15618 Project Proposal: Cache Hierarchy-Aware Task Scheduler On Multicore Architectures

Team members: Calvin Lin, Yen-Shou Su

URL

https://github.com/calvinfornialin/CMU-15618-Final-Project.git

Summary

- This project aims to design and implement a cache hierarchy-aware task scheduling system for multicore architectures. The goal is to optimize task execution on parallel systems by leveraging knowledge of the cache hierarchy to reduce cache misses and improve cache utilization and overall system performance.

Background

- Many modern applications, particularly those in scientific computing, data analytics, and machine learning, are compute-intensive and can benefit significantly from parallel execution. As learned in lectures, optimal performance on multicore systems requires careful consideration of how tasks are scheduled among multiple cores or processors, several crucial factors that we should consider are data locality, workload balance, and synchronization/communication time. However, the static or dynamic scheduling approaches introduced in lectures have neglected the cache hierarchy factor. So in this project we will propose a novel scheduling approach that takes not only data locality and workload balance but also cache hierarchy into account and focus on a compute-intensive

component that involves frequent data accesses where parallelism can be exploited through task partitioning and scheduling based on cache topology and behavior.

- In OpenMP, the type of scheduling scheme that is utilized is basically one of three kinds, static, dynamic, and guided, which is pre-specified in the program during compile time, or is determined by the system/compiler through the auto scheduling type or is deferred until runtime through the runtime scheduling type. No matter which scheduling type is chosen, the scheduler follows the same scheme throughout the parallelization, which may pose a restriction on more complicated tasks that can benefit from the switching of scheduling schemes instead of following a particular scheduling scheme from the beginning all the way until the end.
- The proposed scheduling scheme must be able to adapt to various underlying cache topologies and maintain the performance over several representing compute-intensive test benches:

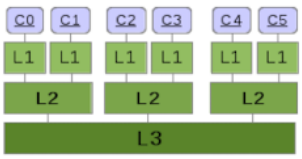

Fig. 1. Intel Dunnington

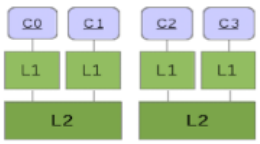

Fig. 2. Intel "Hapertown"

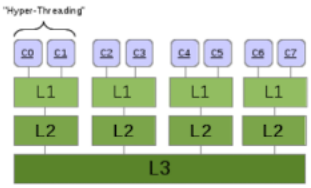

Fig. 3. Intel "Nehalem"

Challenges

- Correctly reflecting what we learned about task scheduling from lecture in the actual implementation requires firm understanding of crucial metrics and tradeoffs between different scheduling approaches.
- Understanding the OpenMP open source code and know how to modify it to meet our needs.
- The proposed scheduling scheme should not incur too much overhead in determining if the program can benefit from a switch in scheduling scheme since there will be no point in doing so if the overhead resulted from this flexibility in switching scheduling schemes surpasses the benefits it can bring. We anticipate that the proposed scheduling scheme should have a similar performance for tasks that are optimal when the scheduling scheme is set to be static, dynamic, or guided. For more complicated tasks, we anticipate the benefits brought by scheduling scheme switching can surpass the overheads incurred and result in a

boost of performance. Thus, the determination for scheduling scheme switching should not be too complicated or require too much synchronization in order to lower the incurred overheads.

Workload Constraints:

- **Parameters of underlying hardware:** Comparing to other conventional task schedulers, our design requires two additional steps. First, we need to retrieve the parameters of the underlying machine's cache, such as cache topology (how many levels of caches, and how are the multiple cores grouped to share specific levels of caches) before scheduling tasks among cores. Second, we need to dynamically monitor the metrics of cache performance (eg. number of cache misses, and cache latency) as a criterion to switch between several scheduling schemes.
- **Dependencies**: Some tasks might depend on the completion of others, which requires careful scheduling to avoid deadlocks and minimize synchronization/communication times.

System Constraints:

- Mapping tasks effectively to a multicore architecture considering the data locality and workload balance is already challenging, considering cache hierarchy and core interconnect bandwidth is even more difficult.
- Just like the Barnes-Hut algorithm and the ocean simulation introduced in lectures, performance of the cache (i.e. miss rate, memory access pattern) might vary a lot. So our approach should be in somewhat a higher level to adapt general cases and avoid overfitting to a specific case. Our goal is to make the common case faster, especially more complicated tasks where the workloads can not be easily partitioned in such a way that each processor works on a similar amount of work.

Resources

Machine used:

- We plan to develop the scheduling approach on local machines, and then run tests and benchmarks on the GHC and PSC machines. One reason of using both the GHC and PSC

machine is because we want study both the effect of number of cores on the scalability and different cache topologies on the performance.

Starting framework:

- We'll start implement our cache hierarchy-aware task scheduler by modifying the existing open source OpenMP scheduler code.

Reference paper:

- Nader Khammassi, Jean-Christophe Le Lann, "Design and Implementation of a Cache Hierarchy-Aware Task Scheduling for Parallel Loops on Multicore Architectures"

Goals and Deliverables

Plan to Achieve

- Implement a cache hierarchy-aware task scheduler that considers cache hierarchy for task execution.
- Extend the scheduler to make it capable of dynamically switching between task distribution strategies like static, dynamic, and cache-aware schedulers on-the-fly based on real-time cache usage and observed behavior.
- Demonstrate improved cache hit rates and reduced execution times for several representing compute-intensive benchmarks like LINPACK, Graph500, and SPEC. Also comparing the cache performance to some standard schedulers like static, dynamic scheduler, and task-stealing to achieve at least 10% of improvement.

Hope to Achieve

- If ahead of schedule, we'd like to explore factors beside cache hierarchy that might also affect task scheduling performance metrics and include that in our scheduler.
- If ahead of schedule, we'd like to explore how to improve the task scheduling performance metrics of our scheduler on more test cases and situations.

Demo

- An interactive demonstration showcasing the performance improvements in task execution times and cache utilization, compared to non-optimized scheduling. Speedup graphs and cache hit/miss statistics will be presented.

Analysis Goals

- There are several variables that we want to know how they can impact the performance of task scheduling:
  - Understand how task granularity influence task performance metrics.
  - Understand how including cache topology factor can influence task scheduling performance metrics.
  - Understand how CPU core number influences task scheduling performance metrics and to identify the optimal task granularity for maximizing cache utilization.

Platform Choice

- We will be doing most of our development and testing locally and on the GHC machines. As for the scalability study, we will be using PSC machine. Using these systems will ensure that our proposed solutions can adapt to various different cache hierarchies and scale to different numbers of processors.
- We will be using OpenMP open scheduler source code as our starting code to further implement cache-aware scheduler.
- We will use C++ as our developing language.

Schedule

Week Number	Checkpoint
1	Study OpenMP’s open scheduler source code
2	Design and revise the algorithm of cache hierarchy-aware scheduler
3	Implementation of the scheduler
4	Perform analysis and gather data among several schedulers
5	Work on report and extending scheduler to be dynamically adjustable