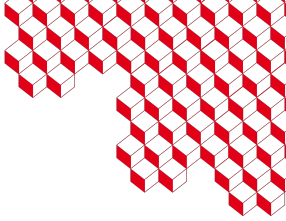


ArcaneFEM: Scalable GPGPU Implicit FEM Solver for Unstructured Meshes

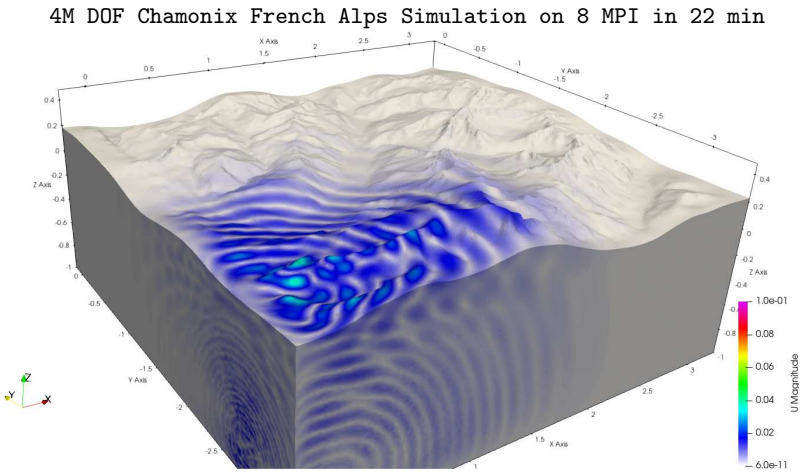
Mohd Afeef Badri[1,✉], G. Grospellier[2], C.A. Leger[1], K. Alaire[1], E. Foerster[1], A. l'Heritier[2]

1 CEA DES/ISAS/DM2S/SGLS, Saclay, France
2 CEA DAM, Bruyères-le-Châtel, France
✉ mohd-afeef.badri@cea.fr

Rencontres Arcane, March 24, 2025

ArcaneFEM: introduction

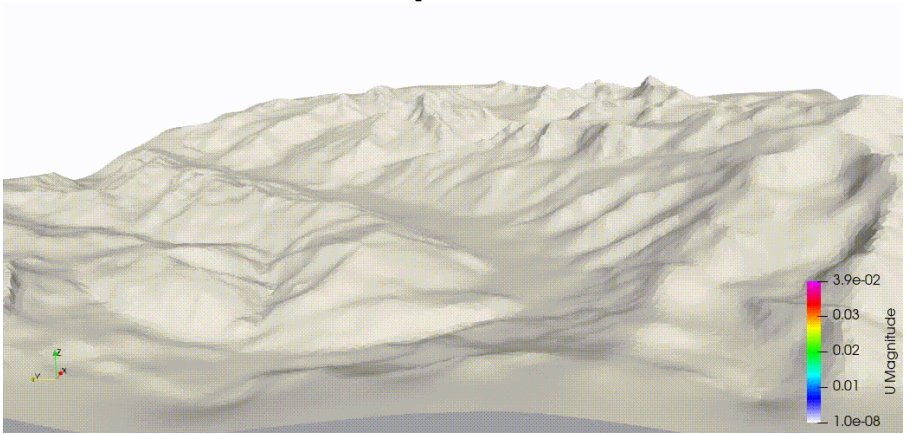
- Arcane + FEM started Nov. 2022 (**Young code**)
- Primary aim, **high-fidelity Earthquake** (source to site)
- ML Assisted Data-Assimilation: Fast solving



ArcaneFEM: introduction

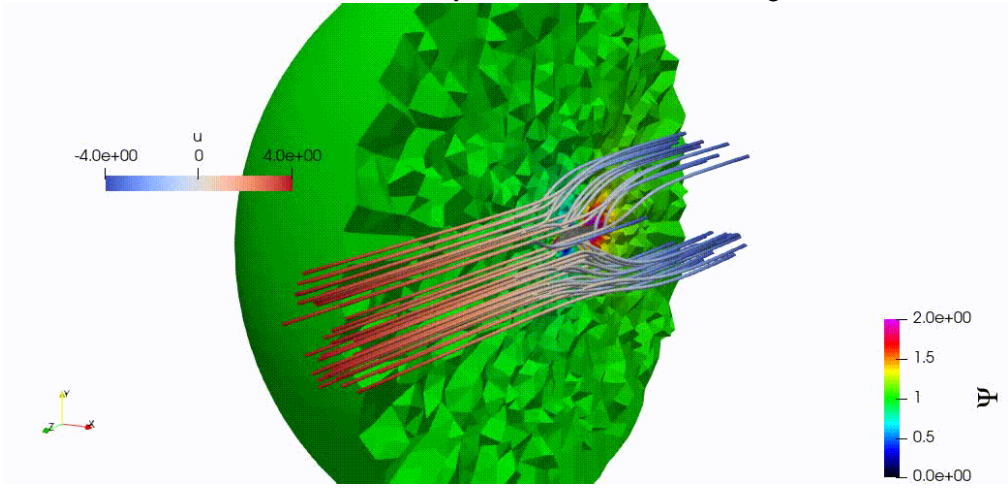
- Arcane + FEM started Nov. 2022 (**Young code**)
- Primary aim, **high-fidelity Earthquake** (source to site)
- ML Assisted Data-Assimilation: Fast solving

4M DOF Chamonix French Alps Simulation on 8 MPI in 22 min

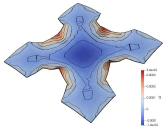


ArcaneFEM: modules

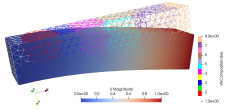
Potential Flow Theory simulation of delta wing UAV



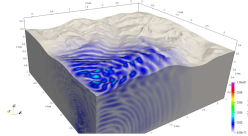
ArcaneFEM: modules



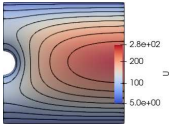
Bilaplacian
Steady-Vector(2)-3D



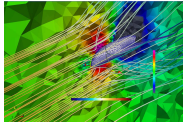
Elasticity
Steady-Vector(3)-3D



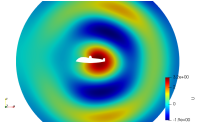
Soildynamics
Transient-Vector(3)-4D



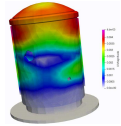
Fourier
Steady-Scalar-3D



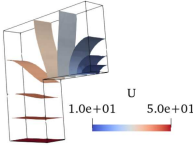
Aerodynamics
Steady-Scalar-3D



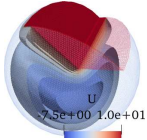
Acoustics
Steady-Scalar-3D



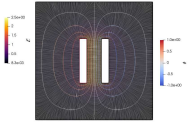
Elastodynamics
Transient-Vector(3)-4D



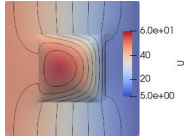
Laplace
Steady-Scalar-3D



Poisson
Steady-Scalar-3D

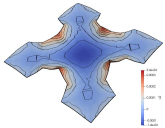


Electrostatic
Steady-Scalar-3D

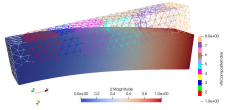


Heat transfer
Transient-Scalar-4D

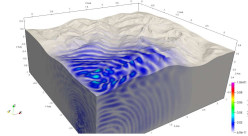
ArcaneFEM: modules



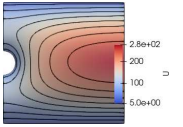
Bilaplacian
Steady-Vector(2)-3D



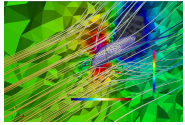
Elasticity (GPU)
Steady-Vector(3)-3D



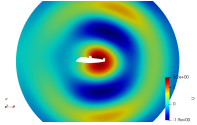
Soildynamics
Transient-Vector(3)-4D



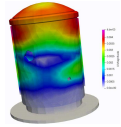
Fourier (GPU)
Steady-Scalar-3D



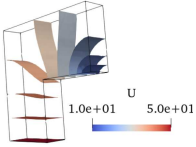
Aerodynamics
Steady-Scalar-3D



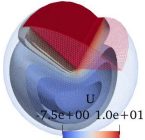
Acoustics
Steady-Scalar-3D



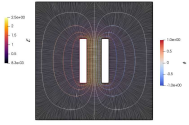
Elastodynamics
Transient-Vector(3)-4D



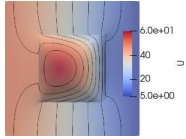
Laplace (GPU)
Steady-Scalar-3D



Poisson (GPU)
Steady-Scalar-3D

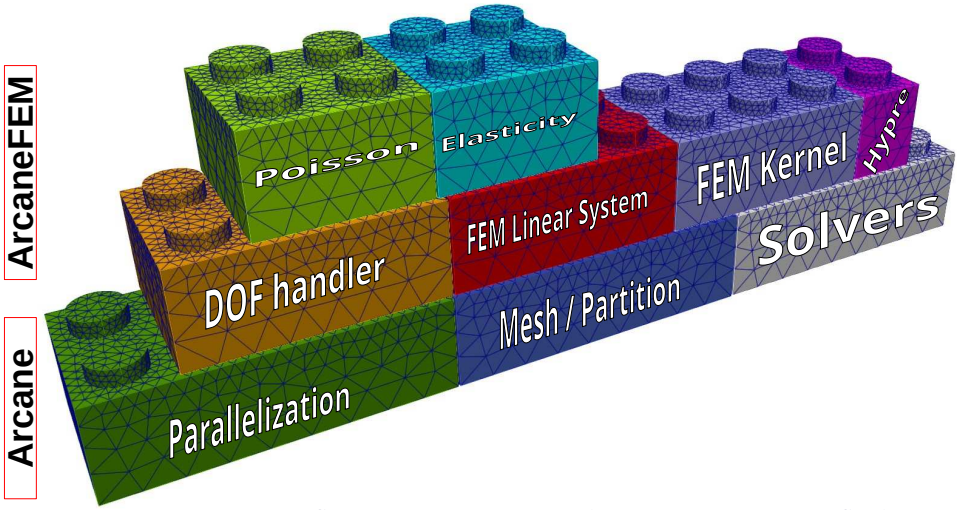


Electrostatic (GPU)
Steady-Scalar-3D

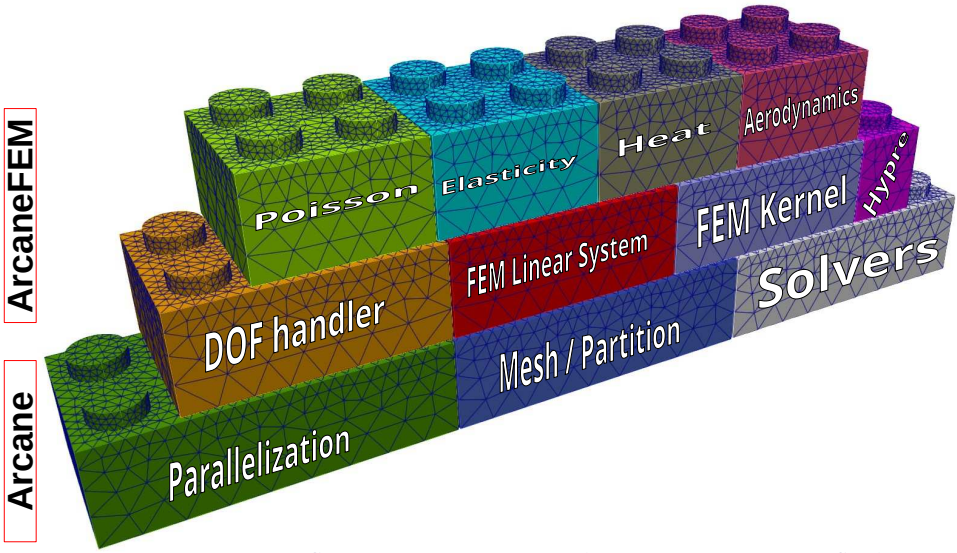


Heat transfer
Transient-Scalar-4D

ArcaneFEM: architecture



ArcaneFEM: architecture



Outline

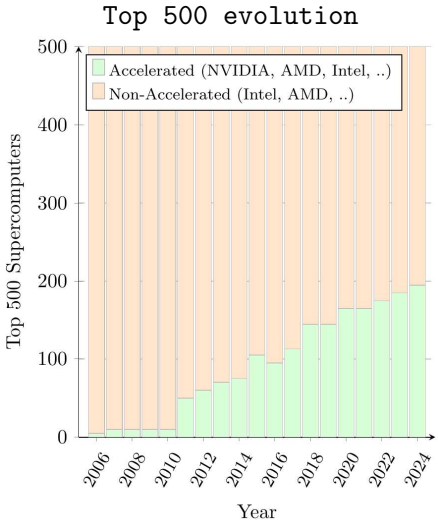
Outline

Motivation: why to evolve and use GPU's ?

- Earthquake simulations require solving billions of unknowns $\mathcal{O}(10^9)$
- Real time for **DA / ML**
- Road to **Exascale** & green simulations
  - top 500: **193/500** are accelerated; top 10: **9/10** are accelerated
  - **70**% of top HPC applications are accelerated
  - 2026 French SC **Alice Recoque**
- FEM is embracing GPU

"*It is not the strongest of the* ~~***species***~~ ***codes*** *that survive, not the most intelligent that survives. It is the one that is most acceptable to change*"

Charles Darwin

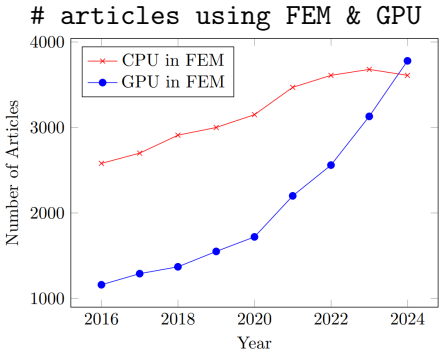


Motivation: why to evolve and use GPU's ?

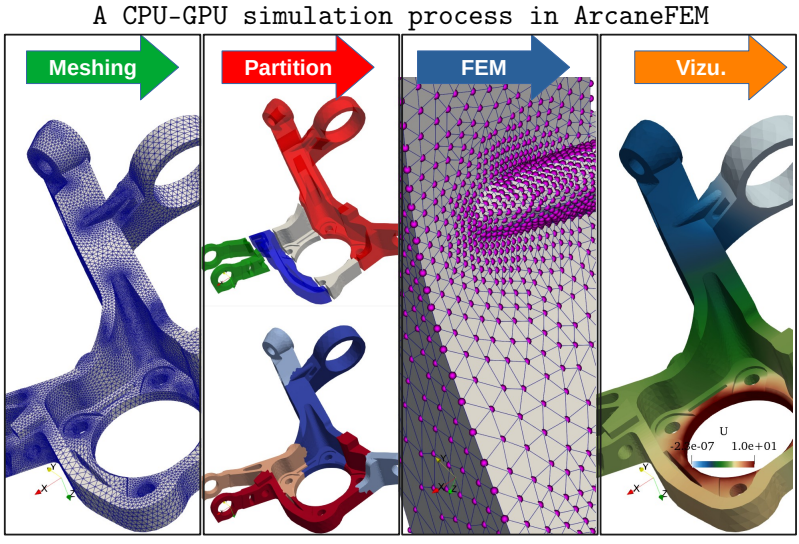
- Earthquake simulations require solving billions of unknowns $\mathcal{O}(10^9)$
- Real time for **DA / ML**
- Road to **Exascale** & green simulations
  - top 500: **193/500** are accelerated; top 10: **9/10** are accelerated
  - **70**% of top HPC applications are accelerated
  - 2026 French SC **Alice Recoque**
- FEM is embracing GPU

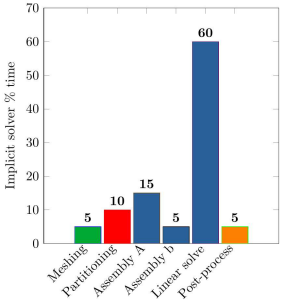
"*It is not the strongest of the* ~~***species***~~ ***codes*** *that survive, not the most intelligent that survives. It is the one that is most acceptable to change*"

Charles Darwin



GPGPU for full FEM



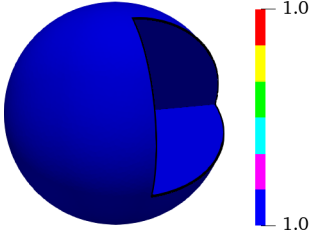


- **Full FEM on GPU**
  - Assembly $\mathbf{A}$, $\mathbf{b}$
  - Solve $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$
- Only Solving issues:
  - Data movement CPU-GPU
  - Low # MPI PU

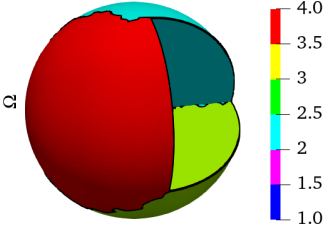
Strategy for GPGPU-CPU simulations

- Domain Decomposition paradigm for multi-GPU
- Attach MPI to GPU

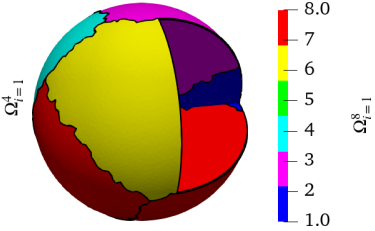
- mono 1 GPU simulation
- No partitioning, no MPI



- multi 4 GPU simulation
- 4 partitions, 4 MPI procs. attached



- multi 4 GPU-CPU simulation
- 8 partitions, 8 MPI procs. attached (batches of 2)



Outline

FEM assembly: classic Galerkin

- Works for CPU, sequential / parallel
- Pitfall : GPGPU race-conditions / race-hazard prone

$$\int_{\Omega^h} \nabla u \cdot \nabla v = \int_{\partial\Omega_N^h} \nabla u \cdot \mathbf{n} + \int_{\Omega^h} fv \qquad \& \qquad \textit{Dirichlet}$$

Implementation classic

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

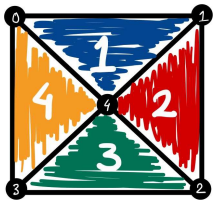
$$\int_{\partial\Omega_N^h} \nabla u \cdot \mathbf{n} + \int_{\Omega^h} fv \quad \mapsto \quad \int_{\mathcal{E}_i} \nabla u \cdot \mathbf{n} + \int_{\mathcal{T}_i} fv \quad \mapsto \quad \mathbf{b}$$

FEM assembly: race-hazard demonstration

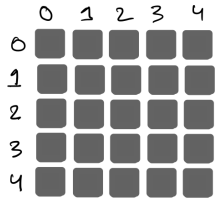
Global matrix via local stiffness matrix

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$A_{ij}^{(K)} = \sum_{p=1}^{3} w_p \nabla \phi_i(x_p, y_p) \cdot \nabla \phi_j(x_p, y_p) \, |K|$$



Mesh



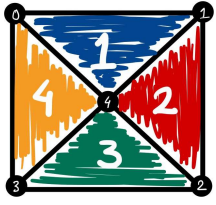
Matrix

FEM assembly: race-hazard demonstration

Global matrix via local stiffness matrix

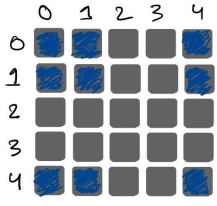
$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$A_{ij}^{(K)} = \sum_{p=1}^{3} w_p \nabla \phi_i(x_p, y_p) \cdot \nabla \phi_j(x_p, y_p) \, |K|$$



Mesh

Element Stiffness Matrix



Matrix

FEM assembly: race-hazard demonstration

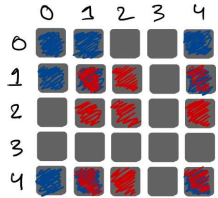
Global matrix via local stiffness matrix

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$A_{ij}^{(K)} = \sum_{p=1}^{3} w_p \nabla \phi_i(x_p, y_p) \cdot \nabla \phi_j(x_p, y_p) \, |K|$$

Mesh

Element Stiffness Matrix



Matrix

FEM assembly: race-hazard demonstration

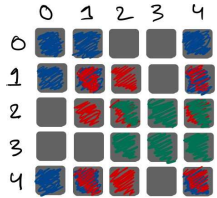
Global matrix via local stiffness matrix

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$A_{ij}^{(K)} = \sum_{p=1}^{3} w_p \nabla \phi_i(x_p, y_p) \cdot \nabla \phi_j(x_p, y_p) \, |K|$$

Mesh

Element Stiffness Matrix



Matrix

FEM assembly: race-hazard demonstration

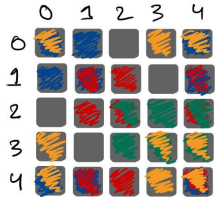
Global matrix via local stiffness matrix

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$A_{ij}^{(K)} = \sum_{p=1}^{3} w_p \nabla \phi_i(x_p, y_p) \cdot \nabla \phi_j(x_p, y_p) \, |K|$$

Mesh

Element Stiffness Matrix



Matrix

FEM assembly: race-hazard demonstration

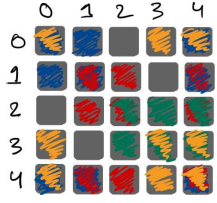
Global matrix via local stiffness matrix

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$A_{ij}^{(K)} = \sum_{p=1}^{3} w_p \nabla \phi_i(x_p, y_p) \cdot \nabla \phi_j(x_p, y_p) \, |K|$$

Mesh

Element Stiffness Matrix



Matrix

```
ax::doAtomic<ax::eAtomicOperation::Add>(in_out_val_csr(begin), v);
```

FEM assembly: Atomic Free Node-Wise Assembly

Implementation classic

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{T}_i} \nabla u \cdot \nabla v \quad \mapsto \quad \mathbf{A}$$

$$\int_{\partial\Omega_N^h} \nabla u \cdot \mathbf{n} + \int_{\Omega^h} fv \quad \mapsto \quad \int_{\mathcal{E}_i} \nabla u \cdot \mathbf{n} + \int_{\mathcal{T}_i} fv \quad \mapsto \quad \mathbf{b}$$

Node-Wise implementation

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{A}$$

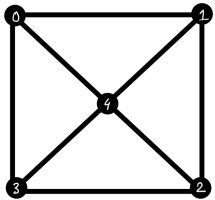
$$\int_{\partial\Omega_N^h} \nabla u \cdot \mathbf{n} + \int_{\Omega^h} fv \quad \mapsto \quad \int_{\mathcal{P}_i \in \partial\Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \partial\Omega^h} (\cdot) + \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{b}$$

Atomic Free Node-Wise Assembly: demonstration

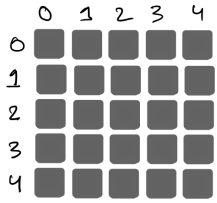
Global matrix via **local stiffness vectors**

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{A}$$

$$\{A_i\} = \sum_{K \in \mathcal{K}_I} \sum_{p=1}^{3} w_p \left(\nabla \phi_i(x_I, y_I) \otimes \nabla \phi_j(x_p, y_p) \right) |K|$$



Mesh



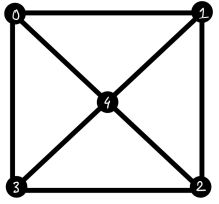
Matrix

Atomic Free Node-Wise Assembly: demonstration

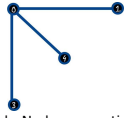
Global matrix via **local stiffness vectors**

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{A}$$

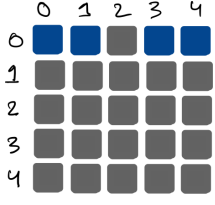
$$\{A_i\} = \sum_{K \in \mathcal{K}_I} \sum_{p=1}^{3} w_p \left(\nabla \phi_i(x_I, y_I) \otimes \nabla \phi_j(x_p, y_p) \right) |K|$$



Mesh


Node-Node connectivity


Nodal Stiffness Vector



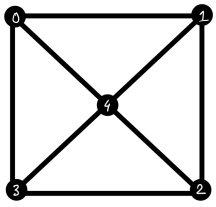
Matrix

Atomic Free Node-Wise Assembly: demonstration

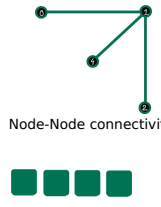
Global matrix via **local stiffness vectors**

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{A}$$

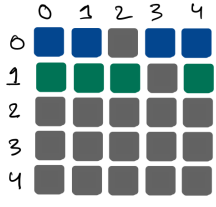
$$\{A_i\} = \sum_{K \in \mathcal{K}_I} \sum_{p=1}^{3} w_p \left(\nabla \phi_i(x_I, y_I) \otimes \nabla \phi_j(x_p, y_p) \right) |K|$$



Mesh



Nodal Stiffness Vector



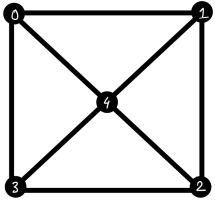
Matrix

Atomic Free Node-Wise Assembly: demonstration

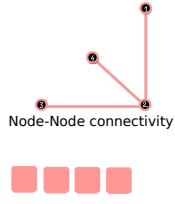
Global matrix via **local stiffness vectors**

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{A}$$

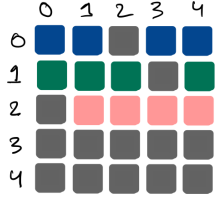
$$\{A_i\} = \sum_{K \in \mathcal{K}_I} \sum_{p=1}^{3} w_p \left(\nabla \phi_i(x_I, y_I) \otimes \nabla \phi_j(x_p, y_p) \right) |K|$$



Mesh



Nodal Stiffness Vector



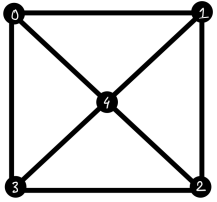
Matrix

Atomic Free Node-Wise Assembly: demonstration

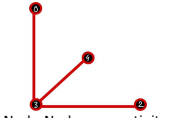
Global matrix via **local stiffness vectors**

$$\int_{\Omega^h} \nabla u \cdot \nabla v \quad \mapsto \int_{\mathcal{P}_i \in \Omega^h} (\cdot) + \int_{\mathcal{E}_i \in \Omega^h} (\cdot) \quad \mapsto \quad \mathbf{A}$$

$$\{A_i\} = \sum_{K \in \mathcal{K}_I} \sum_{p=1}^{3} w_p \left(\nabla \phi_i(x_I, y_I) \otimes \nabla \phi_j(x_p, y_p) \right) |K|$$

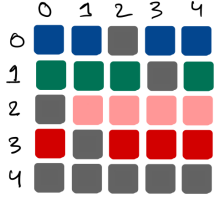


Mesh



Node-Node connectivity


Nodal Stiffness Vector

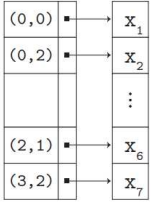


Matrix

FEM matrices: various data-structures in ArcaneFEM

x_1	0	x_2	x_3
0	0	x_4	0
x_5	x_6	0	0
0	0	x_7	0

Dictionary Of Keys (DOK)



C++ `std::unordered_map`

- CPU only, slow search
- random memory access
- Easy to implement, scales well

(Block) Compress Sparse Row ((B)CSR)

row:	0	3	4	6			
col:	0	2	3	2	0	1	2
val:	x_1	x_2	x_3	x_4	x_5	x_6	x_7

matrix with 3 arrays + `row` is compressed

- GPU & CPU, fast search

COOrdinate list (COO)

row:	0	0	0	1	2	3	3
col:	0	2	3	2	0	1	2
val:	x_1	x_2	x_3	x_4	x_5	x_6	x_7

matrix with 3 arrays

- GPU & CPU, fast search

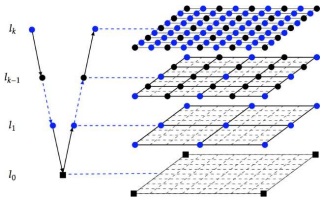
FEM solving: multi-GPU solving via Hypre

What:

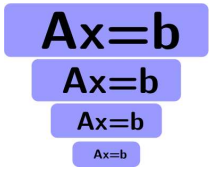
- Use hierarchy discretizations
  - meshes $\mapsto$ GMG, Matrices $\mapsto$ AMG
- Restrict and interpolate cycle
- Equations of interest elliptic : multigrid precod.
- AMG is used as precond. to Krylov solver CG

How:

- Hypre open-source : BoomerAMG
  - + multi-GPU support (NVIDIA, AMD (2.28))
  - + Coarsner on GPU
  - + Smoothner on GPU
  - + Interpolation Kernel on GPU
  - - CPU-GPU not handled
  - - COO format not handled



- Geometric multigrid



- Algebraic multigrid

Outline

Test Bench

NVIDIA Quadro T2000 Max-Q

TU117	1024	64	32	4 GB	GDDR5	128 bit
GRAPHICS PROCESSOR	CORES	TMUS	ROPS	MEMORY SIZE	MEMORY TYPE	BUS WIDTH

NVIDIA A100 SXM4 80 GB

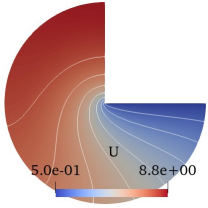
GA100	6912	432	160	80 GB	HBM2e	5120 bit
GRAPHICS PROCESSOR	CORES	TMUS	ROPS	MEMORY SIZE	MEMORY TYPE	BUS WIDTH

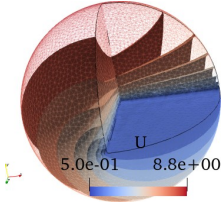
4 GPUs / Node

AMD Radeon Instinct MI250X

Aldebaran	14080	880	N/A	128 GB	HBM2e	8192 bit
GRAPHICS PROCESSOR	CORES	TMUS	ROPS	MEMORY SIZE	MEMORY TYPE	BUS WIDTH

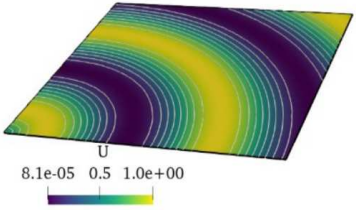
8 GDCs (Graphic Computer Dies) / Node

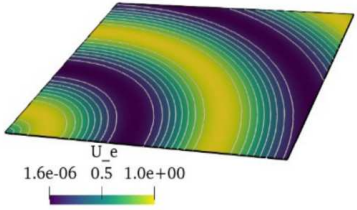


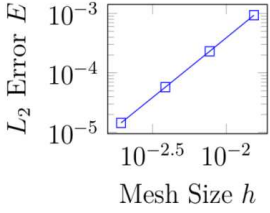


Poissons Problem
2D / 3D

Solver verification via manufactured solution



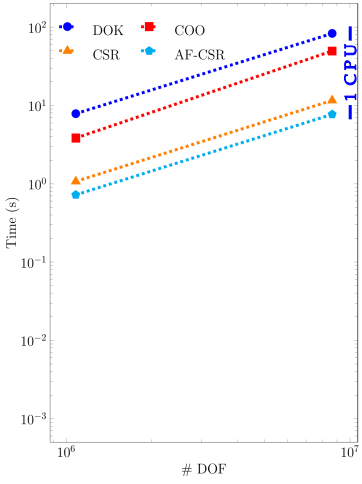


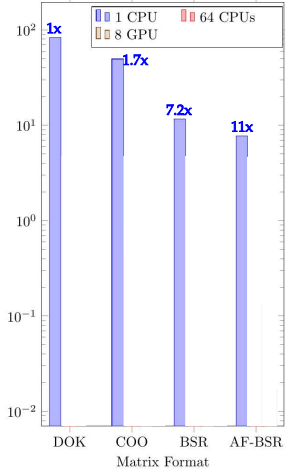


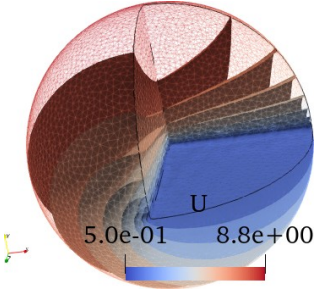
Comparison of FEM solution (left) from ArcaneFEM using the new GPU algorithm and the reference exact solution (right) from the following equation :

$$u_e(\boldsymbol{x}) = \frac{1}{2} + \frac{1}{2}\sin\left(\frac{\pi\sqrt{x^2+y^2}}{\alpha}\right) \quad \forall \boldsymbol{x} \in \Omega$$

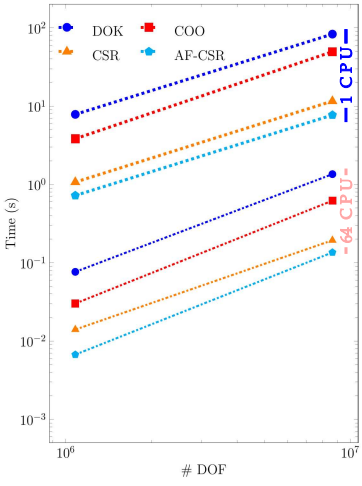
Benchmark Supercomputer $\forall\ N_{\text{DOF}} = \{1\text{M}, 10\text{M}\}$

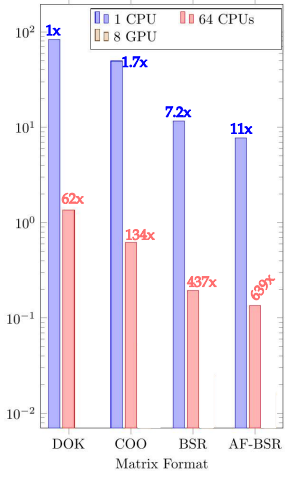


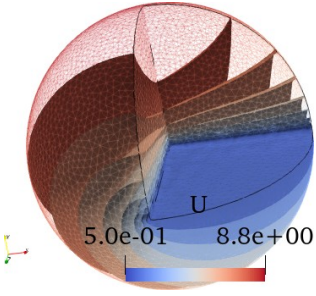




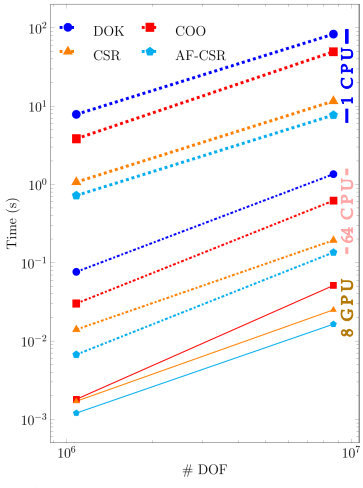
Benchmark Supercomputer $\forall\ N_{\text{DOF}} = \{1\text{M}, 10\text{M}\}$

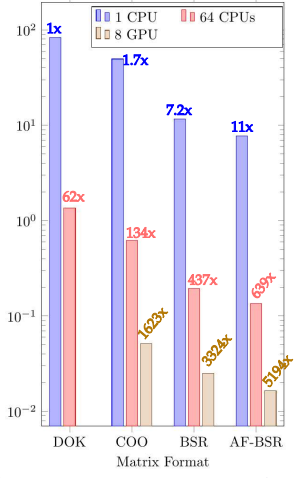


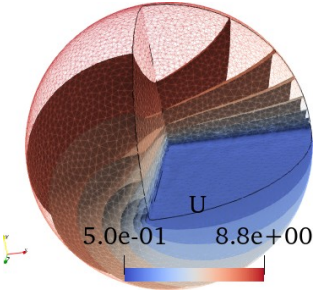




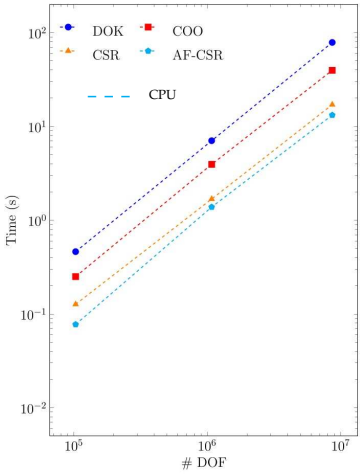
Benchmark Supercomputer $\forall\ N_{\mathrm{DOF}} = \{1\mathrm{M}, 10\mathrm{M}\}$

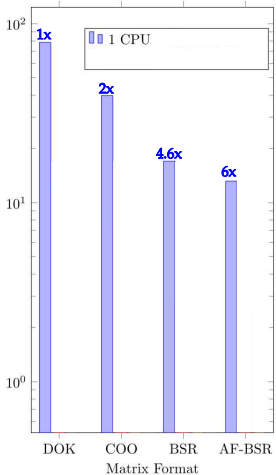


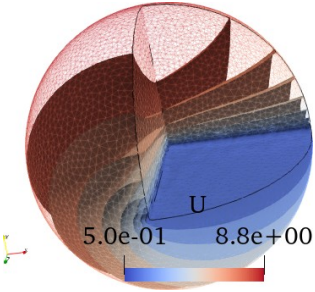




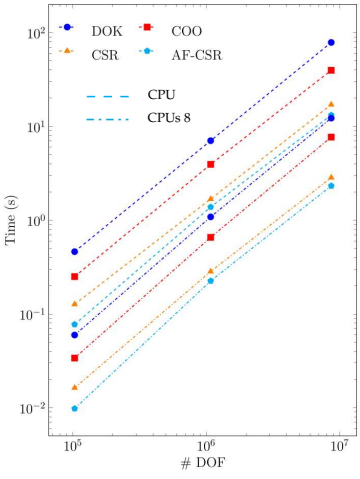
Benchmark Laptop $\forall\ N_{\text{DOF}} = \{100\text{K}, 1\text{M}, 10\text{M}\}$

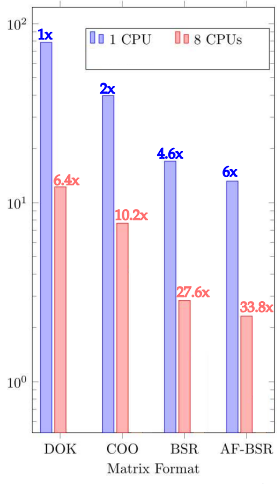


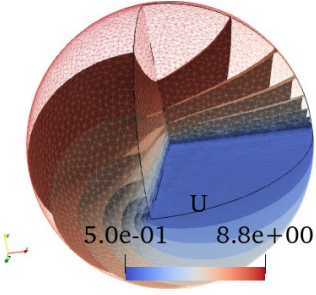




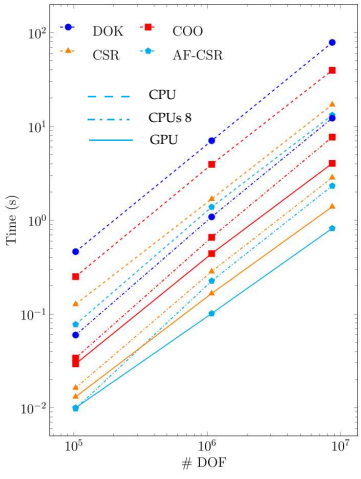
Benchmark Laptop $\forall\ N_{\text{DOF}} = \{100\text{K}, 1\text{M}, 10\text{M}\}$

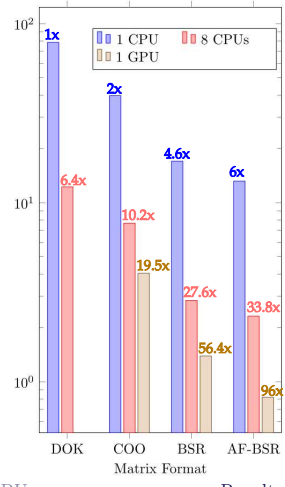


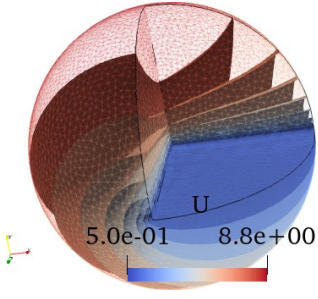




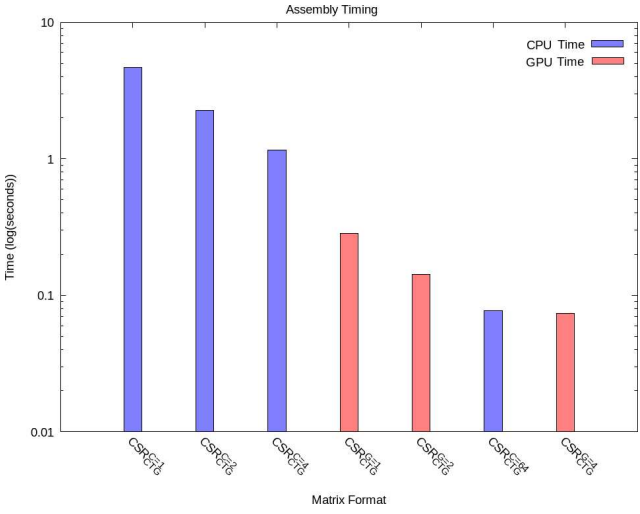
Benchmark Laptop $\forall\ N_{\text{DOF}} = \{100\text{K}, 1\text{M}, 10\text{M}\}$

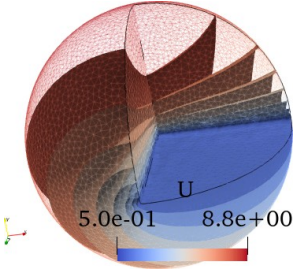




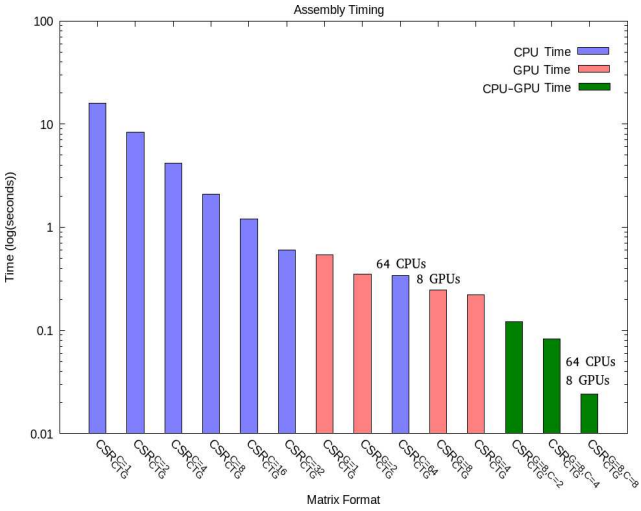


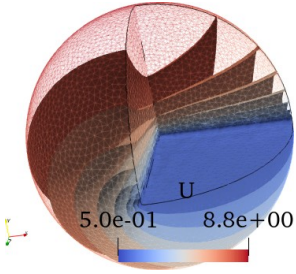
Benchmark Supercomputers $N_{\text{DOF}} = 18\text{M}$



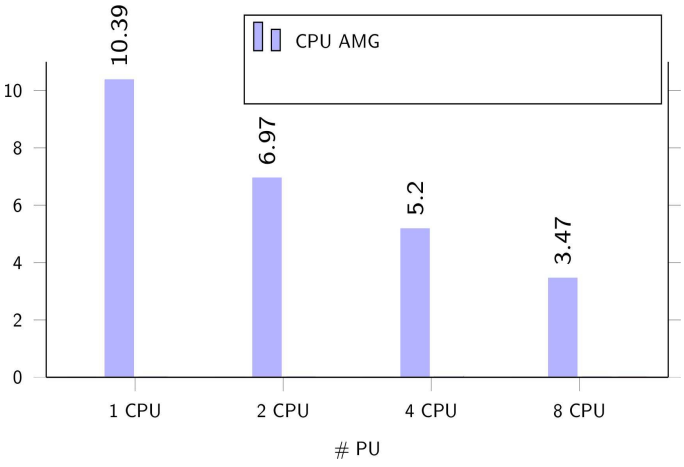


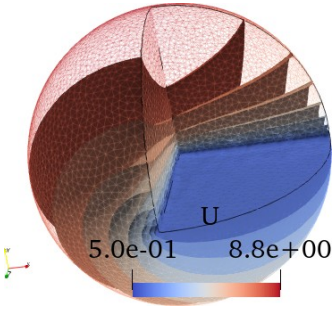
Benchmark Supercomputers $N_{\text{DOF}} = 18\text{M}$



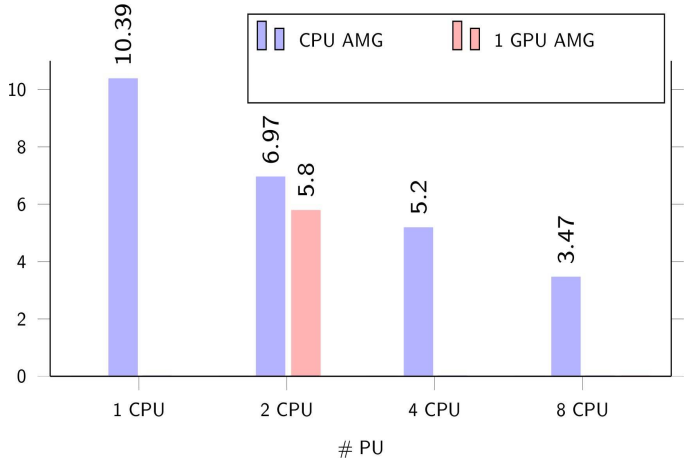


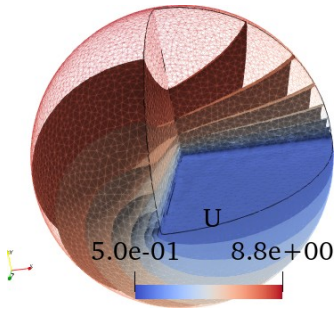
Benchmark Laptop Preconditioner $N_{\text{DOF}} = 3.5\text{M}$



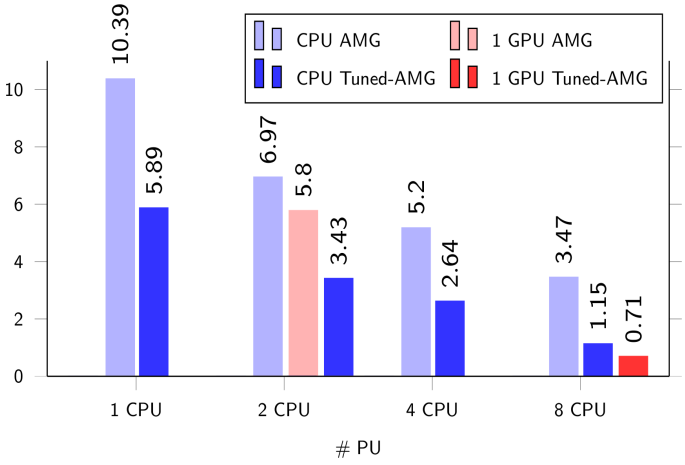


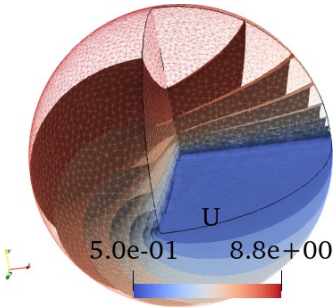
Benchmark Laptop Preconditioner $N_{\text{DOF}} = 3.5\text{M}$



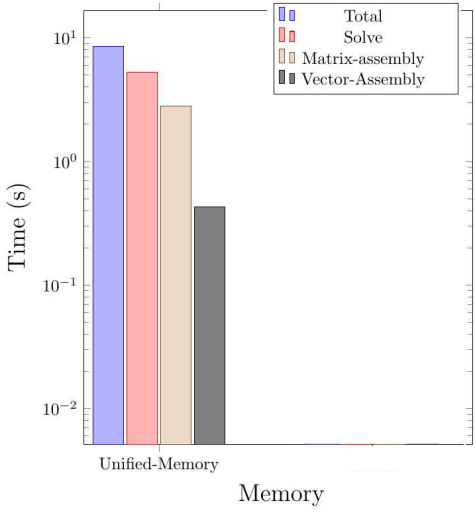


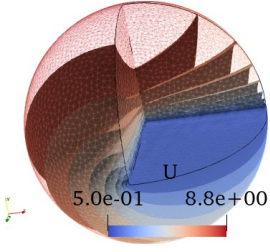
Benchmark Laptop Preconditioner $N_{\text{DOF}} = 3.5\text{M}$



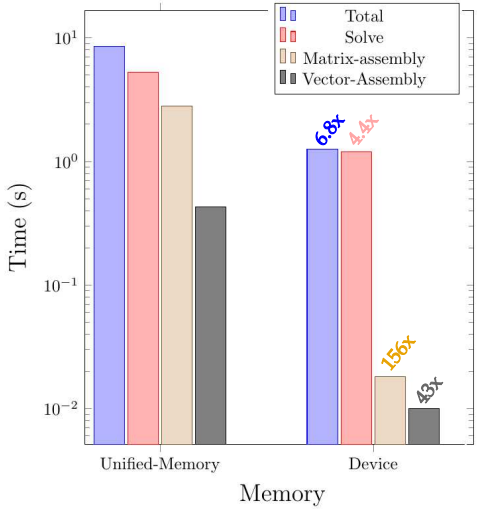


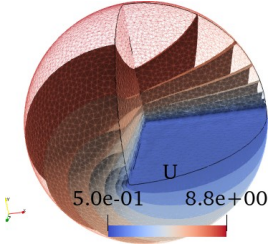
Memory resource should we care ?



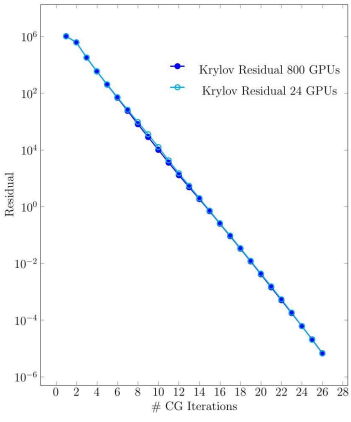


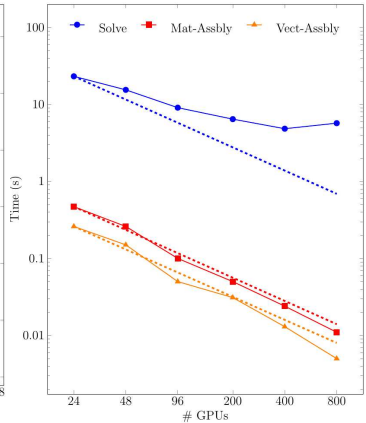
Memory resource should we care ?

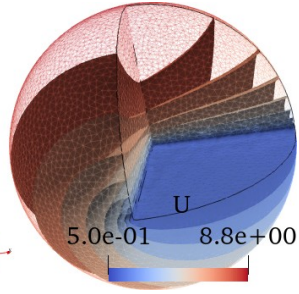




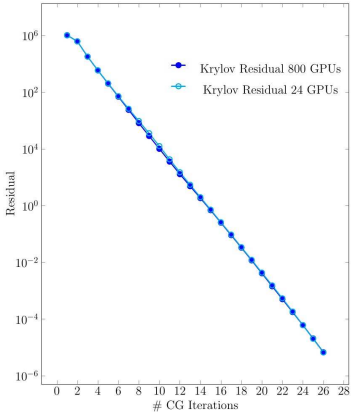
Benchmark $N_{\text{DOF}} = 0.27\text{B}$ on 800 GPUs

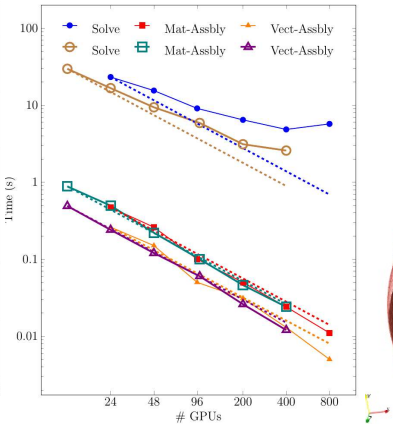


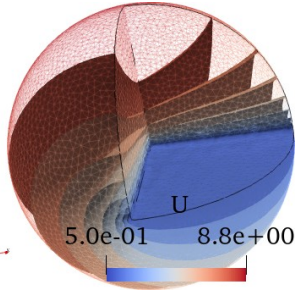




Benchmark $N_{\text{DOF}} = 0.27\text{B}$ on 800 GPUs







1B cell problem on 100 GPUs

Mesh:

- 1 Billion Tetrahedrons
- KNL fat-node (128 Threads Gmsh)

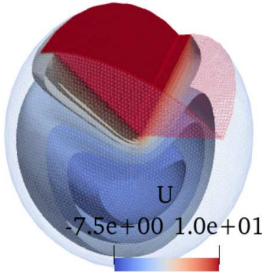
Partitioning:

- dual-level 100 MPI Mesh reading
- Parmetis

Timing:

- Matrix assembly 0.0001 (s)
- Solving 210 (s) (**not tuned**)

NVIDIA A100



Outline

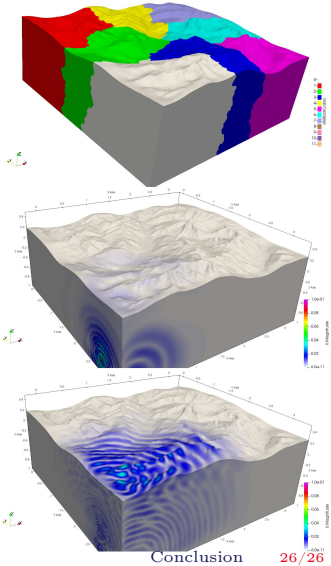
Key Takeaways & What's Next

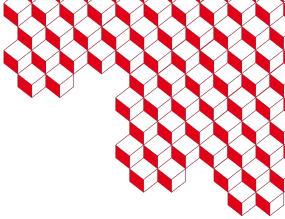
Conclusion

- ArcaneFEM first works on **GPGPU & CPU-GPU** FEM solving
- **Matrix assembly** via **porting** approach : **Atomics**
- **Atomic free assembly**, using **node-wise** approach
- **Benchmarks on GPUs**: **MI250X**, **A100**, & **Laptop**

Future Works

- **GPU** for all modules (Elastodynamic-Soildynamics)
- Focus on **non-linear** physics, **multi-physics**, **PASSMO** modules
- **top-ii-vol** integration: combined **mesher-partitioner on GPUs**
- **APU's** benchmarking: **MI300A, GH200**
- Linear system solve:
  - CPU = theoretical AMG ; **GPU != theoretical AMG**
  - AMG with **full CPU-GPU**
  - **Matrix free solvers**





ArcaneFEM: Scalable GPGPU Implicit FEM Solver for Unstructured Meshes

Backup1

Weak form of Poisson's problem

$$\int_\Omega \nabla u \cdot \nabla v \, d\Omega = \int_\Omega f v \, d\Omega, \quad \forall v \in V.$$

Using basis functions $\{\phi\}_{i=1}^3$

$$u_h = \sum_i U_i \phi_i, \quad v_h = \phi_j.$$

Substituting into the weak form:

$$\sum_i U_i \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, d\Omega = \int_\Omega f \phi_j \, d\Omega.$$

This leads to:

$$\mathbf{A}_{ij} = \int_\Omega \nabla \phi_i \cdot \nabla \phi_j \, d\Omega.$$

Local Stiffness Matrix for Element K

For each finite element K, the local stiffness matrix is:

$$A_{ij}^{(K)} = \int_K \nabla \phi_i \cdot \nabla \phi_j \, dK.$$

For TRIA3 $\{i\}_{k=1}^3$ $\{j\}_{k=1}^3$ element:

$$A^{(K)} = \begin{bmatrix} a_{11}^{(K)} & a_{12}^{(K)} & a_{13}^{(K)} \\ a_{21}^{(K)} & a_{22}^{(K)} & a_{23}^{(K)} \\ a_{31}^{(K)} & a_{32}^{(K)} & a_{33}^{(K)} \end{bmatrix}.$$

Backup1

```
14  /*---------------------------------------------------------------------------*/
15  /**
16   * @brief Computes the element matrix for a triangular element (P1 FE).
17   *
18   * This function calculates the integral of the expression:
19   *       a(u,v) = ∫∫ (∂u/∂x ∂v/∂x  + ∂u/∂y ∂v/∂y)dΩ
20   *
21   * Steps involved:
22   * 1. Calculate the area of the triangle.
23   * 2. Compute the gradients of the shape functions.
24   * 3. Return a(u,v);
25   */
26  /*---------------------------------------------------------------------------*/
27
28  RealMatrix<3, 3> FemModule::
29  _computeElementMatrixTria3(Cell cell)
30  {
31    Real area = ArcaneFemFunctions::MeshOperation::computeAreaTria3(cell, m_node_coord);
32
33    Real3 dxU = ArcaneFemFunctions::FeOperation2D::computeGradientXTria3(cell, m_node_coord);
34    Real3 dyU = ArcaneFemFunctions::FeOperation2D::computeGradientYTria3(cell, m_node_coord);
35
36    return area * (dxU ^ dxU) + area * (dyU ^ dyU);
37  }
38
39  /*---------------------------------------------------------------------------*/
40  /*---------------------------------------------------------------------------*/
41
42  ARCCORE_HOST_DEVICE RealMatrix<3, 3>
43  _computeElementMatrixTria3Gpu(CellLocalId cell_lid, const IndexedCellNodeConnectivityView& cn_cv, const ax::VariableNodeReal3InView& in_node_coord)
44  {
45    Real area = Arcane::FemUtils::Gpu::MeshOperation::computeAreaTria3(cell_lid, cn_cv, in_node_coord);
46
47    Real3 dxU = FemUtils::Gpu::FeOperation2D::computeGradientXTria3(cell_lid, cn_cv, in_node_coord);
48    Real3 dyU = FemUtils::Gpu::FeOperation2D::computeGradientYTria3(cell_lid, cn_cv, in_node_coord);
49
50    return area * (dxU ^ dxU) + area * (dyU ^ dyU);
51  }
52
53  /*---------------------------------------------------------------------------*/
54  /*---------------------------------------------------------------------------*/
55
56  ARCCORE_HOST_DEVICE RealMatrix<1, 3>
57  _computeElementVectorTria3Gpu(CellLocalId cell_lid, const IndexedCellNodeConnectivityView& cn_cv, const ax::VariableNodeReal3InView& in_node_coord, Int32 node_lid)
58  {
59    Real area = Arcane::FemUtils::Gpu::MeshOperation::computeAreaTria3(cell_lid, cn_cv, in_node_coord);
60
61    Real3 dxU = FemUtils::Gpu::FeOperation2D::computeGradientXTria3(cell_lid, cn_cv, in_node_coord);
62    Real3 dyU = FemUtils::Gpu::FeOperation2D::computeGradientYTria3(cell_lid, cn_cv, in_node_coord);
63
64    Real3 node_vector_integral = area * dxU[node_lid] * dxU + area * dyU[node_lid] * dyU;
65    return { node_vector_integral[0], node_vector_integral[1], node_vector_integral[2] };
66  }
67
```

Backup1

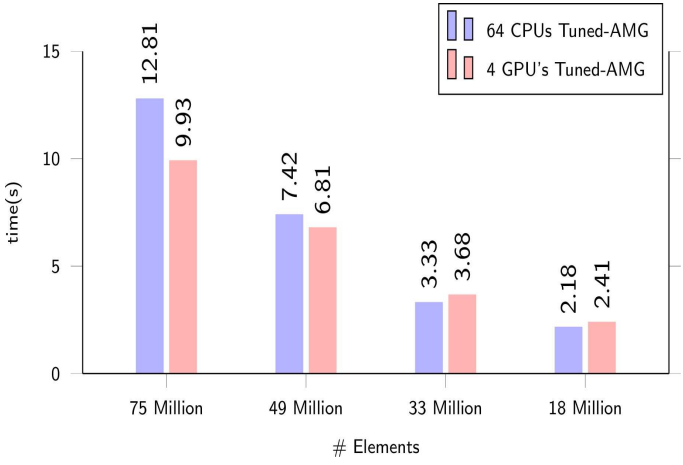
```
/*---------------------------------------------------------------------------*/
/**
 * @brief Computes the element matrix for a triangular element (P1 FE).
 *
 * This function calculates the integral of the expression:
 *       a(u,v) = ∫∫ (∂u/∂x ∂v/∂x  + ∂u/∂y ∂v/∂y)dΩ
 *
 * Steps involved:
 * 1. Calculate the area of the triangle.
 * 2. Compute the gradients of the shape functions.
 * 3. Return a(u,v);
 */
/*---------------------------------------------------------------------------*/

RealMatrix<3, 3> FemModule::
computeElementMatrixTria3(Cell cell)
{
  Real area = ArcaneFemFunctions::MeshOperation::computeAreaTria3(cell, m_node_coord);

  Real3 dxU = ArcaneFemFunctions::FeOperation2D::computeGradientXTria3(cell, m_node_coord);
  Real3 dyU = ArcaneFemFunctions::FeOperation2D::computeGradientYTria3(cell, m_node_coord);

  return area * (dxU ^ dxU) + area * (dyU ^ dyU);
}

/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/

ARCCORE_HOST_DEVICE RealMatrix<3, 3>
computeElementMatrixTria3Gpu(CellLocalId cell_lid, const IndexedCellNodeConnectivityView& cn_cv, const ax::VariableNodeReal3InView& in_node_coord)
{
  Real area = Arcane::FemUtils::Gpu::MeshOperation::computeAreaTria3(cell_lid, cn_cv, in_node_coord);

  Real3 dxU = FemUtils::Gpu::FeOperation2D::computeGradientXTria3(cell_lid, cn_cv, in_node_coord);
  Real3 dyU = FemUtils::Gpu::FeOperation2D::computeGradientYTria3(cell_lid, cn_cv, in_node_coord);

  return area * (dxU ^ dxU) + area * (dyU ^ dyU);
}

/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/

ARCCORE_HOST_DEVICE RealMatrix<1, 3>
computeElementVectorTria3Gpu(CellLocalId cell_lid, const IndexedCellNodeConnectivityView& cn_cv, const ax::VariableNodeReal3InView& in_node_coord, Int32 node_lid)
{
  Real area = Arcane::FemUtils::Gpu::MeshOperation::computeAreaTria3(cell_lid, cn_cv, in_node_coord);

  Real3 dxU = FemUtils::Gpu::FeOperation2D::computeGradientXTria3(cell_lid, cn_cv, in_node_coord);
  Real3 dyU = FemUtils::Gpu::FeOperation2D::computeGradientYTria3(cell_lid, cn_cv, in_node_coord);

  Real3 node_vector_integral = area * dxU[node_lid] * dxU + area * dyU[node_lid] * dyU;
  return { node_vector_integral[0], node_vector_integral[1], node_vector_integral[2] };
}
```

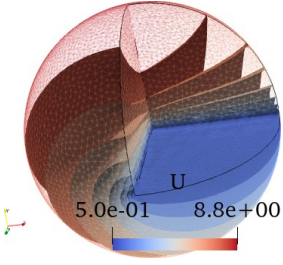
CPU

GPU

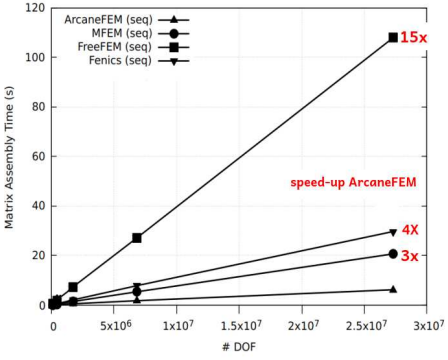
AF-GPU

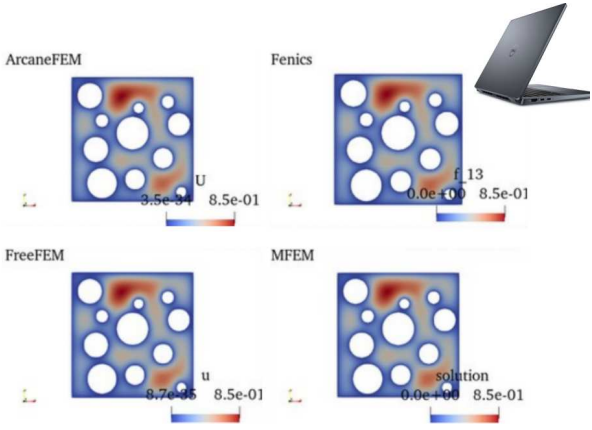
Backup2: Large problems are needed for GPU



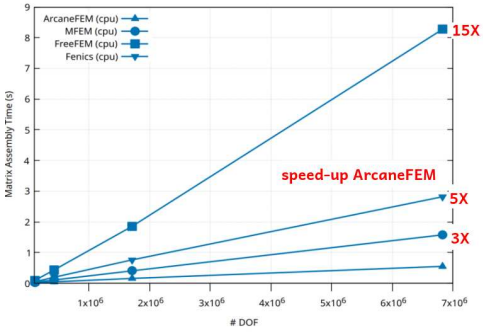


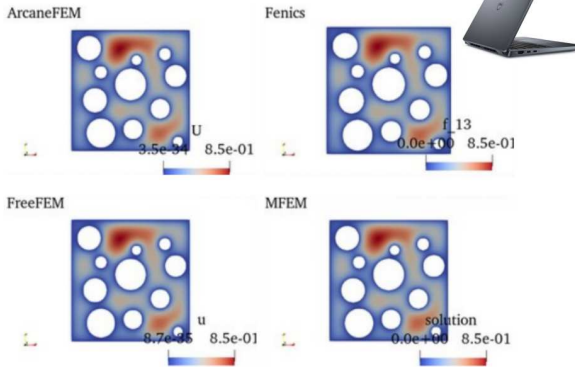
Backup3: Comparison with state of the art FEM solver



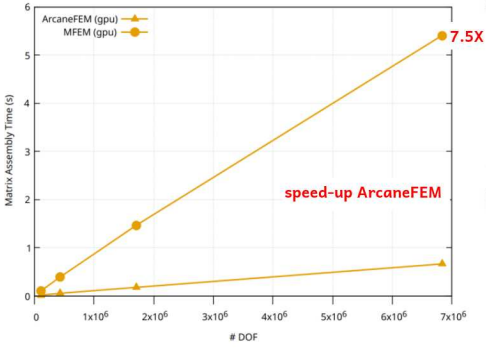


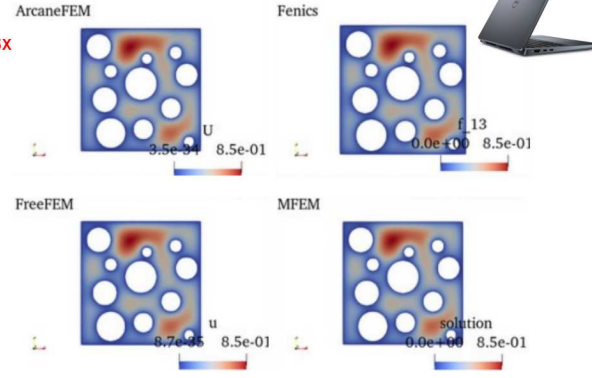
Backup3: Comparison with state of the art FEM solver



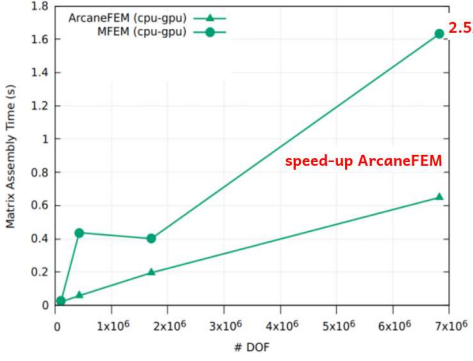


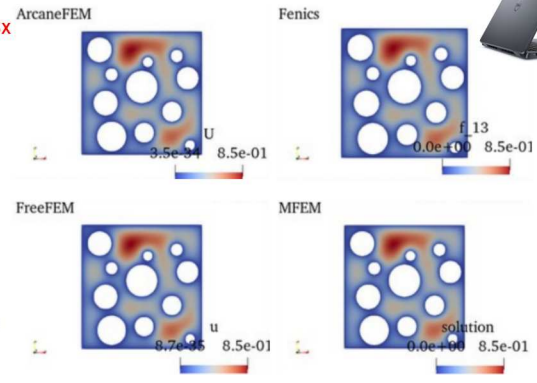
Backup3: Comparison with state of the art FEM solver



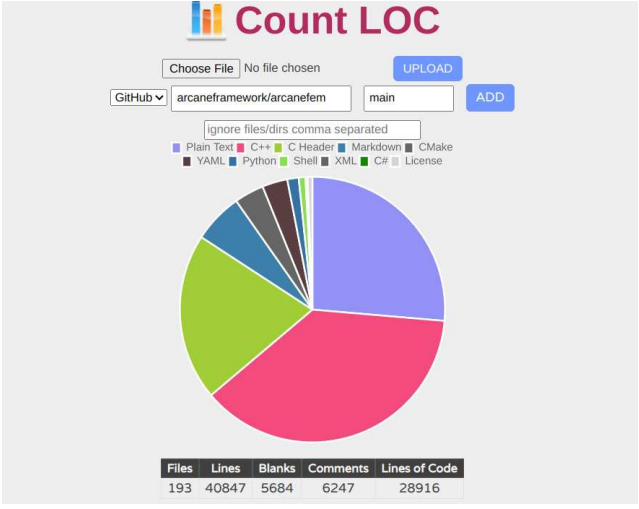


Backup3: Comparison with state of the art FEM solver



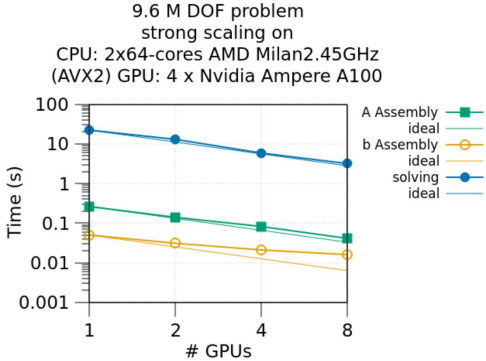


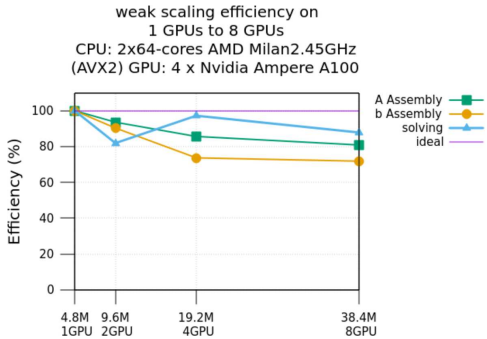
Backup3: Lines of code



Files	Lines	Blanks	Comments	Lines of Code
193	40847	5684	6247	28916

Backup4: Scaling weak and strong NVIDIA





Backup5: Elasticity vs Poisson