Assessing the performance of the FEniCS Project on Graviton3

Michal Habera[1], Jack S. Hale[2]

[1]Rafinex SARL, [2]University of Luxembourg

Overview and goals

- The FEniCS Project is an open-source finite element solver that uses automatic code generation techniques to rapidly build problem-specific solvers.
- Three questions:
  - Can the latest compilers auto-vectorise the automatically generated local finite element kernels?
  - Does FEniCS scale on an Graviton3 MPI cluster?
  - How does the performance compare with uni.lu Aion cluster?

Low order Finite Element Method

Example: Poisson (Laplace) problem. Find $u \in H^1$ s.t.

$$\int \nabla u \cdot \nabla v \, \mathrm{d}x = \int f \cdot v \, \mathrm{d}x$$

holds for all $v \in H^1$ and a known $f \in L^2$ (+ bcs.). Discretly, solve $Ax = b$ where

$$A_{ij} = \int \nabla \varphi_i \cdot \nabla \varphi_j \, \mathrm{d}x, \quad b_i = \int f \cdot \varphi_i \, \mathrm{d}x.$$

1. Assemble sparse matrix $A \in \mathbb{R}^{n \times n}$ and vector $b \in \mathbb{R}^n$.

```
for cell in cells:
    local_data = global_data[cell]        # Copy
    A_local = assemble_local(local_data) # Run kernel
    A_global[cell] = A_local              # Copy
```

Low order Finite Element Method

2. Solve linear system $Ax = b$.

   Example: Preconditioned Conjugate Gradient method.

```
while ||r1|| > tolerance:
    b0 = MatMult(A, p0)                           # BLAS 2
    alpha = VecDot(r0, r0) / VecDot(p0, b0)       # BLAS 1
    x1 = x0 + alpha*p0                            # BLAS 1
    r1 = r0 - alpha*b                             # BLAS 1
    z1 = MatSolve(M, r1)
    beta = VecDot(r1, z1) / VecDot(r0, z0)        # BLAS 1
    p1 = z1 + beta*p0                             # BLAS 1
```

BLAS routines

BLAS Level	Loads and stores	flop	flop/mem.
1	$3n$	$2n$	$2/3$
2	n^2	$2n^2$	2
3	$4n^2$	$2n^3$	$n/2$

- BLAS Level 1 (and 2) are typically memory bandwidth bound.

Aion vs AWS c7g

	Aion	AWS c7g
Processor	2 x (AMD Epyc ROME 7H12, 64 cores @ 2.6 GHz)	Graviton3, 64 cores @ 2.6 GHz
Memory	256 GB DDR4 3200 MT/s = 25.6 GB/s	128 GB DDR5 4800 MT/s = 38.4 GB/s
Arch.	x86_64, Zen 2 (AVX2)	ARMv8.5, Neoverse V1 (SVE)
Total bandwidth	2x200 GB/s	1x300 GB/s

AMD Epyc Aion socket

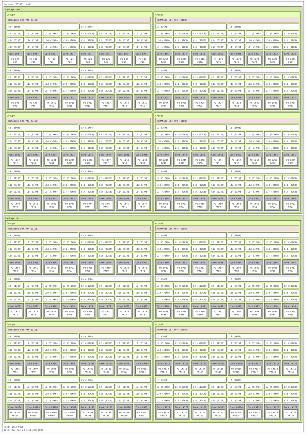

Figure 1: Aion topology

AMD Epyc Aion socket

Figure 2: Aion topology, zoom-in

AWS c7g Graviton3 socket

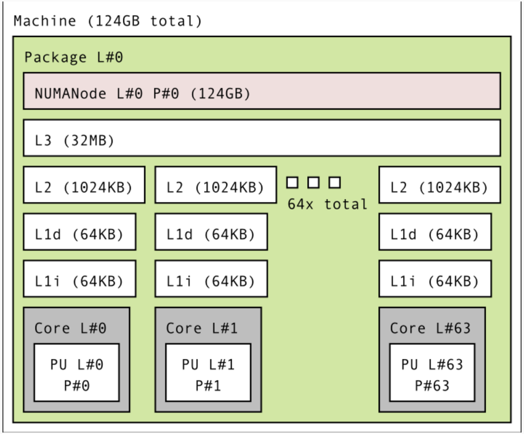

Figure 3: Graviton3 topology

STREAM benchmark

- Industry standard benchmark for measuring sustained memory bandwidth.

Name	Kernel	Bytes	FLOPS
COPY	`a[i] = b[i]`	8	0
SCALE	`a[i] = q*b[i]`	8	1
ADD	`a[i] = b[i] + c[i]`	16	1
TRIAD	`a[i] = b[i] + q*c[i]`	16	2

STREAM results

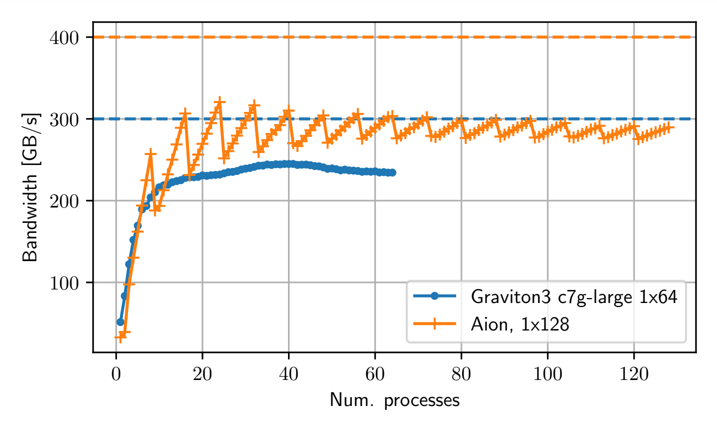

Figure 4: Single-node STREAM benchmark.

STREAM results

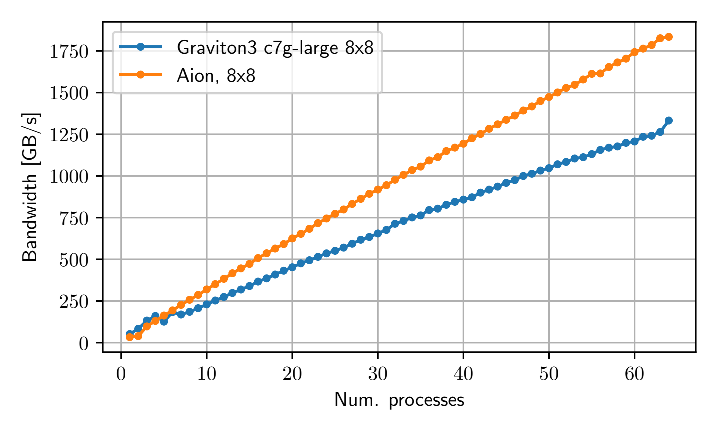

Figure 5: Multi-node STREAM benchmark.

What is the FEniCS Project?

FEniCS PROJECT

- A computing platform for translating scientific models into finite element simulations.
- 19 year history.
- Open Source (LGPLv3 and MIT).

FEniCS Applications

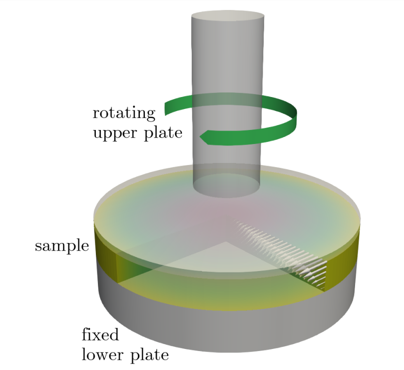

Figure 6: M. Řehoř et al. ‘A comparison of constitutive models for describing the flow of uncured styrene-butadiene rubber’, Journal of Non-Newtonian Fluid Mechanics, vol. 286, 104398, Dec. 2020. doi: 10.1016/j.jnnfm.2020.104398

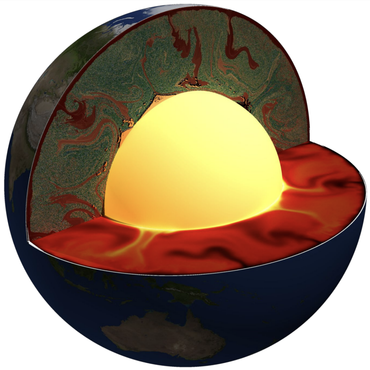

Figure 7: T. D. Jones, N. Sime, and P. E. van Keken, ‘Burying Earth’s Primitive Mantle in the Slab Graveyard’, Geochemistry, Geophysics, Geosystems, vol. 22, no. 3. American Geophysical Union (AGU), Mar. 2021. doi: 10.1029/2020gc009396.

FEniCS Applications

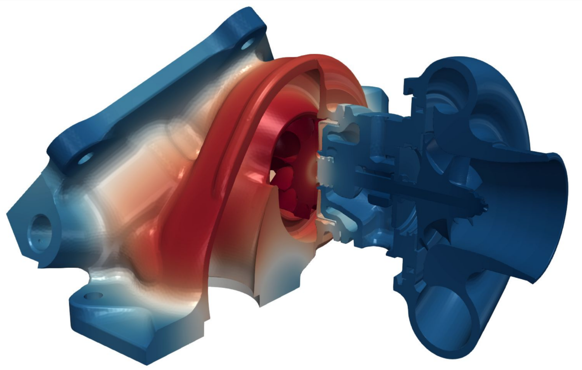

Figure 8: C. N. Richardson, N. Sime, and G. N. Wells, 'Scalable computation of thermomechanical turbomachinery problems', Finite Elements in Analysis and Design, vol. 155. Elsevier BV, pp. 32–42, Mar. 2019. doi: 10.1016/j.finel.2018.11.002.

Local kernel benchmarks

- FEniCS Form Compiler FFCx generates C code.
- Benchmark executes the generated finite element kernel repeatedly on a single mesh cell.
- Source https://github.com/michalhabera/local_operator (fork of Igor Baratta's repository).
- Kernels:
  - Elasticity, 2nd order (memory bandwidth bound).
  - Elasticity, 6th order (compute bound).
- Can the latest compilers produce auto-vectorised code for Graviton3? How is the performance?

Example generated finite element kernel

```
void kernel(...) {
  // 1. Large array definitions
  // 2. Quadrature loop independent computations
  // 3. Quadrature loop body
  for (int iq = 0; iq < NUM_QUAD_POINTS; ++iq) {
    for (int ic = 0; ic < NUM_DOFS; ++ic){
      w1_d100_c0 += _w_4_0[ic] * FE18_C0_D100_Q175[0][0][iq][ic];
      // ...
    }
    // 3.1 Scalar graph evaluation
    double sv_175[129];
    sv_175[0] = w1_d100_c0 * sp_175[14];
    sv_175[1] = w1_d010_c0 * sp_175[17];
    // 3.2 Assignment loop
    for (int i = 0; i < NUM_DOFS; ++i) {
      A[3 * i] += fw0 * FE18_C0_D100_Q175[0][0][iq][i]
               + fw1 * FE18_C0_D010_Q175[0][0][iq][i];
      // ...
    } } }
```

Compilers

- Graviton3:
  - gcc 12.2.0:
    - -Ofast -mcpu=neoverse-v1
    - -Ofast -mcpu=neoverse-v1 -fno-tree-vectorize
    - -O2 -fno-tree-vectorize
  - clang 15.0.7
    - -Ofast -mcpu=neoverse-v1
    - -Ofast -mcpu=neoverse-v1 -fno-slp-vectorize -fno-vectorize
    - -O2 -fno-slp-vectorize -fno-vectorize
- Aion:
  - gcc 12.2.0:
    - -Ofast -march=znver2 -mtune=znver2
    - -Ofast -march=znver2 -mtune=znver2 -fno-tree-vectorize
    - -O2 -fno-tree-vectorize
  - clang 15.0.7
    - -Ofast -march=znver2 -mtune=znver2
    - -Ofast -march=znver2 -mtune=znver2 -fno-slp-vectorize -fno-vectorize
    - -O2 -fno-slp-vectorize -fno-vectorize

Memory bandwidth bound benchmarks

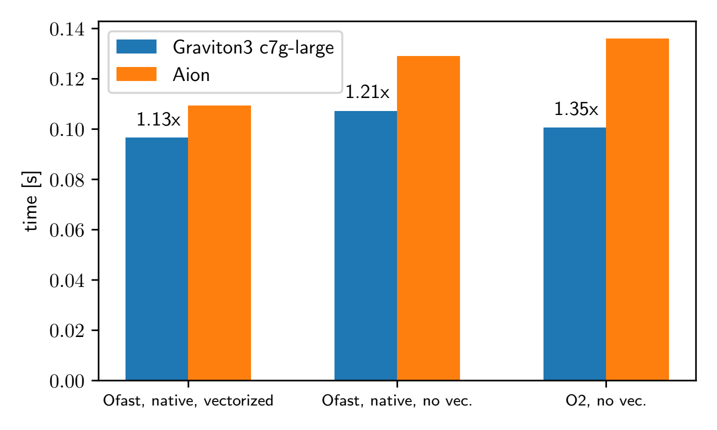

Figure 9: Elasticity, 2nd order, gcc.

Memory bandwidth bound benchmarks

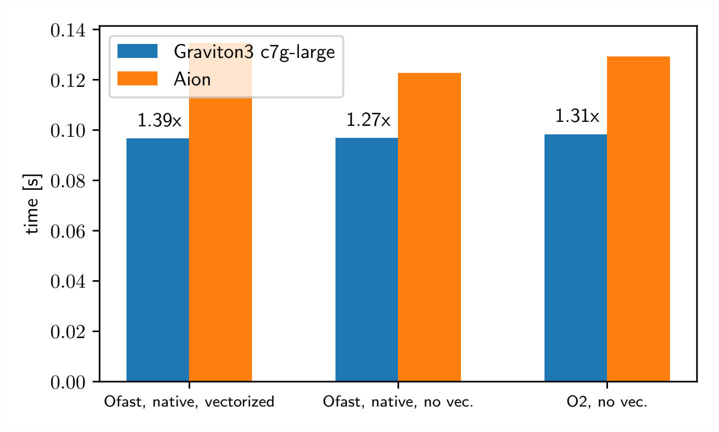

Figure 10: Elasticity, 2nd order, clang.

Compute bound benchmarks

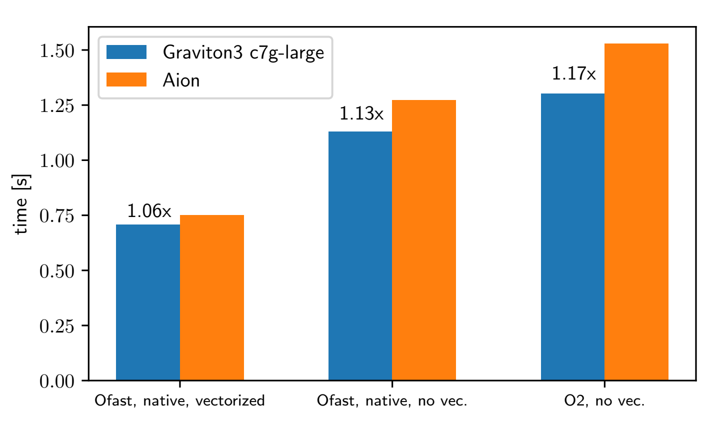

Figure 11: Elasticity, 6th order, gcc.

Compute bound benchmarks

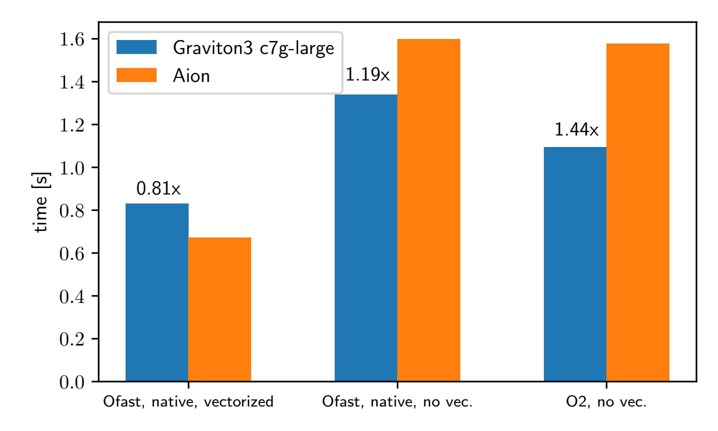

Figure 12: Elasticity, 6th order, clang.

Performance benchmarks

- Executes a full solve of a Poisson problem.
  - Execution of finite element kernels across mesh.
  - Assembly of sparse matrix and vectors.
  - Solution of linear system (PETSc, AMG + Conjugate Gradients)
- Goal: Weak scaling, ~500k degrees of freedom per process. Best case 'constant time'.
- Parameters:
  - Nodes: 1 (on node), 2, 4, 8, 16 (intra-node).
  - 50% socket utilisation, i.e. 32 tasks per node (circumvent memory bandwidth bounds).

Performance benchmarks

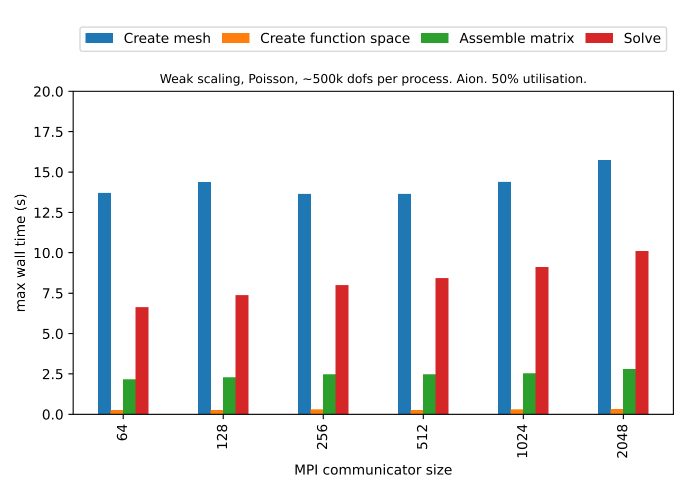

Figure 13: Aion weak scaling.

Performance benchmarks

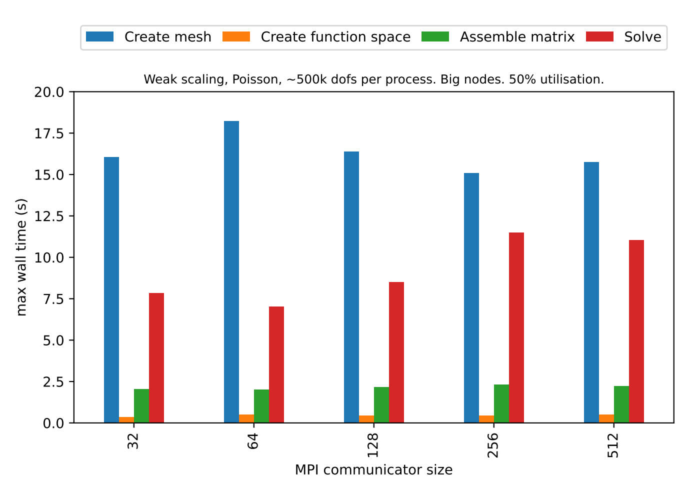

Figure 14: Graviton3 c7g-large weak scaling.

Practical experience with AWS c7g

- Spack worked smoothly for building FEniCS.
- Default GCC still a bit old for practical use.
- Terminal keyboard input lags.
- Resource configuration/allocation takes time (minutes vs. seconds on Aion, if available).

Summary

- Both Aion and AWS c7g reach expected $\approx$ 80% of memory bandwidth per node.
- Memory bandwidth bound kernels benefit from faster memory on AWS c7g.
- Compute bound kernels have some loops (scalable) auto-vectorized.
- GCC 12.2.0 shows more consistent/explainable behaviour comparing clang 15.
- Weak scaling similar on Aion and Graviton3 c7g.

Acknowledgement

The present project is supported by the National Research Fund, Luxembourg, under Industrial Fellowship project "COAT", Ref. 17205623.

Appendix

- Aion has 256 MB L3 cache per socket, i.e. 512 MB per node. On 8 nodes STREAM requires $8 \times 512 \times 4\text{MB} \approx 16\text{GB}$ to avoid measuring cache speeds.
- SLURM requires `--exclusive` option to make sure nodes are memory-idle.
- Vectorization reports:
  - Graviton3, clang: https://godbolt.org/z/j3jY5hEve
  - Graviton3, gcc: https://godbolt.org/z/Pcnz6776n
  - Aion, clang: https://godbolt.org/z/xe81h66eY
  - Aion, gcc: https://godbolt.org/z/ve6fc5x9d