On Using Linux Kernel Huge Pages with Flash, an Astrophysical Simulation Code

Alan Calder

Catherine Feldman, Eva Siegmann, John Dey, Tony Curtis, Smeet Chheda, and Robert Harrison

Institute for Advanced Computational Science

Stony Brook University, NY, USA

EAHPC-2022, September 6, 2022

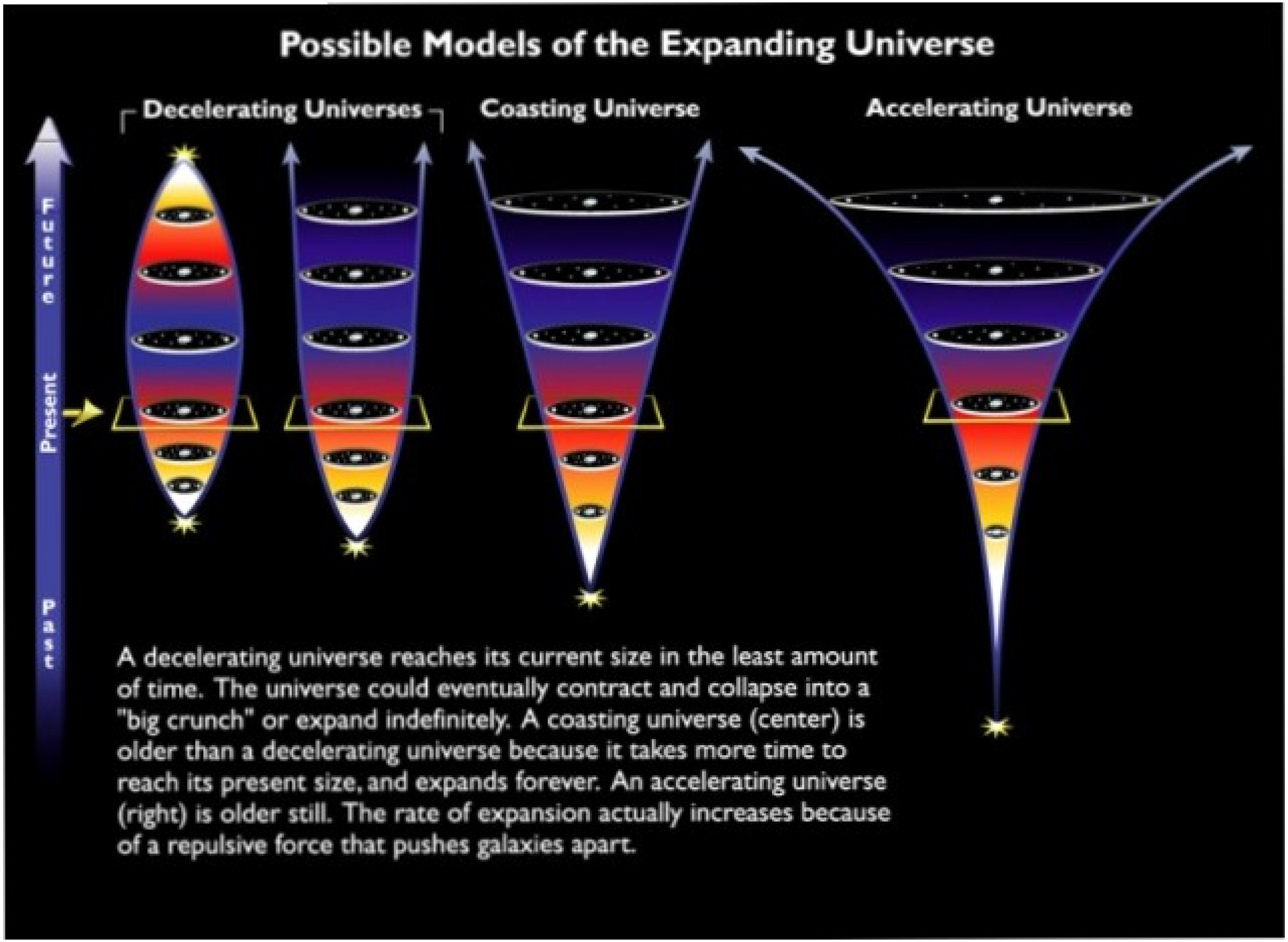
Outline

- Thermonuclear Supernovae
- The Flash Code
- Ookami- our A64-FX machine
- Approach to instrumenting the code and testing huge pages
- Results
- A look to the future

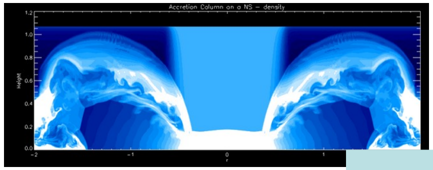
Thermonuclear (Type Ia) Supernovae

- Thermonuclear Explosion in one or more White Dwarf Stars
- Very bright events
- Calibrated via empirical relationship to be “standard candles”
- Setting not completely known

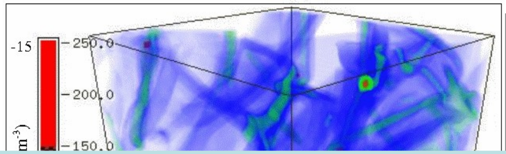
Accelerating Expansion of the Universe

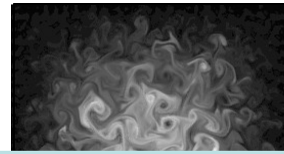


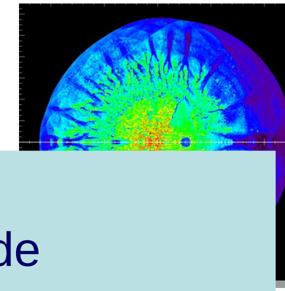
The FLASH Code

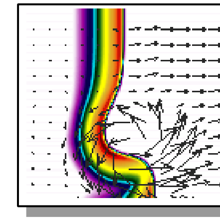


Shortly: Relativistic accretion on

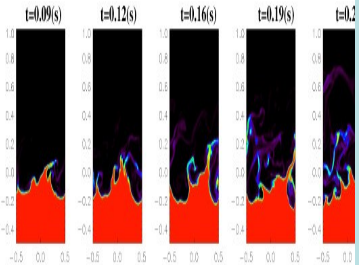








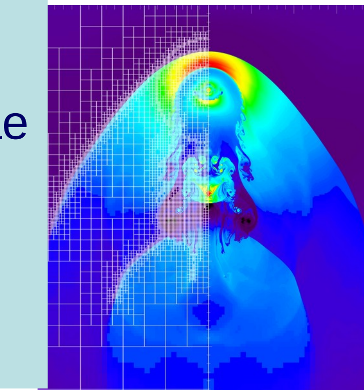
Flame-vortex interactions



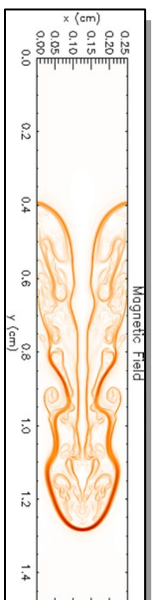
Wave breaking on white dwa

The FLASH code

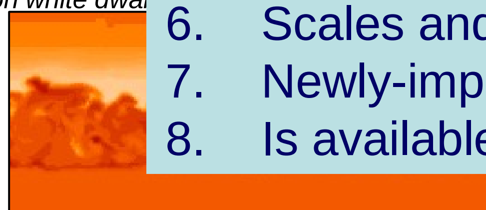
1. Parallel, adaptive-mesh simulation code
2. Written primarily in modern Fortran
3. Designed for compressible reactive flows
4. Features flame model and physics for supernovae
5. Won SC2000 Gordon Bell Prize
6. Scales and performs well- suitable for 3-d
7. Newly-implemented modules for HED physics
8. Is available to the community



Intracluster interactions



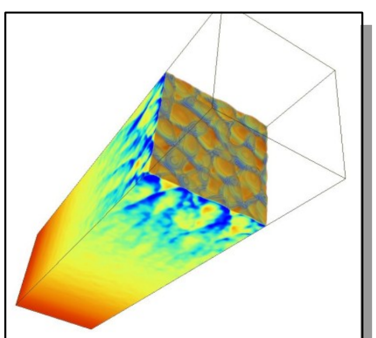
Magnetic Rayleigh-Taylor



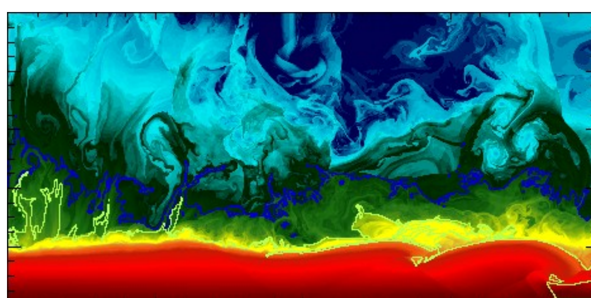
Nova outbursts on white dwarfs

Laser-driven shock instabilities

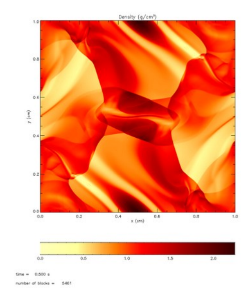
Rayleigh-Taylor instability



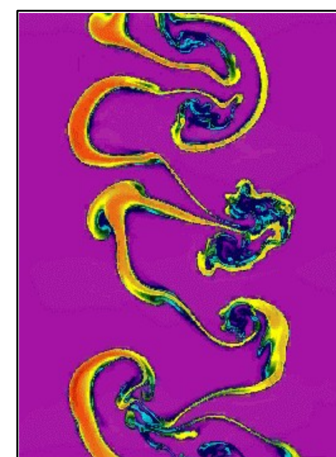
Cellular detonation



Helium burning on neutron stars

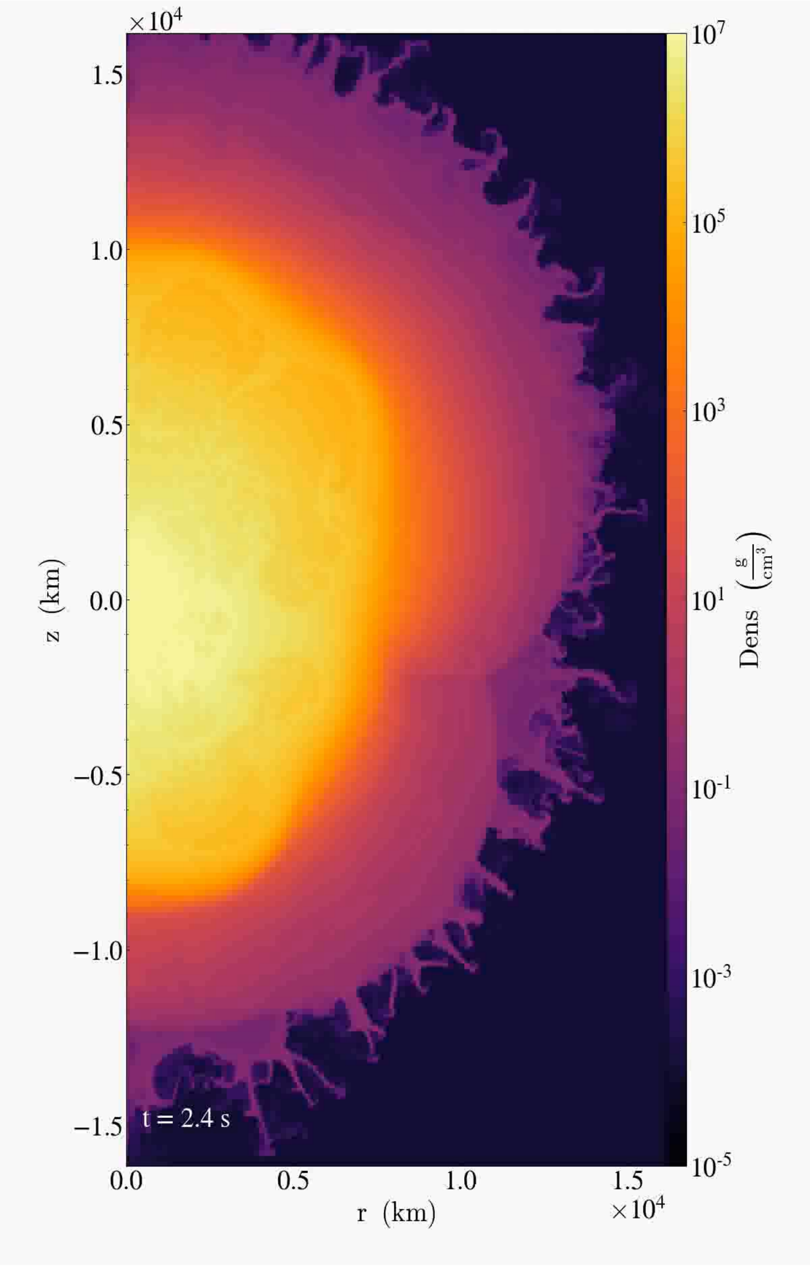


Orszag/Tang MHD vortex



Richtmyer-Meshkov instability

Type Iax Supernova



Flash Tests

- Previous simulation was from a 2-d pilot study.
- Eventually will run full 3-d simulations
- → Test 3-d hydrodynamics module
- Performed preliminary performance analysis with ARM MAP and found the code spends time in the Equation of State (EOS, degenerate electron/positron plasma)
- → Instrument EOS
- Thus two tests
  - 3-d hydro (Sedov explosion simulation)
  - EOS (2-d supernova simulation)

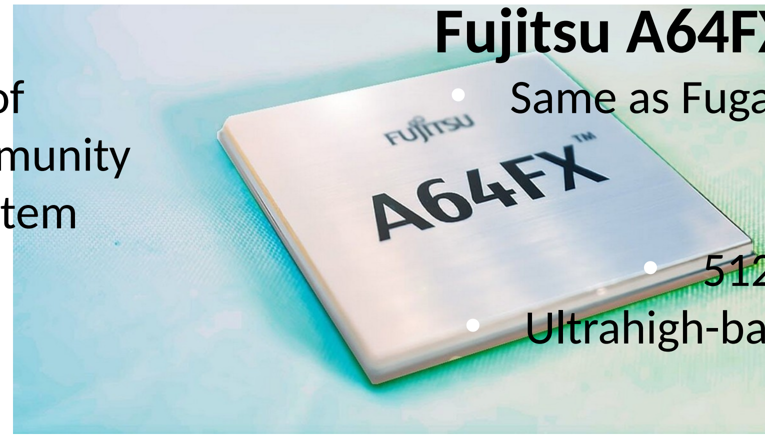
https://iopscience.iop.org/article/10.1086/317361

OOKAMI

A NEW PATH TO EXASCALE COMPUTING

Experimental testbed

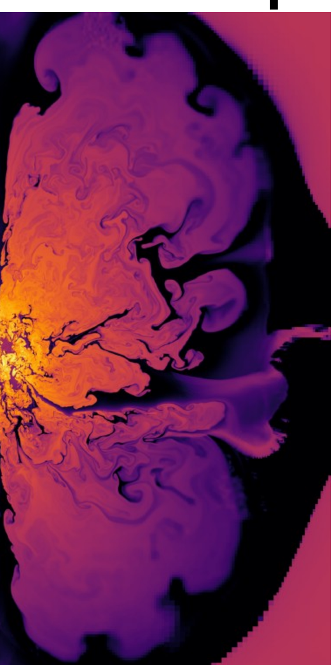
- First machine of its kind outside of Japan freely available to the community
- HPE (formerly Cray) Apollo 80 system
- Fall 2022: NSF ACCESS site



Fujitsu A64FX processors

- Same as Fugaku, #1 on Top500
- ARM-based
- Multi-core
- 512-bit SIMD-vector
- Ultrahigh-bandwidth memor

Current projects include:



- Type Ia supernovae
- Deep learning & AI
- Molecular dynamics
- Cancer cell simulations
- WarpX – E&M particle-in-cell
- Climate and weather modeling
- Plasma wakefield simulations
- Oceanic mesoscale mixing
- Black hole accretion
- Thermoelectric materials

Performance Application Programming Interface (PAPI)

- Interface and methodology for use of the performance counter hardware found in most major microprocessors
- Enables seeing, in near real time, the relation between processor performance and processor events
- Also provides access to a collection of components that expose performance measurement opportunities across the hardware and software stack

https://icl.utk.edu/papi/

PAPI Performance Module

Inspired by Danny Vanpoucke's OOP Timer Example

- Fortran modules for the region and the invocation of the PAPI calls.

- **region_mod** stores the name of the region of the code of interest

- **perf_mod** creates the performance object
  - Initializer creates the object and allocates member variables and calls the PAPI begin function
  - Finalizer calls the PAPI end function and deallocates

- Using the Fortran **block** construct allows instantiation of a performance object at any point in a given routine.

https://dannyvanpoucke.be/oop-fortran-tut4-en/#codesyntax_1

PAPI Performance Module

```
program hellotest

  use region_mod, only: region
  use perf_mod, only: pperf, finalizer

  implicit none

  type(pperf) :: perfmon1

  region = 'Greeting'
  perfmon1 = pperf()

  write(*,*)'Hello, EAHPC pals!'

  call finalizer(perfmon1)
  stop

end program hellotest
```

import modules

Create profiler object and set up region

Code you want to profile

Deallocate profiler object

PAPI Performance Module

- Worked well with the GNU compiler v. 11.2.0
- Slight modification with Cray compiler v. 10.0.3 (sve and nosve)
- Did not work with Fujitsu compiler v. 4.5
- Problem with the calling of the finalizer (early in some cases) Attributed to imperfect implementation of the standard.

→ instrumented with direct calls to PAPI routines

https://stackoverflow.com/questions/19485666/fortran-final-routine-calls-itself-before-variable-goes-out-of-scope

https://stackoverflow.com/questions/59985499/are-fortrans-final-subroutines-reliable-enough-for-practical-use

Huge Pages

- Feature integrated into Linux kernel 2.6
- Allow OS to support memory pages greater than the default (usually 4 KB)
- Can improve performance by reducing resources required to access page table entries
- HugePage sizes vary from 2 MB to 256 MB depending on kernel version and the hardware.
- Note- transparent huge pages disabled.

https://docs.oracle.com/database/121/UNXAR/appi_vlm.htm#UNXAR391

FLASH Data Structure

- Flash uses the PARAMESH Adaptive Mesh Library
- Block-structured 16 X 16 X 16 zones per 3-d block
- Principal data in container **unk**

unk(nvar, il_bnd:iu_bnd, jl_bnd:ju_bnd, kl_bnd:ku_bnd, maxblocks)

nvar = number of variables

il_bnd:iu_bnd, jl_bnd:ju_bnd, kl_bnd:ku_bnd = x, y, z zone limits

maxblocks = number of blocks

PARAMESH designed for loops using data from blocks in column-major Fortran

→ stride in memory for accessing variables in different zones or blocks

→ investigate reducing cache misses/improving performance with Huge Pages

Dedicated Node

- Installed Fujitsu Compiler
  - Required kernel boot time parameters to be set:
    - hugepagesz=2M hugepagesz=512M default_hugepagesz=2M
  - Setting of kernel parameter in /etc/sysctl.d/98-fujitsucompilersettings.conf
    - kernel.perf_event_paranoid=1
- Installed libhugetlbfs-utils
- Specialized unix group created for empowered end user use: 'hugetlb_shm_group'
  - Group members added: **hugetlb_shm_group:*:1234:esiegmann,acalder,jodey,arcurtis,rharrison**
- Instructions were to change settings in /sys/kernel/mm/transparent_hugepage/enabled
  - Enable transparent huge pages by setting file contents to “[always] madvise never” (**echo always > /sys/kernel/mm/transparent_hugepage/enabled**)
  - Disable transparent huge pages by setting file contents to “always madvise [never]” (**echo never > /sys/kernel/mm/transparent_hugepage/enabled**)

https://www.percona.com/blog/2019/03/06/settling-the-myth-of-transparent-hugepages-for-databases/

https://paolozaino.wordpress.com/2016/10/02/how-to-force-any-linux-application-to-use-hugepages-without-modifying-the-source-code/

Huge Pages Testing

Monitored /proc/meminfo for use of huge pages

HugePages_Total:	5396
HugePages_Free:	0
HugePages_Rsvd:	0
HugePages_Surp:	5396
Hugepagesize:	2048 kB
Hugetlb:	11051008 kB

HugePages_Total:	0
HugePages_Free:	0
HugePages_Rsvd:	0
HugePages_Surp:	0
Hugepagesize:	2048 kB
Hugetlb:	0 kB

Ran with variations on

LD_PRELOAD=libhugetlbfs.so HUGETLB_MORECORE=yes hugectl -v -v --bss=2M --heap=2M --shm=2M --data=2M ./xdynamic

Results

- Despite success in getting small test programs to use huge pages, Flash was only able to use huge pages with the Fujitsu compiler.
- A test Fortran program with dynamically allocated arrays was able to use huge pages with the GNU complier
- Flash’s main container, **unk**, is dynamically allocated, but …
- We were able to compare with and without huge pages for the Fujitsu compiler
- Lesson learned- must explicitly turn off huge pages if not wanted!

Results (Supernova EOS)

With	Without
seconds in monitoring period : 333.150	seconds in monitoring period : 339.032
number of evolution steps : 50	number of evolution steps : 50

With

EOS:
- HW cycles: 117330375433
- Seconds: 6.52e+01
- SVE instructions per cycle: 0.51
- Main memory bandwidth (Gbyte/s): 4.45
- DTLB misses/s: 1103732.44

Without

EOS:
- HW cycles: 125376693054
- Seconds: 6.97e+01
- SVE instructions per cycle: 0.47
- Main memory bandwidth (Gbyte/s): 4.19
- DTLB misses/s: 23429655.21

Results (3-d hydro)

With	Without
seconds in monitoring period : 1176.312	seconds in monitoring period : 1203.616
number of evolution steps : 200	number of evolution steps : 200

With

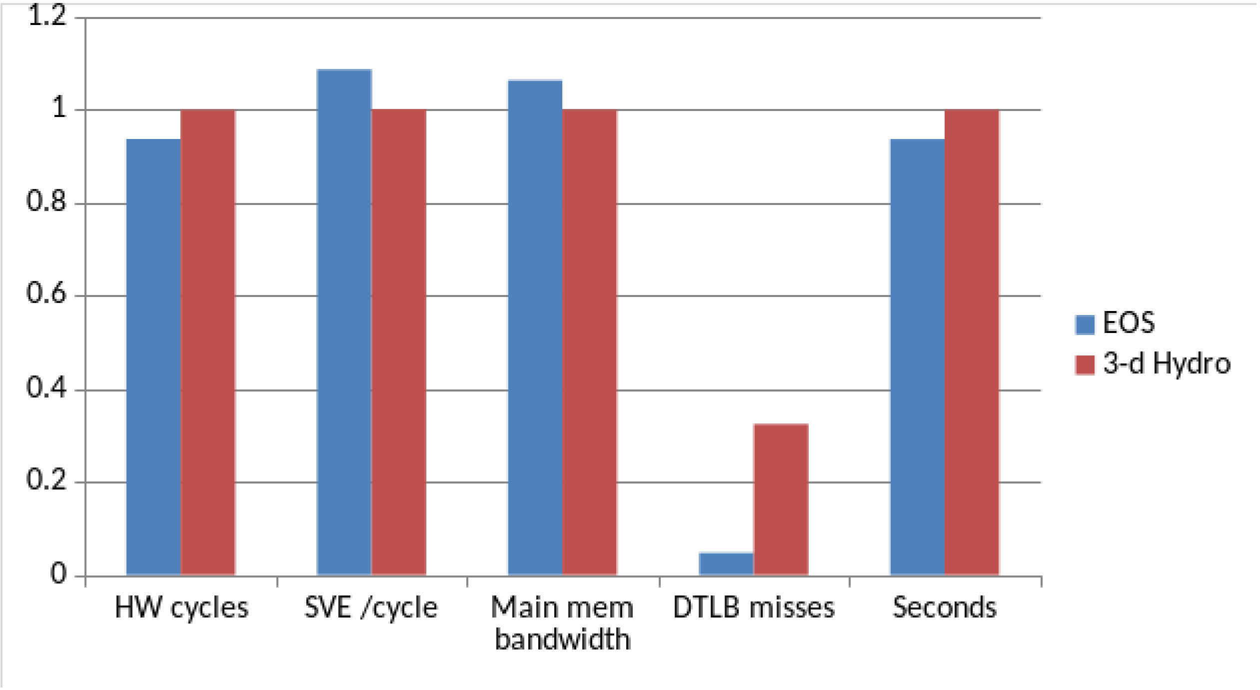
hydro:
- HW cycles: 125376693054
- Seconds: 6.69e+02
- SVE instructions per cycle: 0.11
- Main memory bandwidth (Gbyte/s): 10.09
- DTLB misses/s: 783685.48

Without

hydro:
- HW cycles: 1206574780068
- Seconds: 6.70e+02
- SVE instructions per cycle: 0.11
- Main memory bandwidth (Gbyte/s): 10.10
- DTLB misses/s: 2420376.90

Results

Ratios of quantities with hugepages : without hugepages



Huge pages drastically decreases the number of DTLB misses for both the EOS and 3-d hydro tests. Other measures change slightly, including the simulation time.

Summary

- Flash was only able to use huge pages with the Fujitsu compiler.
- For both tests, we saw a significant reduction in TLB misses
- Simulation time did not change much.
- The dedicated node did not seem to work better than any node with the Fujitsu compiler on it.

Future Work

- Work in progress and we are still learning

- Help from Jens Domke at RIKEN:

With respect to the hugepage issues, I heard from some other group that you can try to force it with this linker flag:
-Wl,-T/opt/FJSVxos/mmm/util/bss-2mb.lds
-L/opt/FJSVxos/mmm/lib64 -lmpg

- Ookami upgraded from CentOS 8.1 with kernel 4.18.0-147.el8.aarch64 to Rocky Linux 8.4 with kernel 4.18.0-305.25.1.el8_4_aarch64

- Repeating study with other compilers as well.