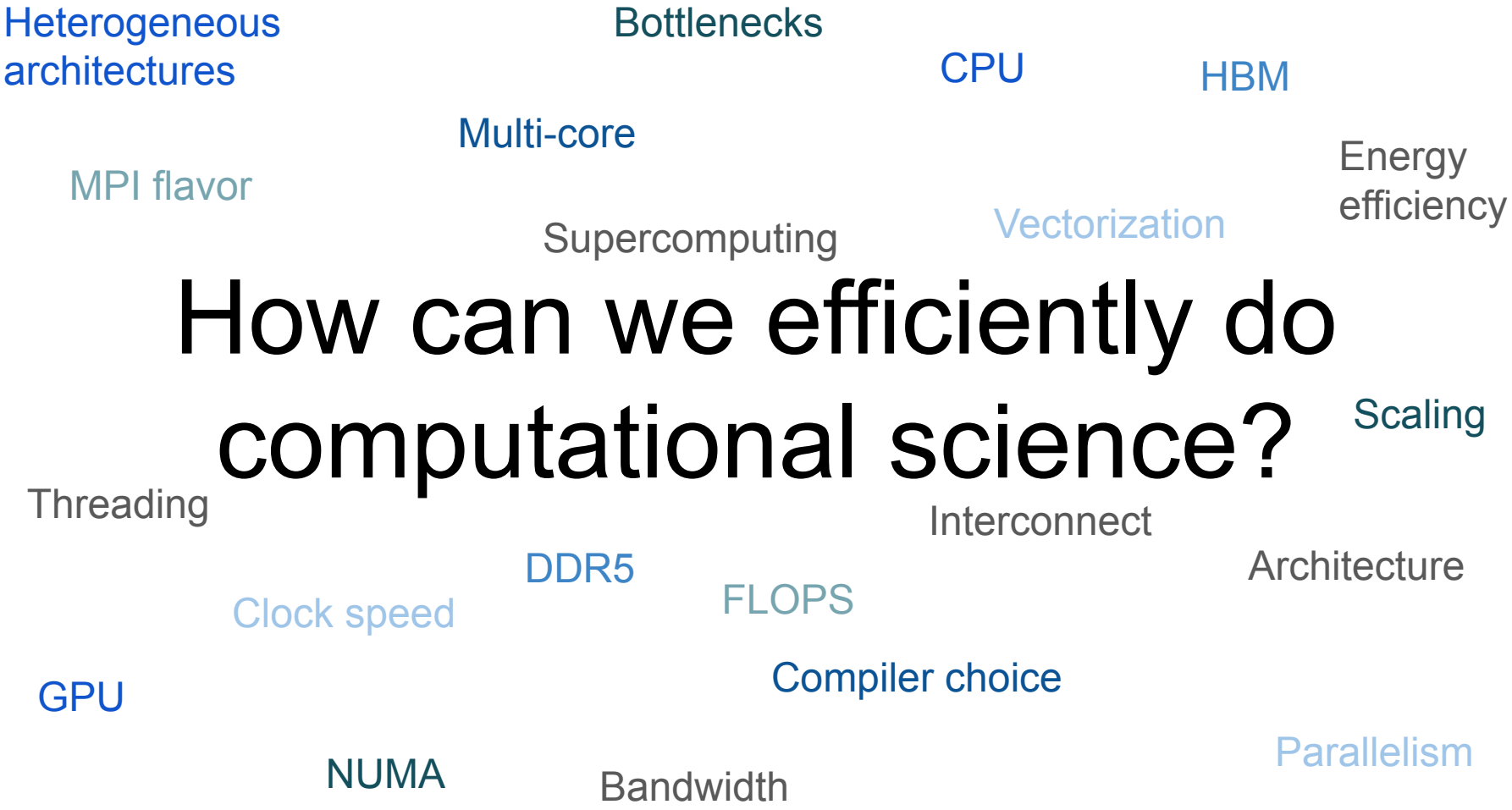
Supernovae in HPC: Benchmarking FLASH Across Advanced Computing Clusters

Joshua Martin, **Catherine Feldman**,
Eva Siegmann, Tony Curtis, David Carlson, Firat Coşkun, Daniel Wood, Raul Gonzalez, Robert J. Harrison, and Alan C. Calder

Institute for Advanced Computational Science
Stony Brook University, NY, USA


Heterogeneous architectures
Bottlenecks
CPU
HBM
Multi-core
MPI flavor
Energy efficiency
Supercomputing
Vectorization
How can we efficiently do computational science?
Scaling
Threading
Interconnect
DDR5
Architecture
FLOPS
Clock speed
Compiler choice
GPU
Parallelism
NUMA
Bandwidth

FLASH and our Supernovae Application

FLASH

Development started in 1998

Funded by the DOE under ASCI program

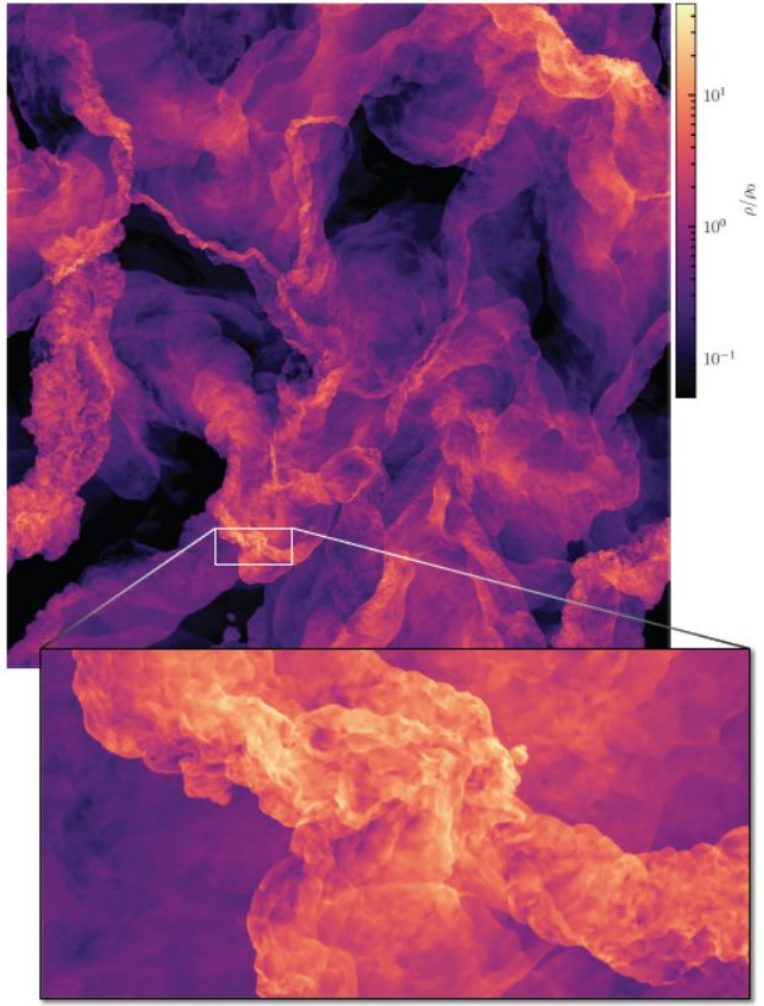
Took 20 years and ~135 person-years to create the base version (i.e. not including group-specific features)

Won SC2000 Gordon Bell Prize

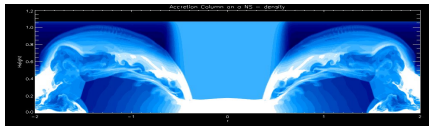
Over 100 yearly citations since 2009

Designed for compressible reactive flows

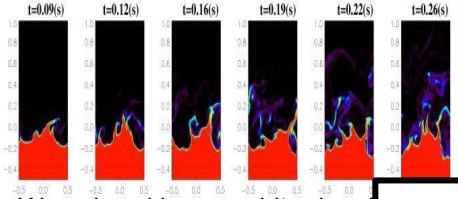
Astrophysics to High Energy Density physics



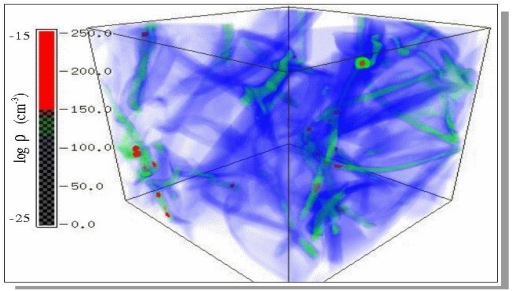
Federrath, C., Klessen, R.S., Iapichino, L. *et al.* The sonic scale of interstellar turbulence. *Nat Astron* 5, 365–371 (2021). https://doi.org/10.1038/s41550-020-01282-z



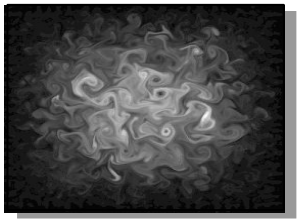
Shortly: Relativistic accretion onto NS



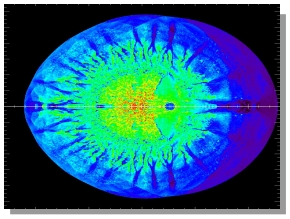
Wave breaking on white dwarf



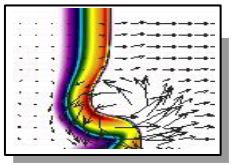
Gravitational collapse/Jeans instability



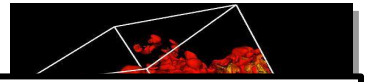
Compressed turbulence

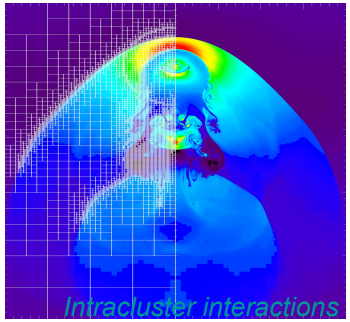


Type Ia Supernova



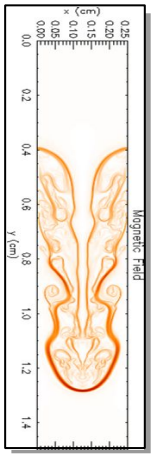
Flame-vortex interactions



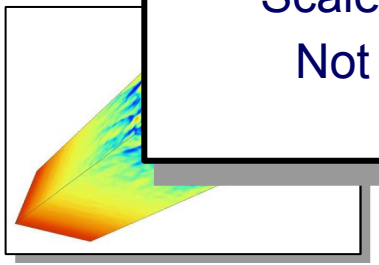


Multi-scale, multi-physics, parallel simulation code
Written primarily in modern Fortran + MPI
PARAMESH adaptive-mesh library
Scales and performs well: suitable for 3-d
Not optimized for any one architecture
Can only use CPUs

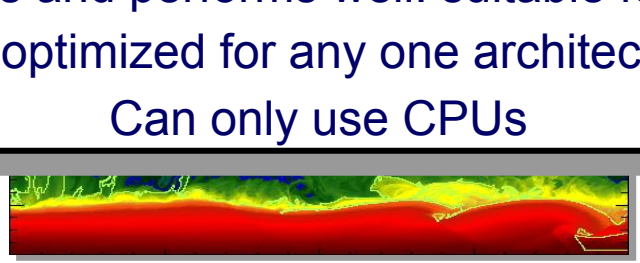
Nova outburs



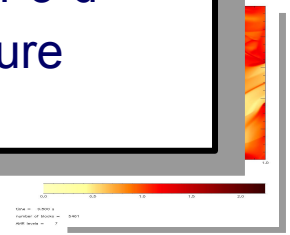
Magnetic Rayleigh-Taylor



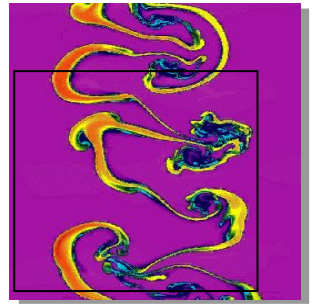
Cellular detonation



Helium burning on neutron stars



Orszag/Tang MHD vortex



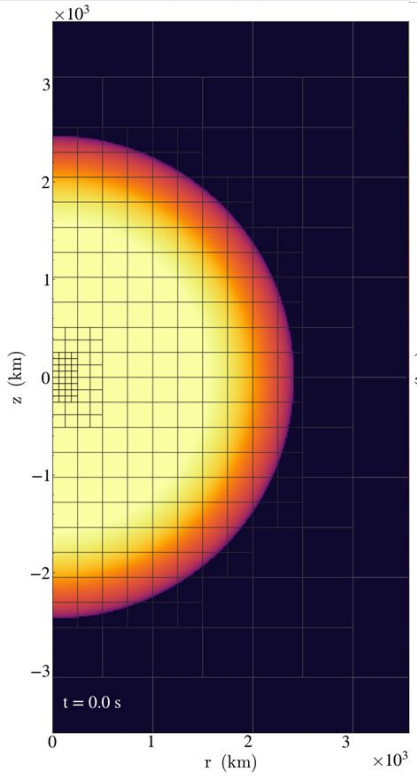
Richtmyer-Meshkov instability

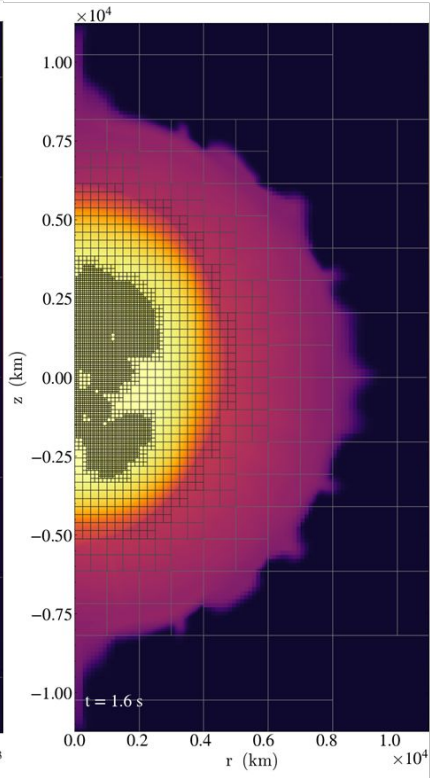
Adaptive mesh grid – computation and memory increase with time

Pass blocks to MPI ranks (cores)

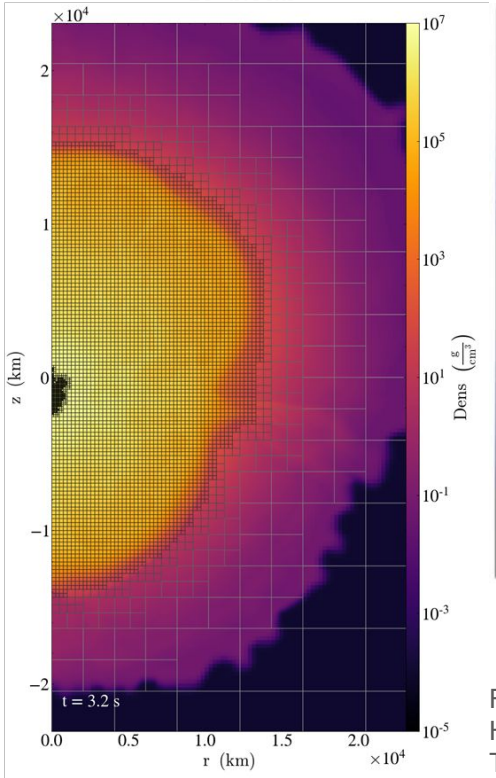
Perform physics calculations on the 16x16x16 cells in each block

10^2 blocks

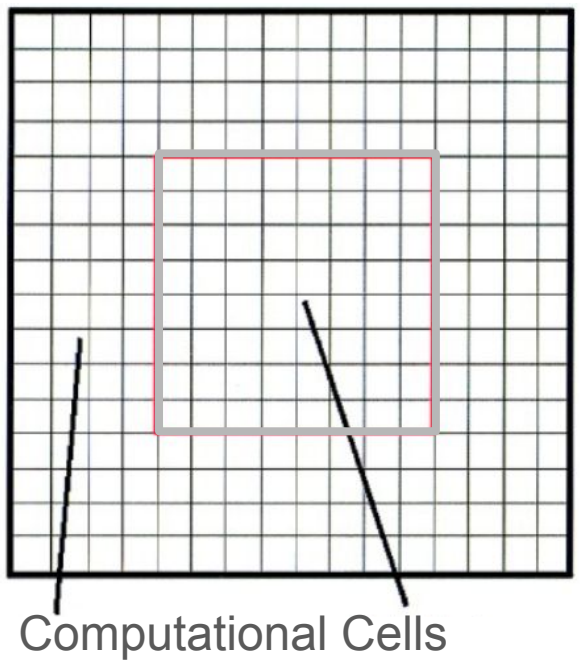




10^5 blocks



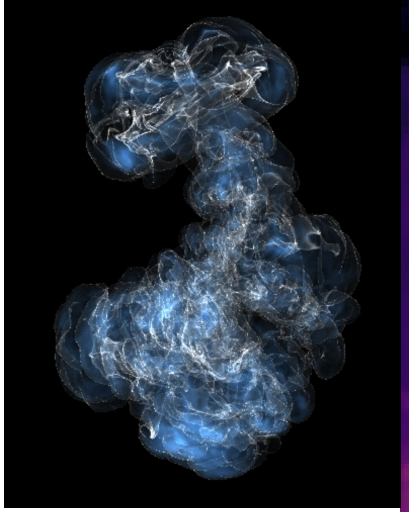
Block

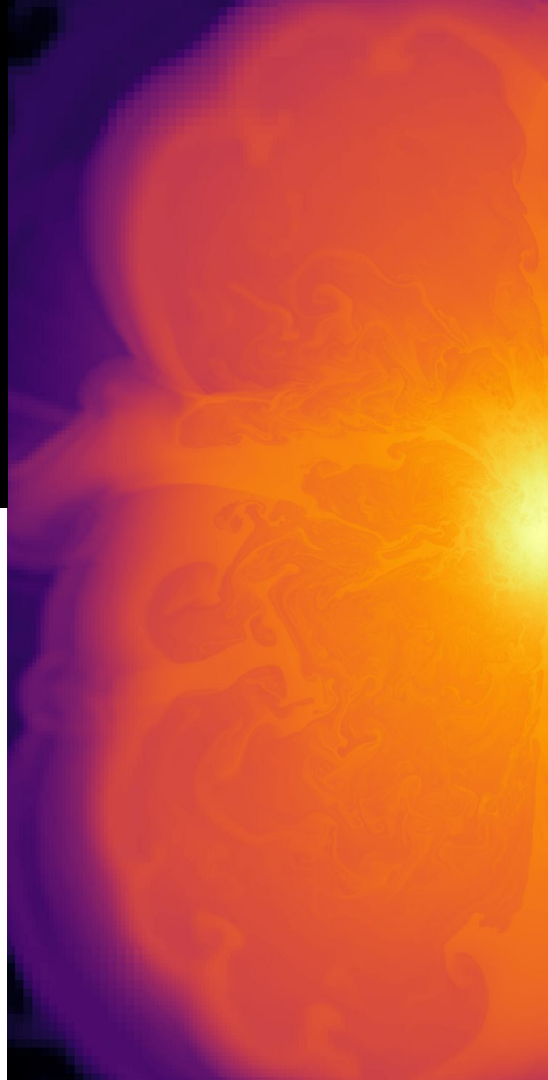


Computational Cells

Fryxell et al., “FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes”. The Astrophysical Journal Supplement Series, 131:273-334, 2000

Type Ia Supernovae





Bright stellar explosions!

Applications throughout astrophysics

Modeling requires multiple physics solvers:

- Hydrodynamics
  - Compute-heavy, local communication
- Gravity
  - Global communication
- Flame & Nuclear fusion
- Turbulence
- Grid Refinement
  - Block redistribution

Available Architectures

Architectures

2 campus clusters!

SeaWulf

NEW!

- 94 Intel Sapphire Rapids w/ HBM nodes
- 48 AMD Milan nodes
- 64 Intel Skylake nodes
- 164 Intel Haswell nodes
- ~ 4 PB of GPFS storage

Ookami

NEW!

- 174 Fujitsu A64FX-700 nodes
- ~800 TB of Lustre storage
- ACCESS resource

Architectures

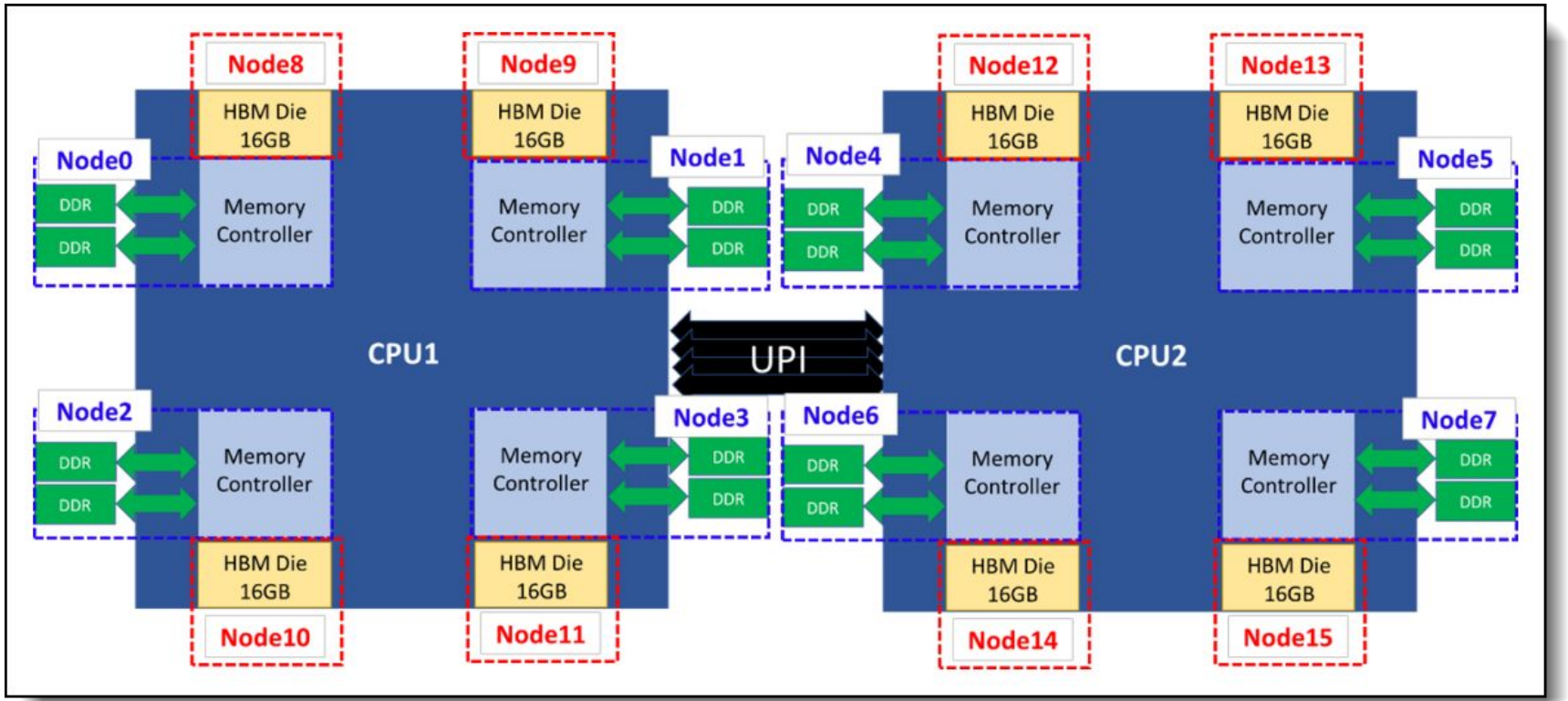
CPU Architecture	Launch Date	Sockets	NUMA Regions	Cores	Freq. (GHz)	RAM (GB)	LLC (MB)	InfiniBand
Intel Xeon Max 9468 "Sapphire Rapids w/ HBM"	2023	2	8	96	0.80 - 3.5	256 DDR5 128 HBM	L3, 105 (48 cores)	NDR (400 Gb/s)
Intel Xeon Gold 6148 "Skylake"	2017	2	N/A	40	2.4	192 DDR4	L3, 27.5	FDR (56 Gb/s)
AMD EPYC 7643 "Milan"	2021	2	8	96	1.5, 1.9, 2.3	256 DDR4	L3, 32 (6 cores)	HDR (100 Gb/s)
Fujitsu A64FX-700	2019	1	4	48	1.8	32 HBM	L2, 8 (48 cores)	HDR (100 Gb/s)

Architectures

CPU Architecture	Launch Date	Sockets	NUMA Regions	Cores	Freq. (GHz)	RAM (GB)	LLC (MB)	InfiniBand
Intel Xeon Max 9468 "Sapphire Rapids w/ HBM"	2023	2	8	96	0.80 - 3.5	256 DDR5 128 HBM	L3, 105 (48 cores)	NDR (400 Gb/s)
Intel Xeon Gold 6148 "Skylake"	2017	2	N/A	40	2.4	192 DDR4	L3, 27.5	FDR (56 Gb/s)
AMD EPYC 7643 "Milan"	2021	2	8	96	1.5, 1.9, 2.3	256 DDR4	L3, 32 (6 cores)	HDR (100 Gb/s)
Fujitsu A64FX-700	2019	1	4	48	1.8	32 HBM	L2, 8 (48 cores)	HDR (100 Gb/s)

SPR Configuration: Flat Mode in SNC4 Clustering

Both DDR5 and HBM are available as RAM
4 NUMA regions per CPU, each with 12 cores, 16 GB HBM, and 32 GB DDR5



https://lenovopress.lenovo.com/lp1738-implementing-intel-high-bandwidth-memory

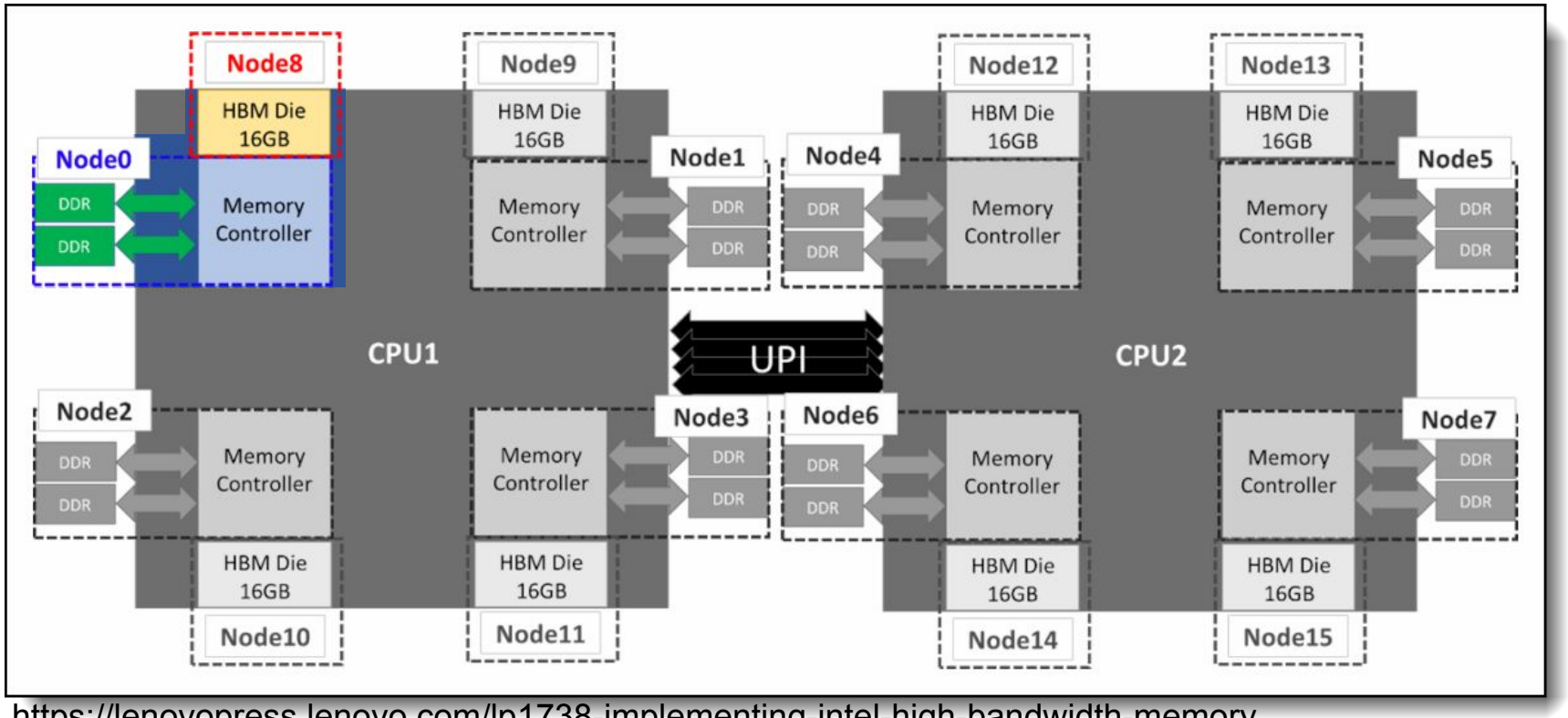
SPR Configuration: Flat Mode in SNC4 Clustering

```
$ numactl -H
```

```
available: 16 nodes (0-15)
node 0 cpus: 0 1 2 3 4 5 48 49 50 51 52 53
node 0 size: 31946 MB
node 0 free: 8250 MB
node 1 cpus: 6 7 8 9 10 11 54 55 56 57 58 59
node 1 size: 32253 MB
node 1 free: 24983 MB
node 2 cpus: 12 13 14 15 16 17 60 61 62 63 64 65
node 2 size: 32253 MB
node 2 free: 25449 MB
node 3 cpus: 18 19 20 21 22 23 66 67 68 69 70 71
node 3 size: 32253 MB
node 3 free: 25720 MB
node 4 cpus: 24 25 26 27 28 29 72 73 74 75 76 77
node 4 size: 32253 MB
node 4 free: 25755 MB
node 5 cpus: 30 31 32 33 34 35 78 79 80 81 82 83
node 5 size: 32253 MB
node 5 free: 25664 MB
node 6 cpus: 36 37 38 39 40 41 84 85 86 87 88 89
node 6 size: 32201 MB
node 6 free: 25410 MB
node 7 cpus: 42 43 44 45 46 47 90 91 92 93 94 95
node 7 size: 32247 MB
node 7 free: 25433 MB
node 8 cpus:
node 8 size: 16384 MB
node 8 free: 16 MB
node 9 cpus:
node 9 size: 16384 MB
node 9 free: 10 MB
node 10 cpus:
node 10 size: 16384 MB
node 10 free: 12 MB
node 11 cpus:
node 11 size: 16384 MB
node 11 free: 15 MB
node 12 cpus:
node 12 size: 16384 MB
node 12 free: 13 MB
node 13 cpus:
node 13 size: 16384 MB
node 13 free: 11 MB
node 14 cpus:
node 14 size: 16384 MB
node 14 free: 12 MB
node 15 cpus:
node 15 size: 16384 MB
node 15 free: 14 MB
```

SPR Configuration: Flat Mode in SNC4 Clustering

In this configuration, numa node 0 is local DDR, and node 8 is local HBM
Nodes 0-7 are DDR5 and have compute cores, Nodes 8-15 are HBM



https://lenovopress.lenovo.com/lp1738-implementing-intel-high-bandwidth-memory

SPR Configuration: Flat Mode in SNC4 Clustering

Prioritize HBM by adding `numactl --preferred-many=8-15` to your run/MPI runtime call

```
mpiexec -n 96 numactl --preferred-many=8-15 ./flash4
```

Test Problem

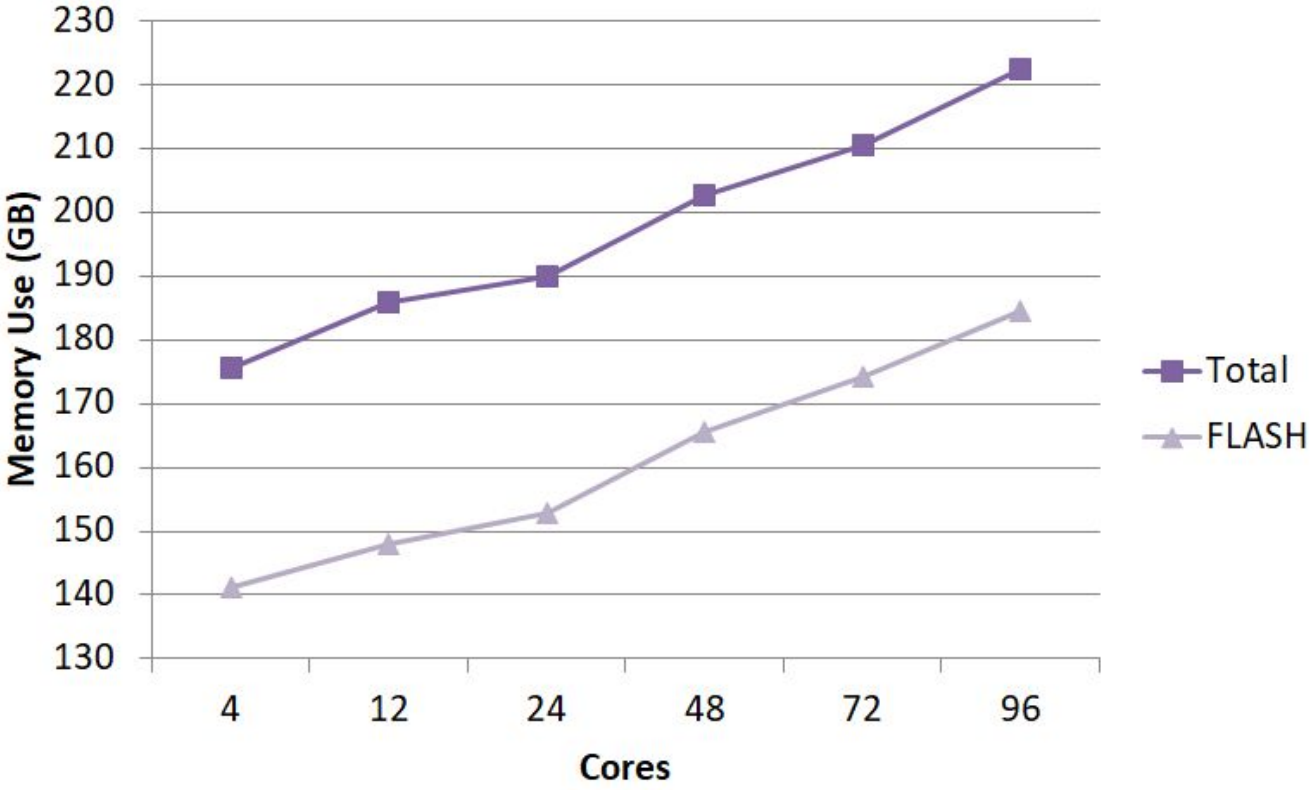
Type Ia Supernovae: Choosing a Test Problem is Hard

- 3D Type Ia Supernova Explosion
- In the middle of explosion, so all physics units computing
- 220 GB (~180 GB for FLASH, the rest for MPI, etc) to fit in DDR5
  - 1 SPR node has 256 GB DDR5, 128 GB HBM
  - Refined the grid to give 330,000 3D blocks
- Ran for 20 timesteps, 10 grid refinements
  - 20 minutes on 96 cores, enough time for data collection

Looked at strong scaling, computational efficiency, and energy consumption

Type Ia Supernovae: Test Problem

Memory footprint increases with # cores



Data Collection

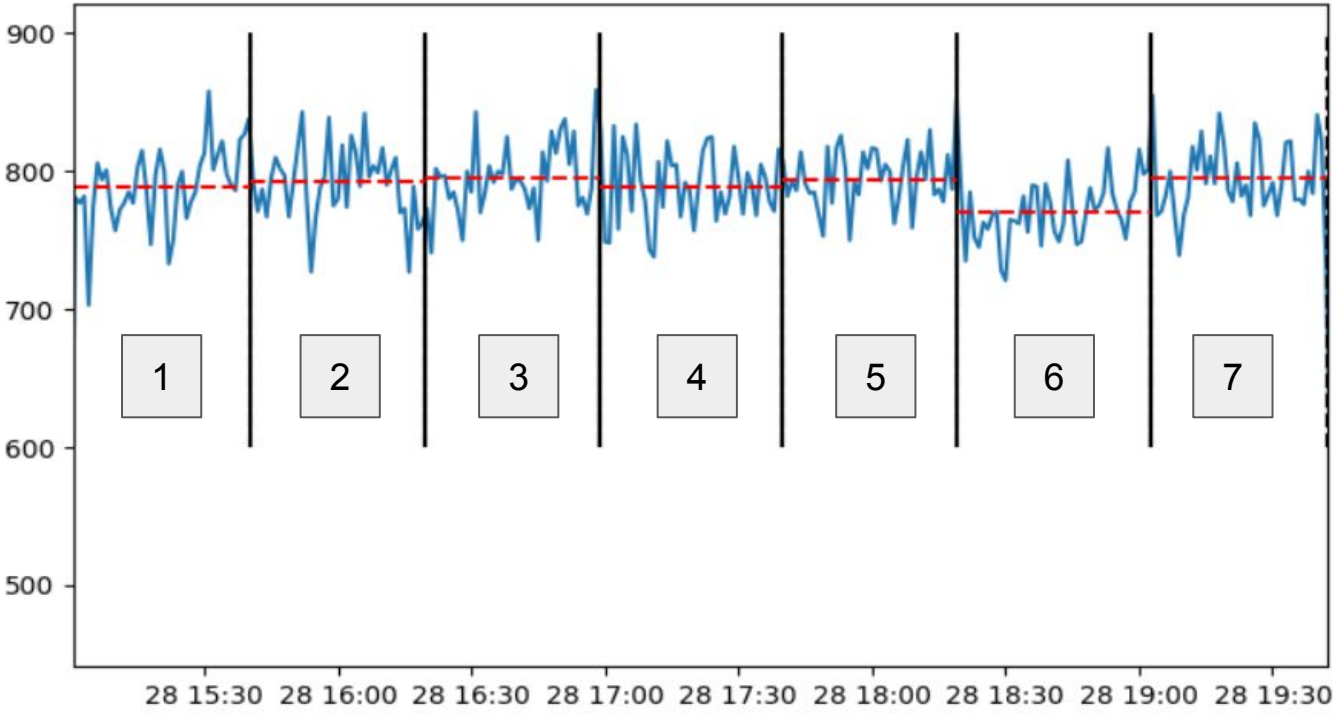
Runtime Measurements

Run the same simulation 7 times on the same node/group of nodes

Take the reported max *evolution* runtime, so initialization, I/O, etc. are not included, of each simulation and average them together

Power Measurements – SeaWulf

Collect power usage data from `ipmpitool`, every minute, for each node.
Average over each sim, remove first and last data points for warm-up / cool-down dips.
Calculate energy used by each node (avg power*runtime) and sum over all nodes.

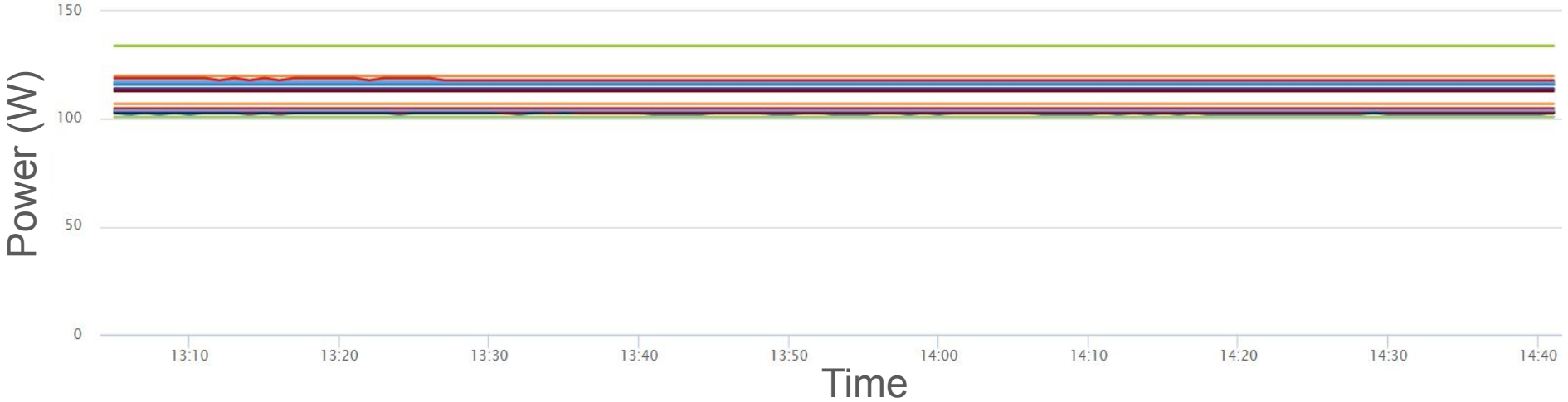


Power Measurements – Ookami

XDMoD gives the average per-node power over the run. Power consumption is stable.
Energy consumption = avg power per-node * # nodes * runtime

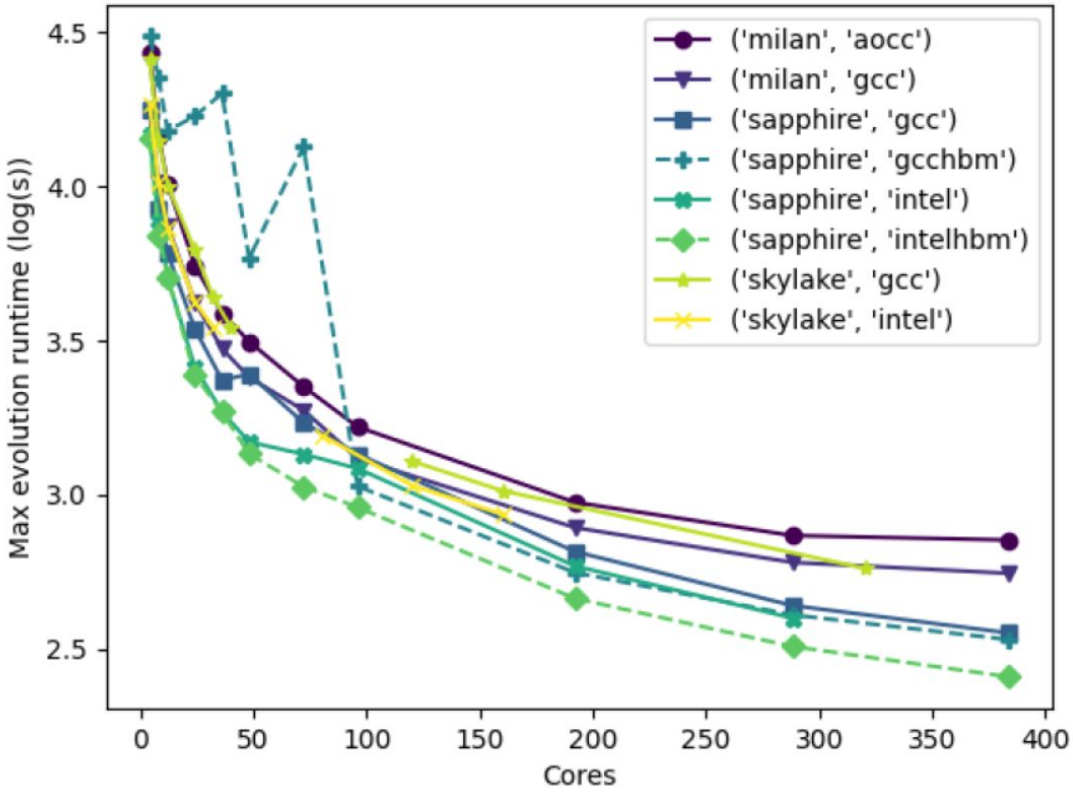
Accounting data | Executable information | Summary metrics | **Detailed metrics**

Device	Average	Count	Standard Dev.
block			
cpu			
ipmi			
power			
max	111.71428571429	21	8.210620300214
mean	111.61656746032	21	8.2386328854527

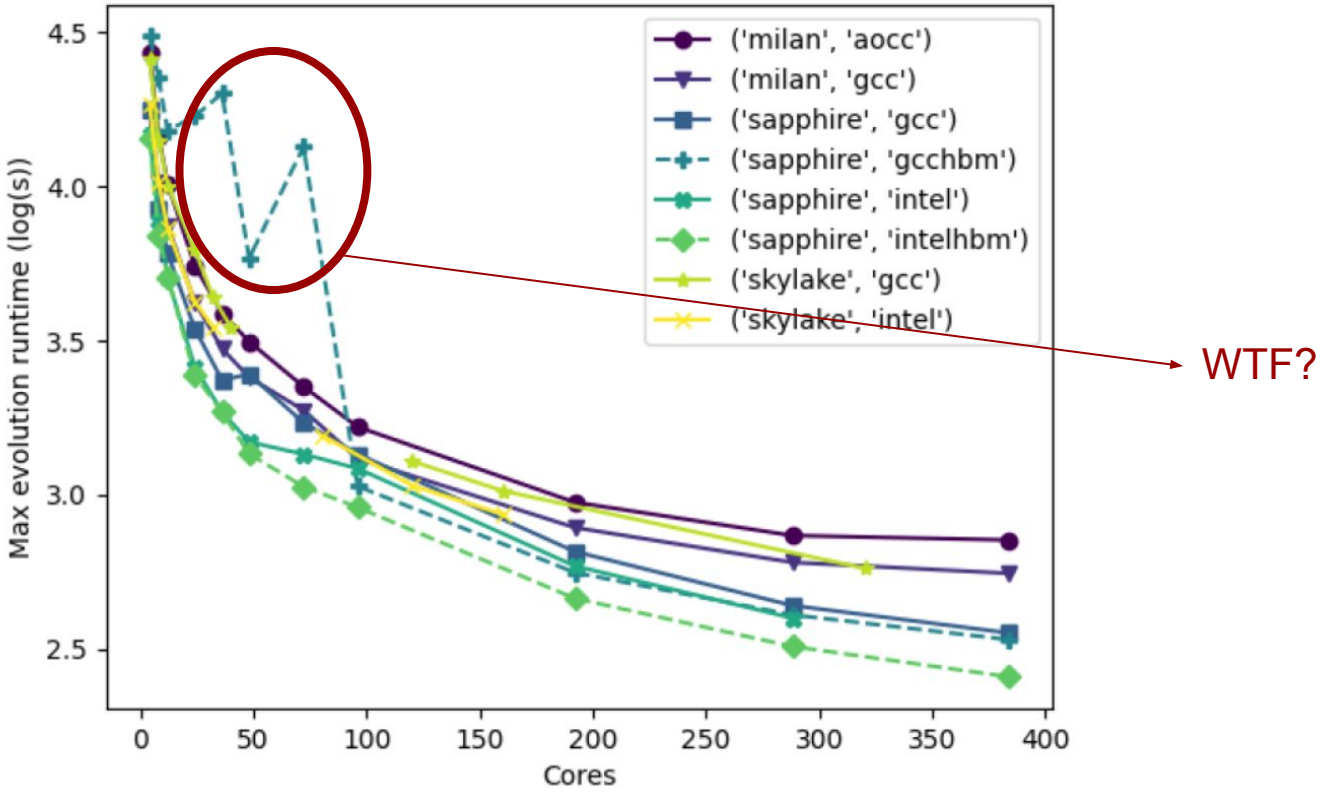


Results!

Strong scaling – SeaWulf

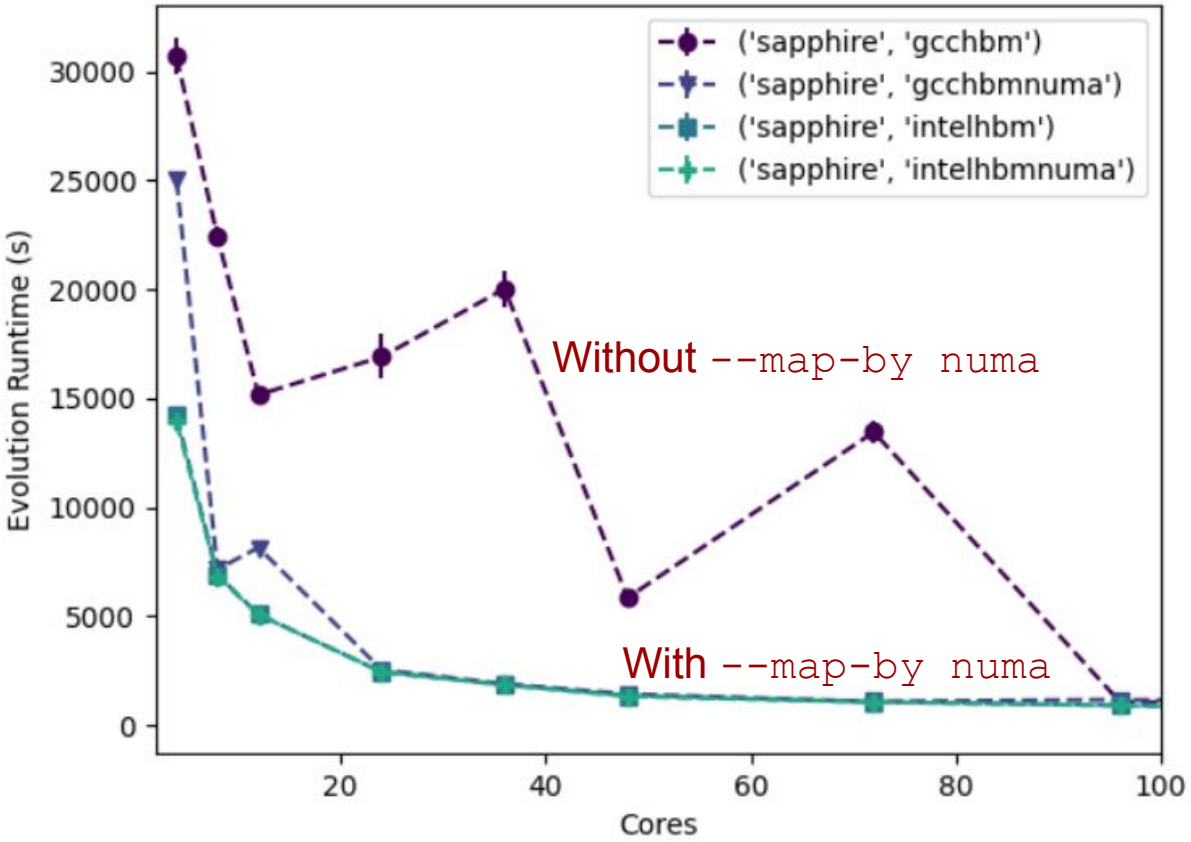


Strong scaling – SeaWulf



The fix: `--map-by numa`

Weirdness only happens when using HBM on < full-subscription with GCC compiler



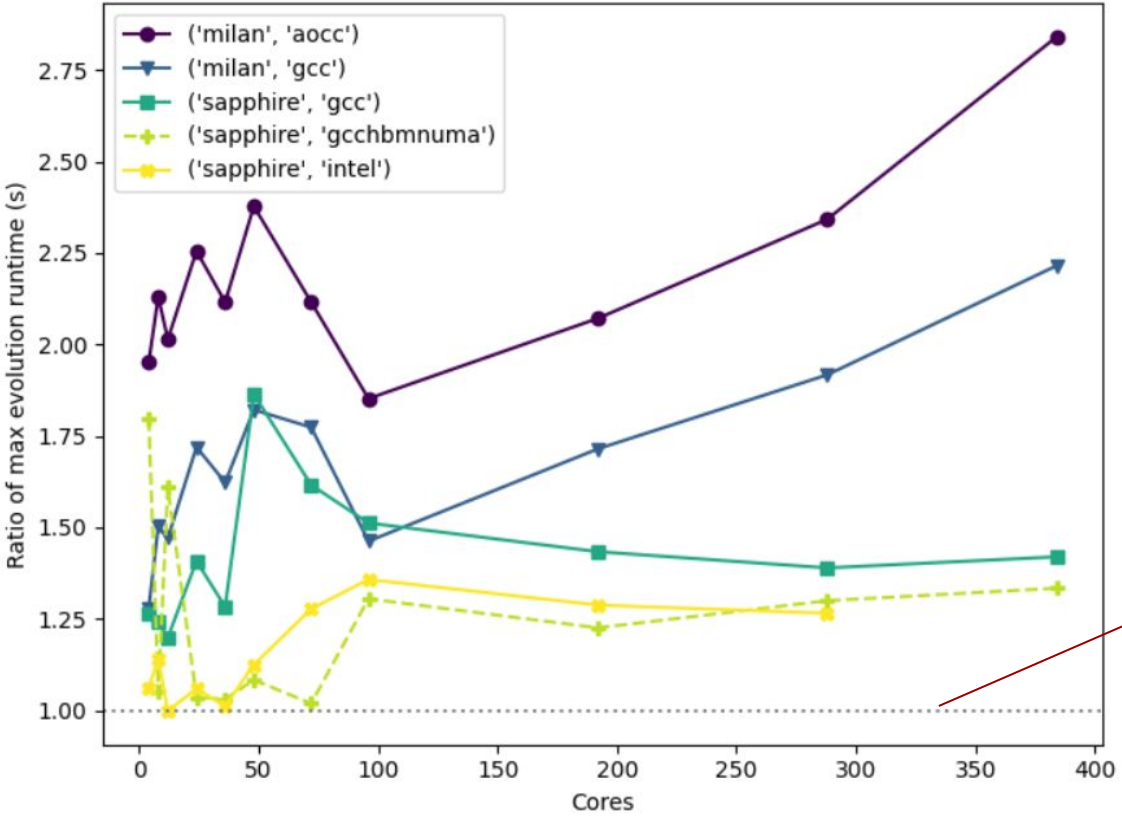
The “Magic” Runtime Flag `--map-by numa`

Default bindings look the same as numa bindings, so not sure why this fixes things…

Rank 0 bound to socket 0: [B/B/B/B/B/B/B/B/B/B/B/B/.][./././././././././././././././././././.]
Rank 1 bound to socket 0: [././././././././B/B/B/B/B/B/B/B/B/B/B/B/./././././././././././.][./././././././././././././././././././.]
Rank 2 bound to socket 0: [./././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/./././.][./.]
Rank 3 bound to socket 0: [./B/B/B/B/B/B/B/B/B/B/B/B][./.]
Rank 4 bound to socket 1: [./.][B/B/B/B/B/B/B/B/B/B/B/B/./././.]
Rank 5 bound to socket 1: [./.][././././././B/B/B/B/B/B/B/B/B/B/B/B/./.]
Rank 6 bound to socket 1: [./.][././././././././././././././B/B/B/B/B/B/B/B/B/B/B/B/./.]
Rank 7 bound to socket 1: [./.][./B/B/B/B/B/B/B/B/B/B/B/B]
Rank 8 bound to socket 0: [B/B/B/B/B/B/B/B/B/B/B/B/./././.][./././.]
Rank 9 bound to socket 0: [./././.][./././.]
Rank 10 bound to socket 0: [./././.][./././.]
Rank 11 bound to socket 0: [./././.][./././.]
Rank 12 bound to socket 1: [./././.][B/B/B/B/B/B/B/B/B/B/B/B/./.]
Rank 13 bound to socket 1: [./././.][./././.]
Rank 14 bound to socket 1: [./././.][./././.]
Rank 15 bound to socket 1: [./././.][./././.]
Rank 16 bound to socket 0: [B/B/B/B/B/B/B/B/B/B/B/B/./.][./.]
Rank 17 bound to socket 0: [./.][./.]
Rank 18 bound to socket 0: [./.][./.]
Rank 19 bound to socket 0: [./.][./.]
Rank 20 bound to socket 1: [./.][B/B/B/B/B/B/B/B/B/B/B/B/./.]
Rank 21 bound to socket 1: [./.][./.]
Rank 22 bound to socket 1: [./.][./.]
Rank 23 bound to socket 1: [./.][./.]

Comparing other configurations to the best

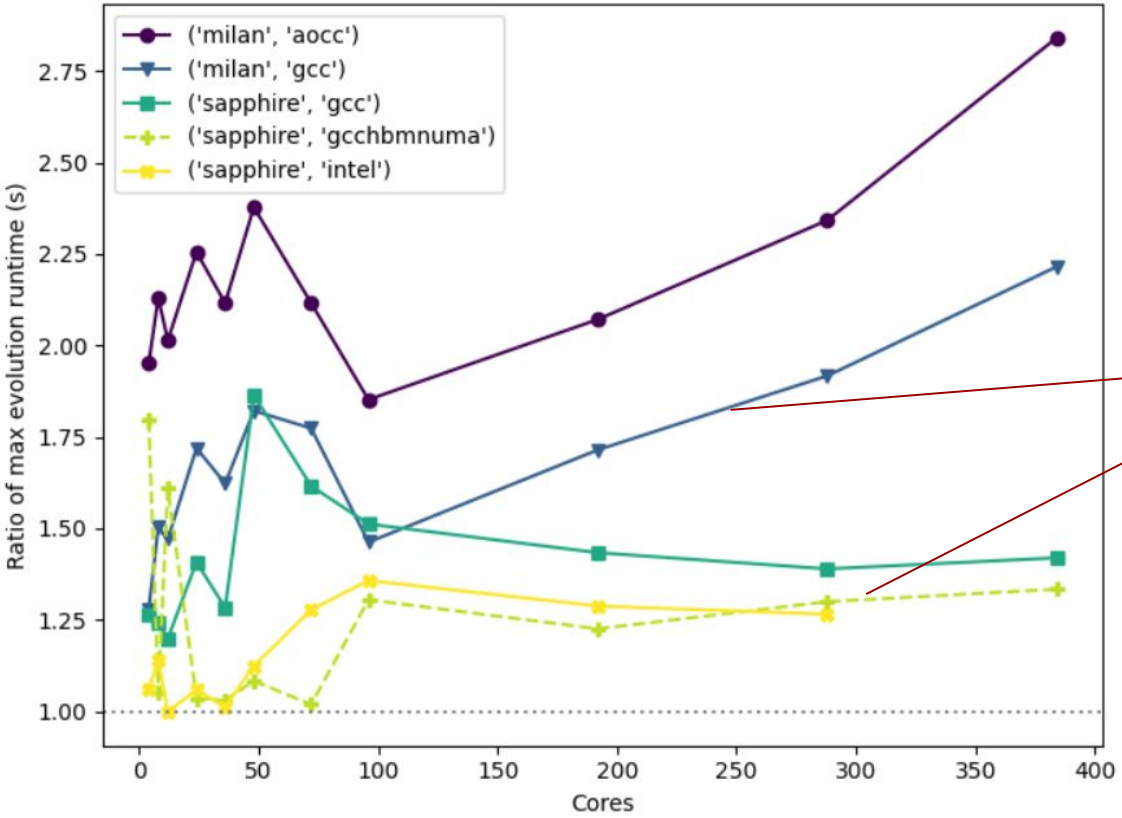
Ratio of max evolution time of configuration : max evolution time of SPR + Intel compiler + HBM + NUMA mapping



Baseline Runtime, Ratio = 1
SPR + Intel + HBM + NUMA

Comparing other configurations to the best

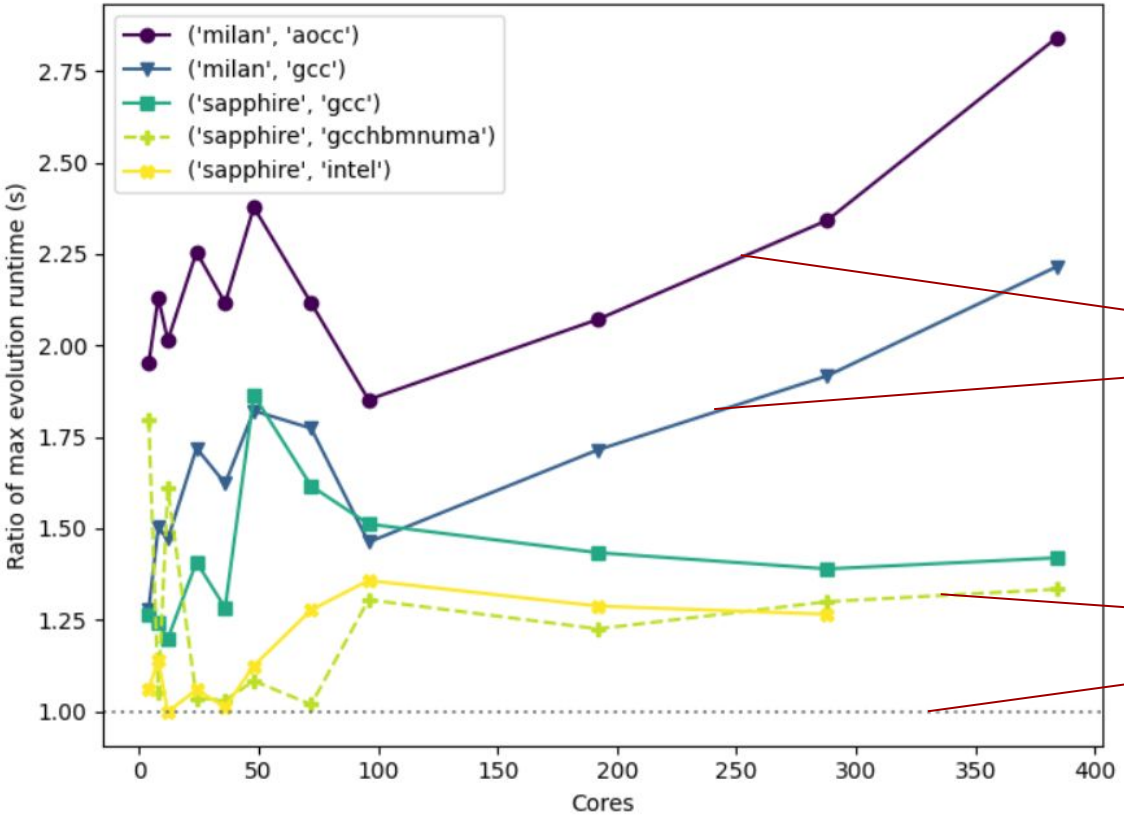
Ratio of max evolution time of configuration : max evolution time of SPR + Intel compiler + HBM + NUMA mapping



Milan scales worse than SPR as you scale across nodes

Comparing other configurations to the best

Ratio of max evolution time of configuration : max evolution time of SPR + Intel compiler + HBM + NUMA mapping

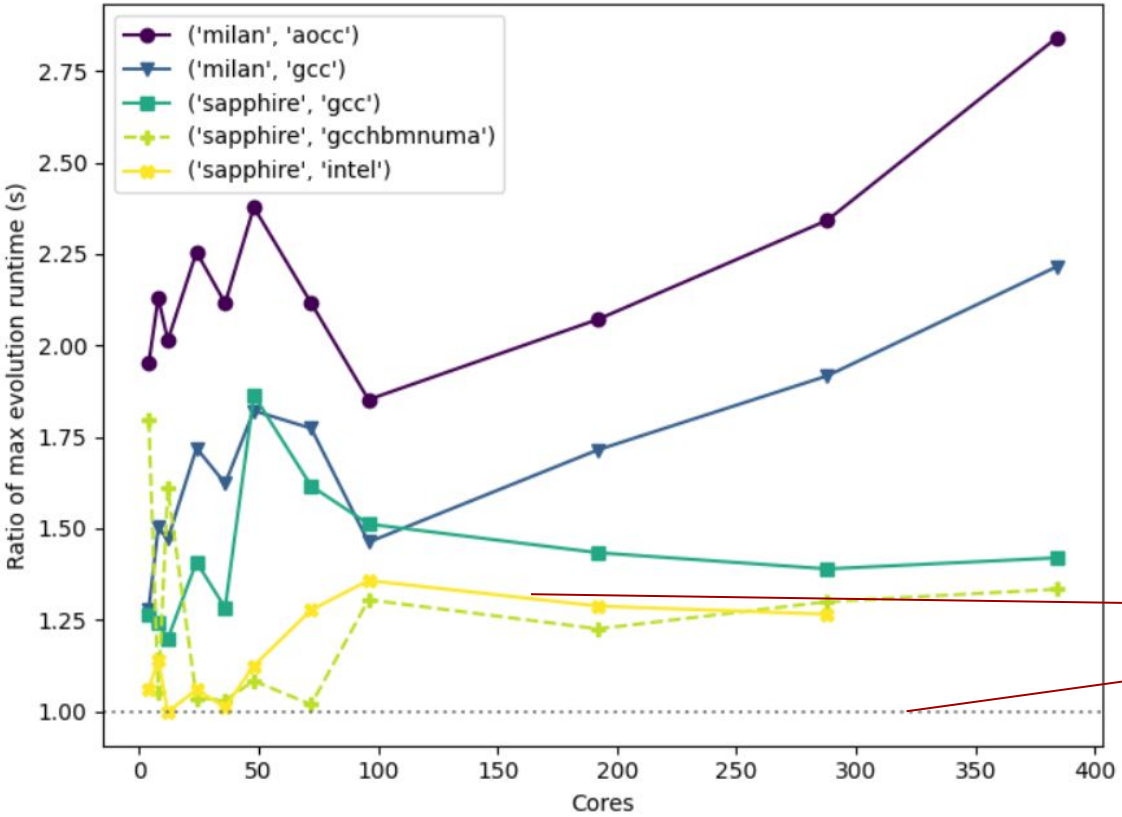


Milan native compiler performs worse than gcc compiler

But Intel native compiler performs better than gcc compiler

Comparing other configurations to the best

Ratio of max evolution time of configuration : max evolution time of SPR + Intel compiler + HBM + NUMA mapping

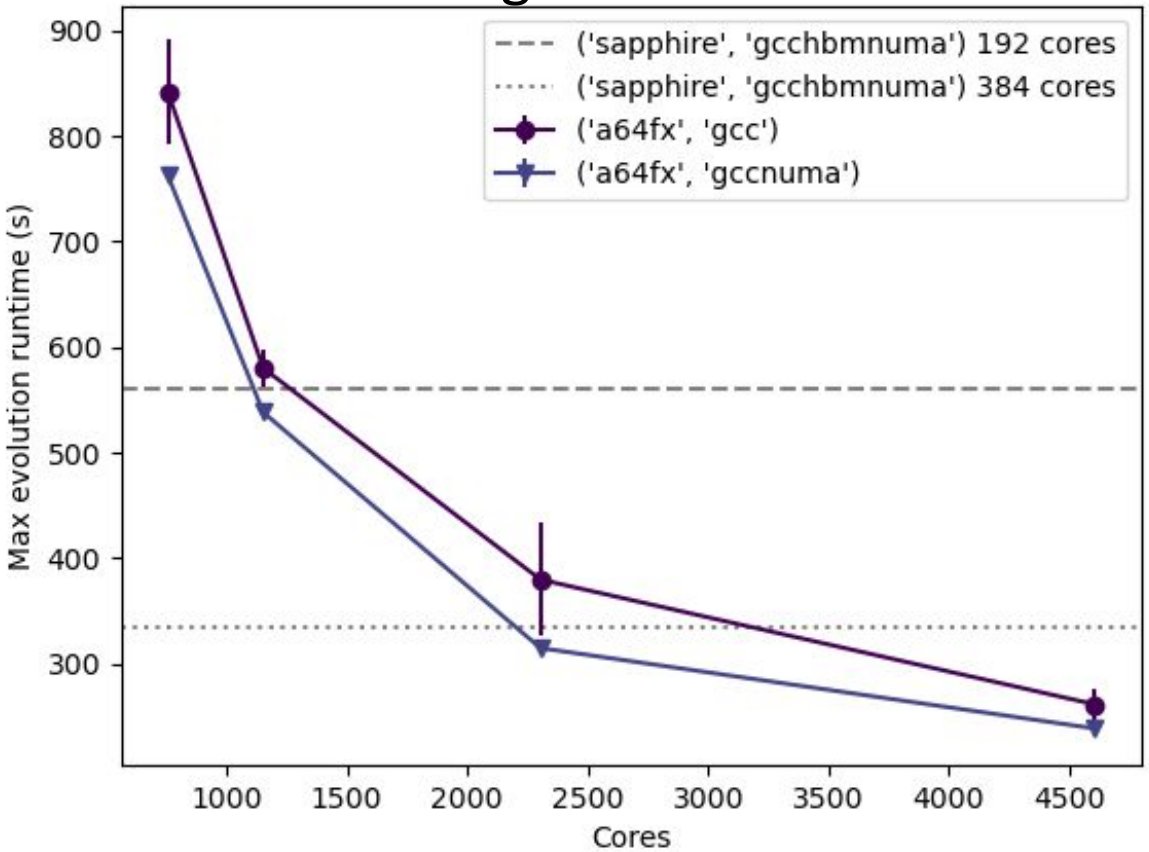


HBM is ~25% faster, but only with full-subscription and multi-node (why?)

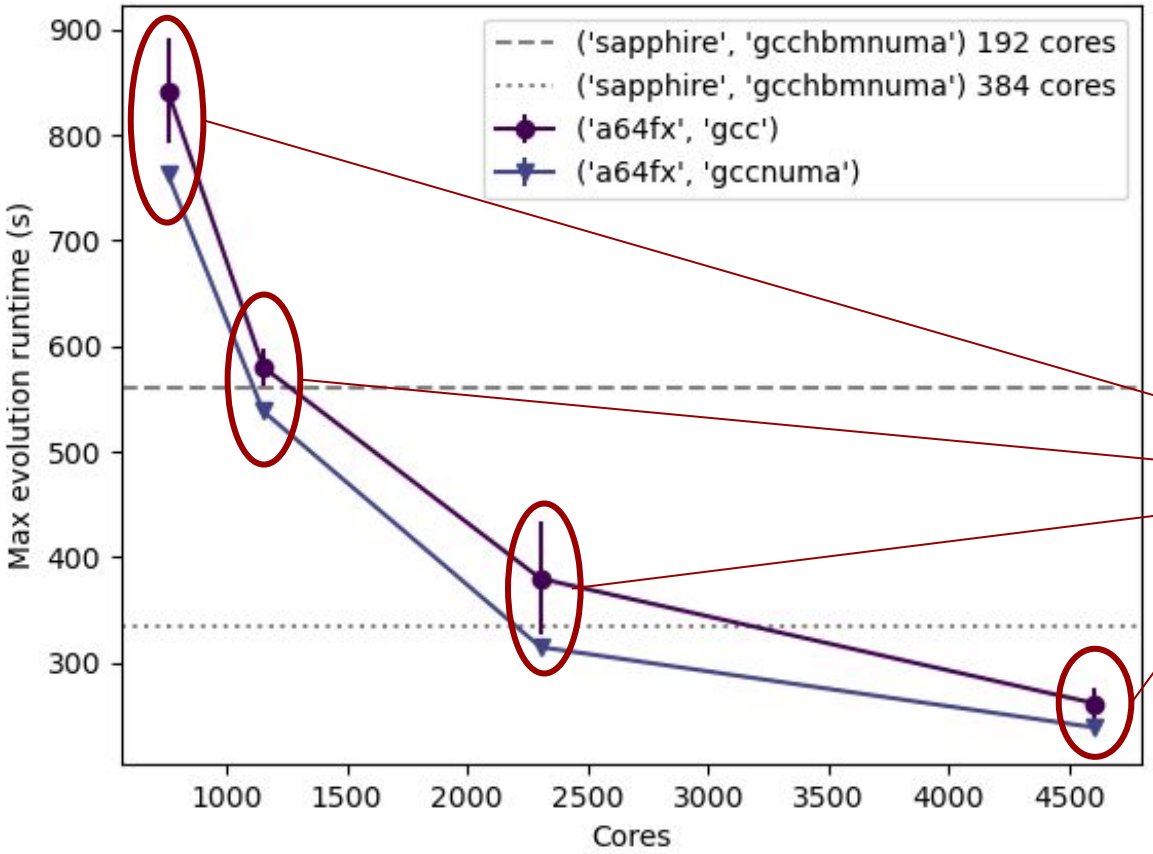
Ookami: A64FX-700 – Strong Scaling

Small 32 GB/node, need at least 21 nodes, 36/48 cores per node

Good scaling across the machine!



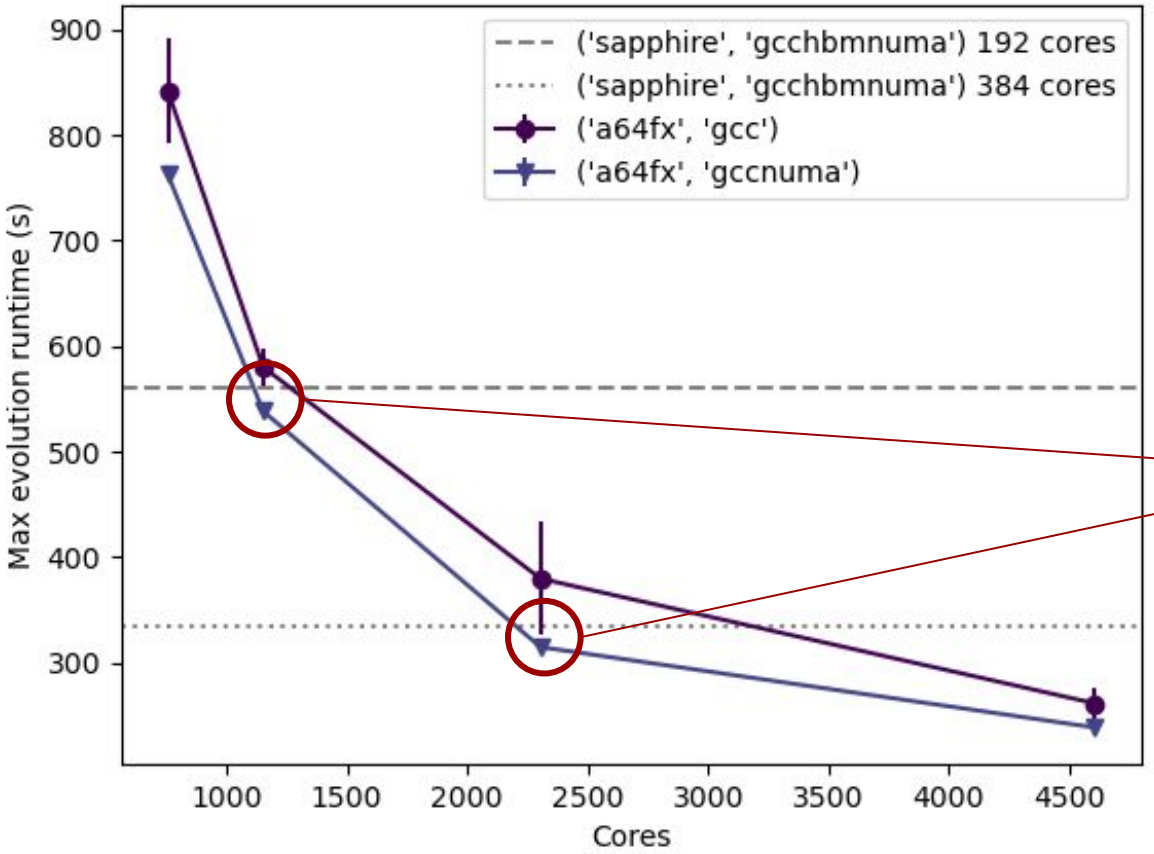
Ookami: A64FX-700 – Strong Scaling



`--map-by numa` decreases runtime variance, and overall runtime.

Explicitly placing memory closest to core that's using it = smaller, more consistent runtimes

Ookami: A64FX-700 – Strong Scaling



Comparable performance to SPR, but using 10x as many cores

What about efficiency?

Increasing # of cores → Corresponding decrease in runtime

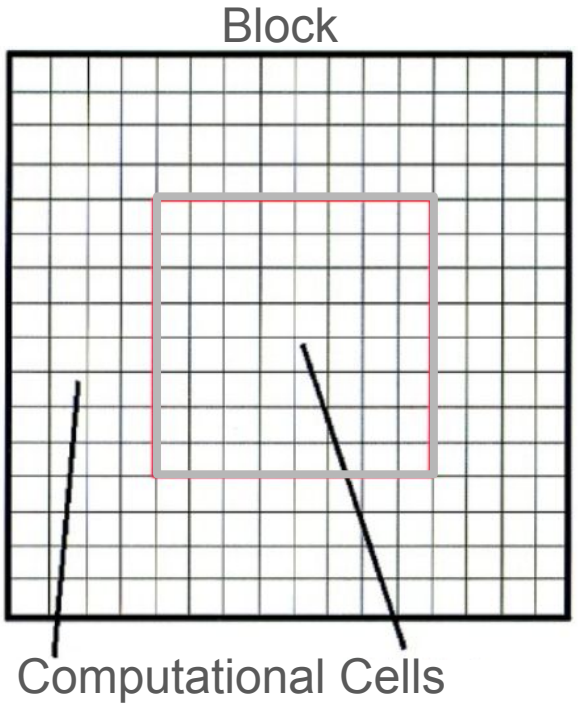
2x cores = ½ runtime

CPU time is scaled by 2* ½ = 1

Look at the **M**aximum **C**ell **C**PU **T**ime per **T**imestep (**MCCTT**)

MCCTT = Evolution runtime * # cores / # cells / # timesteps

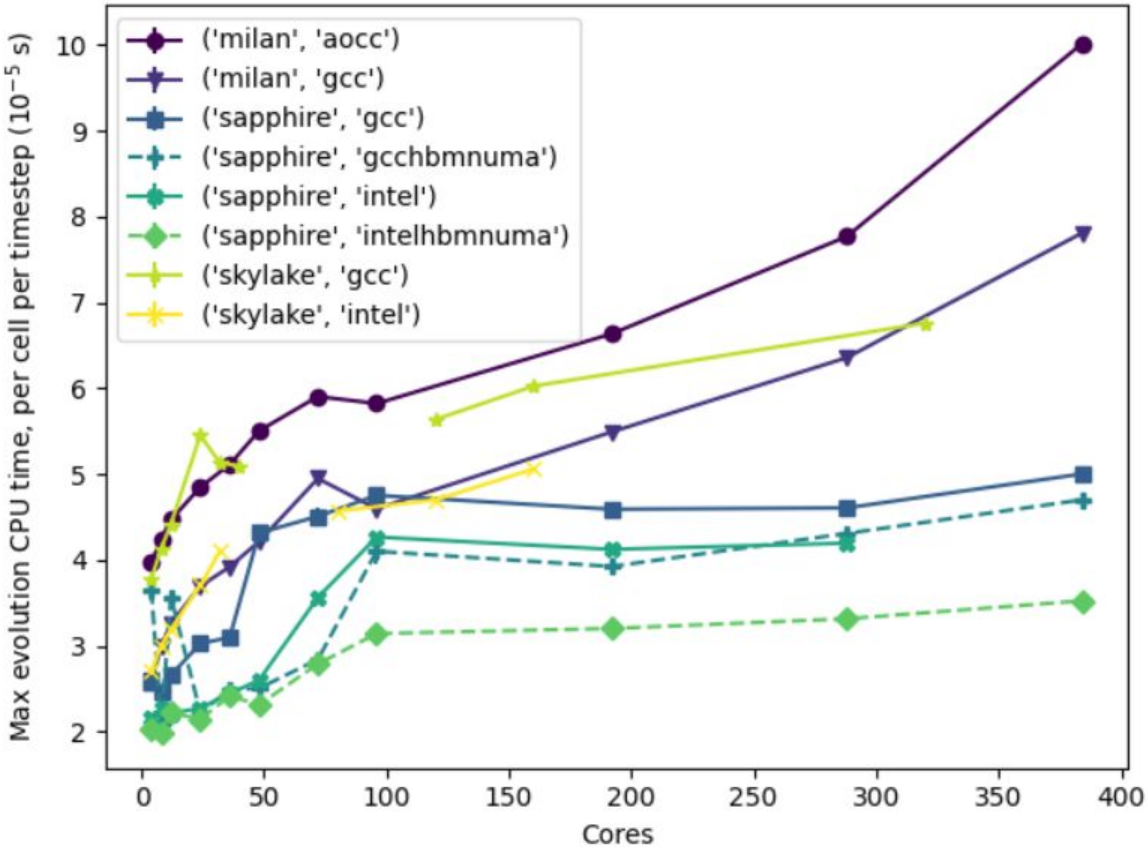
Ideally, MCCTT would remain **constant** as you increase # of cores



What about efficiency?

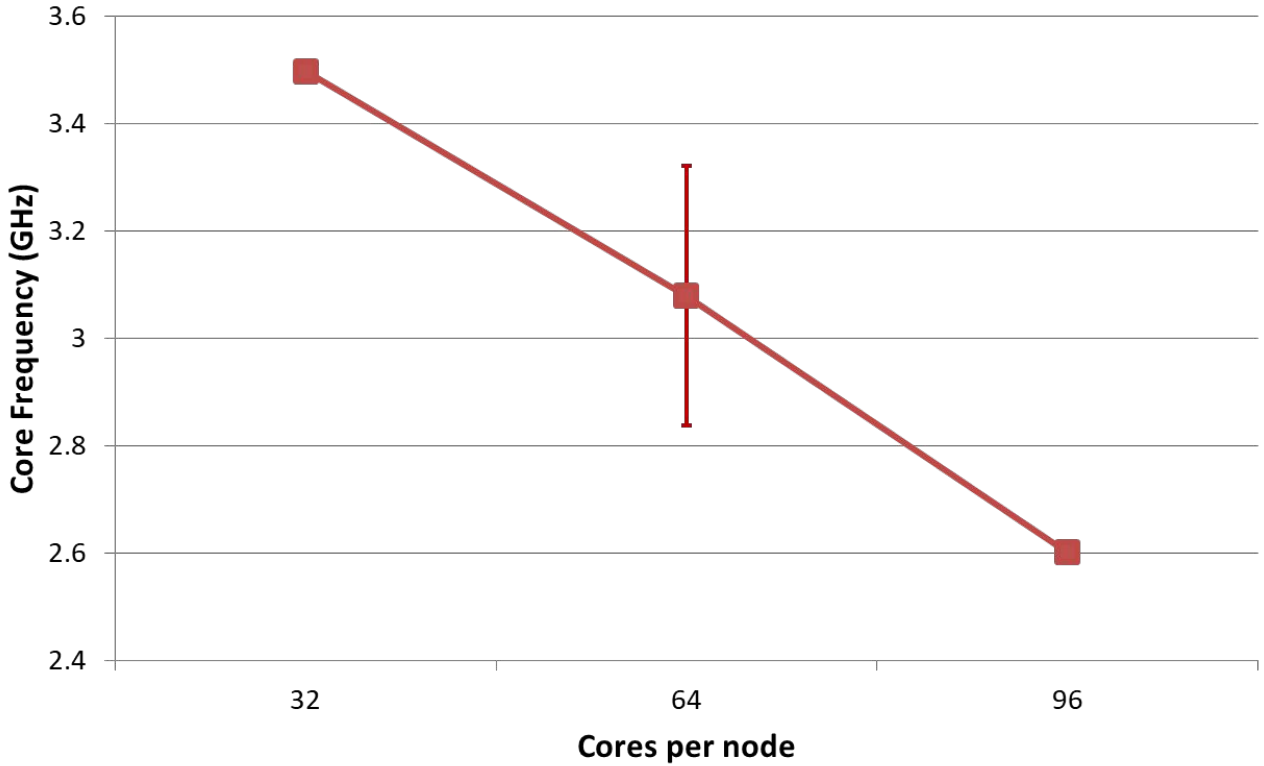
Interestingly, these all increase within a node?

SPR stays more level than Milan when going to multi-node – 4x faster interconnect?



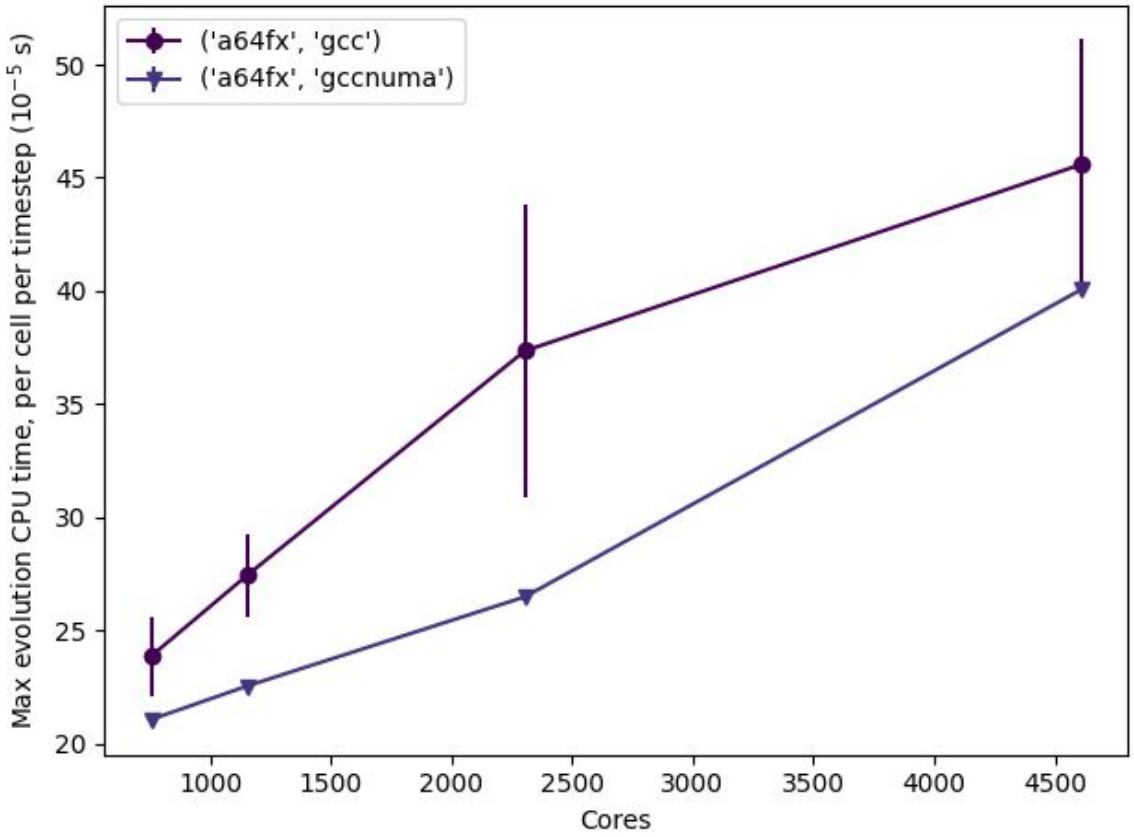
Changes in clock speed

On SPR, idle cores are held at 3.5 GHz. On Milan, idle cores have lower frequency
Using more cores decreases the clock speed



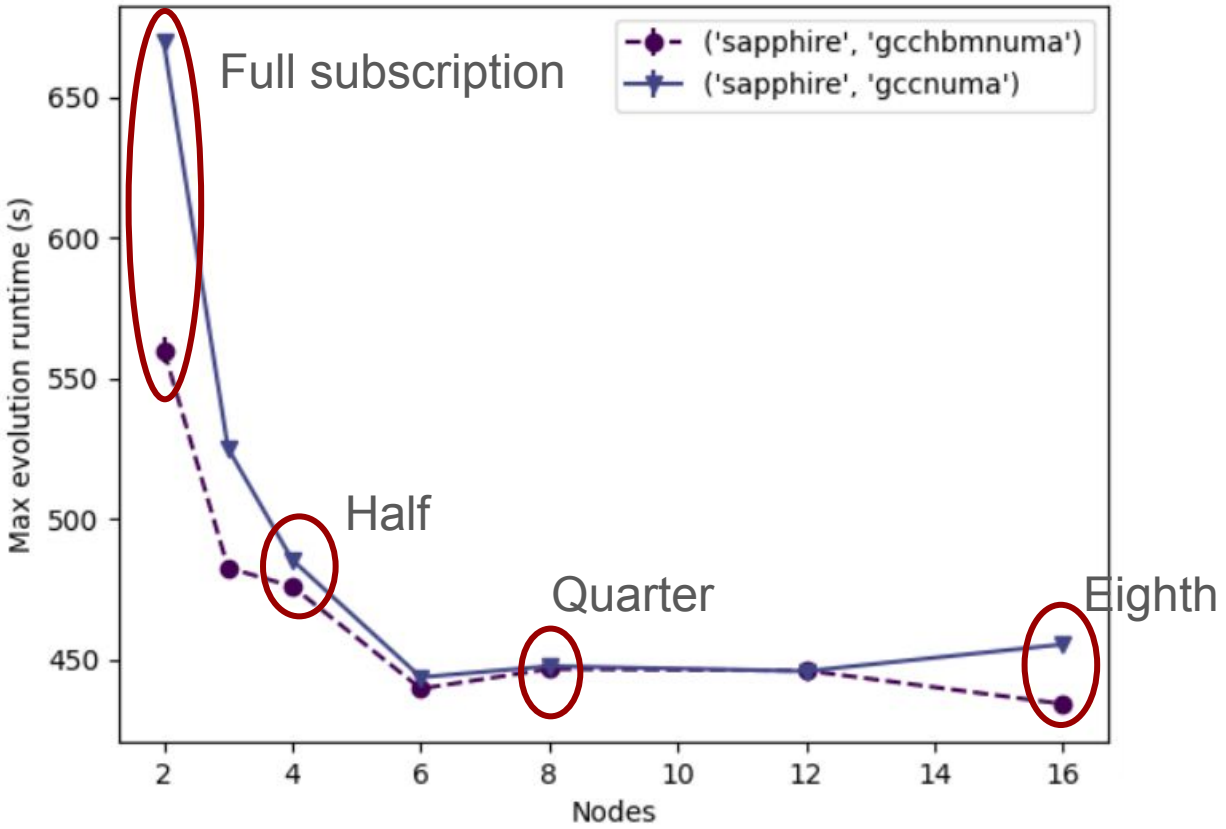
What about efficiency?

MCCTT 10x worse on Ookami, because you need to use 10x more cores



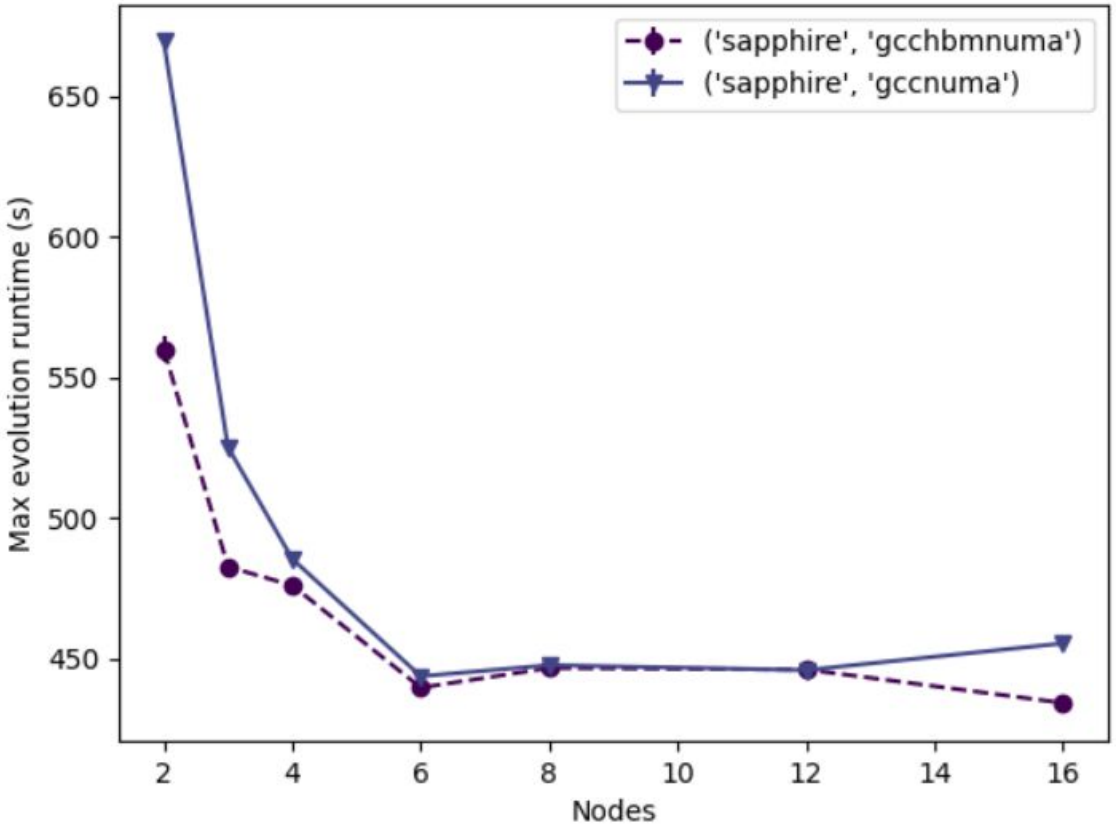
Core-spread

Increasing MCCTT gave the idea – what if we spread cores across nodes?
192 cores total, spread across nodes



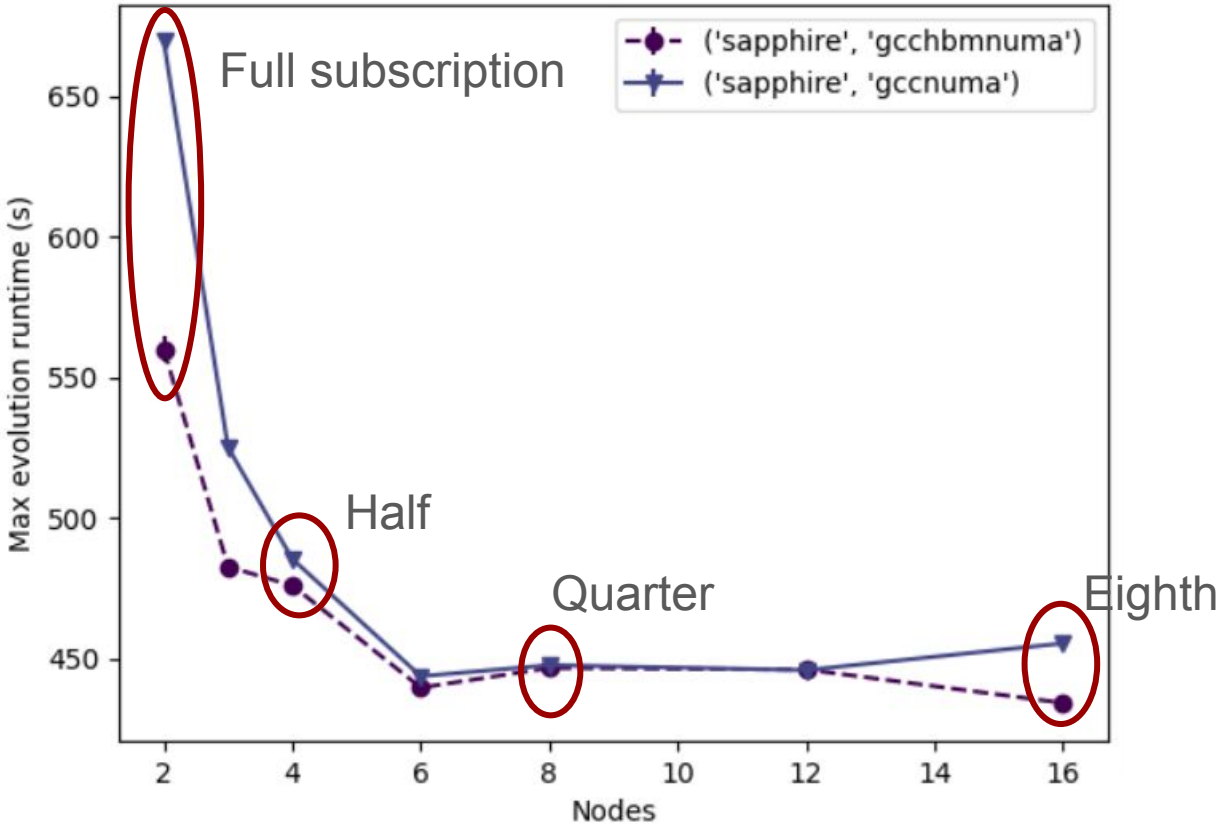
Core-spread

Increasing MCCTT gave the idea – what if we spread cores across nodes?
192 cores total, spread across nodes



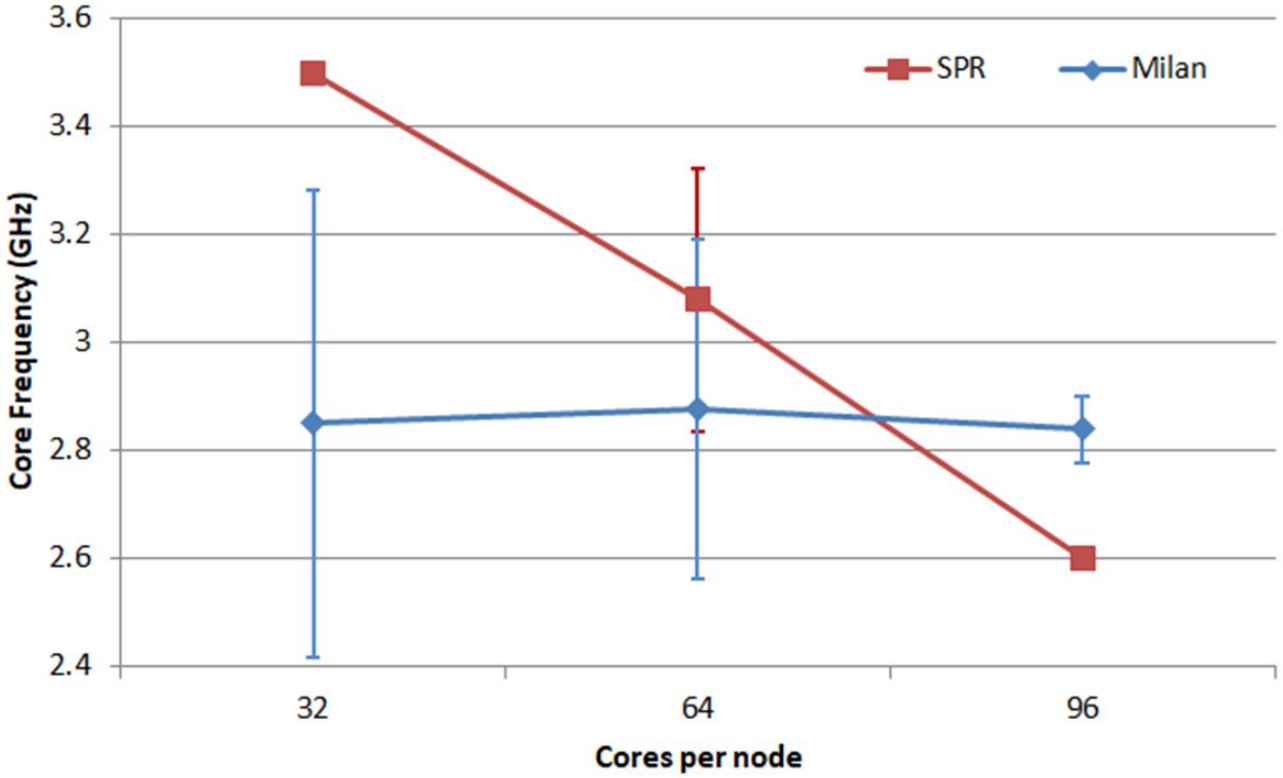
Core-spread

Using the **same # cores** spread across multiple nodes **improved performance!**
Diminishing returns, but from full to half subscription, 20% faster



Changes in clock speed

On SPR, idle cores are held at 3.5 GHz. On Milan, idle cores have lower frequency
Using more cores decreases the clock speed



What can this tell us?

The speedup isn’t just from the clock speed – core spread works on Ookami too, which has a fixed clock speed

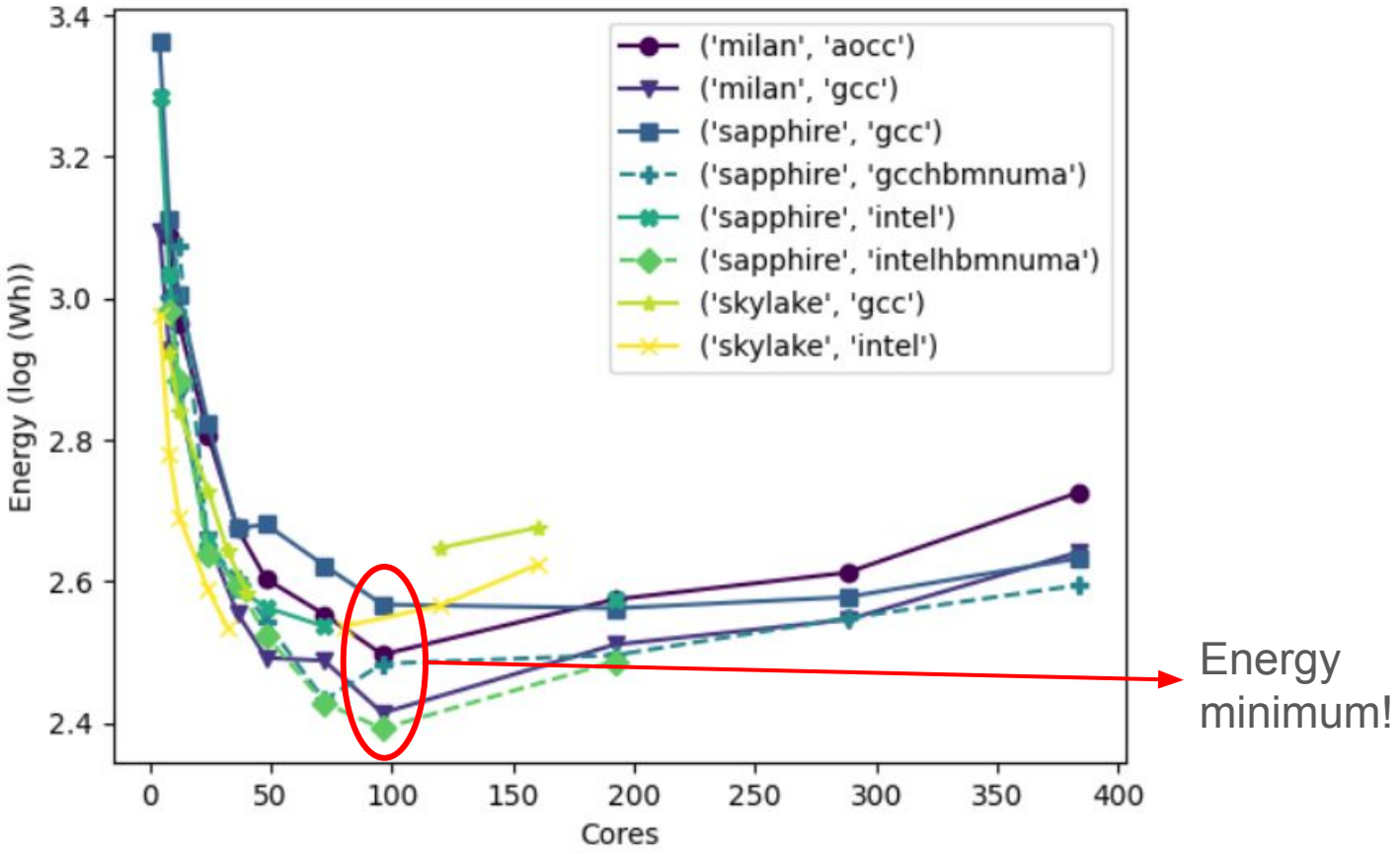
The more cores you use on a node, the more cores that share memory. So less cores = less memory contention

BUT this isn’t the most practical way to run our application because **using all available cores on a node is always faster**

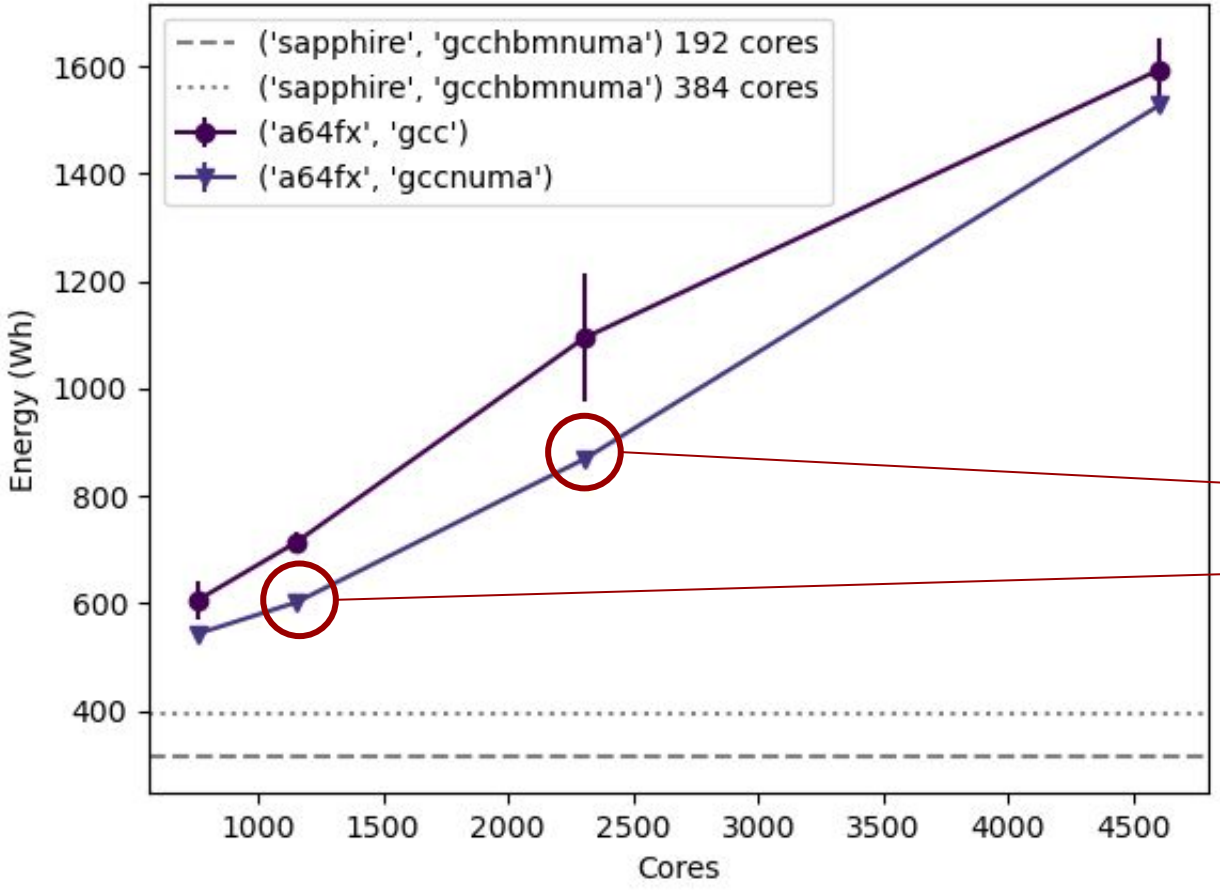
Bottom line: core spread is interesting, but not practical

Not the full picture – what about energy??

SeaWulf – Energy Consumption



Ookami: A64FX-700 – Energy Consumption



Power per node:

- SPR: 950 W
- A64FX: 120 W

Uses 1.5x more energy than SPR to achieve the same runtime

(Kind of impressive considering you're using 10x cores)

Conclusions

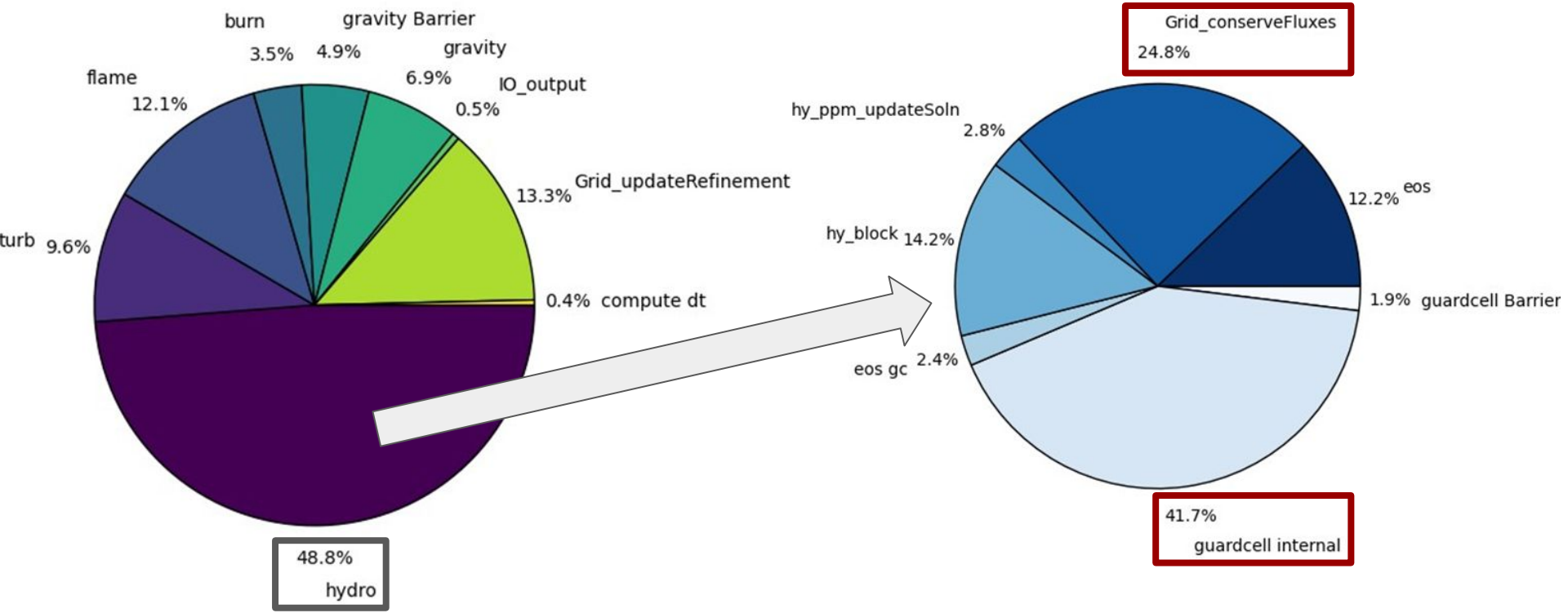
- When moving to 3D, communication rather than computation bottleneck
- HBM provides a speedup, but only in some cases
- Native Intel compiler is faster, but native AMD compiler is not
- Using `--map-by numa` produces faster and more consistent runtimes
- Spreading cores across nodes improves performance, but it's always better to use all cores on a node if you have them.
- A64FX can produce same performance as SPR, but with 10x cores
  - Needs 1.5x more energy
- Supercomputer Fugaku and A64FX-1000 may offer the best solution?
  - Custom Tofu-D interconnect for communication
  - 20% faster clock speed than A64FX-700
  - Additional assistant cores

Thank you to…

- Our IACS Collaborators!
- Dean Townsley for insightful discussion, especially with regards to data analysis and CPU frequencies
- Nikolay Simakov and Joseph White for help obtaining data from Ookami's XDMoD
- The SeaWulf computing system was made possible by NSF grants (#1531492 and #2215987) and the Ookami computing system by a $5M NSF grant (#1927880)
- FLASH 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
- This work was supported in part by the US Department of Energy (DOE) under grant DE-FG02-87ER40317 and by the National Science Foundation (NSF) under grant 1927880

Profiling

Communication is the bottleneck



Guardcell filling

Each 3D block has 16^3 = 4096 interior cells, and 24^3 - 16^3 = 9728 guard cells

4 guard cells deep on each side