Multi-GPU Acceleration for Finite Element Analysis in Structural Mechanics

1. Introduction

The continuous increase in model complexity and fidelity in engineering simulations has made computational efficiency a central concern in modern scientific computing. Among numerical techniques for solving boundary-value problems governed by partial
differential equations, the Finite Element Method (FEM) stands out as one of the most
widely adopted and versatile approaches, particularly in the context of structural mechanics. FEM enables the computation of displacement fields, strains, and stresses in complex geometries under realistic loading and boundary conditions, playing a fundamental role in engineering design and analysis.

Despite its flexibility, FEM is computationally demanding. As mesh resolution increases
or parametric studies are introduced, the size of the resulting algebraic systems grows
rapidly, often reaching millions of degrees of freedom. In such scenarios, traditional CPU-based implementations face scalability limitations, especially during the assembly of large sparse matrices and the iterative solution of linear systems. These challenges motivate the exploration of alternative execution models capable of exploiting parallelism more effectively.

Recent advances in Graphics Processing Units (GPUs) have significantly expanded the landscape of high-performance computing. GPUs provide massive data-parallel throughput and high memory bandwidth, making them particularly suitable for linear algebra workloads that dominate FEM solvers. Furthermore, modern Python-based acceleration frameworks such as NumPy, Numba, and CuPy allow developers to

target heterogeneous architectures with relatively high productivity while retaining access to low-level optimisation mechanisms.
In this context, this project proposes the development of a GPU-accelerated Finite Element Analysis solver for structural mechanics, progressing incrementally from a CPU baseline to parallel CPU and GPU implementations. The work aims to explore execution models, parallelisation strategies, and performance evaluation techniques in
accordance with the objectives of the High Performance Graphical Computing curricular unit.

2. The Finite Element Method (FEM)

2.1 Definition and Core Principles

The Finite Element Method is a numerical discretisation technique used to approximate
the solution of partial differential equations over complex domains. The method subdivides a continuous domain into a finite number of smaller subdomains, called elements, which are interconnected at nodes. Within each element, the unknown field variables are approximated using polynomial shape functions.

By applying a variational (weak) formulation, the governing equations are transformed
into a global system of algebraic equations. In linear elastic structural problems, this process typically results in a sparse system of the form:

$$\mathbf{Ku} = \mathbf{f}$$

where $\mathbf{K}$ is the global stiffness matrix, $\mathbf{u}$ the vector of nodal unknowns, and $\mathbf{f}$ represents external forces. The stiffness matrix is generally sparse, symmetric, and positive definite, properties that strongly influence the choice of numerical solvers and
optimisation strategies.

2.2 Importance and Applications of FEM

FEM is a cornerstone of modern engineering analysis due to its ability to handle complex geometries, heterogeneous material properties, and realistic boundary conditions. It is widely used in civil, mechanical, aerospace, and automotive engineering for tasks such as structural verification, fatigue analysis, vibration studies,
and virtual prototyping.
The ability to perform accurate simulations prior to physical testing reduces development costs and improves safety and reliability. However, these benefits come at the cost of increased computational demand, particularly for large-scale or high¬ resolution problems, making FEM an ideal candidate for high-performance and parallel
computing techniques.

2.3. Types of Problems Addressed by FEM

The Finite Element Method supports a wide range of physical problems, including:

- Linear static structural analysis;
- Dynamic and modal analysis;
- Heat transfer and diffusion problems;
- Coupled thermo-mechanical and multi-physics problems;
- Nonlinear material and geometric analyses.

Within the scope of this project, the focus is placed on linear elastic structural problems, which provide a mathematically well-posed and computationally controlled setting for evaluating parallel execution and performance optimisation strategies while
remaining representative of real engineering applications.

3. Computational Motivation for Parallelism and GPU Acceleration

From a computational perspective, FEM workloads are dominated by linear algebra operations, particularly sparse matrix-vector multiplications (SpMV) and vector updates performed repeatedly within iterative solvers such as the Conjugate Gradient (CG) method. These operations exhibit a high degree of data parallelism, as the same
arithmetic operations are applied independently to large sets of data.
GPUs follow the SIMT (Single Instruction, Multiple Threads) execution model, enabling thousands of lightweight threads to execute identical instructions concurrently on different data elements. This execution paradigm makes GPUs especially effective for accelerating the core kernels of FEM solvers. In addition, multi-core CPUs and modern threading models provide opportunities for parallelism at the
CPU level, offering a progressive optimisation pathway.
Beyond single-device acceleration, multi-GPU execution enables further scalability, particularly for workloads involving multiple independent simulations or parameter sweeps. Frameworks such as Dask-CUDA facilitate the distribution of computational tasks across multiple GPUs, abstracting device management and scheduling while enabling scalable execution.

4. Project Objectives

4.1 General Objective

The primary objective of this project is to design, implement, and evaluate a GPU-accelerated Finite Element Analysis solver for structural mechanics, assessing performance improvements and scalability relative to a CPU-based baseline.

4.2 Specific Objectives

The specific objectives of the project are:

1. To implement a correct and complete CPU baseline FEM solver using Python and NumPy/SciPy, serving as a reference for correctness and performance comparison.
2. To analyse the FEM computational pipeline and identify performance bottlenecks, particularly in sparse linear system solvers.
3. To introduce CPU-level optimisations using Numba JIT compilation and parallel execution where appropriate.
4. To develop a single-GPU accelerated implementation using CuPy and GPU-enabled sparse linear algebra routines.
5. To explore multi-GPU execution strategies, when feasible, using Dask-CUDA to distribute independent simulation workloads.
6. To perform a systematic performance evaluation, comparing CPU, single-GPU, and multi-GPU implementations.
7. To deliver an interactive demonstration application that visualises simulation results and performance metrics.

5 Technologies and Execution Models

The project will explore a progressive set of execution models, subject to feasibility within the course timeline:

Implementation	**Technology**	**Scope**
Pure NumPy/SciPy	Sequential Python	Baseline reference
Numba JIT	LLVM JIT compilation	Single-core optimisation

Implementation	Technology	Scope
Numba Parallel	Multi-threaded CPU execution	Multi-core CPU
CuPy GPU (basic)	CUDA-accelerated arrays	Direct GPU port
CuPy GPU (optimised)	Vectorised GPU kernels	Optimised GPU execution

This progression reflects the concepts covered in Modules 1 and 2 of the course, including concurrency, parallelism, and CUDA execution models.

6. Planned Performance Evaluation

To ensure a rigorous and transparent evaluation, the following metrics are planned, subject to feasibility:

Performance Metrics

- Element assembly time;
- Boundary condition application time;
- Solver time (total and per iteration);
- End-to-end execution time;
- Peak RAM and VRAM usage;
- Number of iterations to convergence.

Scalability Metrics

- Weak scaling (execution time versus mesh size);
- Strong scaling (execution time versus number of CPU cores or GPUs);
- Time per degree of freedom;
- Time per solver iteration.

Efficiency Metrics

- Speedup relative to the NumPy baseline;
- Parallel efficiency for multi-core and GPU execution;
- Energy consumption estimates (when available);
- Cost efficiency (performance relative to hardware cost).

These metrics align with best practices in performance analysis and with the evaluation criteria of the course.

7. Case Study and Problem Definition

7.1 Selection of the Benchmark Problem

The development and evaluation of the GPU-accelerated FEM solver requires a computationally intensive benchmark problem that is mathematically well-posed and representative of engineering applications. Initial consideration was given to simple two-dimensional linear elasticity or Poisson-type problems defined over structured or unstructured meshes.

This case study serves as the primary reference and may be adapted based on the measured performance and results obtained during the later stages of this project.

7.2 Detailed Case Study: Potential Flow in a Channel with Geometric Constrictions

7.2.1 Physical Problem Description

The specific case study analysed in this work consists of a two-dimensional potential flow of an incompressible and inviscid fluid inside a channel with geometric constrictions formed by circular obstacles. The channel is assumed to be infinite in the out-of-plane direction, allowing for a 2D formulation.

A uniform inflow velocity is imposed at the inlet, and the objective is to determine:

- the velocity potential distribution,

- the velocity field,
- the pressure distribution,
- the maximum and minimum values of velocity and pressure and their locations.

This configuration is representative of internal flows in ducts and channels and provides a demanding benchmark due to the presence of strong velocity gradients near the constrictions.

7.2.2 Mathematical Formulation

The flow is described using the velocity potential $\phi(x, y)$, such that:

$$u = \frac{\partial \phi}{\partial x}, v = \frac{\partial \phi}{\partial y}$$

Assuming incompressibility and irrotational flow, the governing equation reduces to the Laplace equation:

$$\nabla^2 \phi = 0 \quad \text{in} \quad \Omega$$

The spatial discretisation is performed using the Galerkin finite element method, leading to a linear system:

$$K\phi = F$$

7.2.3 Boundary Conditions

The problem involves mixed boundary conditions which are necessary to guarantee a unique and physically meaningful solution for the Laplace equation.

- **Inlet Boundary (Neumann)**
  At the inlet section of the channel, a uniform normal velocity is imposed:

$$\frac{\partial \phi}{\partial n} = V_0 = 2.5\,\mathrm{m/s}$$

This corresponds to a natural (Neumann) boundary condition, prescribing the flux (velocity) across the boundary.

- **Outlet Boundary (Dirichlet)**
  At the outlet sections, the velocity potential is set to zero:

$$\phi = 0$$

  This essential (Dirichlet) boundary condition defines a reference level for the potential, effectively grounding a degree of freedom necessary for convergence, since the Laplace equation determines the potential only up to an additive constant.

- **Wall Boundaries**
  Along all solid boundaries of the domain, including the circular constrictions, a no-penetration condition is enforced. This states that the normal component of the velocity vanishes at the walls, which in terms of the velocity potential can be written as:

$$\frac{\partial \phi}{\partial n} = 0$$

  This boundary condition guarantees that the fluid remains confined within the channel and that no mass flux occurs across the solid surfaces. This condition is crucial for accurately modelling internal flows.

7.2.4 Mesh Refinement Strategy

The accuracy of the Finite Element Method is critically dependent on the quality and resolution of the computational mesh. In potential flow problems, regions where the velocity changes rapidly (high-gradient regions) must be resolved with smaller elements to prevent excessive numerical diffusion and ensure precise results.

To accurately capture high-gradient regions near the geometric constrictions, a locally refined mesh is employed in the central region of the channel, while coarser elements are used elsewhere. This targeted refinement strategy ensures high accuracy where it is most needed, without incurring the excessive computational cost of a uniformly fine mesh across todo o domínio.

This refinement strategy, however, drastically increases the computational load:

- **Matrix Size and Sparsity:** Mesh refinement leads to a significant increase in the number of degrees of freedom (DOFs), resulting in a larger global stiffness matrix (K). While the matrix remains sparse, the increased size directly impacts the memory footprint and the time required for both matrix assembly and iterative solution.
- **Benchmark Suitability:** The necessity of handling large, refined meshes makes this case particularly suitable for evaluating the performance and scalability of GPU and multi-GPU FEM implementations. Specifically, it tests the ability of the acceleration frameworks (Numba and CuPy) to manage high-dimensional sparse linear algebra operations efficiently.