Optimizing Gravity and Nuclear Physics in FLASH for Exascale

Hannah Klion
University of California, Berkeley
Berkeley, California
Oak Ridge National Laboratory
Oak Ridge, Tennessee
hannah.klion@gmail.com

O. E. Bronson Messer
Oak Ridge National Laboratory
Oak Ridge, Tennessee
University of Tennessee, Knoxville
Knoxville, Tennessee

J. Austin Harris
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Thomas Papatheodore
Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT

In a Type Ia supernova, runaway fusion ignites in a white dwarf, causing it to explode. The heavy element yields of these events remain uncertain, and high-performance multiphysics simulations with tools like FLASH are critical for our understanding. Current simulations track approximately a dozen nuclear isotopes, as opposed to the thousands required to completely capture the event's nuclear physics.

Simulating nuclear physics and self-gravity accurately and efficiently is critical for modeling a Type Ia supernova, since supernovae are competitions between energy-releasing nuclear reactions and gravity. Currently, the FLASH nuclear reaction network and self-gravity solver requires substantial inter-node communication. We use non-blocking MPI collectives to overlap communication in the self-gravity calculation with the computation-heavy nuclear burning calculation. We find that speedups from this technique are possible, but are MPI implementation-dependent. We highlight some of the challenges associated with this type of optimization.

CCS CONCEPTS

• **Applied computing** → **Astronomy**; • **Computing methodologies** → *Massively parallel and high-performance simulations*;

KEYWORDS

Computational astrophysics/astronomy, chemistry, and physics; Solutions for parallel programming challenges

ACM Reference Format:
Hannah Klion, O. E. Bronson Messer, J. Austin Harris, and Thomas Papatheodore. 2017. Optimizing Gravity and Nuclear Physics in FLASH for Exascale. In *Proceedings of ACM SuperComputing 17, Denver, Colorado, USA, November 2017 (SC'17),* 3 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.
SC'17, November 2017, Denver, Colorado, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

FLASH is a high performance multiphysics code currently used for a wide range of astrophysical simulations [2]. In particular, it is a commonly-used tool for the simulation of Type Ia supernovae (SNe), in which runaway fusion ignites in a white dwarf and causes it to explode. The heavy element yields of these events remain uncertain. Current simulations track about a dozen isotopes, as opposed to the thousands required to completely capture the nuclear physics.

In many multiphysics codes, each spatial grid point has many degrees of freedom, such as composition and thermodynamic and mechanical quantities. Evolving these quantities from timestep to timestep generally requires solving highly-coupled systems of equations and therefore takes many FLOP/s. Components of this update step, though, may depend on the results of global operations, which are frequently bandwidth, not latency bound. When blocking MPI collectives are used to perform these operations, the communication cannot overlap with the local calculation.

Since supernovae are fundamentally competitions between self-gravity and energy-releasing nuclear reactions, accurately computing gravitational potentials is critical. The self-gravity calculation uses an expensive global collective with large message sizes. The nuclear reaction calculation is generally the most computation-heavy physics routine in a FLASH Type Ia SN simulation. We attempt to speed up Type Ia SN simulations by replacing a blocking MPI collective in the self-gravity routine with a non-blocking equivalent and overlap the gravity communication with the nuclear burning calculation.

We run all tests on Titan at OLCF.

2 MULTIPOLE POISSON SOLVER

The gravitational potential $\phi(\mathbf{x})$ at point $\mathbf{x}$ due to a mass distribution is given by the Poisson equation

$$\nabla^2\phi(\mathbf{x}) = -4\pi G\rho(\mathbf{x}), \tag{1}$$

where $\rho(\mathbf{x})$ is the mass density, and G is the gravitational constant. Solving equation 1 directly using the Green's function is not efficient. Instead, for centrally-concentrated problems, FLASH uses the multipole expansion of the potential which is suitable for mass distributions dominated by low spherical

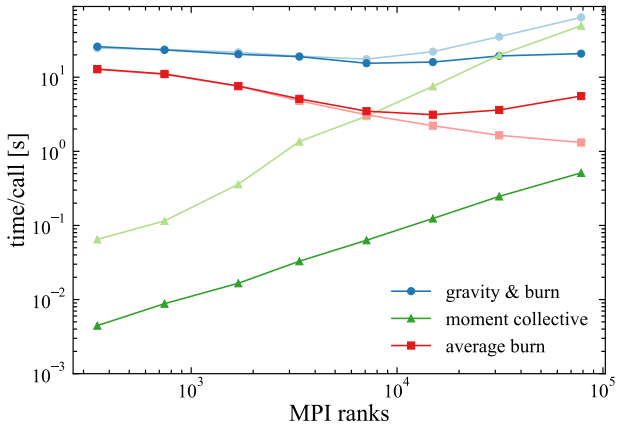

Figure 1: Approximate weak scaling behavior of components of FLASH multipole gravity and nuclear burning calculations, using non-blocking (dark colors) and blocking (light) DMAPP MPI collectives and progress engines. The total time is shown in blue. Starting at $\sim 10^4$ MPI ranks, the non-blocking implementation outperforms the blocking case.

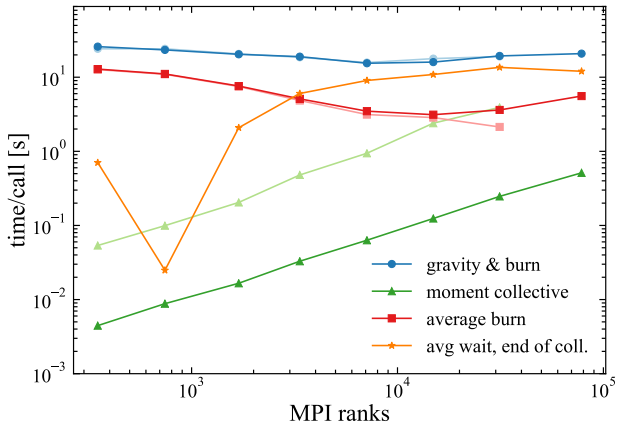

Figure 2: Same as figure 1, but blocking collectives (light colors) use default MPICH implementations and have collective optimizations turned off. Non-blocking collectives (dark colors) use DMAPP collectives and a progress engine, as in figure 1.

orders. Implementation details for this calculation can be found in [1].

Broadly speaking, the calculation can be divided into three parts: determining the center of mass, computing the multipole moments, and finding the final gravitational potential. The center of mass and multipole moment calculations both involve local computation followed by an MPI collective, requiring pairwise communication between every node. The multipole moment collective is more expensive by orders of magnitude due to the larger amount of data, so it is the primary target of our optimization efforts.

In the default version of FLASH, the gravity and burning calculations occur independently in serial. Our approach involves splitting the gravity calculation into two components: the first performs the local calculation to obtain the moments and calls the non-blocking collective. The nuclear burning calculation is then performed. Once the burning calculation has finished, the collective will have made progress; once the communication is finished, each rank computes the gravitational potential at each point.

We compare the weak scaling performance of our implementation with blocking and non-blocking collectives in figures 1 and 2. For the non-blocking timings in both figures, we use the DMAPP MPI implementation and a progress engine with core specialization. Figure 1 shows the performance of blocking collectives with the same MPI configuration, while in figure 2 we use the default MPICH implementation (i.e. not DMAPP) and use `MPICH_COLL_OPT_OFF=1`, which includes turning off shared memory optimizations. We find that non-blocking collectives provide speedups over blocking DMAPP collectives at greater than $\sim 10^4$ ranks. They do not, however, improve overall performance over the unoptimized default MPI implementation, at least up to $\sim 3 \times 10^4$ ranks. The poor performance of DMAPP-based blocking collectives occurs because the message sizes in the collective are too large to take advantage of the OS-bypassing fast memory access afforded by DMAPP. This behavior on the Gemini network of Titan is in contrast to earlier work on the Cray Aries interconnect [3]. We do not find evidence that the application is subject to system throttling of MPI messages in any logs or application output.

The burning calculation is independent of MPI, yet it is slower in the non-blocking case than in the blocking case. There is evidence that this is due to interference and/or competition for resources between the progress engine and the local calculation. Even though we initialize a progress engine in the blocking case, it does not use as many resources.

3 CONCLUSIONS

We have demonstrated that even for large communicator sizes and in cases where the blocking collective is a substantial fraction of the execution time, speedups from non-blocking collectives are not guaranteed. The progress engine can substantially hamper the performance of the concurrent calculation. Further, the speedups are substantially implementation and configuration-dependent. Applications with large message sizes running on a Gemini network may be significantly negatively impacted by DMAPP blocking collectives and shared memory copy optimizations.

Our production target is 3D simulations on the new Summit platform at OLCF. It is unclear how (1) the increased payload size for 3D and (2) the SpectrumMPI implementation of non-blocking collectives will impact these results

Supplementary information: `https://goo.gl/etPtBw`

ACKNOWLEDGMENTS

HK is supported by the Department of Energy Computational Science Graduate Fellowship, provided under grant number DE-FG02-97ER25308. 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. The software used in this work was in part developed by the DOE NNSA-ASC OASCR Flash Center at the University of Chicago.

REFERENCES

[1] S. M. Couch, C. Graziani, and N. Flocke. 2013. An Improved Multipole Approximation for Self-gravity and Its Importance for Core-collapse Supernova Simulations. *The Astrophysical Journal* 778, Article 181 (Dec. 2013).

[2] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. 2000. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement* 131 (Nov. 2000), 273–334.

[3] C. Niethammer, P. Manninen, R. W. Nash, D. Khabi, and Garcia J. 2014. MPI Collectives at Scale. *CRESTA Whitepaper* (2014).