TRA105 - GPU-Accelerated Computational Methods

GPU-Accelerated FEM Solver

Stefano Ribes

Overview

- Problem Description
- FEM Algorithm
- K-Assembly Profiling
- Linear Solvers Profiling
- Next Steps in the Project

FEM Algorithm

- K-Assembly
- Linear Solver

Problem Description

- Stress – Strain Relations for Plane Problem: K @ a = f
  where:

    K : Stress-Strain matrix or Elasticity matrix
    a : the Strain Components in a Plane Solid
    f : the Stress

- Finite Element Approximation (on the right)
- Our Problem is to determine the Strain and Stress.

$$\{a\} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \\ \dfrac{\partial u}{\partial y} \end{Bmatrix} = [\mathbf{B}] \begin{Bmatrix} u_x^0 \\ u_y^0 \\ \vdots \\ u_x^N \\ u_y^N \end{Bmatrix} = [\mathbf{B}]\{q\}$$

FEM Algorithm

1. Calculate the Nodal Displacements

At this stage, we use the simplest structure of the grid to test the algorithm, which means the Nodal Displacements are naively generated by hands.

A more general way to calculate the Nodal Displacements is to use the principle of minimum total potential energy.

2. Calculation of Strains and Stresses

Step 1: Use the Equation given above in the Finite Element Approximation Part to Calculate the Strain Components.

Step 1: Use $Ka = f$ Calculate the Stresses.

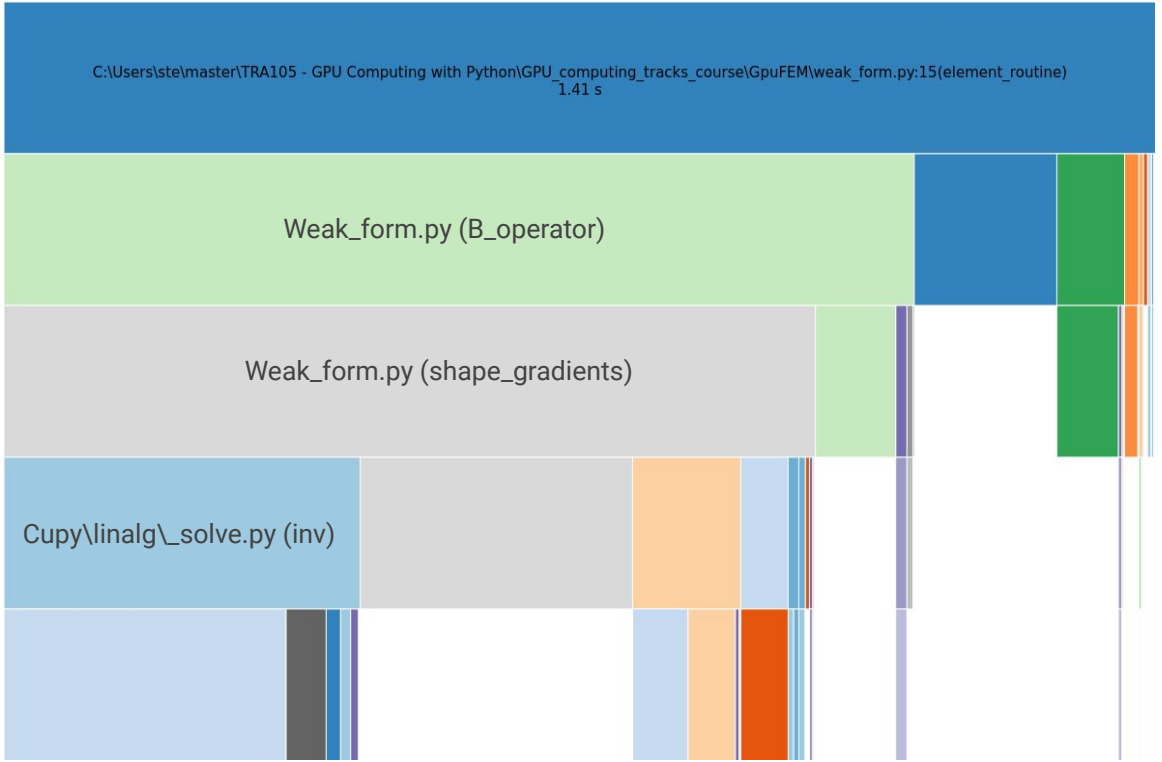
K-Assembly Profiling

- Profiling
- Possible GPU implementations

K-Assembly Profiling on GPU (CuPy)

- The indexing of the sparse matrix K is dominating the computation
- Because of that, we profiled on a small densely-allocated K matrix
- Inverse operation alone occupies around 30% of the cumulative execution time:

Cumtime	%	Filename(Function)
1.415	100%	Weak_form.py (element_routine)
1.118	79%	Weak_form.py (B_operator)
0.9967	70%	Weak_form.py (shape_gradients)
0.4376	31%	Cupy\linalg_solve.py (inv)



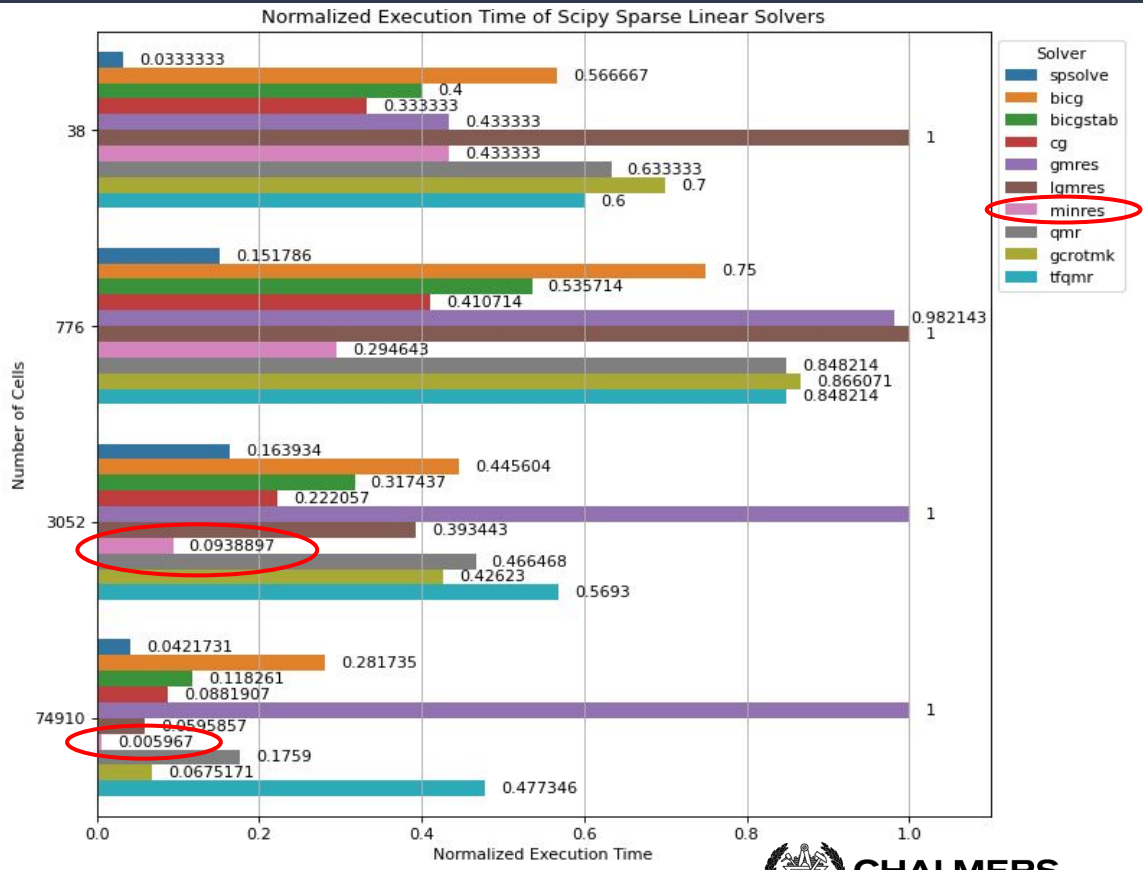
Linear Solvers Profiling

- CPU Profiling
- GPU Profiling

CPU Sparse Linear Solvers Profiling (SciPy)

- Profiled on an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz
- 8 cores without hyperthreading
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 12288K

- Varying Grid mesh granularity (resulting different number of cells)
- Exec. time averaged over 20 runs
- The lower the better
- spsolve and minres achieved best performance

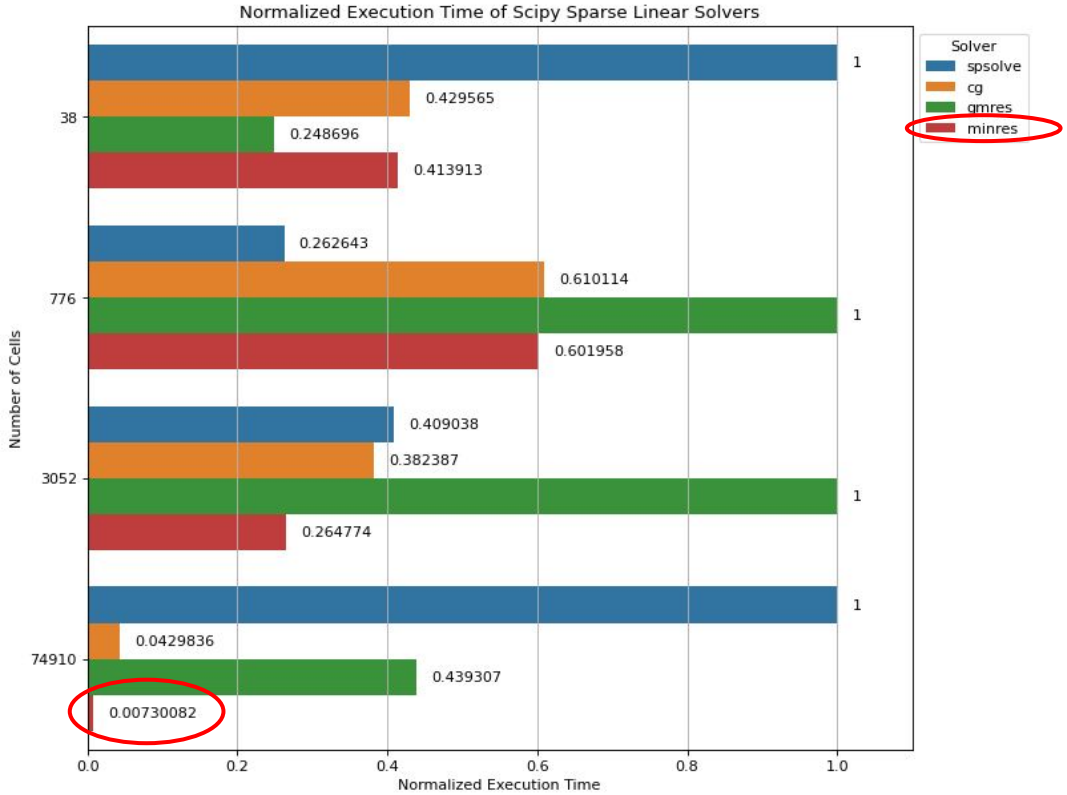


GPU Sparse Linear Solvers Profiling (CuPy)

- NVIDIA GeForce RTX 3060
- 12GB of GDDR6 memory
- 3,584 CUDA cores
- CUDA Version 11.2

- Varying Grid mesh granularity (resulting different number of cells)
- Exec. time averaged over 20 runs
- The lower the better
- spsolve and minres achieved best performance

Fine tuning the linear solver parameters??



Planned Next Steps

- Efficient Store and Indexing of K Matrix
- CuPy Implementation
- Custom GPU Kernels

Planned Next Steps

- Efficient Store and Indexing of the K Matrix
  - Linked-list Format (ll_mat)
  - Compressed Sparse Row Format (csr)
  - Sparse Skyline Format
- Batched CuPy Implementation
- Custom GPU Kernels
  - Writing custom fine-grain kernels to handle independent operations w/in the K-assembly part of the algorithm

Backups