Stony Brook University

Comparing OpenMP Implementations With Applications Across A64FX Platforms

Benjamin Michalowicz[1], **Eric Raut**[1], Yan Kang[1], Tony Curtis[1], Prof. Barbara Chapman[1,2], Dossay Oryspayev[2]

[1]Institute for Advanced Computational Science, Stony Brook University
[2]Brookhaven National Laboratory

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

IWOMP 2021
September 15, 2021

Exascal lab
Exascale Programming Models Laboratory

Motivation

- A64FX processor is relatively new -- limited studies on it
- Most reports established reproducibility of experiments compared to Fugaku-based reports
- OpenMP: had not been studied on the A64FX processor to our knowledge
- Opportunity to cover new ground and show subset of its behavior on the A64FX processor and how compilers may influence that performance

Agenda

- Ookami and Fugaku
- A64FX Processor
- Applications
  - PENNANT, SWIM, Minimod
  - Results of experiments following each application
    - Timings
    - Relative speedups
    - Insights from profiling
- Conclusions and future work

What is Ookami (“baby Fugaku”)?

- Test bed for NSF and US researchers
- First open deployment of the Fujitsu A64FX Post-K processor in US
  - ARM64 + SVE (scalable vector extensions)
  - High-bandwidth (HBM2) memory
- Possibly revolutionary new path to exascale emphasizing scientific productivity, performance, and energy efficiency
- New processor & very high-bandwidth memory promise performance of GPUs with programmability of CPUs

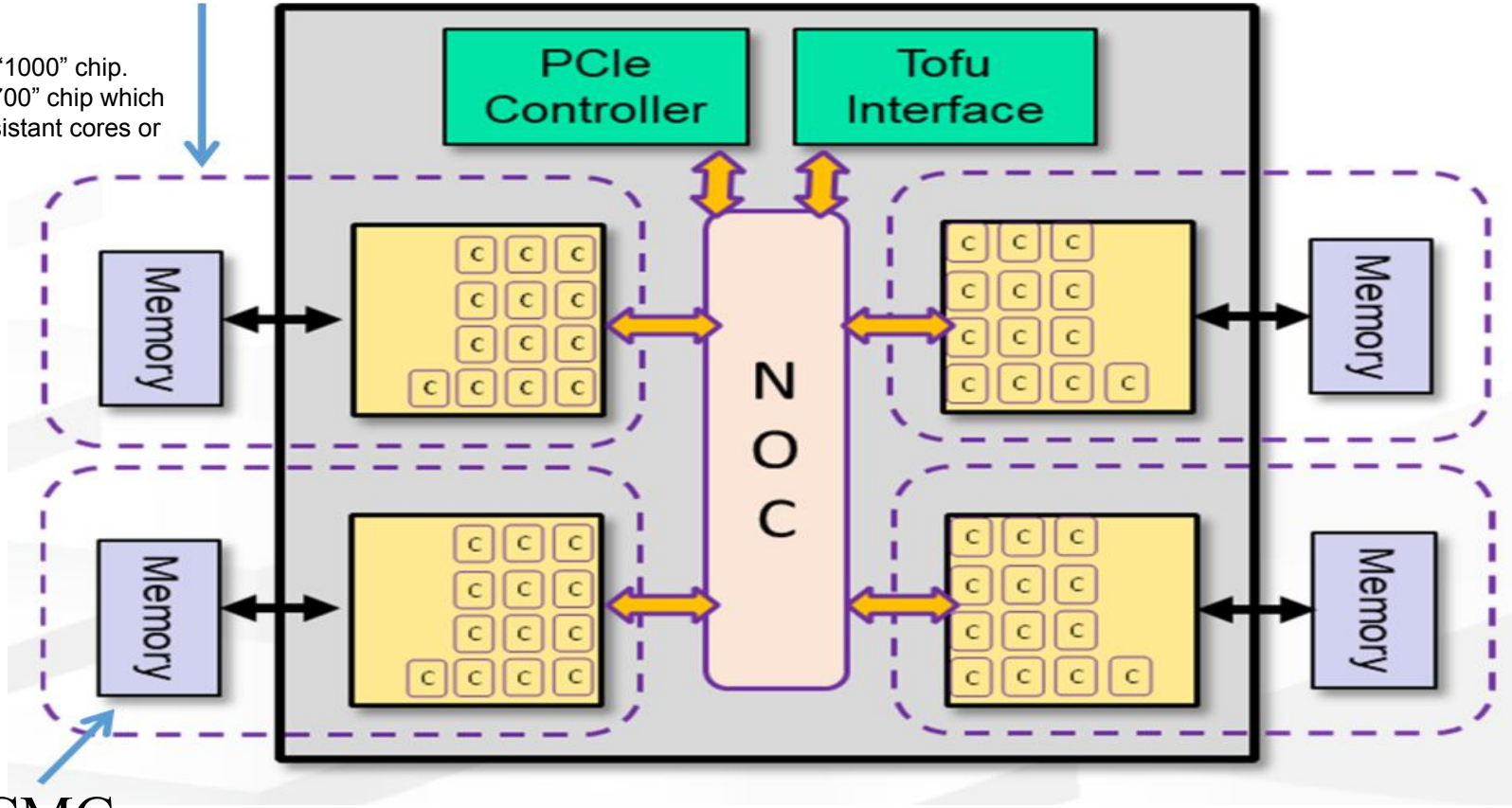
Ookami (狼) means wolf in Japanese --- an homage both to the origin of the processor and the Stony Brook seawolf mascot.

Ookami and Fugaku

- Ookami
  - First A64FX cluster to exist outside of Japan (@ Stony Brook University)
  - **174 A64FX nodes**, Intel Skylake, ThunderX2, AMD Milan nodes for other purposes
  - **FX700 chip (48 cores, 4 Core Memory groups/CMG's)**
- Fugaku
  - Current #1 on Top500
  - Almost 160,000 A64FX Nodes
  - **FX1000 chip (48 + extra cores for OS communication, 4 CMG's)**

A64FX NUMA node architecture

Diagram is of the “1000” chip. Ookami has the “700” chip which does not have assistant cores or the Tofu interface.



CMG – core memory group

http://www.jicfus.jp/jp/wp-content/uploads/2018/11/msato-190109.pdf

Compilers and flags used in our experiments

	Versions	
Compiler Family	**Fugaku**	**Ookami**
ARM	-	20.3
Cray	-	10.0.1
Fujitsu	4.3.0a	-
GCC	8.3.1, 10.2.1	8.3.1, 10.2.1, 11.0.0
LLVM	11.0.0	11.0.0, 12.0.0

Compiler	Flags
Cray	`-homp -hvector3 -hthread3`
GCC	`-mcpu=a64fx` `-Ofast -fopenmp`
LLVM	`-mcpu=a64fx` `-Ofast -fopenmp`
Fujitsu-Traditional	`-Nnoclang -Nlibomp -O3` `-KSVE,fast,openmp,ARMV8_2_A`

Fujitsu-LLVM	`-Nclang -Nlibomp -Ofast`
	`-Kfast,openmp`
	`-mcpu=a64fx+sve`

Experiment breakdown

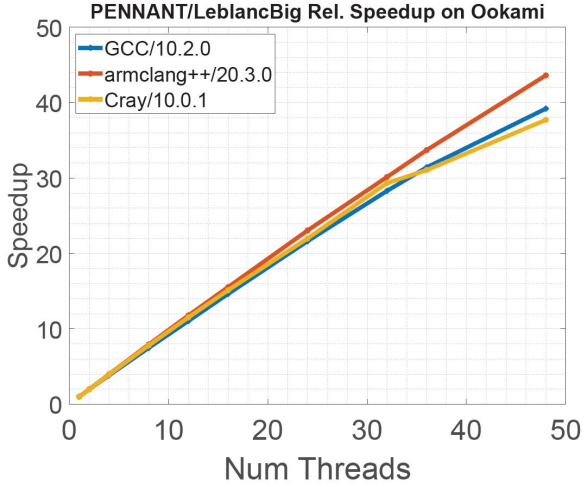
- Relying on intra-node/OpenMP-only communication for experiments
- Threads bound to cores
  - Evenly divided among CMGs
  - Use minimum number of CMGs needed for thread count
- Run application 5 times on each thread value and take arithmetic mean
- Performed for each of the “best-in-class” compilers

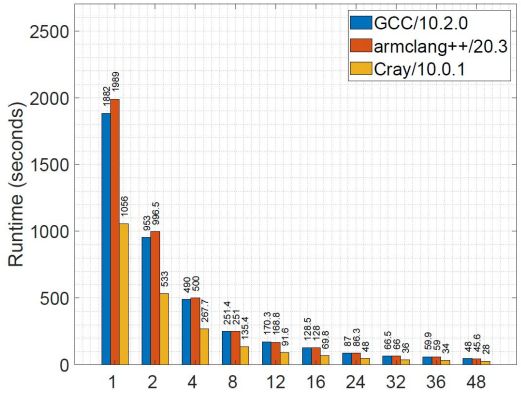
PENNANT

- Mesh physics application developed by LANL
  - Studies performance of unstructured physics
- Hybrid MPI/OpenMP application (static loop scheduling)
- Advanced architecture research dominated by pointer chasing and irregular memory accesses
  - Also features optimizations for SIMD and memory locality
- Our study: Dealing with Sedov and Leblanc meshes: structured, with square zones

PENNANT on Ookami - LeblancBig input

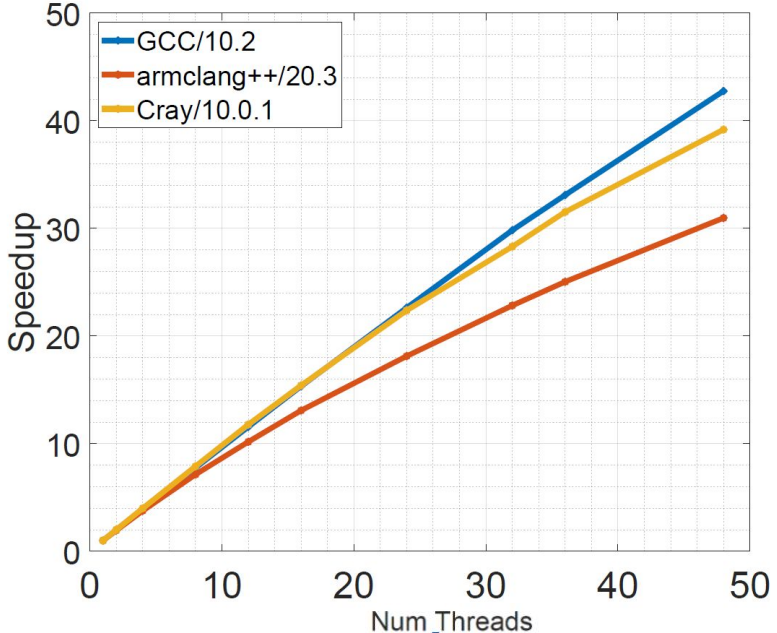
- ARM compiler achieves the most linear speedup here
- Cray compiler: use of SVE maximizes bandwidth use.
  - Leads to lesser speedup at higher thread values
  - Found in analysis of generated assembly code between each compiler

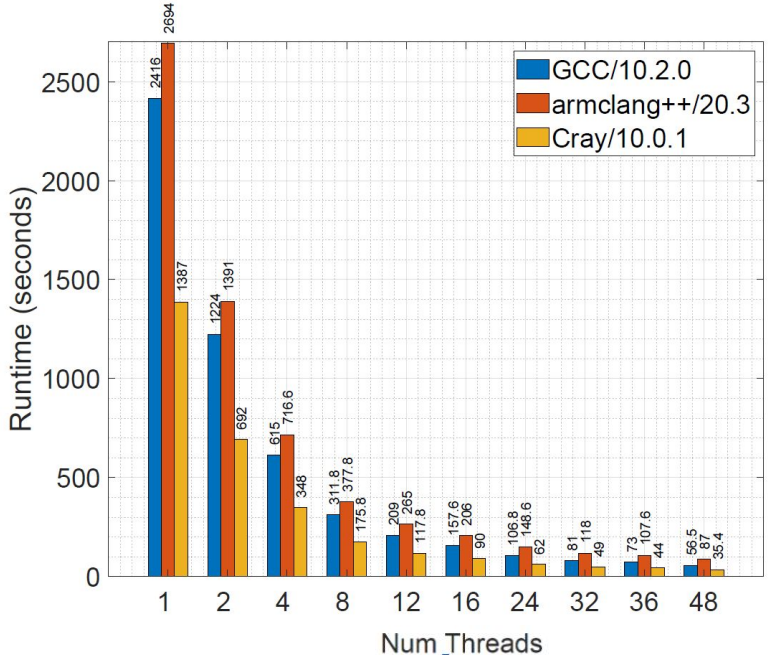




PENNANT on Ookami - SedovBig input

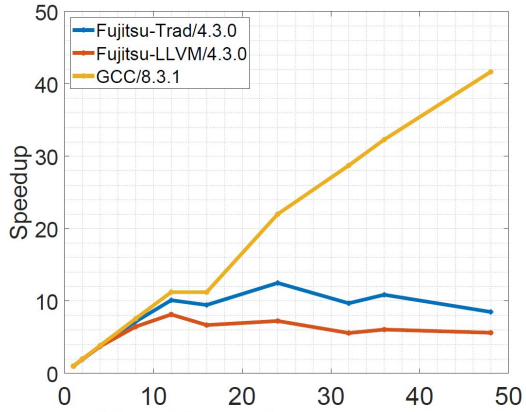
- The “SedovBig” mesh is mildly larger than that of the “LeblancBig” input (540x540 compared to 160x1440)
  - Left chart: Relative Speedup; Right char: Runtime of “best in class” compilers on Ookami as of 01/2021
  - Here armclang has worse speedup as well as performance

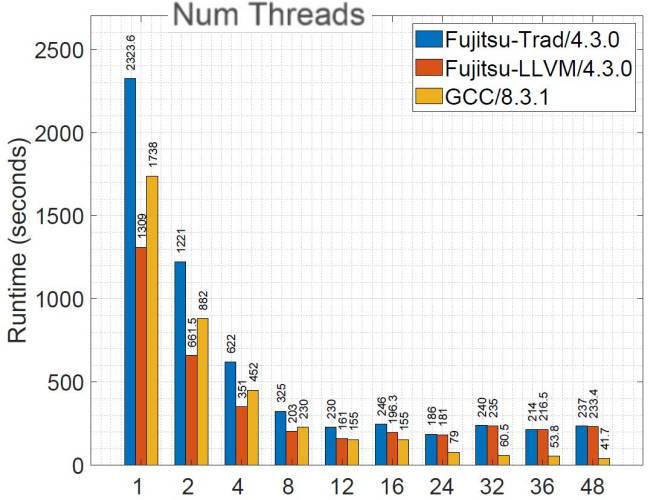




PENNANT on Fugaku - LeblancBig input

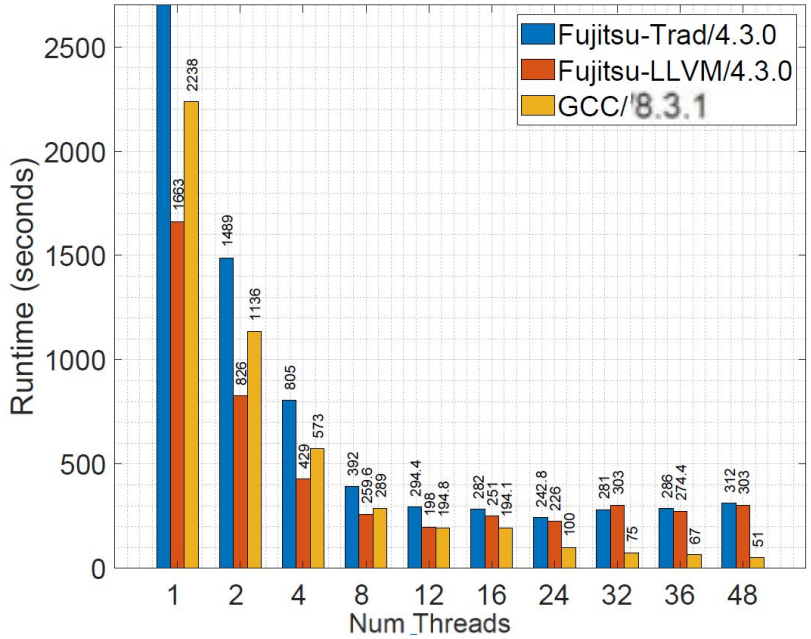
- Fujitsu Compilers: Smaller relative speedup after saturating 2+CMG's.
- GCC: maintains similar performance
  - Deviates greatly from Fujitsu performance as more OpenMP threads requested
- PENNANT is NUMA-aware
- Profiling: traditional Fujitsu backend takes longer, but executes a higher amount of GFLOPS than LLVM backend

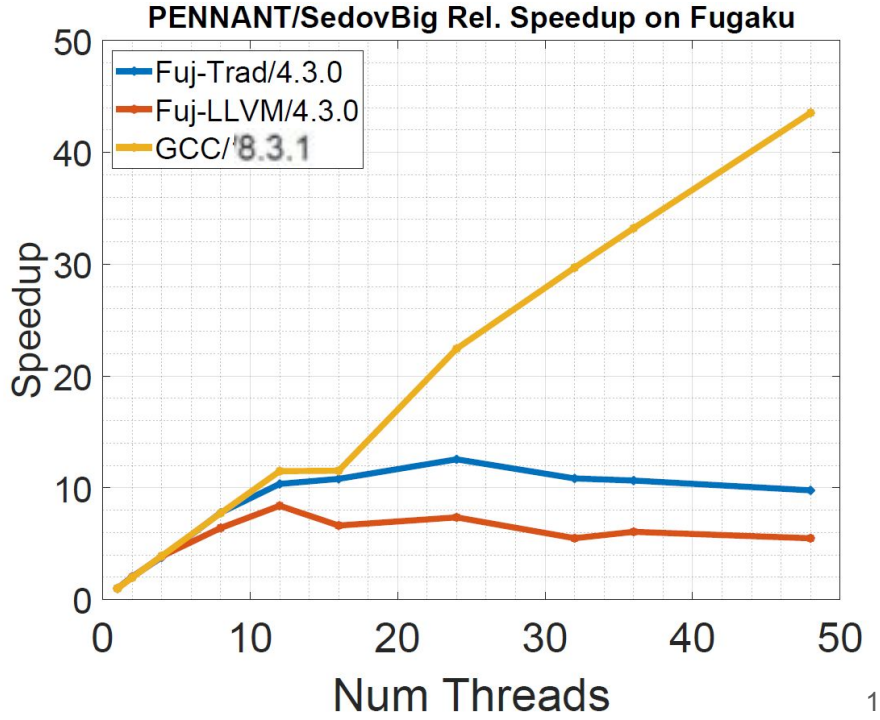




PENNANT on Fugaku - SedovBig input

- Runtime trends are more defined with larger meshes (e.g. SedovBig), though the speedup trends stay consistent.



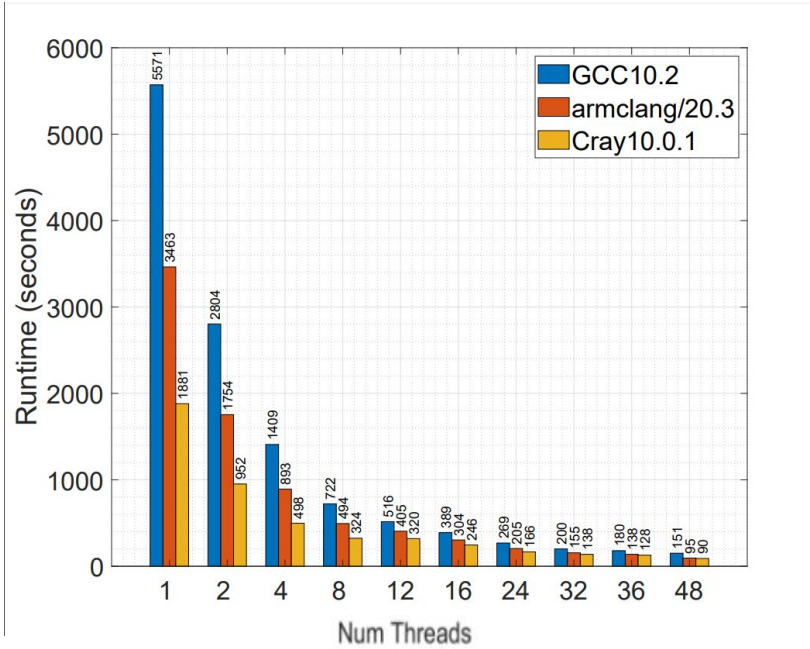


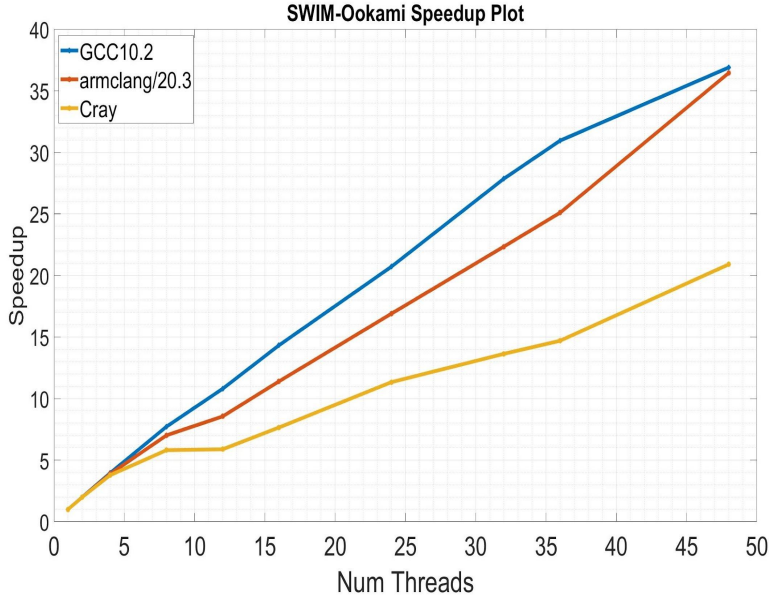
SWIM

- Weather-prediction modeling, solving the shallow-water equations using finite difference methods. Tracks FP speed, bandwidth, and cache characteristics
- OpenMP and Fortran-based benchmark
- Testing with Ookami compilers:
  - armflang (arm compiler/20.3 (based on LLVM 9.0.1))
  - armgfortran (arm compiler/20.3 (it is gcc 9.3.0))
  - Crayftn (10.0.1)
- “Ref” problem size (7701 by 7701 matrix running 3000 iterations)

SWIM on Ookami

- The Cray compiler obtained the lowest speedup but the fastest runtime performance.
- It is generally more efficient on the A64FX processor than the armclang and GCC compilers.
  - Likely due to superior use of SVE





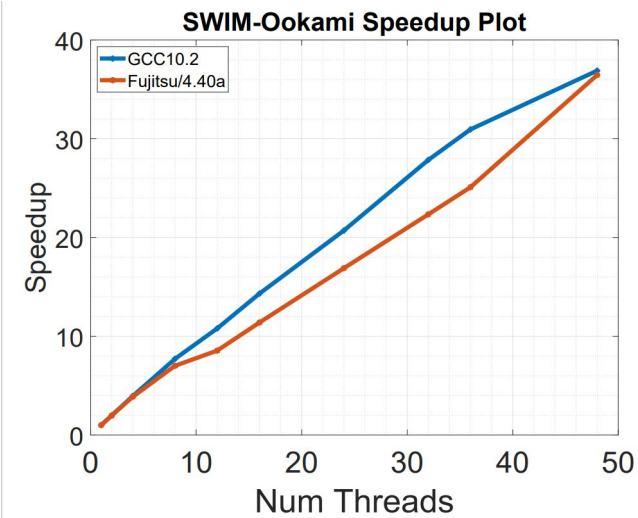
SWIM on Fugaku

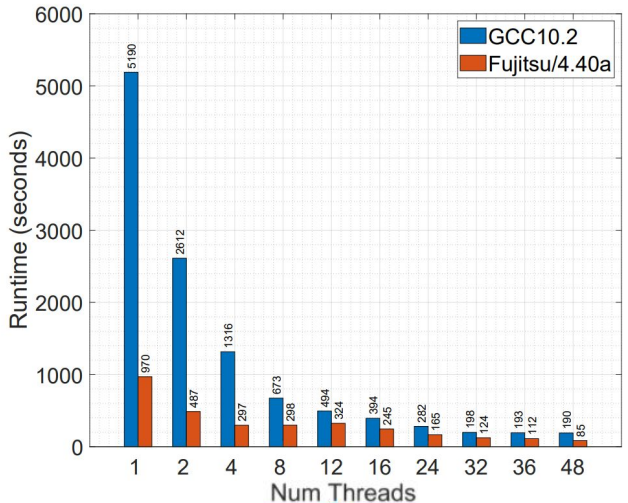
The Fujitsu compiler obtains performance superior to the GNU compiler.

GNU compiler seems to have a greater speed up than the Fujitsu compiler. 36 threads achieve a 32x speed-up over 1 thread. The Fujitsu compiler only obtained 25x speed up with 36 threads over 1 thread. The Fugaku-based runs show a drop in relative speedup starting at 8 OpenMP threads before leveling out at 12 threads. It could be explained by the insufficient optimization of Fujitsu compiler targeting OpenMP applications.

NUMA optimization critical for Fujitsu compiler performance:

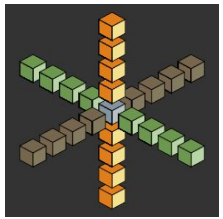
- XOS_MMM_L_PAGING_POLICY=demand:demand:demand for multiple CMGs





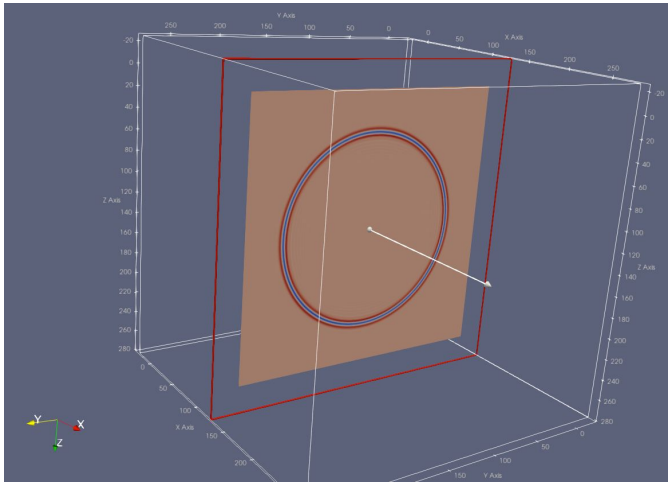
Minimod: Geophysics Exploration

- Wave equation important to many geophysics applications
- One simple solution method is finite difference (FD); involves stencil computation



$$\frac{1}{v_p^2}\frac{\partial^2 p(\mathbf{x},t)}{\partial t^2} - \nabla^2 p(\mathbf{x},t) = f(\mathbf{x},t),$$

- *Minimod*: wave propagation mini-app developed by TotalEnergies
  - Designed to test new and emerging programming models and hardware
  - Contains OpenMP loop-based and task-based implementations

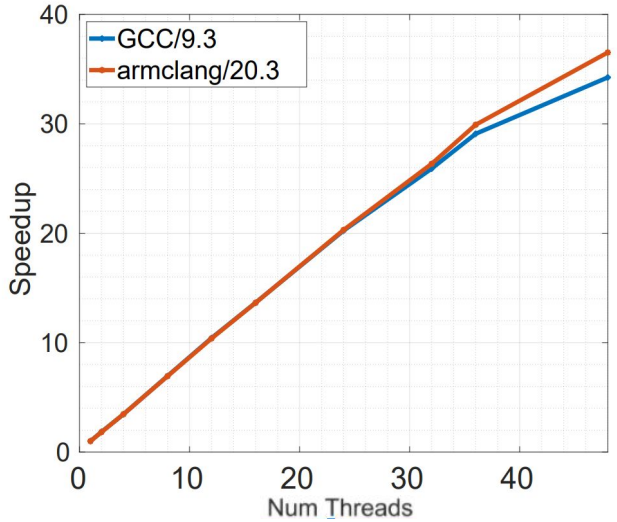


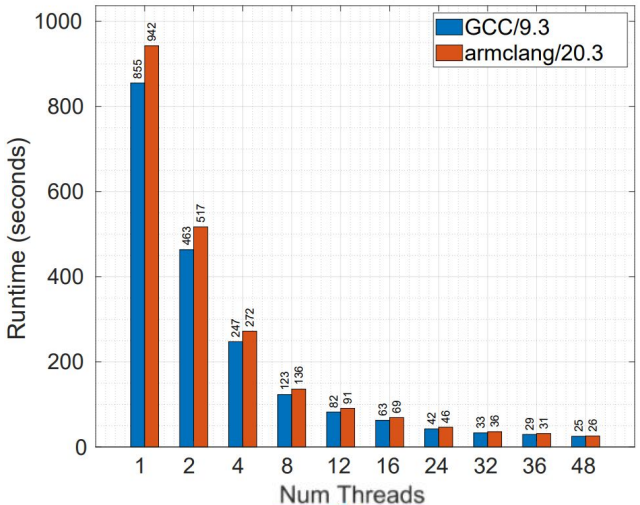
Numerical validation

arXiv:2007.06048 [cs.DC]

Minimod on Ookami - Loop-XY

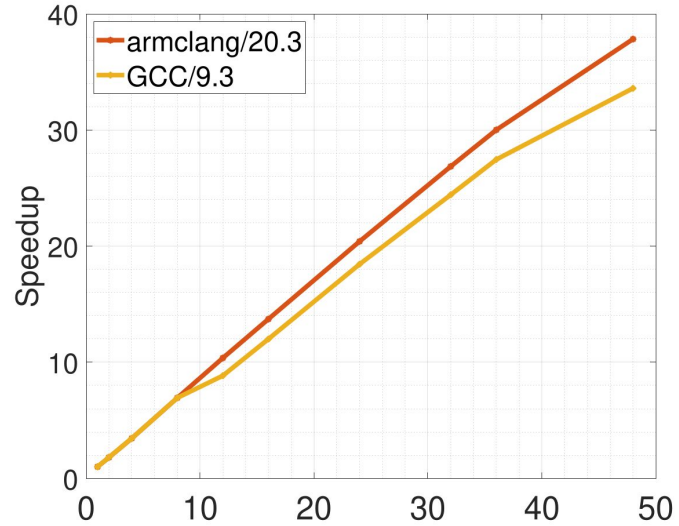
- Speedup of Loop-XY variant is shown.
- The Cray compiler fails to compile this code, resulting in an internal compiler error
- GCC achieves better absolute performance at smaller thread counts than armclang
- At large thread counts, absolute performance is similar between the two compilers; armclang has slightly better speedup

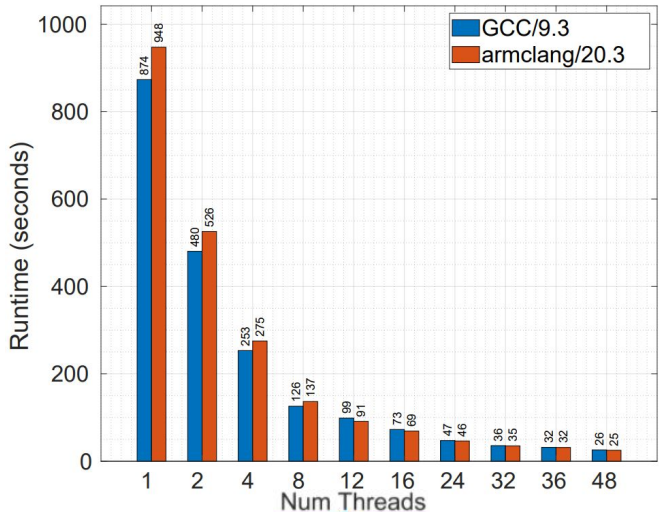




Minimod on Ookami - Tasks-XY

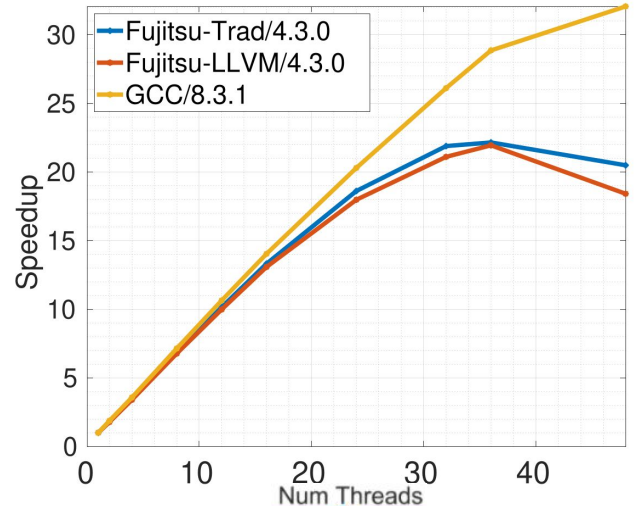
- Speedup of tasks-xy variant is shown.
- Trends are similar to the Loop-XY case.
- Profiling: application (in both configurations) spends almost the entire time in OpenMP regions and is highly memory bound (~80% stalled cycles)
  - Expected behavior for a stencil application

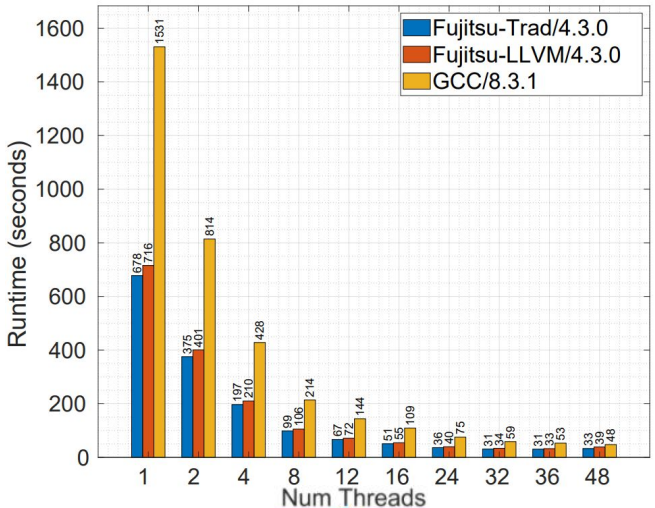




Minimod on Fugaku - Loop-XY

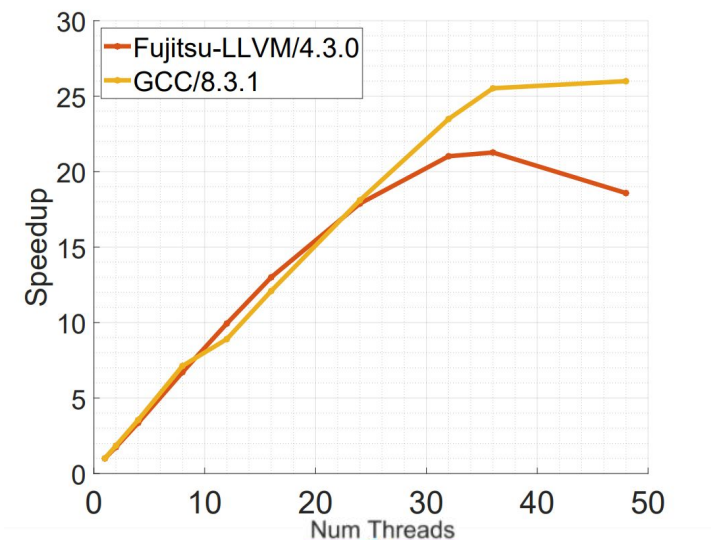
- Speedup of loop-xy variant is shown
- GCC has very poor absolute performance at smaller thread counts, but achieves better scaling
  - Likely due to lack of A64FX/SVE support in GCC 8
- Traditional and LLVM backends have similar performance
- Absolute performance of all compilers is similar at 48 threads

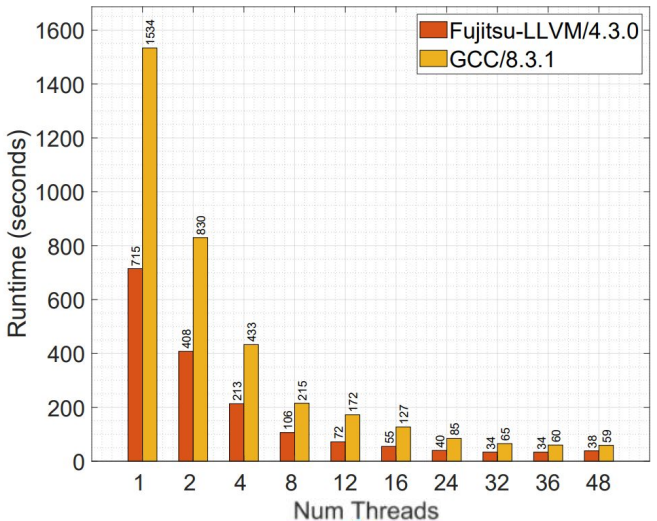




Minimod on Fugaku - Tasks-XY

- Fujitsu compiler (trad mode) fails to run task-based variant due to lack of task dependency support
- Trends for other compilers are similar to the Loop-xy variant





Conclusions and future work

- We evaluated three benchmark applications with different compilers across two A64FX platforms
- In general, Fujitsu and Cray compilers show the best performance on this platform
  - GCC versions used here don't support A64FX/SVE -- the latest GCC has support
  - Scaling issue seen in some Fujitsu compiler experiments
    - Perhaps due to issues with NUMA awareness
  - Cray compiler fails to compile/run some applications (e.g., Minimod)
- Future work
  - On Pennant/SWIM: Analysis of different inputs and mesh types (structured versus unstructured/ zone shapes)
  - On Minimod: Task-based parallelism via OpenMP Tasks, and MPI parallelism
  - Applications featuring other OpenMP directives (data-sharing attributes, SIMD clauses, etc.)

Acknowledgements

We would like to thank the NSF for supporting the Ookami cluster, and the ability to research the A64FX processor by Riken and Fujitsu, through grant OAC 1927880. We would like to thank the Riken Center for Computational Science for providing us with accounts to use the Fugaku supercomputer and conduct research on it. We would also like to thank Stony Brook University and the Institute for Advanced Computational Science for providing the resources to allow us to conduct our studies on Ookami. Finally, we would like to thank TotalEnergies Exploration and Production Research and Technologies for their support of experimentation using MiniMod.

Stony Brook University

Thank you!

Comparing OpenMP Implementations With Applications Across A64FX Platforms

Benjamin Michalowicz[1], **Eric Raut**[1], Yan Kang[1], Tony Curtis[1], Prof. Barbara Chapman[1,2], Dossay Oryspayev[2]

[1]Institute for Advanced Computational Science, Stony Brook University
[2]Brookhaven National Laboratory

iACS INSTITUTE FOR ADVANCED COMPUTATIONAL SCIENCE

IWOMP 2021
September 15, 2021

Exascal lab
Exascale Programming Models Laboratory