Proceedings of
FEniCS 2021
22–26 March 2021

Editors

Igor Baratta

Jørgen S. Dokken

Chris Richardson

Matthew W. Scroggs

Finite elements on accelerators: an experience using FEniCSx and SYCL

Igor Baratta (IgorBaratta), University of Cambridge, United Kingdom
Chris Richardson (chrisrichardson), University of Cambridge, United Kingdom
Garth Wells (garth-wells), University of Cambridge, United Kingdom

24 March 2021

In this presentation, we are going to talk about our experience implementing the finite element method on different architectures and accelerators using the FEniCSx libraries and the SYCL programming model. Our main focus is on performance portability, we would like the FEM program to get consistent performance on a wide variety of platforms, instead of being very efficient on a single one.

SYCL is a modern kernel-based parallel programming model that allows for one code to be written which can run in multiple types of computational devices (eg CPUs and GPUs). A kernel describes a single operation, that can be instantiated many times and applied to different input data (eg cell-wise matrix assembly). This kernel-based model matches nicely with the new FEniCS data-centric design: DOLFINx generates data to operate in parallel (geometry, topology, and dofmaps) and FFCx generates efficient code that can be used as part of the parallel kernels.

We will discuss how different ways of expressing parallelism can affect the performance we ultimately achieve, for instance, we consider different global assembly strategies and data structures. We will also discuss how carefully arranging memory transfer and allocations can reduce latency and increase throughput in different accelerators. Finally, we will show some performance results of simplified finite element simulations on different architectures, ranging from Intel and AMD CPUs to NVIDIA GPUs.

You can cite this talk as:

Igor Baratta, Chris Richardson, and Garth Wells. "Finite elements on accelerators: an experience using FEniCSx and SYCL". In: *Proceedings of FEniCS 2021, online, 22–26 March* (eds Igor Baratta, Jørgen S. Dokken, Chris Richardson, Matthew W. Scroggs) (2021), 413–429. DOI: 10.6084/m9.figshare.14495385.

BibTeX for this citation can be found at https://mscroggs.github.io/fenics2021/talks/baratta.html.

UNIVERSITY OF CAMBRIDGE

Finite elements on accelerators

An experience using **FEniCSx** and **SYCL**

Igor Baratta
Chris Richardson
Garth Wells

Table of Contents

What's Performance Portability?

And why do we care about it?

An application is performance portable if it:

- ✓ Achieves reasonable level of performance
- ✓ Requires minimal platform specific code

Programming Model

SYCL is a high-level single source parallel programming model, that can target a range of heterogeneous platforms:

- uses completely standard C++;
- both host CPU and device code can be written in the same C++ source file;
- open standard coordinated by the **Khronos** group.

SYCL implementations:

Intel SYCL*

hipSYCL*

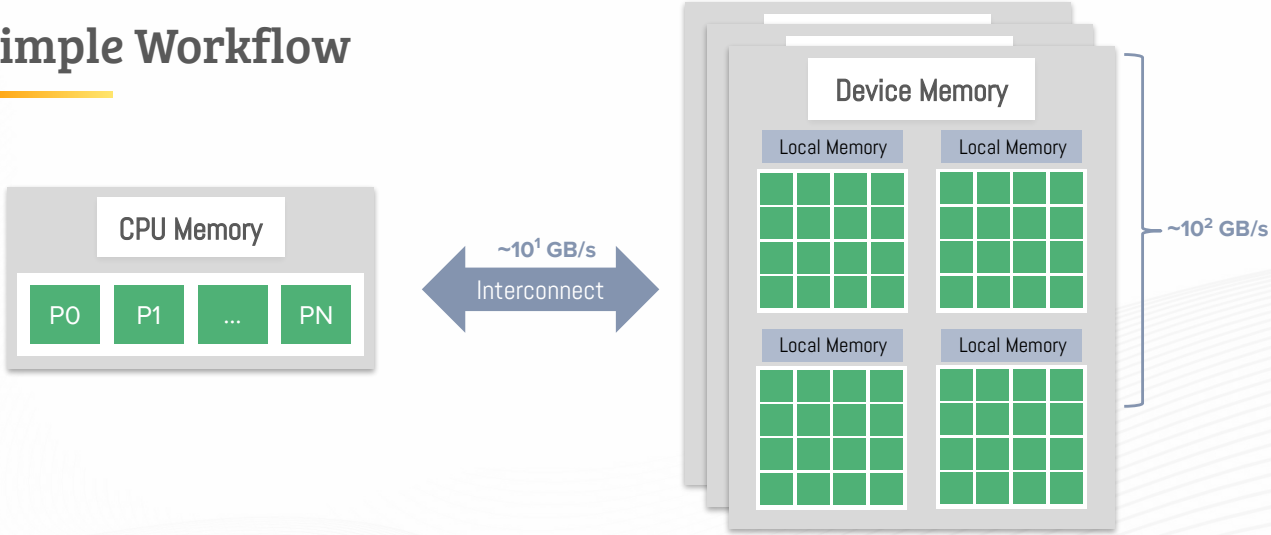
Compute Cpp

triSYCL*

**open source*

```
cl::sycl::queue q{cl::sycl::gpu_selector()};

int N = 100;
auto a = cl::sycl::malloc_device<double>(N, q);
auto b = cl::sycl::malloc_shared<double>(N, q);

auto e = q.fill(a, 3.0, N);

q.parallel_for(cl::sycl::range<1>(N), e,
[=](cl::sycl::id<1> Id) {
  int i = Id.get(0);
  b[i] = 2 * a[i];
});

q.wait();

for (int i = 0; i < N; i++)
  assert(b[i] == 6.);
```

Simple Workflow



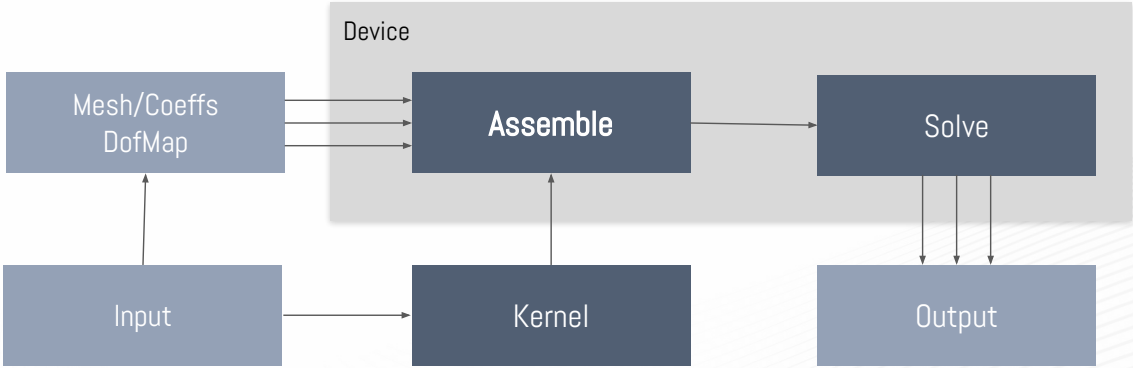
Copy input data from **Host** memory to **Device** memory

Launch kernels for execution on the **Device**

Wait for the execution queue to finish

Copy results back to **Host** from **Device**

An idealised modular Finite Element worflow

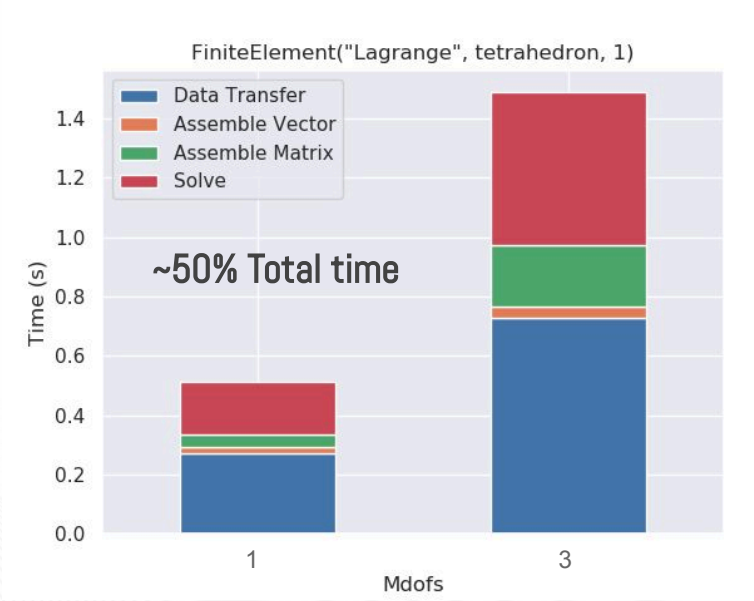


UFL File

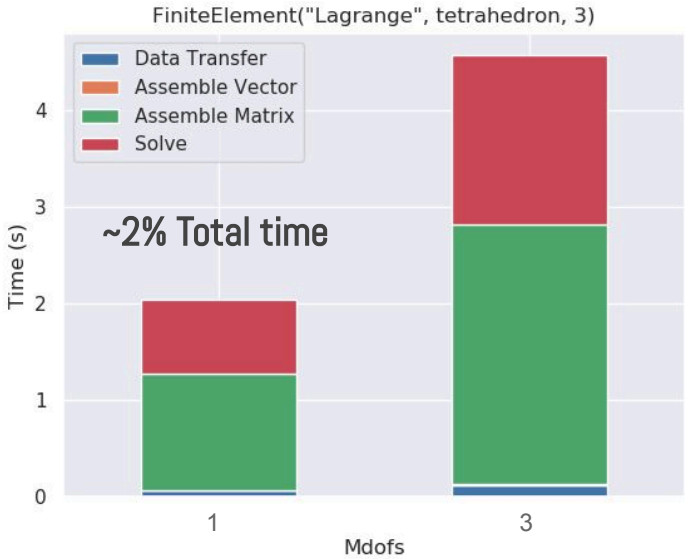
```
element = FiniteElement("Lagrange", tetrahedron, 3)
...
a = inner(grad(u), grad(v)) * dx + k*inner(u, v) * dx
L = inner(f, v) * dx
```

ffcx **--sycl_defines=True** problem.ufl

Data Transfer to Computation Ratio - P1



Data Transfer to Computation Ratio - P3



Matrix Assembly

For each cell:

01 Gather cell coordinates and coefficients

02 Compute element matrix

03 Update global CSR matrix

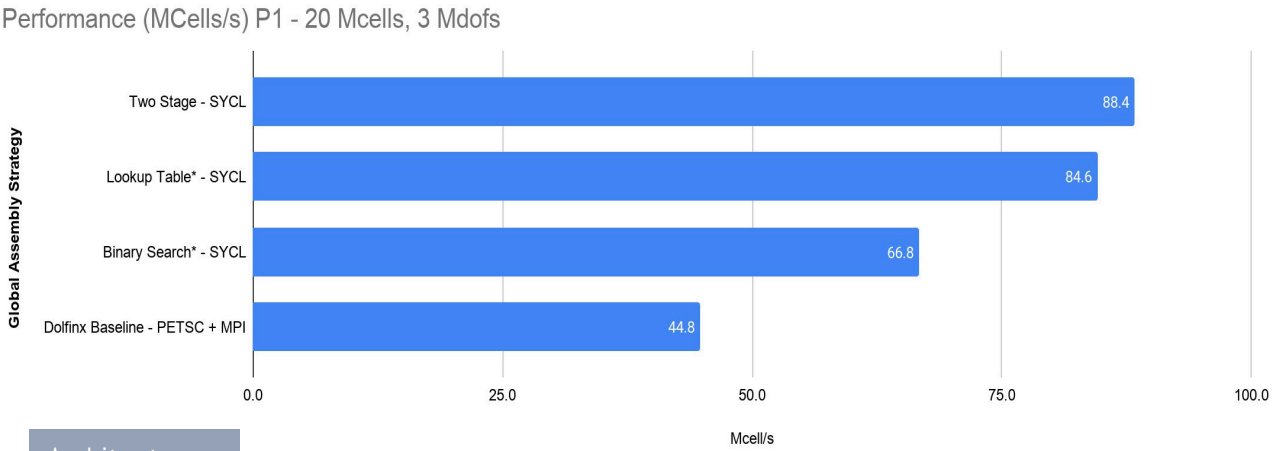
Global assembly strategies:

- Binary Search*
- Lookup Table*
- Two Stage

** atomic operations*

```
auto kernel = [=](cl::sycl::id<1> ID) {
   const int i = ID.get(0);
   ...
   double Ae [ndofs * nofs];

   // Gather cell coordinates and coefficients
   for (std::size_t j = 0; j < 4; ++j)
   {
     const std::size_t dmi = x_coor[i * 4 + j];
     for (int k = 0; k < gdim; ++k)
      cell_geom[j * gdim + k] = x[dmi * gdim + k];
   }
   ...
   // Compute element matrix
   tabulate_cell_a(Ae, coeffs, cell_geom);

   // Update global matrix - Binary Search
   for (int j = 0; j < ndofs; j++)
     for (int k = 0; k < ndofs; k++)
     {
       int ind = dofs[offset + k];
       int pos = find(indices, first, last, ind);
       atomic_ref atomic_A(data[pos]);
       atomic_A += Ae[j * ndofs + k];
     }
};
```

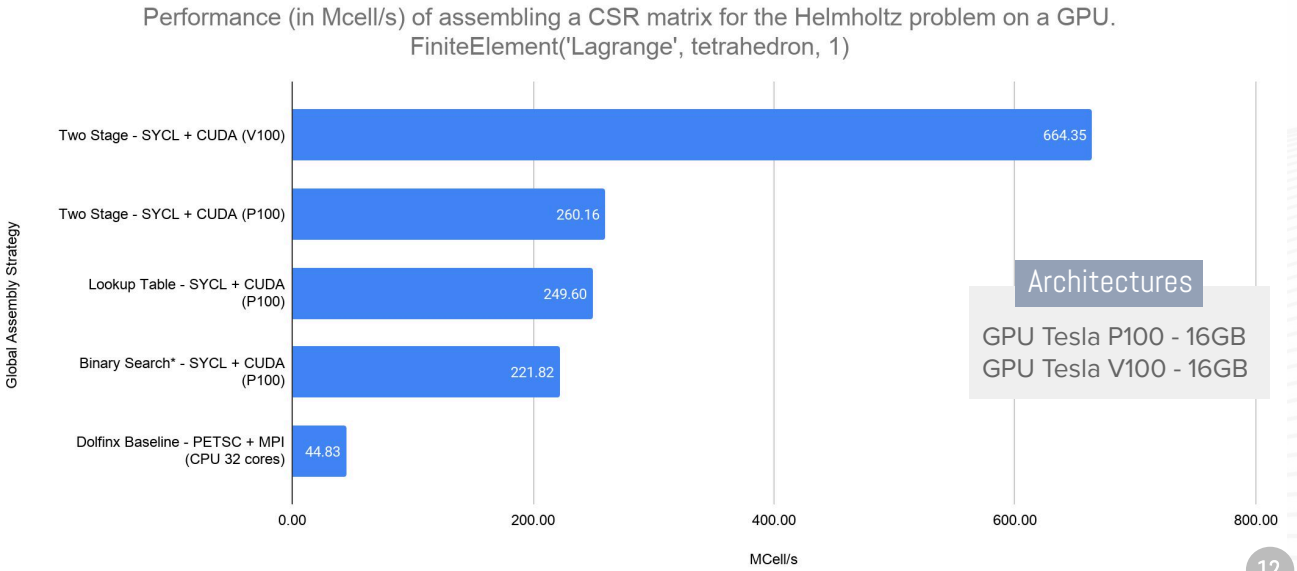
Matrix Assembly - CPU Performance



Architectures

2 x Intel Xeon Skylake 6142 processors, 2.6GHz 16-core
Theoretical peak performance: 2.7 TFlop/s.
192GB RAM

Matrix Assembly - GPU Performance



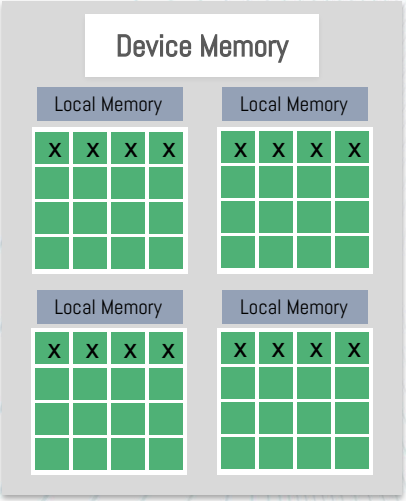
Matrix Assembly - GPU Performance

Performance (in Mcell/s) of assembling a CSR matrix for the Helmholtz problem on a GPU.
FiniteElement('Lagrange', tetrahedron, 1)

Architectures

GPU Tesla P100 - 16GB
GPU Tesla V100 - 16GB

Low Achieved Occupancy



Achieved Occupancy: ~25%
The occupancy limited by register usage.

Solution:
Use shared memory for precomputed tables.

Each thread block (work-group) has shared memory visible to all threads (work-item) of the block.

	Occupancy	MCell/s
1st Version	25%	664 MCell/s
Shared Memory	63%	1660 MCell/s
Reference CUDA[1]	*	1627 MCells/s

[1] James Trotter - High-performance finite element computations - Performance modelling, optimisation, GPU acceleration & automated code generation - Phd Thesis 2021.

Thank you!

The code and reproducibility instructions can be found at **https://github.com/Excalibur-SLE/dolfinx.sycl**

You can reach me via e-mail:
ia397@cam.ac.uk

Future/Ongoing Work

Different problems, and meshes

Linear Elasticity, Maxwell's equations

Profiling in a wider range of devices

AMD GPU, A64FX

Multi-GPU

MPI-based distributed memory computations

Code transformation

Improve generated code