Migration Cost Aware Task Scheduling

Shraddha Joshi  
Carnegie Mellon University  
shraddhj@andrew.cmu.edu

Brian Osbun  
Carnegie Mellon University  
bosbun@andrew.cmu.edu

Abstract—Heterogeneous systems can improve power-performance efficiency by scheduling tasks on the most suitable cores. Dynamic task scheduling provides the flexibility to migrate tasks between cores during execution. However, many dynamic schedulers do not fully consider the additional overhead incurred when migrating tasks. In this project, we propose to more accurately quantify migration overhead, and integrate this calculation into the scheduling algorithm. By the end of the semester, we will display how this smarter scheduling can provide better power-performance tradeoffs.

I. INTRODUCTION

In today’s era of computing, power and power density have become the greatest barriers to overcome. One modern solution to this problem is the use of large-scale parallelism, especially with heterogeneous cores. By scheduling tasks with different attributes on differently featured cores, higher performance can be achieved where it is necessary and lower power can be used elsewhere. In particular, dynamic mapping of tasks allows continuous, “on-the-fly” fine-grained adjustment of power-performance tradeoffs.

However, there can be hidden costs to this dynamic migration. For instance, moving a task to a core with a separate first-level cache will result in cache misses if previous data is reused. These cache misses can cause stalls after the program resumes execution, and requires additional activity on the memory buses. Of course, it is also necessary to copy the architectural state (the register file) between cores. In addition to the time delay generated by these copies, there is a negative effect due to increased congestion on the chip interconnect.

Therefore, it is important for a dynamic scheduler to predict the impact of this overhead when deciding whether to move a task between cores. If the cost is high, then it may be preferable to continue execution on the original core. In terms of performance, the delay may be worse than the speedup gained by moving to an accelerator. In terms of power, the increased memory activity may offset the gains of moving to a more efficient core.

Despite these caveats, many current schedulers simply disregard the migration overhead when considering task mapping. Thus, we believe that scheduling can be improved upon by adding this information into the algorithm. If migration cost is high, it is possible that schedulers are too active and sacrificing performance. If migration cost is low, it is possible that rescheduling could be performed at even finer grains to extract more performance. Regardless, providing the scheduler with more information will allow for more accurate and intelligent decisions.

II. RELATED WORK

A. Dynamic Task Scheduling

Significant work has been done in scheduling tasks on heterogeneous multicore systems. R. Kumar et al. [3] use two oracle algorithms for dynamic core selection. The first oracle is based on energy metric and the second oracle utilizes the energy-product metric. It uses the instructions committed per second (IPS) as an indicator for task scheduling. Craeynest et al. [4] schedules the threads based on the performance impact estimation. It considers different characteristics of the thread such as Memory Level Parallelism (MLP), Instruction Level Parallelism (ILP) and IPC. IPC characteristics are a good indicator of what type of core the thread should be scheduled on. Here it is considered in conjunction with ILP and MLP. We have decided to use a similar idea in our scheduling algorithm. We are planning to use IPC as an indicator for scheduling as High-IPC is a good indicator for moving to bigger core. The paper also tries to quantify the migration overhead by testing on various cache hierarchy designs at fine grained dynamic scheduling. The performance overhead is measured for different benchmarks on these cache designs.

B. Migration Cost Estimation

Rangan et al. [1] have presented thread motion (TM) to overcome the limitations of conventional DVFS. The author has studied the program variability at a fine grained level and shows the relationship between overall IPC and variability of a program. Two approaches of thread motion have been considered: miss driven TM and time driven TM. The algorithm used predicts the effective IPC after taking into account the predicted TM cost. In order to predict the TM cost, the algorithm uses the count of memory instructions. The architectural scheme used in this paper is based on Sun’s Rock architecture which we have decided to use in our study as well.

Hardy et al. [2] have proposed a method to estimate the worst case cache reload cost due to task migration between cores. The paper is divided into two parts. In the first part, the author proposes a migration aware cache analysis method. In the latter part, the author proposes a method to compute a tight upper bound of the cache related migration delay using static analysis. The author uses the notion of “useful blocks” where useful blocks at any execution point is defined as the memory blocks which maybe referenced before being replaced. The usefulness is calculated by Reaching Memory Block (RMB) Analysis and the Live Memory Block (LMB) Analysis. Reaching Memory block (RMB) analysis determines all the memory blocks that may be in the cache at a program point. LMB analysis determines all the memory blocks that may be referenced before their eviction via any outgoing...
path from the current execution point. The usefulness is then calculated as the intersection of the results obtained by the RMB and LMB analysis. The useful cache blocks calculated are then used in the calculation of Cache Related Migration Delay.

Our scheduler needs to make on-the-fly schedulability decisions considering the migration cost that maybe incurred. As a result offline analysis of the workload is not possible. But we have decided to use the concept of useful blocks for making schedulability decisions on the fly. We slightly modify the definition of useful blocks at any point of execution as the memory blocks in the cache that were referenced by the program up to now. We make a basic assumption that all the memory blocks that were used till now will be reused in the future. Also, the number of accesses to the cache will be directly proportional to the number of useful cache blocks at any given time in the program execution. Thus we have decided to use the number of cache accesses as a predictor for number of cold cache misses incurred due to migration of the task.

III. PROPOSED PROJECT FRAMEWORK

A. Simulation Framework

We plan to use the Sniper simulator for our project. Sniper is a highly accurate execution-driven x86 simulator. It should be well suited to our project, due to its wide range of options for configuring heterogeneous systems. Through configuration files, it is possible to provide our own configuration of the heterogeneous system by providing details like number of cores, frequency, number of levels in the cache hierarchy, associativity, cache size, operating voltage, etc. Each core can be configured differently thus allowing true heterogeneous simulation. Python interfaces are available for runtime analysis and manipulation of the state of the simulator.

B. Proposed Architecture Configuration

Our system will be set up similar to the Sun Rock architecture [1]. This configuration is composed of cores that are grouped into clusters as shown in Fig. 1. Each cluster is composed of some mix of heterogeneous cores (proposed, 4 cores). L1 data and instruction caches are shared between all cores in the cluster. The system contains multiple instances of (proposed, 4 clusters). An L2 unified cache is shared between all clusters in the system. The number of clusters, and number of cores per cluster, may be adjusted as we experiment. Whether the clusters will be homogeneous or of varying power/performance levels will be a decision taken as we experiment more.

This configuration is interesting because there are two levels of migration possible (intra-cluster and inter-cluster). Inter-cluster migration may not seem intuitive, but there are situations where it has benefits. For example, consider the case where all tasks within a cluster become very compute-intensive. It may be superior to move the task on the smallest core to another cluster with an open big core.

C. Scheduler Design

We plan to use the calculated migration cost as an important part of the task mapping process. We will conduct analysis and experiments to produce an appropriate weighting for this information. Additionally, we hypothesize an initial threshold structure for tasks in our architecture:

- If migration cost is high, continue execution on original core.
- If migration cost is moderate, consider migration within cluster.
- If migration cost is low, consider migration to another cluster.

We are planning to use IPC as a metric to take the main scheduling decision. After incorporating the migration cost, final decision to migrate inter or intra cluster, or to continue execution on the same core will be then taken.

IV. METHODOLOGY

The architectural configuration described in the previous section was simulated on Sniper by specifying the configuration file passed into the simulations. However, for the initial experiments conducted to obtain a correlation between number of extra misses incurred and the number of cache accesses, homogeneous configuration was considered. All the cores in the cluster were of the same size running at a nominal frequency of 2.66 GHz. The effect of heterogeneity will be considered in the latter stage of the project. Each cluster had a private L1 cache, whereas, both the clusters shared an L2 cache. The size of the L2 cache was set to be 256KB.

Two benchmarks were used:

- **Splash2-radix**, which is a compute intensive benchmark.
- **Parsec-x264**, which is a memory intensive benchmark.

The experiment was conducted in two parts. In the first part, simulations were run on the defined configuration by running the above mentioned benchmarks. A single thread was created and the thread was then migrated to another core outside the cluster. As a base case, the entire thread was run
on one core without migration. The extra misses incurred due to this migration were noted, and the number of cache accesses before the migration was done were also noted. This experiment was repeated by changing the time of migration and performing the statistical collection again.

In the second part of the experiment, the cache size and its associativity were changed, and the same statistics were collected for each cache configuration. Only the memory-intensive benchmark was used for this portion. The cache size and its associativity can have an effect on the cache misses. Hence we felt a need to study the effect of the cache parameters on the extra misses incurred due to migration. The results of this sweep could be used for our final testing configuration.

V. RESULTS AND OBSERVATIONS

The results for the first part of the experiment with cache size fixed are shown in Fig. 2. In case of radix which is a compute intensive benchmark, migration causes essentially no extra misses in the L1D cache. On the other hand, for x264 benchmark which is a memory intensive benchmark, it can be seen that the number of extra misses incurred follow a logarithmic trend to the number of total cache accesses made before migration. When there are few previous accesses, there are few extra cache misses after migration because fewer blocks need to be reloaded. As the number of previous cache accesses increases, the number of extra misses increase substantially and eventually saturates once the program is no longer raising the useful count.

The second part of the experiment was carried out by running simulations by using the following cache parameters:

- 32KB 8-way cache
- 32KB 4-way cache
- 16KB 8-way cache
- 16KB 4-way cache

The results are shown in Fig. 3. As expected, the number of extra cache misses are greater in the cache with cache size of 32KB as compared to the cache of size 16KB. As the size of the cache is larger, more useful cache blocks can be present in the cache, and migration will cause a loss of these useful cache blocks.

The results of the experiment for associativity were not as obvious. Given the reasoning for the cache size, we were expecting highly-associative caches to experience more additional misses, since they are less likely to evict useful data before a migration. Instead we found that the 32KB caches were very similar. Between the 16KB caches, the 8-way cache had fewer additional misses than the 4-way cache. This could be because the 4-way cache simply had more cache evictions in the program execution after migration.

Thus, from the results it can be seen that a correlation exists between the number of previous cache accesses and the extra misses incurred due to migration. We intend to use this correlation to make scheduling decisions.

VI. UPDATED PROJECT SCHEDULE

A. Milestones

The project will be divided into three major milestones. The first two are somewhat independent and will synthesize to produce the third.

1) Determine an accurate formula for quantifying migration cost, in terms of cycles or equivalent performance metrics.
2) Develop or modify a dynamic task scheduling algorithm that is well-suited to incorporate the migration cost factor.
3) Synthesize the migration cost into the scheduling decision to provide an improvement over ignorant algorithms.

As a part of the first milestone, an attempt was made to quantify the migration cost in terms of a metric. We chose the number of extra misses incurred as an indicator to the migration overhead. However, in order to accurately quantify the migration cost, more experiments are needed to be done to find an accurate correlation between the number of previous accesses and the extra misses. Additionally, we should quantify the impact that these additional misses have on program performance. Thus the latter milestones remain the same with
an immediate milestone of quantifying the migration cost to suit the proposed design.

B. Deliverables

Our deliverables are related to the milestones above. At the project checkpoint, we should have a firm understanding and formulation of our migration cost model. The immediate deliverable is to quantify the migration cost suitable to the proposed scheduler design. At the end of the semester, we should have fully integrated this model into our scheduling algorithm and have simulation results showing the benefit of considering the full cost.

C. Project Site

We have started a web page for our project that will hold our status reports and results throughout the semester. The site can be accessed at http://www.andrew.cmu.edu/user/bosbun/.

VII. INDIVIDUAL CONTRIBUTIONS

Shraddha Joshi ran the simulations for the part 1 of the experiment with no change in cache parameters and updated the report. Brian Osbun ran the simulations for part 2 of the experiment with varying cache parameters and made the website. Brainstorming, configuration setup and presentation preparation was done by both.

REFERENCES