HPC Benchmark Game: Comparing Programming Languages Regarding Energy-Efficiency

for Applications from the HPC Field

Max Lübke[1], Dorian Stoll[1], Bettina Schnor[1], Stefan Petri[2]

[1]University of Potsdam, Institute of Computer Science

[2]Potsdam Institute for Climate Impact Research

September 22, 2025

Motivation

- Energy efficiency is important (for HPC ...) $\rightarrow$ see Green500 list
  "The Green500 is a biannual ranking of supercomputers, from the TOP500 list of supercomputers, in terms of energy efficiency. The list measures performance per watt using the TOP500 measure of high performance LINPACK benchmarks at double-precision floating-point format." [from Wikipedia]
- Inspired by Pereira et al.'s research on runtime vs. energy efficiency in the Computer Language Benchmarks Game (CLBG) and the Rosetta Code.
- Most of those are not representative for HPC applications
- HPC Benchmark Game aims to address the gap with HPC-relevant languages and benchmarks

HPC Benchmark Game

Languages chosen: C, C++, Fortran, Julia

HPC Benchmark Game

Languages chosen: C, C++, Fortran, Julia

Research Questions:

1. Comparison between JIT-compiled and native compiled code
2. Impact of the compiler on energy efficiency
3. Impact of hardware architecture and over-threading on energy efficiency

HPC Benchmark Game

Languages chosen: C, C++, Fortran, Julia

Research Questions:

1. Comparison between JIT-compiled and native compiled code
2. Impact of the compiler on energy efficiency
3. Impact of hardware architecture and over-threading on energy efficiency

Points we did not consider:

- Memory vs. Energy consumption (Pereira et al.: 89% from CPU)
- Frequency scaling policy – just used default installed governor
- Vector instruction widths (AVX2 vs. AVX512)
- Platform-specific OpenMP thread mappings
- Influence of problem size

HPC Benchmark Game
Criteria

Benchmarks selection criteria:

- non-trivial mathematical algorithm
- shared-memory parallelization
- and less than 2000 LOC.

HPC Benchmark Game

Benchmarks

One Reference benchmark chosen from each language:

- C: Cellular Automaton (CA) (based on Conways Game of Life).
  9-point stencil, 8-bit integer operations.
- C++: Transport on uniform grids (TUG).
  Solves diffusion with an alternating direction implicit method. (GFZ and Uni Potsdam)
- Fortran: NAS Parallel Benchmarks FT
  (NPB – Fourier Transform)
- Julia: SRAD (Rodinia benchmark)
  SRAD is a diffusion method for ultrasonic and radar imaging applications based on partial differential equations and is used to remove locally correlated noise.

HPC Benchmark Game

Re-Implementation Details

- Idiomatic re-implementation of the benchmarks
- Standardized pseudo-random number generation, for reproducibility of results across languages.
- OpenMP parallelization; Julia `FLoops`
- C++ used Eigen library
- Bounds checking disabled (C++ Eigen and Julia)
- Plain C-style arrays for C
- Correctness was verified (reproducibility across languages, independent of parallelization)

HPC Benchmark Game

Re-Implementation Details

- Idiomatic re-implementation of the benchmarks
- Standardized pseudo-random number generation, for reproducibility of results across languages.
- OpenMP parallelization; Julia `FLoops`
- C++ used Eigen library
- Bounds checking disabled (C++ Eigen and Julia)
- Plain C-style arrays for C
- Correctness was verified (reproducibility across languages, independent of parallelization)

Time and energy measurements by: EMA

RAPL (Running Average Power Limit)

- Control and real-time measurements of CPU and RAM energy consumption
- From Intel, since Sandy Bridge architecture, 2011
- Our AMD CPUs provide data for “core” and for “package” domains, but not for dram memory nor I/O
- Measured every microsecond, in microJoule
- On Linux accessible via sysfs virtual file system `/sys/class/powercap/intel-rapl*/energy_uj`
- Here, we add energy values for both ”package” domains.

Testbed

Measurements on HPC2024 cluster at PIK:

- Two AMD EPYC 9554 “Genoa” CPUs (2×64 cores)
- 6 GB RAM per core $\rightarrow$ 768 GB DDR5 RAM total
- 8 cores share one L3 cache, 16 cores share 3 memory channels
- “zen4” core architecture: $2 \times$ AVX2 or $1 \times$ AVX512 vector engines

Testbed

Measurements on HPC2024 cluster at PIK:

- Two AMD EPYC 9554 “Genoa” CPUs (2×64 cores)
- 6 GB RAM per core $\rightarrow$ 768 GB DDR5 RAM total
- 8 cores share one L3 cache, 16 cores share 3 memory channels
- “zen4” core architecture: $2 \times$ AVX2 or $1 \times$ AVX512 vector engines

$\Rightarrow$ Use all 128 cores of one node for the measurements.

Testbed

Measurements on HPC2024 cluster at PIK:

- Two AMD EPYC 9554 “Genoa” CPUs (2×64 cores)
- 6 GB RAM per core $\rightarrow$ 768 GB DDR5 RAM total
- 8 cores share one L3 cache, 16 cores share 3 memory channels
- “zen4” core architecture: $2 \times$ AVX2 or $1 \times$ AVX512 vector engines

$\Rightarrow$ Use all 128 cores of one node for the measurements.

Four Compilers used:

- Intel Classic oneAPI 2023.2.0 (icc, ifort; `avx2`-optimization)
- Intel oneAPI 2024.2.1 (icx, ifx; `avx2`-optimization)
- AOCC 4.2.0 (`zen4`-optimization)
- GCC 14.1.0 (`zen4`-optimization)

Testbed

Measurements on HPC2024 cluster at PIK:

- Two AMD EPYC 9554 “Genoa” CPUs (2×64 cores)
- 6 GB RAM per core $\rightarrow$ 768 GB DDR5 RAM total
- 8 cores share one L3 cache, 16 cores share 3 memory channels
- “zen4” core architecture: $2 \times$ AVX2 or $1 \times$ AVX512 vector engines

$\Rightarrow$ Use all 128 cores of one node for the measurements.

Four Compilers used:

- Intel Classic oneAPI 2023.2.0 (icc, ifort; `avx2`-optimization)
- Intel oneAPI 2024.2.1 (icx, ifx; `avx2`-optimization)
- AOCC 4.2.0 (`zen4`-optimization)
- GCC 14.1.0 (`zen4`-optimization)

Julia version 1.11.0 – package and app (`-O3 -C znver4 --check-bounds=no`)

Testbed

Measurements on HPC2024 cluster at PIK:

- Two AMD EPYC 9554 “Genoa” CPUs (2×64 cores)
- 6 GB RAM per core $\rightarrow$ 768 GB DDR5 RAM total
- 8 cores share one L3 cache, 16 cores share 3 memory channels
- “zen4” core architecture: $2 \times$ AVX2 or $1 \times$ AVX512 vector engines

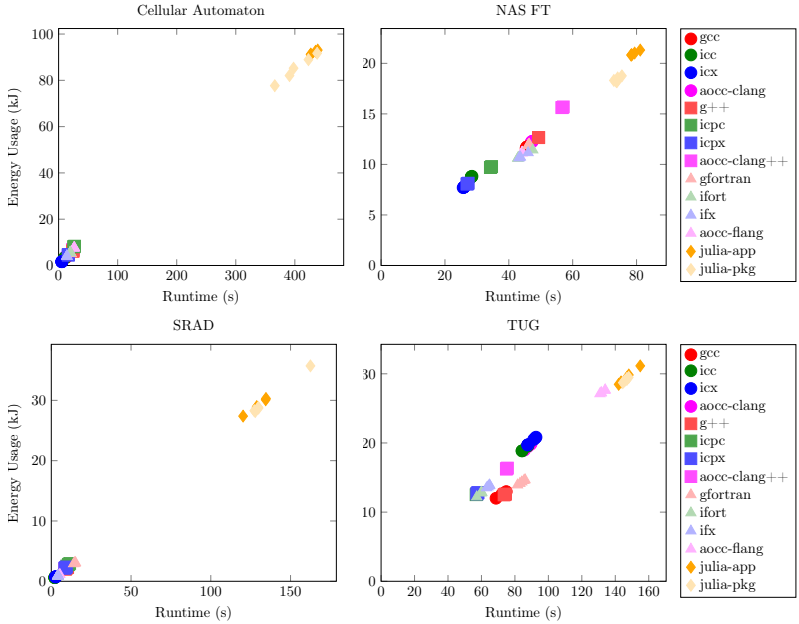
$\Rightarrow$ Use all 128 cores of one node for the measurements.

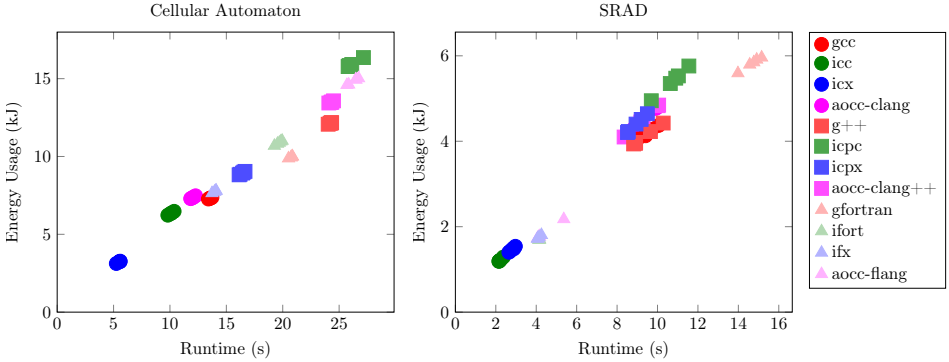
Four Compilers used:

- Intel Classic oneAPI 2023.2.0 (icc, ifort; `avx2`-optimization)
- Intel oneAPI 2024.2.1 (icx, ifx; `avx2`-optimization)
- AOCC 4.2.0 (`zen4`-optimization)
- GCC 14.1.0 (`zen4`-optimization)

Julia version 1.11.0 – package and app (`-O3 -C znver4 --check-bounds=no`)

$\Rightarrow$ All measurements repeated 5 times.

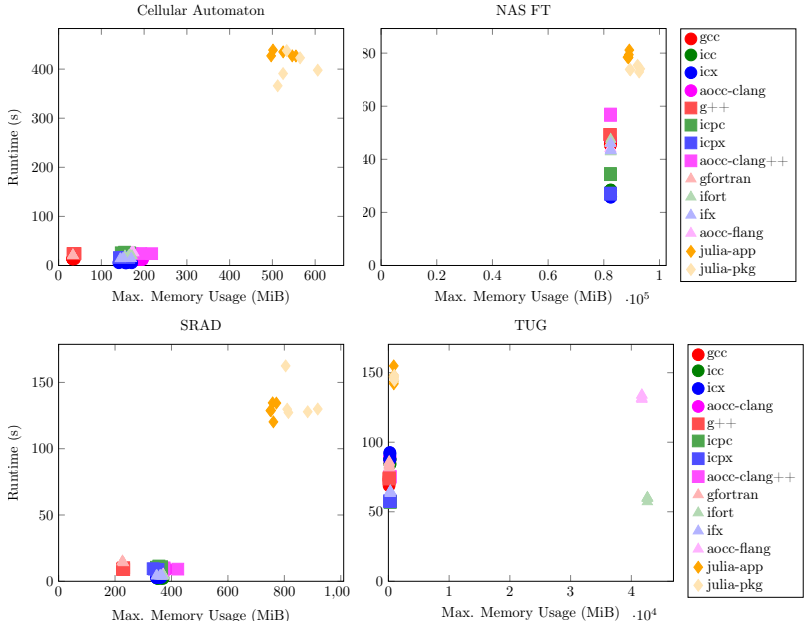

Cellular Automaton
NAS FT
SRAD
TUG
Energy Usage (kJ)
Runtime (s)
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
julia-app
julia-pkg

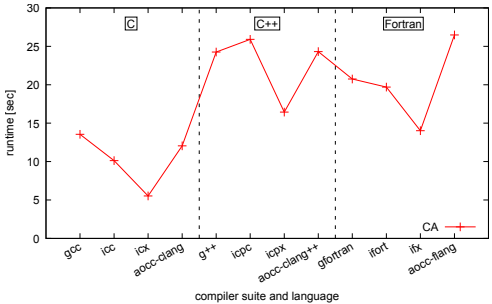

Cellular Automaton
Energy Usage (kJ)
Runtime (s)
0
5
10
15
0
5
10
15
20
25
SRAD
Energy Usage (kJ)
Runtime (s)
0
2
4
6
0
2
4
6
8
10
12
14
16
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang

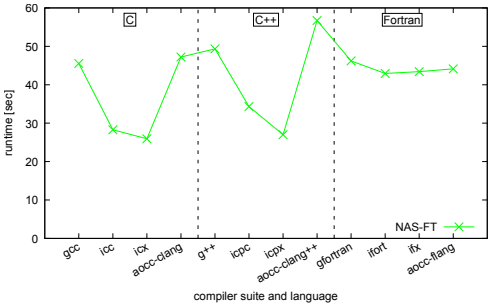
Research Question 1

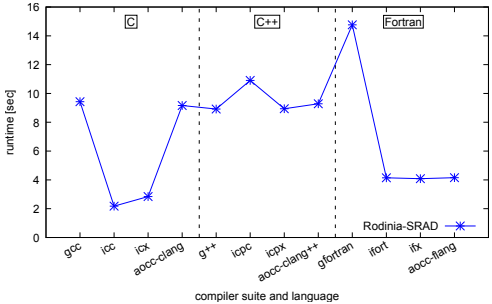
Comparison between JIT-compiled and native compiled code

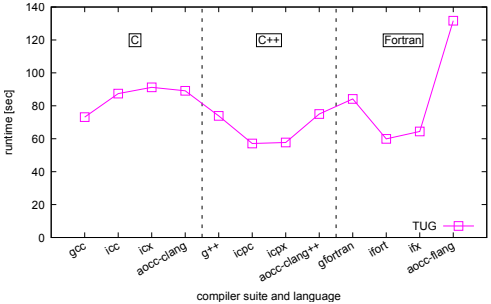
- Energy consumption is roughly linear with runtime
- Julia is not competitive with C/C++/Fortran
- Julia app approach did not improve performance over the packaged approach

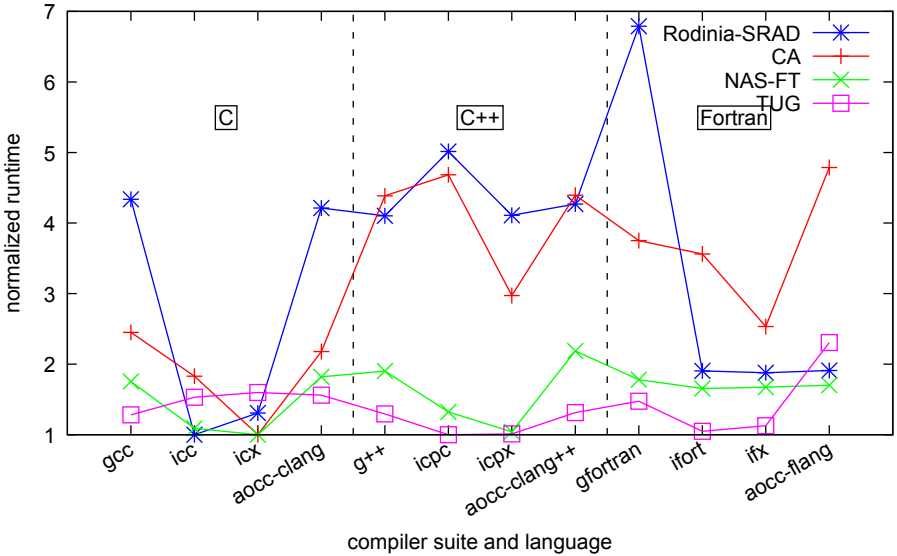

Cellular Automaton
NAS FT
SRAD
TUG
Runtime (s)
Max. Memory Usage (MiB)
·10^5
·10^4
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
julia-app
julia-pkg


C
C++
Fortran
runtime [sec]
CA
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
compiler suite and language


C
C++
Fortran
runtime [sec]
NAS-FT
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
compiler suite and language


C
C++
Fortran
runtime [sec]
Rodinia-SRAD
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
compiler suite and language


C
C++
Fortran
runtime [sec]
TUG
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
compiler suite and language


normalized runtime
compiler suite and language
C
C++
Fortran
Rodinia-SRAD
CA
NAS-FT
TUG
gcc
icc
icx
aocc-clang
g++
icpc
icpx
aocc-clang++
gfortran
ifort
ifx
aocc-flang
1
2
3
4
5
6
7

Benchmark	“Winner”
CA	C/icx
NAS FT	C/icx, C/icc, C++/icpx
SRAD	C/icc, C/icx
TUG	C++/icpc, C++/icpx, Fortran/ifort, Fortran/ifx

Research Question 2

Impact of the compiler on energy efficiency

- Compiler impact is application-dependent
- Larger runtime variance for SRAD and CA
- Compiler choice can have a bigger impact than language choice
- AVX2 optimization in Intel compilers outperforms “zen4” in GNU/AMD compilers [1]

[1] Treat with caution

Research Question 3

Impact of hardware architecture and over-threading on energy efficiency

Scaling results for TUG:

# Threads	1	2	4	8	16	32	64	128
Runtime [s]	174	122	87	70	62	59.1	**58.9**	59.6
Energy [kJ]	56	40	29	23	21	21	22	25

Global Results

Normalized Runtime		Normalized Energy	
C/icx	1.20	C/icx	1.20
C/icc	1.32	C/icc	1.37
Fortran/ifx	1.73	Fortran/ifx	1.51
Fortran/ifort	1.85	Fortran/ifort	1.60
C++/icpx	1.89	C++/icpx	1.81
C++/g++	2.11	C/gcc	1.85
C/gcc	2.21	C++/g++	2.11
C/aocc-clang	2.26	C/aocc-clang	2.14
C++/aocc-clang++	2.26	C++/aocc-clang++	2.14
C++/icpc	2.36	Fortran/aocc-flang	2.15
Fortran/aocc-flang	2.45	Fortran/gfortran	2.26
Fortran/gfortran	2.86	C++/icpc	2.30
Julia/pkg	13.32	Julia/pkg	10.66
Julia/app	13.74	Julia/app	11.23

(geometric mean over normalized results)

Conclusion

- Confirm correlation between runtime and energy consumption
- Julia is not competitive with native-compiled languages
- Negative effect of over-threading on energy-efficiency is shown
- Compiler has higher impact than chosen programming language

$\rightarrow$ Make users aware to try different compilers

Conclusion

- Confirm correlation between runtime and energy consumption
- Julia is not competitive with native-compiled languages
- Negative effect of over-threading on energy-efficiency is shown
- Compiler has higher impact than chosen programming language

→ Make users aware to try different compilers

Thank you!

References 1

Green500 https://top500.org/lists/green500/

Pereiraetal17 Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, João Saraiva: *Energy efficiency across programming languages: how do energy, time, and memory relate?*. SLE 2017: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, pp 256–267. https://doi.org/10.1145/3136014.3136031

Pereiraetal21 Rui Pereira, Marco Couto, Francisco Ribeiro, Rui Rua, Jácome Cunha, João Paulo Fernandes, João Saraiva: *Ranking programming languages by energy efficiency*. Science of Computer Programming, Vol 205, 102609, May 2021. https://doi.org/10.1016/j.scico.2021.102609

CLBG The Computer Language Benchmarks Game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html

Rosetta Code http://rosettacode.org/

References 2

WhatisRAPL Mateusz: What is RAPL. Web site, Green Compute UK, 23. August 2024. https://greencompute.uk/Measurement/RAPL

EMA PERFACCT GmbH: EMA - Energy Measurement for Applications. https://github.com/PERFACCT/EMA

CELL+25 S. Christgau et al: On the Usability and Energy Efficiency of High-Level Synthesis for FPGA-based Network-Attached Accelerators. In: 2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 886–895. Milano, Italy (in press) (2025). https://doi.org/10.1109/IPDPSW66978.2025.00139

CA https://www.cs.uni-potsdam.de/bs/research/labsCa.html

TUG https://git.gfz-potsdam.de/naaice/tug

NBP NAS Parallel Benchmarks. https://www.nas.nasa.gov/software/npb.html

Rodinia Rodinia Benchmark Suite. Fork at https://github.com/JuliaParallel/rodinia (the original site at virginia.edu is gone)

References 3

Stoll24 Dorian Stoll: Performance und Energiebedarf von HPC-Anwendungen in Abhängigkeit der gewählten Programmiersprache. Project Report, University of Potsdam, 2024. https://www.cs.uni-potsdam.de/bs/teaching/docs/lectures/2024/stoll.pdf

LSSP25 Max Lübke, Dorian Stoll, Bettina Schnor, Stefan Petri: *HPC Benchmark Game: Comparing programming languages regarding energy-efficiency for applications from the HPC field*. In: PECS2025 – Workshop on Performance and Energy Efficiency in Concurrent and Distributed Systems, Dresden, 26 Aug 2025. Springer LNCS, in press. https://pecs-workshop.github.io/2025

LSSP25data Max Lübke, Dorian Stoll, Bettina Schnor, Stefan Petri: *HPC Benchmark Game Source Code and Results (PECS 2025)*. [code] Zenodo, July 2025, https://doi.org/10.5281/zenodo.15785010

HBG HPC Benchmark Game, GIT Repository. https://gitup.uni-potsdam.de/bsvs/public/hpc-benchmark-game