Karlsruhe Institute of Technology

Efficient parallel C++
Final Project Presentation

Online Linear Equation Solving

Robert Andreas Fritsch `robert.fritsch@student.kit.edu` | February 27, 2023

Outline

Project Overview: Components

- **Solvers:**
  - Gaussian elimination (sequential)
  - LU factorization (both sequential and parallel implementations)
- **Thread Pool:**
  - Constructed using multiple layers of queues
- **Scheduler:**
  - Trivial (processes all problems sequentially)
  - Parallel (processes all problems in parallel)
  - Mixed (selects strategy based on problem size and load)
- **Benchmarking and Tests:**
  - Comprehensive tests and timing routines (via `bench.hpp`) to assess performance.

Thread Pool

- **Task Queue Design:**
  1. A per-thread lock-free static (SPMC) queue for local subtasks.
  2. A shared lock-free static (SPMC) queue for unassigned tasks.
  3. A shared locked (MPMC) overflow queue that also allows threads to wait.
- **Overhead:**
  - Each worker logs binary timestamps for the start and end of tasks into a thread-local file. The additional overhead is minimal and does not significantly affect benchmark results.
- **Implications:**
  - Under heavy load, subtasks are processed locally by each thread.
  - The design permits threads to await or sleep when no tasks are available.

Online Scheduler: Strategies and Implications

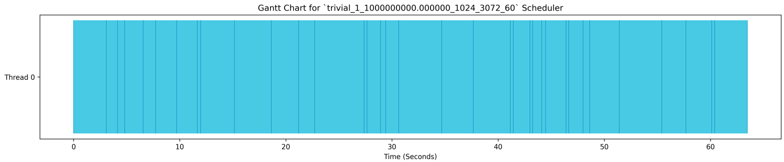
- **Policy Variants:**
  - **Trivial Scheduler:** Enqueues every problem to be solved with the sequential LU solver.

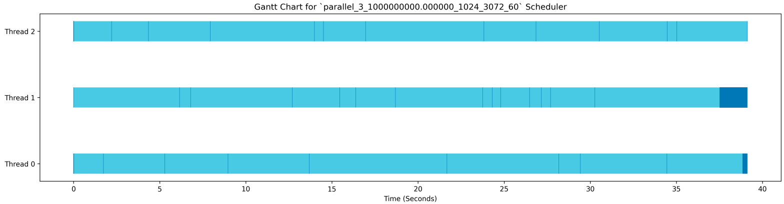
    When problems arrive with significant latency, many threads remain idle.
  - **Parallel Scheduler:** Enqueues every problem to be solved with the parallel LU solver.

    For small problems, the parallel LU solver is inefficient. In the long run, the trivial scheduler is more effective.
  - **Mixed Scheduler:** Dynamically selects between the sequential and parallel LU solvers based on problem size and current load.

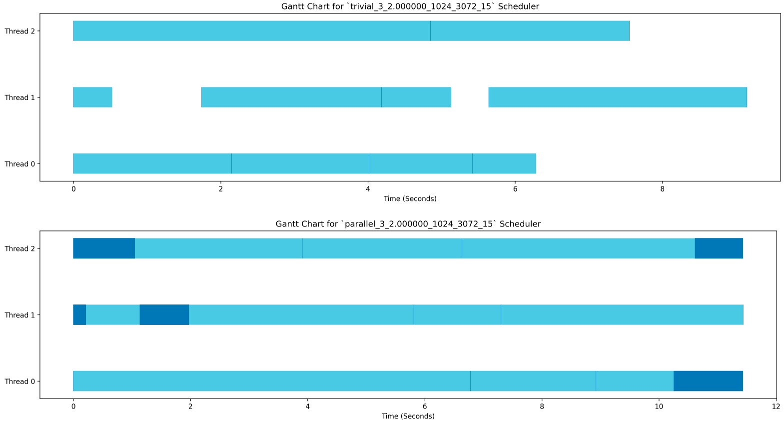
    It achieves near-parallel performance for the problem cases considered. In theory, it should combine the advantages of both trivial and parallel scheduling, although this has not yet been conclusively demonstrated.

Benchmarking I

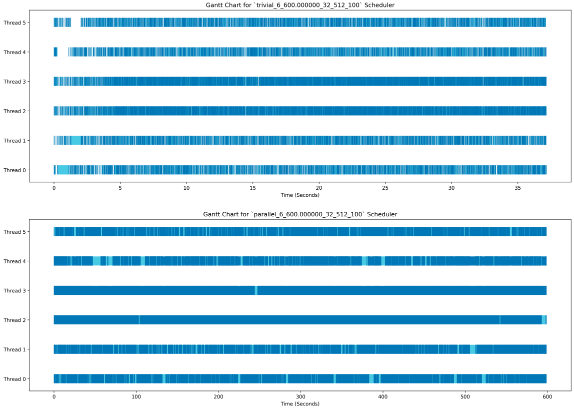




Benchmarking II

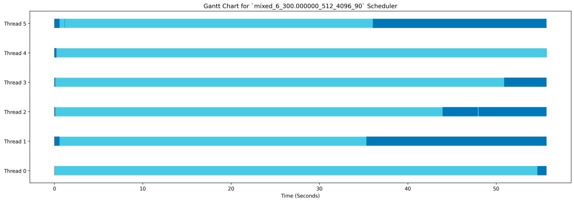


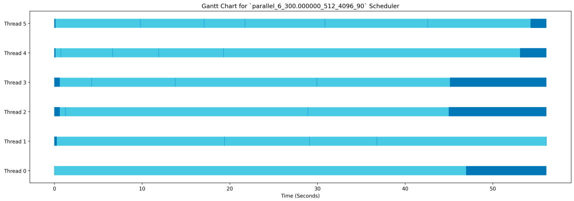
Benchmarking III

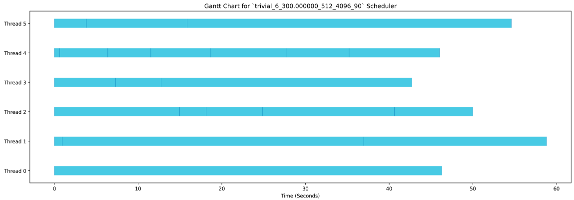


Karlsruhe Institute of Technology

Benchmarking IV







Conclusion and Outlook

Karlsruhe Institute of Technology

- **Summary:**
  - An efficient, low-overhead thread pool with logging enables the creation of detailed Gantt diagrams.
  - The multi-layered queue design minimizes contention while providing robust fallback mechanisms.
  - The scheduling strategies (Trivial, Parallel, Mixed) all perform well; however, in most cases, trivial scheduling proves to be the most efficient.
- **Future Work:**
  - Extend benchmarks across a broader range of hardware configurations.