# Lessons Learned

## An In-Depth Look at Running FLASH on Ookami

Presented by Catherine Feldman, and Benjamin Michalowicz, and Alan Calder
Institute for Advanced Computational Science, Stony Brook University




# Ookami Cluster

## Introduction

New, **experimental** supercomputer here at Stony Brook University!

Same A64FX processors as Fugaku, world's *fastest* supercomputer






First open system outside of Japan with this architecture

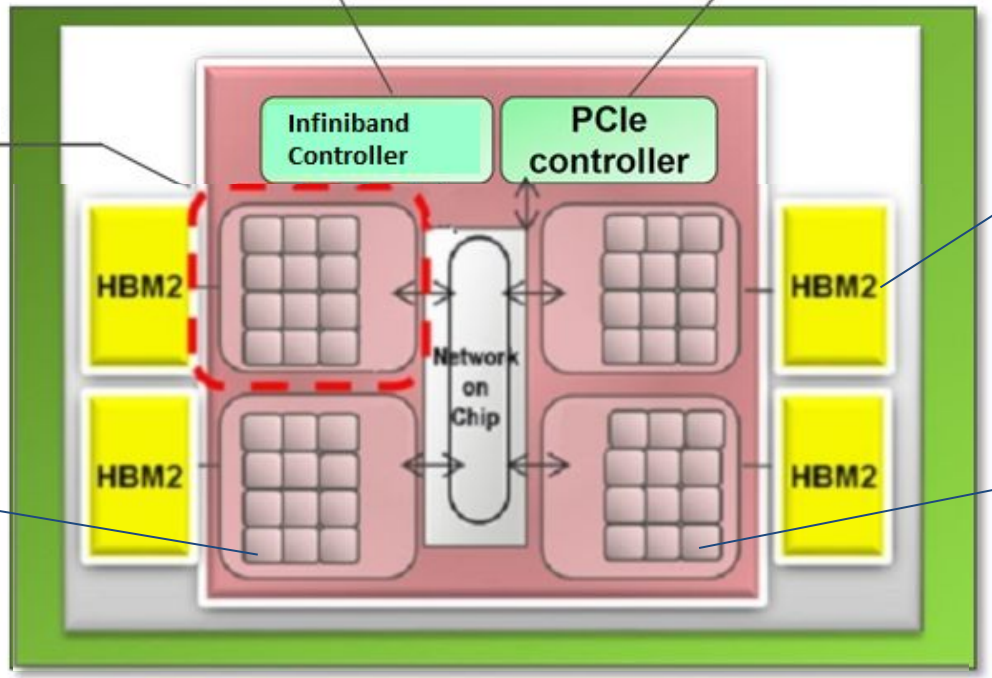
# Ookami Cluster

## A64FX Main Features

**NUMA Node**
- **4 groups of 12 cores (CMG)**
- **13 on Fugaku**
- **Communication time faster between cores in a CMG**

**In lieu of Fugaku's TofuD interconnect**

**I/O**
**PCIe Gen3 16 lanes**




**1024 GB/s = 1 TB/s per node**
**But only 32GB RAM per node**
**Smaller but faster**

**1.8 GHz (2.0 GHz on Fugaku)**
**Power efficient**

**SVE registers for vectorization**

# FLASH
## A bit of history

Founded in 1997 as part of the DOE's ASCI


THE ASTROPHYSICAL JOURNAL SUPPLEMENT SERIES, 131:273–334, 2000 November


FLASH: AN ADAPTIVE MESH HYDRODYNAMICS CODE FOR MODELING ASTROPHYSICAL THERMONUCLEAR FLASHES

B. FRYXELL,[1,2] K. OLSON,[1,2] P. RICKER,[2,3] F. X. TIMMES,[2,3] M. ZINGALE,[2,3] D. Q. LAMB,[1,2,3] P. MACNEICE,[4] R. ROSNER,[1,2,3] J. W. TRURAN,[1,2,3] AND H. TUFO[2,5]

*Received 1999 November 9; accepted 2000 April 13*

ABSTRACT

We report on the completion of the first version of a new-generation simulation code, FLASH. The FLASH code solves the fully compressible, reactive hydrodynamic equations and allows for the use of adaptive mesh refinement. It also contains state-of-the-art modules for the equations of state and thermonuclear reaction networks. The FLASH code was developed to study the problems of nuclear flashes on the surfaces of neutron stars and white dwarfs, as well as in the interior of white dwarfs. We expect, however, that the FLASH code will be useful for solving a wide variety of other problems. This first version of the code has been subjected to a large variety of test cases and is currently being used for production simulations of X-ray bursts, Rayleigh-Taylor and Richtmyer-Meshkov instabilities, and thermonuclear flame fronts. The FLASH code is portable and already runs on a wide variety of massively parallel machines, including some of the largest machines now extant.

*Subject headings:* equation of state — hydrodynamics — methods: numerical — nuclear reactions, nucleosynthesis, abundances — stars: general

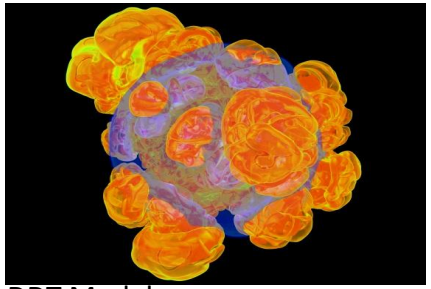


FLASH CENTER

flash.uchicago.edu

# FLASH

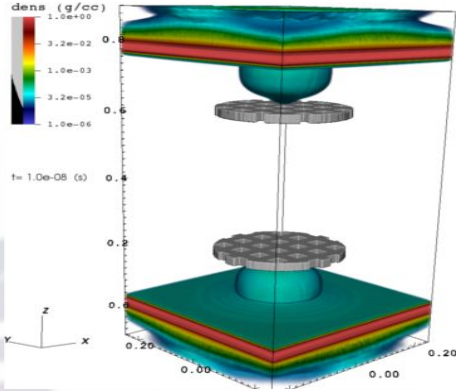
## Multi-scale, Multi-physics applications

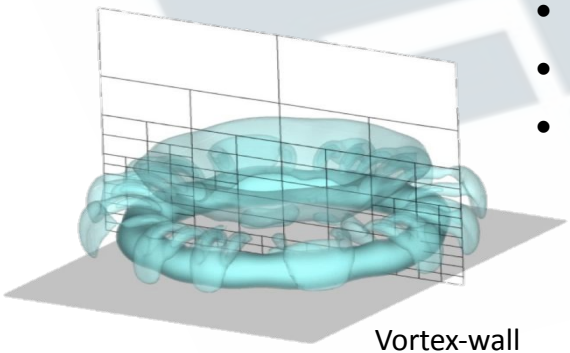


DDT Model

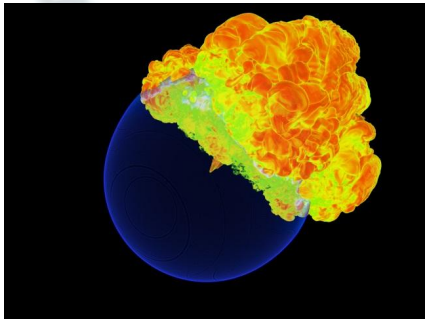
- Thermonuclear flashes
- Galaxy cluster mergers
- Combustion, detonation
- High powered laser experiment design
- Plasmas
- Cosmic ray transport and acceleration
- Fluid-structure interaction
- Whole-blood simulations
- Cardiovascular device design

Omega Experiment



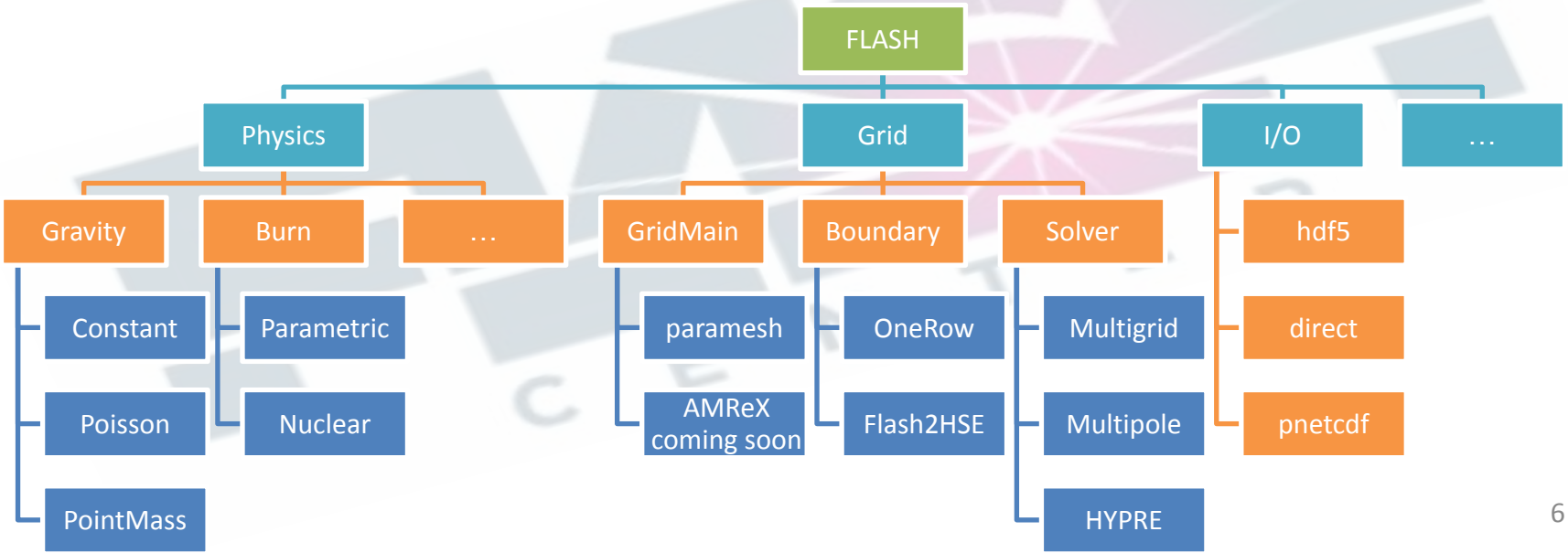
Vortex-wall interaction



Thermonuclear Plume

# FLASH
## How it works

- **Modular**, **extensible**, **highly parallelizable** software system.
- Physics modules exploit **mock inheritance** to be easily combined, added, and edited to create unique problems.



# Paramesh Adaptive Grid

## Grid refinement -- the basics

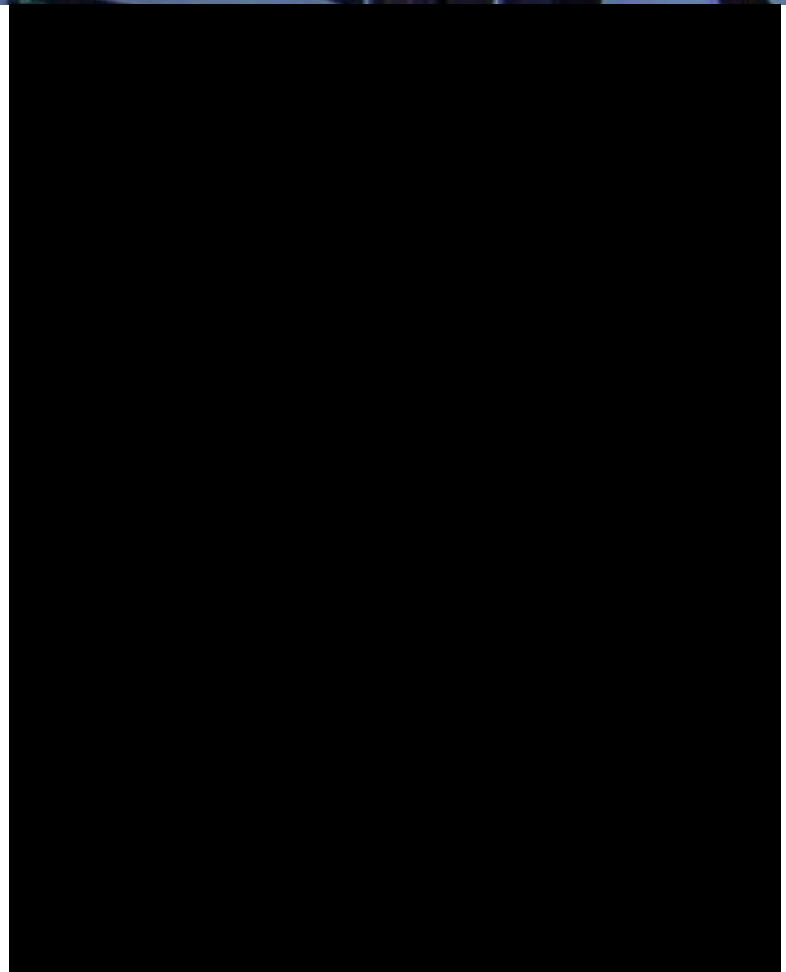
1. Set refinement criteria
   a. (in the video, density limit)

2. Calculate *dt*, the timestep

3. Perform grid calculations -- solve equations for hydro, gravity, magnetism, etc.,
   a. Communicate between blocks when needed

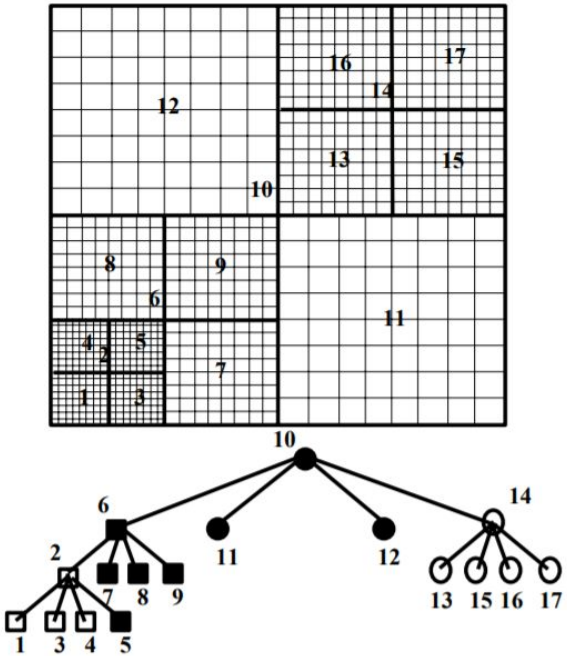
4. If necessary, refine the grid and redistribute blocks

5. Repeat 2-4 until maximum simulation time is reached



# FLASH
## PARAMESH Adaptive Grid

Block-structured AMR uses a Morton space-filling curve to distribute blocks to processor elements



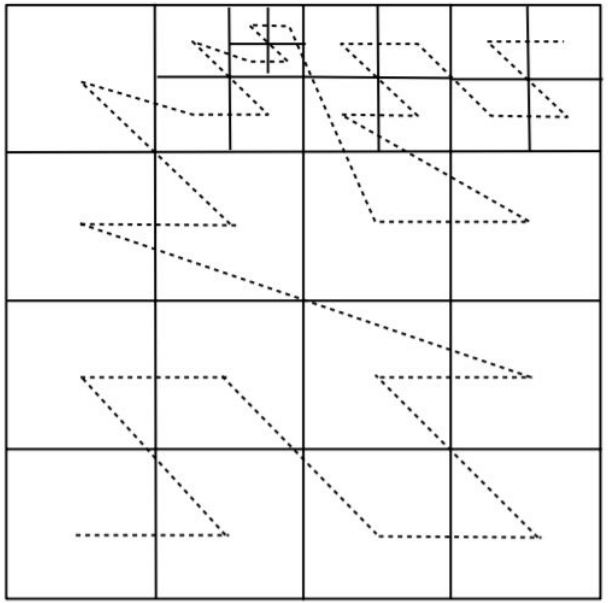
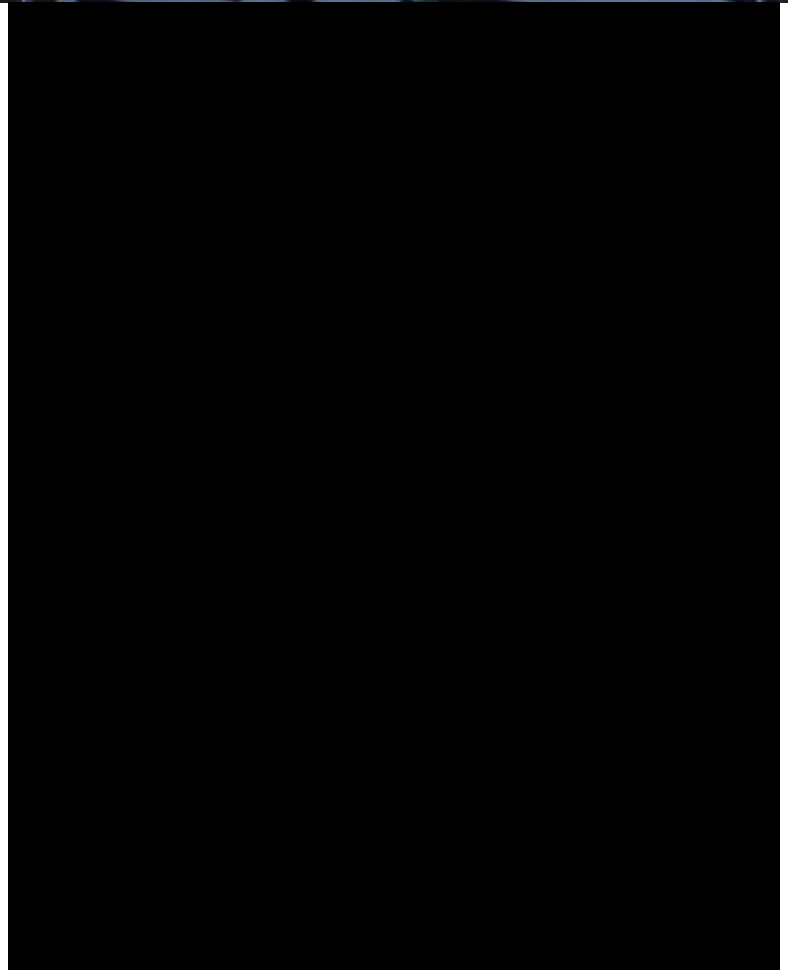



FIG. 3.—Morton space-filling curve for an arbitrary set of blocks of differing spatial resolution

# Paramesh Adaptive Grid



# Type Ia Supernovae

## Our application

**Exploring the progenitor system**

New class of white dwarf progenitors – hybrid CONe models!

**Exploring the triggering mechanism**

Previous studies have looked at detonations. What can pure deflagrations do?

**Exploring Ookami**

What is the best compiler, MPI, memory distribution? How can we use SVE instructions to produce a speedup?

**Goal**

3D suite of simulations -- new science, tremendous computing power

# Ookami

**Progress has been made…**

**Go team!**

# Ookami
## Memory -- Maxblocks

***Lesson Learned*:** *A64fx processors have a lot less memory to work with! Make sure your problem fits in the proper memory (swap, stack, global, etc.), which is less than the total 32GB per node.*

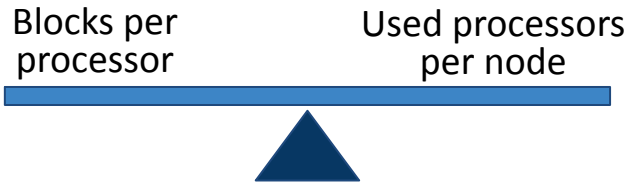
For our 2D supernova problem, default settings from machines with Intel processors allowed up to 10,000 blocks per processor.

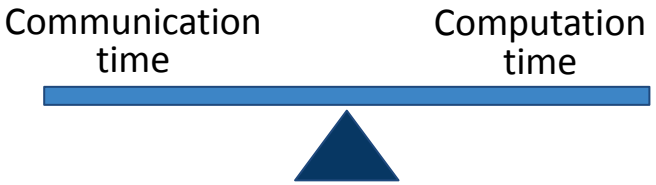
We had to find a new value that could fit within memory -- maximum of 3000 blocks per processor allowed us to use all of the processors on a node, and we chose to run with 2500 to be safe.

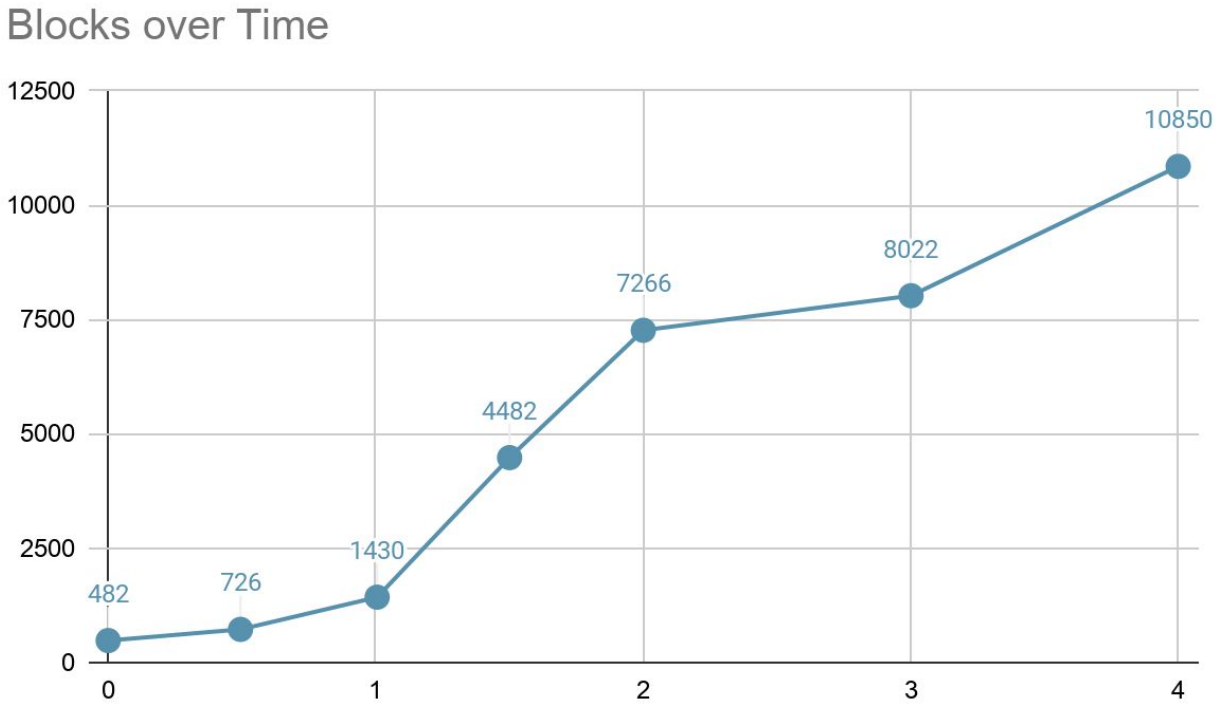
# Ookami

## Memory - Making it Fit

Need balance



# Ookami
## Compiler + MPI test

Run 2D supernova problem for 4s simulation time on 240 cores (5 nodes, 48 cores/node)

**Compilers:**

- GCC
- Cray
- ARM
- NVIDIA

+ SVE

**MPI Implementations:**

- Open-MPI
- MVAPICH

(+ CUDA for GPUs)

Different combinations
are better for
different programming languages…
…so test away!

***Lesson Learned*:** *When choosing a compiler, also choose the matching compiled MPI implementation*

# Ookami
## Compiler + MPI test

***Lesson Learned***: *When choosing a compiler, also choose the matching compiled MPI implementation*

Each MPI is compiled with specific flags and generates header files unique to the compiler used. These may or may not be compatible with different brands of compiler.

So to be safe, we use a Cray-compiled MPI with the Cray compiler, an ARM-compiled MPI with the ARM compiler, and a GCC-compiled MPI with the GCC compiler.

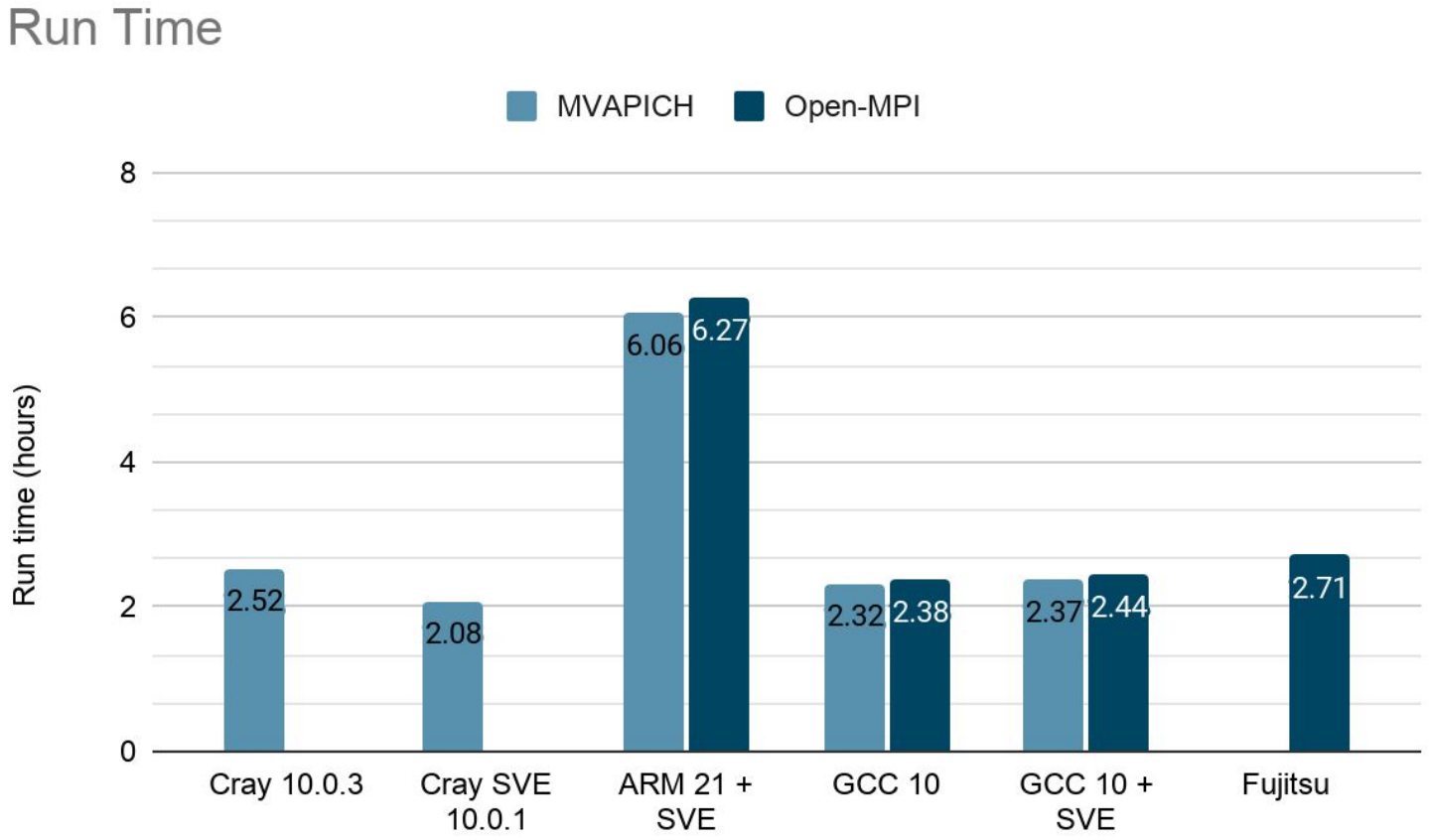
# Ookami
## Memory - MVAPICH adventures

**MVAPICH**: MPI Implementation developed and distributed by The Ohio State University, through the lab of Dr. Dhabaleswar K. Panda.

- MVAPICH + Cray Compilers: Expresses explicit “concern” over homogeneity of a computer cluster and permitted entries in Infiniband cache
  - Can tune environment variables (MV2_HOMOGENEOUS_CLUSTER, MV2_NDREG_ENTRIES(_MAX)) to obtain better performance

- MVAPICH + ARM and GNU Compilers: environment variable warnings do not show up
- Generally gives marginally better runtimes compared to OpenMPI (more on this later)

# Ookami
## Compiler + MPI test



# Ookami
## Compiler + MPI test

| Compiler | MPI | Flags | Runtime (h) |
|---|---|---|---|
| Cray SVE 10.0.1 | MVAPICH 2.3.5 | `-O3 -h vector3` | 2.08 |
| Cray 10.0.3 | MVAPICH 2.3.4 | `-O3 -h vector3` | 2.52 |
| ARM 21 | OpenMPI 4.0.5 | `-O3 -armpl -mcpu=a64fx` | 6.27 |
| ARM 21 | MVAPICH 2.3.5 | `-O3 -armpl -mcpu=a64fx` | 6.06 |
| GCC 10 | OpenMPI 4.0.5 | `-O3 -mcpu=a64fx` | 2.44 |
| GCC 10 | OpenMPI 4.0.5 | `-O3` | 2.38 |
| GCC 10 | MVAPICH 2.3.5 | `-O3 -mcpu=a64fx` | 2.37 |
| GCC 10 | MVAPICH 2.3.5 | `-O3` | 2.32 |
| Fujitsu 4.4.0a (run on Fugaku) | Fujitsu-OpenMPI | `-KSVE,A64FX,ARMV8_3_A` | 2.71 |

# Ookami
## Compiler + MPI test

**Takeaways:**

- MVAPICH is slightly faster than Open-MPI

- ARM compiler is reeeeally slow ☹
  - ***Lesson Learned*****:** *Different compilers and MPI implementations are better for different problems, so test them all!*

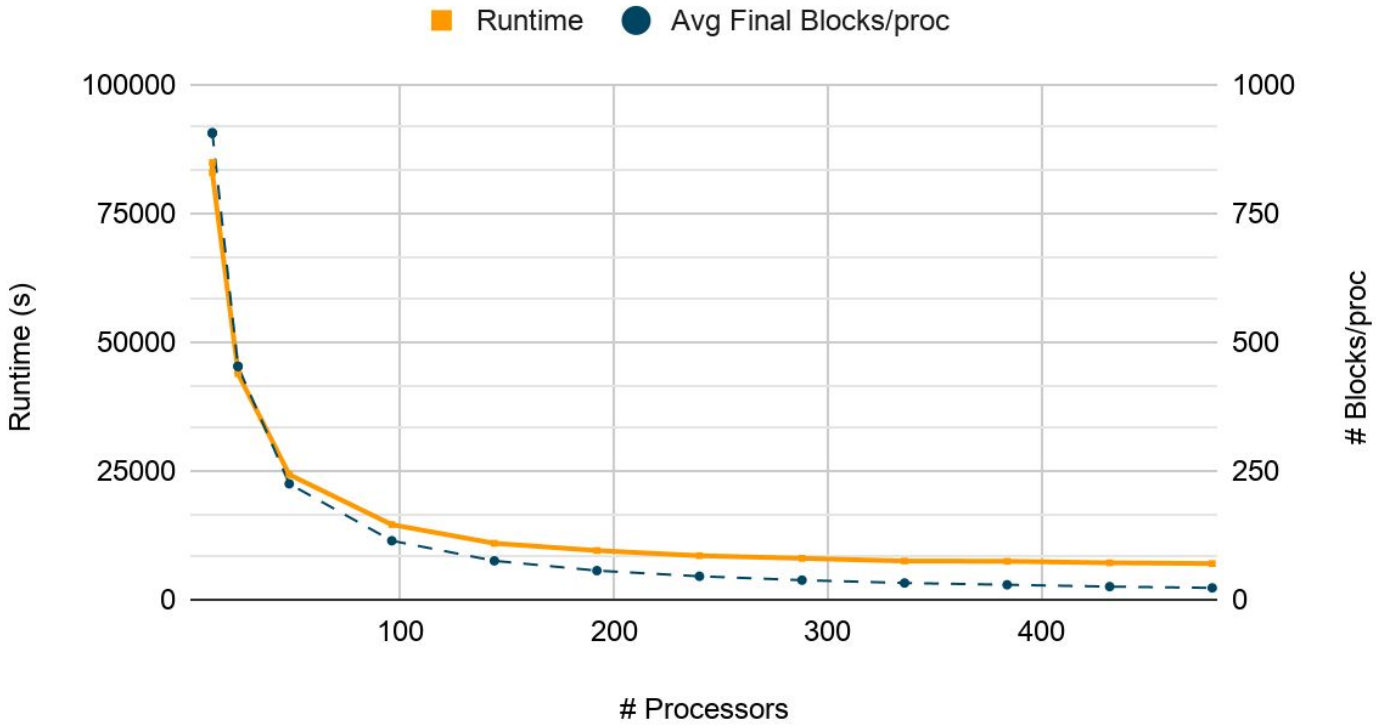
- Adding SVE instructions doesn't make much difference
  - ***Lesson Learned*****:** *SVE might not work right away if your code isn't specifically made for it*

# Ookami
## Comparison with SeaWulf

| Cluster | SeaWulf | vs | Ookami |
|---|---|---|---|
| **Processor** | 328 Intel Xeon E5-2683v3 processors<br>● Several queues<br>○ 2 CPUs per node, with 24/28/40 cores per node | | 174 A64FX processors<br>● 48 cores per node<br>○ NUMA node; processors are in groups of 12 |
| **Processor Speed** | 2.0 GHz | | 1.8 GHz |
| **Memory** | 128 GB DDR4<br>--16GB reserved for system | | 32 GB HBM2 |
| **Run time** | Compile with GCC 9.2.0 + MVAPICH 2.3.4, run on 140 cores: 1.50 hours | | Fastest run on Ookami so far is 2.08h with 240 cores, Cray 10.0.1 SVE compiler + MVAPICH 2.3.5 |

# Ookami
## Scaling Study

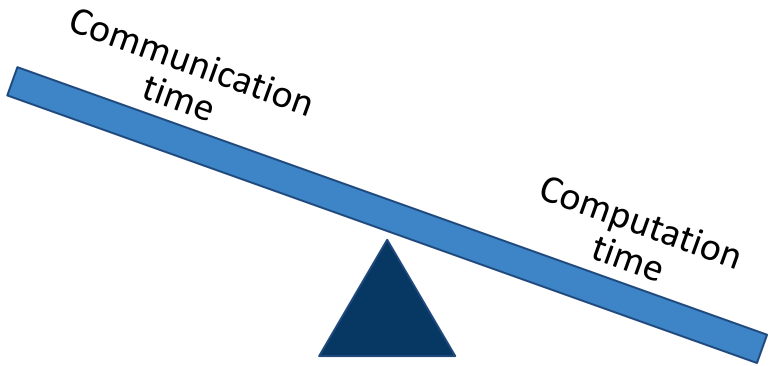
Run 2D supernova problem for 4s simulation time with GCC 10, MVAPICH 2.3.5, `-O3` (no SVE)



# Ookami
## Scaling Study

**Takeaways:**

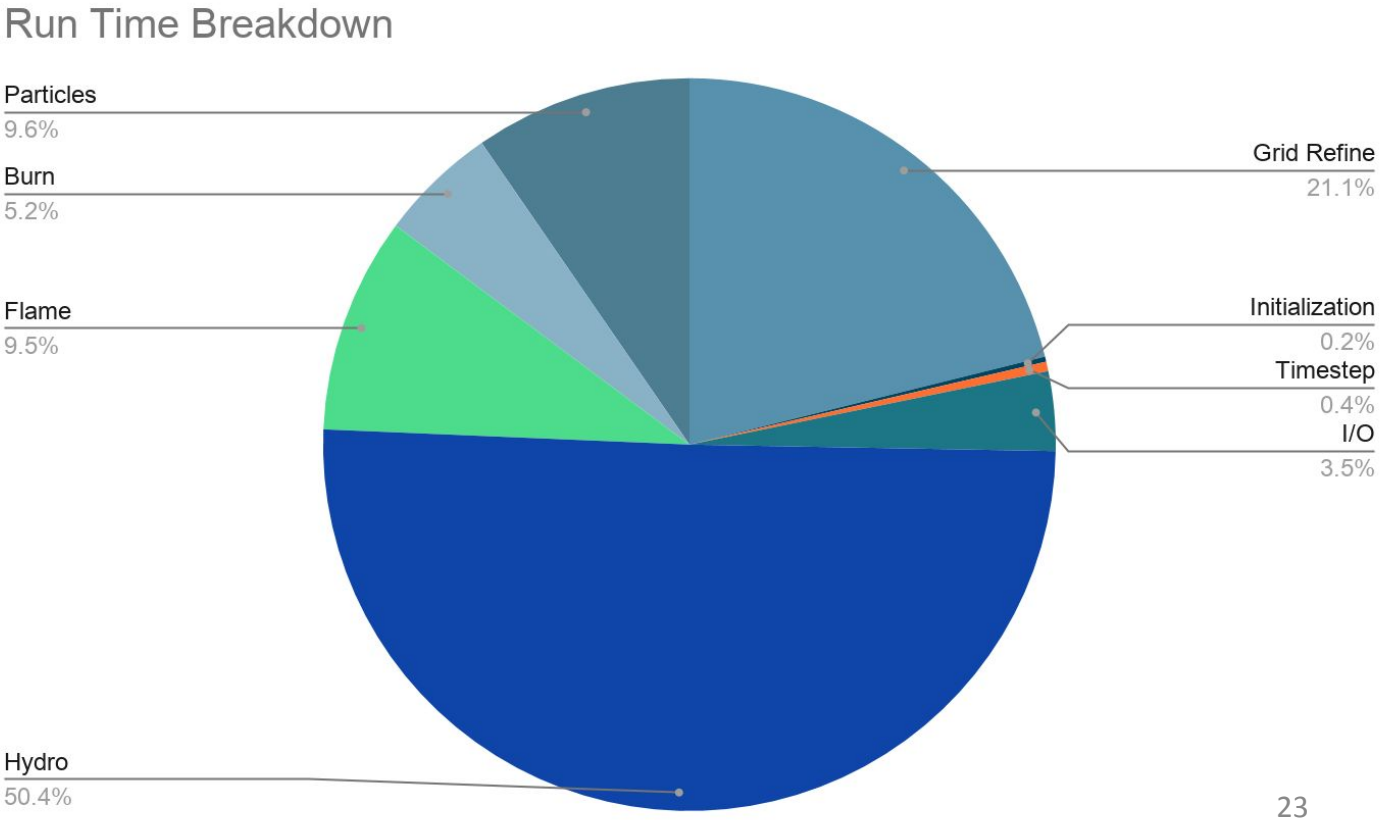
- Computationally, not communication, bounded
- This may be different in 3D



# Ookami
## Basic Profiling

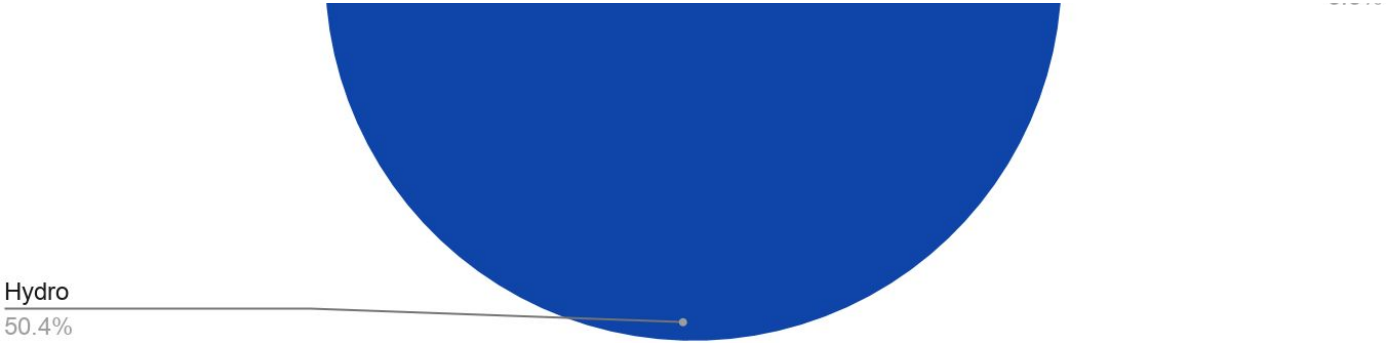
**Supernova problem uses many different modules…**

Where does it spend its time?



# Ookami
## Basic Profiling

***Lesson Learned*:** *When looking for speedups in a complicated simulation, break it into smaller test problems and optimize those individually*



# Ookami
## Sedov Explosion - 3D

- Sedov: A simulation of the Euler inviscid hydrodynamic equations
  - Only uses the hydrodynamics module
  - VERY resource heavy… especially in 3D.

- Drastic increase in memory per block:
  - 3,000 blocks can fit on a processor for 2D simulations… barely 300 for 3D simulations on Ookami

- Currently a work in progress!

# Ookami
## Sedov Explosion - 3D

- Sedov: A simulation of the Euler inviscid hydrodynamic equations
  - Only uses the hydrodynamics module
  - VERY resource heavy… especially in 3D.

- Attempting to run on 5 A64FX node/240 PE's requires a drastic drop in block count:
  - 3,000 for 2D simulations… barely 300 for 3D simulations on Ookami
  - Massive memory issue requires careful setups for each of the Cray, ARM and GNU compilers (compiler flags, MPI support)

- Currently a work in progress with the Ookami cluster @ Stony Brook University and the Fugaku supercomputer in Kobe, Japan with GNU, ARM, Cray (Ookami), and Fujitsu (Fugaku) compilers

# Ookami
## SVE

Different compilers/compiler flags turn this option on and off
Other flags print out what was and wasn't vectorized

- GCC 10, ARM

  `-O3 -mcpu=a64fx`

- Cray

  Load the specific SVE compiler (Cray 10.0.1 (with SVE) → -O3 -h vector3)

SVE instructions use the z registers – you can check your executable

`objdump -d executable | grep 'z[0-9]'`

# Ookami
## SVE

**How can we introduce SVE into this?**

- Profile the code to see where speedups can be introduced
- Note that GCC compiler cannot vectorize some math functions, use Cray compiler instead
- Old code FLASH 1 was vectorized… we're now on FLASH 4 and all that was removed
- Compare the old with the new and find places for vectorization

# In Summary…

**There is much to be done, but we have learned a lot along the way!**

***Lessons Learned*:**

1. *A64fx processors have a lot less memory to work with! Make sure your problem fits in the proper memory (swap, stack, global, etc.), which is less than the total 32GB per node*
2. *When choosing a compiler, also choose the matching compiled MPI implementation*
3. *Different compilers and MPI implementations are better for different problems, so test them all!*
4. *SVE might not work right away if your code isn't specifically made for it*
5. *When looking for speedups in a complicated simulation, break it into smaller test problems and optimize those individually*

# Acknowledgements


- Ookami is a computer technology testbed supported by the National Science Foundation under grant OAC 1927880. The authors are grateful to the entire Ookami team for their efforts in procuring and deploying the machine, and in particular for the arduous process of setting up the software used in this project.
- The authors would like to thank Stony Brook Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science at Stony Brook University for access to the high-performance SeaWulf computing system, which was made possible by a $1.4M National Science Foundation grant (#1531492).
- The FLASH code was developed in part by the DOE NNSA ASC- and DOE Office of Science ASCR-supported Flash Center for Computational Science at the University of Chicago. Work involving supernovae research was supported in part by the US Department of Energy under grant DE-FG02-87ER40317.