Parallelization and Optimization of Adaptive Mesh Refinement using OpenMPI

Group Members:

Xingyuan Wang (xingyuaw@andrew.cmu.edu) Hangrui Cao (hangruic@andrew.cmu.edu)

Project Website: Link

Summary

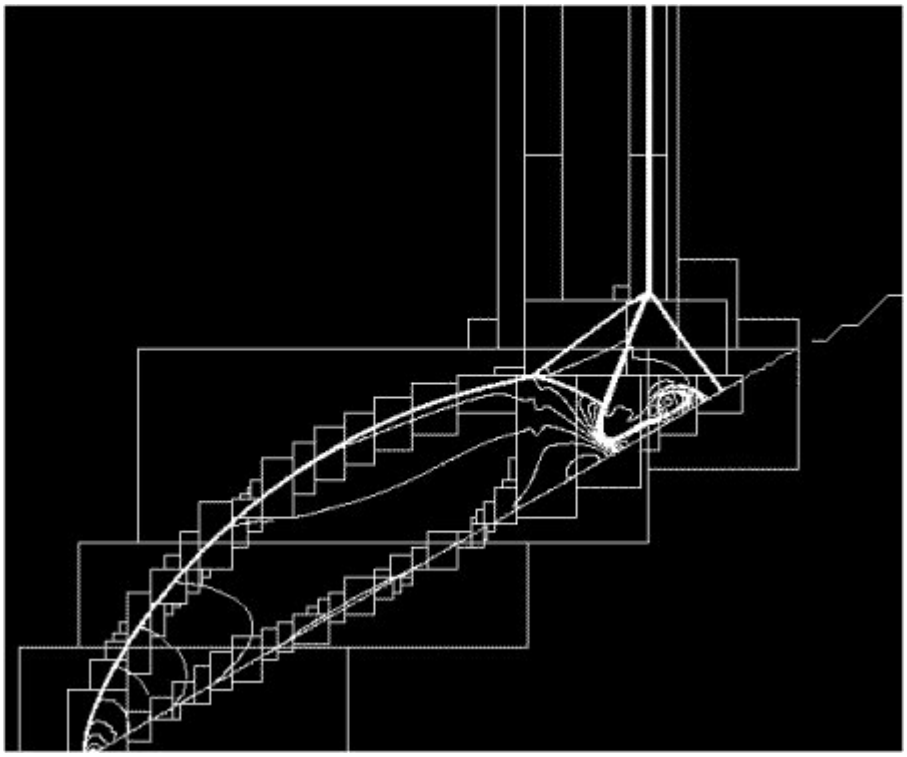
We aim to parallelize and optimize an Adaptive Mesh Refinement (AMR) code using OpenMPI that will run on GHC and PSC machines. The project involves parallelizing an existing serial base code to leverage parallel processing, exploring techniques to reduce work imbalance, communication and synchronization overhead, thus enhancing its performance and scalability.

Background

Adaptive Mesh Refinement (AMR) is a numerical technique used in computational simulations to dynamically adjust the resolution of a computational grid based on the features of the solution being computed. In many scientific and engineering applications, such as fluid dynamics, astrophysics, and structural analysis, certain regions of interest may require higher resolution to capture complex phenomena accurately, while other regions can suffice with coarser resolution to reduce computational cost.

AMR algorithms typically work by initially defining a coarse grid covering the entire domain of interest. As the simulation progresses, the algorithm identifies regions where higher resolution is needed and refines the grid locally in those regions, creating finer grids known as refinement patches or refinement levels. This adaptive refinement process allows computational resources to be concentrated where they are most needed, leading to more accurate results with less computational expense.

The following figure from Wikipedia shows the grid structure of an example AMR calculation. Each of the boxes is a grid; the more boxes it is nested within, the higher the level of refinements.



Application Description

The application we are focusing on is an implementation of an AMR algorithm for simulating physical phenomena, such as fluid flow or structural deformation, in a computational domain. The algorithm consists of several key components:

- **Grid Initialization:** The simulation domain is discretized into a coarse grid, representing the entire computational domain.
- **Time Integration:** The simulation advances in time using numerical integration techniques. At each time step, the solution is computed on the current grid configuration.
- **Refinement Criteria:** Based on predefined criteria, regions requiring higher resolution are identified. These regions are candidates for refinement.
- **Grid Refinement:** Refinement patches or refinement levels are added to the grid in regions identified by the refinement criteria. These patches have finer resolution compared to the coarse grid.
- **Data Transfer:** Data from the coarse grid are interpolated onto the refinement patches.
- **Simulation on Refined Grid:** The simulation continues on the refined grid, capturing fine-scale features with higher accuracy.
- **Stencil Operations:** Stencil operations are applied to the grid for solving partial differential equations governing some certain physical phenomena.

Things for Parallelism

Several aspects of the AMR algorithm can benefit from parallelism.

- **Interpolation and Stencil Operations:** Both interpolation from the coarse grid to refinement patches and stencil operations involve computationally intensive operations that might be parallelized efficiently. These operations are typically performed independently on different grid points or patches, making them suitable for parallel execution.
- **Load Balancing:** One of our major focus is to allow the workload to be distributed evenly across multiple processing units, ensuring efficient utilization of computational resources. Load balancing

becomes essential, especially in AMR algorithms where the workload varies dynamically as refinement patches are added or removed.
- **Scalability:** Parallelizing the AMR algorithm enables scaling to larger problem sizes and utilizing multi-core processors effectively. By distributing the workload across multiple cores, the computational time can be significantly reduced, leading to faster simulations and increased productivity.

Challenges

- **Workload Characteristics:** Stencil computations have a high degree of spatial locality but may face challenges in terms of load imbalance due to the adaptive nature of AMR. Dependencies between grid points also introduce synchronization overhead in parallel environments.
- **System Constraints**: Mapping the AMR-enabled stencil computations to parallel architectures poses challenges, including efficiently distributing the dynamically changing workload across processors and minimizing communication overhead between them.
- **What We Hope to Learn**: We aim to explore strategies for dynamic/semi-static load balancing, effective parallelization of AMR processes, and optimization of memory access patterns in stencil computations to achieve high performance on parallel systems.

Resources

- **Computing Resources**:

Access to PSC machines and the university's GHC cluster with multi-core CPUs.

- **Starter Code:**

We will start from an existing open-source AMR implementations (which include the serial implementation version), extending and optimizing it for our needs. Link

- **Reference Materials:**

Berger, M. J., and Oliger, J. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 1984.

Diachin, Lori A., Richard D. Hornung, Paul E. Plassmann, and Andrew M. Wissink. "Parallel Adaptive Mesh Refinement." In *Parallel Processing for Scientific Computing*, 2005. Link

We will base our AMR strategy primarily on the principles outlined in the aforementioned references. Additionally, we will look for and incorporate ideas from new studies on parallel AMR.

Goals and Deliverables

- **Parallelization with OpenMPI:** Parallelize the existing serial AMR codebase using OpenMPI to exploit distributed-memory parallelism. This involves identifying parallelizable tasks, such as stencil operations and data transfer, and implementing them using appropriate parallelization constructs.
- **Optimization for Load Balancing:** Develop strategies for dynamic/semi-static load balancing to distribute the computational workload evenly across processing units. This includes investigating

techniques such as task stealing, workload partitioning, or dynamic scheduling to minimize idle time and improve overall efficiency.

- **Reduction of Communication Overhead:** Minimize communication overhead between processing units by optimizing data transfer routines and message passing patterns. This involves reducing unnecessary data exchanges, overlapping communication with computation, and utilizing non-blocking communication primitives to improve parallel efficiency.
- **Performance Analysis and Profiling:** Conduct thorough performance analysis and profiling to identify bottlenecks and areas for optimization. Measure performance metrics such as execution time, speedup, scalability, cache utilization, and memory access patterns to evaluate the effectiveness of parallelization and optimization efforts.

Hope to Achieve:

- **4x Speedup on AMR-Enabled Stencil Computations on GHC machine with 8 cores:** Aim to achieve at least a 4x speedup on AMR-enabled stencil computations compared to uniform grid computations for selected Partial Differential Equations (PDEs). This target is based on the expected parallel efficiency and the computational resources available.
- **Deep System Performance Analysis:** Perform in-depth analysis of system performance under various configurations, including different problem sizes, core counts, and communication patterns. Evaluate performance metrics to understand the impact of parallelization strategies on overall system behavior.

Demo:

During the poster session, we plan to demonstrate the following:

- **Live Simulation Comparison:** Show a live simulation of fluid flow around an obstacle using the parallelized AMR code and compare it with simulations using uniform grid computations. Highlight the differences in accuracy and performance achieved with AMR.
- **Speedup Graphs and Resource Utilization Metrics:** Present speedup graphs demonstrating the performance improvement achieved through parallelization and optimization. Showcase resource utilization metrics such as CPU utilization, memory bandwidth, and cache misses to illustrate the efficiency of the parallel implementation.

Platform Choice

Since we choose to parallelize a piece of code designed for multi-core machines, early-stage tests will be conducted on GHC machines, while final measurements will include results from PSC machines to examine the implementation's scalability under a large number of cores. We temporarily choose OpenMPI as our implementation framework.

Schedule

Week	Dates	Tasks

Week	Dates	Tasks
0	March 25 – March 31	Literature review and finalization of the high-level project design.
1	April 1 – April 7	Basic implementation of the AMR framework. Understand the baseline code.
2	April 8 – April 14	Integration of the AMR framework with MPI. Basic parallelism finished with test results on GHC machines.
3	April 15 – April 21	Testing, validation, and benchmarking against uniform grid solutions. Optimize work balance and reduce communication overhead.
4	April 22 – April 28	Keep optimizing the solution, but with a focus on reducing lock and contention. Obtain experimental results from PSC machines.
5	April 29 – May 5	Finish up comparison analysis, final report, and poster preparation.

This schedule allows us to allocate time efficiently, considering the project's complexity and our academic workload. We plan to reassess our progress weekly to ensure we stay on track to meet our goals.