The Forgotten ‘Uncore’: On the Energy-Efficiency of Heterogeneous Cores

Vishal Gupta[*] Paul Brett[†] David Koufaty[†] Dheeraj Reddy[†] Scott Hahn[†]
Karsten Schwan[*] Ganapati Srinivasa[‡]

[*]*Georgia Tech* [†]*Intel Labs* [‡]*Intel Corporation*

Abstract

Heterogeneous multicore processors (HMPs), consisting of cores with different performance/power characteristics, have been proposed to deliver higher energy efficiency than symmetric multicores. This paper investigates the opportunities and limitations in using HMPs to gain energy-efficiency. Unlike previous work focused on server systems, we focus on the client workloads typically seen in modern end-user devices. Further, beyond considering core power usage, we also consider the ‘uncore’ subsystem shared by all cores, which in modern platforms, is an increasingly important contributor to total SoC power. Experimental evaluations use client applications and usage scenarios seen on mobile devices and a unique testbed comprised of heterogeneous cores, with results that highlight the need for uncore-awareness and uncore scalability to maximize intended efficiency gains from heterogeneous cores.

1 Introduction

Energy-efficiency remains a critical concern for both mobile devices and server systems. To improve energy-efficiency while providing high-performance, chip vendors have adopted heterogeneous multicore processors (HMPs). Examples include Variable SMP from NVIDIA [1] and Big.LITTLE processing from ARM [4]. This work focuses on HMPs consisting of a mix of cores that expose the same instruction-set-architecture (ISA), but differ in their power/performance characteristics. HMPs make it possible for different applications within a diverse mix of workloads to be run on the ‘most appropriate’ cores [3, 5, 6, 7]. For example, applications that do not produce a result that is time critical to the user or that are I/O heavy, can be run on low-power small cores, while compute-intensive threads or applications with their output visible to the user, such as browsing, can be allocated to high-performance big cores.

Previous work on heterogeneous processors has primarily focused on core power [5, 7], but modern multicore processors also contain *uncore* subsystem (see Figure 1), with components like the last level cache, integrated memory controllers, etc. With growing cache sizes, increasing complexity of the interconnection network, various core power optimizations, and the integration of SoC (system-on-a-chip) components on CPU die, the uncore is becoming a significant power component in total SoC power [8]. For energy-efficient operation, therefore, it becomes increasingly important to account for uncore while executing on heterogeneous cores.

This paper investigates the importance of uncore power on the energy-efficiency of heterogeneous multicore platforms. Unlike previous work on heterogeneous processors focused on server workloads [3, 6, 7], it targets client devices where energy is a premium resource and workload profiles are diverse. Since server workloads are not representative of the usage model of client devices, it characterizes the behavior of a diverse set of real-world client applications which are typical of end-user mobile devices and describes different ways in which they can exploit heterogeneity. Using these workloads, it further analyzes the impact of heterogeneity on workload performance and energy-efficiency, including both core and uncore components.

Experimental evaluations use a unique, experimental, heterogeneous multicore platform, comprised of both high and low power cores operating in a shared coherence domain. Results demonstrate that heterogeneous core architectures can provide significant performance improvements while also lowering energy consumption for a diverse set of applications when compared to homogeneous processor configurations. They also demonstrate that potential savings are strongly affected by the ‘uncore’ contribution, which motivates the need for uncore-awareness in managing heterogeneous multicore platforms and a scalable uncore design to completely realize the intended gains.

2 Beyond Core: Uncore

2.1 What is uncore?

The uncore is a collection of components of a processor not in the core but essential for core performance. The CPU core contains components involved in executing instructions, including execution units, L1 and L2 cache, branch prediction logic, etc. Uncore functions include the last level cache (LLC), integrated memory controllers (IMC), on-chip interconnect (OCI), power control logic (PWR), etc. as shown in Figure 1. With growing cache sizes and the integration of various SoC components on CPU die, the uncore is becoming an increasingly important contributor to total SoC power.

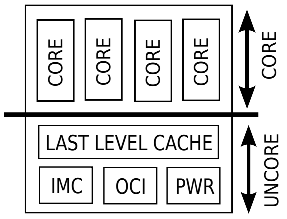


Figure 1: Core and uncore in multicore processors

2.2 Idle State Coordination

Modern multicore processors contain core idle states (C-states) to progressively turn off components in order to conserve power. These C-states are denoted as Cx, where x is a digit. C0 is the active C-state when processor is executing instructions, while a higher numbered C-state (e.g., C2) is a *deeper sleep* state consuming lesser power.

Package		Core 1		
PCx		C0	C1	C2
Core 0	C0	PC0	PC0	PC0
	C1	PC0	PC1	PC1
	C2	PC0	PC1	PC2

‘

Table 1: Core and package idle state coordination

In addition to core C-states, processors also contain package idle states (PCx states) that govern uncore power consumption. These package C-states are related to core C-states in that the processor can only enter a low-power package C-state when all of the cores are ready to enter that same core C-state. Table 1 shows this coordination of core and package idle states for a two-core system with three idle states. The resultant package C-state is always the lower of the two core C-states. Thus, the uncore subsystem remains active and consumes power as long as there is any active core on the CPU.

2.3 Impact of uncore

Figure 2 illustrates the impact of uncore power on the energy consumption of an application executing on heterogeneous cores. A big core running an application finishes its execution faster and enters a low-power idle state. The same application when executed on a small core takes longer (t_{small}) to finish, which also keeps the uncore active for a longer period of time. If uncore power is substantial in comparison to core power, then the energy gains from running on a small core can be strongly affected by the uncore power. For such a system, energy-efficiency gains from small core execution may be offset by the increase in uncore energy consumption due to longer execution time. This observation is in line with prior work that highlights the tradeoff between CPU and system-level power reduction in the context of frequency scaling [9].

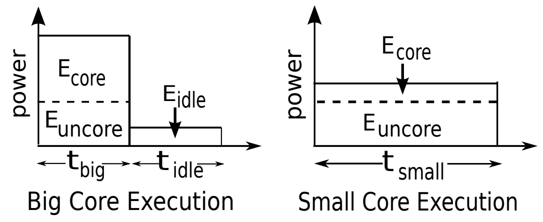


Figure 2: Effect of uncore power on the energy-efficiency of heterogeneous cores

Energy consumption for big and small core execution for such platforms can be modeled using Equations 1 and 2, respectively. Here, E refers to the energy consumed, t denotes execution time, and P_{core} and P_{uncore} represent average core and uncore power, respectively. P_{idle} is the idle platform power, and t_{idle} is the corresponding idle time, as shown in the figure.

$$E_{big} = t_{big} * (P^{big}_{core} + P^{big}_{uncore}) + P_{idle} * t_{idle} \quad (1)$$

$$E_{small} = t_{small} * (P^{small}_{core} + P^{small}_{uncore}) \quad (2)$$

To understand the impact of uncore power, the analysis in Section 4 considers two uncore configurations: fixed and scalable. The fixed uncore configuration uses the same uncore subsystem when executing on either big or small cores. The scalable uncore scenario models an uncore where certain uncore components are turned off or powered down as we move to the small core. For example, fewer memory channels, memory controllers, or a smaller cache can be used with a slow small core that imposes smaller resource requirement on the cache and memory subsystem. Hence, in this case, the uncore power scales along with core power when a workload moves to a different core.

Workload	Description	Metric
browse	loads a set of web-pages at an interval of 3 sec. to emulate user's think time	Load time
javascript	Javascript benchmark performs a series of standard browser operations	Load time
palbum	photo-album application that flips through photographs at 0.5 sec. interval	Load time
mplayer	a H/W accelerated version of mplayer plays an HD movie clip	FPS
mytube	plays an H.264 video inside the browser for 120 seconds	FPS
openarena	plays a benchmarking demo from a 3D first-person-shooter game	FPS
strike	replays a demo session of a web-based 2D game (120 sec.)	FPS
7zip	a parallelized version of 7zip compress a text file using LZMA compression	Time
eclipse	Java based benchmark runs performance tests for the Eclipse IDE	Time
filescan	I/O intensive workload that scans through the Linux source tree	Time
gmagick	GraphicsMagick image editing application is used to resize a set of images	Time
x264	x264 media encoder is used to encode a media file	Time

Table 2: Client workload summary

3 Client Workloads

To assess the viability of using heterogeneity for client systems, we choose a diverse set of real-world applications which are typical of modern end-user devices since prior server-centric research on heterogeneous processors [3, 6, 7] does not directly address the needs and processor usage models seen on client devices. Table 2 provides a summary of the applications used in our analysis and relevant performance metrics. This section categorizes these applications based on their behavior and discusses opportunities for exploiting heterogeneity.

Intermittent Workloads: Client devices like cellphones and tablets are typically powered on for long periods of time, but often perform their heavy processing in short bursts. Web-browsing is an example of such usage, and workloads browse and palbum in Table 2 belong to this category. A timeline trace of IPC (instructions-per-cycle) for the browse workload is shown in Figure 3(a). Idle periods are marked by low IPC periods, while page-loads correspond to spikes in the graph. Since page-loads generate high IPC activity, a big core can be used for rendering the pages and improving page-load performance, while resorting to a small core during low activity periods to conserve power.

Sustained Workloads: Sustained workloads differ from intermittent workloads in that their behavior is uniform over a longer duration. They can be further classified into two sub-categories: sustained-high and sustained-low.

Sustained-low: Client applications like gaming and media playback typically run for a long duration (a few minutes to hours). Moreover, the wide adoption of accelerators allows them to offload significant portions of their computation to accelerators. Figure 3(b) shows the IPC trace of the openarena gaming benchmark. As the observed IPC is low for the application, it can be run on a small core without significant degradation in performance and at lower power (see results in Section 4).

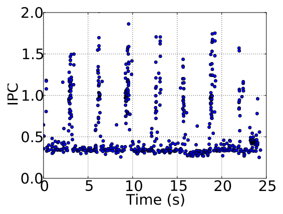
Sustained-high: Mobile devices are also used for compute-intensive tasks such as media encoding, video editing etc. These applications typically have a high IPC (e.g., see x264 encoder in Figure 3(c)), and their performance scales well on a big core. This makes big cores suitable for these applications when they require high performance, e.g., when they are user-facing, while a small core may provide higher energy-efficiency when they run in background mode (e.g, virus-scan).

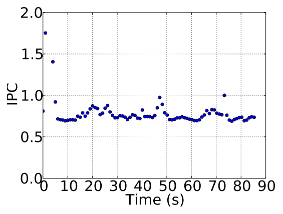
Multi-threaded Workloads: Increasing core count and parallelization of applications on mobile devices present additional opportunities for exploiting heterogeneity. 7zip, gmagick, and eclipse workloads are examples of parallel applications. Similarly, the mytube workload also uses multiple threads for media decoding and rendering. Figure 3(d) highlights heterogeneity in the mytube workload as various threads within the application differ significantly in their IPC. Since such threads differ in their behavior, heterogeneity can be leveraged by appropriate task scheduling.

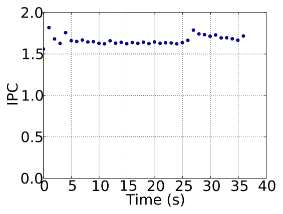
4 Experimental Evaluation

4.1 Testbed

Our experimental platform consists of a quad-core Intel i7-2600 client processor. To create heterogeneity, we use proprietary Intel tools to defeature a subset of the cores in order to emulate the performance of low-powered small cores [6]. A block diagram of the platform configuration is shown in Figure 4. An on-die graphics processor is used to accelerate graphics workloads. All of the cores operate at a frequency of 2.6 GHz and share an LLC of size 8 MB. All the workloads are run using Linux kernel 3.0 and automated using scripts.


(a) Intermittent (browse)


(b) Sustained-low (openarena)


(c) Sustained-high (x264)

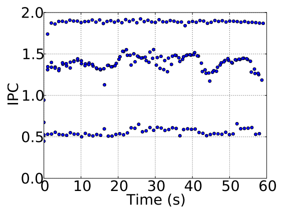

(d) Multi-threaded (mytube)

Figure 3: Diverse client workload profiles (IPC vs. Time)

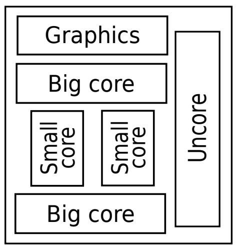


Figure 4: Experimental heterogeneous platform

4.2 Methodology

Experimental evaluation and analysis are carried out as the multiple steps summarized below.

- Each workload is first evaluated on a system configured to use only big cores. Multi-threaded applications are configured for a one to one mapping of threads to big cores.
- Next, the workloads are run using only small cores.
- The metrics collected include: application performance, IPC, LLC accesses, and various core and package C-state residencies.
- With the help of data collected in the previous steps and the power models described in Section 4.3, we calculate the performance improvement and the energy savings of using small vs. big cores.

Our analysis assumes the use of big or small cores for the entire application run. The implementation and evaluation of a dynamic scheduling algorithm for client devices remains part of our future work.

4.3 Power Model

The emulated heterogeneous platform mimics the performance of small cores. However, it does not match the power characteristics of an actual small core built using a different process technology for low power consumption. We, therefore, rely on power models to obtain core and uncore energy consumption.

4.3.1 Core Power

The average power consumption of a CPU core can be modeled using the following equations:

$$P_{core} = R_{active} * P_{active}^{core} + R_{idle} * P_{idle}^{core} \quad (3)$$
$$P_{active}^{core} = C_{dyn} * V^2 * f \quad (4)$$

Here, R_{active} and R_{idle} denote core active and idle state residencies (%), and P_{active}^{core} and P_{idle}^{core} are the corresponding power values. C_{dyn} is the dynamic capacitance, V denotes the operating voltage, and f represents the switching frequency. Big core C_{dyn} is modeled as a function of IPC in Equation 5, as shown and validated by other researchers [10]. Similarly, Equation 6 models the capacitance for a small core having three-times smaller area than that of the big core.

$$C_{big} = 0.499 * ipc_{big} + 0.841 \quad (5)$$
$$C_{small} = 0.472 * ipc_{small} + 0.176 \quad (6)$$

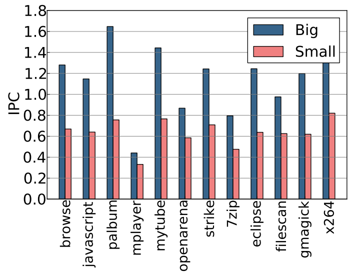
4.3.2 Uncore Power

Similar to core power, uncore power is modeled using package idle state residencies (U_x) as shown below:

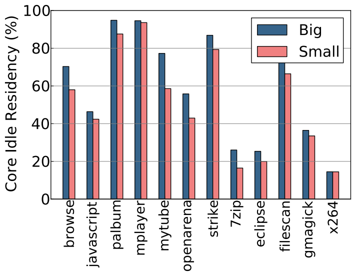
$$P_{uncore} = U_{active} * P_{active}^{uncore} + U_{idle} * P_{idle}^{uncore} \quad (7)$$
$$P_{active}^{uncore} = P_{wake} + P_{activity} * LLC_{rate} \quad (8)$$

Further, uncore active power (P_{active}^{uncore}) is modeled as a function of LLC activity in Equation 8 where P_{wake} is the fixed power cost of waking up various uncore components, while the $P_{activity}$ component scales with the LLC access rate LLC_{rate} (relative to peak access rate including both cache hits and misses).

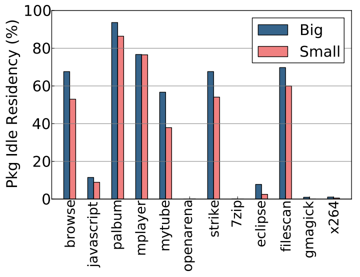
The analysis uses a value of 0.9 V for the voltage (V), and frequency (f) is kept at 2.6 GHz. For this platform, the average big core and small core power for all our workloads is obtained to be 2.37 W and 0.95 W respectively. A comparable uncore is modeled using a value of 1.2 W for P_{wake} and $P_{activity}$ in case of a fixed uncore and scaled down to half for a scalable uncore. Core and uncore idle power are assumed to be 0.1 W and a 1.5 W power component is attributed to the on-die graphics processor which also scales with the LLC activity.



(a) Instructions-per-cycle

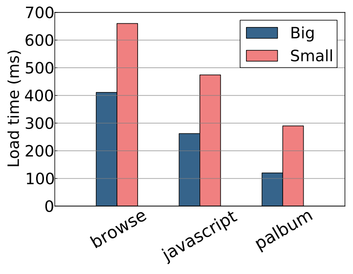


(b) Core Idle Residency

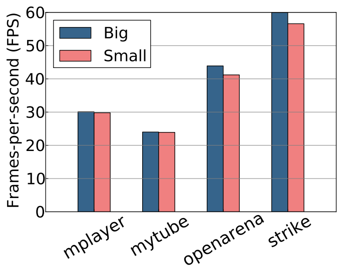


(c) Package Idle Residency

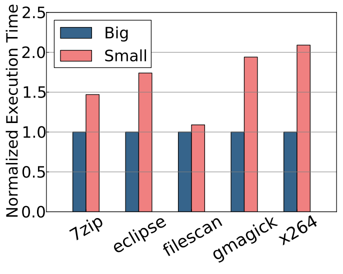
Figure 5: A comparison of the behavior of several client workloads on big vs. small cores



(a) Load time (ms)



(b) Frames-per-second (FPS)



(c) Normalized Execution Time

Figure 6: Application performance comparison on big and small cores

4.4 Results

The results shown in Figure 5 provide a comparison of application behavior on heterogeneous cores. Specifically, they compare average IPC (instructions-per-cycle), core idle residency, and package idle state residency for all of the workloads in Table 2, for big and small core execution. As evident from Figure 5(a), most of the applications observe a significant decrease in their IPC when running on the small vs. big cores. This reduction in IPC results in the small cores being active for longer durations, thereby causing a decrease in core and package idle residency (see Figures 5(b) and 5(c)). Further, many applications are seen to have almost negligible package idle residency. These applications either heavily use the graphics processor (e.g., openarena), or they always keep one of the cores busy (e.g., 7zip, gmagick, x264), and thus do not allow the uncore to enter into an idle state.

The results shown in Figure 6 evaluate the impact on performance of using heterogeneous processors for various client applications in Table 2, categorized by the respective performance metrics. Figure 6(a) compares the average load-time for the browse, javascript, and palbum workloads. We see that the latency is significantly decreased for these applications when using a big core. Thus, a big core provides a notable performance boost for such intermittent applications. In contrast and as depicted in Figure 6(b), when considering the frames-per-second (FPS) metric for various graphics and media applications, we see only minor performance degradation on a small core, at levels not perceivable to end-users. Therefore, they can be run on a small core, to gain potential decreases in energy consumption (discussed further below). The last graph (see Figure 6(c)) compares the normalized execution times for various applications. All of the applications except filescan in this category show a significant improvement in performance with the big core.

Energy savings results computed based on our power models are shown in Figure 7. The figure shows savings for three configurations: core-only savings (C), total SoC-wide savings (C+UC) with a fixed uncore, and with a scalable uncore. As seen in the figure, all of the applications show significant gains on a small core in terms of core energy savings. The palbum application has the lowest savings of 17.58%, while openarena has the largest savings of 52.79%. However, these savings are strongly affected when the power consumption of the uncore is taken into account. Some applications even exhibit negative energy savings. On the other hand, when a scalable uncore is used, these savings increase and become comparable to core-only energy savings. Further, Figure 8 shows the relative contribution of core and uncore energy consumption for all the applications dur-

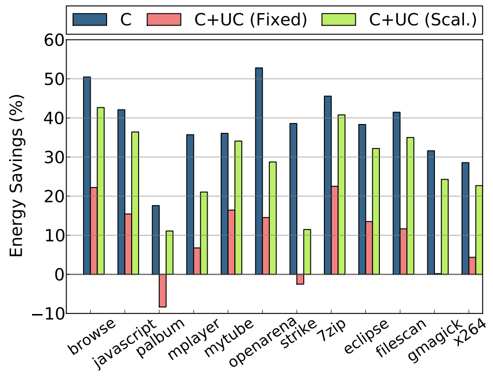


Figure 7: Energy savings of small core execution over big cores for three configurations: core-only savings (C), SoC-wide savings (C+UC) with a fixed uncore, and with a scalable uncore

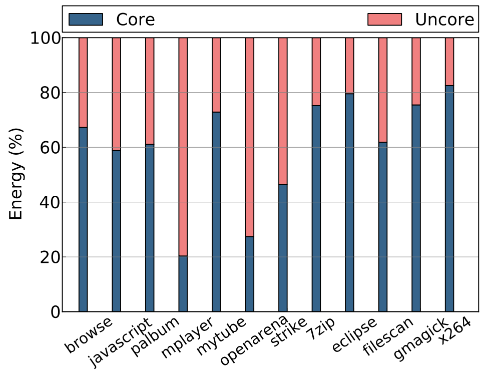


Figure 8: Core and uncore energy contribution

ing big core execution, on a fixed uncore configuration. These results include graphics power in the uncore component. As evident, CPU-intensive applications (e.g., 7zip, gmagick, x264) show a significant core power component, while the uncore fraction dominates for other applications like openarena and mplayer. These results not only demonstrate the importance of taking uncore power into account for scheduling operations, but they also motivate the need for a scalable uncore design to obtain large gains from heterogeneous multicores.

5 Related Work

Substantial prior work has proposed the use of heterogeneous processors to improve the energy efficiency of multicore platforms [3, 5, 7]. Researchers have developed appropriate scheduling algorithms to efficiently run applications on heterogeneous cores [3, 6]. Further, the cost of uncore resources in many-core processors has been modeled and analyzed [8]. In addition, arguments have been made in favor of low-powered cores for the design of datacenters (e.g., FAWN [2]).

In comparison, our work targets client devices where energy is a premium resource, with diverse application behavior and performance metrics. In this context, we highlight the significance of uncore power in total SoC power and analyze its impact on the energy efficiency of several real-world client applications.

6 Conclusions & Future Work

This paper investigates the impact of uncore power on the energy-efficiency of heterogeneous multicore processors for client devices. Using a diverse mix of emerging client applications and an experimental heterogeneous platform, we show that heterogeneous core architectures can provide significant performance and energy gains over homogeneous configurations for client devices. Further, we highlight the growing importance of uncore power with respect to total platform power consumption, thereby motivating the need for uncore-awareness and a scalable uncore design for energy-efficient execution on heterogeneous multicore platforms.

Our future work is investigating client-centric energy-aware scheduling algorithms for heterogeneous multicores. Another interesting venue for research would be to investigate the ideal ratios between the number of big and small cores for different client systems.

References

[1] Variable SMP: A multi-core CPU architecture for low power and high performance. White paper, NVIDIA Corporation, 2011.

[2] Andersen, D. G., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. FAWN: a fast array of wimpy nodes. In *Proceedings of the SOSP* (2009), ACM.

[3] Fedorova, A., Saez, J. C., Shelepov, D., and Prieto, M. Maximizing power efficiency with asymmetric multicore systems. *Commun. ACM 52*, 12 (Dec. 2009), 48–57.

[4] Greenhalgh, P. Big.LITTLE processing with ARM CortexTM-A15 & Cortex-A7. White paper, ARM, Sept 2011.

[5] Hill, M. D., and Marty, M. R. Amdahl's law in the multicore era. *Computer 41*, 7 (July 2008), 33–38.

[6] Koufaty, D., Reddy, D., and Hahn, S. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th EuroSys* (Paris, France, 2010), ACM, pp. 125–138.

[7] 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 *36th MICRO* (San Diego, CA, USA, 2003), pp. 81—-.

[8] Loh, G. H. The cost of uncore in throughput-oriented many-core processors. In *In Proc. of Workshop on Architectures and Languages for Throughput Applications (ALTA)* (2008).

[9] Miyoshi, A., Lefurgy, C., Van Hensbergen, E., Rajamony, R., and Rajkumar, R. Critical power slope: understanding the runtime effects of frequency scaling. In *ICS* (2002).

[10] Spiliopoulos, V., Kaxiras, S., and Keramidas, G. Green governors: A framework for continuously adaptive DVFS. In *Green Computing Conference (IGCC)* (July 2011).