A64FX – Your ~~Path~~ Compiler You Must Decide!

[1]

Jens Domke, Dr. rer. nat.
< jens.domke@riken.jp >
High Performance Big Data Research Team, RIKEN R-CCS, Kobe, Japan

RIKEN R-CCS Computer simulations create the future

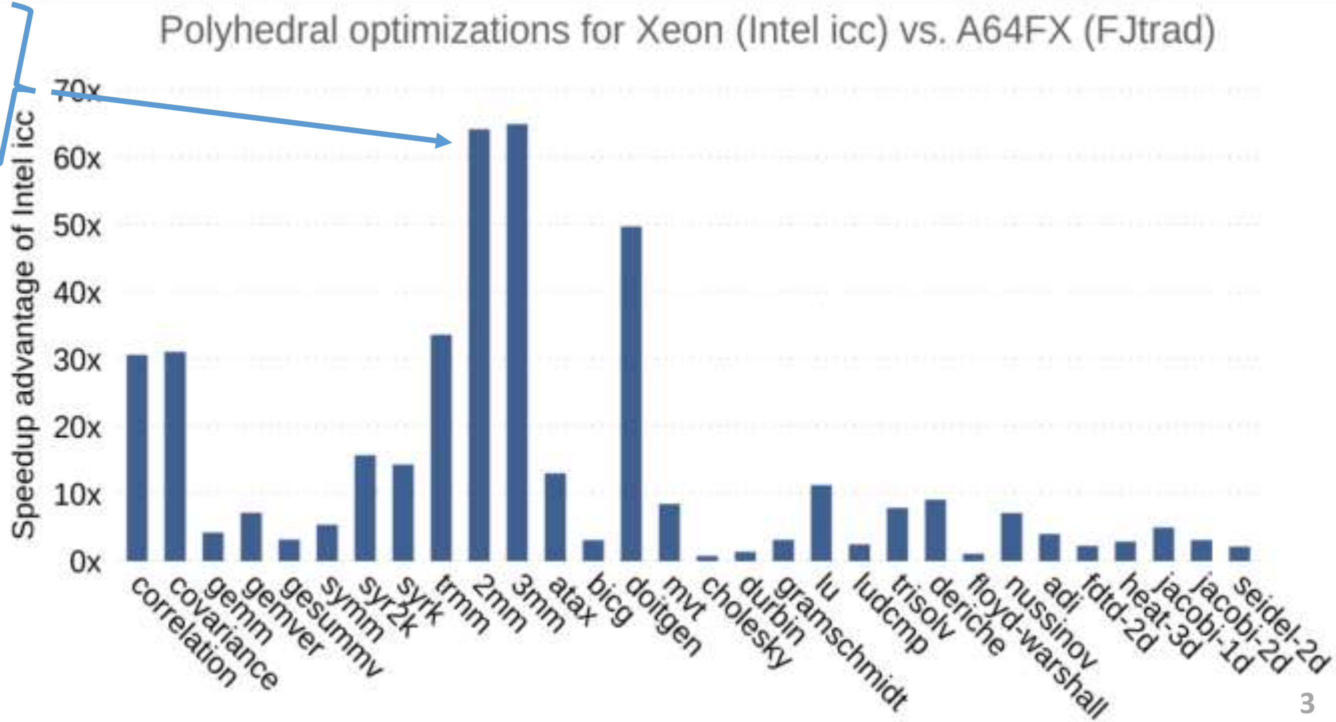
Outline

- Motivation for this Study
- Measurement Methodology
  - Compiler Selection
  - HPC Workloads
- Discussion of Fugaku's Results
- Summary, Conclusion, Future Work

Unexpected advantage of Xeon vs. A64FX?

- Comparing PolyBench (s. later) against Intel Xeon E5-2650v4
- Xeon core w/ less than ½ of a A64FX core's theoretical peak
- ***2mm* is matmul**
- **A64FX (w/ fcc) 64x slower???**

Reason:
Intel's icc applies **Loop Reordering** and other polyhedral optimizations!



Other Reports and Research Questions

- **Performance portability** (x86→A64FX) not easy to achieve
  - Fujitsu compilers and 4 MPI + 12 OMP threads not always best?
    - A. Poenaru, "An Evaluation of the Fujitsu A64FX for HPC Applications," Presentation in AHUG WS ISC 21; and
    - B. Michalowicz et al., "Comparing the behavior of OpenMP Implementations with various Applications on two different Fujitsu A64FX platforms," in PEARC '21, 2021; and
    - E3SM Pathfinding on Fugaku: https://e3sm.org/e3sm-pathfinding-on-fugaku/; etc... ➔ GNU better than FJ's compiler
- **Research Question:**
  - **Q1:** Is **recommended usage model**, i.e., compiler+flags and MPI/OMP config, ideal or **just a starting point?**
  - **Q2:** Is there a **"silver bullet" compiler** choice for A64FX?
  - **Q3:** Can **performance differences**, compared to similar x86-based hardware, be **attributed to the compiler?**

Compilers for our Measurements

Simple Idea: Throw many science codes at compilers and look for trends!

Three compilers and five variations:

- **FUJITSU** Software Technical Computing Suite (v4.5.0):
  - ***FJtrad*** (traditional mode) and ***FJclang*** (based on LLVM 7)
  - -Kfast,ocl,largepage,lto additionally to benchmark's individual flags
- **LLVM** Compiler Infrastructure (v12):
  - ***native*** (-Ofast -ffast-math -flto=thin) and ***polly***; with Fujitsu's frtr for Fortran
- **GNU** Compiler Collection (v10.2.0):
  - -O3 -march=native -flto additionally to benchmark's individual flags

- **Alternatives**: Arm and HPE/Cray unavailable on Fugaku at time of writing ☹

Testing >100 Kernels and HPC Workloads

- RIKEN's **FS2020 TAPP-kernels** (micro kernels)
  - 22 kernels from RIKEN's Priority Issue Target Applications
  - OMP-para. kernels of FFB, GENESIS, NICAM, QCD, etc.; Target: 1 CMG
- Polyhedral Benchmark suite (in short, **PolyBench**)
  - 30 single-threaded, simple scientific kernels written in C
  - Input sizes [*Mini*→*ExtraLarge*] to target diff. memory levels (we use *Large*)
- TOP500 benchmarks: **HPL and HPCG** (default version)
  - Used by community for world-wide supercomputer ranking
- Additionally: **BabelStream** and **Dlproxy** (GEMM-based convolution)

Testing >100 Kernels and HPC Workloads

Computer simulations create the future

- Exascale Computing Project **(ECP) Proxy Applications**
  - Used for procurement of exascale systems by HPC centers in USA
  - Version 1.0 contains 12 workloads (we excluded CANDLE)
- RIKEN CCS' **Fiber Miniapp Suite**
  - 8 proxy apps used in procurement of Fugaku (we excluded NGSA)
  - Represent the priority areas of the Japanese government
- **SPEC Benchmarks** (**CPU 2017[speed]** V1.1 & **OMP 2012** V1.1)
  - Widely accepted benchmark set for industry and HPC vendors (use *train*)
  - Single-threaded: CPU[speed] Integer
  - OMP-parallelized: CPU[speed] Floating Point and SPEC OMP

Overview of our 55 traditional HPC Workloads

Set	Name	Sci. / Eng. / AI Domain	Name	Sci. / Eng. / AI Domain	Name	Sci. / Eng. / AI Domain
TOP500	HPL	Math/Computer Science	HPCG	Math/Computer Science		
ECP	AMG	Physics and Bioscience	miniAMR	Geoscience/Earthscience	SW4lite	Geoscience/Earthscience
	CoMD	Material Science/Engineering	miniFE	Physics	SWFFT	Physics
	Laghos	Physics	miniTRI	Math/Computer Science	XSBench	Physics
	MACSio	Math/Computer Science	Nekbone	Engineering (Mechanics, CFD)		
RIKEN	FFB	Engineering (Mechanics, CFD)	mVMC	Physics	NTChem	Chemistry
	FFVC	Engineering (Mechanics, CFD)	~~NGSA~~	~~Bioscience~~	QCD	Lattice QCD
	MODYLAS	Physics and Chemistry	NICAM	Geoscience/Earthscience		
SPEC CPU	~~blender(R)~~	~~Math/Computer Science~~	exchange2	Artificial Intelligence	omnetpp	Math/Computer Science
	~~cam4(R)~~	~~Geoscience/Earthscience~~	fotonik3d	Physics	perlbench	Math/Computer Science
	~~namd(R)~~	~~Material Science/Engineering~~	gcc	Math/Computer Science	pop2	Geoscience/Earthscience
	~~parest(R)~~	~~Bioscience~~	imagick	Math/Computer Science	wrf	Geoscience/Earthscience
	~~povray(R)~~	~~Math/Computer Science~~	lbm	Engineering (Mechanics, CFD)	roms	Geoscience/Earthscience
	bwaves	Physics	leela	Artificial Intelligence	x264	Math/Computer Science
	cactuBSSN	Physics	mcf	Math/Computer Science	xalancbmk	Math/Computer Science
	deepsjeng	Artificial Intelligence	nab	Material Science/Engineering	xz	Math/Computer Science
SPEC OMP	applu331	Engineering (Mechanics, CFD)	fma3d	Physics	mgrid331	Engineering (Mechanics, CFD)
	botsalgn	Bioscience	ilbdc	Engineering (Mechanics, CFD)	nab	Chemistry
	botsspar	Math/Computer Science	imagick	Math/Computer Science	smithwa	Bioscience
	bt331	Engineering (Mechanics, CFD)	kdtree	Math/Computer Science	swim	Geoscience/Earthscience
	bwaves	Engineering (Mechanics, CFD)	md	Material Science/Engineering		

Measurement Methodology and Environment

- **2-stage approach for each BM/compiler combination**
  - Check benchmark for **strong-scaling** runs (☹ none for MiniAMR/XSBench) (➔ important for fair comparison!)
  - **Identify kernel/solver** section ➔ wrap with additional instructions for timing
  - **Find "optimal" #MPI + #OMP** configuration for each benchmark+compiler (try under-/over-subscr.; each 3x runs; "best" based on time, or Gflop/s, etc.)
  - **Run 10x** of "best" configuration for lowest **time-to-solution metric**
- **Single-node experiments on Fugaku**
  - **Disabled power-saving** features, but **otherwise default env** variables (**exc. SPEC**: XOS_MMM_L_PAGING_POLICY=demand:demand:demand XOS_MMM_L_ARENA_LOCK_TYPE=0)
  - Rank & thread placement (**spread & close**, resp.) controlled by Fujitsu MPI
  - Benchmark **files cached to first-layer storage** (SSD shared by 16 nodes)

Results for FS2020 Micro Kernels

Computer simulations create the future

- **Gain** $\Delta = {}^{T_{FJtrad}}/_{T_X} - 1$ colored in range [-1 , +1]; **Bold name**: ≥2x speed-up
- Prog.lang in [ ] after name; Run config. [#MPI | #OMP] in headline or in cells
- Dark pink: unsolvable compiler/runtime error

➔ **Fjtrad w/ mostly better** results

➔ **GNU beats Fjtrad in 4 tests** but also runtime errors (6x)

➔ Using always "best" compiler ➔ **17% avg. runtime reduction**

Time-to-Solution [in s] (FJtrad) and Relative Performance Gain (others)

	runtime	Micro Kernels (all with [1|12])			
	FJtrad	**FJclang**	**LLVM**	**LLVM+Polly**	**GNU**
Kernel 1 [C]	0.001	-0.594	-0.961	-0.961	-0.964
Kernel 2 [C]	0.002	-0.580	-0.985	-0.985	-0.879
Kernel 3 [F]	0.010	-0.196	-0.194	-0.196	-0.958
Kernel 4 [F]	0.002	-0.015	-0.019	-0.016	-0.797
Kernel 5 [F]	0.001	-0.406	-0.407	-0.406	-0.668
Kernel 6 [F]	0.003	0.003	0.005	0.005	-0.506
Kernel 7 [F]	0.001	-0.028	0.002	-0.003	-0.634
Kernel 8 [F]	0.009	0.056	0.059	0.076	1.270
Kernel 9 [F]	0.009	-0.003	-0.002	0.011	0.798
Kernel 10 [F]	0.001	-0.027	-0.016	-0.022	-0.833
Kernel 11 [F]	0.009	-0.005	-0.008	-0.005	1.350
Kernel 12 [F]	0.001	-0.043	-0.059	-0.035	-0.688
Kernel 13 [F]	0.009	0.007	-0.005	-0.008	run error
Kernel 14 [F]	0.009	0.000	-0.007	-0.004	run error
Kernel 15 [F]	0.008	0.004	-0.001	0.005	run error
Kernel 16 [C]	0.001	-0.066	-0.634	-0.638	-0.581
Kernel 17 [F]	0.001	-0.044	-0.034	-0.041	run error
Kernel 18 [F]	0.001	0.127	0.120	0.129	run error
Kernel 19 [F]	0.009	0.009	0.012	0.010	run error
Kernel 20 [F]	0.009	-0.007	-0.007	-0.008	-0.416
Kernel 21 [C]	0.001	-0.682	-0.618	-0.637	-0.571
Kernel 22 [C]	0.004	compile error	compile error	compile error	0.269

Compiler Variant

← abs. → ← relative →

Relative Performance Gain: +1.0 … -1.0

Results for PolyBench

- PolyBench with *Large* input (≈25MB mem.)

➔ **LLVM+Polly: best** results (followed by FJclang)

➔ **FJtrad worst** option exc. in 4 cases

➔ Highlights:

  ➔ Over **250.000x speedup** for *mvt*

  ➔ **Median speedup** of **3.8x** by using "best" compiler

Time-to-Solution [in s] (FJtrad) and Relative Performance Gain (others)

	runtime	PolyBench (all with [1|1])			
	FJtrad	**FJclang**	**LLVM**	**LLVM+Polly**	**GNU**
correlation [C]	10.743	1.152	0.965	208.559	0.987
covariance [C]	10.735	1.150	0.984	300.513	0.990
gemm [C]	1.629	2.857	0.389	0.606	0.991
gemver [C]	0.092	19.289	1.169	7.152	1.408
gesummv [C]	0.025	0.008	0.004	4.658	-0.020
symm [C]	12.491	0.913	-0.062	1.931	-0.053
syr2k [C]	6.424	0.556	0.315	0.576	0.378
syrk [C]	3.011	0.490	0.169	0.231	0.377
trmm [C]	6.267	0.906	1.065	2711.882	0.977
2mm [C]	18.272	2.800	0.936	15.296	1.264
3mm [C]	30.365	2.856	0.966	17.264	1.197
atax [C]	0.047	13.141	1.480	2.847	1.498
bicg [C]	0.057	-0.073	-0.071	2.046	0.020
doitgen [C]	5.819	12.307	1.499	1303.331	1.587
mvt [C]	0.085	25.995	1.163	250130.507	1.485
cholesky [C]	0.878	-0.862	-0.854	-0.192	-0.852
durbin [C]	0.001	-0.544	-0.579	-0.577	-0.781
gramschmidt [C]	12.863	0.024	0.077	0.522	0.383
lu [C]	28.580	1.246	1.389	26.973	1.345
ludcmp [C]	7.142	-0.025	-0.292	-0.534	-0.375
trisolv [C]	0.022	1.387	1.437	1.819	1.450
deriche [C]	0.753	-0.096	-0.014	-0.145	-0.043
floyd-warshall [C]	0.364	1.515	1.537	-0.741	1.209
nussinov [C]	22.950	0.941	1.511	1.726	0.896
adi [C]	48.401	0.572	0.078	0.935	0.057
fdtd-2d [C]	3.393	3.578	0.818	-0.300	0.669
heat-3d [C]	8.549	3.806	1.122	-0.421	0.362
jacobi-1d [C]	0.004	4.173	1.648	-0.541	1.132
jacobi-2d [C]	5.163	2.750	1.188	-0.486	0.543
seidel-2d [C]	54.664	1.732	1.683	1.570	0.476

Compiler Variant

Relative Performance Gain: +1.0 … -1.0

Results for x500, Babel, ECP, Fiber

- ➔ Surprising **≈5% gain for HPL** (LLVM or FJclang) despite main time in SSL2
- ➔ Same for DLproxy (matmul convolution; SSL2) but even higher gain w/ GNU
- ➔ **GNU: 51%** runtime reduction in **stream** (➔ eq. to higher GB/s)
- ➔ For **ECP apps use LLVM** or GNU and for **Fiber apps use Fujitsu**'s compiler
- ➔ **Avg. speedup: 1.65x** (median 1.09x) with **max. 6.7x** in XSBench
- ➔ **[4 | 12] rarely best option**

Time-to-Solution [in s] (FJtrad) and Relative Performance Gain (others)

	runtime	Ranking			
	FJtrad	**FJclang**	**LLVM**	**LLVM+Polly**	**GNU**
HPL [C]	21.506 [48|1]	0.001 [48|1]	0.043 [48|1]	0.046 [48|1]	-0.023 [48|1]
HPCG [C++]	0.528 [48|1]	0.031 [48|1]	-0.116 [48|1]	-0.191 [48|1]	-0.518 [48|1]
Babel [C++]	1.676 [1|36]	-0.004 [1|36]	0.377 [1|24]	0.296 [1|24]	0.512 [1|32]
DLproxy [C]	0.048 [1|48]	-0.071 [1|48]	0.016 [1|48]	0.019 [1|48]	0.155 [1|48]
		ECP proxy apps			
AMG [C]	5.042 [4|12]	0.058 [8|6]	0.206 [32|1]	-0.304 [32|1]	-0.524 [4|12]
CoMD [C]	4.460 [48|1]	0.086 [48|1]	0.124 [48|1]	0.122 [48|1]	0.132 [48|1]
Laghos	45.436 [48|1]	run error	0.586 [48|1]	0.611 [48|1]	0.399 [48|1]
MACSio [C,C++]	24.343 [48|1]	-0.067 [48|1]	0.184 [48|1]	0.172 [48|1]	0.043 [48|1]
miniAMR [C]	18.857 [48|1]	-0.045 [48|1]	0.016 [48|1]	0.024 [48|1]	0.113 [48|1]
miniFE [C++]	0.373 [4|12]	-0.097 [4|12]	-0.374 [4|12]	-0.459 [4|12]	-0.718 [4|12]
miniTRI [C++]	29.182 [32|1]	0.217 [32|1]	3.607 [1|48]	3.539 [1|48]	3.329 [1|48]
Nekbone [F]	1.656 [48|1]	run error	0.027 [48|1]	0.026 [48|1]	-0.304 [48|1]
SW4lite [F,C++]	0.853 [48|1]	0.011 [48|1]	-0.022 [48|1]	0.003 [48|1]	-0.484 [48|1]
SWFFT [F,C]	1.194 [32|1]	0.035 [32|1]	0.055 [32|1]	0.045 [32|1]	0.063 [32|1]
XSBench [C]	1.649 [1|48]	0.323 [1|48]	0.754 [1|48]	5.865 [1|48]	-0.029 [1|48]
		RIKEN miniapps			
FFB [F,C,C++]	29.877 [4|1]	compile error	compile error	compile error	2.457 [48|1]
FFVC [C++,F]	11.558 [1|36]	-0.026 [1|36]	-0.260 [48|1]	-0.254 [48|1]	-0.926 [48|1]
MODYLAS [F]	29.262 [16|3]	0.001 [16|3]	-0.134 [16|3]	-0.139 [16|3]	-0.768 [16|3]
mVMC [C,F]	15.026 [24|2]	-0.002 [24|2]	-0.081 [48|1]	0.173 [48|1]	-0.334 [48|1]
NICAM [F]	7.645 [10|4]	-0.003 [10|4]	-0.001 [10|4]	0.014 [10|4]	-0.768 [10|4]
NTChem [F]	9.440 [12|4]	-0.001 [12|4]	0.061 [12|4]	0.061 [12|4]	-0.418 [24|1]
QCD [F]	8.048 [24|2]	0.002 [24|2]	0.003 [24|2]	0.003 [24|2]	0.036 [24|2]

Compiler Variant

Relative Performance Gain: +1.0 … -1.0

Results for SPEC CPU

- ➔ **SPEC int**: FJtrad better than clang-based, but **GNU outperforms all others**
  - ➔ Likely result of GNU's prevalence in embedded space and Arm's continued investments into GNU compilers (see: https://community.arm.com/developer/tools-software/tools/b/tools-software-ides-blog/posts/gcc-10-better-and-faster)
- ➔ **SPEC float: GNU is worst** option and most are dominantly **written in Fortran (➔ no real benefit from LLVM12 except LTO?)**
- ➔ "Real" flang (not F18) (eg. https://github.com/flang-compiler/flang) might improve situation

Time-to-Solution [in s] (FJtrad) and Relative Performance Gain (others)

	runtime	SPEC CPU int (all with [1|1])			
perlbench [C]	95.842	-0.497	-0.458	-0.459	0.249
gcc [C]	144.654	-0.656	-0.662	-0.660	0.237
mcf [C]	107.932	-0.546	-0.554	-0.555	0.266
omnetpp [C++]	153.212	0.127	0.038	0.021	0.025
xalancbmk [C++]	227.200	0.856	0.860	1.054	0.852
xs [C]	66.105	-0.840	-0.844	-0.844	0.145
deepsjeng [C++]	174.909	0.292	0.225	-1.000	0.338
leela [C++]	225.918	0.293	0.255	0.286	0.388
exchanges [F]	137.069	0.000	-0.010	-0.009	0.778
xz [C]	77.578	-0.638	-0.656	-0.647	0.036
		SPEC CPU float			
bwaves [F]	1.690 [1|32]	-0.004 [1|32]	-0.009 [1|32]	-0.011 [1|32]	-0.774 [1|32]
cactuBSSN [C++,C,F]	4.956 [1|48]	0.249 [1|48]	-0.031 [1|48]	0.011 [1|48]	-0.426 [1|36]
lbm [C]	11.397 [1|48]	0.109 [1|48]	-0.118 [1|48]	-0.123 [1|48]	-0.446 [1|48]
wrf [F,C]	5.482 [1|32]	0.002 [1|32]	0.007 [1|32]	0.002 [1|32]	-0.864 [1|32]
cams [F,C]	11.925 [1|48]	0.006 [1|48]	0.012 [1|48]	0.007 [1|48]	-0.322 [1|32]
pops [F,C]	15.115 [1|32]	0.004 [1|32]	0.002 [1|32]	0.001 [1|32]	invalid output
imagick [C]	18.778 [1|8]	5.651 [1|8]	2.114 [1|8]	1.265 [1|8]	1.423 [1|8]
nab [C]	17.824 [1|48]	0.207 [1|48]	0.082 [1|48]	0.155 [1|48]	-0.013 [1|48]
fotonik [F]	8.893 [1|48]	0.002 [1|48]	0.001 [1|32]	-0.004 [1|32]	-0.521 [1|32]
roms [F]	8.134 [1|48]	-0.013 [1|48]	-0.004 [1|48]	-0.003 [1|48]	-0.779 [1|48]
	FJtrad	FJclang	LLVM	LLVM+Polly	GNU

Compiler Variant

Relative Performance Gain: +1.0 … -1.0

Results for SPEC OMP

Computer simulations create the future

- Similar to SPEC CPU float: **GNU is worst** exc. for kdtree (16.5x speedup)
- **Average time-to-solution improvement of 49% in SPEC CPU and 2.5x speedup in SPEC OMP** with "best" compiler (over FJtrad)
- **Median** improvement across both SPEC suites is **14%**
- Many of SPEC benchmarks **don't scale to 48 cores** (eg. full A64FX) (biggest "offender" is **SPEC CPU's imagick** with sweet-spot at **only 8 OMP threads**)

Time-to-Solution [in s] (FJtrad) and Relative Performance Gain (others)

	runtime	SPEC OMP			
	FJtrad	FJclang	LLVM	LLVM+Polly	GNU
md [F]	109.786 [1|48]	-0.008 [1|48]	-0.008 [1|48]	-0.008 [1|48]	-0.870 [1|48]
bwaves [F]	0.541 [1|48]	-0.045 [1|48]	-0.045 [1|48]	-0.056 [1|48]	-0.824 [1|32]
nab [C]	25.995 [1|48]	0.185 [1|48]	-0.011 [1|48]	0.055 [1|48]	-0.150 [1|36]
bt [F]	18.427 [1|48]	-0.002 [1|48]	-0.001 [1|48]	-0.001 [1|48]	0.168 [1|36]
botsalgn [C]	0.686 [1|48]	-0.043 [1|48]	0.009 [1|48]	0.007 [1|48]	-0.223 [1|48]
botsspar [C]	0.492 [1|48]	0.926 [1|16]	1.244 [1|32]	2.134 [1|32]	0.525 [1|12]
ilbdc [F]	15.715 [1|48]	-0.001 [1|48]	-0.001 [1|48]	-0.002 [1|48]	-0.972 [1|48]
fma [F]	11.982 [1|48]	-0.008 [1|48]	0.016 [1|48]	0.011 [1|48]	-0.091 [1|36]
swim [F]	1.367 [1|32]	-0.015 [1|32]	-0.016 [1|32]	-0.015 [1|32]	-0.559 [1|24]
imagick [C]	4.573 [1|48]	0.911 [1|48]	-0.253 [1|48]	-0.379 [1|48]	-0.232 [1|48]
mgrid [F]	0.220 [1|32]	-0.102 [1|32]	-0.108 [1|32]	-0.105 [1|32]	-0.308 [1|32]
applu [F]	3.196 [1|32]	-0.008 [1|32]	-0.012 [1|32]	-0.010 [1|32]	0.055 [1|32]
smithwa [C]	3.082 [1|48]	2.361 [1|48]	1.367 [1|48]	1.417 [1|48]	invalid output
kdtree [C++]	166.330 [1|48]	10.355 [1|48]	10.691 [1|48]	10.704 [1|48]	15.470 [1|48]

Compiler Variant

Relative Performance Gain: +1.0 … -1.0

Summary & Conclusion

- **Across all 108** benchmarks and realistic workloads: **median runtime improvement of 16%** is possible (simply by selecting right compiler)
- **Performance discrepancy for PolyBench solved** by switching from the FJtrad to LLVM 12 compiler, but **otherwise polly seems rarely useful**

Revisit initial questions:

- **A1**: recomm. usage model of **4 ranks and 12 threads often suboptimal**
- **A2**: **no "silver bullet" compiler** for A64FX (yet)
  - Dep. on situation, but some hint: **Fujitsu for Fortran codes,** and **GNU for integer-intensive apps,** and **any clang-based compilers for C/C++**
- **A3**: Twitter summary: "if Xeon is 70x faster than A64fx, suspect the compiler"

Recommendation:

- Install & test all avail. compilers, and explore other rank/thread mappings!

Future Work

- Anyone interested with access to an **intern/student** to continue this?
- Testing **Arm compilers**
  - Reviewer (Arm employee) offered help
- Testing **HPE/Cray compiler**
- In-depth analysis on **reasons for performance difference?**:
  - SVE, loop transformations, prefetcher, cache behavior, etc.
- **Automatic compiler-flag tuning** for all Fugaku workloads
  - Eg. https://github.com/ctuning/ck/wiki/Compiler-autotuning#Autotuning_LLVM_flags

Open-Source & Acknowledgements

- Reproducing our data? Doing your own analysis?
  ➔ Use our framework:

  https://gitlab.com/domke/a64fxCvC or

This work was supported by

- New Energy and Industrial Technology Development Organization (**NEDO**); and
- Japan Society for the Promotion of Science **KAKENHI Grant Number 19H04119**

Job/Collaboration Opportunities

- Collaborations and job opportunities:
  - Check out our research teams and open positions: https://www.riken.jp/en/research/labs/r-ccs/ and https://bit.ly/3faax8v
- Internship/fellowship for students (Bachelor→PhD):
  - Fellowship: https://www.riken.jp/en/careers/programs/index.html
  - Internship: https://www.r-ccs.riken.jp/en/about/careers/internship/
- Supercomputer Fugaku:
  - Apply for node-hours: https://www.r-ccs.riken.jp/en/fugaku/user-guide/
  - Interactive, virtual tour: https://www.r-ccs.riken.jp/en/fugaku/3d-models/ and https://www.youtube.com/watch?v=f3cx4PGDGmg

Figure sources

- [1] https://www.starwars.com/news/8-great-life-teachings-from-yoda