Lucky scheduling for energy-efficient heterogeneous multi-core systems

Vinicius Petrucci, Orlando Loques
Universidade Federal Fluminense, Brazil

Daniel Mossé
University of Pittsburgh, USA

Abstract

Heterogeneous multi-core processors with big/high-performance and small/low-power cores have been proposed as an alternative design to improve energy efficiency over traditional homogeneous multi-cores. We make the case for proportional-share scheduling of threads in heterogeneous processor cores aimed at improving combined energy efficiency and performance. Our thread scheduling algorithm, *lucky*, is based on lottery scheduling and has been implemented using Linux performance monitoring and thread-to-core affinity capabilities at user-level. Our preliminary results show that lucky scheduling can provide better performance and energy savings over state-of-the-art heterogeneous-aware scheduling techniques.

1 Motivation & background

Heterogeneous (or asymmetric) multi-core processors have been proposed as an alternative design to traditional homogeneous multi-cores to improve energy efficiency [8]. Such a heterogeneous platform includes cores with same ISA (instruction set architecture) but different power and performance characteristics. Recent research has shown that two types of cores (*big* high-performance vs *small* low-power) are able to capture most of the power/performance benefits from core heterogeneity while running typical workloads [4, 6, 7].

Having different core types in a multi-core system opens up new challenges and possibilities for power/energy management, thread scheduling and load balancing. Improvements in energy efficiency and performance can be achieved by allowing each thread to run on the core type (big or small) that is best suited for it. Thread-to-core assignment decisions leverage runtime observation of compute-intensive vs memory-intensive execution phases of the threads [5, 8]. An optimal *energy-efficient* thread assignment has to match the thread demands with the capabilities of the heterogeneous cores. We argue that, in addition to satisfying the threads' computational demands, thread scheduling has to provide *fair allocation* of the platform resources to the threads over time.

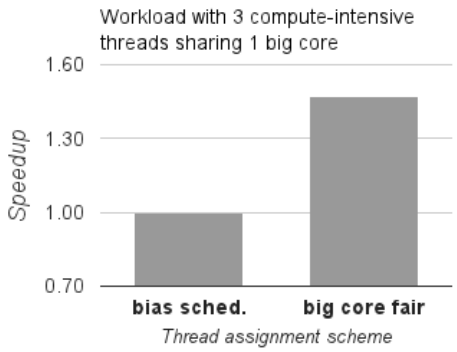

Figure 1: Performance comparison between bias scheduling and big core fair sharing

Previous works have proposed taking advantage of performance counters provided by the hardware to determine the characteristics for the running threads and schedule them on the right core type [7, 12, 14]. To make thread scheduling decisions, these scheduling techniques calculate the thread *bias* through measuring either compute-intensity (IPC or IPS, instructions per cycle or per second) [1, 8] or memory intensity (e.g., LLC miss rate) [7, 12] which determines the thread execution efficiency between the core types. The intuition is that a thread with a high IPS ratio or low-memory intensity bias is expected to take best advantage of a big core rather than a small core.

Most of those scheduling algorithms [1, 7, 8, 12, 13] are unfair by design since they assign to big cores the threads that experience the highest bias towards those cores. As such, some threads can monopolize the available big cores and hinder the progress of other threads. Figure 1 shows the results of an experiment where one memory-intensive and three compute-intensive threads were running on a quad-core multi-core system with one big core and three small cores (Section 2 describes our experimental setup).

For this particular workload (Figure 1), we can observe a performance *speedup* of 1.47 times (energy consumption reduction of 37%) over running bias scheduling [7] by simply providing equally fair sharing of the big core among all threads (25% for each thread). Using bias scheduling, the highest compute-intensive thread had possession of

Core	Peak power	Idle power	Avg. power	Capacity
big	18.75 W	9.625 W	15.63 W	6,307 MIPS
small	2.15 W	0.7 W	1.6 W	1,592 MIPS

Table 1: Power and performance measures of heterogeneous cores

the big core 96% of the workload execution time, leading to severe performance degradation of the other compute-intensive threads. The workload execution time for each scheduling algorithm is the elapsed time of the last completed thread.

In this paper we make the case for proportional-share scheduling of threads to heterogeneous processor cores aimed at improving combined energy efficiency and performance. Our thread scheduling algorithm, *lucky*, is based on abstraction and mechanism of lottery scheduling [15] to provide energy-aware proportional-share of the available big/small cores in the system. The running threads are given some number of tickets derived from runtime performance monitoring. The given tickets of a thread corresponds to its energy efficiency estimate and determine relative chance of all of the other threads competing for the available big cores.

2 Performance-asymmetric core system

We consider a multi-core system having the following core types: big/fast cores targeted for high-performance and small/slow cores optimized for low-power. Such a system with two distinct types of cores is able to capture most of the benefits from heterogeneity [4, 6].

We use the following performance-asymmetric multi-core system in our experiments: a quad-core x86_64 chip capable of individual core frequency scaling where one core runs at $3.2Ghz$ and the other three cores run at $0.8Ghz$. We consider that the die area of a big core is equivalent to three small cores (3:1 ratio) [7]. All cores share a L3 (last level) cache of 6MB and off-chip memory subsystem of 8GB.

2.1 Core power/performance measures

Table 1 describes the power and performance characteristics of the heterogeneous cores in the system. Performance (capacity) of a core is defined as follows. We measured the highest thread IPC, by running the SPEC CPU benchmarks one thread at a time. Multiplying this number by the clock speed, we find out how many instructions can be executed in one second. We use MIPS (million instructions per second) as the measure of core computational capacity/performance.

Since our multi-core system includes identical cores only differing in clock frequency, we estimate core power consumption based on measurements [11] of a heterogeneous platform configuration composed of typical big (Intel Xeon) and small (Intel Atom) cores [4]. In our multi-core system model, a big core delivers 4-fold performance but a small core is 2.2 times more power-efficient (i.e., MIPS per Watt). A small core consumes on average much less power than a big core and is also attractive in terms of the performance obtained in proportion to the power consumed (power-peformance proportionality). This is because of its very low idle power ($0.7W$) and wide dynamic power range between idle and peak power ($0.7W$ to $2.15W$).

2.2 Thread performance characterization

We use Linux 2.6.34 kernel with *perf* monitoring tool to gather hardware performance events namely `retired_instructions` and `L3_cache_misses` in order to characterize the behavior of a thread execution in the multi-core system. We measure LLC misses per core since a LLC event is an event shared across all cores in the system [2]. To individualize per core measures, we set up the Linux *perf* tool to read the 4E1 (L3 Cache Misses) raw register with different core selection masks.

Each core runs a separate thread and thus we monitor the performance counters for a given core (thread) for a given monitoring interval. Given these counters, we compute the *MIPS* (million instructions per second) to determine the CPU demands of a thread. Similarly, we obtain the number of LLC misses per second, *LLCMS*, to characterize memory demands, where each LLC miss represents an off-chip memory request.

We use real performance measurements when a thread is running on the same core type, and assume the next scheduling interval can be approximated by the current interval. To obtain the performance values of a thread running on different core types, previous works [1, 8] require each thread to run on every core type. As noticed in [13], such a direct IPS measurement on different core types poses many practical issues and incurs much overhead. In contrast to those works, we use the equation $MIPS_{j,k} = \alpha_j \cdot MIPS_{i,k} + \beta_j \cdot LLCMS_{i,k} + \gamma_j$ to estimate the performance behavior of a thread k currently running on a given core type i when it is assigned to a different core type j in the next scheduling interval.

The linear regression model above is key to determine the energy efficiency ($MIPS^2/Watt$) of a thread between big and small cores, as introduced in Section 3. The regression coefficients α_j, β_j and γ_j are derived from offline performance data collected running programs from the SPEC 2006 benchmark suite (shown in Figure 2) individually on each core type j. Running the method of least squares to

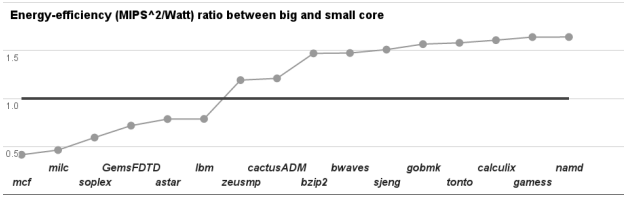

Figure 2: Energy-efficiency ratio (big-to-small core) of threads when running on different core types

the performance data yielded a coefficient correlation of approximately 98%. Note that such a performance characterization needs to be done only once at design stage. Given a set of representative workloads intended to run on the multi-core system, there is no need to recalibrate the parameters of the prediction model. Nevertheless, when substantial changes are observed in the behavior of the workloads, better results can be obtained by updating the coefficients of the prediction model or determining the coefficients online.

3 Lucky scheduling

Lucky scheduling carries out thread assignment to heterogeneous cores for optimized performance and energy savings. The assignment of threads to cores are periodic (*reassignment interval*) to cope with thread execution phase changes. Lucky scheduling builds on the concepts and mechanism from the lottery scheduler [15] to implement such a dynamic scheduling strategy.

The novelty of lucky is in how the ticket assignment is done in the scheduling algorithm. Each thread receives a dynamic number of tickets which is determined by the *energy efficiency* ratio between running the thread on a big core vs small core.

3.1 Energy efficiency metric

The goal of the metric used by *lucky* scheduling to guide thread assignment decisions is to minimize the energy-delay product per instruction given by $Watt/MIPS^2$, which can be rewritten as $(Watt \times S \times S)/I^2 = (Energy \times Delay)/I^2$. Thus lucky scheduling aims to minimize both the energy and the amount of time required to execute thread instructions [3]. In fact, we maximize the inverse of the energy-delay product, given by $MIPS^2/Watt$, to facilitate the computation of the energy efficiency ratio between big and small cores.

Figure 2 shows a comparison of the energy efficiency ratio between big and small cores when running the CPU SPEC 2006 benchmarks on different core types. We can observe that the energy efficiency varies over different programs and can be used for guiding dynamic thread assignment decisions. Although not shown in Figure 2, similar behavior exists for different execution phases of programs [8]. That is, some threads go through compute-intensive or memory-intensive phases where the $MIPS^2/Watt$ changes during the course of their executions [8]. We use this information to periodically reassign the running threads to big or small cores in an energy-efficient manner.

Looking at Figure 2, we observe that the *zeusmp* benchmark should run on a big core, whereas *astar* is best suited on a small core to improve energy efficiency. Bias scheduling [7], for example, would map both programs on a small core type since both experience high memory stalls, about 14 million *LLCMS*. A typical compute-intensive thread has 1-2 million *LLCMS*. We also noted that *bwaves* is an instance of benchmark that bias scheduling would run it on a small core because of its high memory-intensive (29 millions of *LLCMS*). However, *bwaves* is best suited for a big core to improve energy efficiency.

The observations above indicate that explicitly taking into account the core power consumption in thread scheduling can improve energy-efficient scheduling decisions. Furthermore, different power/performance ratios can be observed in the design of heterogeneous multi-core systems. This makes it much more challenging to derive energy-efficient thread assignment decisions based only on either computational demands (MIPS) [1, 8] or memory stalls (LLC misses) [7, 12].

3.2 Algorithm outline

We introduce the following notation for the heterogeneous multi-core system. Let N be the set of core types, N_i be the set of cores of type $i \in N$, and M be the set of all available cores; that is $M = N_1 \cup N_2 \cup \ldots N_n$, where $n = |N|$. Each core of type $i \in N$ has same computational capacity C_i (million instructions per second) and peak/busy power B_i and static/idle power I_i (Watts).

Let K be the set of threads to run in the system. Each thread $k \in K$ requires computational execution rate $MIPS_{i,k}$ and memory access rate $LLCMS_{i,k}$ when executing on core type $i \in N$. We estimate thread power consumption as $P_{i,k} = (B_i - I_i) \cdot (MIPS_{i,k} \ / \ C_i) + I_i$, when thread k runs on a core of type i. We estimate the energy consumption (in Joules) as $E = \sum_{k \in K} P_k \cdot S$, given the configuration of each thread allocated to a core type in a given scheduling interval S (in seconds). The energy efficiency of a thread-to-core assignment is $energy\_efficiency(i,k) = MIPS^2_{i,k} \ / \ P_{i,k}$ in a given scheduling interval.

In *lucky* scheduling, the ticket allocation of a thread $k \in K$ is determined periodically by its energy efficiency ratio between two types of cores: big and small cores (where $N = \{big, small\}$). A given thread is expected to run on a big vs small core proportionally to the number of tickets it holds. More precisely, allocating more tickets to a

thread gives it a higher priority to run on a big core. This allows to adjust dynamically the priority of a given thread currently running on a small core to move to a big core, thereby reducing unfairness and improving overall performance in the system. The activity of always exchanging two running threads between different core types help preserve load balancing.

The initial thread assignment is given by the underlying operating system (OS) scheduler. Typically the OS assigns each thread to a core so that the workload is balanced across the available cores in the system. The algorithm of *lucky* scheduling is outlined as follows:

1. Measure $MIPS_{i,k}$ and $LLCMS_{i,k}$ of each thread $k \in K$ running on a core of type $i \in N$ and predict $MIPS_{j,k}$ on the other core of type $j \in N - \{i\}$ (see Section 2.2)

2. Evaluate $big\_core\_benefit(k) = \frac{energy\_efficiency(b,k)}{energy\_efficiency(s,k)}$ for each thread k, big core b, and small core s

3. Generate a number of $tickets(k) = 100 \cdot big\_core\_benefit(k)$ to assign for each thread k

4. Determine *winning_ticket* as a random number uniformly distributed between $[0, total\_tickets)$ where $total\_tickets = \sum_{k \in K} tickets(k)$

5. Let T be the thread that holds *winning_ticket* and F be the least-loaded big core in the system

6. If T is not running on a big core then *swap* thread T with a thread T' that is running on F and has the minimum number of tickets

We exploit a lottery-based approach to implement *lucky* scheduling because of the *simplicity* of its implementation along with the *flexibility* when including/removing threads in the system. The current implementation of *lucky* scheduling uses a list data structure to keep threads updated with their current number of tickets and a global variable to track the sum of threads' tickets. The random number generator implemented in C++ standard library is used to select the winning ticket/thread.

Since the number of core types is small and fixed, the algorithm complexity is $O(m)$ where m is the number of threads, because it is dominated by searching for a thread with *winning_ticket*. For future many-core systems with hundreds or thousands of cores, a more efficient implementation using balanced tree or heap structures could reduce the complexity to $O(log\ m)$ [15].

4 Preliminary results

We evaluate the performance and energy consumption of **lucky** scheduling against the heterogeneous-aware **fair** share policy [12] and **bias** scheduling [7] using a mix of thread workloads. The fair share algorithm allocates the big core among threads in a round robin fashion. For the bias scheduling, we compute $bias_k = LLCMS_{i,k} \ / \ MIPS_{i,k}$ for each thread k running on a core of type i. Threads with the lowest bias are mapped to the available big cores, whereas the highest bias are mapped to small cores.

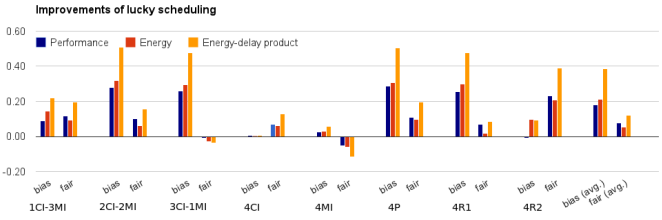

Figure 3: Improvements of lucky scheduling over bias scheduling and big core fair policy

Binding threads to cores of the same type is done via `sched_setaffinity` system call. The Linux scheduler thus makes scheduling decisions respecting this affinity. The reassignment interval is set up to $200ms$ which approximately corresponds to the Linux load balancing granularity, responsible for migrating threads among cores to maximize system utilization [9]. We found this interval suitable to capture thread phase changes while helping mitigate thread migration overhead and cold-cache effect.

The SPEC 2006 benchmark suite was used as the workload on the system. Each SPEC benchmark program is considered a thread in the system and the number of threads is equal to the sum of cores [7]. The benchmarks were selected to test a variety of CPU and memory requirements which are composed of 4-threads mixture varying from compute-intensive (`4CI`) to memory-intensive (`4MI`) workloads [12]. The workload `4P` represent benchmarks that exhibit different phases and workloads `4R1`, and `4R2` are random combinations of threads.

The experiment ends when each thread has run at least once; that is, until the longest thread finishes, also known as *makespan* in the literature. While the longest thread is not finished, the other threads restart their executions as soon as they are finished. We measure the execution time (wall clock) of a workload by calling Linux `gettimeofday` system call at the start and end of the workload execution. We estimate the energy consumption of the running threads in a workload using the formula from Section 3.2. We multiply the sum of estimated energy consumption of all threads executed in the system by the workload execution time to derive the *energy delay product* (EDP).

Figure 3 shows an experiment to demonstrate that inherently unfair thread scheduling leads to more performance loss than improvement for the given mixed set of workloads. When considering all workloads executions, a simple

big core fair policy provides EDP gains of 16% over bias scheduling. Beyond that, lucky scheduling outperforms big core fair policy in EDP by 12% (avg.) and 20% (max.). This indicates that energy-efficient proportional-share brings higher performance/energy improvements rather than simply providing equally fair sharing of the big core among all threads. Lucky scheduling achieved better EDP when compared to bias scheduling over all workloads executions (avg. 39% and max. 51%).

The big core fair policy outperformed lucky scheduling in EDP by 4% in the workload 3CI-1MI and 12% in the workload 4MI. This highlights that more biased/unfair decisions can negatively affect the energy and performance for some workloads. Nevertheless, we show that lucky scheduling was able to provide adequate balance between fairness and core execution bias/efficiency in most cases. Although these results are encouraging, further experiments are required to broaden our understanding of such bias/fairness trade-off decisions.

5 Conclusions & future directions

Energy-efficient heterogeneous multi-core systems pose challenging scheduling problems. In this paper we advocated a proportional-share scheduling strategy based on lottery/ticket mechanisms that optimizes for combined performance and energy savings. We proposed a simple and effective way to determine ticket/thread assignment by estimating thread performance and energy efficiency between core types in the system. Lucky scheduling has shown energy savings and performance improvements over state-of-the-art thread assignment schemes designed for heterogeneous multi-core systems.

Adapting lucky scheduling to support multi-threaded benchmarks and addressing other types of core heterogeneity (e.g., micro-architectural differences) are interesting research directions. To help minimize migration overhead and preserve cache affinity, we may exploit *ticket inflation* to allocate additional tickets to the threads that are already running on a big core, giving more chances to keep those threads running on the same type of core. Sensitivity analysis is needed to determine the best parameter for the ticket inflation and which workloads would benefit from such a scheduling adjustment.

Migrating specific threads between cores with different processing capabilities can also cause performance variability due to contention in the shared resources (last-level cache, memory controller/bus) [10]. Extending lucky scheduling to incorporate *resource contention awareness* to cater to both computational and memory measures in the ticket estimate/assignment is an interesting avenue for future work.

Exploring *thread consolidation* may provide additional energy-savings by maximizing the utilization of a set of active cores and quickly powering down idle cores, bringing them up when thread workloads increase. This would require a better prediction model that incorporates the performance degradation of co-running threads in the system to make effective thread allocation decisions. This would be important when deciding where to allocate a given set of threads that would provide optimized energy efficiency and incur the least performance slowdown.

Acknowledgements. This work is supported by the National Science Foundation, under grant CNS-1012070, and the Brazilian Government agencies CAPES and CNPq. We thank Rami Melhem, Nevine M. AbouGhazaleh and Sameh Gobriel for their constructive feedback on this work.

References

[1] BECCHI, M., AND CROWLEY, P. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Computing frontiers* (2006).

[2] BLAGODUROV, S., AND FEDOROVA, A. User-level scheduling on numa multicore systems under linux. In *Linux Symposium* (2011).

[3] BROOKS, D. M., ET AL. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro 20* (November 2000), 26–44.

[4] CHITLUR, N., ET AL. QuickIA: Exploring heterogeneous architectures on real prototypes. *HPCA'12*.

[5] FEDOROVA, A., SAEZ, J. C., SHELEPOV, D., AND PRIETO, M. Maximizing power efficiency with asymmetric multicore systems. *Commun. ACM 52* (December 2009).

[6] GREENHALGH, P. Big.LITTLE processing with ARM CortexTM-A15 and Cortex-A7. White Paper, 2011.

[7] KOUFATY, D., REDDY, D., AND HAHN, S. Bias scheduling in heterogeneous multi-core architectures. In *EuroSys'10*.

[8] KUMAR, R., FARKAS, K. I., JOUPPI, N. P., RANGANATHAN, P., AND TULLSEN, D. M. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO 36* (2003).

[9] LE, T. M. A study on linux kernel scheduler version 2.6.32. Available online: `http://www.scribd.com/thangmle/d/24111564-Project-Linux-Scheduler-2-6-32`.

[10] MARS, J., VACHHARAJANI, N., HUNDT, R., AND SOFFA, M. L. Contention aware execution: online contention detection and response. In *CGO '10*.

[11] REDDI, J., ET AL. Web search using mobile cores: quantifying and mitigating the price of efficiency. In *ISCA '10*.

[12] SAEZ, J. C., SHELEPOV, D., FEDOROVA, A., AND PRIETO, M. Leveraging workload diversity through os scheduling to maximize performance on single-isa heterogeneous multicore systems. *J. Parallel Distrib. Comput. 71*, 1 (Jan. 2011), 114–131.

[13] SHELEPOV, D., SAEZ ALCAIDE, J. C., JEFFERY, S., FEDOROVA, A., PEREZ, N., HUANG, Z. F., BLAGODUROV, S., AND KUMAR, V. HASS: a scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev. 43* (April 2009).

[14] SRINIVASAN, S., ZHAO, L., ILLIKKAL, R., AND IYER, R. Efficient interaction between os and architecture in heterogeneous platforms. *SIGOPS Oper. Syst. Rev. 45* (February 2011).

[15] WALDSPURGER, C. A., AND WEIHL, W. E. Lottery scheduling: flexible proportional-share resource management. In *OSDI '94*.