Theoretical Study on the Performance of an Asymmetry-Aware Round-Robin Scheduler

Adrian Pousa, Juan Carlos Saez, Fernando Castro
Daniel Chaver and Manuel Prieto

`Technical Report - Ref. 5028A - December, 2012`

1 Introduction

Symmetric-ISA (Instruction Set Architecture) asymmetric-performance multicore processors (AMPs) were shown to deliver higher performance per watt and area than its symmetric counterparts [6]. So it is likely that future multicore processors will combine a few *fast* cores characterized by complex pipelines, high clock frequency, high area requirements and power consumption, and many *slow* ones, characterized by simple pipelines, low clock frequency, low area requirements and power consumption.

Despite their benefits, AMPs pose significant challenges to the system software [5]. One of the main challenges is how to efficiently distribute fast-core cycles among the various applications running on the system. Previous work has demonstrated that a simple asymmetry-aware round-robin (RR) scheduler [2] provides better performance than default schedulers on most general-purpose OSes [12], which are asymmetry agnostic. The RR scheduler simply fair-shares fast cores among all threads in the workload.

Although the asymmetry-aware RR scheduler has been used extensively as a baseline for comparison in previous work [2, 11, 12, 10] no comprehensive analytical analysis on the performance of this scheduler has been carried out to date. In this report, we derive a set of analytical formulas enabling to assess the real potential of this scheduler regarding system throughput and fairness.

The remainder of this report is structured as follows. Section 2 presents the metrics we used to assess throughput and fairness on AMP systems. Section 3 illustrates the derivation of the formulas to estimate system throughput and fairness for multi-application workloads running under RR. Finally, Section 4 shows how to estimate the performance benefit that a regular data-parallel multithreaded application derives when running alone on an AMP under RR.

2 Metrics to assess throughput and fairness

To assess *system throughput* we use the *Aggregate Speedup* (ASP), which is defined as follows:

$$\textit{Aggregate Speedup} = \sum_{i=1}^{n} \left(\frac{CT_{slow,i}}{CT_{sched,i}} - 1 \right) \tag{1}$$

where n is the number of applications in the workload, $CT_{slow,i}$ is the completion time of application i when it runs alone in the system and uses slow cores only, and $CT_{sched,i}$ is the completion time of application i under a given scheduler.

Unlike other metrics to measure throughput, such as the Weighted Speedup [13] or the Harmonic Mean Speedup [8], the ASP does not depend on *instructions per cycle* (IPC). Relying on completion time to assess application performance is known to be more suitable than using the IPC in scenarios where multithreaded applications are present in the workload [1].

Regarding *fairness*, previous works have employed diverse definitions. Some of them define a scheme to be fair if it assigns the same CPU share to equal-priority threads [7]. Others consider a scheme as fair if equal-priority applications suffer the same slowdown due to sharing the system with respect to the situation in which the whole system is available to each application [4, 9, 3]. The latter definition is more suitable for CMP systems where degradation due to contention on shared resources may occur. Therefore, we opted to use this definition and employ the *unfairness* metric [9, 3], which is defined as follows:

$$Unfairness = \frac{MAX(Slowdown_1, ..., Slowdown_n)}{MIN(Slowdown_1, ..., Slowdown_n)} \tag{2}$$

where $Slowdown_i = CT_{sched,i}/CT_{fast,i}$, and $CT_{fast,i}$ is the completion time of application i when running alone in the AMP.

3 Derivation of analytical for the unfairness and the aggregate speedup

Our goal is to derive a set of analytical formulas to approximate the unfairness and the aggregate speedup under RR for a multi-programmed workload consisting of n single-threaded applications. For the derivation, we considered the scenario where applications in the workload run continuously for a certain amount of time T.

For the sake of generality, we begin by deriving a set of formulas to compute both metrics under any work-conserving[1] asymmetry-aware thread scheduler, whose behavior is expressed only in terms of how fast-core cycles are distributed among applications. Specifically, throughout the execution the scheduler allots each application *app* a certain fast-core time fraction, denoted as F_{app}, such that $0 \leq F_{app} \leq 1$, where $F_{app} = 1$ means that the application would be mapped to a fast core the whole time. Since RR equally shares fast cores among applications, the ASP and the unfairness metrics for RR can be computed by replacing F_{app} with $\frac{N_{FC}}{n}$ in the resulting formulas, where N_{FC} denotes the number of fast cores.

To make the derivation tractable, we make the following simplifying assumptions:

- The number of applications in the workload does not exceed the total number of cores.
- Each application exhibits a uniform fast-to-slow speedup throughout execution.
- The analytical formulas derived in this section do not account for migration overheads and shared-resource contention. Note, however, that the OS may take into account these aspects when making scheduling decisions.

Prior to deriving the formulas, we introduce some auxiliary notation:

- $N_{FC},\ N_{SC}$: Number of fast and slow cores of the AMP system, respectively.

[1]Such a scheduler does not leave idle cores when the total thread count is greater or equal to the number of cores in the platform.

- SPI_{FC}, SPI_{SC} : Average number of seconds per instruction for a given application when running on a fast and a slow core, respectively.
- SF : Fast-to-slow speedup or *Speedup factor* (SF) of the application. For single-threaded applications, $SF = \frac{SPI_{SC}}{SPI_{FC}}$.
- NI: Total number of instructions the application executes until completion.
- F_{fast}, F_{slow}: Time fraction over CT_{sched} that the application runs on a fast and a slow core, respectively. Obviously, $F_{fast} + F_{slow} = 1$.
- f_{fast}, f_{slow}: Fraction of instructions (over NI) that the application completes on a fast and a slow core under a given scheduler. Obviously, $f_{fast} + f_{slow} = 1$.

To compute the Unfairness, we first obtain a formula enabling to approximate an application's slowdown under a given scheduler in terms of f_{fast} and the application's SF. The derivation process is as follows:

$$\begin{aligned}
Slowdown &= \frac{CT_{sched}}{CT_{fast}} \\
&= \frac{NI \cdot SPI_{FC} \cdot f_{fast} + NI \cdot SPI_{SC} \cdot f_{slow}}{NI \cdot SPI_{FC}} \\
&= \frac{NI \cdot SPI_{FC} \cdot f_{fast} + NI \cdot SPI_{FC} \cdot SF \cdot (1 - f_{fast})}{NI \cdot SPI_{FC}} \\
&= \frac{NI \cdot SPI_{FC} \cdot (f_{fast} + SF \cdot (1 - f_{fast}))}{NI \cdot SPI_{FC}} \\
&= f_{fast} + SF \cdot (1 - f_{fast}) \qquad (3)
\end{aligned}$$

We now derive the equation for the aggregate speedup under a given scheduler in terms of the SF and f_{fast} of each application i.

$$\begin{aligned}
Aggregate\ Speedup &= \sum_{i=1}^{n} \left(\frac{CT_{slow,i}}{CT_{sched,i}} - 1 \right) = \\
&= \sum_{i=1}^{n} \left(\frac{NI_i \cdot SPI_{SC,i}}{NI_i \cdot SPI_{FC,i} \cdot f_{fast,i} + NI_i \cdot SPI_{SC,i} \cdot f_{slow,i}} - 1 \right) \\
&= \sum_{i=1}^{n} \left(\frac{NI_i \cdot SPI_{SC,i}}{NI_i \cdot \frac{SPI_{SC,i}}{SF_i} \cdot f_{fast,i} + NI_i \cdot SPI_{SC,i} \cdot f_{slow,i}} - 1 \right) \\
&= \sum_{i=1}^{n} \left(\frac{NI_i \cdot SPI_{SC,i}}{NI_i \cdot SPI_{SC,i} \cdot \left(\frac{f_{fast,i}}{SF_i} + (f_{slow,i}) \right)} - 1 \right) \\
&= \sum_{i=1}^{n} \left(\frac{1}{\frac{f_{fast,i}}{SF_i} + (1 - f_{fast,i})} - 1 \right) \qquad (4)
\end{aligned}$$

Since the behavior of the investigated schedulers is expressed in terms of the fast-core time distribution they make, we now devise a means to approximate the slowdown and the aggregate speedup based on F_{fast} and SF. To make this possible, we derive Equation 7, which establishes the relationship between F_{fast} and f_{fast}.

$$\begin{aligned} CT_{sched} &= NI \cdot SPI_{FC} \cdot f_{fast} + NI \cdot SPI_{SC} \cdot f_{slow} \\ &= NI \cdot SPI_{FC} \cdot f_{fast} + NI \cdot SPI_{FC} \cdot SF \cdot (1 - f_{fast}) \\ &= NI \cdot SPI_{FC} \cdot (f_{fast} + SF \cdot (1 - f_{fast})) \end{aligned} \tag{5}$$

Note also that:

$$\begin{aligned} CT_{sched} \cdot F_{fast} &= NI \cdot f_{fast} \cdot SPI_{FC} \\ \Rightarrow \quad CT_{sched} &= \frac{NI \cdot f_{fast} \cdot SPI_{FC}}{F_{fast}} \end{aligned} \tag{6}$$

Merging equations 5 and 6, we obtain:

$$\begin{aligned} &\frac{NI \cdot f_{fast} \cdot SPI_{FC}}{F_{fast}} = \\ &= NI \cdot SPI_{FC} \cdot (f_{fast} + SF \cdot (1 - f_{fast})) \\ &\Rightarrow f_{fast} = \frac{1}{\frac{1}{SF} \cdot \left(\frac{1}{F_{fast}} - 1\right) + 1} \end{aligned} \tag{7}$$

The fraction of instructions executed by an application under RR ($f_{fast-RR}$) can be obtained by replacing F_{fast} with $\frac{N_{FC}}{n}$ in Equation 7.

$$\Rightarrow f_{fast-RR} = \frac{1}{\frac{1}{SF} \cdot \left(\frac{n}{N_{FC}} - 1\right) + 1} \tag{8}$$

Finally, the aggregate speedup and unfairness for RR can be computed by using $f_{fast-RR}$ instead of f_{fast} in Equations 2–3 and $f_{fast-RR,i}$ instead of $f_{fast,i}$ in Equation 4 respectively.

4 Speedup of multithreaded application running under RR

Our goal is to derive a formula to estimate how much a regular data-parallel multithreaded application would speed up in the scenario where all fast cores in the AMP are equally shared among the application threads. The speedup is computed with respect to a *slow scenario*, namely, compared to the performance that would result from running all the application threads on slow cores.

To make the derivation tractable, we make several simplifying assumptions about the nature of the parallel application:

- We assume that the application is perfectly balanced (i.e. the work is evenly distributed among the threads).

- For the derivation, we also assume that the application consists of k parallel phases ($k \geqslant 1$) separated by synchronization barriers. Because the application is perfectly balanced, we assume that threads reach the synchronization barriers at the same time in the slow scenario.

- In many data-parallel applications, all their threads exhibit very similar speedup factors since they execute the same code with different data. So for simplicity, we assume the SF is the same across threads.

- We further assume that the number of threads in the application does not exceed the number of cores in the AMP. This assumption is reasonable because CPU-bound applications are not likely to be run with more threads than cores.

- We do not account for migration overheads, so the formulas shown below approximate an upper bound of the achievable speedup.

In deriving the speedup formula, we first obtain an estimate of the speedup for a single parallel phase and then demonstrate that the obtained formula is similar to the one derived assuming k parallel phases.

Prior to illustrating the derivation of the formulas, we introduce some auxiliary notation:

- $CT_{SC,i}$: Completion time of the i-th parallel phase in the *slow scenario*. Given that the multithreaded application is perfectly balanced, $CT_{SC,i}$ matches the completion time of any of its threads for the parallel phase. Note also that $CT_{slow} = \sum_{i=1}^{k} CT_{SC,i}$.

- CT_{RR} : Completion time of the application under RR.

- $CT_{RR,i}$: Completion time of the i-th parallel phase under RR, such that $CT_{RR} = \sum_{i=1}^{k} CT_{RR,i}$. Because RR assigns the same fast-core share across threads and the application is perfectly balanced, $CT_{RR,i}$ matches the completion time of any of its threads for the parallel phase.

- N : Number of threads in the application.

- NI_i : Number of (dynamic) instructions in the i-th parallel phase.

- $NI_{T,i}$: Number of instructions executed by any application thread in the i-th parallel phase. Since the application is balanced, all threads execute the same number of instructions till reaching the barrier. As such, $NI_{T,i}{=}NI_i/N$.

- $F_{fast,i}, F_{slow,i}$: Time fraction over $CT_{RR,i}$ that a thread runs on a fast and a slow core in the i-th parallel phase. Under RR, all threads receive the same share of fast core time: $F_{fast,i} = \frac{N_{FC}}{N}$. Note that to ensure this ideal fast-core cycle distribution, threads must be swapped between fast and slow cores at an infinitesimally small swap period during the parallel phase.

- $f_{fast,i}, f_{slow,i}$: Instruction fraction over $NI_{T,i}$ that a thread executes on a fast and a slow core in the i-th parallel phase. Obviously, $f_{fast,i} + f_{slow,i} = 1$.

As stated earlier, we can approximate CT_{slow} with the time that any thread in the application takes to execute all its instructions on a slow core:

$$\begin{aligned} CT_{slow} &= \sum_{i=1}^{k} CT_{SC,i} = \sum_{i=1}^{k} (SPI_{SC} \cdot NI_{T,i}) \\ &= \sum_{i=1}^{k} (SPI_{SC} \cdot \frac{NI_i}{N}) \\ &= \sum_{i=1}^{k} (SF \cdot SPI_{FC} \cdot \frac{NI_i}{N}) \end{aligned} \tag{9}$$

Similarly, we can express CT_{RR} as follows:

$$CT_{RR} = \sum_{i=1}^{k} CT_{RR,i} \tag{10}$$

where, for each parallel phase $i \in \{1..k\}$, we have that:

$$\begin{aligned} CT_{RR,i} &= f_{fast,i} \cdot NI_{T,i} \cdot SPI_{FC} + f_{slow,i} \cdot NI_{T,i} \cdot SPI_{SC} \\ &= NI_{T,i} \cdot \ (f_{fast,i} \cdot SPI_{FC} + f_{slow,i} \cdot SPI_{SC}) \\ &= NI_{T,i} \cdot \ (f_{fast,i} \cdot SPI_{FC} + (1 - f_{fast,i}) \cdot SPI_{SC}) \\ &= NI_{T,i} \cdot \ (f_{fast,i} \cdot SPI_{FC} + (1 - f_{fast,i}) \cdot SPI_{FC} \cdot SF) \\ &= \frac{NI_i}{N} \cdot SPI_{FC} \cdot \ (f_{fast,i} + (1 - f_{fast,i}) \cdot SF) \end{aligned} \tag{11}$$

Note also that:

$$\begin{aligned} CT_{RR,i} \cdot F_{fast,i} &= f_{fast,i} \cdot NI_{T,i} \cdot SPI_{FC} \\ &= f_{fast,i} \cdot \frac{NI_i}{N} \cdot SPI_{FC} \end{aligned} \tag{12}$$

Clearly, if $N \leq N_{FC}$, RR would map all application threads to fast cores ($f_{fast,i} = F_{fast,i} = 1$). Hence, the speedup under this scheduler would match the SF:

$$Speedup_{RR(N \leq N_{FC})} = \frac{CT_{slow}}{CT_{RR}} = \frac{\sum_{i=1}^{k} CT_{SC,i}}{\sum_{i=1}^{k} CT_{RR,i}} =$$

$$= \frac{\sum_{i=1}^{k} \left(SF \cdot SPI_{FC} \cdot \frac{NI_i}{N} \right)}{\sum_{i=1}^{k} \left(\frac{NI_i}{N} \cdot SPI_{FC} \right)} = SF \tag{13}$$

To aid in approximating the speedup when $N > N_{FC}$, we now proceed to derive an equation that captures the relationship between $f_{fast,i}$ and $F_{fast,i}$ provided that $F_{fast,i} = \frac{N_{FC}}{N}$ under RR. From Equations 11 and 12, we can isolate $f_{fast,i}$ as follows:

$$\begin{aligned}
\frac{f_{fast,i} \cdot \frac{NI_i}{N} \cdot SPI_{FC}}{F_{fast,i}} &= \frac{NI_i}{N} \cdot SPI_{FC} \cdot \left(f_{fast,i} + (1 - f_{fast,i}) \cdot SF\right) \\
\Rightarrow f_{fast,i} &= \frac{SF}{\frac{1}{F_{fast,i}} - 1 + SF} = \frac{1}{\frac{1}{SF} \cdot \left(\frac{1}{F_{fast,i}} - 1\right) + 1} \\
\Rightarrow f_{fast,i} &= \frac{1}{\frac{1}{SF} \cdot \left(\frac{N}{N_{FC}} - 1\right) + 1} \qquad (14)
\end{aligned}$$

Combining Equations 9-12 and 14, we derive the speedup for the parallel application under RR when $N > N_{FC}$:

$$Speedup_{RR(N>N_{FC})} = \frac{CT_{slow}}{CT_{RR}} = \frac{\sum_{i=1}^{k} CT_{SC,i}}{\sum_{i=1}^{k} CT_{RR,i}} =$$

$$= \frac{\sum_{i=1}^{k} \left(SF \cdot SPI_{FC} \cdot \frac{NI_i}{N}\right)}{\sum_{i=1}^{k} \left(\frac{NI_i}{N} \cdot SPI_{FC} \cdot \left(f_{fast,i} + (1 - f_{fast,i}) \cdot SF\right)\right)}$$

$$= \frac{\sum_{i=1}^{k} \left(SF \cdot SPI_{FC} \cdot \frac{NI_i}{N}\right)}{\sum_{i=1}^{k} \left(\frac{NI_i}{N} \cdot SPI_{FC} \cdot \left(f_{fast,i} \cdot (1 - SF) + SF\right)\right)}$$

$$= \frac{\frac{SPI_{FC}}{N} \cdot SF \cdot \sum_{i=1}^{k} (NI_i)}{\frac{SPI_{FC}}{N} \cdot \left(\frac{1}{\frac{1}{SF} \cdot \left(\frac{N}{N_{FC}} - 1\right) + 1} \cdot (1 - SF) + SF\right) \cdot \sum_{i=1}^{k} (NI_i)}$$

$$= \frac{SF}{\left(\frac{1}{\frac{1}{SF} \cdot \left(\frac{N}{N_{FC}} - 1\right) + 1} \cdot (1 - SF) + SF\right)}$$

$$\Rightarrow Speedup_{RR(N>N_{FC})} = \frac{N_{FC}}{N} \cdot (SF - 1) + 1 \qquad (15)$$

More generally, we can rewrite the speedup formula as follows:

$$Speedup_{RR} = \frac{MIN(N_{FC}, N)}{N} \cdot (SF - 1) + 1 \tag{16}$$

References

[1] A. R. Alameldeen and D. A. Wood. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4), 2006.

[2] M. Becchi and P. Crowley. Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures. In *Proc. of CF '06*, pages 29–40, 2006.

[3] E. Ebrahimi et al. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. ASPLOS '10, 2010.

[4] R. Gabor, S. Weiss, and A. Mendelson. Fairness and throughput in switch on event multithreading. In *Proc. of MICRO '06*, 2006.

[5] M. Gillespie. Preparing for The Second Stage of Multi-Core HW: Asymmetric (Heterogeneous) Cores. *Intel White Paper*, 2008.

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

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

[8] K. Luo, J. Gummaraju, and M. Franklin. Balancing thoughput and fairness in SMT processors. In *ISPASS*, pages 164–171, 2001.

[9] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Proc. of MICRO '07*, 2007.

[10] V. Petrucci, O. Loques, and D. Mossé. Lucky scheduling for energy-efficient heterogeneous multi-core systems. In *Proc. of the 2012 USENIX HotPower'12*, pages 7–7, 2012.

[11] J. C. Saez et al. A Comprehensive Scheduler for Asymmetric Multicore Systems. In *Proc. of ACM Eurosys '10*, 2010.

[12] J. C. Saez et al. Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems. *J. Parallel Distrib. Comput.*, 71:114–131, January 2011.

[13] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading processor. In *ASPLOS*, 2000.