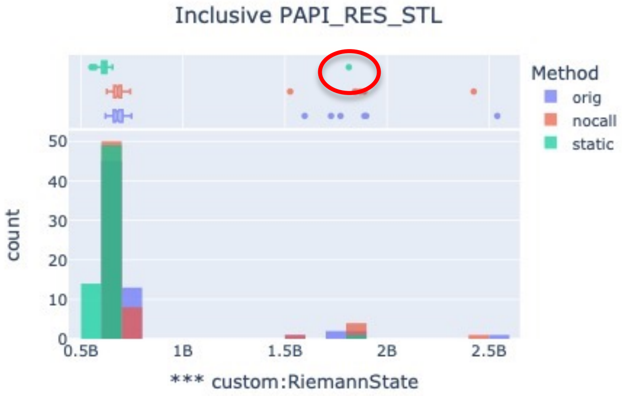


Reduction in L1_DCM, Resource Stalls





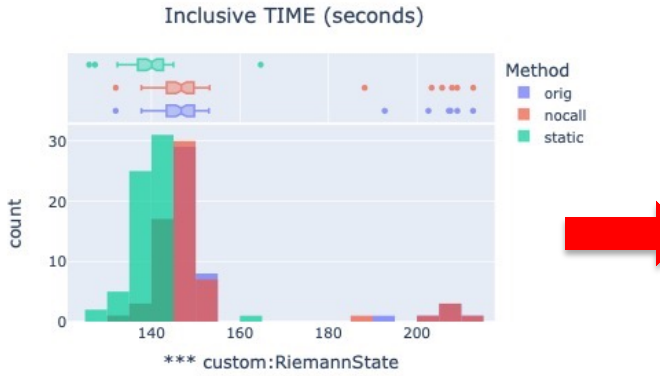




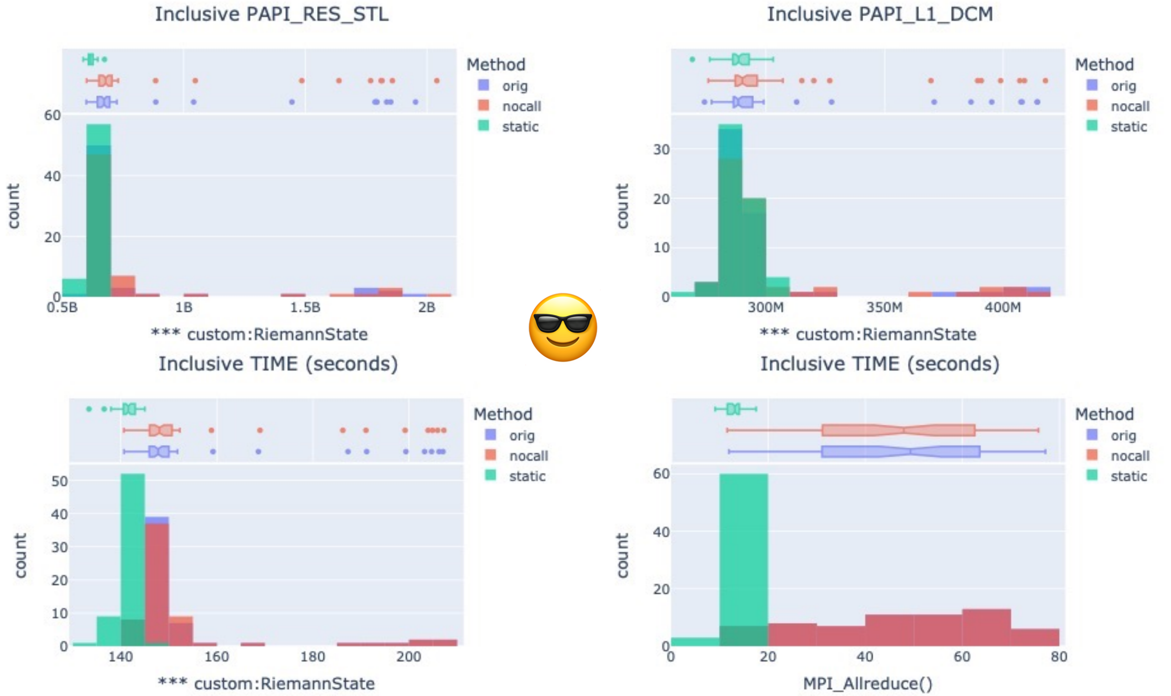
UNIVERSITY OF OREGON

Reserving a Core for the OS

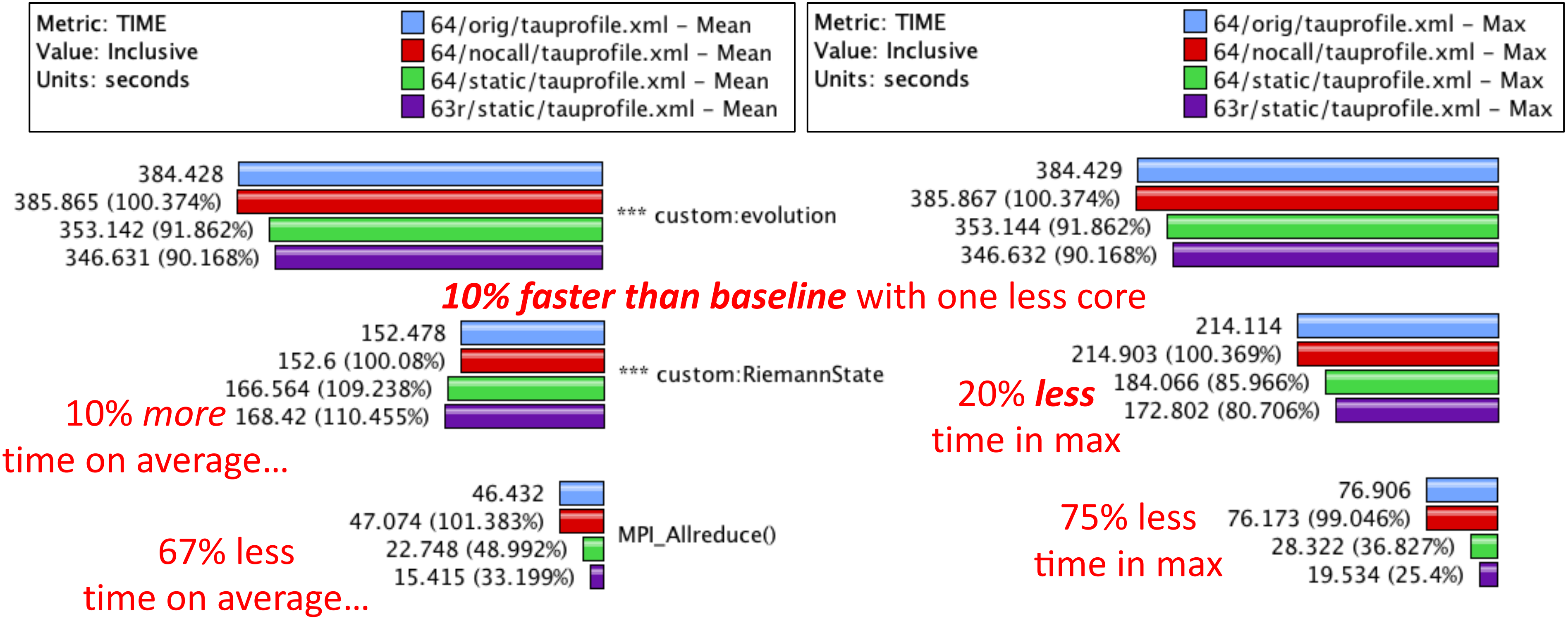
- Determined that lowest rank process on each node is this outlier!
- ALCF has instructions on reserving a core for the OS (specialization)



Change from 64 ranks per node to 63 ranks per node, reserving one core for OS



Final Results: Theta



Overall ***faster*** with ***less contention***, and a more ***balanced load***

Related Work

- Many ad-hoc Python solutions to similar data analysis problems
- PerfExplorer – uses Octave/R, Weka but not performant (Java), not as flexible
- Hatchet – GraphFrame data model – same analysis in this paper *could be done with Hatchet data*
- Load balance analysis not new in HPC, but Python, Plotly, JupyterLab introduces ***ease, flexibility, extensibility***

Conclusions, Future Work

- Removing communication code led to a slowdown in computation
- We developed a set of **Jupyterlab Python scripts** that utilize **Pandas and Plotly** to automate the generation of distribution and correlation visualizations for better understanding of performance behavior
- In this process, we **discovered and removed or mitigated two additional performance limiting bottlenecks** for performance tuning
- Still working on AMReX optimizations to help it benefit from regular mesh in Flash-X

Relevant Links

- Flash-X Project Website: https://flash-x.org
- Flash-X source code: https://github.com/Flash-X/Flash-X
- TAU Website: https://tau.uoregon.edu
- TAU GitHub mirror: https://github.com/UO-OACISS/tau2
- Presented Scripts and Results: https://github.com/Flash-X/SC-22-artifacts

UNIVERSITY OF OREGON

Acknowledgements

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. http://energy.gov/downloads/doe-public-access-plan.

Parts of this research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

U.S. DEPARTMENT OF ENERGY | Office of Science

RAPIDS

ECP EXASCALE COMPUTING PROJECT