



Chapter 2

The FEniCS Project on AWS Graviton3

Michal Habera and Jack S. Hale

Abstract ARM architecture central processing units are increasingly prevalent in high-performance computers due to their energy efficiency, scalability, and cost-effectiveness. The overall goal of this study is to evaluate the suitability of ARM-based cloud computing instances in executing finite element computations. Specifically, we present performance results for running the FEniCS Project finite element software on Amazon Web Services (AWS) c7g and c7gn instances with Graviton3 processors. These processors support the ARMv8.4-A instruction set with Scalable Vector Extension (SVE) for Single Instruction Multiple Data operations and the Elastic Fabric Adaptor for communications between instances. Both clang 18 and GNU Compiler Collection 13 compilers successfully generated optimised code using SVE instructions, ensuring that users can achieve optimised performance without extensive manual tuning. Testing a distributed memory parallel DOLFINx Poisson solver with up to 512 Message Passing Interface processes, we find that the performance and scalability of the AWS instances are comparable to those of a dedicated AMD EPYC Rome cluster installed at the University of Luxembourg. These findings demonstrate that ARM-based cloud computing instances, exemplified by AWS Graviton3, can be competitive for distributed memory parallel finite element analysis.

Michal Habera e-mail: michal.habera@rafinex.com
Rafinex SARL, and Institute of Computational Engineering, Department of Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg

Jack S. Hale e-mail: jack.hale@uni.lu
Institute of Computational Engineering, Department of Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg, Luxembourg

2.1 Introduction

The FEniCS Project (Alnæs et al., 2015; Baratta et al., 2023b) has been used to write finite element solvers for problems in fields that involve solving partial differential equations, including mathematics, biology, physics, engineering, geophysics, and mechanics.

Exploring ARM-based processors and cloud computing instances for executing FEniCS Project-based solvers is worthwhile due to ARM’s potential advantages in cost-effectiveness, energy efficiency, and scalability with respect to x86-64-based machines (Simakov et al., 2023; Suárez et al., 2024). Examples of ARM adoption in the high-performance computing (HPC) space include the Isambard project (Isambard 3, NVIDIA Grace, (BCS, 2025)), the Mont-Blanc project (Phase 3, Cavium Thunder X2, (Rajovic et al., 2016)), the Fugaku supercomputer (Fujitsu A64FX, (Fujitsu, 2024)), and the Astra supercomputer (Cavium ThunderX2, (Sandia, 2018)). Publicly available cloud services with ARM instances include those of Amazon Web Services AWS (Graviton3 CPU based on Neoverse V1 and Graviton4 CPU with Neoverse V2), Google Cloud (Axion CPU based on Neoverse V2, (Google, 2025)), and Microsoft Azure (Azure Cobalt 100 based on Neoverse N2, (Microsoft, 2024)).

AWS Graviton3-based instances aim to provide cost-effective computing resources for scientific computing and machine learning applications by including both Scalable Vector Extension (SVE) instructions for Single Instruction Multiple Data (SIMD) parallelism and the Elastic Fabric Adaptor (EFA) interconnect for high-bandwidth low-latency communication between instances. This makes the AWS cloud offering particularly appealing for executing scientific computing code such as the FEniCS Project.

A key technology in the FEniCS Project is the use of automatic code generation (compilation). The user expresses a finite element problem in Unified Form Language (UFL) (Alnæs et al., 2014), and the FEniCSx Form Compiler (FFCx) (Kirby and Logg, 2006) then compiles the UFL description of the problem into a low-level C kernel for computing the cell-local finite element tensor.

One aspect of good performance of a compute-bound kernel is ensuring the assembly code of the compiled kernel contains calls to Single Instruction Multiple Data (SIMD) operations. SIMD operations can apply the same operation to multiple data items in a single CPU clock cycle. For a recent overview of SIMD programming strategies, see, for example, Rocke (2023). The current strategy of FFCx with respect to SIMD is to ensure that its kernels are amenable to the compiler applying automatic vectorisation, a process that automatically converts a scalar program into a vectorised equivalent that uses SIMD operations.

Consequently, to achieve good performance when using FEniCSx on Graviton3, it is important that users verify that the latest compilers automatically produce SVE and/or Neon SIMD instructions when compiling the generated C finite element kernels and that these kernels display reasonable runtime performance.

In addition to SIMD parallelisation at the kernel level, DOLFINx, the FEniCS Project’s finite element problem solving environment, also supports distributed memory parallel assembly of global finite element data structures (sparse matrices and vectors) using the Message Passing Interface (MPI; for full details, see Baratta et al. (2023b)). Users running large-scale DOLFINx simulations on AWS must verify that the EFA interconnect’s performance is sufficient for parallel scalability.

In summary, the contribution of this chapter is to examine both the SIMD performance and multi-node parallel scaling of the FEniCS Project’s software on Amazon’s Graviton3-based instances.

2.2 Methodology and Results

2.2.1 Systems

AWS c7g and c7gn instances are compared to the Aion computing instances available at the University of Luxembourg’s HPC facilities (Varrette et al., 2022). These instances have different hardware configurations (see Table 2.1 for full details).

The FEniCS Project components are written in a mixture of Python, ‘modern’ C++20 and Standard C17. The Python interface is a wrapper around the core data structures and computationally intensive algorithms written in C and C++.

	Aion node	AWS c7g instance
Processor	2 x (AMD EPYC ROME 7H12, 64 cores @ 2.6 GHz)	1 x (Graviton3, 64 cores @ 2.6 GHz)
Architecture	x86_64, Zen 2 (AVX2)	ARMv8.4-A, Neoverse V1 (SVE)
Memory	256 GB DDR4 3200 MT/s = 25.6 GB/s 8 non-uniform memory access (NUMA) nodes	128 GB DDR5 4800 MT/s = 38.4 GB/s Uniform memory access (no NUMA)
Total mem. bandwidth	2 x 200 GB/s	1 x 300 GB/s

Table 2.1: Configuration of the Aion nodes (University of Luxembourg’s HPC facilities) and AWS c7g (Amazon) instances. The c7gn instance used in the Poisson weak scaling test has the same hardware as c7g with the addition of a 200 GB s⁻¹ interconnect between instances for MPI-based communication.

2.2.2 Memory Bandwidth

Low-order finite element methods are typically memory bandwidth constrained, since the time to load and store data from main memory (e.g. the mesh geometry) dominates the time to perform the arithmetic operations to compute the finite element cell tensor. Understanding a processor’s memory bandwidth is therefore important for ensuring optimal performance.

STREAM (McCalpin, 1995, 1991–2007) is the industry standard benchmark for measuring sustained memory bandwidth performance, with McCalpin estimating memory bandwidth from memory-intensive operations (copy, scale, add) on large contiguous arrays.

Figure 2.1 shows the results for the copy operation for a single-node benchmark. For the single-node benchmark, 80% of the theoretical peak memory bandwidth of 400 GB s^{-1} for Aion and 300 GB s^{-1} for AWS c7g is reached. This is considered a reasonable outcome of the STREAM benchmark (McCalpin, 2023). Bandwidth saturation is observed at around 20% of the node utilization. Both curves show different saturation point characteristics due to different memory access configurations. On the Aion instances, there are eight NUMA nodes of 16 cores each, while AWS c7g instances are set up with unified memory access.

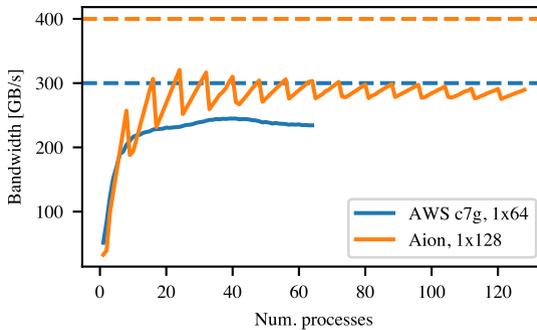


Fig. 2.1: Single-node STREAM benchmark. The theoretical peak bandwidth of each system is shown as a dashed line.

2.2.3 Finite Element Kernels

To measure the performance of standard FEniCS user finite element code, we use the Local Finite Element Operator Benchmarks repository (Baratta et al., 2023a). The benchmark measures the execution time of a local finite element kernel generated by the FFCx v0.9.0 (Kirby and Logg, 2006). We generate a matrix-free three-

dimensional Laplace kernel representing finite element discretization of the action of Laplace operator A_{ij} with spatially varying material property $\kappa(x)$:

$$v_i = A_{ij}w_j, \quad A_{ij} = \int_K \kappa J_{mk} J_{mn} \nabla_k \phi_i \nabla_n \phi_j |\det J| dx, \quad (2.1)$$

where K is a fixed reference tetrahedron; $w_j \in \mathbb{R}^n$ is a fixed, prescribed vector; J is a Jacobian transformation matrix; and the ϕ terms are finite element basis functions.

The generated kernel calculates a double-precision vector $v_i \in \mathbb{R}^n$, where $n = 4$ for first-order (low-order) discretization and $n = 165$ for eighth-order (high-order) discretization. Low-order kernels are expected to be memory bandwidth limited, while high-order kernels have higher arithmetic intensity. In addition, the matrix-free (operator action) version requires fewer load and store operations in comparison to the assembly of a matrix, increasing the ratio of floating-point operations to memory loads and stores. Consequently, significant performance improvements are possible for high-order kernels if the compiler can automatically emit SIMD operations.

Generated Code Structure. Compiler (loop) SIMD auto-vectorisation is usually performed for innermost loops with known compile time bounds. Analysis of the FFCx autogenerated code is required to understand the potential and determine missed optimisations.

Code Listing 2.1: Abbreviated FFCx-generated finite element kernel

```
void kernel(double* restrict A, const double* restrict w, ...){
    // 1. Static arrays of basis functions and quadrature weights
    .
    // 2. Quadrature rule--independent computations.

    for (int iq = 0; iq < NUM_QUAD_POINTS; ++iq) {
        // 3. Quadrature loop body.
        for (int ic = 0; ic < NUM_DOFS; ++ic){
            // 3.1 Coefficient evaluation.
            w1_d100 += w[4 + (ic)] * FEO_CO_D100_Q530 [0] [0] [iq] [
ic];
            // ...
        }

        // 3.2 Scalar graph evaluation.
        double sv_530_0 = w1_d100 * sp_530_18;
        double sv_530_1 = w1_d010 * sp_530_22;
        // ...

        for (int i = 0; i < NUM_DOFS; ++i) {
            // 3.3 Tensor assignment loop.
            A[(i)] += fw0 * FEO_CO_D100_Q530 [0] [0] [iq] [i];
            // ...
        }
    }
}
```

An abbreviated example of generated C code is shown in code listing 2.1. First, there are arrays defining finite element basis functions at quadrature points. These require no arithmetic operations. Computations independent of the quadrature loop contain more intense arithmetic operations (e.g. determinant of the Jacobian) but are executed only once. Non-affine geometry would require the evaluation of geometric quantities at each quadrature point, which would increase the arithmetic intensity and yield more opportunities for vectorisation.

The most performance-critical part of the code is contained in the quadrature loop body. For the eighth-order Laplace operator, `NUM_QUAD_POINTS = 214` and `NUM_DOFS = 165`. There are two innermost loops: coefficient evaluation and tensor assignment. Both contain a set of multiply–add operations that are candidates for automatic vectorisation via fused multiply–add operations in both the SVE (Graviton3) and AVX2 (AMD EPYC) cases.

Experimental Results. For the finite element kernel benchmarks, we compile the kernels with LLVM/clang 18.1.3 and GNU Compiler Collection (GCC) 13.2.0. Full details are provided in Table 2.2.

	Compiler	Aion	AWS c7g
Ofast, native, vectorised	GCC 13.2.0	-Ofast -march=znver2 -mtune=znver2	-Ofast -mcpu=neoverse-v1
	clang 18.1.3	-Ofast -march=znver2 -mtune=znver2	-Ofast -mcpu=neoverse-v1
Ofast, native, no vec.	GCC 13.2.0	-Ofast -march=znver2 -mtune=znver2 -fno-tree-vectorize	-Ofast -mcpu=neoverse-v1 -fno-tree-vectorize
	clang 18.1.3	-Ofast -march=znver2 -mtune=znver2 -fno-slp-vectorize -fno-vectorize	-Ofast -mcpu=neoverse-v1 -fno-slp-vectorize -fno-vectorize
O2, no vec.	GCC 13.2.0	-O2 -fno-tree-vectorize	-O2 -fno-tree-vectorize
	clang 18.1.3	-O2 -fno-slp-vectorize -fno-vectorize	-O2 -fno-slp-vectorize -fno-vectorize

Table 2.2: Compiler versions and compilation flags used for finite element kernel benchmarks

The kernel benchmark results are presented in Figs. 2.2 and 2.3. Low-order kernels (Fig. 2.2) show no dependence on the compiler vectorisation setup. On the other hand, AWS c7g demonstrates a $1.3\times$ speedup over Aion, which we attribute to greater memory bandwidth for a single process.

High-order kernels (Fig. 2.3), which are expected to benefit from SIMD operations, reveal a clear link between compiler settings and performance. Both clang and GCC auto-vectorisers perform well, producing a noticeable speedup ($>2\times$) in the most optimised setting. The vectorisation speedup ($>4\times$) is more significant with the Aion nodes.

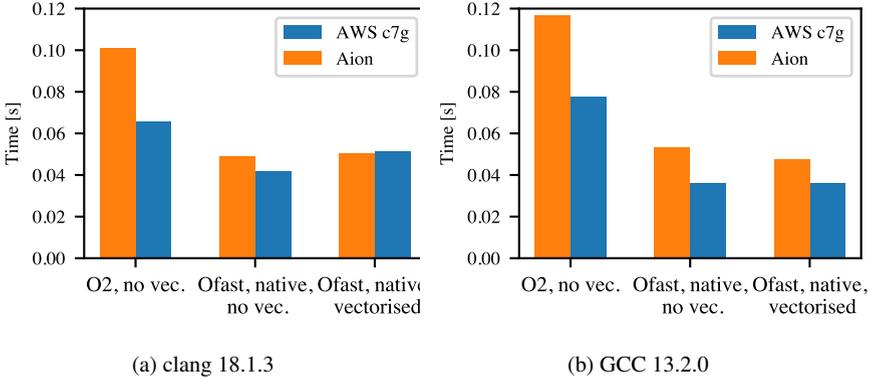


Fig. 2.2: Low-order Laplace operator action assembly

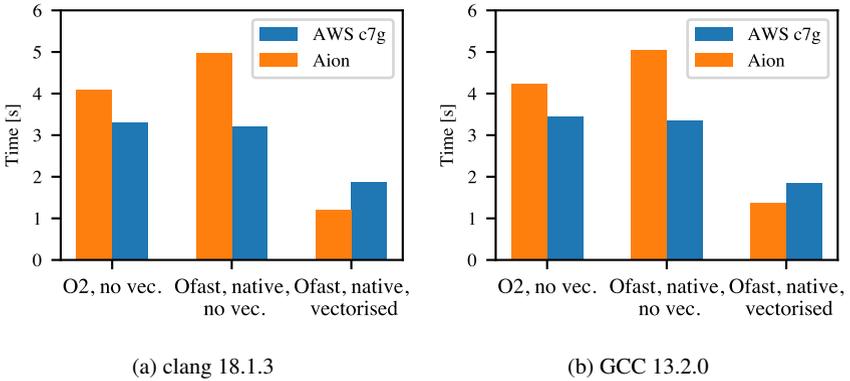


Fig. 2.3: High-order Laplace operator action assembly

Optimisation reports (with `-Rpass=loop-vectorize` for clang, and `-fopt-info-vec-optimised` for GCC) and analysis of the generated assembly reveal that, for low-order operator action, using the `-Ofast` compiler optimisation level enhances constant folding and allows more operations to be computed at compile time (e.g. partial sums of the static constant arrays) (Godbolt, 2024e).

On Graviton3, both GCC and clang generate SVE floating-point fused multiply–add (FMLA) instructions (Arm, 2024) for both the coefficient evaluation and tensor assignment loops (Godbolt, 2024b,a). FMLA is a SIMD instruction that multiplies two vectors stored in SVE registers and adds the result to a third vector. The coefficient evaluation loop with no interdependencies between iterations is a perfect example of compiler auto-vectorisation. Moreover, for higher-order discretization, there is potential for exploiting wider SVE registers (up to 2,048 bits).

An assembly excerpt for the coefficient evaluation is shown below.

```
ld1d    {z0.d}, p0/z, [x7, x0, lsl #3]
ld1d    {z25.d}, p0/z, [x3, x0, lsl #3]
fmla    z3.d, p0/m, z25.d, z0.d
...
faddv   d1, p1, z1.d
```

As expected, there are two contiguous loads LD1D in two of the available SVE Z0–Z31 registers followed by a fused multiply–add instruction. The result is accumulated in an SVE register Z3 that is then horizontally summed outside of the vectorised loop (FADDV). Here P0 is a predicate register without any constraints on the available elements.

On Aion, both GCC and clang vectorise both the coefficient evaluation and tensor assignment loops and rely on the VFMADD231PD instructions on the YMM registers, that is, the vectorisation width of four doubles (Godbolt, 2024c,d).

2.2.4 Parallel Scalability

The parallel scalability results were obtained using performance test codes for FEniCSx (Wells and Richardson, 2023) built against DOLFINx 0.6.0 and PETSc 3.18 (Balay et al., 2023) with the Spack package manager setup, using GCC 12.2.0. We set up Spack to use a version of OpenMPI provided by AWS that includes the appropriate libfabric with native support for the EFA interconnect. Libfabric is a network communication library that abstracts networking technologies from fabric and hardware implementation, ensuring optimal data transfer across Amazon’s proprietary EFA interconnect.

The Poisson equation solver benchmark consists of the following measured steps:

1. Create a mesh. Create a unit cube mesh and discretize using linear tetrahedral cells. Partition the mesh with the ParMETIS 4.0.3 partitioner (Karypis and Kumar, 1998) and distribute.
2. Assemble the matrix. Execute the local Poisson equation kernel over the mesh and assemble into a PETSc MATMPIAIJ (distributed compressed sparse row) matrix.

3. Solve the linear system. Run the Conjugate Gradient solver with a classical algebraic multigrid (BoomerAMG; see (Falgout and Yang, 2002)) preconditioner.

Creating the mesh (including partitioning), assembling matrices, and solving the resulting linear system are typically the most expensive steps in a finite element solution. They also contain significant parallel communication steps that can highlight issues in either the finite element solver or the underlying MPI hardware/software stack, leading to poor parallel scaling. Weak scaling results (with a constant workload of approximately 5×10^5 degrees of freedom per process) are shown in Fig. 2.4. Both Aion and AWS c7gn show almost constant times for mesh creation ($< 5\%$ difference).

Matrix assembly is expected to have ideal weak parallel scalability due to the cell-local nature of the assembly loop and negligible MPI communication during matrix finalization. Aion and AWS c7gn show a small increase in time (10–15%) for 512 processes.

The time for the solving step increases by 40% for 512 processes on AWS c7gn and by 27% on Aion. However, the number of Krylov iterations of the preconditioned Conjugate Gradient solver grows from 16 to 20 for 512 processes (a 25% increase) due to the inefficiency of the algebraic multigrid preconditioner on an unstructured 3D mesh. Taking this into account, the time per iteration is almost constant on Aion ($< 5\%$), and a small increase of 15% on AWS c7gn is observed.

2.3 Conclusions

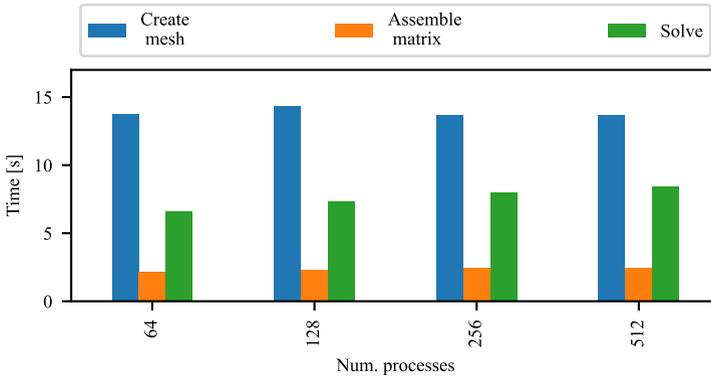
Benchmarks for the memory bandwidth, local finite element kernels, and parallel scalability of a Poisson solver were executed on Aion nodes and on AWS c7g(n) instances.

Memory bandwidth measured using STREAM MPI confirms the higher memory transfer rate of AWS c7g(n) but the superior total bandwidth of 310 GB s^{-1} per Aion node.

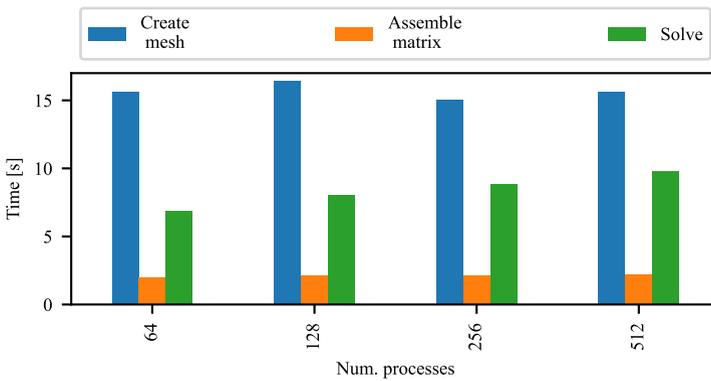
In terms of the auto-vectorisation capabilities of GCC 13.2.0 and clang 18.1.3, both produce optimised instructions for the targeted microarchitectures (Zen 2 for Aion and Neoverse V1 for AWS c7g). This observation is confirmed with performance benchmarks based on local finite element kernels for the Laplace operator.

The MPI-based distributed memory Poisson equation solver shows weak scaling with a 15% increase in time per iteration for 512 processes on the c7gn-based cluster. The results for the University of Luxembourg's Aion system are slightly superior, with almost constant ($< 5\%$ difference) times per iteration for 512 processes.

Based on our results, we conclude that AWS Graviton3 instances are a viable alternative for HPC tasks using the FEniCS Project's automated finite element solver.



(a) Aion, 5×10^5 degrees of freedom per process, 25% utilization (32 processes per node)



(b) AWS c7gn, 5×10^5 degrees of freedom per process, 50% utilization (32 processes per node)

Fig. 2.4: Weak parallel scalability of the DOLFINx Poisson equation solver on Aion and AWS c7gn systems

These instances are likely to be particularly interesting for users with infrequent or highly elastic large-scale computational demands (Emeras et al., 2016).

In the future, we plan to work on other, more complex problems (e.g. linear elasticity) and performance benchmarks of direct solvers. Additionally, the latest generation Graviton4 instances provide an improved Neoverse V2 instruction set, with a smaller SVE vector length of 128 bits, (Arm, 2025), warranting further investigation.

Supplementary Material

Raw data and plotting scripts are archived at Habera and Hale (2025).

Acknowledgements This project received computing resources from Amazon Web Services (AWS) through the first and second collaborative University of Luxembourg and AWS Graviton3 calls. The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg (Varrette et al., 2022); see <https://hpc.uni.lu>.

This research was funded in whole or in part, by the National Research Fund (FNR), grant reference COAT/17205623. For the purpose of open access, and in fulfilment of the obligations arising from the grant agreement, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

JSH declares that a family member was working at Rafinex during the period of this project. This person was not involved in this study.

References

- Alnæs MS, Blechta J, Hake JE, Johansson A, Kehlet B, Logg A, Richardson C, Ring J, Rognes ME, Wells GN (2015) The FEniCS Project Version 1.5. *Archive of Numerical Software* 3, doi:10.11588/ans.2015.100.20553
- Alnæs MS, Logg A, Ølgaard KB, Rognes ME, Wells GN (2014) Unified Form Language: A domain-specific language for weak formulations of partial differential equations. *ACM Trans Math Softw* 40(2):9:1–9:37, doi:10.1145/2566630
- Arm (2024) Arm architecture reference manual for A-profile architecture. <https://developer.arm.com/documentation/ddi0487/ka>, [Accessed 11-09-2024]
- Arm (2025) Arm neoverse v2 core technical reference manual. <https://developer.arm.com/documentation/102375/latest/>, [Accessed 11-01-2025]
- Balay S, et al. (2023) PETSc Web page. URL <https://petsc.org/>
- Baratta I, Richardson C, Dokken JS, Hermano A (2023a) Local finite element operator benchmarks. URL https://github.com/IgorBaratta/local_operator
- Baratta IA, Dean JP, Dokken JS, Habera M, Hale JS, Richardson CN, Rognes ME, Scroggs MW, Sime N, Wells GN (2023b) DOLFINx: The next generation FEniCS problem solving environment. doi:10.5281/zenodo.10447666
- BCS (2025) Specs - Bristol Centre for Supercomputing Documentation — docs.isambard.ac.uk. <https://docs.isambard.ac.uk/specs/>, [Accessed 12-01-2025]
- Emeras J, Varrette S, Bouvry P (2016) Amazon elastic compute cloud (EC2) vs. in-house HPC platform: A cost analysis. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD), pp 284–293, doi:10.1109/CLOUD.2016.0046
- Falgout RD, Yang UM (2002) hypre: A library of high performance preconditioners. In: Sloot PMA, Hoekstra AG, Tan CJK, Dongarra JJ (eds) *Computational Science — ICCS 2002*, no. 2331 in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp 632–641, doi:10.1007/3-540-47789-6_66
- Fujitsu (2024) Supercomputer Fugaku retains first place worldwide in HPCG and Graph500 rankings — fujitsu.com. <https://www.fujitsu.com/global/about/resources/news/press-releases/2024/1119-01.html>, [Accessed 10-01-2025]
- Godbolt M (2024a) Compiler Explorer - (ARM64 gcc 13.2.0). <https://godbolt.org/z/sxGo17Wq9>, [Accessed 11-09-2024]

- Godbolt M (2024b) Compiler Explorer - high-order (armv8-a clang 18.1.0). URL <https://godbolt.org/z/WzYEefEGK>, [Accessed 11-09-2024]
- Godbolt M (2024c) Compiler Explorer - high-order (x86-64 clang 18.1.0). <https://godbolt.org/z/fEz64zzWx>, [Accessed 11-09-2024]
- Godbolt M (2024d) Compiler Explorer - high-order (x86-64 gcc 13.2). <https://godbolt.org/z/aYeYcb6z1>, [Accessed 11-09-2024]
- Godbolt M (2024e) Compiler Explorer - low-order (armv8-a clang 18.1.0). <https://godbolt.org/z/4Mdbvndrf>, [Accessed 11-09-2024]
- Google (2025) Arm VMs on Compute | Compute Engine Documentation | Google Cloud — cloud.google.com. <https://cloud.google.com/compute/docs/instances/arm-on-compute>, [Accessed 12-01-2025]
- Habera M, Hale JS (2025) Supplementary material: The FEniCS Project on AWS Graviton3. doi:10.5281/zenodo.13748404
- Karypis G, Kumar V (1998) A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* 20(1):359–392, doi:10.1137/s1064827595287997
- Kirby RC, Logg A (2006) A compiler for variational forms. *ACM Trans Math Softw* 32(3):417–444, doi:10.1145/1163641.1163644
- McCalpin JD (1991–2007) STREAM: Sustainable memory bandwidth in high performance computers. Tech. rep., University of Virginia, Charlottesville, Virginia, URL <http://www.cs.virginia.edu/stream/>
- McCalpin JD (1995) Memory bandwidth and machine balance in current high performance computers. IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter pp 19–25
- McCalpin JD (2023) The evolution of single-core bandwidth in multicore processors — sites.utexas.edu. <https://sites.utexas.edu/jdm4372/2023/04/25/the-evolution-of-single-core-bandwidth-in-multicore-processors/>, [Accessed 10-01-2025]
- Microsoft (2024) Announcing the preview of new Azure VMs based on the Azure Cobalt 100 processor | Microsoft Community Hub — techcommunity.microsoft.com. <https://techcommunity.microsoft.com/blog/azurecompute/announcing-the-preview-of-new-azure-vm-based-on-the-azure-cobalt-100-processor/4146353>, [Accessed 12-01-2025]
- Rajovic N, Rico A, Mantovani F, Ruiz D, Vilarrubi JO, Gomez C, Backes L, Nieto D, Servat H, Martorell X, Labarta J, Ayguade E, Adeniyi-Jones C, Derradji S, Gloaguen H, Lanucara P, Sanna N, Mehaut JF, Pouget K, Videau B, Boyer E, Allal M, Auweter A, Brayford D, Tafani D, Weinberg V, Brommel D, Halver R, Meinke JH, Beivide R, Benito M, Vallejo E, Valero M, Ramirez A (2016) The Mont-Blanc prototype: An alternative approach for HPC systems. In: SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, pp 444–455, doi:10.1109/sc.2016.37
- Rocke FJ (2023) Evaluation of C++ SIMD libraries. Bachelor’s thesis, Der Ludwig-Maximilians-Universität München, URL <https://www.mnm-team.org/pub/Fopras/rock23/>
- Sandia NL (2018) Astra supercomputer at Sandia Labs is fastest ARM-based machine on TOP500 list — sandia.gov. <https://www.sandia.gov/labnews/2018/11/21/astra-2/>, [Accessed 10-01-2025]
- Simakov NA, Deleon RL, White JP, Jones MD, Furlani TR, Siegmann E, Harrison RJ (2023) Are we ready for broader adoption of ARM in the HPC community: Performance and Energy Efficiency Analysis of Benchmarks and Applications Executed on High-End ARM Systems. In: Proceedings of the HPC Asia 2023 Workshops, Association for Computing Machinery, New York, NY, USA, HPCAsia ’23 Workshops, pp 78–86, doi:10.1145/3581576.3581618
- Suárez D, Almeida F, Blanco V (2024) Comprehensive analysis of energy efficiency and performance of ARM and RISC-V SoCs. *The Journal of Supercomputing* 80(9):12771–12789, doi:10.1007/s11227-024-05946-9
- Varrette S, Cartiaux H, Peter S, Kieffer E, Valette T, Ollou A (2022) Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0. In: Proc. of the 6th ACM

High Performance Computing and Cluster Technologies Conf. (HPCCT 2022), Association for Computing Machinery (ACM), Fuzhou, China, doi:10.1145/3560442.3560445
Wells G, Richardson C (2023) Performance test codes for FEniCSx. URL <https://github.com/FEniCS/performance-test>

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

