Optimizing Gravity and Nuclear Physics in FLASH for Exascale

Hannah Klion[1,2], O. E. Bronson Messer[2,3], J. Austin Harris[2], Thomas Papatheodore[2]
University of California Berkeley[1], Oak Ridge National Laboratory[2], University of Tennessee Knoxville[3]

Supplementary Information

Introduction and Motivation

FLASH is a high-performance multiphysics code used for a wide range of astrophysical simulations. It is commonly used to simulate Type Ia supernovae (SNe), in which runaway fusion ignites in a white dwarf (WD), causing it to explode.

Computational challenges

- Simulating nuclear physics and self-gravity accurately and efficiently is critical for modeling a Type Ia supernova.
- Common pattern in multiphysics codes: spatial grid points have many degrees of freedom (requiring heavy local computation) but also perform global operations, which are frequently not latency-bound.
- With blocking MPI collectives, global operations cannot overlap with local calculations.

Approach

- Self-gravity calculation has an expensive global collective with large message sizes.
- Nuclear reaction network calculation is local and takes a large number of FLOP/s.
- Replace blocking MPI collective in self-gravity with a non-blocking one and overlap non-blocking collective with computation-heavy nuclear burn.
- All tests are run on Titan at OLCF.

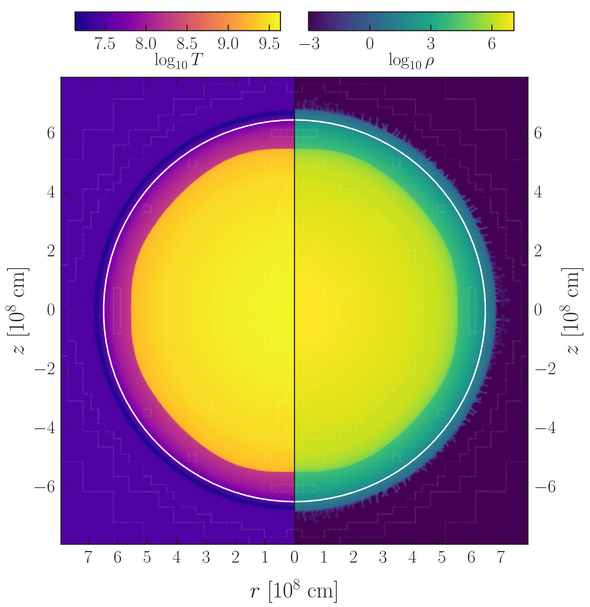
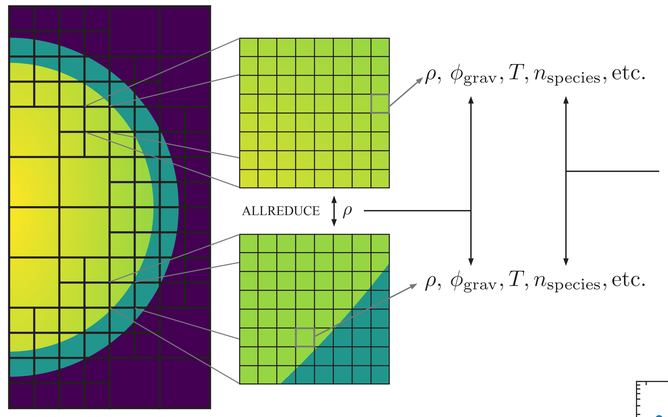


Fig 1 (left). Temperature (left) and density (right) of a simulation of a Type Ia supernova. The snapshot is after the shock front has reached the surface. The white line shows the surface of the original WD model.



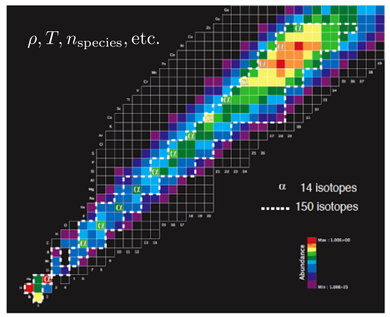


Fig 2. Schematic representation of FLASH's physics evolution. Each block in the AMR grid contains a fixed-resolution grid. At each gridpoint, there are many degrees of freedom. An update requires an ALLREDUCE collective and a local nuclear reaction network calculation.

Fig 3. Pseudocode of new evolution routine

```
1: procedure EVOLVE__NEW
2:     dt ← initdt
3:     t ← 0
4: loop:
5:     HYDRO
6:     DIFFUSION
7:     GRAVITY__INIT
8:     BURN
9:     GRAVITY__FINALIZE
10:    dt ← NEWDT
11:    if END__CONDITION then
12:        return
13:    else
14:        t ← t + dt
15:        goto loop
16: function GRAVITY__INIT
17:    CENTER__OF__MASS
18:    MOMENTS__INIT
19:    return
20: function GRAVITY__FINALIZE
21:    MOMENTS__FINALIZE
22:    POTENTIAL
23:    return
24: function MOMENTS__INIT
25:    LOCAL__CALCULATION
26:                          ▷ non-blocking collective
27:    MPI__IALLREDUCE
28:    return
29: function MOMENTS__FINALIZE
30:                          ▷ wait for communication to finish
31:    MPI__WAITALL
32:    MPI__BARRIER
33:    STORE__DATA
34:    NORMALIZE
35:    return
```

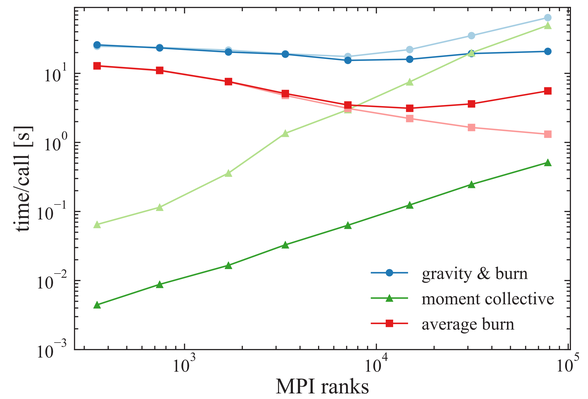


Fig 4. Weak scaling behavior of components of the gravity & burn calculation, using non-blocking (dark) and blocking (light) DMAPP MPI collectives. The non-blocking collectives provide a significant overall speedup.

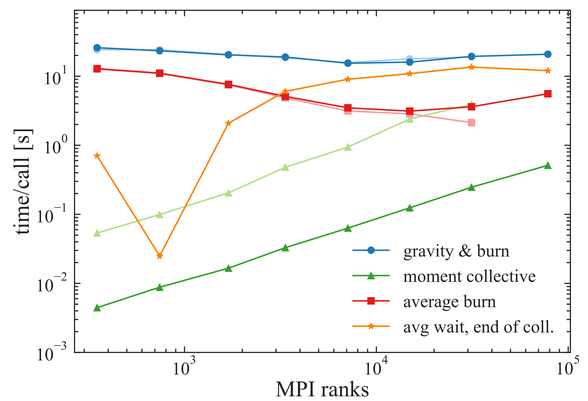


Fig 5. Weak scaling behavior of components of the gravity & burn calculation. Blocking collectives (light) use the default MPICH implementation (i.e. not DMAPP) and use `MPICH_COLL_OPT_OFF=1` (including turning off the default shared memory collective behavior). Non-blocking collectives (dark) are the same as in Fig. 4. This configuration for the blocking collectives is competitive with the DMAPP-based non-blocking collectives.

Results

- Naively, one would expect that using non-blocking collectives to overlap communication and computation would always lead to a speedup.
- This occurs in Fig. 4 — compared with DMAPP-based blocking collectives, non-blocking collectives show a speedup at ~10^4 ranks.
- However, for large message sizes, DMAPP blocking collectives are an order of magnitude slower than the default MPICH implementation (with `MPICH_COLL_OPT_OFF=1`) . Fig. 5 demonstrates this behavior — the non-blocking collectives do not provide a speedup over the unoptimized blocking collectives. The size of the messages in the collective are too large to take advantage of the OS-bypassing fast memory access (FMA) afforded by DMAPP. This behavior on the Gemini network of Titan is in contrast to earlier work on the Cray Aries interconnect (cf. Niethammer et al. 2014).
- The burning calculation is entirely local, 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. Though we initialize a progress engine in the blocking case, it is not as expensive.

Conclusions

- Care should be taken when selecting an MPI implementation and tuning environment variables.
- There may be unexpected interaction between application components.
- 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.

References and Acknowledgments

B. Fryxell et al, 2000. *The Astrophysical Journal Supplement* **131** (Nov. 2000), 273–334.
S. Couch et al, 2013. *The Astrophysical Journal* **778**, Article 181 (Dec. 2013).
C. Niethammer et al, 2014. *CRESTA Whitepaper* (2014).

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.