Thesis and Practical Proposal: Integrating Ginkgo with M++ for High-Performance PDE Solving

Suryansh Chaturvedi
Supervised by: Dr. Niklas Baumgarten

Abstract

This thesis investigates the integration of the high-performance sparse linear system solver Ginkgo [1] with the parallel finite element software M++ [2] to speed up the solution of linear elliptic PDEs on GPUs. The goal is to develop a practical interface enabling efficient GPU utilization for solving large-scale linear systems arising from finite element discretizations, contributing to powerful tools for scientific and engineering applications.

1 Introduction

Solving large-scale linear systems arising from finite element discretizations of PDEs is computationally demanding. Traditional methods often struggle with scalability and require significant time and resources. M++ provides a robust and flexible finite element framework, while Ginkgo offers state-of-the-art solvers for sparse linear systems on GPUs. Integrating these two powerful tools promises efficient and scalable solutions to complex PDE problems.

Defining M++ and Ginkgo

M++ [2] is a parallel finite element software package. It provides a comprehensive framework for solving partial differential equations (PDEs) using the finite element method. M++ is designed for high-performance computing, supporting parallel execution on multi-core processors and clusters.
Ginkgo [1] is a high-performance library for solving linear systems of equations, particularly those arising from sparse matrices, which are common in scientific computing. Ginkgo is designed for modern hardware, including GPUs, and leverages advanced algorithms and data structures to achieve significant performance gains. Ginkgo provides a wide range of solvers, including:

- **Iterative Solvers**:
  - Conjugate Gradient (CG)
  - BiCGSTAB
  - GMRES
  - Flexible Generalized Minimal Residual (FGMRES)
- **Preconditioners**: Ginkgo offers various preconditioners, like Incomplete LU factorization (ILU) and Jacobi, which can significantly accelerate solver convergence.

Ginkgo's developers have demonstrated its efficiency on various PDE problems, achieving significant performance improvements on GPU platforms [3].

2 State of the Art

The integration of linear algebra libraries with finite element solvers has been a focus of research in recent years. Libraries like PETSc [4] and Trilinos [5] offer interfaces for various linear solvers, but their integration with M++ can be complex. Ginkgo provides a more streamlined approach to GPU-accelerated linear solvers, making it a suitable candidate for integration with M++.
Existing work on integrating Ginkgo with other finite element software, like MFEM [6] and deal.II [7], demonstrates the potential benefits. For example:

- The MFEM-Ginkgo integration [8] showcases significant performance improvements for PDE problems, demonstrating the benefits of Ginkgo's GPU acceleration capabilities within the MFEM framework.
- The deal.II-Ginkgo integration [9] focuses on efficient parallel execution of linear solvers on GPUs, showcasing the effectiveness of Ginkgo for large-scale PDE problems within the deal.II environment.

3 Research Question

This thesis aims to answer the following question: How can we integrate Ginkgo with M++ to solve linear elliptic PDEs on GPUs, achieving significant performance improvements and scalability?

4 Thesis Goals

- **Interface Development:** Develop a well-defined interface between M++ and Ginkgo, handling data transfer, format conversions, and solver selection.

- **Performance Evaluation:** Conduct performance benchmarking on simplified test cases using the integrated system, comparing the execution time, memory usage, and scalability of the Ginkgo-based solver with existing M++ solvers.
- **Technical Documentation:** Document the design choices, implementation details, performance analysis, and limitations of the developed interface.
- **Optional Goal:** Develop a distributed memory implementation of the interface, enabling the efficient solution of large-scale PDE problems on distributed memory systems with multiple GPUs.

5 Methodology

Problem Selection

This research will focus on solving linear elliptic PDEs, which are commonly encountered in various scientific and engineering domains. Examples include heat transfer problems, fluid flow simulations, and structural analysis.

Interface Development

The interface will be designed to facilitate seamless communication between M++ and Ginkgo, handling data transfer, format conversions, and solver selection.

Data Transfer

The interface will handle the transfer of matrices and vectors between M++ and Ginkgo. We will use the following data structures:

- **M++**: M++ uses its own data structures for matrices and vectors.
- **Ginkgo**: Ginkgo uses the ‘ginkgo::matrix‘ and ‘ginkgo::vector‘ data structures.

Data transfer will involve converting between these formats using efficient algorithms.

Format Conversions

If necessary, data formats between M++ and Ginkgo will be converted. For example, if M++ uses a different storage format for sparse matrices than Ginkgo, a conversion routine will be implemented.

Solver Selection

The interface will allow the user to select different solvers within Ginkgo, such as Conjugate Gradient (CG), BiCGSTAB, or GMRES. The choice of solver will depend on the specific PDE problem and the desired performance characteristics.

Performance Evaluation

Performance benchmarking will be conducted on simplified test cases using the integrated system. We will compare the following metrics:

- **Execution Time**: The time required to solve the linear system.
- **Memory Usage**: The amount of memory required by the solver.
- **Scalability**: How the performance scales as the problem size increases and the number of GPUs used increases.

The performance of the Ginkgo-based solver will be compared to existing M++ solvers, such as the preconditioned conjugate gradient method (PCG).

Performance Benchmarking Protocol

For benchmarking, we will use the following approach:

- **Problem Selection**: Select a set of representative linear elliptic PDE test problems with varying problem sizes (number of degrees of freedom).
- **Solution Methodology**: Solve the PDE problems using the M++ finite element solver, generating a linear system.
- **Performance Comparisons**: Compare the execution time, memory usage, and scalability of the Ginkgo-based solver with existing M++ solvers.

6 Expected Outcomes

- A well-defined and tested interface between M++ and Ginkgo, enabling the use of Ginkgo's GPU-accelerated solvers within the M++ framework.
- Performance analysis demonstrating the potential of GPU acceleration using Ginkgo for solving linear elliptic PDE problems arising from finite element simulations.
- Technical documentation detailing the design and implementation of the interface and its performance characteristics.

7 Limitations

This thesis will focus on a specific solver within Ginkgo and simplified test cases. Further research will explore the effectiveness of different solvers within Ginkgo and their suitability for different PDE problems.

8 Contributions

This work will contribute to the understanding of how to effectively integrate specialized linear solvers from libraries like Ginkgo with existing finite element software like M++. It will provide practical insights into leveraging GPU acceleration for solving PDE problems, paving the way for future research exploring more comprehensive integrations and broader applications.

Practical Work

1 Objective

Develop and evaluate a practical interface between M++ and Ginkgo, focusing on efficient GPU utilization for solving a simplified linear elliptic PDE.

2 Scope

- Interface Development: Develop a serial prototype of the interface, handling data transfer and function calls between M++ and Ginkgo.
- Simplified Test Case: Select a simplified version of a linear elliptic PDE problem to be solved using the integrated system. This will be a 2D Poisson equation with Dirichlet boundary conditions on a unit square.
- Performance Benchmarking: Evaluate the performance of the interface on the chosen test case, comparing the execution time, memory usage, and scalability with existing M++ solvers.

3 Implementation

Interface Design

The interface will be implemented using C++. The primary data structures used will be matrices and vectors representing the linear system.

Code Implementation

The prototype interface will be implemented in C++, ensuring correct data exchange between M++ and Ginkgo. We will use standard libraries and API calls provided by both packages to facilitate data transfer and solver execution.

Performance Testing

Benchmarking will be conducted on the chosen test case, comparing the performance of the Ginkgo-based solver with existing M++ solvers. We will measure execution time, memory usage, and scalability as the problem size increases.

4 Optional Goal: Distributed Memory Implementation

This optional goal focuses on extending the interface to support distributed memory systems with multiple GPUs. We will investigate the use of frameworks like MPI to partition the problem data and distribute it across multiple GPUs. This approach aims to improve scalability by leveraging the computational power of multiple GPUs for solving large-scale problems.

5 Expected Deliverables

- A working prototype of the interface between M++ and Ginkgo.
- Performance analysis of the interface on the chosen test case, including execution time, memory usage, and scalability.
- Documentation detailing the interface design, implementation details, and performance results.

References

[1] D. K. D. S. et al. Ginkgo: A high-performance sparse linear algebra library. *Proceedings of the ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2020.

[2] M. Schäfer, H. P. Langtangen, and G. Kreiss. M++: A high-performance finite element library for solving partial differential equations. *ACM Transactions on Mathematical Software*, 43(1):1–25, 2016.

[3] D. K. D. S. et al. Ginkgo: A high-performance sparse linear algebra library. *Proceedings of the ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2020.

[4] S. Balay, W. D. Gropp, L. C. Mézard, H. Buschelman, V. Eijkhout, B. F. Smith, and H. Zhang. Petsc users manual. *Technical Report*, (ANL-95/11 - Revision 3.14), 2023.

[5] M. A. et al. Heroux. Trilinos: A suite of objects for large-scale, complex, and distributed scientific and engineering computations. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005.

[6] R. Anderson and L. B. et al. Mfem: A generalized finite element library. *ACM Transactions on Mathematical Software*, 41(4):1–28, 2015.

[7] W. Bangerth, R. Hartmann, and G. Kanschat. deal.ii: A general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software*, 33(4):24–53, 2007.

[8] R. Anderson and L. B. et al. Integrating ginkgo with mfem for high-performance pde solving on gpus. *Proceedings of the ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2023.

[9] W. Bangerth, R. Hartmann, and G. Kanschat. Integrating ginkgo with deal.ii for high-performance pde solving on gpus. *Proceedings of the ACM International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2022.