Extending the Flexibility of ACMPs for Mobile Devices Using Alternative Cache Configurations

Daniel Nemirovsky[1], Nikola Markovic[1], Osman Unsal[1], Adrian Cristal[1], and Mateo Valero[1]

Barcelona Supercomputing Center, Barcelona 08034, Spain

Abstract. Advances in integrated circuit fabrication technologies and techniques have supported the growth of Chip Multiprocessors (CMP) integrated within mobile devices. Recently, the need to develop faster, smaller, and more energy efficient CPUs has motivated research related to heterogeneous or asymmetric CMPs (ACMPs). These heterogeneous processors offer significant performance and energy efficiency benefits over homogeneous approaches when used in conjunction with scheduling policies effective at managing the resource diversity.

The focus of this work is to highlight how architects can optimize the energy efficiency and size of ACMP systems by using alternative cache configurations. This approach offers a method to tailor an ACMP system to provide suitable performance, energy efficiency, and size for very demanding mobile devices. We propose three alternative cache configurations and examine their effects on system performance and processor size when executing applications concurrently. Our results show that adopting these configurations in conjunction with a scheduler targeting asymmetrical systems can lead to substantial energy savings of over 17%, power reductions of over 5%, and over 19% reductions in physical size while still outperforming execution times achieved with conventional operating system schedulers on a CMP with larger caches by over 10%.

1 Introduction

In order to keep pace with expectations for ever faster, smaller, and more energy efficient processors, hardware architects have needed to rethink traditional architecture designs in terms of performance and power tradeoffs for targeted applications. A silver lining in this regard has come from advances in integrated circuit manufacturing technologies which have provided computer architects with ever increasing transistor densities available per chip. It is nowadays common to find individual chips with several billions of transistors allowing for powerful multi and many core processors.

The homogeneous, or symmetrical, approach employed by many conventional CPUs which bundle together multiple identical computing cores is becoming limited by power dissipation and efficiency concerns. This problem, otherwise known as Dark Silicon [9], has served to highlight the bounds of current design practices and has become a key motivating factor for moving towards heterogeneous computing. Another critical factor driving this trend has been the increase of parallel applications. Apart from running multiple programs in parallel, multithreading enables individual programs to be divided into parts and executed concurrently on separate hardware threads. Furthermore, since workload sizes and computational and memory footprints vary quite significantly from application to application and even from thread to thread, the performance and power benefits of using an heterogeneous over a homogeneous hardware configuration appears intuitive.

In addition to general purpose processors, heterogeneous systems may include a variety of processing elements including GPUs, DSPs, accelerators and sensors, and FPGAs. However, the focus of this paper is on single-ISA heterogeneous processors, also commonly called Asymmetric Chip Multiprocessors and from here on referred to as ACMPs for short. The defining feature of ACMPs is that they are composed of multiple computing cores that share a similar ISA but may be of different sizes (e.g. pipeline stages, issue width), complexities (e.g. in-order vs. out-of-order), and/or run at different frequencies. Examples of the benefits ACMPs can provide are witnessed in such systems as ARM's bigLittle [7] and Nvidia's Tegra3 [25].

An important problem to overcome in order to fully exploit the potential ACMPs offer when running parallel applications is how to optimally schedule threads onto the computational cores. For example, modern operating systems such as Linux may include different levels of software scheduling routines that can be triggered periodically at runtime but are not configured to take advantage of the differences between cores in ACMPs. Recently, however, several novel dynamic scheduling mechanisms have been proposed which trigger thread swaps between cores during execution time resulting in significant performance gains over typical operating system schedulers on ACMPs. A caveat of these new approaches is that most require a substantial amount of implementation overhead and increase the overall power consumption of the system.

In this work we focus on leveraging the performance gains of a simple ACMP scheduler known as the Hardware Round-Robin Scheduler (HRRS) [20] with three alternative cache configurations in order to provide significant size, power, and energy reductions without compromising the speedup benefits gained by the HRRS over an ACMP with a larger cache utilizing a Linux OS based scheduling technique. The benefits gained through these proposals highlight a reasonable method architects may utilize in order to customize ACMPs for mobile devices with specific performance, energy, and size budgets.

Our contributions include:

– Extending the flexibility of ACMPs for mobile devices using an alternative cache configuration technique. We introduce three alternative cache configurations (Larger, Asymmetric, and Distributed) for an ACMP featuring one large core and three small cores. While all three configurations achieve benefits, the Distributed approach is shown to be the most novel and beneficial for ACMP systems that have frequent context swaps.

– Experimental results drawing from both the SPEC2006 and SPLASH-2 benchmark suites which show that alternative cache schemes utilized in conjunction with a simple ACMP scheduler achieves notable energy (over 17%), power (over 5%), physical size (over 19%), and performance (over 18%) benefits over an ACMP containing a larger cache and greatly outperforms an alternative frequency reduction technique.

2 Motivation

The diversity of devices and environments reflect the differing design priorities and choices architects make depending on the target market and usage expectations. For example, though a fitness tracking wristband and a tablet are both mobile devices, they differ in form factor, utilization, and expectations. While a fitness tracker can be expected to last for several days of constant usage running only a small selection of applications, a tablet may be expected to last a day or two but while executing a wider selection of applications much faster than a smaller device but still not as quick as a desktop or server. The main ideas motivating our research has been the need to decide how to balance the design decisions when constrained with specific performance, energy consumption, and area design parameters. In this work, we focus on how a conventional ACMP system may be improved for both energy efficiency and size constraints using a dedicated scheduler in conjunction with a rethinking of the cache hierarchy. Doing so enables us to represent the potential beneficial options a processor architect may choose from in needing to adapt an ACMP processor to different mobile device and usage expectations.

Specifically, our work seeks to improve the energy efficiency of ACMPs by balancing the execution speedups achieved using ACMP schedulers with gains in power and energy efficiency through the use of alternative cache hierarchy configurations. These alternative organizations also help to decrease the footprint of the caches which are the largest elements on the physical chip. This section explores the performance benefits achieved with ACMP schedulers and the energy and size footprint of the conventional ACMP cache hierarchy.

2.1 ACMP schedulers

The vast majority of current CMP scheduling policies focus on mapping software threads to physical cores within systems that contain several identical cores, or Symmetric Chip Multiprocessors (SCMPs). Conversely, Asymmetric Chip Multiprocessors (ACMPs) are much more of a recent development and have yet to become widely adopted. As a result, conventional scheduling policies do not take advantage of differences between computational cores and hence may underutilize the hardware resources. Several scheduling policies have been proposed that target ACMP systems [5, 6, 12] and show significant improvements over the Linux OS pinned scheduler. In this work we will leverage a recently proposed modest yet simple to implement ACMP scheduler known as the Hardware

Round-Robin Scheduler (HRRS). However, it should be noted that while ACMP schedulers can provide substantial speedup benefits, there are various situations where energy efficiency may take greater priority.

Pinned scheduler The pinned scheduling policy is one of the thread scheduling mechanisms implemented in the Linux 2.6 kernel and commonly used in CMP systems. When a software thread is ready to begin execution, the scheduler maps it onto a computational core such that all threads are assigned and all cores are utilized unless there are fewer active threads than number of cores. All threads run to completion on the core they were originally assigned to. If there are more active threads than numbers of cores, then the scheduler proceeds to assign the threads in an overlapping manner. For example, in a two core system with four active threads, the scheduler will map thread A to core 0, thread B to core 1, thread C to core 0, and thread D to core 1. In this case, the scheduler may periodically be activated to swap the active thread on a core. For instance, thread A which is running on core 0 may be swapped with the waiting thread C which is also assigned to core 0. However, the scheduler will not initiate thread swaps between cores, for instance swapping thread A with thread B or D. This mechanism therefore ensures that threads remained pinned onto the computing core they were originally assigned to until they finish execution. A drawback of conventional schedulers such as the Linux OS pinned scheduler which do not swap threads between cores is that they are not able to take advantage of the resource diversity within ACMP systems.

Hardware round-robin scheduler (HRRS) Similar to the fair scheduler, HRRS [20] intends to equalize the amount of time all threads get on the large core. However, since HRRS is a hardware scheduler implementation, it relies on a faster hardware quantum (typically 1ms as opposed to 4ms) and focuses on swapping hardware threads and not software threads. It does so by abstracting the physical hardware and representing it as an SCMP composed of several identical logical cores to the OS. The OS scheduler is not modified and proceeds to schedule threads onto the logical cores. The HRRS mechanism then maps the logical cores onto the physical cores such that the thread chosen by the OS to run on the logical core becomes assigned to a physical hardware thread on a computational core. During each hardware quantum, the HRRS will swap the logical core running on the large core with one running on the small cores chosen in round-robin fashion. If two software threads are assigned to one logical core, then with HRRS they will never be scheduled on different physical cores at the same time which is allowed by the Fair scheduler.

2.2 Comparative performance

The results presented in (Fig. 1) are gathered from simulations based an ACMP system with one large core and three small cores with (processor characteristics described in section 4.1) running the SPEC2006 benchmark suite and the

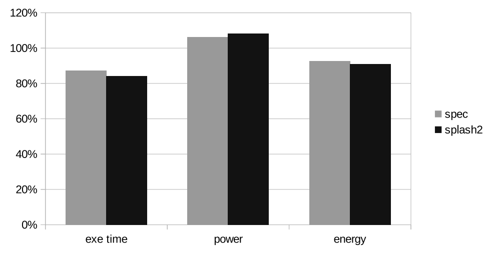


Fig. 1: HRRS scheduler performance (speedup, power, and energy) normalized to the pinned scheduler utilized by the Linux OS when running the SPEC2006 and SPLASH-2 benchmark suites. Lower numbers are better.

SPLASH-2 benchmark suite with the Sniper simulator [3]. The results compare the HRRS performance normalized to the pinned scheduler performance and represent the geometric mean taken from all the applications within the corresponding benchmark suite. Performance is broken down into total execution time, power usage (as a measure of Watts), and energy consumption (as a measure of Joules).

There are clear performance enhancing opportunities of using the HRRS ACMP scheduler over the conventional Linux OS pinned scheduler technique. For SPEC2006, using the HRRS method results in execution speedup gains of nearly 16% and energy savings of almost 10% while requiring about 8% more power. Similar numbers are achieved when running the SPLASH-2 benchmark as it results in about 13% speedup gains and 7% energy savings while requiring 6% more power. The consistent benefits of the HRRS implementation over the pinned scheduler on the ACMP provides added flexibility in the architectural design choices within the processor.

2.3 Cache footprint

The cache hierarchy of the ACMP system when running both SPEC and SPLASH-2 consumes on average about 30% of the total energy and power budget of the processor. The last level L3 cache (LLC) alone consumes a significant chunk of the processor's energy, power, and size budget. Using measurements taken with McPAT when running the 4 core ACMP, the LLC is responsible for on average about 10% of the total execution energy. However, LLC is also responsible for over 20% of the processor's subthreshold leakage power (note that the total processor leakage power was upwards of 25% of the total peak power). The four cores, which include the L1 and L2 cache structures, take up about 67% of the total chip area while the LLC takes up a substantial 32%, and the interconnection network (NoC) a mere 1%. For our simulations based on a 45nm transistor technology, the total area of the processor corresponds to 190 squared millimeters with the LLC taking up about 60 squared millimeters. Along with other alternatives such as frequency reduction, altering the cache configurations is a viable path towards maintaining the speedup gained using HRRS while reducing power and size requirements and increasing energy savings without modifying the internal microarchitecture of the computational cores.

3 Alternative cache configurations

In order to balance the performance gains achieved by ACMP schedulers with size, energy, and power efficiency benefits for the overall ACMP system, we focus on reorganizing the existing cache hierarchy shown earlier to have substantial size and power footprints. The alternate configurations we have proposed and evaluated, which are detailed below, have been configured to eliminate the large last level cache (LLC) and reorganized in order to mitigate the resulting increase in total DRAM accesses. The loss of a shared LLC results in smaller processor area but longer average memory access latencies since more of the memory accesses will end up reaching the DRAM. Additionally, context swaps may incur heftier penalties since the working set sizes of the active threads may not be fully contained or easily transferred between L2 caches. For example, a recently swapped thread may request the data that is still stored in its previous L2 and when it arrives, it may evict the data from the current L2 which could be requested by another thread. In the case of a shared LLC, this evicted data would still reside in the LLC, but with no shared LLC, any requests for this evicted data will have to reach the DRAM. Conversely, the amount of time it takes for a memory request to reach DRAM will slightly decrease by the amount of time it took to perform a tag access on the eliminated L3 cache.

Each of the four cores in every ACMP cache configuration use a private L1 data cache and a separate and private L1 instruction cache. All L2 caches are 8 way associative and only the baseline cache configuration has a shared last level cache. Tag-Data access latencies in cycles are 1-4, 3-8, 10-30 for the L1, L2, L3 caches respectively. The coherency protocol utilized is a directory based MSI.

1. **Baseline** : The baseline configuration consists of a four core ACMP (one large core and three small cores defined in Section 4.1) where each core has a private L1 (32KB) and L2 (256KB) data cache and a shared L3 last level cache (LLC) of 8MB.
2. **Larger** : The most simple approach, this alternative configuration alleviates the loss of the L3 cache by increasing the size of each L2 cache from 256KB to 512KB. Though doubling the overall size of each L2 increases the physical size of each core, we have chosen to use this approach to demonstrate the effects of the most simplistic and intuitive approach at increasing L2 cache hits without resulting in significant energy/power penalties.
3. **Asymmetric** : This heterogeneous cache configuration keeps the size of the small cores' L2 caches steady at 256KB but increases the size of the large core's L2 cache from 256KB to 1MB. Though one of the characteristics of the large core is that it is capable of sustaining more outstanding misses

than the small cores, the extra quantity of cache allows for more data from potentially other threads to be locally available which can enable recently swapped threads to avoid some cache warmup and run faster. This increase in the large L2 cache size and subsequent L2 large to small cache ratio was chosen as a reflection of the large core to small core size and performance capabilities (i.e. the large core is configured to be generally between three to four times more powerful than the small cores).;

4. **Distributed** : This distributive cache proposal intends to emulate a larger shared cache in order to alleviate the adverse memory latency effects caused by eliminating the shared LLC. Similar to the first alternative approach, this proposal doubles each of the cores' L2 caches to 512KB but instead of each L2 being private to each core, they are distributed such that every core can access every L2 cache, albeit with non uniform cache access latencies. Unique to this cache organization, this scheme forbids data to be replicated across L2 caches. Using this method, an L1 miss from core 0 is sent to core 0's L2, if it misses again, then it is sent to core 1's L2, and if it misses there then it is sent to core 2's L2, then to core 3's L2 upon which if it still misses it is sent to DRAM. Every access to the different L2's will result in different access latencies due to the varying distance from core to cache as well as the extra latency penalties for each L2 miss. (e.g. the time for core 0 to access core 1's L2 will be higher than accessing its own L2 but lower than accessing core 3's L2). This is similar to approaches taken in conventional non-uniform cache architectures (NUCA). A memory management unit makes sure that two identical memory accesses from separate cores are not replicated in both of their L2's. The distributed and non-duplicate nature of this approach eliminates the need for cache coherency management and also helps to reduce the costs of cache warm up and pollution effects after every context swap.

4 Methodology

The experimental setup of this work consists of the four cache hierarchy configurations (one baseline and three alternate proposals) to be tested using a fixed processor configuration running applications from two benchmark suites on a parallel multi-core simulator. In addition, these results are compared with an alternative scheme of running the baseline configuration at a lower frequency in order to reduce energy consumption.

4.1 Processor configuration

The processor that is used for all experimental runs in this work is a quad-core ACMP consisting of one large core and three identical small cores. Both types of cores are based on the Intel Nehalem x86 architecture running at 2.66GHz (2.1GHz is used for the lower frequency configuration). Each core type has a 4 wide dispatch width, but whereas the large core has 128 instruction window size, 8 cycle branch misprediction penalty, and 48 entry load/store queue, the small core has a 16 instruction window size, 14 cycle branch misprediction penalty, and a 6 entry load/store queue.

4.2 Benchmarks

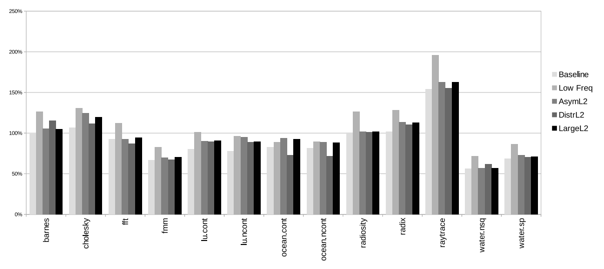
In order to measure the behavior of multithreaded applications running on our system configurations, we have used the popular SPEC2006 and SPLASH-2 benchmark suites. The SPEC2006 benchmark suite is an industry-standardized, CPU-intensive benchmark suite, stressing a systems proces- sor, memory subsystem and compiler. We have utilized the SPEC2006 benchmarks to run multiple instances of the single threaded applications concurrently on the system. We run each workload on the four simulated cores. The SPLASH-2 benchmark suite is composed of eleven different multithreaded workloads focusing on high performance computing, graphics, and signal processing. Each multithreaded benchmark is configured to run with 4 parallel threads and is executed separately. Therefore, at any given time, there will be a maximum of four threads running from one application. All applications or threads are run from start to finish. In the case of having several applications or threads running at the same time, the threads that finish first are restarted such that the number of threads running at any one time on the system remains constant. Once the longest thread/application has been completed, the simulation is ended.

4.3 Simulator

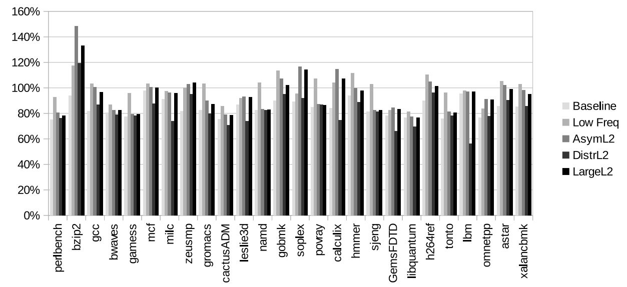
This work uses the Sniper [3] simulation platform. Sniper is a popular hardware-validated parallel x86-64 multicore simulator capable of executing multithreaded applications as well as running multiple programs concurrently. The simulator can be configured to run both homogeneous and heterogeneous multicore architectures and uses the interval core model to obtain performance results. To model power and area, Sniper integrates the Multicore Power, Area, and Timing (McPAT) framework [18]. McPAT is configured in this work to measure results based on a 45nm technology.

5 Experiments and evaluation

Fig. 2b and Fig. 2a reflect the execution time results obtained by running the different configurations on each application of both benchmark suites. The results are normalized to the ACMP system with the baseline cache configuration which utilizes the Linux OS pinned scheduling policy. As expected, the execution time of the alternative cache configurations were slightly slower than the baseline configuration due to the loss of a large share LLC and extra DRAM accesses. However, though execution speeds may take slightly longer using these techniques, they are outweighed by significant gains in energy and space savings. Workload size, instruction length, and memory access patterns of the different applications result in different overheads due to frequent context swaps (due to



(a) SPLASH-2 Results

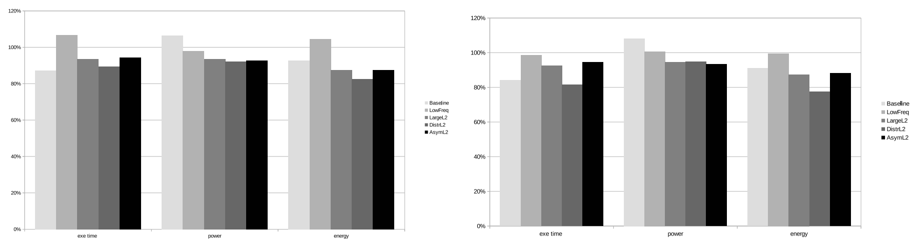


(b) SPEC2006 Results

Fig. 2: Execution time performance results obtained by running all configurations on SPLASH-2 and SPEC2006. Baseline configuration uses HRRS policy. Lower numbers are better.

using the HRRS policy) which explains deviations in application performance running on different cache configurations. For instance, while the frequency reduction scheme consistently underperforms in nearly all benchmarks, the 'Distributed' cache configuration (DistrL2 in the figures) is able to attenuate context swap overheads in applications that suffer from heavier context swap penalties (e.g. ocean in SPLASH-2 and lbm and calculix in SPEC).

Fig. 3b and Fig. 3a illustrate the geometric average for execution time, power, and energy consumption of all SPEC2006 and SPLASH-2 benchmarks for each different cache configuration. The results are also normalized to the ACMP system with the baseline cache configuration which utilizes the Linux OS pinned scheduling policy. The bars in the charts representing the alternative cache configurations all utilize the HRRS scheduling policy. Note that the bar representing the baseline configuration uses HRRS while it is normalized to the baseline configuration utilizing the pinned scheduler. Fig. 4 highlights the difference in processor sizes resulting from the component modifications of the separate cache configurations normalized to the baseline configuration (which includes a shared LLC).



(a) SPLASH-2 Results (b) SPEC2006 Results

Fig. 3: Geometric average for execution time, power, and energy consumption of all cache configurations on SPLASH-2. Lower numbers better.

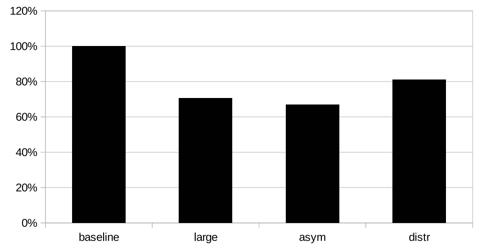


Fig. 4: The resulting processor sizes of each alternative cache scheme normalized to the baseline configuration which includes a LLC. Note that a processor that is 70% the size of the baseline processor can be also read as a 30% reduction in processor size.

The 'Large' cache configuration is able to alleviate extra misses to DRAM by being able to contain more data in the cores' L2 cache and achieves modest average speedup (7.5% for SPEC2006, 6.5% for SPLASH-2), power (5.5% for SPEC2006, 6.3% for SPLASH-2), and energy (12.5% for SPEC2006, 12.5% for SPLASH-2) gains compared to the baseline configuration using the pinned scheduler. Even though the size of the L2's were doubled, the elimination of the L3 results in a 30% reduction in processor size relative to the baseline ACMP configuration.

The 'Asymmetric' cache configuration is an interesting alternative of how to allocate a limited cache budget to different core types. Since it has a larger amount of L2 cache compared with the small cores, the discrepancy between the performance of both core types becomes even greater and since all threads can benefit from its added effectiveness since they evenly share execution time on the large core. This configuration results in adequate average speedup (5.6% for SPEC2006, 5.6% for SPLASH-2), power (6.6% for SPEC2006, 7.2% for SPLASH-2), and energy (11.8% for SPEC2006, 12.4% for SPLASH-2) gains. In terms of size, this cache configuration, which is overall smaller than the 'Large' cache configuration, results in a reduced processor size of 33%.

The 'Distributed' cache configuration produces the most promising results. Since data may not be replicated between L2 caches but all cores may access each others' L2, this setup avoids many of the cache warmup and pollution overheads incurred in the other configurations during the frequent context swaps. The distributed cache also acts as a pseudo-shared L2 LLC such that is four times larger than any individual L2 cache thus increasing the probability of a memory access hit. Moreover, since the HRRS policy essentially provides each thread equal time on all cores, the penalty for non uniform L2 access is shared by all cores which results in what could be considered a shared L2 with shorter access time than the L3 found in the baseline configuration. The gains are significant showing average speedup (18.4% for SPEC2006, 10.4% for SPLASH-2), power (5.1% for SPEC2006, 7.8% for SPLASH-2), and energy (22.6% for SPEC2006, 17.4% for SPLASH-2) benefits. The discrepancies in power savings from this scheme compared to the 'Large' configuration are mainly due to the directory cache coherency mechanism in the 'Large' configuration which is not needed in the 'Distributed' scheme. The distributed configuration results in a total processor size reduction of 19% which is significant but is not as impressive as the other two configurations due to the need for extra resources to provide the extra inter-L2 communication and data access management needed to implement this scheme.

Conversely, the trivial approach of minimizing power and energy by utilizing a reduced frequency with the baseline cache configuration running the HRRS policy produces uninspiring results in speedup (1.4% for SPEC2006, -6.8% for SPLASH-2), power (-0.7% for SPEC2006, 1.9% for SPLASH-2), and energy consumption (1.6% for SPEC2006, -4.7% for SPLASH-2). In addition, the processor size is the same in this case as the baseline configuration processor.

While all three alternative cache configurations show promise in reducing size, power, and energy while maintaining speedup, the results from the 'Distributed' configuration standout. Compared to the baseline configuration running the HRRS it is able to maintain to within 3% the speedup gains achieved by the HRRS running on the baseline configuration while requiring nearly 13% less power and consuming 11% less energy. Overall, the results show that adopting an alternative cache hierarchy in conjunction with a scheduler targeting asymmetrical systems such as HRRS can achieve energy savings of over 17%, power reductions of over 5%, and speedups of over 10% over a ACMP with more cache using a pinned scheduler found in conventional operating systems. In addition, a processor implemented with an alternative cache configuration can provide physical size reductions of 19% up to 33% compared with a conventional ACMP. The benefits of implementing an alternative cache configuration significantly increases the flexibility of an ACMP system and makes the option much more tempting for architects seeking to fit fast and energy efficient processors into ever smaller and more agile mobile devices.

6 Future work

In order to scale to larger many core systems, a cache configuration such as those proposed should be implemented in groups of cores. That is to say that a 16 core system with 4 large cores and 12 small should be broken up into 4 groups of 4 cores (1 large and 3 small). However, if these alternative cache configurations are to be scaled, evaluations on large many core systems including hundreds of cores should also be conducted. Exploring combinations of heterogeneous and distributed cache configurations for ACMPs appears to be a promising research avenue to pursue. More radical alternative cache hierarchy proposals combining asymmetric and distributed cache configurations will need to be studied. Furthermore, different ACMP scheduling strategies can be examined and incorporated into the simulations. Additional performance benefits gained from scheduling mechanisms offer more flexibility in designing alternative cache configurations that can enhance the system's energy efficiency and performance tradeoff.

7 Related work

An ACMP system containing various cores of the same ISA but of different types was proposed by Kumar et al. in [16]. Their process consists of deciding on the core that will perform in the most power efficient manner each time a new phase or program is detected using sampling techniques. This work is expanded in [17] which includes performance maximization of multithreaded applications. Non-uniform cache architecture (NUCA) caches [14], [2] are able can be used to combine several separate cache modules into an emulated large shared cache.

Fairness-aware Scheduler [5] is a software implemented ACMP scheduling mechanism that enhances the pinned scheduler by triggering a software thread swap after a specified software quantum (typically 4ms). The scheduling approach used in Annavaram et al. [1] focuses on staying within a specified power envelope by measuring the energy per instruction of an ACMP running multithreaded applications and running parallel sections of code on the small cores and then migrating to the large cores for the sequential sections.

Due to possible performance and efficiency gains, there has been increasing interest in heterogeneous multicore architectures with various scheduling proposals being recently presented. Grochowski et al. [8] studied the potential of such architectures to save energy and improve throughput. Moncrieff et al. [24] and Menasce et al. [21] analytically examined the tradeoffs between utilizing fast and slow processors in heterogeneous processors. Their study showed that a system composed of few fast cores and many slow cores are effective in terms of cost and performance. Optimal scheduling of independent applications running on a preemptive heterogeneous CMP has been studied by Liu et al. [19]. A scheduling algorithm for an asymmetric system called Single Architecture Heterogeneous Multiprocessor or SAHM was presented in [22] though it did not support multi-programming. Scheduling based power management techniques have been examined in the work by Winter et al. [26] which employ several sampling based algorithms in order to analyze the optimal thread to core mapping.

8 Conclusion

Technology scaling has brought about a recent move towards Asymmetric Chip Multiprocessors (ACMPs) and the development of thread schedulers specific to these new architectures enabling chip designers to be more flexible in their cache configurations in order to reduce power and energy consumption and area. In this work we have proposed three alternative cache configurations and shown how they outperform typical frequency reduction strategies in achieving better energy efficiency. Additionally, these configurations also result in substantial reductions in the physical size of the processor. By utilizing a simple hardware based round-robin scheduler in conjunction with an alternative cache configuration such as a 'Distributed' scheme, it is possible to achieve substantial energy savings of over 17%, power reductions of over 5%, and 19% physical processor size reductions while still outperforming the execution times achieved with conventional operating system schedulers on an ACMP with larger caches by over 10%. These benefits show feasible options architects can consider when choosing which processor designs to implement within demanding performance, energy, and size budgets such as with mobile devices.

References

1. Annavaram, M., Grochowski, E., Shen, J.: Mitigating amdahl‘s law through epi throttling. In: Proc. of the 32nd Ann. Symp. on Comp. Arch. pp. 298–309 (2005)
2. Beckmann, B., Wood, D.: Managing wire delay in large chip-multiprocessor caches. In: Proc. of Int. Symp. Microarchit. pp. 319–330 (2004)
3. Carlson, T.E., Heirman, W., Eeckhout, L.: Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In: Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. pp. 1–12 (2011)
4. Chen, J., John, L.K.: Efficient program scheduling for heterogeneous multi-core processors. In: Proc. Annu. Design Automation Conf. (DAC). pp. 927–930 (2009)
5. Craeynest, K.V., Akram, S., Heirman, W., Jaleel, A., Eeckhout, L.: Fairness-aware scheduling on single-isa heterogeneous multi-cores. In: Proc. 22nd Int. Conf. Parallel Archit. Compilation Tech. pp. 177–187 (2013)
6. Craeynest, K.V., Jaleel, A., Eeckhout, L., Narvaez, P., Emer, J.: Scheduling heterogeneous multi-cores through performance impact estimation (pie). In: Proc. 39th Annu. Int. Symp. Comput. Archit. pp. 213–224 (2012)
7. Greenhalgh, P.: big.little processing with arm cortex-a15 & cortex-a7 (2011), [Online] Available: http://www.arm.com /files/downloads/bigLITTLE_Final_Final.pdf
8. Grochowski, E., Ronen, R., Shen, J., Wang, H.: Best of both latency and throughput. In: Proc. of the Int. Conf. on Computer Design (ICCD). pp. 236–243 (2004)
9. H. Esmaeilzadeh, and E. Blem, and R. St. Amant, and K. Sankaralingam, and D. Burger: Dark silicon and the end of multicore scaling. In: Proc. 38th Int. Conf. International Symposium on Computer Architecture. p. 1 (2011)
10. J. Henning, S.M.: Spec cpu2006 benchmark descriptions. In: Proc. of of the ACM SIGARCH Computer Arch. News. pp. 1–17 (2006)

11. Joao, J., Suleman, M.A., Mutlu, O., Patt, Y.N.: Bottleneck identification and scheduling in multithreaded applications. In: Proc. 17th Int. Conf. Architectural Support Program Languages Operating Syst. pp. 223–234 (2012)
12. Joao, J., Suleman, M.A., Mutlu, O., Patt, Y.N.: Utility-based acceleration of multithreaded applications on asymmetric cmps. In: Proc. 40th Annu. Int. Symp. Comput. Archit. pp. 154–165 (2013)
13. Jones, M.: Inside the linux 2.6 completely fair scheduler (2009), [Online] Available: http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/l-completely-fair-scheduler-pdf.pdf
14. Kim, C., Burger, D., Keckler, S.: An adaptive, non-uniform cache structure for wire-dominated on-chip caches. In: Proc. of Int. Conf. Architectural Support Program Languages Operating Syst. pp. 211–222 (2002)
15. Kim, C., Burger, D., Keckler, S.: Nuca: a non-uniform cache access architecture for wire-delay dominated on-chip caches (2003)
16. Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., Tullsen, D.M.: Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In: Proc. 36nd Annu. IEEE/ACM Int. Symp. Microarchit. p. 81 (2003)
17. Kumar, R., Tullsen, D.M., Ranganathan, P., Jouppi, N.P., Farkas, K.I.: Single-isa hterogeneous multi-core architectures for multithreaded workload performance. In: Proc. 31st Annu. Int. Symp. Comput. Archit. p. 64 (2004)
18. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: Mcpat: An integrated power, area, and timing modeling framework for multicore architectures. In: Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchit. pp. 469–480 (2009)
19. Liu, J., Yang, A.: Optimal scheduling of independent tasks on heterogeneous computing systems. In: Proc. of the 1974 annual conference. pp. 38–45 (1974)
20. Markovic, N., Nemirovsky, D., Unsal, O., Cristal, A., Valero, M.: Hardware round-robin scheduler for single-isa asymmetric multi-core. In: Proc. Euro-Par. pp. 122–134 (2015)
21. Menasce, D., Almeida, V.: Cost-performance analysis of heterogeneity in supercomputer architectures. In: Proc. of the 4th Int. Conf. on Supercomputing. pp. 169–177 (1990)
22. Miller, L.: A heterogeneous multiprocessor design and the distributed scheduling of its task group workload. In: Proc. of the 9th Ann. Symp. on Comp. Arch. pp. 283–290 (1982)
23. Mishra, A., Vijaykrishnan, N., Das, C.: A case for heterogeneous on-chip interconnects for cmps. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. pp. 389–400 (2011)
24. Moncrieff, D., Overill, R.E., Wilson, S.: Heterogeneous computing machines and amdahl's law. Parallel Computing 22, 407 – 413 (1996)
25. NVIDIA: Tegra 3 (kal-el) quad-core mobile processor (2011), [Online] Available: http://www.nvidia.com/object/tegra-3-processor.html
26. Winter, J.A., Albonesi, D.H., Shoemaker, C.A.: Scalable thread scheduling and global power management for heterogeneous many-core architectures. In: Proc. Int. Conf. Parallel Archit. Compilation Tech. (PACT). pp. 29–40 (2010)
27. Woo, S., Ohara, M., Torrie, E., Singh, J.P., Gupt, A.: The splash-2 programs: characterization and methodological considerations. In: Proc. 22nd Annu. Symp. Comput. Archit. pp. 24–36 (1995)