Exploring compiler behavior on applications from Miniapps up to large scale applications on modern processors

Hugo Bolloré, Emmanuel Oseret, Kévin Camus, Cédric Valensi, William Jalby, Université de Versailles Saint-Quentin-en-Yvelines

ACKNOWLEDGEMENTS: ALTAIR, AWS, MEGWARE, ATOS/EVIDEN and CALMIP

HORIZON-EUROHPC-JU-2023-COE

1 January 2024– 31 December 2026

Grant Agreement No 101143931

Objectives

The first goal is to analyze and assess quantitatively, compiler and compiler options impact on performance.

The ultimate goal is to optimize application performance

- Identify strengths/weakness of various compilers options and compilers
- Leverage optimizations between compilers
- Study ISA impact
- Develop corresponding methodology and test on a real industrial strength application

TALK OVERVIEW

- Methodology (MAQAO/ONEVIEW/QaaS)
- Mini App results
- LAMMPS Results
- OpenRadioss results
- Conclusion

METHODOLOGY

Methodology (1)

Standard case: Compiler A with options OPT1 is slightly faster than compiler B with options OPT2.

Easy (superficial) conclusion: use Compiler A with options OPT1

Better (more complex) approach: analyze compiler behavior

ISSUE: Compiler A can outperform compiler B for Function FOO and the situation can be reversed for another Function BOO. There can be compensation between various functions.

Performance analysis has to go down at least to the function level and very often down to the loop level.

Due to the number of functions/loops involved, Analysis/assessment has to be automated.

Methodology (2)

Timing will be performed at three levels: whole app, function level, loop level

Goals require detailed analysis of compiler outputs: assembly code

Relying on MAQAO/ONE View capabilities

- To evaluate capabilities, weakness and strengths of various ASM codes using simplified simulators
- To perform matching between source code and different ASM variants: essential requisite to analyze advanced compiler strategies multiversioning

Analyzing Code Quality (1)

Focus on loops: innermost/in between/outermost

Evaluate ASM using CQA (Code Quality Analysis) included in MAQAO.

- Generic topics of interest
  - Port / Functional Units usage
  - Vectorization
  - Instruction set use
  - Vectorization Roadblocks
  - Data access
- Two types of analysis: **Static** at the ASM level and **Dynamic** requiring measurement

IMPORTANT: By looking directly at ASM, both compiler mistakes but also source code issues will be taken into account.

Analyzing Code Quality (2)

Classify performance issues into 5 main categories

1. **Loop computation:** issues related to the computation organization (FMA, SQRT/DIV, etc…)
2. **Control Flow:** issues relevant to control (branches, call, ….)
3. **Data access:** issues essentially related to memory operations (stride, indirect, spill/fill, ….)
4. **Vectorization roadblocks:** issues preventing vectorization (complex control flow, …)
5. **Inefficient vectorization:** issues related to vectorization quality (vector length, masked, …)

Loop ID	Analysis	Penalty Score
▼ Loop 29963 - engine_linuxa64_gf_ompi	Execution Time: 1 % - Vectorization Ratio: 21.05 % - Vector Length Use: 59.87 %	
▼ Loop Computation Issues		6
○	[SA] Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA - Reorganize arithmetic expressions to exhibit potential for FMA. This issue costs 4 points.	4
○	[SA] Presence of a large number of scalar integer instructions - Simplify loop structure, perform loop splitting or perform unroll and jam. This issue costs 2 points.	2
▼ Control Flow Issues		2
○	[SA] Several paths (2 paths) - Simplify control structure or force the compiler to use masked instructions. There are 2 issues (= paths) costing 1 point each.	2
▼ Data Access Issues		12
○	[SA] Presence of constant non unit stride data access - Use array restructuring, perform loop interchange or use gather instructions to lower a bit the cost. There are 2 issues (= data accesses) costing 2 point each.	4
○	[SA] Presence of indirect accesses - Use array restructuring or gather instructions to lower the cost. There are 2 issues (= indirect data accesses) costing 4 point each.	8
▶ Vectorization Roadblocks		14

PENALTY SCORE: metric evaluating effort required to fix the corresponding issue

Global Characteristics: OneView typical output (1)

Global Metrics	
Total Time (s)	7.51 E3
Max (Thread Active Time) (s)	7.42 E3
Average Active Time (s)	6.77 E3
Activity Ratio (%)	90.2
Average number of active threads	86.565
Affinity Stability (%)	100.0
Time in analyzed loops (%)	85.8
Time in analyzed innermost loops (%)	81.3
Time in user code (%)	87.7
Compilation Options Score (%)	74.9
Array Access Efficiency (%)	70.6

48 MPI Ranks + 2 OMP Threads/ranks

Link to MAQAO report

4 essential metrics:

1. **Average active time:** Sum over all threads of their active time divided by thread count. **IDEAL = total execution time**
2. **Activity ratio :** Sum over all threads of their active time divided by the sum of their wall time. **IDEAL: 100%**
3. **Average active number of threads :** Sum over all threads of their active time divided by longest wall clock time: **IDEAL = number of threads used**
4. **Affinity stability:** evaluates percentage of time spent without thread migration between physical cores: **IDEAL 100%**

OpenRadioss Taurus GFortran –O3 on G4

Global Characteristics: OneView typical output (2)

Potential Speedups		
Perfect Flow Complexity		1.05
Perfect OpenMP + MPI + Pthread		1.05
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.19
No Scalar Integer	Potential Speedup	1.20
	Nb Loops to get 80%	32
FP Vectorised	Potential Speedup	1.11
	Nb Loops to get 80%	24
Fully Vectorised	Potential Speedup	1.28
	Nb Loops to get 80%	41
FP Arithmetic Only	Potential Speedup	1.56
	Nb Loops to get 80%	41

48 MPI Ranks + 2 OMP Threads/ranks

OpenRadioss Taurus GFortran –O3 on G4

For each thread, we gather a full picture of their activity: time spent in various functions including time spent in various libraries: OpenMP, MPI and Pthreads.

From these measurements we extrapolate impact of time spent in OpenMP/MPI/Pthread libraries and load distribution.

Perfect OpenMP + MPI + Pthread: max (Original Thread time)/ **max** (Stripped Thread Time): Stripped ⬄ time spent in OpenMP/MPI/Pthread set to 0.

Perfect OpenMP + MPI + Pthread + Perfect Load Distribution: max (Original Thread time)/ **average** (Stripped Thread Time):

A performance optimization goal is to have these 2 metrics under 1.1

https://datafront.exascale-computing.eu/public/OpenRadioss/TAURUS10M/G4_AWS/gnu/OpenRadioss_TAURUS10M-short_g4-aws_o2_m48_gnu_engine/

QaaS Characteristics (1)

QaaS is an **Open Source collaborative** project aiming at providing a software environment exploring (understanding/characterizing) applications/systems performance, the ultimate goal being application optimization.

QaaS (Quality as a Service) relies on the following features

- Automation: pushed as much as possible. Large amount of data automatically generated
- Systematic exploration of key parameters space:
  - hardware platforms (AMD, ARM (Ampere, Grace, G3/G4), INTEL, etc…)
  - Compilers and compiler settings
  - number of cores and core mapping, more generally runtime and run time settings
  - More generally: software stack parameters

QaaS Characteristics (2)

- Uniform exploration: to enable systematic comparison
- Reliability: built in repetition mechanisms to ensure reliable results
- Detailed level of comparison: use of MAQAO/OneView to provide several profiling levels : whole application, functions, loops and key application characteristics.
- Flexible search and formatting through results
- Two levels of optimizations:
  - automatic (compilers/compiler switches/transformations),
  - recommendations for “manual” optimizations.

For more details see https://www.compqual.org/qaas_page/

MINIAPP RESULTS

Miniapps

For the miniapps used in our tests, a good description/reference is provided on the site : https://proxyapps.exascaleproject.org/app/.

- AMG
- Cloverleaf
- CoMD
- Hacc MK
- MiniQMC
- Kripke

COMPILER AND COMPILER OPTIONS FOR G3 RUNS (64 cores)

orig	default
gcc	default
armclang 1	O2 -mcpu=native -armpl
armclang 2	O2 -mcpu=native -fno-vectorize -armpl
armclang 3	O3 -mcpu=native -armpl
armclang 4	O3 -mcpu=native -fno-vectorize -armpl
armclang 5	Ofast -mcpu=native -armpl
armclang 6	Ofast -mcpu=native -fno-vectorize -armpl
gcc 1	O2 -mcpu=native -funroll-loops -larmpl
gcc 2	O2 -mcpu=native -fno-tree-vectorize -fno-openmp-simd -funroll-loops -larmpl
gcc 3	O3 -mcpu=native -funroll-loops -larmpl
gcc 4	O3 -mcpu=native -fno-tree-vectorize -fno-openmp-simd -funroll-loops -larmpl
gcc 5	Ofast -mcpu=native -funroll-loops -larmpl
gcc 6	Ofast -mcpu=native -fno-tree-vectorize -fno-openmp-simd -funroll-loops -larmpl

PERFORMANCE OF MINIAPPS ON G3 (64 cores)

	AMG	CLOVERLEAF	COMD	HACC mk	MINIQMC	KRIPKE
orig	1,01	1,01	1,10	4,40	1,00	1,02
gcc	1,00	1,00	1,00	1,00	1,01	2,13
armclang 1	1,02	1,02	1,08	4,39	1,00	1,01
armclang 2	1,01	1,02	1,07	9,01	1,01	1,00
armclang 3	1,01	1,02	1,09	4,39	1,00	1,01
armclang 4	1,01	1,01	1,09	9,01	1,00	1,01
armclang 5	1,01	1,01	1,04	1,44	1,00	1,23
armclang 6	1,01	1,01	1,04	2,50	1,02	1,00
gcc 1	1,02	1,00	1,02	2,62	1,02	1,23
gcc 2	1,03	1,00	1,02	2,62	1,02	1,23
gcc 3	1,00	1,00	1,00	1,00	1,01	2,00
gcc 4	1,00	1,00	1,00	2,62	1,01	1,23
gcc 5	1,00	1,00	1,04	1,00	1,01	1,97
gcc 6	1,00	1,00	1,02	2,62	1,01	1,23

BEST: TIMING RATIOS: current timing over best timing across all compilers and compiler options.

- Green cells corresponds to best or very close to best (< 1%)
- Yellow cells correspond to performance losses between 1% and 5%
- Red cells correspond to performance losses > 5%

A FEW CONCLUSIONS

	AMG	CLOVERLEAF	COMD	HACC mk	MINIQMC	KRIPKE
orig	1,01	1,01	1,10	4,40	1,00	1,02
gcc	1,00	1,00	1,00	1,00	1,01	2,13
armclang 1	1,02	1,02	1,08	4,39	1,00	1,01
armclang 2	1,01	1,02	1,07	9,01	1,01	1,00
armclang 3	1,01	1,02	1,09	4,39	1,00	1,01
armclang 4	1,01	1,01	1,09	9,01	1,00	1,01
armclang 5	1,01	1,01	1,04	1,44	1,00	1,23
armclang 6	1,01	1,01	1,04	2,50	1,02	1,00
gcc 1	1,02	1,00	1,02	2,62	1,02	1,23
gcc 2	1,03	1,00	1,02	2,62	1,02	1,23
gcc 3	1,00	1,00	1,00	1,00	1,01	2,00
gcc 4	1,00	1,00	1,00	2,62	1,01	1,23
gcc 5	1,00	1,00	1,04	1,00	1,01	1,97
gcc 6	1,00	1,00	1,02	2,62	1,01	1,23

For half of the miniapp (AMG, Cloverleaf, Min,iQMC) compiler and compiler options have limited impact.

For the other half, compiler (CoMD, HACC mk, Kripke) compiler and compiler options have a large impact

The original compiler options are not the best choice….

Neither GCC nor ARMCLANG are always winning.

LAMMPS RESULTS

LAMMPS CODE

Source: https://www.lammps.org/#gsc.tab=0

- *LAMMPS is a classical molecular dynamics code with a focus on materials modeling. It's an acronym for Large-scale Atomic/Molecular Massively Parallel Simulator.*
- *LAMMPS has potentials for solid-state materials (metals, semiconductors) and soft matter (biomolecules, polymers) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale.*
- *LAMMPS runs on single processors or in parallel using message-passing techniques and a spatial-decomposition of the simulation domain. Many of its models have versions that provide accelerated performance on CPUs, GPUs, and Intel Xeon Phis. The code is designed to be easy to modify or extend with new functionality.*

LAMMPS on Graviton 3 (64 cores) AWS: Armclang versus GCC

	ARMCLANG 1 - O3 - mcpu=neoverse-v1 -armpl -ffast-math	ARMCLANG 2 - O3 - mcpu=neoverse-v1 -armpl -msve-vector-bits=256 -ffast-math	ARMCLANG 4 - O3 - mcpu=neoverse-v1 +nosve+nosve2 -armpl -ffast-math	ARMCLANG 5 - O3 - mcpu=neoverse-v1 -armpl -fno-vectorize -fno-slp-vectorize -fno-openmp-simd -ffast-math	ARMCLANG 6 -O2 - mcpu=neoverse-v1 -armpl -ffast-math	GCC1 -O3 - mcpu=neoverse-v1 -larmpl -funroll-loops -ffast-math	GCC2 -O3 - mcpu=neoverse-v1 -larmpl -msve-vector-bits=256 -funroll-loops -ffast-math	GCC3 -O3 - mcpu=neoverse-v1 -larmpl -msve-vector-bits=128 -funroll-loops -ffast-math	GCC4 -O3 - mcpu=neoverse-v1+nosve+nosve2 -larmpl -funroll-loops -ffast-math	GCC5 -O3 - mcpu=neoverse-v1 -larmpl -fno-tree-vectorize -fno-openmp-simd -funroll-loops -ffast-math	GCC6 -O2 - mcpu=neoverse-v1 -larmpl -funroll-loops -ffast-math
Metric	**ARMCLANG 1**	**ARMCLANG 2**	**ARMCLANG 4**	**ARMCLANG 5**	**ARMCLANG 6**	**GCC 1**	**GCC 2**	**GCC 3**	**GCC 4**	**GCC 5**	**GCC 6**
Total Time (s)	420,57	422,54	423,42	430,12	421,46	434,58	433,02	433,38	434,74	432,86	440,65
Best	1,00	1,00	1,01	1,02	1,00	1,03	1,03	1,03	1,03	1,03	1,05
Max (Thread Active Time) (s)	418,66	420,65	421,57	428,29	419,53	432,68	431,32	431,52	432,7	431,22	438,59
Average Active Time (s)	418,43	420,29	421,23	427,84	419,19	432,35	430,79	431,13	432,43	430,67	438,33
Activity Ratio (%)	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5
Average number of active threads	63,67	63,66	63,67	63,66	63,66	63,67	63,67	63,67	63,66	63,68	63,66
Affinity Stability (%)	99,9	100	99,9	100	99,9	99,9	100	100	99,9	100	99,9

BEST: TIMING RATIOS: current timing over best timing across all compilers and compiler options.

- Green cells corresponds to best or very close to best (< 1%)
- Yellow cells correspond to performance losses: 1% < and < 3%
- Red cells correspond to performance losses > 3%

LAMMPS on Graviton 3 (64 cores) AWS: Armclang versus GCC

	ARMCLANG 1	ARMCLANG 2	ARMCLANG 4	ARMCLANG 5	ARMCLANG 6	GCC 1	GCC 2	GCC 3	GCC 4	GCC 5	GCC 6
LAMMPS_NS::PairEAM::compute(int, int)	1,00	1,00	1,01	1,02	1,00	1,03	1,04	1,04	1,04	1,03	1,05
LAMMPS_NS::NPairBin<1, 1, 0, 0, 1>::build(LAMMPS_NS::NeighList*	1,00	1,01	1,00	1,03	1,01	1,05	1,02	1,02	1,03	1,03	1,04
LAMMPS_NS::FixNVE::initial_integrate(int)	1,00	1,00	1,00	1,00	1,01	1,00	1,00	1,00	1,00	1,00	1,00
LAMMPS_NS::FixNVE::final_integrate()	1,00	1,00	1,00	1,00	1,00	1,01	1,00	1,01	1,00	1,00	1,00
LAMMPS_NS::CommBrick::borders()	1,02	1,00	1,01	1,01	1,01	1,01	1,00	1,01	1,03	1,00	1,01
LAMMPS_NS::AtomVec::pack_comm(int, int*, double*, int, int*)	1,04	1,05	1,04	1,04	1,03	1,01	1,03	1,02	1,01	1,06	1,00
LAMMPS_NS::AtomVec::unpack_reverse(int, int*, double*)	1,08	1,10	1,09	1,10	1,10	1,00	1,05	1,01	1,00	1,02	1,05
LAMMPS_NS::Neighbor::check_distance()	1,00	1,00	1,01	1,00	1,00	1,01	1,01	1,00	1,00	1,00	1,00

TIMING RATIOS: current timing over best timing across all compilers and compiler options.

- Green cells corresponds to best or very close to best (< 1%)
- Yellow cells correspond to performance losses 1% < and < 5%
- Red cells correspond to performance losses > 5%

LAMMPS on Granite Rapids (256 cores) MEGWARE: ICX versus GCC

	STANDARD LAMMPS PACKAGE				X86 OPTIMIZED LAMMPS PACKAGE			
Metric	orig_default	gcc_default	icx_2	gcc_5	orig_default	gcc_default	icx_2	gcc_7
Total Time (s)	107,83	112,62	108,66	107,06	50,75	89,41	51,02	86,55
BEST	2,12	2,22	2,14	2,11	1,00	1,76	1,01	1,71
Max (Thread Active Time) (s)	92,39	99,71	93,2	95,12	34,68	77,39	34,91	75,07
Average Active Time (s)	91,71	99,38	92,5	95,04	34,05	76,94	34,19	74,69
Activity Ratio (%)	85,6	88,7	85,3	89,4	68	86,6	67,4	87
Average number of active threads	218	226	218	227	172	220	172	221
Affinity Stability (%)	87,5	89,3	87,1	90,2	71,8	87,4	71,2	87,8
Time in analyzed loops (%)	90,6	90,2	91	92,6	84,5	91	84,9	90,7
Time in analyzed innermost loops (%)	87,3	87,1	87,7	88,2	57	88,5	57,4	86,1
Time in user code (%)	90,9	90,5	91,3	92,9	88,8	91,2	89,1	92,7
Compilation Options Score (%)	100	50	100	100	100	50	100	87,5
Array Access Efficiency (%)	51,7	45,5	51,9	56,7	50,8	65,5	51,2	71,1

TIMING RATIOS: current timing over best timing across all compilers and compiler options.

- Green cells corresponds to best or very close to best (< 1%)
- Yellow cells correspond to performance losses 1% < and < 5%
- Red cells correspond to performance losses > 5%

LAMMPS on Granite Rapids (256 cores) MEGWARE: ICX versus GCC

		STANDARD LAMMPS PACKAGE				X86 OPTIMIZED LAMMPS PACKAGE			
Metric		orig_default	gcc_default	icx_2	gcc_5	orig_default	gcc_default	icx_2	gcc_7
Total Time (s)		107,83	112,62	108,66	107,06	50,75	89,41	51,02	86,55
BEST		2,12	2,22	2,14	2,11	1,00	1,76	1,01	1,71
Potential Speedups									
Perfect Flow Complexity		3,06	3,5	3,08	3,59	1,01	1,01	1,01	1,97
Perfect OpenMP/MPI/Pthread/TBB		1,02	1,02	1,02	1,01	1,06	1,02	1,07	1,02
Perfect OpenMP/MPI/Pthread/TBB + Perfect Load Distribution		1,1	1,1	1,1	1,07	1,11	1,09	1,11	1,07
No Scalar Integer	Potential Speedup	1,3	1,21	1,3	1,3	1,06	1,06	1,06	1,07
	Nb Loops to get 80%	3	3	3	3	7	6	7	6
FP Vectorised	Potential Speedup	1,66	1,83	1,67	1,64	1,01	2,05	1,01	1,64
	Nb Loops to get 80%	3	3	3	3	4	5	4	5
Fully Vectorised	Potential Speedup	3,63	4,27	3,67	4,41	1,1	5,95	1,1	5,88
	Nb Loops to get 80%	4	5	5	6	15	17	15	18
Only FP Arithmetic	Potential Speedup	1,5	1,28	1,5	1,41	2,41	1,11	2,42	1,2
	Nb Loops to get 80%	3	3	3	4	12	10	12	12

TIMING RATIOS: current timing over best timing across all compilers and compiler options.

- Green cells corresponds to best or very close to best (< 1%)
- Yellow cells correspond to performance losses 1% < and < 5%
- Red cells correspond to performance losses > 5%

OpenRadioss RESULTS

Target code: OpenRadioss

Altair® Radioss® & OpenRadioss™
The Industry Standard Open Platform for Crash & Impact

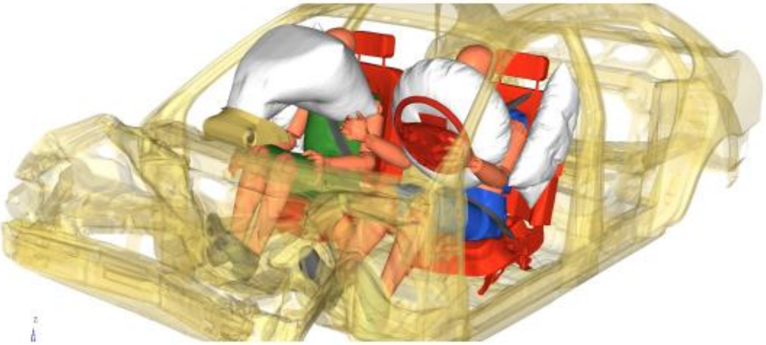
OpenRadioss™ Open-Source Version

- Source code publicly accessible from: https://github.com/OpenRadioss
- Upstream version, with contributions from a fast-growing worldwide community
- Precompiled Linux & Windows executables to run latest builds with no license check
- Support from the community, via forum

www.openradioss.org

Altair® Radioss® Commercial Version

- Commercial releases with extensive QA, professional support, documentation and maintenance priority
- Available under Altair Units license
- Encrypted models for dummies & barriers
- Channels valuable community contributions into industrial release



www.altair.com/radioss

ALTAIR

Dataset and code parallelism

Realistic data sets have been used. Due to domain decomposition, data set size varies depending upon the number of MPI ranks used

- TAURUS (Public): 10 millions elements, reference 1.7 GB, after partitioning for 96 MPI ranks 17 GB

Code in Fortran combining MPI + OpenMP parallelism

- Excellent MPI scaling due to very good domain decomposition
- More limited OpenMP parallelism

OUR TWO REFERENCE TESTS: 96 MPI Ranks and 48 MPI Ranks + 2 OpenMP threads per rank

Experimental constraints

NUMERICAL ACCURACY

The code uses explicit method with very small time steps.

Accuracy and reproducibility is major concern. Therefore any code transformation which has an impact on numerical accuracy is prohibited. For example, fast math compiler flag and even the option enabling fused multiply add (FMA) is prohibited

EXECUTION TIME

Due to the nature of the code (mechanical crash), various code segments are used throughout the whole simulation => to be realistic (involving all key code segments) the run has to be long enough…

TAURUS on 96 cores Neoverse V2 takes over 2 hours!

Vectorization Taurus ACfl –O3 on G4

Loop id	Source Location	Source Function	Level	Exclusive Coverage mpi_96_ranks_omp_1_thread (%)	Vectorization Ratio (%)	Vector Length Use (%)
40382	engine_linuxa64_ompi_O3 - i7optcd.F:202-243 [...]	i7optcd	Innermost	5.06	48.11	66.04
13900	engine_linuxa64_ompi_O3 - czproj.F:1205-1466 [...]	czprojn	Innermost	3.09	0	49.8
6105	engine_linuxa64_ompi_O3 - asspar4.F:165-173	asspar4	Innermost	2.73	45.45	72.73
44945	engine_linuxa64_ompi_O3 - sigeps36c.F:269-269	sigeps36c	Single	2.55	97.83	98.91
11492	engine_linuxa64_ompi_O3 - cupdtn3.F:653-692	cupdtn3p	Single	2.36	25.96	62.5
40362	engine_linuxa64_ompi_O3 - i7buce_crit.F:131-150	i7buce_crit	Outermost	1.97	27.45	58.33
6415	engine_linuxa64_ompi_O3 - depla.F:95-104	depla	Innermost	1.63	42.11	71.05
40724	engine_linuxa64_ompi_O3 - gravit.F:157-159	gravit	Innermost	1.54	88.89	100
52519	engine_linuxa64_ompi_O3 - mulawc.F:2378-2384	mulawc	Innermost	1.53	59.02	79.1
6507	engine_linuxa64_ompi_O3 - vitesse.F:80-86	vitesse	Innermost	1.40	46.15	72.12
13455	engine_linuxa64_ompi_O3 - czfintn.F:355-443	czfintn1	Single	1.36	92.36	96.84

Link to MAQAO report

Compiler flags: ACfl versus GFortran on AWS G4

Limited impact of Compiler Options

Global metric	GNU				AcfL			
	O2	O3	O3 no SVE2	O3 no SVE	O2	O3	O3 no SVE2	O3 no SVE
Total Time (s)	7.89 E3	7.51 E3	7.65 E3	7.41 E3	7.92 E3	7.91 E3	7.92 E3	8.26 E3
Max (Thread Active Time) (s)	7.71 E3	7.42 E3	7.47 E3	7.33 E3	7.89 E3	7.88 E3	7.88 E3	8.07 E3
Average Active Time (s)	7.02 E3	6.77 E3	6.78 E3	6.66 E3	7.77 E3	7.77 E3	7.77 E3	7.86 E3
Activity Ratio (%)	89.0	90.2	88.7	90.0	98.2	98.1	98.1	95.9
Average number of active threads	85.4	86.7	85.1	86.3	94.210	94.193	94.179	91.284
Affinity Stability (%)	99.2	100.0	99.1	100.0	100.0	100.0	100.0	98.5
Time in analyzed loops (%)	86.4	85.8	85.7	85.4	78.1	78.0	77.8	76.3
Time in analyzed innermost loops (%)	78.0	81.3	81.2	80.9	70.5	71.4	71.2	69.6
Time in user code (%)	88.1	87.7	87.6	87.4	80.9	80.8	80.6	79.2
Compilation Options Score (%)	62.5	74.9	74.9	74.9	99.5	99.5	99.5	99.5
Array Access Efficiency (%)	62.0	70.6	70.6	74.0	66.9	65.1	65.1	66.8

GFortran compiler flags: impact on AWS G4

Limited impact of Compiler Options

Global metric	GNU				AcfL			
	O2	O3	O3 no SVE2	O3 no SVE	O2	O3	O3 no SVE2	O3 no SVE
Total Time (s)	7.89 E3	7.51 E3	7.65 E3	7.41 E3	7.92 E3	7.91 E3	7.92 E3	8.26 E3
Max (Thread Active Time) (s)	7.71 E3	7.42 E3	7.47 E3	7.33 E3	7.89 E3	7.88 E3	7.88 E3	8.07 E3
Average Active Time (s)	7.02 E3	6.77 E3	6.78 E3	6.66 E3	7.77 E3	7.77 E3	7.77 E3	7.86 E3
Activity Ratio (%)	89.0	90.2	88.7	90.0	98.2	98.1	98.1	95.9
Average number of active threads	85.4	86.7	85.1	86.3	94.210	94.193	94.179	91.284
Affinity Stability (%)	99.2	100.0	99.1	100.0	100.0	100.0	100.0	98.5
Time in analyzed loops (%)	86.4	85.8	85.7	85.4	78.1	78.0	77.8	76.3
Time in analyzed innermost loops (%)	78.0	81.3	81.2	80.9	70.5	71.4	71.2	69.6
Time in user code (%)	88.1	87.7	87.6	87.4	80.9	80.8	80.6	79.2
Compilation Options Score (%)	62.5	74.9	74.9	74.9	99.5	99.5	99.5	99.5
Array Access Efficiency (%)	62.0	70.6	70.6	74.0	66.9	65.1	65.1	66.8

- **FOR GFortran: O2 is worse (6%), O3 no-sve is the best**

ACfl compiler flags: impact on AWS G4

Limited impact of Compiler Options ?

Global metric	*GNU*				*AcfL*			
	O2	*O3*	*O3 no SVE2*	*O3 no SVE*	*O2*	*O3*	*O3 no SVE2*	*O3 no SVE*
Total Time (s)	7.89 E3	7.51 E3	7.65 E3	7.41 E3	7.92 E3	7.91 E3	7.92 E3	8.26 E3
Max (Thread Active Time) (s)	7.71 E3	7.42 E3	7.47 E3	7.33 E3	7.89 E3	7.88 E3	7.88 E3	8.07 E3
Average Active Time (s)	7.02 E3	6.77 E3	6.78 E3	6.66 E3	7.77 E3	7.77 E3	7.77 E3	7.86 E3
Activity Ratio (%)	89.0	90.2	88.7	90.0	98.2	98.1	98.1	95.9
Average number of active threads	85.4	86.7	85.1	86.3	94.210	94.193	94.179	91.284
Affinity Stability (%)	99.2	100.0	99.1	100.0	100.0	100.0	100.0	98.5
Time in analyzed loops (%)	86.4	85.8	85.7	85.4	78.1	78.0	77.8	76.3
Time in analyzed innermost loops (%)	78.0	81.3	81.2	80.9	70.5	71.4	71.2	69.6
Time in user code (%)	88.1	87.7	87.6	87.4	80.9	80.8	80.6	79.2
Compilation Options Score (%)	62.5	74.9	74.9	74.9	99.5	99.5	99.5	99.5
Array Access Efficiency (%)	62.0	70.6	70.6	74.0	66.9	65.1	65.1	66.8

- **For ACfl: O3 no-sve is the worst (5%), O2/O3/O3 no-sve are identical and the best**

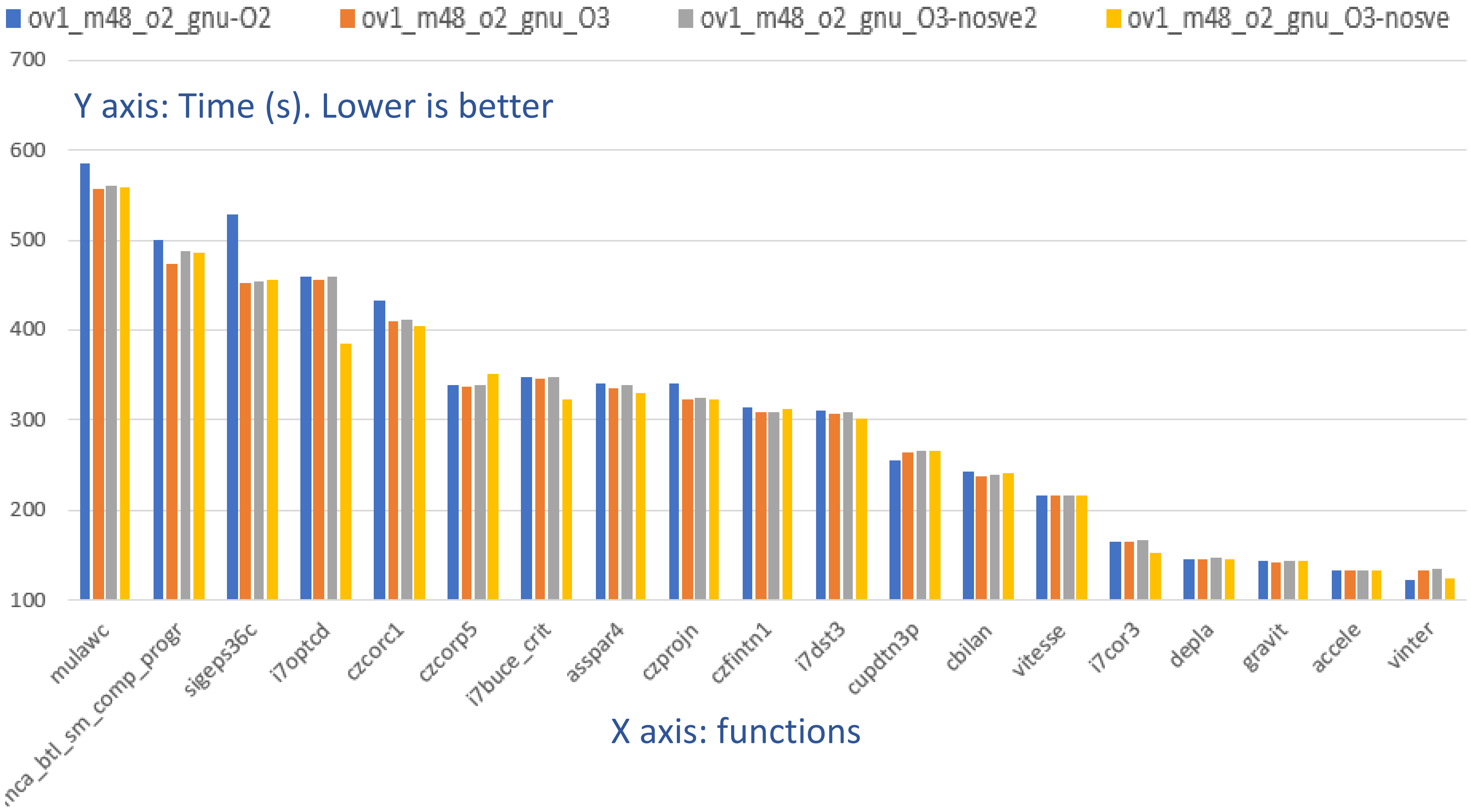
Compiler flags: ACfl versus GFortran on AWS G4

Limited impact of Compiler Options ?

Global metric	*GNU*				*AcfL*			
	O2	*O3*	*O3 no SVE2*	*O3 no SVE*	*O2*	*O3*	*O3 no SVE2*	*O3 no SVE*
Total Time (s)	7.89 E3	7.51 E3	7.65 E3	7.41 E3	7.92 E3	7.91 E3	7.92 E3	8.26 E3
Max (Thread Active Time) (s)	7.71 E3	7.42 E3	7.47 E3	7.33 E3	7.89 E3	7.88 E3	7.88 E3	8.07 E3
Average Active Time (s)	7.02 E3	6.77 E3	6.78 E3	6.66 E3	7.77 E3	7.77 E3	7.77 E3	7.86 E3
Activity Ratio (%)	89.0	90.2	88.7	90.0	98.2	98.1	98.1	95.9
Average number of active threads	85.4	86.7	85.1	86.3	94.210	94.193	94.179	91.284
Affinity Stability (%)	99.2	100.0	99.1	100.0	100.0	100.0	100.0	98.5
Time in analyzed loops (%)	86.4	85.8	85.7	85.4	78.1	78.0	77.8	76.3
Time in analyzed innermost loops (%)	78.0	81.3	81.2	80.9	70.5	71.4	71.2	69.6
Time in user code (%)	88.1	87.7	87.6	87.4	80.9	80.8	80.6	79.2
Compilation Options Score (%)	62.5	74.9	74.9	74.9	99.5	99.5	99.5	99.5
Array Access Efficiency (%)	62.0	70.6	70.6	74.0	66.9	65.1	65.1	66.8

- For GFortran: O2 is worse (6%), O3 no-sve is the best
- For ACfl: O3 no-sve is the worst (5%), O2/03/O3 no-sve are identical and the best
- ACfl is using active waiting counted as activity while GFortran is using passive waiting counted as inactivity
- GFortran is better than ACfl: 6%
- For both compilers and options: a large amount of time is spent in loops

Compiler flags impact: GFortran on AWS G4



Every compiler option is losing sometimes, winning at other times

GFortran on AWS G4: Some detailed analyses

l7optcd function

Difference comes from a loop conditionally setting arrays values to zero

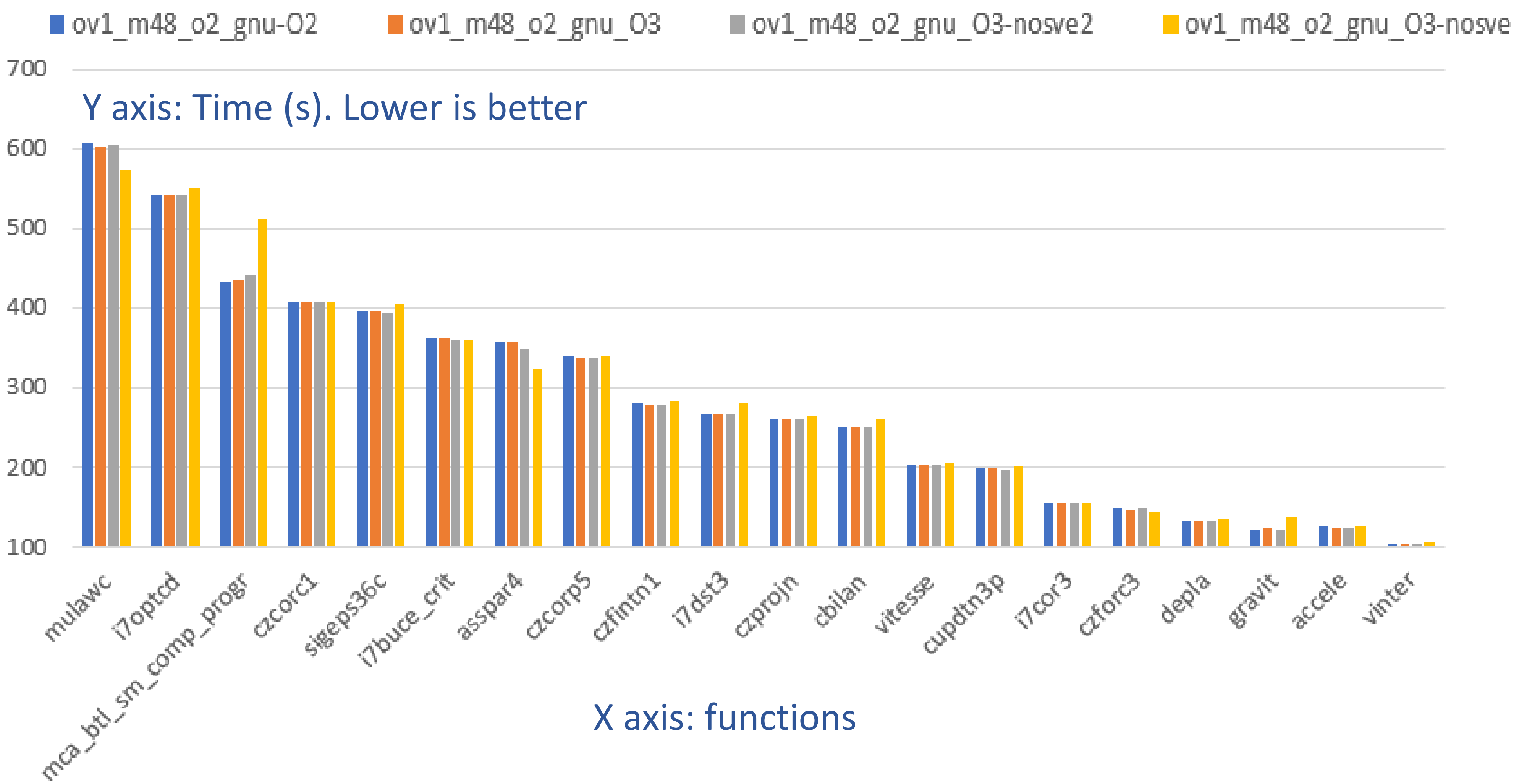
- no-sve option produces a scalar version taking ~25s
- Without excluding option, SVE version is taking ~90s

Multiple factors: scalar version can use dedicated instructions/registers when comparing/setting with zero whereas SVE instruction set lack such feature and requires to have every store instruction under predication

l7bucecrit function

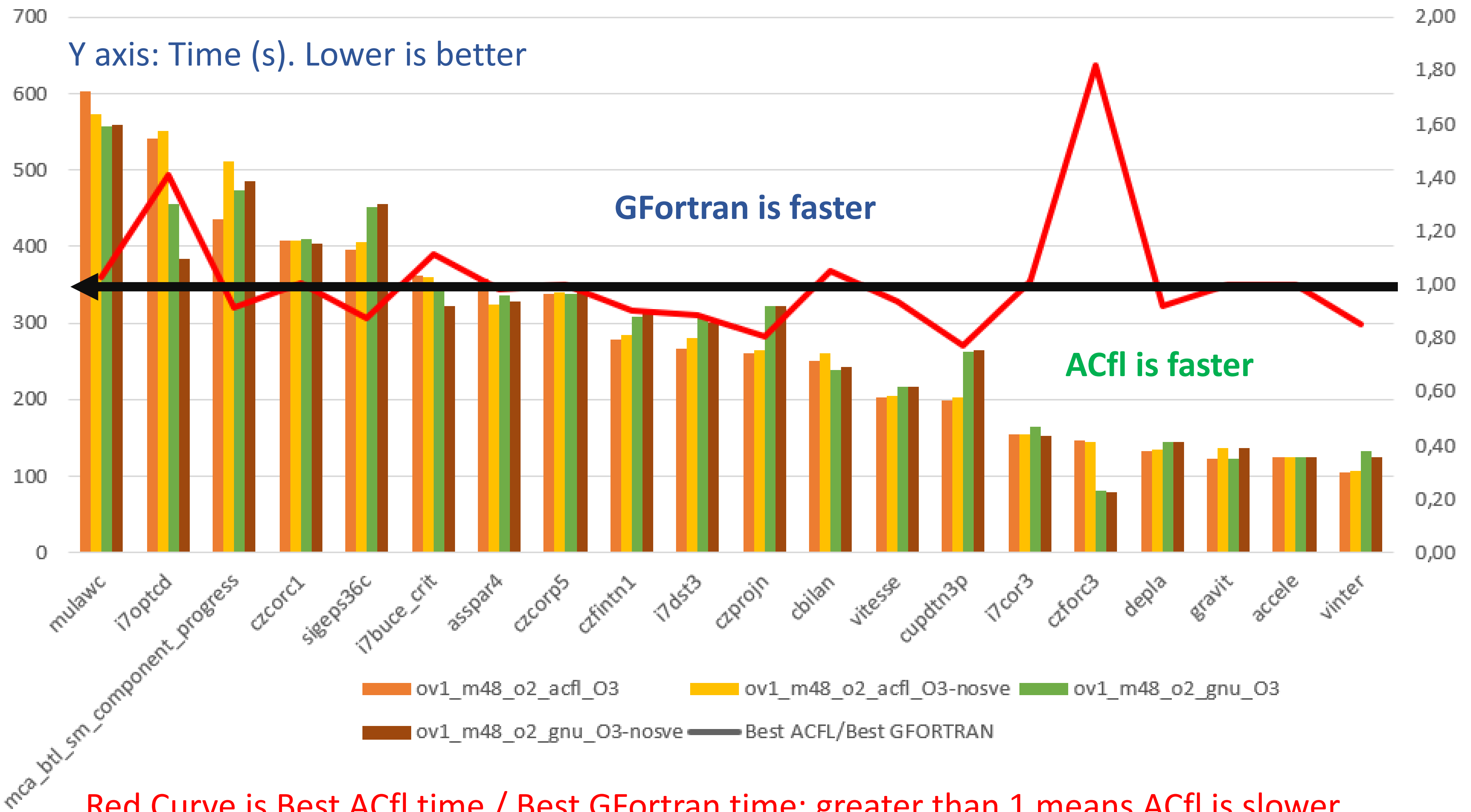
One of the loop is vectorized in NEON ONLY if SVE is disabled
Bug/Cost model ?

Compiler flags impact: ACfl on AWS G4



O3 no-sve is the compiler option with the largest impact (positive or negative !)

Best ACfl/best GFortran on AWS G4



Red Curve is Best ACfl time / Best GFortran time: greater than 1 means ACfl is slower, lower than 1 means GFortran is slower

Compiler Analysis of OpenRadiOss

1) Fine tuning compiler options: up to 5% performance gain

2) Fine tuning across compiler and compiler options: up to 10% performance gain

Future work

- Analyze more compiler options: fastmath, funroll, etc….
- Automate backport from one compiler to the other: pragma insertion: if a compiler has been able to vectorize a loop, the information could be provided through pragmas to compilers which were unsuccessful in vectorizing the same loop.
- Interact with compiler developers to refine/improve cost models
- Interact with application developers to use this technology

CONCLUSIONS

Conclusions

- Non uniform behavior of compiler options across subroutines/loops: some options perform better with some subroutines
- Non uniform behavior between compilers across subroutines/loops: no silver bullet compiler….
- Non uniform means non negligible performance difference: these performance difference are worth exploring/exploiting

Above all of these are well known “generic facts”

Conclusions

- Non uniform behavior of compiler options across subroutines/loops: some options perform better with some subroutines
- Non uniform behavior between compilers across subroutines/loops: no silver bullet compiler….
- Non uniform means non negligible performance difference: these performance difference are worth exploring/exploiting

Above all of these are well known “generic facts” but MAQAO/QaaS allows to bring in:

- Quantitative estimation : essential for driving optimization
- Explanation of performance differences: this opens the door to backport optimization between compilers

Performance Optimisation and Productivity 3

A Centre of Excellence in HPC

Contact:

https://www.pop-coe.eu

pop@bsc.es

@POP_HPC

youtube.com/POPHPC

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101143931. The JU receives support from the European Union's Horizon Europe research and innovation programme and Spain, Germany, France, Portugal and the Czech Republic.

Backup Results

A FEW SPECIFIC MAQAO METRICS (1)

First, most of our dynamic measurements are based on sampling.

We distinguish and measure for every thread two types of wasted time

- Inactive time (obtained by comparing wall clock with active time = sum of collected samples)
- Time spent in OpenMP, MPI, Pthreads activity

With these measurements we generate 4 essential metrics:

1. **Average active time:** Sum over all threads of their active time divided by thread count. **IDEAL = total execution time**
2. **Activity ratio :** Sum over all threads of their active time divided by the sum of their wall time. **IDEAL: 100%**
3. **Average active number of threads :** Sum over all threads of their active time divided by longest wall clock time: **IDEAL = number of threads used**
4. **Affinity stability:** evaluates percentage of time spent without thread migration between physical cores: **IDEAL 100%**

A FEW SPECIFIC MAQAO METRICS (2)

For each thread, we gather a full picture of their activity: time spent in various functions including time spent in various libraries: OpenMP, MPI and Pthreads.

From these measurements we extrapolate impact of time spent in OpenMP/MPI/Pthread libraries and load distribution.

Perfect OpenMP + MPI + Pthread: max (Original Thread time)/ **max** (Stripped Thread Time): Stripped ⇔ time spent in OpenMP/MPI/Pthread set to 0.

Perfect OpenMP + MPI + Pthread + Perfect Load Distribution: max (Original Thread time)/ **average** (Stripped Thread Time):

A performance optimization goal is to have these 2 metrics under 1.1.

OpenMP Results

Backup Slides

G4 NEONM11 runs: 1 MPI Rank, 96 OpenMP threads, ACFL –O3 (1/3)

Global Metrics	
Total Time (s)	3.90 E3
Max (Thread Active Time) (s)	2.82 E3
Average Active Time (s)	2.59 E3
Activity Ratio (%)	66.4
Average number of active threads	63.729
Affinity Stability (%)	100.0
Time in analyzed loops (%)	48.0
Time in analyzed innermost loops (%)	40.6
Time in user code (%)	50.9
Compilation Options Score (%)	100
Array Access Efficiency (%)	73.6

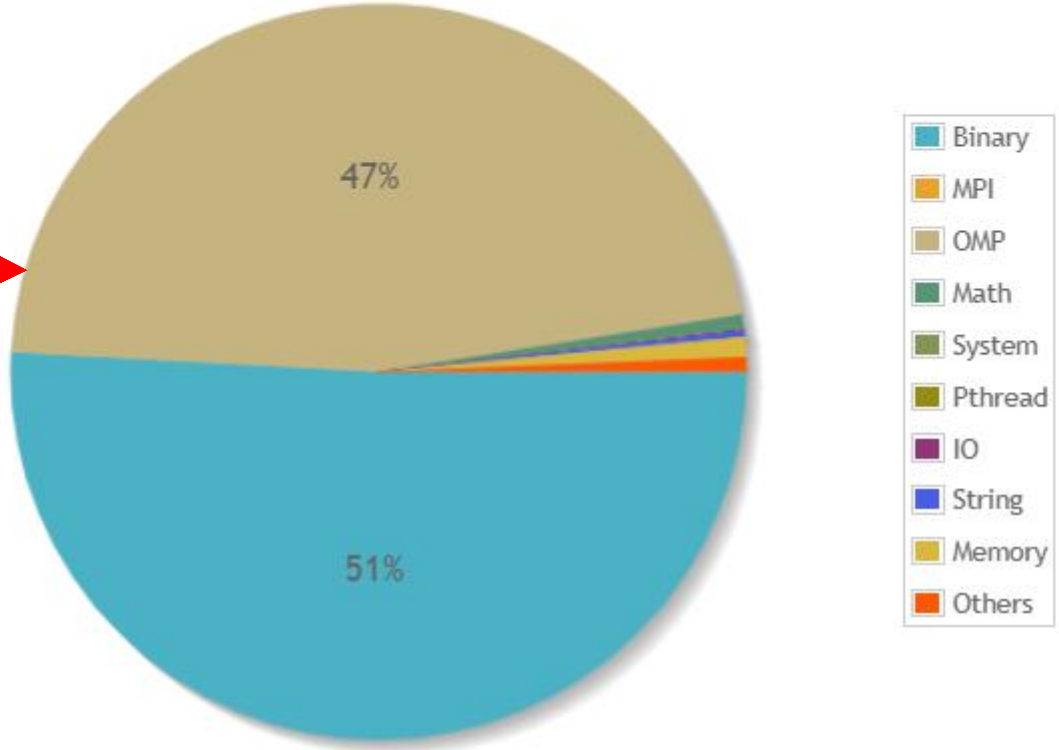
Large gap indicates a large amount of thread inactive time

Low numbers: poor resource usage

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_o96_m1_acfl-o3/index.html

G4 NEONM11 runs: 1 MPI Rank, 96 OpenMP threads, ACFL –O3 (2/3)

Potential Speedups		
Perfect Flow Complexity		1.01
Perfect OpenMP + MPI + Pthread		1.87
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		2.04
No Scalar Integer	Potential Speedup	1.11
	Nb Loops to get 80%	17
FP Vectorised	Potential Speedup	1.02
	Nb Loops to get 80%	7
Fully Vectorised	Potential Speedup	1.11
	Nb Loops to get 80%	19
FP Arithmetic Only	Potential Speedup	1.29
	Nb Loops to get 80%	26



G4 NEONM11 runs: 1 MPI Rank, 96 OpenMP threads, ACFL – O3 (3/3)

Name	Module	Coverage mpi_1_rank_omp_96_threads (%)	Nb Threads mpi_1_rank_omp_96_threads
○ __aarch64_cas4_acq_rel	libomp.so	14.96	96
○ int __kmp_dispatch_next_algorithm<int>(int, dispatch_private_info_template<int>*, dispatch_shared_info_template<int> volatile*, int*, int*, int*, traits_t<int>::signed_t*, int, int)	libomp.so	14.21	96
○ __kmp_wait_4	libomp.so	7.74	96
○ __aarch64_ldadd4_acq_rel	libomp.so	7.28	96
▶ asspar4	engine_linuxa64_ompi	6.63	96
▶ mulawc	engine_linuxa64_ompi	6.56	96
▶ i7buce_crit	engine_linuxa64_ompi	5.12	96
▶ m2cplr	engine_linuxa64_ompi	3.95	96
▶ cupdt3p	engine_linuxa64_ompi	3.26	96

MAIN ISSUES

- critical section (OMP CRITICAL) in i7buce_crit
- locks (OMP_SET_LOCK) mostly from i7optcd (lockon.inc:28)

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_o96_m1_acfl-o3/fcts_and_loops.html

MPI Results

G4 NEONM11 runs: **96 MPI Ranks**, 1 OpenMP threads, ACFL – O3 (1/3)

Global Metrics

Total Time (s)	1.71 E3
Max (Thread Active Time) (s)	1.63 E3
Average Active Time (s)	1.61 E3
Activity Ratio (%)	93.7
Average number of active threads	90.878
Affinity Stability (%)	98.9
Time in analyzed loops (%)	71.3
Time in analyzed innermost loops (%)	63.7
Time in user code (%)	76.5
Compilation Options Score (%)	100
Array Access Efficiency (%)	73.4

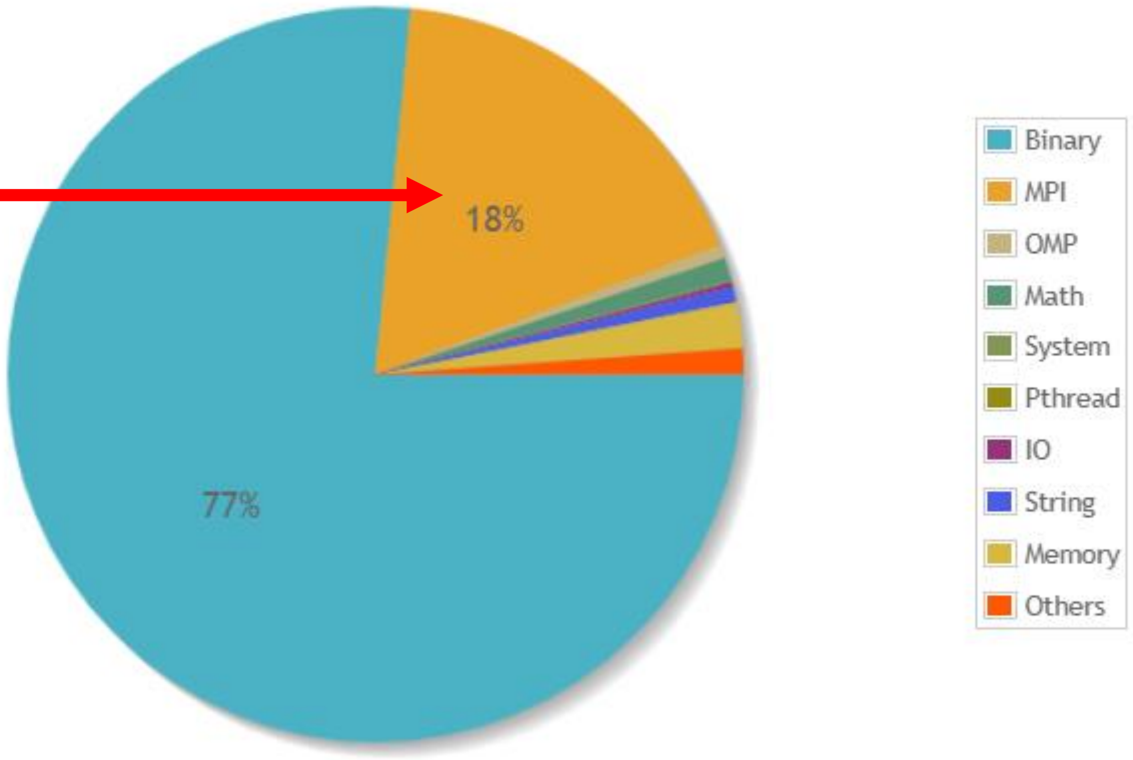
Small gap indicates a small amount of thread inactive time

High numbers: good resource usage

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_o1_m96_acfl-o3/

G4 NEONM11 runs: **96 MPI Ranks**, 1 OpenMP threads, ACFL – O3 (2/3)

Potential Speedups		
Perfect Flow Complexity		1.01
Perfect OpenMP + MPI + Pthread		1.15
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.24
No Scalar Integer	Potential Speedup	1.15
	Nb Loops to get 80%	26
FP Vectorised	Potential Speedup	1.03
	Nb Loops to get 80%	8
Fully Vectorised	Potential Speedup	1.16
	Nb Loops to get 80%	29
FP Arithmetic Only	Potential Speedup	1.50
	Nb Loops to get 80%	41



https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_o1_m96_acfl-o3/

G4 NEONM11 runs: **96 MPI Ranks**, 1 OpenMP threads, ACFL – O3 (3/3)

Name	Module	Coverage mpi_96_ranks_omp_1_thread (%)	Nb Threads mpi_96_ranks_omp_1_thread
○ mca_btl_sm_component_progress	libopen-pal.so.80.0.5	13.09	96
▶ mulawc	engine_linuxa64_ompi	10.79	96
▶ asspar4	engine_linuxa64_ompi	7.57	96
▶ m2cplr	engine_linuxa64_ompi	6.62	96
▶ cupdt3p	engine_linuxa64_ompi	5.41	96
▶ chvis3	engine_linuxa64_ompi	4.65	96

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_o1_m96_acfl-o3/fcts_and_loops.html

AMPERE SKYCAB runs: 80/160/3220/640 MPI Ranks, 1 OpenMP threads, ACFL – O3 (1/2)

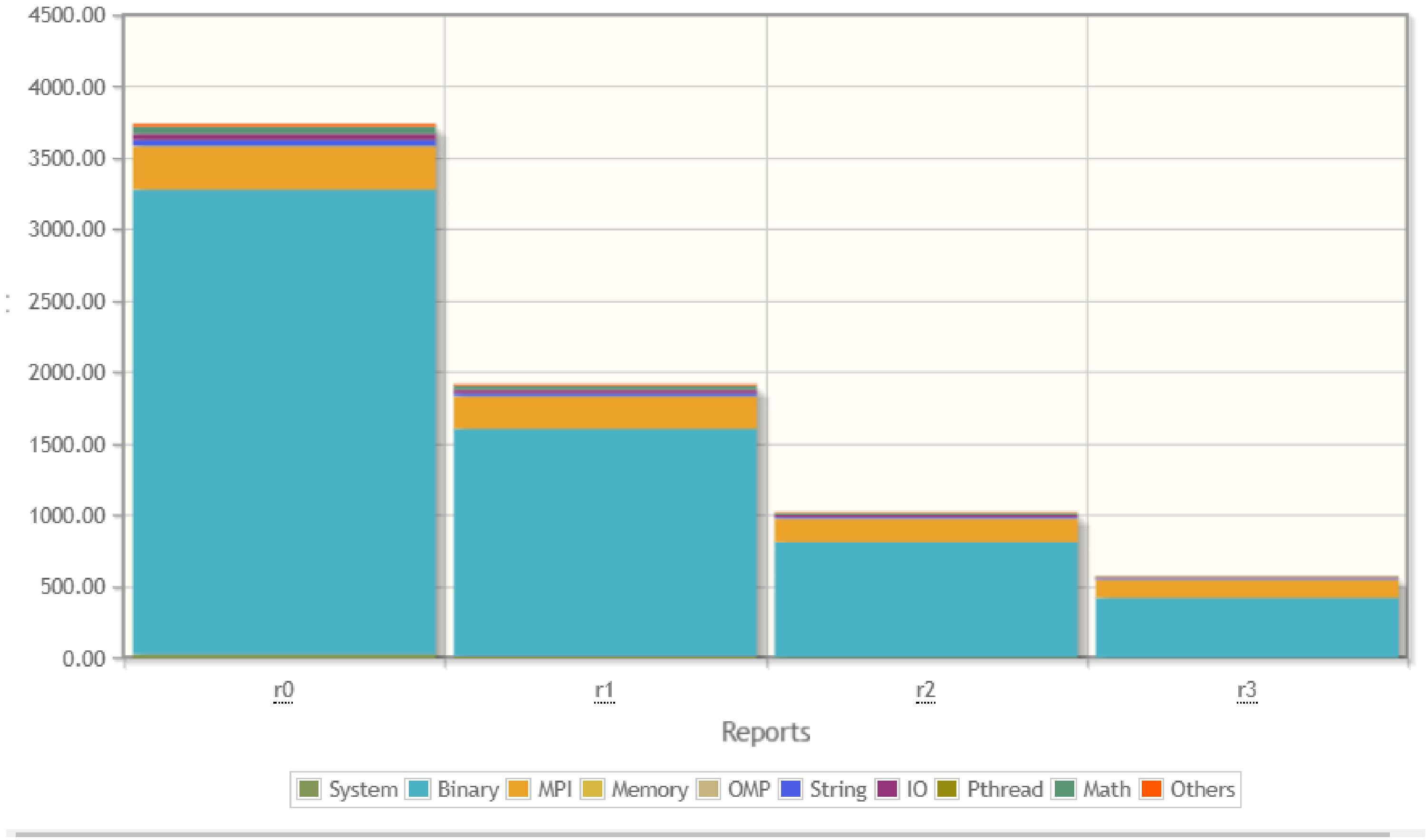
Global Metrics

Metric	r0	r1	r2	r3
Total Time (s)	3.76 E3	1.93 E3	1.04 E3	585.62
Max (Thread Active Time) (s)	3.74 E3	1.92 E3	1.02 E3	575.77
Average Active Time (s)	3.73 E3	1.91 E3	1.02 E3	569.99
Activity Ratio (%)	99.3	99.0	98.0	97.3
Average number of active threads	79.451	158.408	313.725	622.917
Affinity Stability (%)	0.89	0.97	1.39	1.14
Time in analyzed loops (%)	82.4	78.6	74.5	68.7
Time in analyzed innermost loops (%)	79.4	75.6	71.4	65.1
Time in user code (%)	86.9	83.0	78.8	72.9
Compilation Options Score (%)	0	0	0	0
Array Access Efficiency (%)	69.9	70.6	68.4	71.1

r0: 1 node (80 cores), r1: 2 nodes (160 cores), r2 (320 cores), r3 (640 cores)

https://datafront.exascale-computing.eu/public/OpenRadioss/compare/OpenRadioss_SkyCAB_turpan_o3_o1_m80-640_acfl_compare/

AMPERE Skycab runs: 80/160/320/640MPI Ranks, 1 OpenMP threads, ACFL – O3 (2/2)



r0 : 1 node (80 cores),
r1 : 2 nodes (160 cores),
r2 : 4 nodes (320 cores),
r3 : 8 nodes (640 cores)

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_o1_m96_acfl-o3/fcts_and_loops.html

G4 NEONM11 runs: hybrid, ACFL – O3 (1/2)

Global Metrics

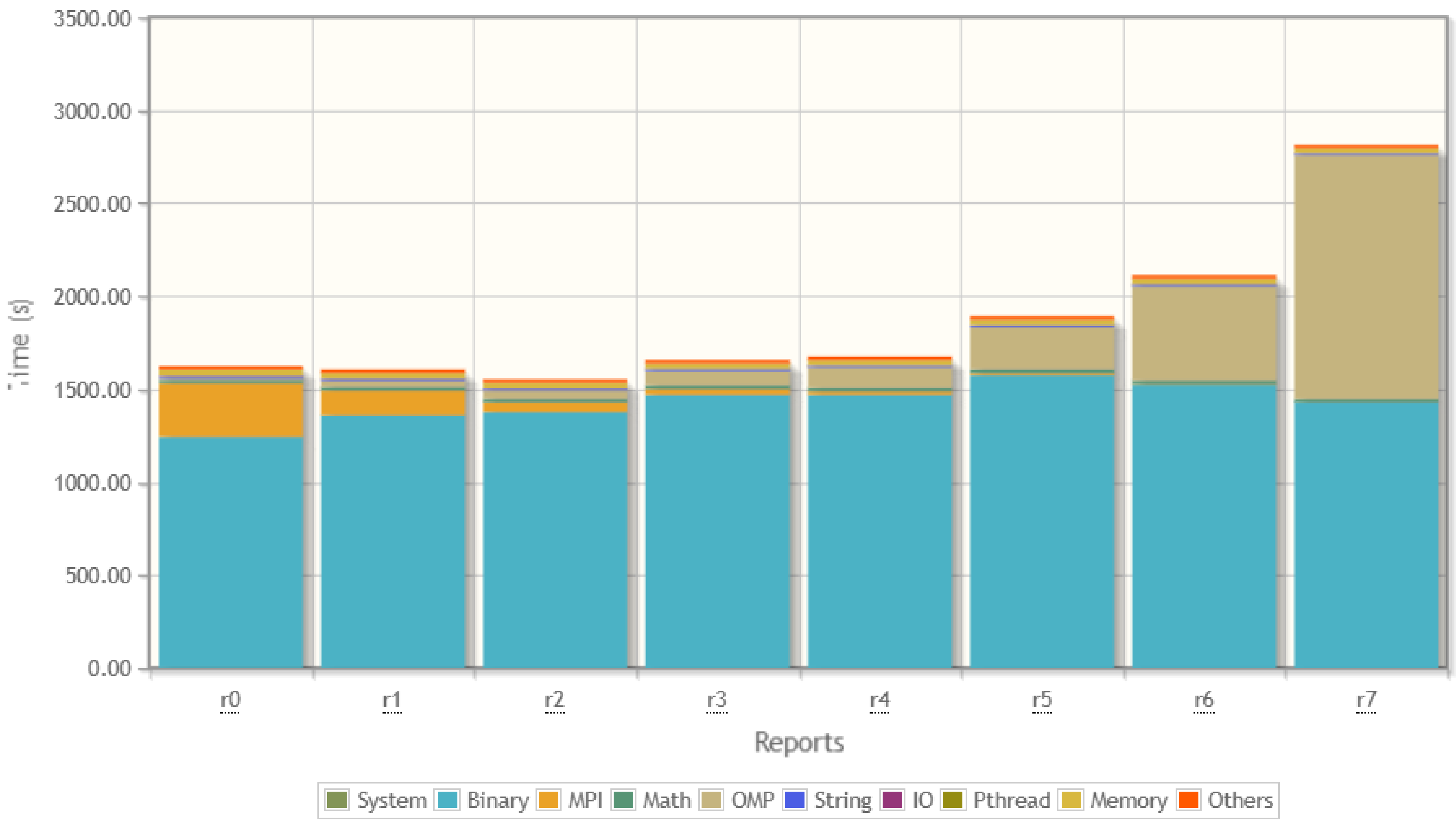
Metric	r0	r1	r2	r3	r4	r5	r6	r7
Total Time (s)	1.71 E3	1.75 E3	1.78 E3	2.01 E3	2.07 E3	2.48 E3	2.86 E3	3.90 E3
Max (Thread Active Time) (s)	1.63 E3	1.61 E3	1.55 E3	1.66 E3	1.68 E3	1.89 E3	2.12 E3	2.82 E3
Average Active Time (s)	1.61 E3	1.45 E3	1.37 E3	1.37 E3	1.40 E3	1.50 E3	1.73 E3	2.59 E3
Activity Ratio (%)	93.7	82.6	77.5	68.2	67.6	60.5	60.7	66.4
Average number of active threads	90.878	79.247	74.324	65.388	64.901	58.014	58.206	63.729
Affinity Stability (%)	98.9	99.9	100.0	100.0	100.0	100.0	100.0	100.0
Time in analyzed loops (%)	71.3	79.1	83.1	83.3	82.5	78.5	68.0	48.0
Time in analyzed innermost loops (%)	63.7	70.4	73.5	71.9	71.0	66.6	57.9	40.6
Time in user code (%)	76.5	84.7	88.7	88.6	87.7	83.3	72.2	50.9
Compilation Options Score (%)	100	100	100	100	100	100	100	100
Array Access Efficiency (%)	73.4	74.2	74.3	75.1	74.9	75.1	74.9	73.6

- r0: ov1_n1m_m96-o1_o3_acfl
- r1: ov1_n1m_m48-o2_o3_acfl
- r2: ov1_n1m_m24-o4_o3_acfl
- r3: ov1_n1m_m12-o8_o3_acfl
- r4: ov1_n1m_m8-o12_o3_acfl
- r5: ov1_n1m_m4-o24_o3_acfl
- r6: ov1_n1m_m2-o48_o3_acfl
- r7: ov1_n1m_m1-o96_o3_acfl

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/OpenRadioss_NEON1M11-full_G4-aws_cr_o1-96_m96-1_acfl-o3/

G4 NEONM11 runs: hybrid, ACFL – O3 (2/2)

Time



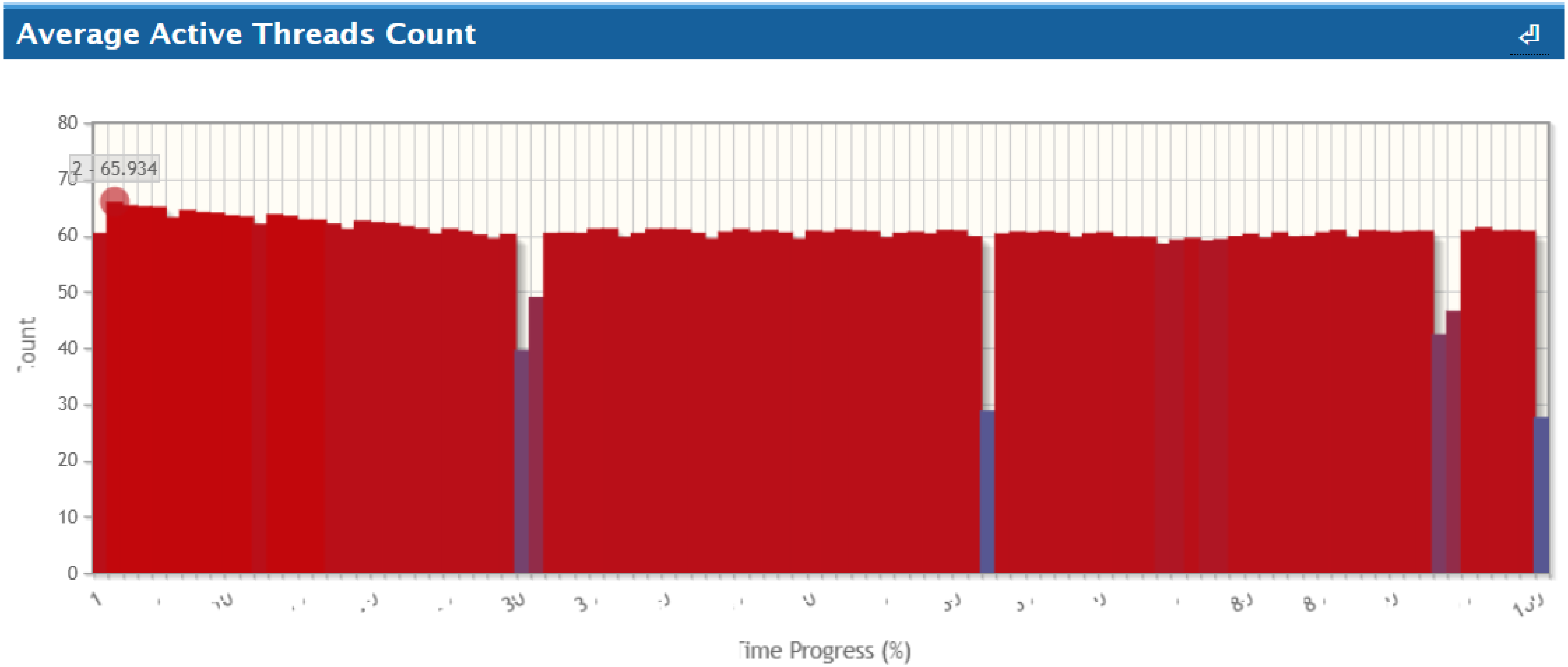
- r0: ov1_n1m_m96-o1_o3_acfl
- r1: ov1_n1m_m48-o2_o3_acfl
- r2: ov1_n1m_m24-o4_o3_acfl
- r3: ov1_n1m_m12-o8_o3_acfl
- r4: ov1_n1m_m8-o12_o3_acfl
- r5: ov1_n1m_m4-o24_o3_acfl
- r6: ov1_n1m_m2-o48_o3_acfl
- r7: ov1_n1m_m1-o96_o3_acfl

G4 NEONM11 runs: hybrid, 8 MPI ranks, 12 OMP threads ACFL – O3 (1/2)

Global Metrics	
Total Time (s)	2.08 E3
Max (Thread Active Time) (s)	1.67 E3
Average Active Time (s)	1.40 E3
Activity Ratio (%)	67.2
Average number of active threads	64.434
Affinity Stability (%)	100.0
Time in analyzed loops (%)	82.5
Time in analyzed innermost loops (%)	70.9
Time in user code (%)	87.7
Compilation Options Score (%)	100
Array Access Efficiency (%)	75.2

https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/EO_OpenRadioss_NEON1M11-full_G4-aws_o12_m8_acfl-o3/index.html

G4 NEONM11 runs: hybrid, 8 MPI ranks, 12 OMP threads ACFL – O3 (2/2)



https://datafront.exascale-computing.eu/public/OpenRadioss/NEON1M11/G4_AWS/acfl/EO_OpenRadioss_NEON1M11-full_G4-aws_o12_m8_acfl-o3/index.html

G4 TAURUS runs: hybrid, ACFL – O3 (1/2)

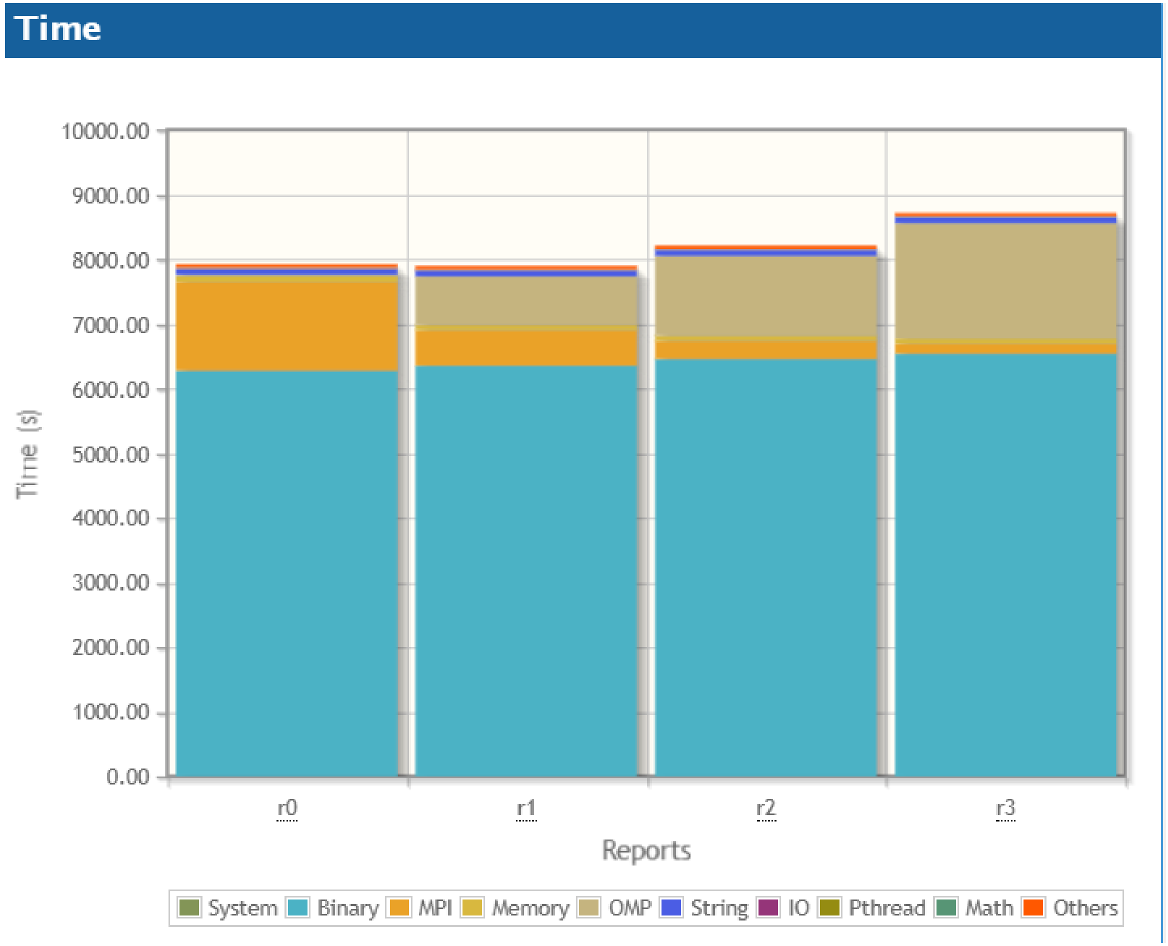
Global Metrics

Metric	r0	r1	r2	r3
Total Time (s)	7.96 E3	7.96 E3	8.27 E3	8.84 E3
Max (Thread Active Time) (s)	7.93 E3	7.91 E3	8.22 E3	8.73 E3
Average Active Time (s)	7.83 E3	7.79 E3	8.04 E3	8.44 E3
Activity Ratio (%)	98.5	98.0	97.2	95.6
Average number of active threads	95.506	93.990	93.282	91.635
Affinity Stability (%)	0.42	0.12	0.07	0.06
Time in analyzed loops (%)	76.3	77.6	75.9	72.5
Time in analyzed innermost loops (%)	69.6	71.1	69.7	66.4
Time in user code (%)	79.2	80.4	78.6	75.0
Compilation Options Score (%)	99.5	99.5	99.5	99.5
Array Access Efficiency (%)	65.3	65.2	65.0	64.8

- r0: mpi_96_omp_1_thread
- r1: mpi_48_omp_2_threads
- r2: mpi_24_omp_4_threads
- r3: mpi_12_omp_8_threads

https://datafront.exascale-computing.eu/public/OpenRadioss/TAURUS10M/G4_AWS/acfl/OpenRadioss_TAURUS10M-short_g4-aws_o1-8_m12-96_acfl_engine/

G4 TAURUS runs: hybrid, ACFL – O3 (2/2)



- r0: mpi_96_omp_1_thread
- r1: mpi_48_omp_2_threads
- r2: mpi_24_omp_4_threads
- r3: mpi_12_omp_8_threads

https://datafront.exascale-computing.eu/public/OpenRadioss/TAURUS10M/G4_AWS/acfl/OpenRadioss_TAURUS10M-short_g4-aws_o1-8_m12-96_acfl_engine/

GLOBAL CHARACTERISTICS TAURUS ACFL –O3 ON G4 (2)

Loop id	Source Location	Source Function	Level	Exclusive Coverage mpi_48_ranks_omp_2_threads (%)	Vectorization Ratio (%)	Vector Length Use (%)
12756	engine_linuxa64_gf_ompi - czproj.F:1205-1462 [...]	czprojn	Single	4.32	4.82	52.33
6583	engine_linuxa64_gf_ompi - asspar4.F:172-172	asspar4	Innermost	3.88	100	100
11257	engine_linuxa64_gf_ompi - cupdtn3.F:637-690 [...]	cupdtn3p	Innermost	3.50	40.68	70.34
30855	engine_linuxa64_gf_ompi - i7optcd.F:200-233 [...]	i7optcd	Innermost	2.64	0	45.54
30830	engine_linuxa64_gf_ompi - i7buce_crit.F:133-147	i7buce_crit	Innermost	2.01	6.25	52.6
6802	engine_linuxa64_gf_ompi - depla.F:95-103	depla	Innermost	1.96	50	75
31178	engine_linuxa64_gf_ompi - gravit.F:157-159	gravit	Innermost	1.91	0	46.83
6883	engine_linuxa64_gf_ompi - vitesse.F:80-85	vitesse	Innermost	1.63	41.67	75
23947	engine_linuxa64_gf_ompi - i7cor3.F:127-176	i7cor3	Innermost	1.61	12.04	52.19
12489	engine_linuxa64_gf_ompi - czfintn.F:355-445	czfintn1	Innermost	1.54	100	100
30835	engine_linuxa64_gf_ompi - i7buce_crit.F:155-169	i7buce_crit	Innermost	1.44	6.52	39.67
30856	engine_linuxa64_gf_ompi - i7optcd.F:200-238	i7optcd	InBetween	1.37	0	48.03
10865	engine_linuxa64_gf_ompi - cbilan.F:111-152	cbilan	Single	1.33	18.03	59.02
6887	engine_linuxa64_gf_ompi - vit	vitesse	Innermost	1.29	41.67	75

https://datafront.exascale-computing.eu/public/OpenRadioss/TAURUS10M/G4_AWS/gnu/OpenRadioss_TAURUS10M-short_g4-aws_o2_m48_gnu_engine/

LAMMPS on Graviton 3 (64 cores) AWS: Armclang versus GCC

	ARMCLANG 1 - O3 -mcpu=neoverse-v1 -armpl -ffast-math	ARMCLANG 2 - O3 -mcpu=neoverse-v1 -armpl -msve-vector-bits=256 -ffast-math	ARMCLANG 4 - O3 -mcpu=neoverse-v1 +nosve+nosve2 -armpl -ffast-math	ARMCLANG 5 - O3 -mcpu=neoverse-v1 -armpl -fno-vectorize -fno-slp-vectorize -fno-openmp-simd -ffast-math	ARMCLANG 6 -O2 -mcpu=neoverse-v1 -armpl -ffast-math	GCC1 -O3 -mcpu=neoverse-v1 -larmpl -funroll-loops -ffast-math	GCC2 -O3 -mcpu=neoverse-v1 -larmpl -msve-vector-bits=256 -funroll-loops -ffast-math	GCC3 -O3 -mcpu=neoverse-v1 -larmpl -msve-vector-bits=128 -funroll-loops -ffast-math	GCC4 -O3 -mcpu=neoverse-v1+nosve+nosve2 -larmpl -funroll-loops -ffast-math	GCC5 -O3 -mcpu=neoverse-v1 -larmpl -fno-tree-vectorize -fno-openmp-simd -funroll-loops -ffast-math	GCC6 -O2 -mcpu=neoverse-v1 -larmpl -funroll-loops -ffast-math
Metric	**ARMCLANG 1**	**ARMCLANG 2**	**ARMCLANG 4**	**ARMCLANG 5**	**ARMCLANG 6**	**GCC 1**	**GCC 2**	**GCC 3**	**GCC 4**	**GCC 5**	**GCC 6**
Total Time (s)	420,57	422,54	423,42	430,12	421,46	434,58	433,02	433,38	434,74	432,86	440,65
Best	1,00	1,00	1,01	1,02	1,00	1,03	1,03	1,03	1,03	1,03	1,05
Max (Thread Active Time) (s)	418,66	420,65	421,57	428,29	419,53	432,68	431,32	431,52	432,7	431,22	438,59
Average Active Time (s)	418,43	420,29	421,23	427,84	419,19	432,35	430,79	431,13	432,43	430,67	438,33
Activity Ratio (%)	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5	99,5
Average number of active threads	63,67	63,66	63,67	63,66	63,66	63,67	63,67	63,67	63,66	63,68	63,66
Affinity Stability (%)	99,9	100	99,9	100	99,9	99,9	100	100	99,9	100	99,9
Time in analyzed loops (%)	97,8	97,9	97,7	97,7	97,9	97,7	97,9	97,8	97,8	97,7	97,8
ime in analyzed innermost loops (%	65,7	65,6	65,3	94,5	65,8	84,2	84,6	84,6	84,5	84,4	85,3
Time in user code (%)	98,1	98,2	97,9	98,1	98,2	98	98,1	98	98	98	98
Compilation Options Score (%)	100	100	100	100	87,5	100	100	100	100	100	87,5
Array Access Efficiency (%)	62,5	62,6	63	44,5	61,5	44,3	44,3	44,3	44,3	44,3	44,5

BEST: TIMING RATIOS: current timing over best timing across all compilers and compiler options.

- Green cells corresponds to best or very close to best (< 1%)
- Yellow cells correspond to performance losses < 3%
- Red cells correspond to performance losses > 3%

Hardware/Software platforms

Model Name	Frequency (GHz)	Number of cores/socket	Number of sockets	L1D (KB)	L1I (KB)	L2 (KB)	L3 (MB)	L3/Core (MB)
Neoverse N1 Ampere ALTRA Q80-30 (CALMIP)	3	80	1	64	64	1024	32	0,40
Neoverse V1 G3E (AWS)	2,6	64	1	64	64	1024	32	0,50
Neoverse V2 G4 (AWS)	2,8	96	1	64	64	2048	36	0,38

Neoverse N1: Arm C/C++/Fortran Compiler version 22.1 (build number 12) (based on LLVM 13.0.1)

Neoverse V1/V2: ACfL 24.10, GNU Fortran2008 13.2.0

Compiler options: O2, O3, O3 + no-sve, O3 + no-sve2

TALK FOCUS: AWS Graviton 4 Results