Delivering Fairness and Priority Enforcement on Asymmetric Multicore Systems via OS Scheduling

Juan Carlos Sáez Fernando Castro Daniel Chaver Manuel Prieto
Department of Computer Architecture. School of Computer Science
Complutense University of Madrid[*]
{jcsaezal,fcastror,dani02,mpmatias}@pdi.ucm.es

ABSTRACT

Symmetric-ISA (instruction set architecture) asymmetric-performance multicore processors (AMPs) were shown to deliver higher performance per watt and area than symmetric CMPs for applications with diverse architectural requirements. So, it is likely that future multicore processors will combine big power-hungry *fast* cores and small low-power *slow* ones. In this paper, we propose a novel thread scheduling algorithm that aims to improve the throughput-fairness trade-off on AMP systems. Our experimental evaluation on real hardware and using scheduler implementations on a general-purpose operating system, reveals that our proposal delivers a better throughput-fairness trade-off than previous schedulers for a wide variety of multi-application workloads, including single-threaded and multithreaded applications.

Categories and Subject Descriptors

D.4.1 [**Process Management**]: Scheduling; C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Heterogeneous (hybrid) systems*

General Terms

Algorithms, Performance, Measurement

Keywords

Asymmetric Multicore, Operating Systems, Scheduling

1. INTRODUCTION

Asymmetric multicore processors (AMP) [3] were proposed as a more power-efficient alternative to conventional multicore processors that consist of identical cores. An AMP contains at least two core types, "fast" and "slow", which support the same instruction-set architecture, but differ in microarchitectural features, size, power consumption and performance. Early studies have demonstrated that having just two core types is sufficient to extract most of the benefits from AMPs [3] and simplifies their design. For that reason, in this work we target AMP systems with two core types.

Most existing proposals on scheduling for AMPs strive to maximize the system thoughput [5, 2]. Pursuing other important goals such as delivering fairness or priority enforcement on AMPs has drawn less attention from the research community. Schedulers such as RR [5] or A-DWRR [4] aim to provide fairness by alloting the same amount of heterogeneous CPU share to equal-priority applications. However, because applications in a multiprogram workload may derive different benefit from using the fast cores in an AMP, assigning the same heterogeneous CPU share to equal-priority application does not ensure an even slowdown (due to sharing the AMP) across applications. In addition, not taking into account the diversity in relative speedups may lead to degrading the system throughput significantly [5]. To address these shortcomings, we propose Prop-SP, a novel scheduling algorithm that supports priority enforcement on AMPs and attempts to even out the slowdown experienced by equal-priority applications due to sharing the system, while ensuring a high system throughput. Prop-SP is equipped with support to effectively accelerate some types of parallel applications on AMPs. We evaluated Prop-SP using real hardware and compared it against other scheduling schemes.

2. DESIGN

The Prop-SP scheduler assigns threads to fast and slow cores so as to preserve load balance in the AMP system, and migrates threads among fast and slow cores to ensure that they run on fast cores for a specific amount of time.

To enforce that threads receive a specific fast-core share, Prop-SP relies on a mechanism inspired by Xen's Credit Scheduler [1]. At a high level, this mechanishm works as follows. Each thread has a fast-core credit counter associated with it. When a thread runs on a fast core it consumes credits. Threads that have fast-core credits left (i.e., their credit counter is greater than zero) are preferentially assigned to fast cores by the scheduler.

Every so often, the OS triggers a *credit assignment process* that allots fast-core credits to active applications (i.e., with runnable threads). Each application receives credits in proportion to its dynamic weight, which is defined as the product of its priority and its net speedup (speedup minus one). The application priority is set by the user. The speedup indicates the relative benefit that the application would derive if all fast cores in the AMP were devoted to running threads from this application, with respect to running all threads on slow cores. This speedup is estimated at runtime by Prop-SP without the user intervention. For single-threaded applications, determining the speedup entails determining the

[*]This research was funded by the Spanish government's research contracts TIN2012-32180 and the Ingenio 2010 Consolider ESP00C-07-20811.

Table 1: Metrics to assess throughput and fairness for a workload consisting of n applications running simultaneously under a given thread scheduler.

Metric	Definition
$CT_{fast,app}$	Completion time of application *app* when running alone in the system (with all the fast cores available)
$CT_{slow,app}$	Completion time of application *app* when it runs alone in the system but using slow cores only
$CT_{sched,app}$	Completion time of application *app* in the multiprogram workload running under a given scheduler.
$Slowdown_{app}$	$CT_{sched,app}/CT_{fast,app}$
Aggregate Speedup	$\sum_{i=1}^{n}\left(\frac{CT_{slow,i}}{CT_{sched,i}}-1\right)$
Unfairness	$\frac{MAX(Slowdown_1,\ldots,Slowdown_n)}{MIN(Slowdown_1,\ldots,Slowdown_n)}$

speedup factor (SF) of its single runnable thread: $\frac{IPS_{fast}}{IPS_{slow}}$, where IPS_{fast} and IPS_{slow} are the thread's instructions per second ratios achieved on fast and slow cores respectively. To this end we leverage the technique proposed in [5]. For multithreaded programs, several factors in addition to the SF must be taken into account to estimate the speedup, such as the amount of thread-level parallelism (TLP).

Credits awarded to the application are distributed among its runnable threads. For single-threaded programs, this boils down to increasing the fast-core credit counter of its single thread by the amount of credits awarded. For multi-threaded programs, Prop-SP supports several credit distribution schemes to meet the needs of diverse applications.

3. RESULTS AND CONCLUSIONS

We assessed the effectiveness of two variants of Prop-SP (static and dynamic), which follow different approaches to determine a thread's SF. The *base* implementation of Prop-SP, referred to as *Prop-SP (dynamic)*, estimates SFs online using hardware counters (using the technique from [5]). *Prop-SP (static)*, on the other hand, asummes a constant SF value for each thread, measured prior to the execution. We compare both versions of Prop-SP against the RR (Round Robin) scheduler, which simply fair-shares fast cores among all applications in the workload, and HSP (High-SPeedup) [2, 5], which aims to maximize the system throughput by mapping to fast cores those applications that derive the greatest performance benefit from these cores.

All the evaluated algorithms have been implemented in the Solaris kernel and tested on real multicore hardware made asymmetric by reducing the processor frequency of a subset of cores in the platform. Figure 1 shows the results for various multi-application workloads consisting of both single-threaded and multithreaded applications running under the various schedulers. Overall, HSP, yields the best system throughput but fails to deliver fairness accross the board. RR, on the other hand, does rather a good job in terms of both fairness and throughput for workloads including single-threaded applications only. However, when multithreaded applications are present in the workload, it degrades the system throughput significantly. In contrast, Prop-SP is able to make efficient use of the AMPs and improve the throughput-fairness tradeoff for a wider range of workloads. Moreover, the potential inacuracies of the SF

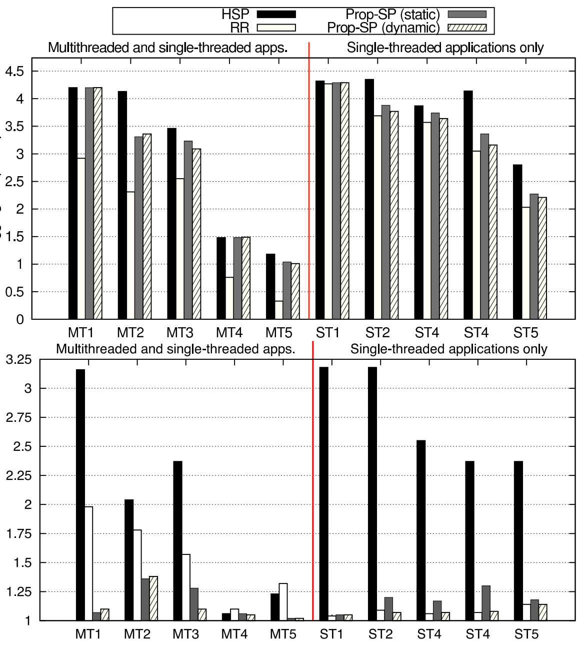

Figure 1: Results for various multi-application workloads running on an AMD-based emulated AMP system where fast cores run at 2.6Ghz and slow cores at 0.8Ghz. Both the aggregate speedup and unfairness metrics are defined in Table 1.

estimation model used by Prop-SP (dynamic), do not prevent this scheduler from reaping benefits similar to those of the static version. Overall, these benefits are especially pronounced for workloads including multithreaded programs.

We also experimented with scenarios where we gradually increase the priority of one or several *high-priority* (HP) applications in the workload. The results reveal that Prop-SP is able to effectively reduce the completion time of HP applications as the priority increases, while ensuring higher throughput and lower unfairness than RR.

Key elements for the success of Prop-SP are the credit-based mechanism enabling the scheduler to adjust the fast-core share alloted to the different programs, its reliance on estimation models to approximate per-thread speedup factors (SFs) online and the specific support to accelerate certain types of multithreaded programs.

4. REFERENCES

[1] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three CPU schedulers in Xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, Sept. 2007.

[2] D. Koufaty et al. Bias Scheduling in Heterogeneous Multi-core Architectures. In *Proc. of Eurosys '10*, 2010.

[3] R. Kumar et al. Single-ISA Heterogeneous Multi-Core Architectures: the Potential for Processor Power Reduction. In *Proc. of MICRO 36*, 2003.

[4] T. Li et al. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *HPCA'10*, pages 1–12, 2010.

[5] J. C. Saez et al. Leveraging core specialization via OS scheduling to improve performance on asymmetric multicore systems. *ACM Trans. Comput. Syst.*, 30(2):6:1–6:38, Apr. 2012.