15618 Project Milestone: Cache Hierarchy-Aware Task Scheduler on Multicore Architectures

Team members: Calvin Lin, Yen-Shuo 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.
- The scheduling mechanism is aimed to have the ability to adapt to various different kinds of cache hierarchies within systems and achieve a boost in performance compared to the scheduling approaches implemented within OpenMP, mainly static, dynamic, and guided scheduling types. Various types of cache hierarchies are shown in the figure below.

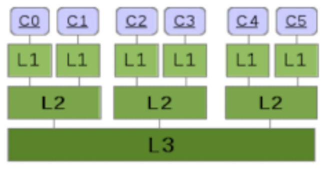

Fig. 1. Intel Dunnington

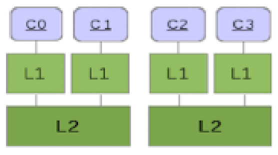

Fig. 2. Intel "Hapertown"

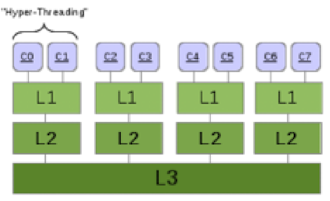

Fig. 3. Intel "Nehalem"

Updated Goals and Deliverables

Plan to Achieve

- Implement a cache hierarchy-aware task scheduler that considers cache hierarchy for task execution.
- Demonstrate improved cache hit rates and reduced execution times for several representative compute-intensive benchmarks like LINPACK, MLPerf, and SPEC. Also, compare the cache performance to some standard schedulers such as 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 integrate them into 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.
- 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.

Work Done So far

- We have went over the source code of GNU implementation of OpenMP, which is located under the libgomp directory of the gcc Github repository. The file that is responsible for task partitioning using the specified scheduling protocol is loop_ull.c. We have familiarized with how the existing scheduling protocols are implemented, specifically, static, dynamic, and guided scheduling types in order to integrate our proposed scheduling mechanism properly into the existing file.
- We also looked into several works regarding cache-hierarchy aware task scheduling, and found the work "Design and Implementation of a Cache Hierarchy-Aware Task Scheduling for Parallel Loops on Multicore Architectures" to provide multiple insights into the topic. The hybrid approach of using static and dynamic scheduling according to cache hierarchy can bring improvements in performance compared to pure static and dynamic scheduling

approaches. We have currently set up the environment and started working on the implementation of this approach. We would like to conduct experiments on how the performance will be affected when the tasks are partitioned differently into the static and dynamic counterparts.

- Currently, we are right on track according to the schedules we have proposed in our initial proposal, working our way into the implementation of the new schedulers this week. We are expected to meet all of our goals and produce all of the deliverables that were mentioned in our proposal. The “nice to haves” will depend on how the implementation and the initial results of our implementation of the scheduler. If everything go well and as expected, we will work our way into the “nice to haves” mentioned.

Poster Session Goals

- Since our results will mainly be the performance comparison between our newly proposed scheduler and the ones implemented within OpenMP, such as static, dynamic, and guided, the main results we will showcase during the poster session will be in the form of charts that can easily convey the performance comparison between the different types of schedulers across various tasks. Some performance metrics that will be focused on are cache miss rates, execution time, and speedups. Since we also want to test how the ratio between static and dynamic subparts of our scheduler affects the performance, we will also showcase a plot that looks for the “sweet spot” of this ratio that generates the optimal performance for a given task under a specific cache hierarchy.

Updated Schedule for Project Milestone

Below is the updated schedule for the coming 2.5 weeks in half-week granularity (between milestone and project deadline).

Week Number	Checkpoint	Person in charge	Task Status
0 - 0.5	Build the initial version of the proposed scheduler.	Yen-Shuo	
0.5 - 1	Debug and customize the scheduler to make it easier to conduct experiments related to sensitivity analysis.	Calvin	

1 - 1.5	Use the scheduler and test its performance on one of the tasks used for performance comparison between different types of schedulers.	Yen-Shuo	
1.5 - 2	Measure the performance gains of the proposed schedulers against OpenMP implemented scheduler types using various tasks. Make adjustments if needed.	Calvin	
2 - 2.5	Buffer time to conduct any extra experiments needed. If everything goes well, look into the tasks listed in “nice to haves.”	Calvin, Yen-Shuo	

Most Concerned Issues

1. **Complexity and Interdependencies:** Scheduling in a OpenMP runtime involves numerous components and is tightly integrated with other runtime features like synchronization, load balancing, and communication. Modifying the scheduler might have several side effects on these other components. Understanding and managing these interdependencies is crucial to ensure that the modifications won’t introduce unexpected behaviors or degrade performance in other parts of the system.

2. **Performance Effect:** Changes in scheduling strategies might lead to performance regressions in other metrics:

   - **Load Balancing**: Ensuring that the workload is evenly distributed across threads to avoid idle resources.
   - **Overhead**: Introducing a new scheduler that is cache-hierarchy aware might increase the runtime overhead due to more complex decision-making and some additional steps for cache metrics collecting.
   - **Scalability**: Ensuring the modified scheduler can scale effectively with increasing numbers of threads or cores.

3. **Portability and Hardware Specific Optimizations:** Since we are taking the cache-hierarchy factor into account, which means the scheduler will reach more lower-level in hardware abstractions. We need to make sure that the scheduler is portable across different

underlying machine, accommodating different number of CPU cores and various cache topologies, rather than being overfitted and optimized for a specific machine.