Preparing for Mira: experience with FLASH multiphysics simulations

Petascale Simulations of Turbulent Nuclear Combustion

Christopher Daley [1] [2]

[1]The Flash Center for Computational Science at the University of Chicago
[2]Argonne National Laboratory

March 7, 2013

Science objectives

To improve our understanding of the explosion mechanism of Type Ia Supernova

Simulate two key physical processes with the FLASH code:

1. Buoyancy-driven turbulent nuclear combustion
   - Determines the amount of nuclear energy released during the ordinary flame burning phase (the so-called "deflagration phase")
2. Transition from the nuclear flamelet regime to distributed nuclear combustion
   - Necessary condition for initiation of a detonation in the Deflagration to Detonation Transition (DDT) model

In this talk I will refer to these simulations as "RTFlame" and "DDT", respectively

The FLASH code

FLASH is a multi-physics finite-volume Eulerian code and framework with the following capabilities relevant to the early science applications:

- Directionally unsplit hydrodynamics solver
- Multipole gravity solver
- Nuclear burning network
- Equation of state (EOS) for degenerate matter
- Turbulent Flame Interaction (TFI) model
- Adaptive Mesh Refinement (AMR) with Paramesh
- Parallel I/O using HDF5 library
- Lagrangian tracer particles

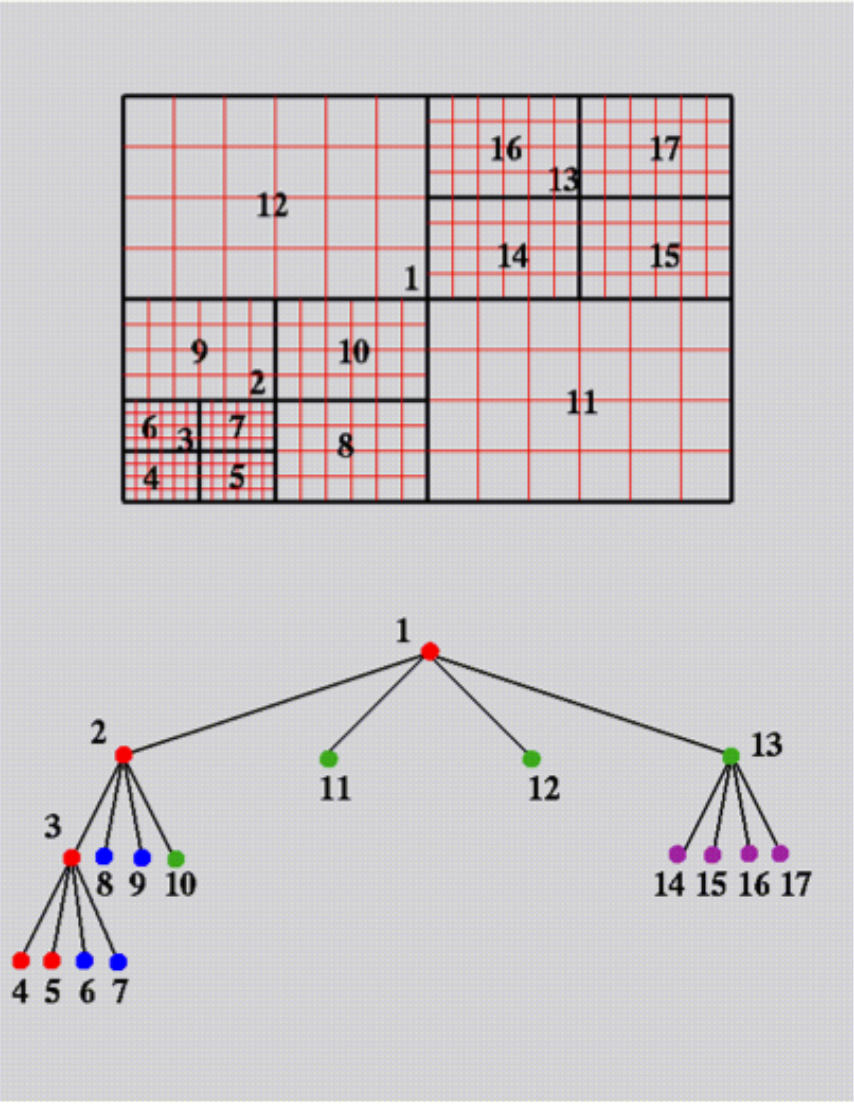
Capabilities in blue are being used for the first time in Flash Center Type Ia simulations

- 1.2 million lines of code (75% code, 25% comments)
- Written in Fortran90 and C
- Collection of code units which a user assembles into a custom application

Updating FLASH for BG/Q

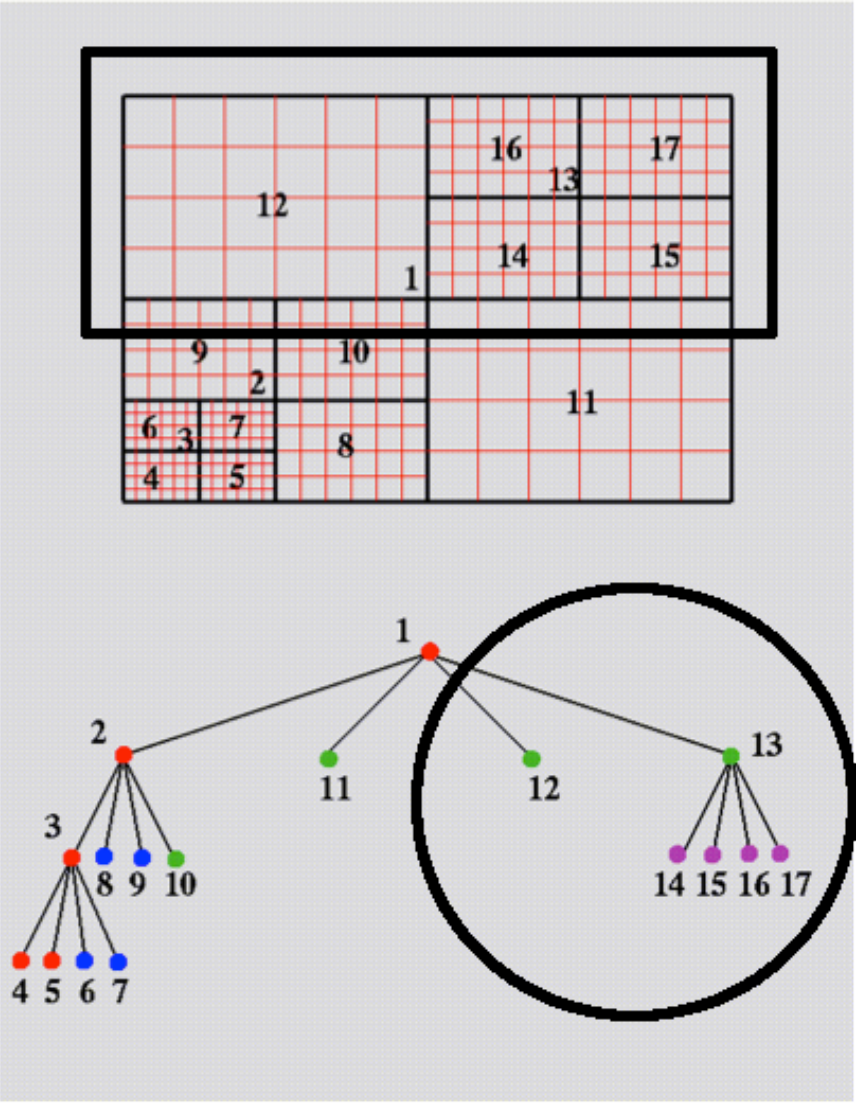
- We multithreaded FLASH using OpenMP directives in order to run more efficiently on BG/Q
  - A natural choice because FLASH is mostly Fortran
- This allows us to make use of multiple hardware threads on a core and hide memory latency
  - Multiple MPI ranks on a core is not really an option because of the memory overhead of FLASH early science applications
- The placement of the OpenMP directives and how it fits with the standard MPI decomposition provided by Paramesh is described in the following slides

Overview of AMR with Paramesh



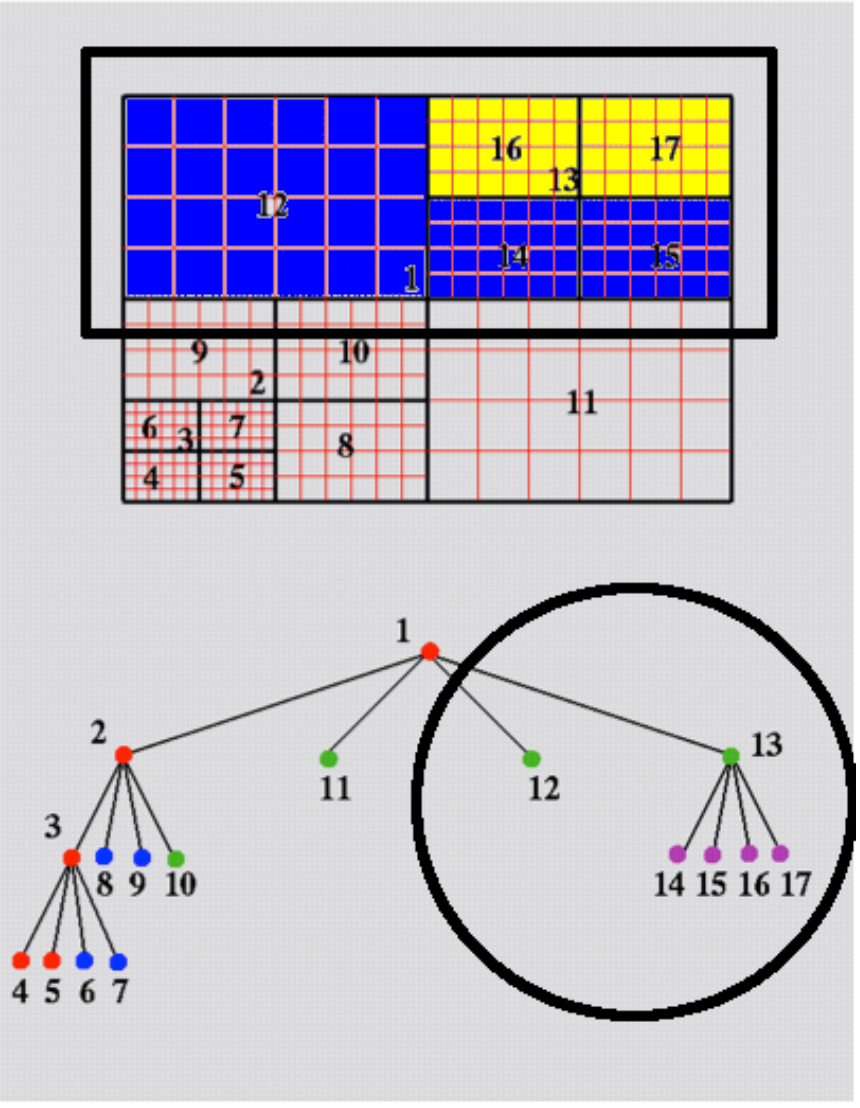
- Mesh is divided into blocks of fixed size (typically 16^3 cells)
- Blocks contain a layer of guard cells containing a copy of neighboring blocks solution data
  - Explicit solvers in FLASH perform stencil updates to advance solution data
- Blocks refine/derefine according to a user-specified refinement criteria and are organized in a Oct-Tree (3D) hierarchy

Standard MPI parallelism in Paramesh



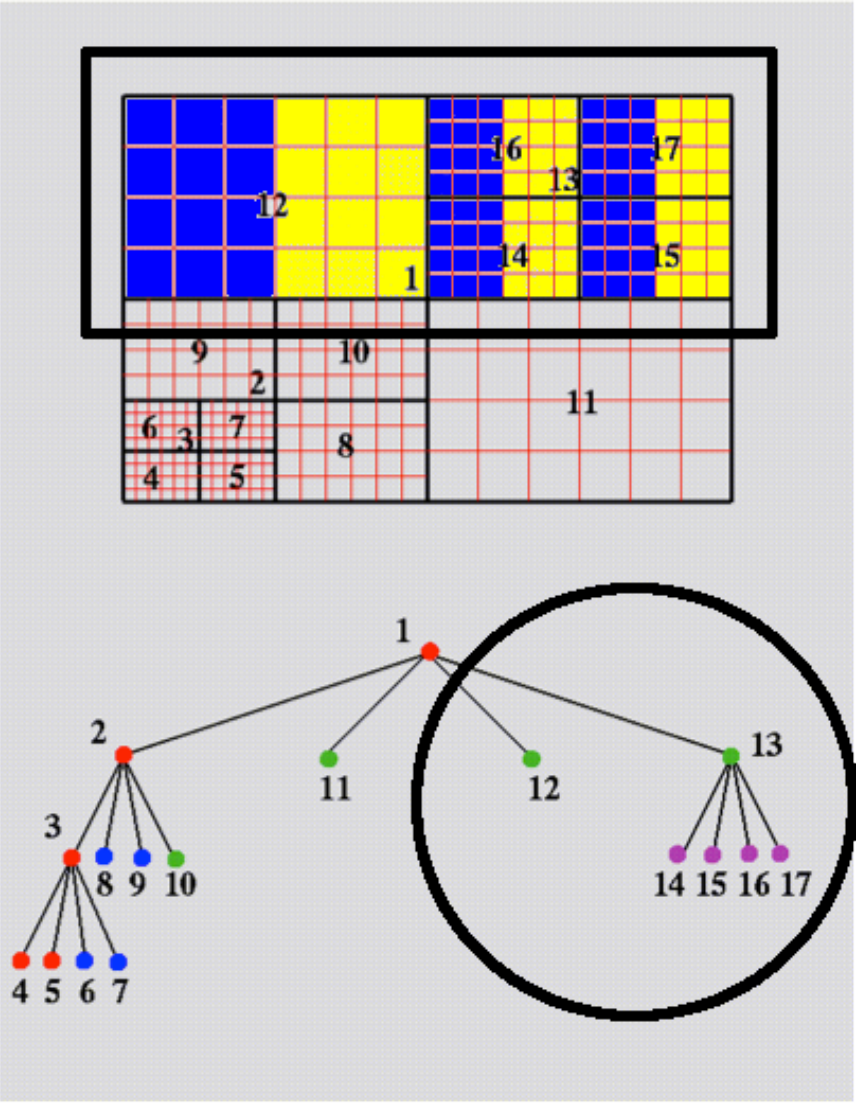
- The thick black lines show blocks 12 through 17 being assigned to a single MPI rank
  - 6 total blocks
    - 5 leaf blocks
    - 1 parent block
- FLASH solvers update the solution on **local** leaf blocks
- We keep this decomposition and add OpenMP directives to expose more data parallelism for BG/Q

Multithreading strategy 1



- Assign different blocks to different threads
- Assuming 2 threads per MPI rank
  - Thread 0 (blue) updates 3 full blocks - 72 cells
  - Thread 1 (yellow) updates 2 full blocks - 48 cells
- This will be referred to as "thread block list"

Multithreading strategy 2



- Assign different cells from the same block to different threads
- Assuming 2 threads per MPI rank
  - Thread 0 (blue) updates 5 partial blocks - 60 cells
  - Thread 1 (yellow) updates 5 partial blocks - 60 cells
- This will be referred to as “thread within block”

Optimization

- A complete FLASH run consists of 3 phases: initialization, evolution and finalization
- The evolution phase controls the time-stepping of the simulation and is normally the dominant cost
- We needed to optimize initialization and evolution to run more efficiently on BG/Q. We show our optimizations in 2 parts:

1. Optimizing the initialization phase of a DDT simulation
   - This reduced initialization time from hours to minutes
2. Optimizing the evolution phase of a RTFlame simulation
   - This reduced time to solution by 33% (it would be 39% if including unused optimizations)
   - Note that these optimizations also benefit DDT simulations

Slow initialization in the DDT simulation

- The new DDT simulations were previously only run at scale on Ranger at Texas Advanced Computing Center (TACC)
- A test problem failed to initialize in 2 hours when using 8192 MPI ranks on Vesta BG/Q. Two reasons for the slowness
  1. Fortran `random_number` function is relatively slow on BG/Q
     - $4\mu s$ on BG/Q with xlf-14.1
     - $30ns$ on a x86_64 platform with gfortran-4.4.4
     - Removing many unnecessary calls to `random_number` resolved this issue - simple
  2. Custom read of the turbulence field data from a HDF5 file is slow
     - Puzzling because data is read using collective parallel I/O and the file is only 385 MB...

Slow initialization in the DDT simulation

- Core files showed many MPI ranks in the call stack of `MPI_file_read_at` when job was terminated
  - This is an independent MPI-IO function!
- The HDF5 function `H5Pget_mpio_no_collective_cause` (HDF5 $>=$ 1.8.10) revealed that a datatype conversion prevented collective I/O data transfer
  - The turbulence field dataset has type `H5T_IEEE_F64LE`
    - "LE" indicates little-endian
  - Blue Gene is a big-endian platform
- Changing the dataset type to `H5T_IEEE_F64BE` enabled collective I/O and improved read performance by 2 orders of magnitude (see Figure 1 in next slide)

Collective I/O and HDF5 dataset endianness

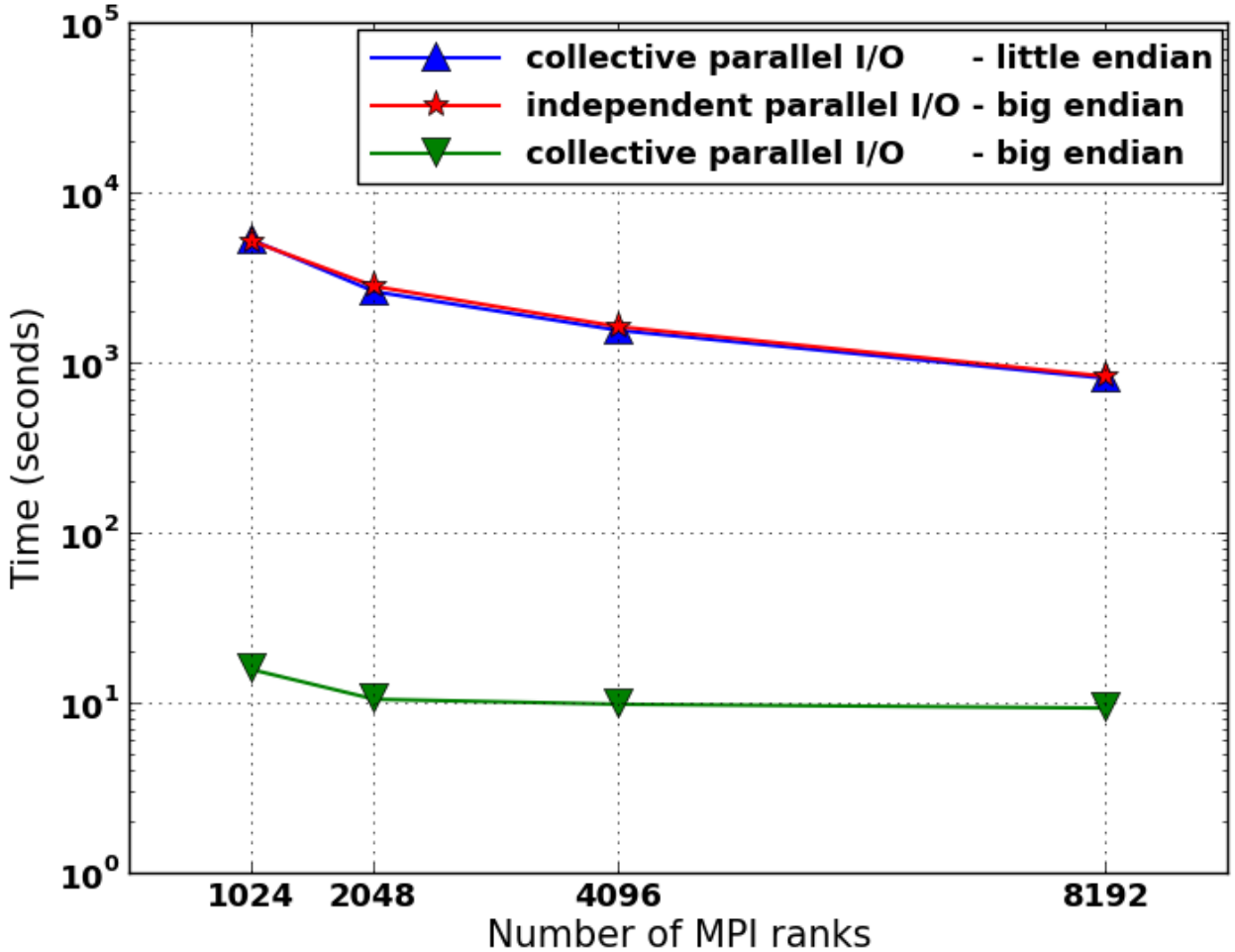


Figure 1: Time to read the turbulence field file on Vesta BG/Q (used 16 MPI ranks per node in MPI-only configuration)

Optimizing the evolution in the RTFlame simulation

- Many optimization opportunities found using IBM's High Performance Computing Toolkit (HPCT)
  - Linking against the `libmpihpm_smp.a` library provides
    - Statement level profiling through `vprof`
    - Hardware counter summary information
    - MPI performance data
- Analyzing the `vprof` data led us to
  - Explicitly link against the Mathematical Acceleration Subsystem Software (MASS) library to get a faster log function
  - Reorder arrays in unsplit hydro to avoid expensive temporary array copies (see next slide)
- We show the impact of these changes (and other changes) later in Figure 2

Unsplit hydro array layout optimization

```
689   479   call hy_uhd_dataReconstOnestep &
...
711           leig(1:NDIM,i,j,k,1:HY_WAVENUM,1:HY_VARINUM),&
712           reig(1:NDIM,i,j,k,1:HY_VARINUM,1:HY_WAVENUM))
```

479 counts (approximately 4.79 seconds)

Unsplit hydro array layout optimization

```
689   479   call hy_uhd_dataReconstOnestep &
...
711           leig(1:NDIM,i,j,k,1:HY_WAVENUM,1:HY_VARINUM),&
712           reig(1:NDIM,i,j,k,1:HY_VARINUM,1:HY_WAVENUM))
```

479 counts (approximately 4.79 seconds)

- Reorder arrays so that i,j,k are the slowest varying dimensions
  - allows us to pass a memory address instead of creating a temporary array

Unsplit hydro array layout optimization

```
689   479   call hy_uhd_dataReconstOnestep &
...
711           leig(1:NDIM,i,j,k,1:HY_WAVENUM,1:HY_VARINUM),&
712           reig(1:NDIM,i,j,k,1:HY_VARINUM,1:HY_WAVENUM))
```

479 counts (approximately 4.79 seconds)

- Reorder arrays so that i,j,k are the slowest varying dimensions
  - allows us to pass a memory address instead of creating a temporary array

```
689    37   call hy_uhd_dataReconstOnestep &
...
711           leig(1,1,1,i,j,k),&
712           reig(1,1,1,i,j,k))
```

37 counts (approximately 0.37 seconds)

Optimizations that reduce FLASH evolution time

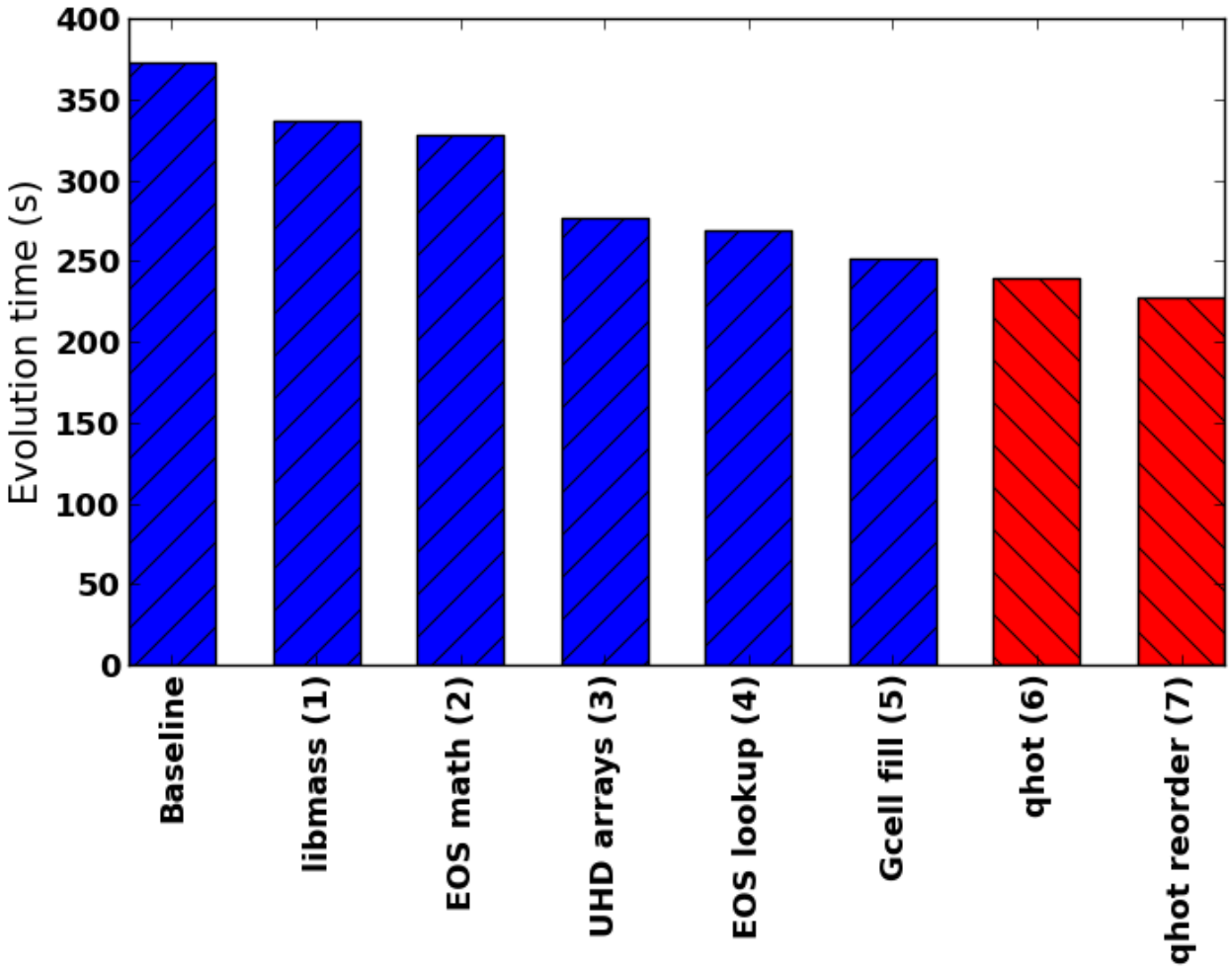


Figure 2: Impact of each successive optimization on RTFlame evolution time (used 16 MPI ranks per node and 4 threads per MPI rank on Vesta BG/Q). Blue bars indicate optimizations currently being used in FLASH early science applications.

Performance overview

The performance section is divided into 2 parts:

1. Most efficient way to run a **fixed-size** FLASH test problem
   - A BG/Q node is very flexible. Must choose
     - MPI ranks per node
     - OpenMP threads per MPI rank
   - FLASH has 2 different styles of multithreading
     - The coarse grained "thread block list"
     - The finer grained "thread within block"
2. Performance of the best configuration of a fully-optimized FLASH binary
   - Strong and weak scaling
   - Hardware counter summary data

Performance matrix

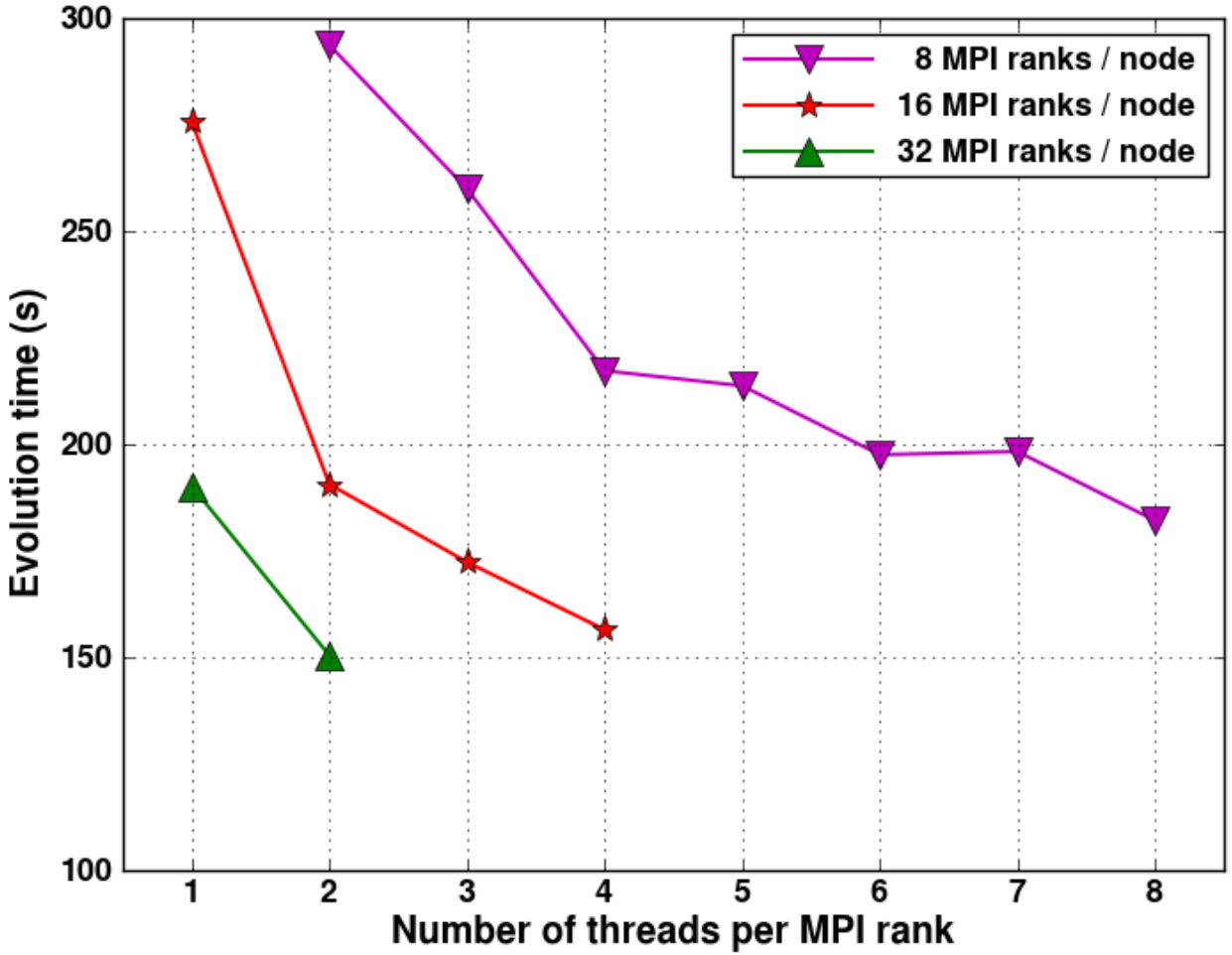


Figure 3: Time to solution in a fixed RTFlame test problem on 128 nodes of Vesta BG/Q when varying the number of MPI ranks and OpenMP threads per node.

Performance matrix

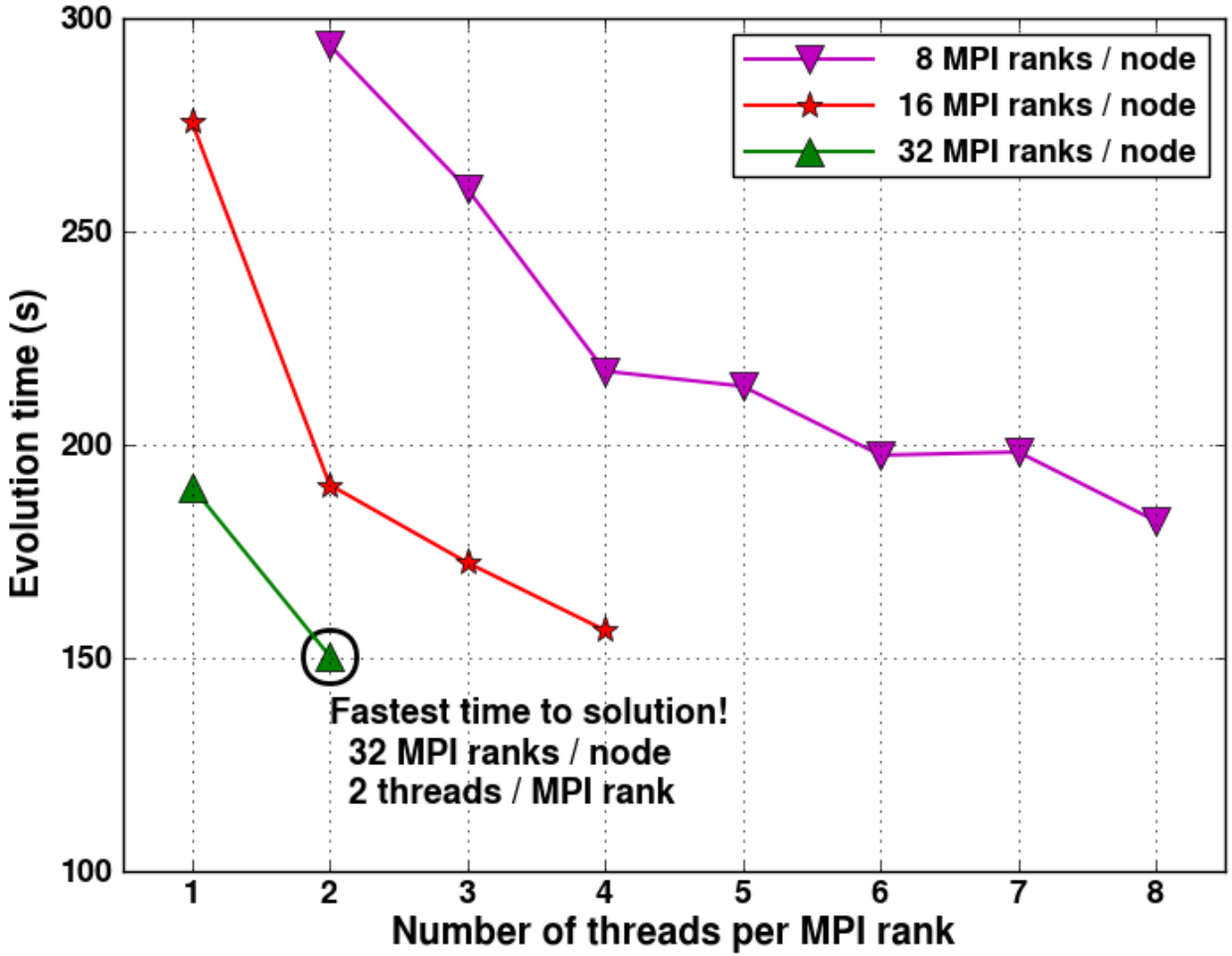


Figure 4: Time to solution in a fixed RTFlame test problem on 128 nodes of Vesta BG/Q when varying the number of MPI ranks and OpenMP threads per node. The fastest time to solution is circled.

Performance matrix

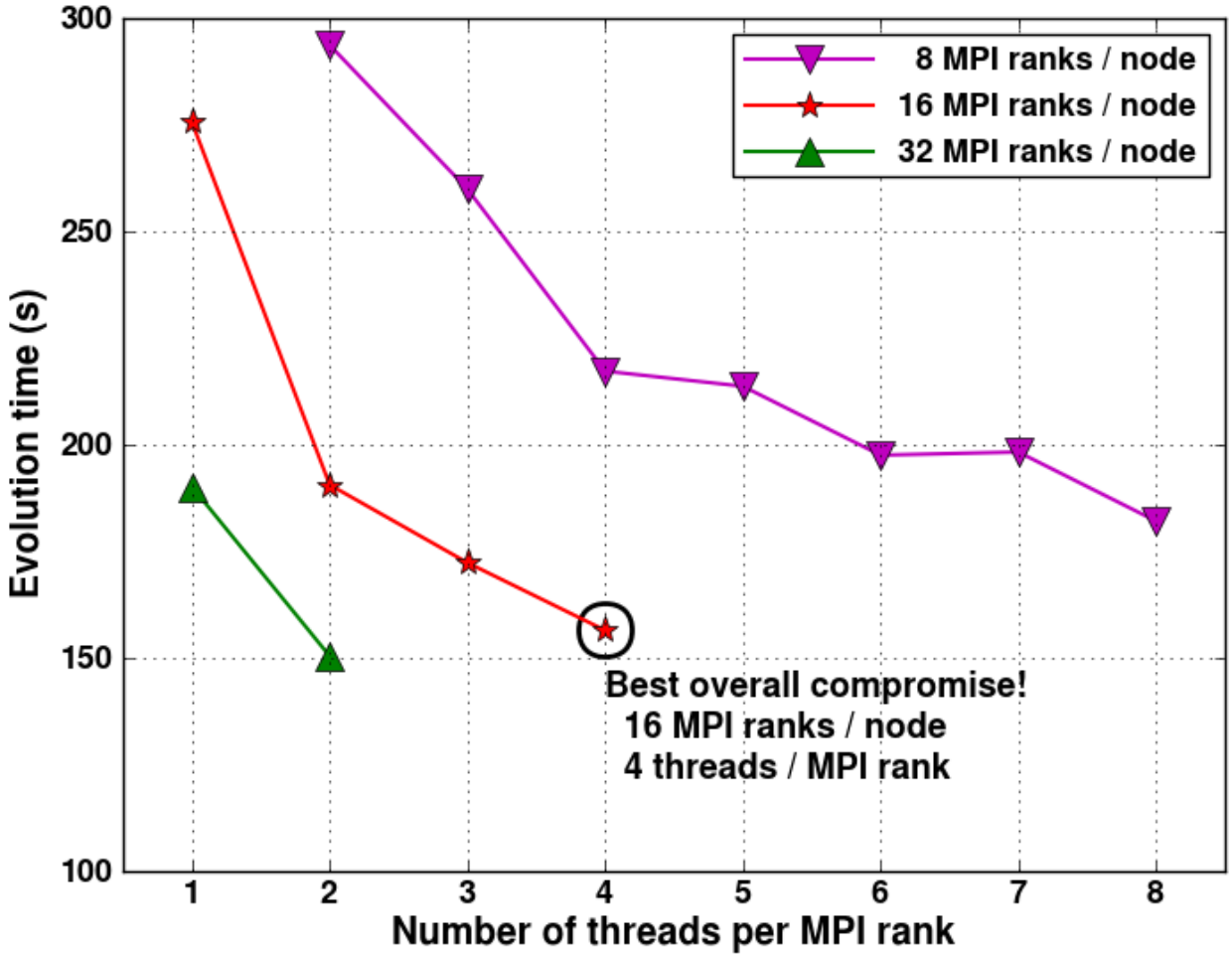


Figure 5: Time to solution in a fixed RTFlame test problem on 128 nodes of Vesta BG/Q when varying the number of MPI ranks and OpenMP threads per node. The best compromise between time to solution and memory usage is circled.

Speedup

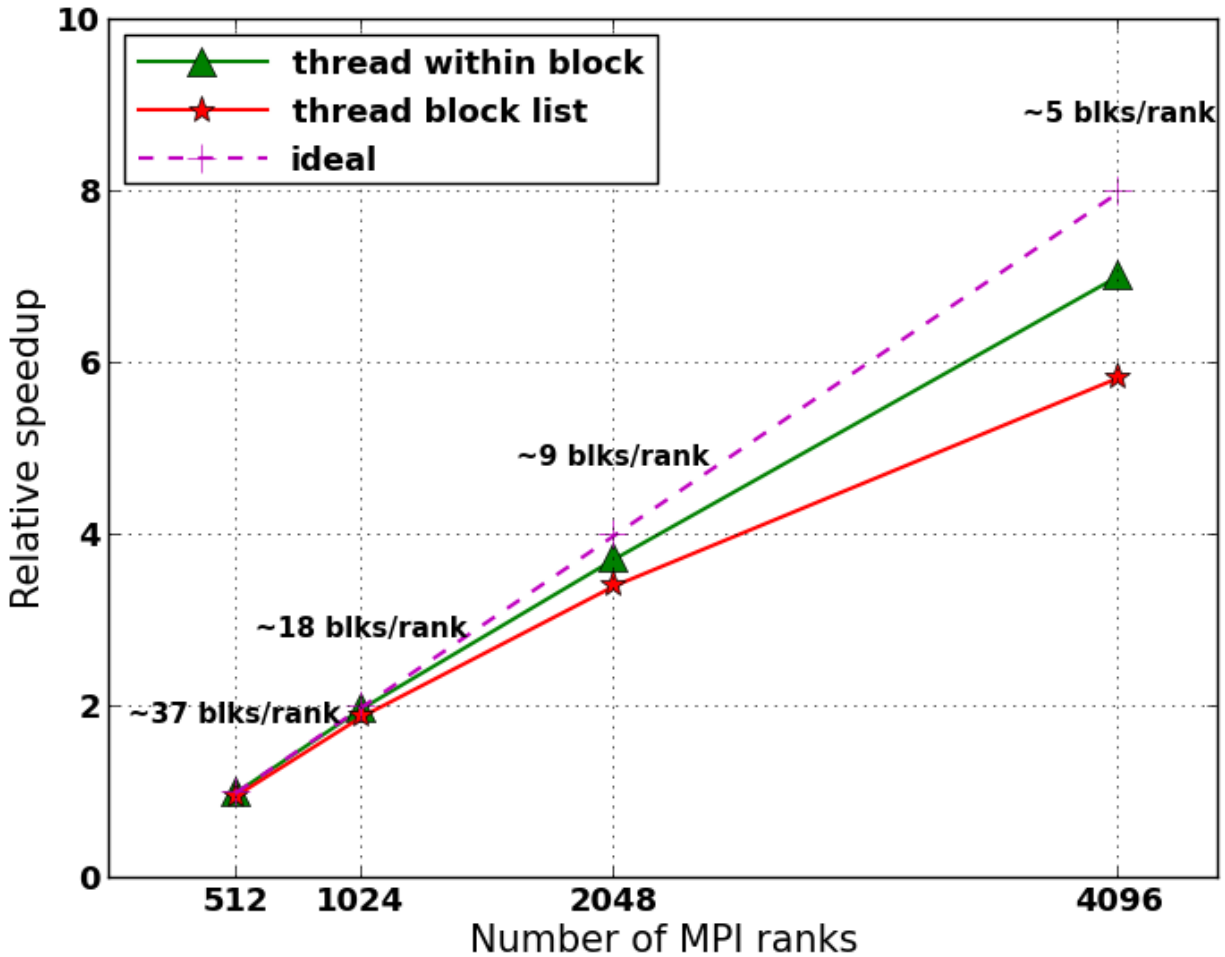


Figure 6: Speedup in a fixed RTFlame test problem on 32, 64, 128 and 256 nodes of Vesta BG/Q (used 16 MPI ranks per node, 4 OpenMP threads per MPI rank and both FLASH multithreading strategies).

Best FLASH configuration on BG/Q summary

- Best performance: 32 MPI ranks/node, 2 threads/MPI rank
  - Tricky to fit application in 512MB/MPI rank
    - The unsplit hydro solver is more memory hungry than the split hydro solver used for science runs on BG/P in previous years
- Best compromise: 16 MPI ranks/node, 4 threads/MPI rank
  - Comfortable to fit application in 1GB/MPI rank. Buffers can be sized larger to accommodate
    - Rapid refinement of problem
    - Congregation of many tracer particles on some MPI ranks
- Finer-grained threading in FLASH performs better
- BG/Q to BG/P node-to-node ratio of 8.9x (RTFlame) and 7.9x (DDT)

Strong scaling

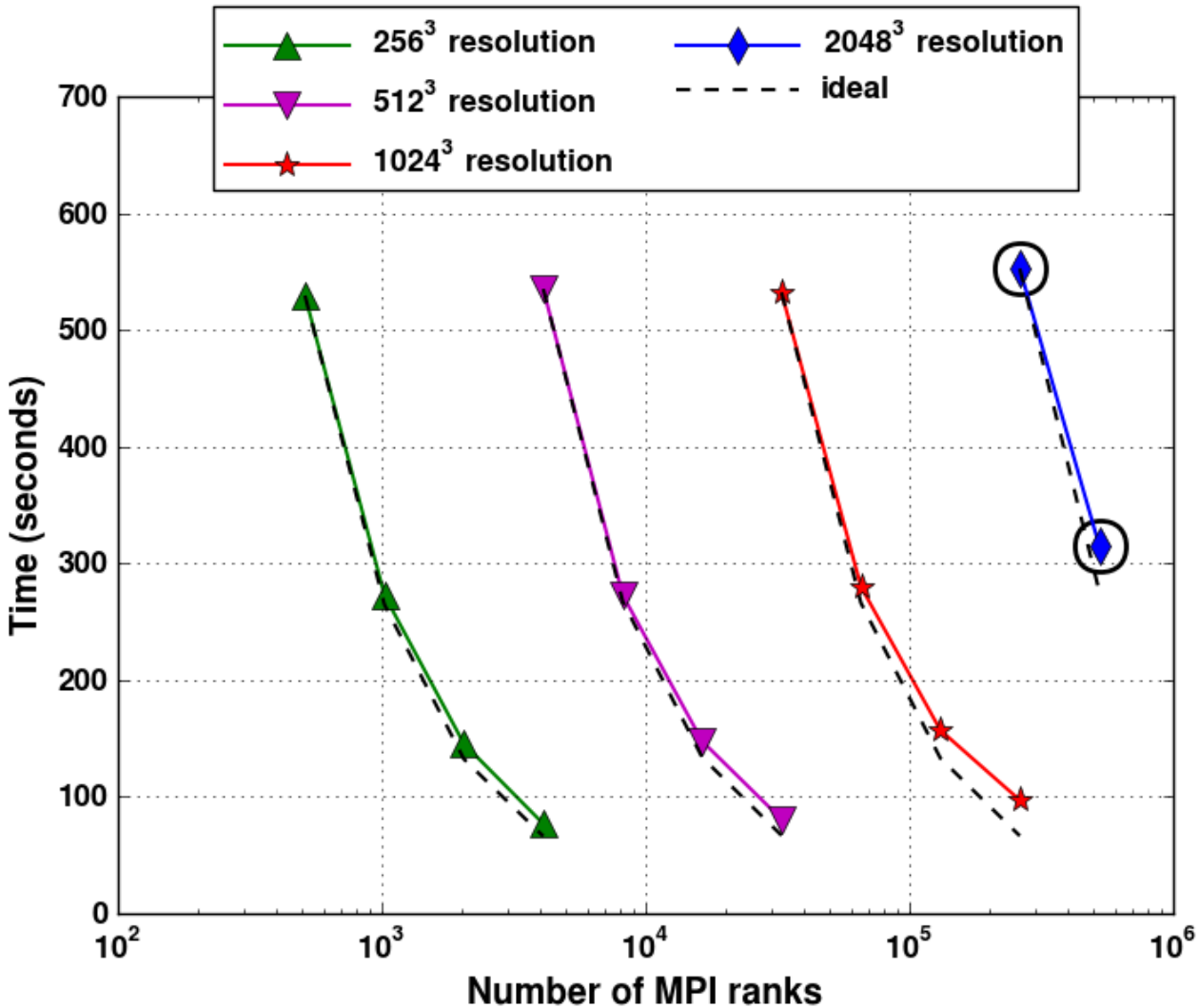


Figure 7: Strong scaling of various resolution RTFlame test problems on Mira BG/Q (used 16 MPI ranks per node, 4 OpenMP threads per MPI rank and fine-grained threading).

Weak scaling

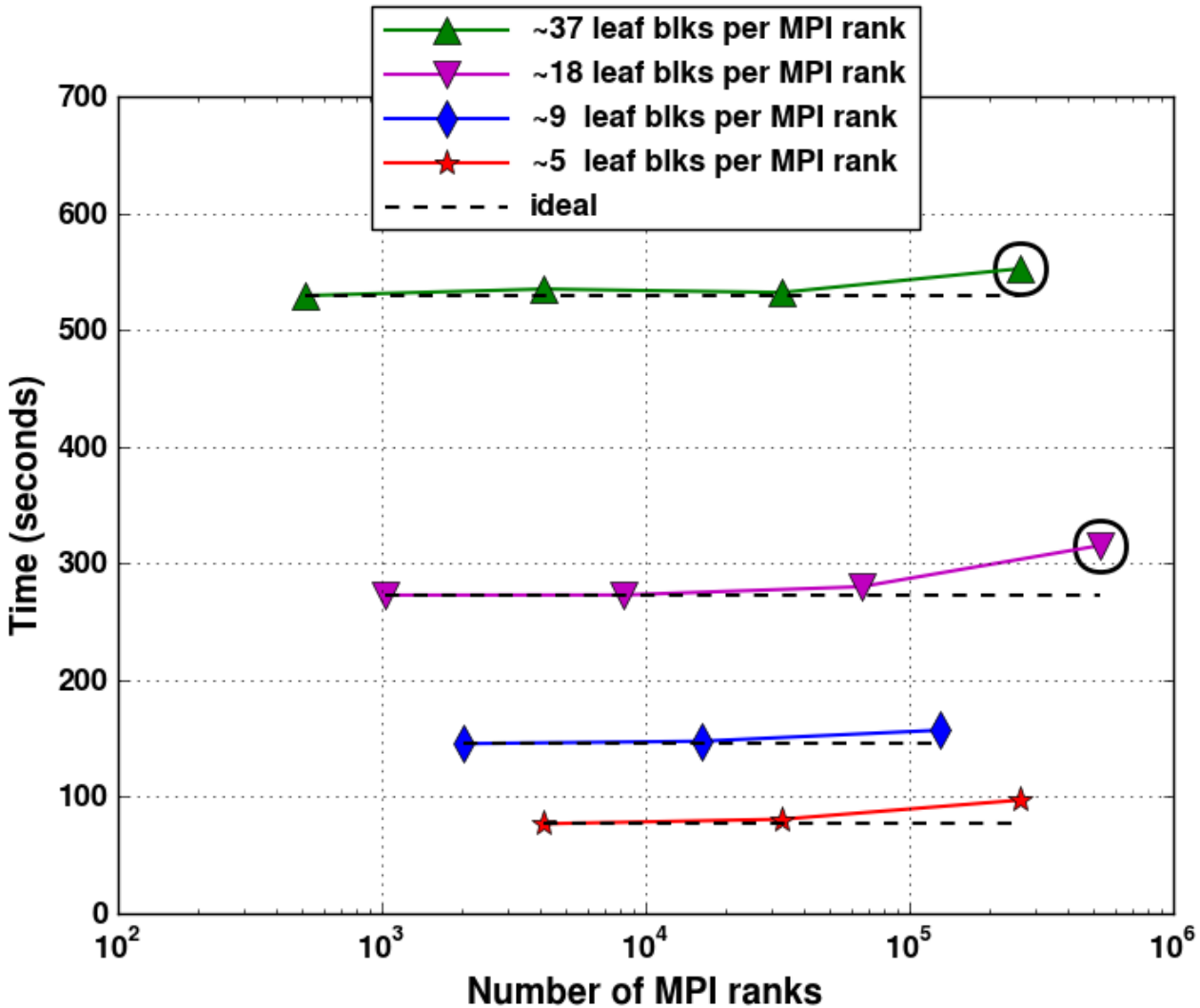


Figure 8: Weak scaling of various resolution RTFlame test problems on Mira BG/Q (used 16 MPI ranks per node, 4 OpenMP threads per MPI rank and fine-grained threading).

Scaling summary

- The scaling tests used up to 32,768 (2^{15}) nodes
  - >2 million-way parallelism at 32,768 nodes (524,288 MPI ranks and 4 OpenMP threads per MPI rank)
- The circled data points in Figures 7 and 8 are from runs with PAMI_ALLREDUCE_REUSE_STORAGE=N and PAMI_ALLTOALL_PREMALLOC=N
- These environmental variables reduced memory usage and allowed 2048^3 resolution runs to succeed
- Good weak scaling: maximum scaling loss of 13.5% when going from 64 to 32,768 nodes
- Production early science runs are using approximately 18 leaf blocks per MPI rank

Hardware counter data

```
================================================================
Hardware counter report for BGQ  - sum for node <0,0,0,0,0>.
cores in use = 16, active threads per core = 4.
================================================================
----------------------------------------------------------------
FLASH_evolution, call count = 1, avg cycles = 364210026210, max cycles = 364213555879 :
  -- Counter values summed over processes on this node ----
0      71127700508   Committed Load Misses
0     996543943304   Committed Cacheable Loads
0      57759693733   L1p miss
0    2370956187483   All XU Instruction Completions
0     827216441627   All AXU Instruction Completions
0    1261426933713   FP Operations Group 1
  -- L2 counters (shared for the node) -----------------
100   569692088884  L2 Hits
100     7121203216  L2 Misses
100     8169110557  L2 lines loaded from main memory
100     6253069690  L2 lines stored to   main memory


Derived metrics for code block "FLASH_evolution" averaged over process(es) on node <0,0,0,0,0>:
Instruction mix:  FPU = 25.87 %,  FXU = 74.13 %
Instructions per cycle completed per core = 0.5488
Per cent of max issue rate per core = 40.69 %
Total weighted GFlops for this node = 5.541
Loads that hit in L1 d-cache =  92.86 %
                  L1P buffer =   1.34 %
                  L2 cache   =   5.08 %
                  DDR        =   0.71 %
DDR traffic for the node: ld = 2.871, st = 2.198, total = 5.069 (Bytes/cycle)
```

Conclusion

- Hybrid MPI+OpenMP FLASH applications are faster than MPI-only FLASH applications on BG/Q
- Good scaling to large processor counts
- Good node-to-node performance advantage over BG/P
  - 8.9x for RTFlame
  - 7.9x for DDT
- The BG/Q platform allows us to run 2048^3 effective resolution simulations
  - Previously not possible on BG/P

Any questions?

Creating a custom Makefile

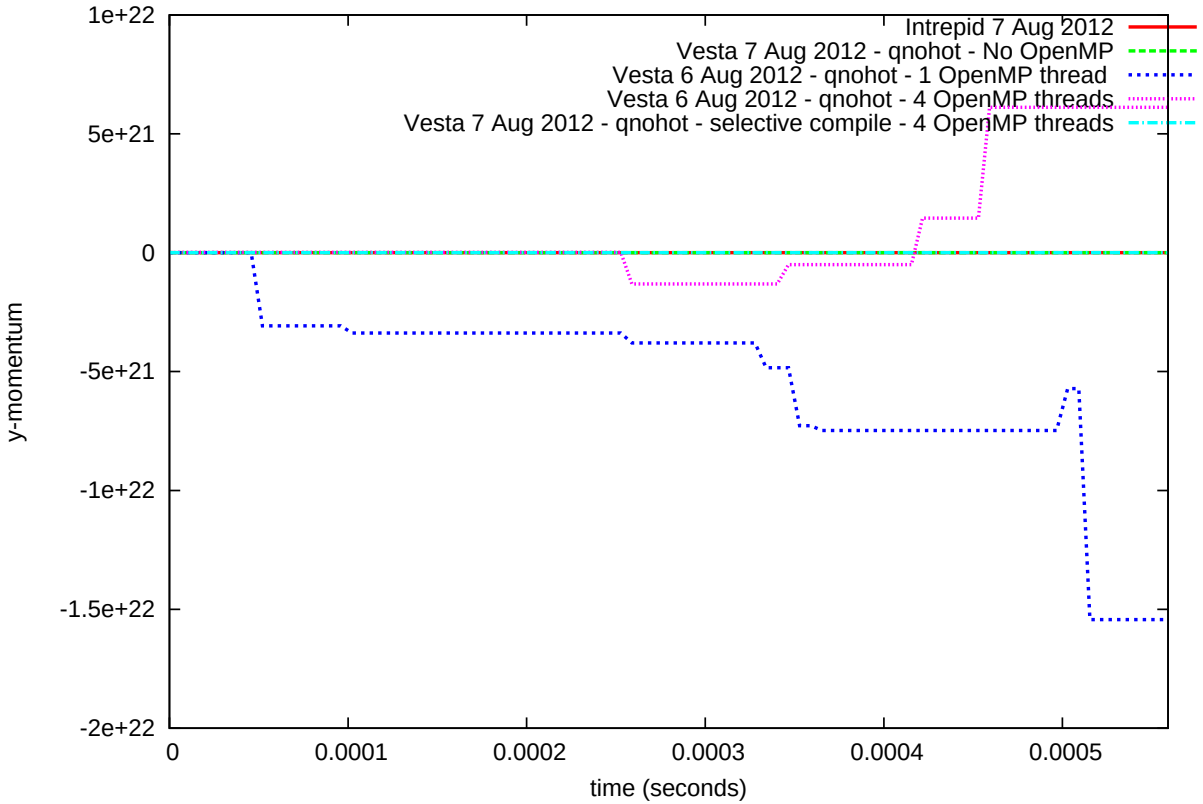


Figure 9: Issues with OpenMP compiler option (-qsmp=omp:noauto).

- "-O3 -qnohot" gave good answers
- "-qsmp=omp:noauto -O3 -qnohot" gave bad answers (even with 1 OpenMP thread)???
- Selective use of the option on files containing OpenMP gave good answers

Finding a safe optimization level

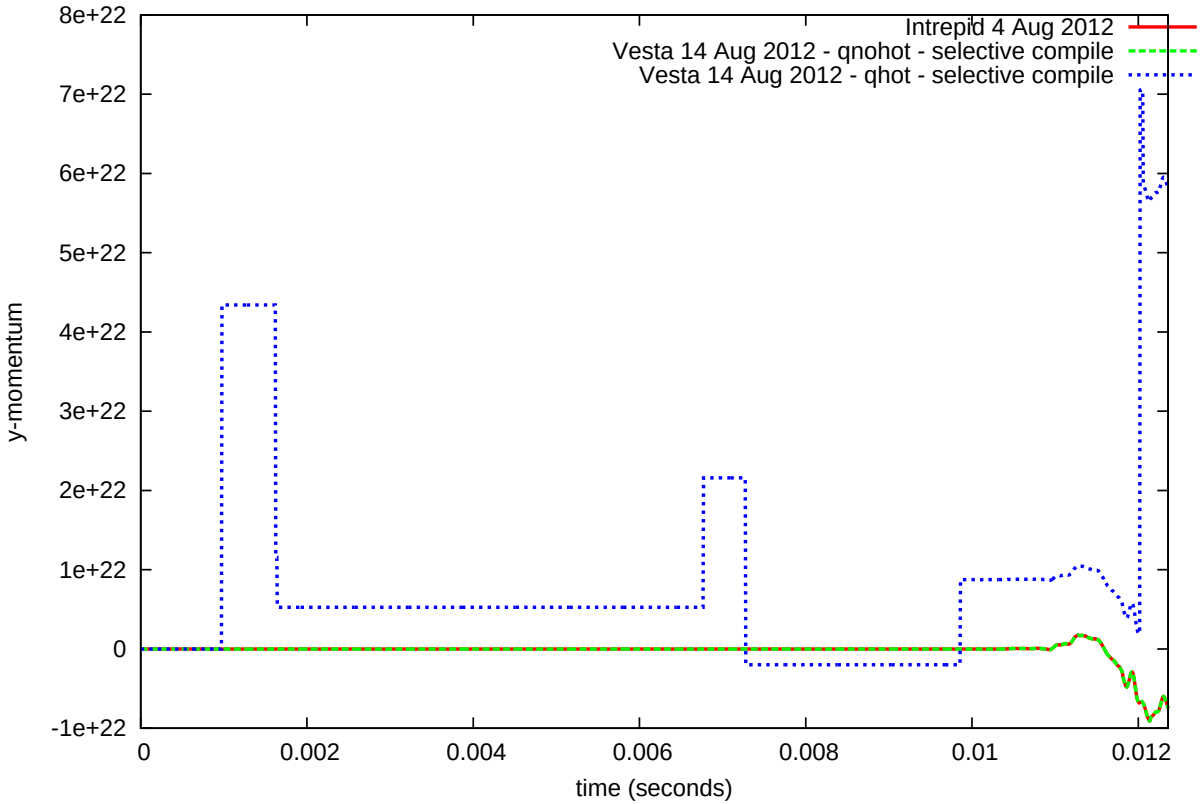


Figure 10: Issues with aggressive compiler option (-qhot).

- Selective "-O3 -qnohot" gave good answers
- Selective "-O3 -qhot" gave bad answers
- Consistent good answers with selective "-O3 -qnohot" in various test problems.

Working around memory usage issue

- Some runs (particularly >= 4096 nodes) crashed during initialization because of out-of-memory errors
- Found huge memory consumption on Mira BG/Q... but surprisingly not on Intrepid BG/P
- Memory wrappers show growth happens in the call-stack of the Paramesh subroutine named `find_surrblks`
  - called once at initialization and once at restart
  - heavy (synchronous) communication
- `mtrace` shows unfreed memory in mpich2 and PAMI memory allocation wrappers - *unhelpful*
- Created an optimized version of `find_surrblks` with less communication. The effect on memory usage is shown in Figure 11

Memory usage issue on BG/Q

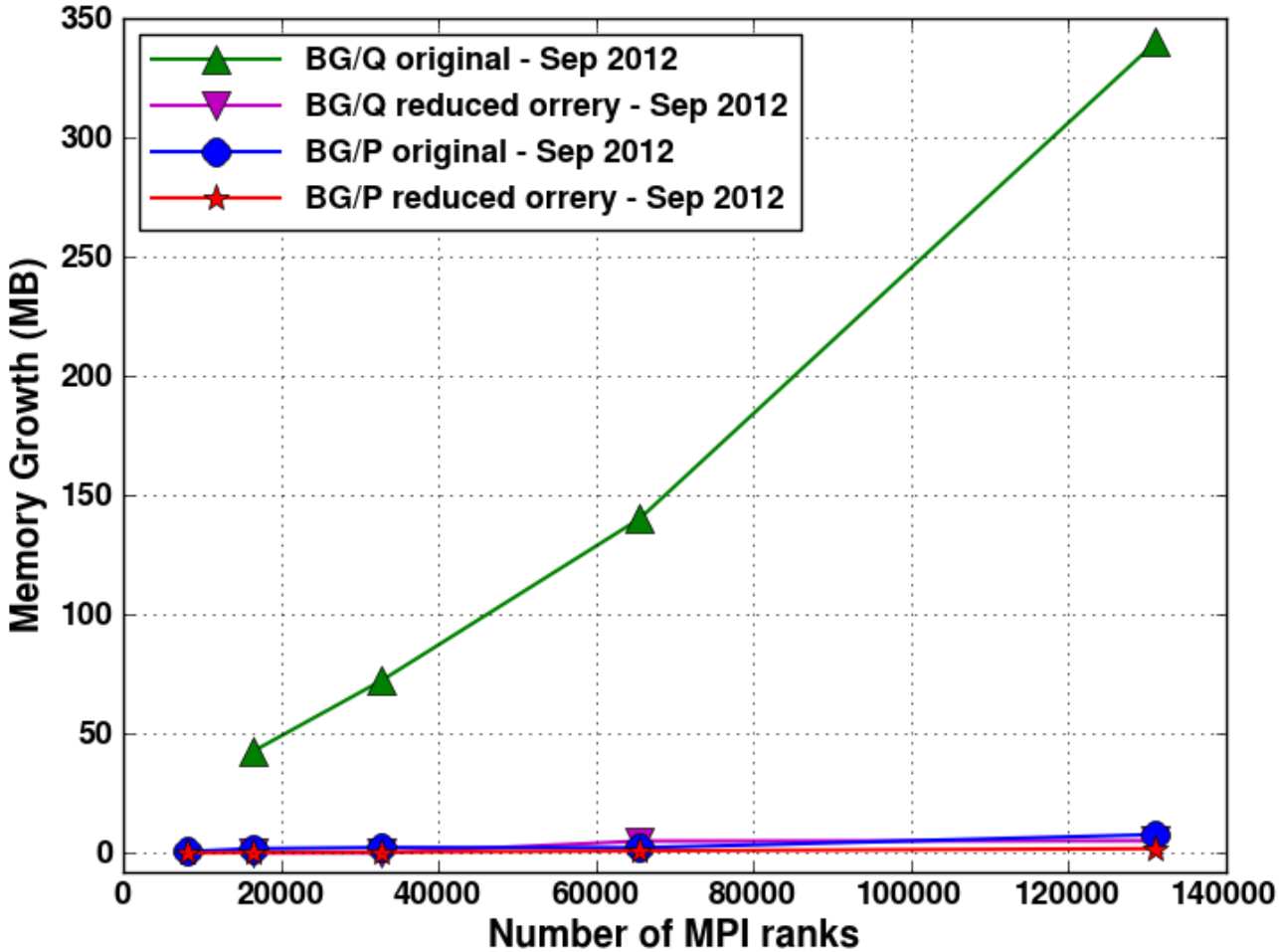


Figure 11: Memory growth after running `find_surrblks` subroutine on Intrepid BG/P (4 MPI ranks per node) and Mira BG/Q (16 MPI ranks per node).