Performance-Portable Matrix-Free Finite Element Solvers with SYCL

Igor Baratta, Chris Richardson and Garth N. Wells
Department of Engineering

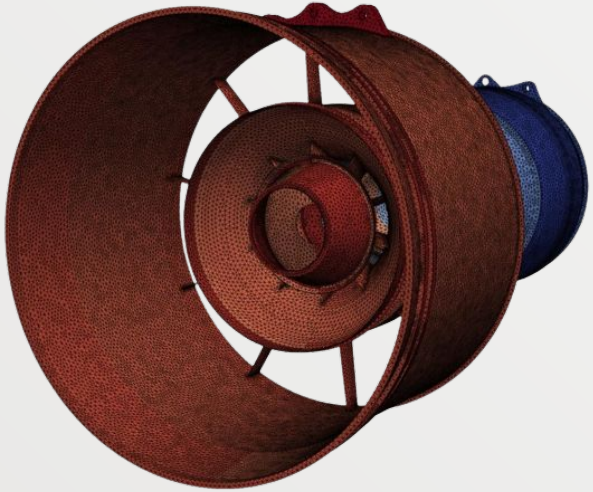
UNIVERSITY OF CAMBRIDGE

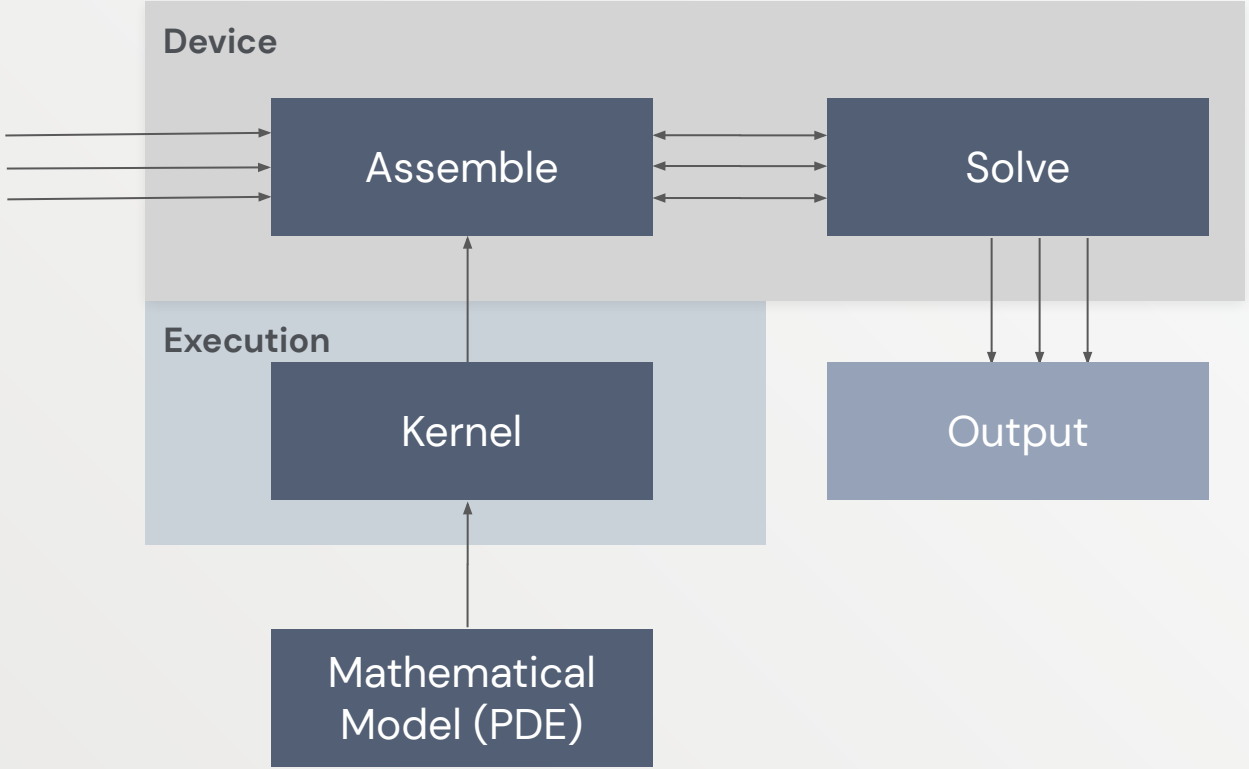
Compute variety at scale or MPI is not enough

TOP 500 The List.

Frontier	Fugaku	LUMI	Leonardo	Summit
AMD EPYC 64C 2GHZ AMD INSTINCT MI250X	A64FX 48C 2.2GHz	AMD EPYC 64C 2GHZ AMD INSTINCT MI250X	XEON PLATINUM, NVIDIA A100 SXM4	IBM POWER9 22C 3.07GHz NVIDIA Volta GV100
Rmax (PFlop/s) 1,685.65	Rmax (PFlop/s) 537.21	Rmax (TFlop/s) 428.70	Rmax (TFlop/s) 255.75	Rmax (TFlop/s) 200.79

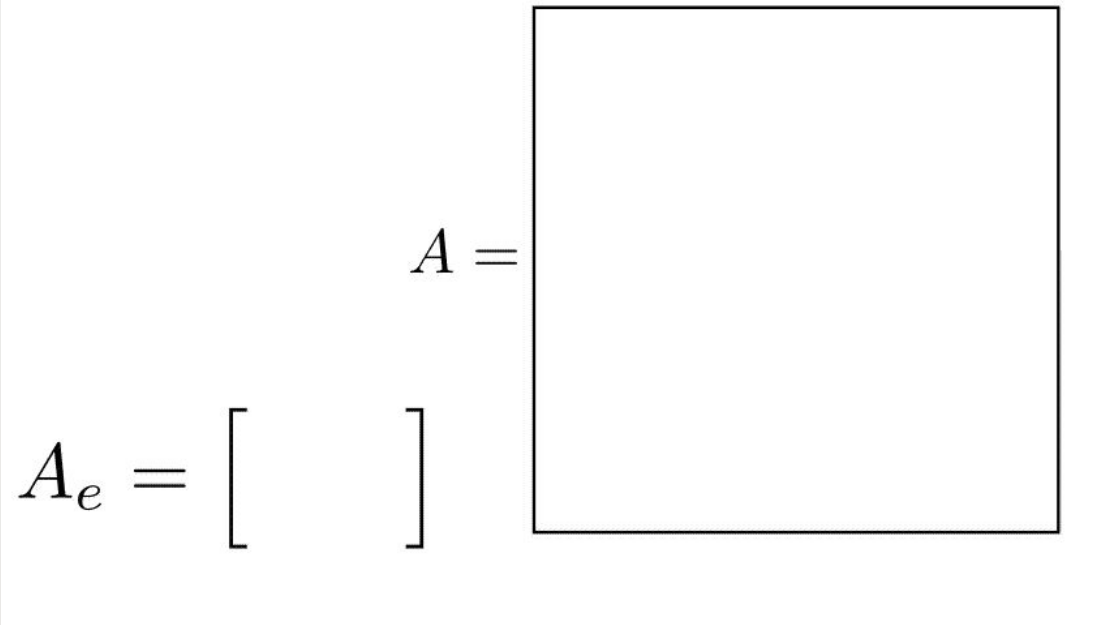
Modular finite element method





FENICS PROJECT

Finite element overview



Linear Algebra Backend

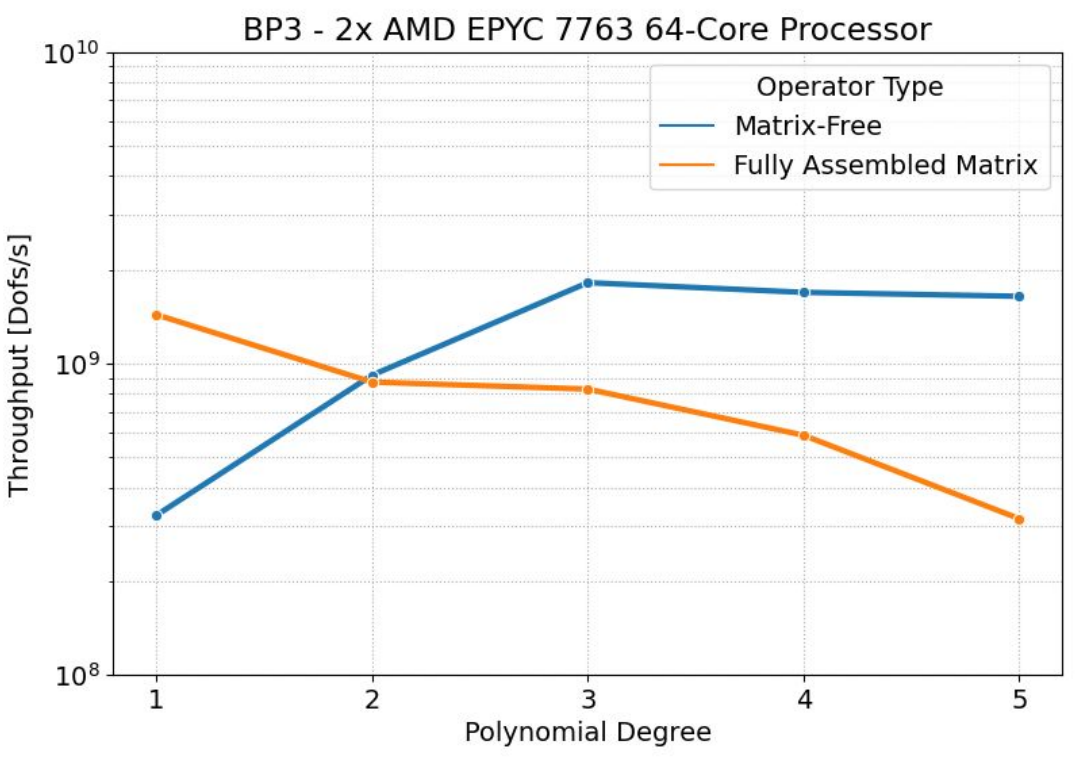
PETSc

TRILINOS

Ginkgo

AmgX

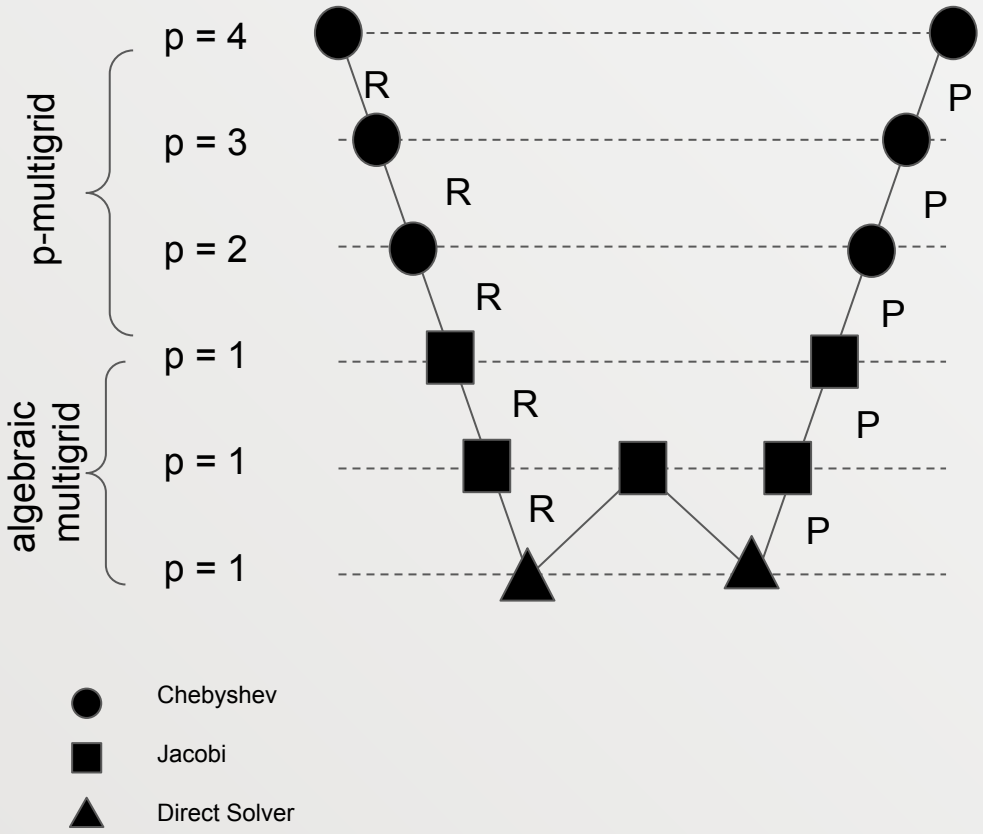
Matrix-free Vs Standard sparse matrices



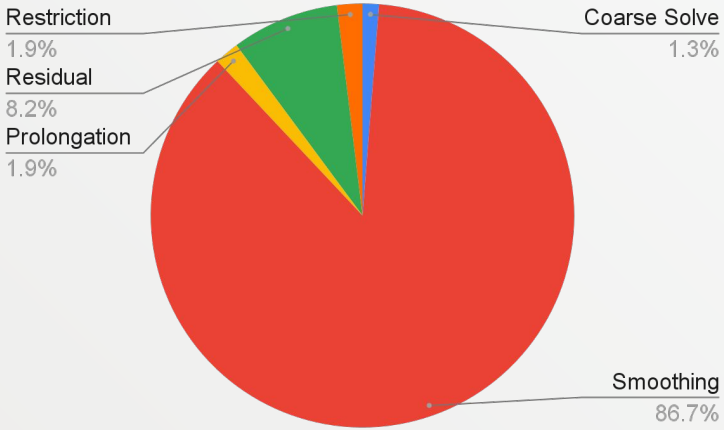
Matrix Free require only $O(n)$ data movement and storage.

Fully assembled matrix performance falls rapidly at high orders while matrix-free operators scale well.

p-Multigrid



Method	dot	axpby	matvec
CG	2	3	1
Jacobi	-	-	1
Chebyshev	-	2	1



Conjugate Gradient to solve Ax=b

$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
$i = 0$
while $i < i_{\max}$ and $\mathbf{r}_i \cdot \mathbf{r}_i > \epsilon$ **do**
  $\mathbf{y}_i = A\mathbf{p}_i$
  $\alpha_i = \dfrac{\mathbf{r}_i \cdot \mathbf{r}_i}{\mathbf{p}_i \cdot \mathbf{y}_i}$
  $\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha\mathbf{p}_i$
  $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha\mathbf{y}_i$
  $\beta_i = \dfrac{\mathbf{r}_{i+1} \cdot \mathbf{r}_{i+1}}{\mathbf{r}_i \cdot \mathbf{r}_i}$
  $\mathbf{p}_{i+1} = \beta_i\mathbf{p}_i + \mathbf{r}_i$
  $i = i + 1$
end while

- $\mathbf{y} = \alpha\mathbf{x} + \beta\mathbf{y}$
- $\alpha = \mathbf{x} \cdot \mathbf{y}$
- $\mathbf{y} = A\,\mathbf{x}$

$$\mathbf{y} = Z^T A^e Z\,\mathbf{x}$$
$$\mathbf{y} = P^T G^T A^e G\,P\,\mathbf{x}$$
$$\mathbf{y} = P^T G^T \phi^T D\,\phi\,G\,P\,\mathbf{x}$$

Performance Model (memory bound kernels)

$$t = t_0 + \frac{N_b}{W_a}$$

t is the execution time

N_b is the number of bytes streamed

t_0 is the latency

W_a is the asymptoptic memory bandwith

W is the observed memory bandwith

Inner product using SYCL 2020 built-in reduction

```
template <typename T>
T dot(sycl::queue& queue,
      std::size_t n, const T* x, const T* y,
      std::size_t wgs, std::size_t bs,
      const std::vector<sycl::event>& events = {})
{
 // Get execution range
 sycl::nd_range<1> range{n / bs, wgs};

 sycl::buffer<T> sum{1};
 auto init = sycl::property::reduction::initialize_to_identity{};
 queue.submit([&](sycl::handler & h)
 {
   h.depends on(events);
   auto reductor = sycl::reduction(sum, h, T{0.0}, std::plus<T>(),init);
   h.parallel_for(range, reductor, [=](sycl::nd_item<1> it, auto& sum) {
     std::size_t idx = it.get_global_id(0);
     std::size_t size = it.get_global_range(0);
     for (std::size_t i = idx; i < n; i += size)
       sum += x[i] * y[i];
   });
 });

 sycl::host_accessor sum_host{sum};
 return sum_host[0];
}
```

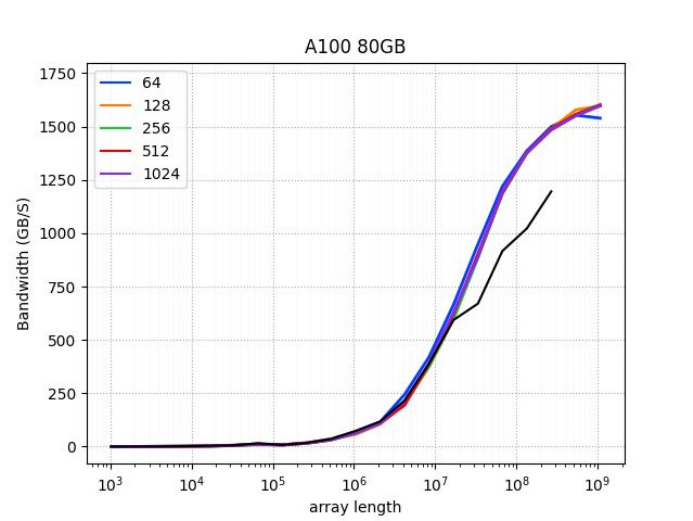
$$\alpha = \mathbf{x} \cdot \mathbf{y} = \sum_{i}^{n} x_i y_i$$

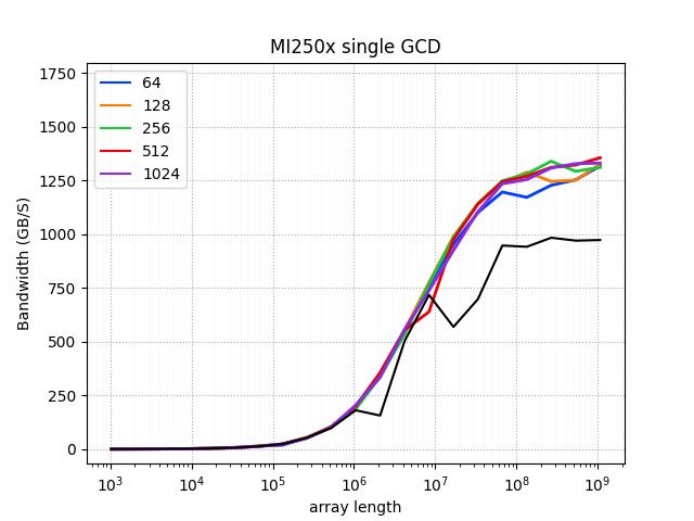
$2n$ reads

n multiplications + n additions

$$AI = \frac{2n}{16n} = \frac{1}{8}$$

Inner product using SYCL 2020 built-in reduction



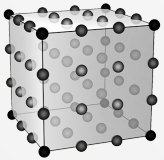


	A100		MI250X	
WG Size	T_0 (µs)	W_a (GB s^{-1})	T_0 (µs)	W_a (GB s^{-1})
64	167	1648	93	1312
128	191	1657	82	1320
256	211	1661	62	1319
512	200	1691	82	1359
1024	195	1692	77	1341.

Baratta, Igor, Chris Richardson, and Garth Wells. "Performance analysis of matrix-free conjugate gradient kernels using SYCL." International Workshop on OpenCL. 2022.

Performance Model (fem kernels)

Tensor Product Elements



$$T_{max}(dofs/s) = \min\left\{\frac{F_{max}}{f_d}, \frac{B_{max}}{b_d}, \frac{\beta(B_c)}{b_c}\right\} \Longrightarrow T_{max} = \frac{B_{max}}{b_d}$$

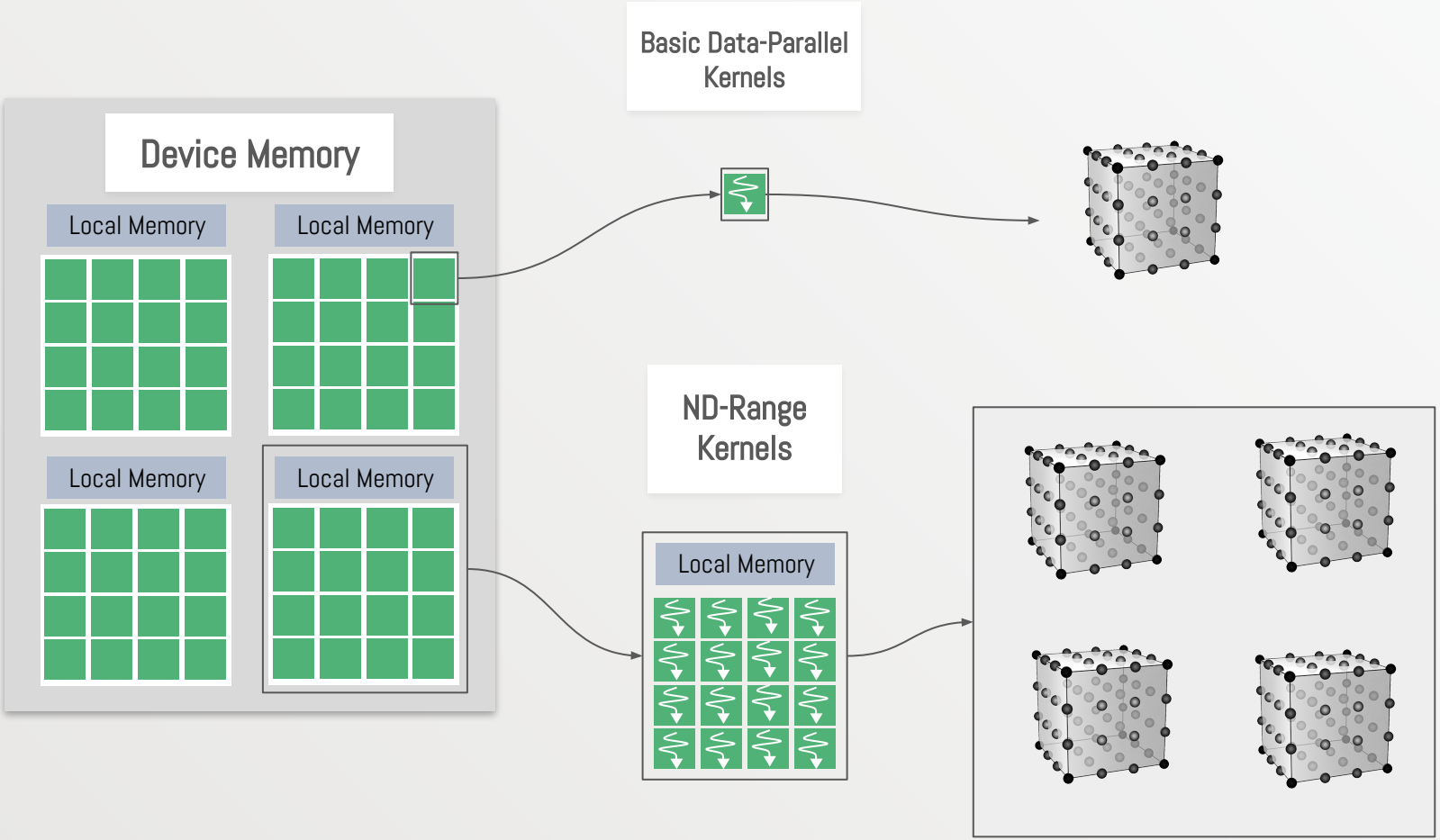
F_{max} is the peak flop/s
B_{max} is the asymptoptic (max) memory bandwith
β is the memory bandwith of a given cache level

Architecture

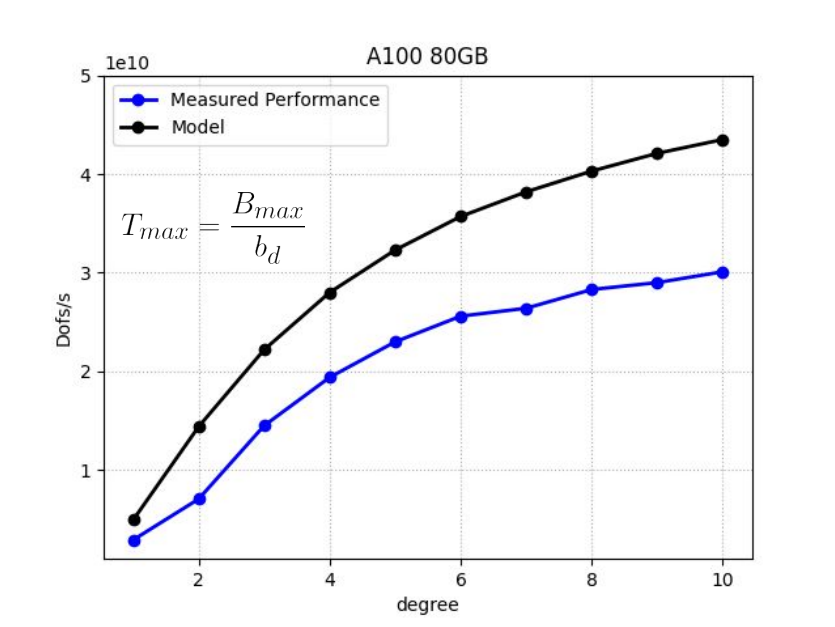
f_d flops per dof
b_d bytes moved per dof (from and to DRAM)
b_c bytes moved per dof (from and to cache)

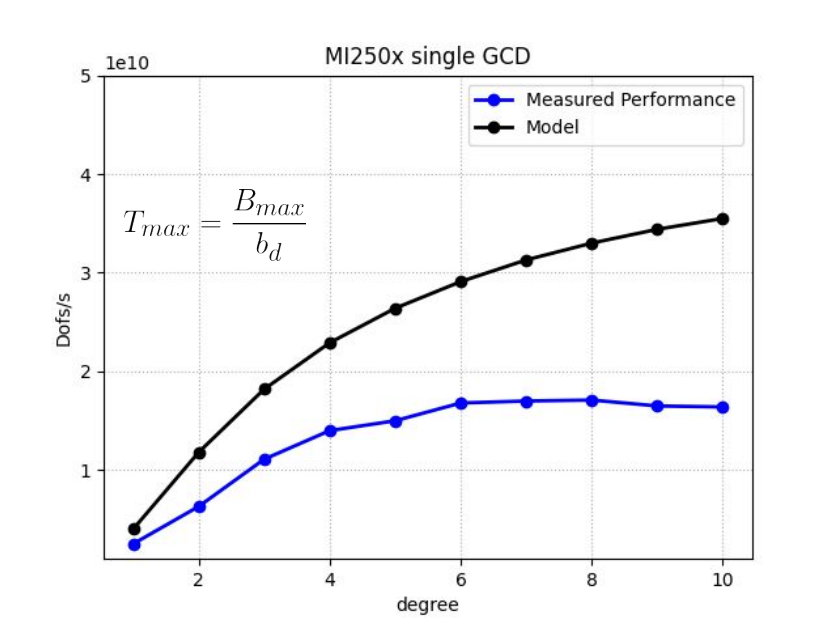
Algorithm/code

Parallelization strategies



Results Mass operator - single device



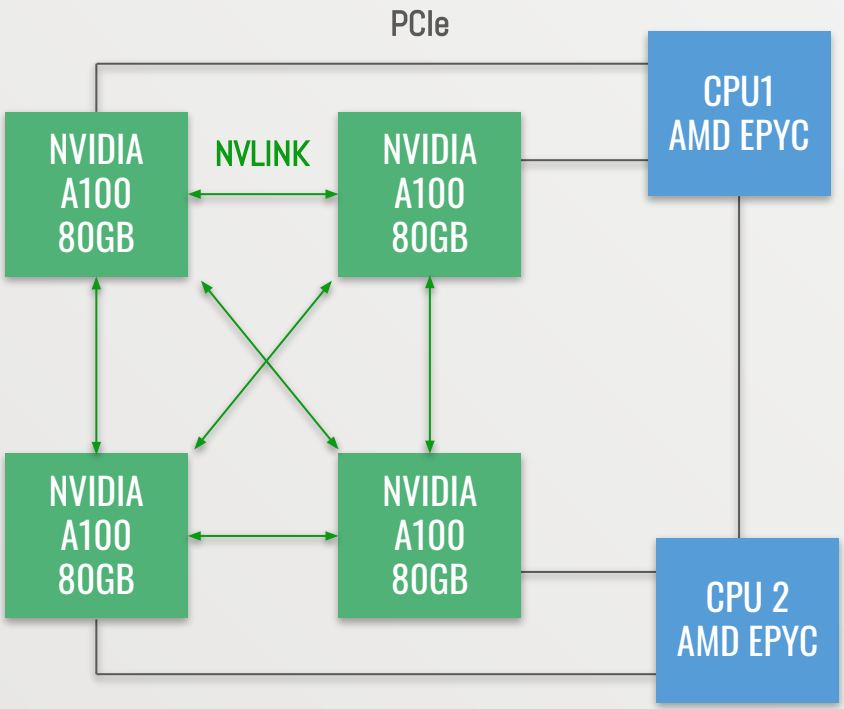


60%-70% fraction of peak performance

40%-60% fraction of peak performance

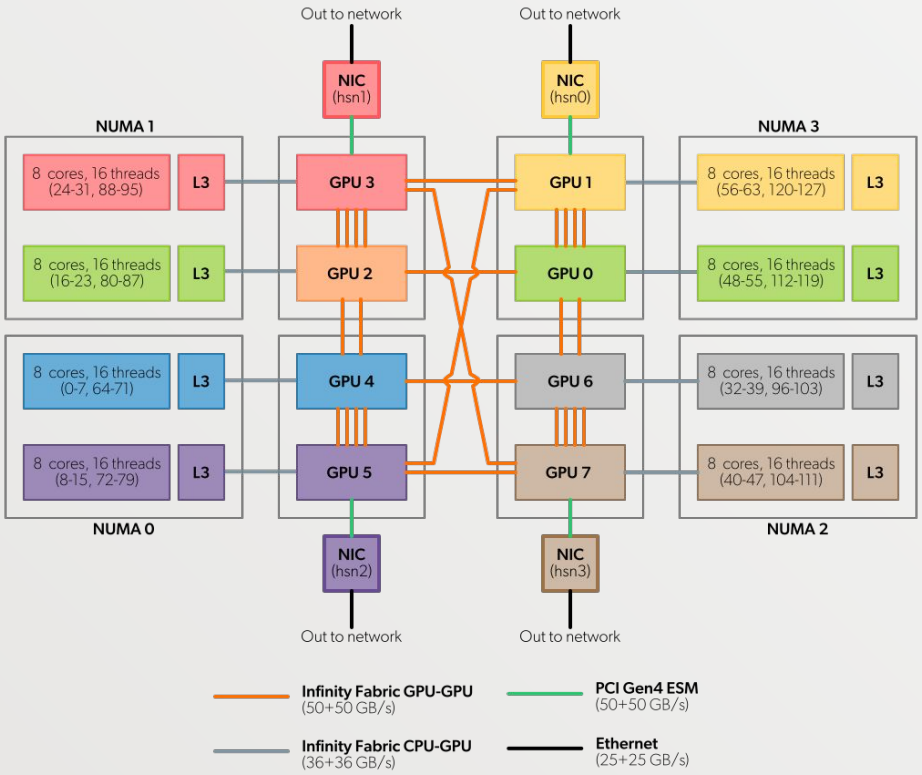
Irregular memory access pattern

A100 node topology - Device selection



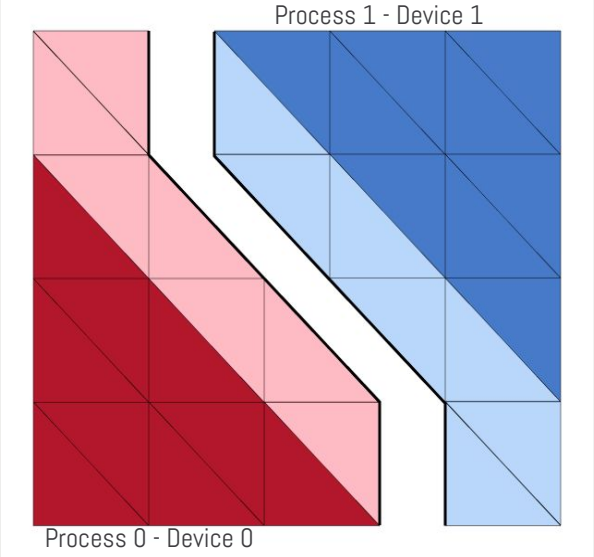

```
// Global communicator, size 4 * number of nodes
MPI_Comm comm{MPI_COMM_WORLD};

// Create node communicator, size 4.
MPI_Comm node_comm;
MPI_Comm_split_type (comm, MPI_COMM_TYPE_SHARED , 0, MPI_INFO_NULL , &node_comm);

// Get rank in local communicator
int node_size, node_rank;
MPI_Comm_size (node_comm, &node_size);
MPI_Comm_rank (node_comm, &node_rank);

// Get list of devices available per node
std::vector<sycl::device> gpus;
std::vector<sycl::platform> platforms = sycl::platform::get_platforms ();
for (auto &p : platforms) {
 auto devices = p.get_devices ();
 for (auto &device : devices)
   if (device.is_gpu())
     gpus.push_back (device);
}

sycl::queue queue{gpus[node_rank]};
```

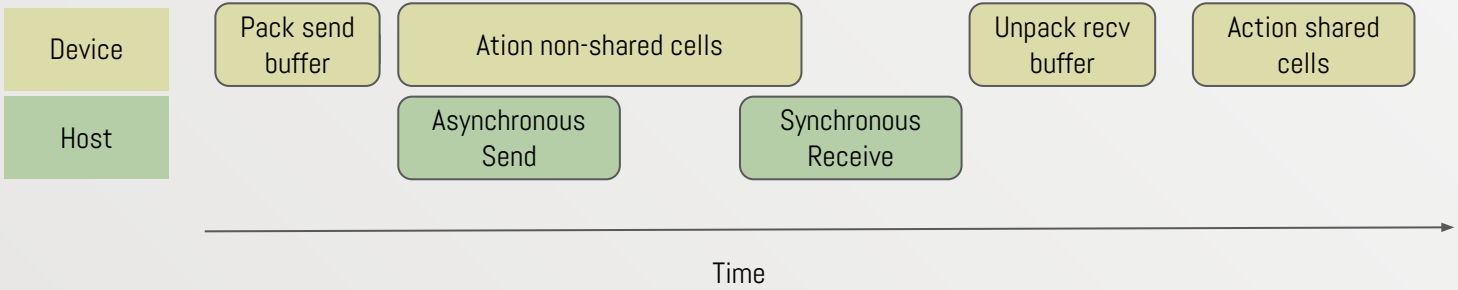
AMD Instinct™ MI250X node topology



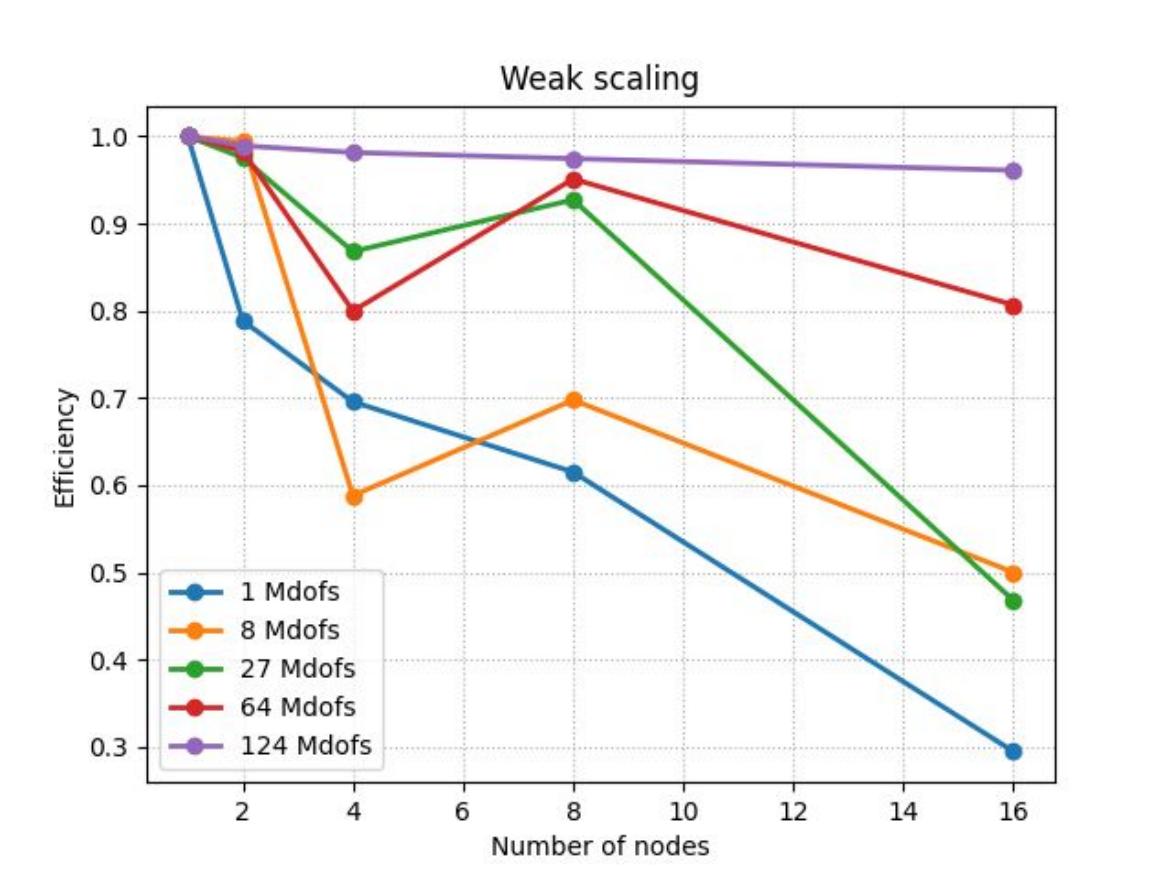

```
// Global communicator, size 8 * number of nodes
MPI_Comm comm{MPI_COMM_WORLD};

// Create node communicator, size 8.
MPI_Comm node_comm;
MPI_Comm_split_type (comm, MPI_COMM_TYPE_SHARED , 0, MPI_INFO_NULL , &node_comm);

// Get rank in local communicator
int node_size, node_rank;
MPI_Comm_size (node_comm, &node_size);
MPI_Comm_rank (node_comm, &node_rank);

// Get list of devices available per node
std::vector<sycl::device> gpus;
std::vector<sycl::platform> platforms = sycl::platform::get_platforms ();
for (auto &p : platforms) {
 auto devices = p.get_devices ();
 for (auto &device : devices)
   if (device.is_gpu())
     gpus.push_back (device);
}

sycl::queue queue{gpus[node_rank]};
```

Computation-Communication Overlap





Weak scaling full CG solver



Summary

- It's possible to write performance portable finite element kernels with SYCL
  - Up to 70% of peak performance on the A100
  - Up to 60% of peak performance on the MI250x
- Scaling tests on up to 16 nodes (8 GCD per node - MI250x)
  - > 95% efficiency on the performance saturated limit
- In the strong scale limit both intra-node performance and scaling suffer:
  - Latency is a major issue
  - Difficult to hide communication (<30% parallel efficiency)
- **Challenge**: Hide latency and communication on a Multigrid method!