Review #5

Bias Scheduling in Heterogeneous Multi-core Architectures

D. Koufaty, D. Reddy, S. Hahn

Proceedings of the EuroSys 2010 Conference, 2010

Jean-Pierre Lozi

May 6, 2010

1 Problem

Heterogeneous architectures whose processing cores differ in computing power are very attractive: they offer high single-threaded performance and hardware multi-threading capabilities at lower energy costs than conventional chips. However, traditional scheduling algorithms perform poorly on these systems since they usually assume that all cores are equivalent. Some algorithms have been specially designed for heterogenous systems but they usually add a significant overhead since they require offline profiling or dynamic sampling on all cores. This paper proposes a novel algorithm that avoids this problem since it collects all of its data on-the-fly.

2 Solution

The authors propose new metrics to evaluate the performance gain of executing a process on a big core rather than a small core[1] (i.e. *application bias*): in addition to the traditional *Clocks Per Instruction* (CPI) metric, they also monitor internal stalls (caused by accesses to resources internal to the core) and external stalls (caused by accesses to shared last level caches, memory and I/O). They show that a process has a *big core bias* (i.e. its speedup from running on a big core compared to a small core is large) when the number of stalls (especially external stalls) is low. The number of internal and external stalls is measured at runtime and used to compute application bias. As this bias is not constant during the execution of an application, it is measured over a sliding instruction window. This allows the algorithm to schedule more efficiently applications whose behavior changes throughout their execution.

When the system is imbalanced, the scheduler tries to migrate threads from the busiest core to the idlest core, using application biases to optimize its choice of threads to move. When the system is balanced, the runqueues of each core are periodically checked: the scheduler looks for processes that would benefit from being swapped from big to small cores and conversely according to their application bias.

3 Evaluation

Since heterogeneous chips are uncommon, the authors emulate one *via* an homogeneous quad-core Intel® Xeon® processor. However, they show that the standard approach of downclocking some of the cores to turn them into "small cores" is not representative of heterogeneous architectures in the general case: these architectures usually have structurally diverse cores. Instead, they use proprietary tools to enable a debug mode on some cores that reduces instruction retirement from four to one micro-op per cycle. They show that this throttling method gives performance results similar to actual heterogeneous architectures whose small cores are in-order whereas their big cores are out-of-order.

[1] This article only focuses on architectures having cores of exactly two different types with the same ISA.

To evaluate performance, the authors perform experiments with heterogeneous workloads (i.e. workloads whose processes have very different biases) based on the SPEC CPU2006 benchmark. On average, they obtain a 9% performance improvement over the default Linux scheduler. This is close to an upper bound found by running the tests on the same processor without throttling any of the cores. They also perform experiments with more homogeneous workloads and obtain a 5% gain on average. They show that this improvement is due to the fact that even though the application biases are similar overall, they vary throughout the execution of each benchmark.

4 Benefits

Bias scheduling is more efficient than traditional scheduling algorithms on heterogeneous architectures. Its overhead is minimal. It is also relatively simple to implement (a few thousands lines of code were needed to implement it in the Linux kernel).

5 Shortcomings

The performance gain is relatively limited (less than 10% on average) and precisely estimating internal and external stalls is difficult and architecture-dependant: the authors explain that manufacturers should provide better ways to estimate these metrics in the future for bias scheduling to be efficient. Also, the proposed algorithm only works for architectures having exactly two types of cores. Moreover, finding efficient thresholds for the application bias can be difficult, especially since these thresholds can vary depending on the workload. Finally, in NUMA systems, bias scheduling can unpredictably worsen performance by decreasing data locality though the migration of processes from one core to another.