Challenges of Computing with FLASH, a Highly Capable Multiphysics Multiscale AMR Code, on Leadership Class Machines

Anshu Dubey, Chris Daley, George Jordan, Paul Rich, Don Lamb, Rob Latham, Katherine Riley, Dean Townsley and Klaus Weide
DOE NNSA ASC Flash Center
University of Chicago

SciComp16
May 10, 2010
The FLASH Code Contributors

- **Current Group:**
  - Chris Daley, Shravan Gopal, Dongwook Lee, Paul Rich, Klaus Weide, Guohua Xia

- **Other Current Contributors:**
  - Paul Ricker, John Zuhone, Marcos Vanella

- **Past Major Contributors:**
  - Katie Antypas, Alan Calder, Jonathan Dursi, Robert Fisher, Murali Ganapathy, Timur Linde, Bronson Messer, Kevin Olson, Tomek Plewa, Lynn Reid, Katherine Riley, Andrew Siegel, Dan Sheeler, Frank Timmes, Dean Townsley, Natalia Vladimirova, Greg Weirs, Mike Zingale
Outline

- FLASH
- Description of Science Applications
- Early Use Challenges
- Changes to FLASH
- What We Have Learned
Development of the FLASH code was made possible by nearly a decade of funding and support by NNSA ASC Academic Strategic Alliance Program.
More than 600 scientists have been co-authors on papers published using the FLASH code.
FLASH Basics

- An application code, composed of units/modules. Particular modules are set up together to run different physics problems.
- Fortran, C, Python, …
  - More than 500,000* lines of code, 75% code, 25% comments
- Very portable, scales to tens of thousand processors

Capabilities

- Infrastructure
  - Configuration (setup)
  - Mesh Management
  - Parallel I/O
  - Monitoring
    - Performance and progress
  - Verification
    - FlashTest
      - Unit and regression testing
- Physics
  - Hydrodynamics, MHD, RHD
  - Equation of State
  - Nuclear Physics and other Source Terms
  - Gravity
  - Particles, active and passive
  - Material Properties
  - Cosmology
I/O In FLASH

- Distribution comes with support for HDF5 and PnetCDF libraries and basic support for direct binary I/O
  - Direct binary format is for “all else failed” situation only
- Both libraries are
  - Portable
  - Use MPI-IO mappings
  - Are self describing and translate data between systems
  - The libraries can be used interchangeably in FLASH
- Can group writes into N files, extreme cases are N=1 file and N=NPROCS

- Large Files:
  - Checkpoint files
    - Save full state of the simulation
- Plot files
  - Data for analysis
- Smaller Files:
  - Dat files
    - Integrated quantities, output in serial files
  - Log files
    - Status report of the run and logging of important run specific information

- Input files
  - Some simulations need to read files for initialization or table look-up purposes
Running FLASH on largest machines presents some special challenges:

**The Machines**
- Cutting edge == less well tested systems software
- Highly specialized hardware
- A new generation every few years
- Parallel I/O always a challenge
- Availability is limited
- Stress testing the code before big runs is extremely challenging (or impossible)

**The Code**
- More than half a million lines
- Multiphysics with AMR
- Public code with reasonably large user base
- Must run on multiple platforms
- Must be efficient on most platforms
General Experience on New Platforms

- FLASH has historically walked into almost every hardware or software fault in the high end systems.

- Very intolerant of bad data
  - leads to unphysical situations, causes crashes
  - very demanding of hardware and system software

- I/O data—same order of magnitude as system memory
  - Checkpointing
    - Simulation state at full precision
  - Analysis
    - many state variables (large) relatively frequently
    - particles data (small) very frequently
Applications of Interest: specs

**RTFlame**
- Physics
  - Hydrodynamics
  - Gravity (constant)
  - Flame
  - Particles
  - Eos (helmholtz)
- Infrastructure
  - Infrequent regridding
  - Number of blocks grow modestly

**GCD**
- Physics
  - Hydrodynamics
  - Gravity (Newtonian)
  - Flame and Burn
  - Particles
  - Eos (helmholtz)
- Infrastructure
  - Frequent regridding
  - Number of blocks grow significantly
  - Frequent particle I/O
Applications of Interest: RTFlame

The Center for Astrophysical Thermonuclear Flashes

Simulation of Buoyancy-Driven Turbulent Nuclear Burning for a Froude Number of 0.010

This work was supported in part at the University of Chicago by the DOE NNSA ASC ASAP and by the NSF. This work also used computational resources at LBNL NERSC awarded under the INCITE program, which is supported by the DOE Office of Science.

An Advanced Simulation and Computation (ASC) Academic Strategic Alliances Program (ASAP) Center at The University of Chicago
Application of Interest: GCD
Design of a Scaling Test

- Developed a test to examine/demonstrate scaling on HPC platforms

- Initially designed for INCITE proposal
  - Found to be useful in quickly checking expected performance on new platforms

- Primary purpose: verify weak scaling
  - For production runs weak scaling is of primary interest
  - Strong scaling of limited interest – queue characteristics
  - Problems scale as fourth order of resolution change
  - Use important features of science simulations

Has been used on Seaborg and Franklin at NERSC, Later ALCF BG/P, Jaguar etc.
More on Scaling Test

- Test build includes most important features of GCD production runs:
  - AMR (with regridding), hydro, gravity, flame, tracer particles
  - I/O code unit omitted

- For FLASH workload/proc = blocks/proc
  - Increase number of procs, keeping workload / proc the same
  - Carefully select a set of initial conditions so that #procs / proc ≈ 64, 128, 256, …, 16384, …
  - No noticeable loss in performance initially
  - pretty bad for 4000 and more procs on most machines, beyond 8000 on Intrepid!
Revelations of the Scaling Test

- Some slowdown caused by code “improvements”
  - unexpected side effects of code cleanup
  - For example: unnecessary EOS calls on guard cells

- Most of poor scaling accounted for by regridding
  - later identified as “orerry problem” (PERI collaboration)

- Otherwise good weak scaling
  - RTFlame, which has infrequent regridding scaled to full machine on Intrepid
Early Use Experience

- BG/P at Argonne National Laboratory
- FLASH part of the acceptance suite
- Application ran very early in the lifecycle of the machine

  - system and application problems ran into each other
    - insertion of extra barriers reduced frequency of crashes
  - Hangs happened as often as aborts,
    - aborts reveal more useable information
  - Non deterministic failures
    - Changing partition made a difference
    - while watching our run, we’d see others get to the same partition and sometimes fail
    - gave indication of chip problems
Early Use : I/O

- I/O (performance AND correct behavior) has usually been a major problem when first running large simulations on large new machines. This was also true for BG/P at Argonne.
- Parallel filesystems, libraries, hardware all in a state of flux
  - Filesystems that slow down to a crawl, for mysterious reasons
  - I/O failures caused by unavailability of locks
    - Mysterious runtime flags and modifiers need to be set to get reasonable behavior. Turns out we don’t need all that locking anyway.
  - Libraries may work correctly in “vanilla” mode, but failed when we tried non-default (but sensible!) settings for optimization
    - Example: “collective” vs “independent” mode of HDF5 I/O
  - Several bugs fixed (or worked around) by vendor, ALCF support
  - Several workarounds by us, for lacking support in libraries
    - Example: library handling of single -> double conversion
  - We developed new file format for binary output to take better advantage of data locality and buffering
- We assumed memory shortage when IO problems occurred, sometimes wrongly.
Can we improve the performance of FLASH I/O?

Motivating factors:
- Up to 35% total runtime spent in I/O for production runs!
- Preparation for peta-scale computing.

Questions:
- Performance impact of collective I/O?
  - Silent error, data corruption
  - Fixed in Romio, also fixed memory leak
- FLASH AMR data restriction – reuse of metadata
- Modification of FLASH file format
  - Write all variables into the same 5D dataset, MPI datatype
- Performance bottleneck in Parallel-netcdf library?
Collective I/O Importance

Time for one checkpoint / one plotfile during Sedov weak scaling experiments
Average of 32 leaf blocks (16^3 cells) / process - used HDF5 I/O library
(lrefine_min=lrefine_max=5 fixed and nblock[xyz] varied).
Grid Data File Format

Time to write one checkpoint file during Sedov weak scaling experiments
Average of 32 leaf blocks (16^3 cells) / process
(Irefine_min=Irefine_max=5 fixed and nblock[xyz] varied).

The ASC/Alliance Center for Astrophysical Thermonuclear Flashes
The University of Chicago
Data Restriction

Time for one restriction during Sedov weak scaling experiments
Average of 32 leaf blocks (16^3 cells) / process
(lrefine_min=lrefine_max=5 fixed and nblock[xyz] varied).

The ASC/Alliance Center for Astrophysical Thermonuclear Flashes
The University of Chicago
Memory Optimization

- We repeatedly ran into problems with insufficient memory / PE:
  - used to have at least 4 GB / proc on local cluster
    - Our algorithms like to have this much memory!
  - Problems adapting production runs (and tests) to 2 GB, 1 GB
    - I/O libraries like to allocate a lot of memory for buffers…
    - → a lot of swapping! (where OS memory system allows this)
  - More severe problems adapting runs to .5 GB for BG/P
    - Initially we could only have a few AMR grid blocks (16^3 cells each) per proc, otherwise code would fail in mysterious ways
    - Suspected memory fragmentation (No)
    - Careful analysis of memory usage for large buffers
    - Free large allocate buffers as soon as possible
    - Other memory efficiency improvements
    - Number of blocks per proc we can use now increased from 5..15 to 50..60

- Memory problems often connected with I/O problems
The underlying AMR package, PARAMESH, in recent versions introduced a “Digital Orrery” algorithm for updating some block neighbor information.

- It works like a rotating restaurant.
- Eventually you get to see everything, but it may take a while.
- (It’s not even obvious that it’s there!)
The “Digital Orrery” hands meta-information about blocks around until every PE has seen it.

- called once after each regridding
  - function is important but auxiliary, non-obvious
- linear scaling $\sim O(n_{proc})$
  - was not noticeable up to $\sim 1000 \text{ procs (procs == PEs)}$
  - pretty bad for 4000 and more procs!

- Problem showed up in timers
  - FLASH timers, a very useful feature for coarse timing
  - Confirmed by Tau profiling
  - PERI collaboration analyzed problem, suggested solution
Orerry Elimination III

Inclusive time spent in a single call to amr_refine_derefine. Weak scaling test. Experiments run on Intrepid (BG/P).

- standard surr_blks construction (orery)
- custom surr_blks construction

Time (seconds)

Number of cores

The ASC/Alliance Center for Astrophysical Thermonuclear Flashes
The University of Chicago
Performance in a Production Run

**time for one step per block**

- **unoptimized**
- **optimized**

**Number of Processors**

- 2048
- 4096
- 8192
- 16384
- 32768
- 65536

**time in seconds**

The ASC/Alliance Center for Astrophysical Thermonuclear Flashes
The University of Chicago
time/blk for one step, 8192 processors

Average blocks per processors
Summary - Lessons Learned

- If your code scales nicely up to N cores, don’t assume it will automatically scale up to M (> N) cores
- Expect the Unexpected
- Know Your Code
  - Corollary: You will (if you really want to scale up)
  - Code improvements often a side effect of scaling
- Know Your Physics (or other application domain)
  - Essential for making good decisions about simulation modifications
- Many things are within our control - but many others aren’t
  - usability of new machines (frequency of failures, …)
  - Availability of libraries
  - Memory constraints, I/O difficulties
  - Documentation, user support may or may not be there
- You get what you get and you don’t get upset