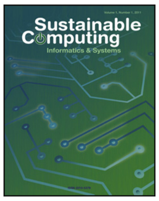
Sustainable Computing: Informatics and Systems 3 (2013) 194–206

Contents lists available at SciVerse ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom



Core groups: System abstractions for extending the dynamic range of client devices using heterogeneous cores

Vishal Gupta[a,*], Paul Brett[b], David Koufaty[b], Dheeraj Reddy[b], Scott Hahn[b], Karsten Schwan[a], Ganapati Srinivasa[b]

[a] *College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, USA*
[b] *Intel Corporation, Hillsboro, OR 97124, USA*

ARTICLE INFO

Article history:
Received 4 September 2012
Accepted 29 January 2013

Keywords:
Heterogeneous multicores
Client devices
Dynamic range
Uncore power

ABSTRACT

Mobile devices and applications exhibit highly diverse behavior in their usage and power/performance requirements. In order to accommodate such diversity, this paper presents 'HeteroMates' system that uses heterogeneous processors to extend the dynamic power/performance range of client devices, i.e., offer both high performance and reduced power consumption. It proposes *core group* abstraction that groups a small number of heterogeneous cores to form a single execution unit. Group heterogeneity is exposed as multiple *heterogeneity (H) states*, an interface similar to the P-state interface already used for frequency scaling. Further, the core group abstraction is extended to a *multicore group* to allow multiple cores within a group to be active concurrently. Also demonstrated is the importance of 'uncore' power in total SoC power consumption and the need for uncore-aware operation and uncore power scalability when seeking to extend a platform's dynamic power/performance range using heterogeneity. Experimental evaluations use real-world client applications and a unique experimental testbed comprised of heterogeneous cores and a shared uncore component. Results show that HeteroMates can provide significant performance improvements while also lowering energy consumption for a diverse set of applications when compared to homogeneous processor configurations.

Published by Elsevier Inc.

1. Introduction

Mobile devices have emerged as a dominant computing platform for end users, resulting in an unprecedented increase in the range of performance demands imposed on them by their many rich applications, and at the same time, battery life and energy efficiency remain critical concerns. Yet modern processors are typically designed to meet only one, not both, of these two conflicting goals of performance vs. efficiency. In response, chip vendors have adopted heterogeneous multicore processors (HMPs) as their platforms of choice, which consist of cores with different performance/power characteristics. Examples include Variable SMP from NVIDIA [1] and Big.LITTLE processing from ARM [2]. HMPs make it possible for different applications within a diverse mix of workloads to be run on the 'most appropriate' cores [3–6]. For example, applications not time critical to the user can be run on low-power small cores, while applications with their outputs visible to the user can be allocated to high-performance big cores.

In addition to performance and energy-efficiency goals, the processors used in mobile platforms also need to operate under stringent power constraints, in part due to the devices' limited cooling options. At the same time, growing transistor counts under such limited budgets, coupled with poor per-transistor power scaling, are not only creating a *utilization wall* that limits the fraction of a chip that can run at full speed at one time, a trend also known as *dark silicon* [7], but they also create new opportunities for realizing application performance goals under stringent power constraints.

This paper presents the *HeteroMates* solution that utilize such 'excess' silicon by intentionally over-provisioning the CPU using heterogeneous cores in order to provide a wider *dynamic power-performance range* for client devices, to meet both their high-performance and low-power demands. Within a heterogeneous mix, it opportunistically uses the right set of cores for execution to meet the diverse requirements of mobile applications. Due to over-provisioning, only a subset of the cores are activated for concurrent use that fit within the power budget, while the rest are forced into inactive mode to save power. The presence of heterogeneity allows the system to make optimal use of constrained power resources, by matching heterogeneous CPU resources to current application needs.

* Corresponding author at: 266 Ferst Drive, Atlanta, GA 30313, USA.
Tel.: +1 404 385 1353; fax: +1 404 385 2295.
E-mail address: vishal@cc.gatech.edu (V. Gupta).

2210-5379/$ – see front matter. Published by Elsevier Inc.
http://dx.doi.org/10.1016/j.suscom.2013.01.010

HeteroMates forms execution units using the *core group* abstraction, where each group consists of a small number of (e.g., 2–4) heterogeneous cores. Cores within a core group are exposed to the system as multiple *heterogeneity (H) states*, similar to the P-states used for voltage and frequency scaling. An *H-state controller* module performs H-state transitions based upon workload behavior and user-defined policies. Depending on the selected H-state, the workload is transparently migrated to the appropriate core by a *core switcher*. The work also highlight the growing importance of uncore power in total SoC power consumption and the need for a scalable uncore in order to fully realize the potential gains obtained from the use of heterogeneity. To this end, we augment the H-state controller for uncore-aware operation by adding *energy-override* condition and thus improve energy-efficiency. H-state abstraction decouples heterogeneity from scheduling such that the scheduler perceives only homogeneous cores. The performance/power differences among cores are transparently handled by a separate H-state driver. H-states can be implemented in hardware, firmware, or software, thereby providing a way to hide heterogeneity from the operating system to support legacy software for wider adoption. Further, core groups allow the system to easily accommodate a variable number of different heterogeneous cores, by adding an H-state for each core. Finally, core groups can be useful in thermal-constrained scenarios (also known as *dark silicon* [7]) which allow only a fraction of the chip components to be active simultaneously.

The core group abstraction is extended to multiple concurrently executing cores using *multicore groups*. A multicore group is a set of different multicore configurations of a platform by choosing a subset of the heterogeneous cores that meet the power budget. The platform's activity in different CPU configurations is exposed using *multicore* states (M-states). Each M-state represents a different platform configuration and a switch between two M-states causes the execution to move a different set of cores, i.e., the ones matching the chosen M-state. Transitions among these states are governed by a controller module depending on the application behavior and platform power constraints.

HeteroMates is implemented on top of the Linux kernel. Experimental evaluations use a unique, experimental heterogeneous multicore platform comprised of both high and low power cores, along with client applications typically seen in modern end-user devices. Two different usage policies are compared for H-state solution: a performance-driven policy favors high performance for user-facing applications, whereas a power-driven policy favors reduced power consumption and longer battery life. Similarly, an analysis is performed for M-state solution comparing several under-provisioned (all-small, all-big, heterogeneous) and over-provisioned configurations (with parallelism-aware and static oracle M-state controllers). Experimental results demonstrate that by opportunistically utilizing heterogeneous cores, HeteroMates can provide both improved performance and lowered energy consumption for various client applications when compared to homogeneous cores.

The rest of the paper is organized as follows: Section 2 presents motivational examples and use cases from client domain. Section 3 discusses emerging trends in the semiconductor industry. A detailed design of HeteroMates system describing core group and multicore group abstractions is presented in Section 4. Section 5 analyzes the impact of uncore power on the design. Evaluation methodology and experimental results are provided in Sections 6 and 7, respectively. Finally, Sections 8 and 9 summarize related work and conclusions from the work.

2. Motivation

Users perform a wide variety of tasks on mobile devices, resulting in diverse platform demands. Since their battery capacities are severely restricted due to constraints on size and weight, energy efficiency is critical to their usability. To provide extended battery life and at the same time, meet the rapidly increasing demands of high performance mobile use cases, a client device must offer a wide *dynamic power-performance range* – it must be able to operate both in high-performance and in power-savings modes. As explained in detail in Section 3.1, heterogeneous cores can be used to extend the dynamic power/performance range offered by homogeneous processor configurations. In that context, this section discusses examples of client workloads (see the list in Table 2), and the usage patterns of client devices that motivate the need for a wide dynamic power/performance range and discusses opportunities for using heterogeneous cores for this purpose.

2.1. Client workloads

Client applications exhibit highly diverse behavior in their processor usage and performance requirements. These applications can be categorized based on their behavior as described below.

2.1.1. Intermittent workloads

Client devices like cell phones 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 Fig. 1(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.

2.1.2. Sustained workloads

These 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. Fig. 1(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

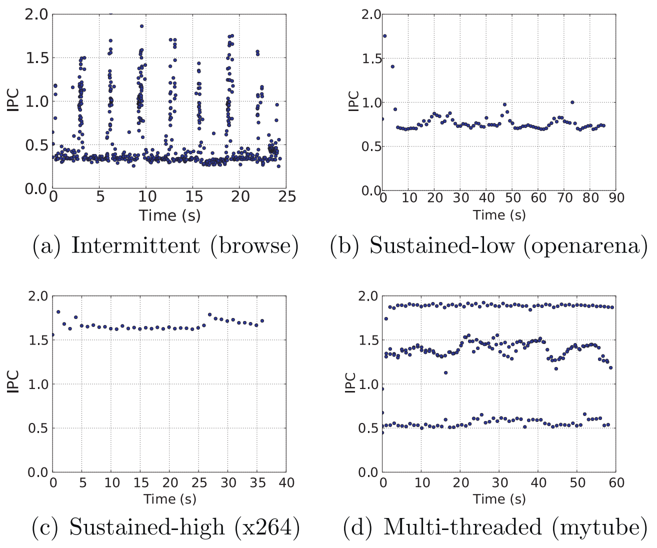


Fig. 1. Diverse client workload profiles (IPC vs. time).

core without significant degradation in performance and at lower power.

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 Fig. 1(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).

2.1.3. Multi-threaded workloads

With increasing numbers of cores on mobile devices, parallelization of client applications is key to further performance enhancement. Such multi-threaded applications also present opportunities for exploiting heterogeneity. 7zip, gmagick, and eclipse are examples of parallel applications. The mytube workload also uses multiple threads for audio, video decoding, and rendering, for instance. Since such threads differ in behavior and needs, their performance will be affected by how they are mapped to different heterogeneous cores. For example, Fig. 1(d) shows that various threads within the mytube workload differ significantly in their IPC, which can be leveraged by task mapping and scheduling methods.

2.2. Client devices

2.2.1. Mobility constraints

Mobility means that client devices will either be powered via wall-power or by battery. Wall-power usage does not impose energy constraints, so that big cores can provide desired high levels of performance. During battery-driven operation, however, a user may be willing to accept lower performance at the benefit of higher battery life. Low-powered energy-efficient small cores may be more suitable under such conditions.

2.2.2. Thermal constraints

Client devices like cell phones and tablets rely on natural cooling. Therefore, these devices are quite sensitive to platform thermal constraints that impose limits on the extent to which it is possible to use power-hungry big cores for sustained periods of time. A small core can be used for moving the execution away from a big core when thermal constraints are violated.

3. Processor design trends

Several emerging trends in microprocessor design including heterogeneous cores and processor over-provisioning influence the design of HeteroMates which are summarized in this section.

3.1. Heterogeneous cores

Modern processors are typically designed to satisfy only one of the two conflicting requirements: high-performance and energy-efficiency. Current low-power cores (e.g., Intel's Atom processor) are energy efficient, but their performance is limited. More powerful big cores like Intel Core® processors provide high performance, but at the cost of higher levels of power consumption. The technological reasons for this are the fact that the power consumption of a processor core consists of static (leakage) power and dynamic (switching) power. During high activity periods, the total power consumption of the device is dominated by dynamic power consumption, while during low activity periods, leakage power becomes a significant component of the total power consumption. Current high performance cores are built from transistors on fast process technologies that have high leakage power and very fast switching times [1]. Such big cores, therefore, consume high

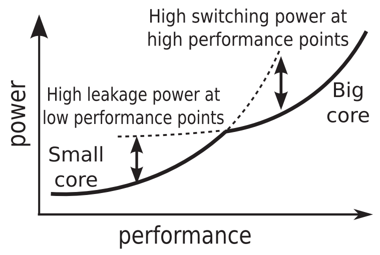

High switching power at high performance points
High leakage power at low performance points
Big core
Small core
power
performance

Fig. 2. Big cores are less efficient at low activity points, while small cores are less efficient at high activity points. Using a heterogeneous processor provides a wide dynamic power/performance range.

leakage power under idle or near-idle conditions, but can provide high performance without significant increase in dynamic power, as shown in Fig. 2. Conversely, low power small cores are built from low power process technologies with low leakage power but slower switching times [1]. Such processors consume small amounts of leakage power, but significantly increase dynamic power consumption to provide a high-performance mode (see Fig. 2). Static power is mainly determined by the silicon process technology, and dynamic power is determined by silicon process technology and by operating voltages and frequencies.

The intuitive outcome is that by using both types of cores, a single platform can be optimized for both high performance and low power consumption. The objective of such a system would be to always use its most efficient cores for the tasks at hand (shown by the solid line in Fig. 2). Such a heterogeneous platform exhibits a higher power-performance range than individual big or small cores. This paper explores whether and to what extent the hardware-based arguments for heterogeneity stated above lead to realistically achievable gains on client devices.

3.2. Over-provisioned processors

The transistor density on modern processors continues to grow, but due to poor Dennard scaling, i.e., reduction in CMOS threshold and supply voltages, per-transistor power is decreasing at a much slower rate. Under constant power budgets, which are governed by the cooling technology of the platform, the gap between the transistors that can be fit into the die area and the transistors that can be activated simultaneously is widening (see Fig. 3) [7]. Thus, future processors are likely to have significant *dark* resources, e.g., it is estimated that in 8 years, we will be faced with designs that would have 93.75% silicon over its provisioned power limit. This phenomenon is more severe in passively cooled mobile devices such as netbooks, tablets, and smart phones.

Mitigating or exploiting dark silicon presents new challenges and opportunities in terms of finding novel ways to utilize such *over-provisioned* resources to improve performance or energy-efficiency. One way to address the problem of dark silicon is to build smaller processors and discard dark silicon, but various economic

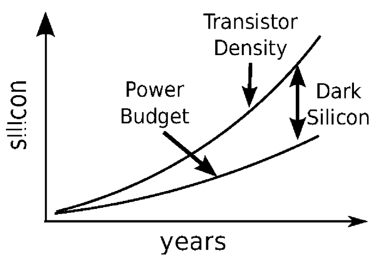


Fig. 3. Future processors are likely to be over-provisioned due to the increasing gap between transistor density and transistors that can be sustained within power budgets.

constraints are likely to promote large size chips that will have far more transistors than can be used simultaneously. Another approach to mitigate dark silicon is to slow down all of the components on the processor to fit into the power envelope, but this approach is energy-inefficient as DVFS incurs a quadratic power cost for a linear frequency boost. Therefore, the approach explored in this work is to design processors which are provisioned above their TDP (thermal-design-power) limits, but it employs heterogeneity to exploit over-provisioning such that only a subset of components are used for concurrent execution.

4. HeteroMates design

HeteroMates enables a wide dynamic power/performance range using heterogeneous cores. This section describes its key concepts and components.

4.1. Core groups

A heterogeneous *core group* is a collection of a small number of (e.g., 2–4) heterogeneous cores that are grouped together to form a single execution unit. For example, Fig. 4 shows a core group consisting of three heterogeneous cores: a big (B), a small (S), and a tiny (T) core. The core group appears as a single execution unit with multiple performance/power levels. Depending on application behavior and user-defined policies, an appropriate core is dynamically chosen to run the user task in question, by transparently moving the task's execution to that core, and by placing the other inactive cores into a low power idle state to conserve power. For example, the tiny core can be used for background tasks like email update checks, the small core for normal user operation, and the big core is reserved for performance-critical tasks.

4.1.1. Heterogeneity-states

Different cores within a core group are exposed using *heterogeneity-states* (H-states), an interface similar to the P-state (performance-states) interface defined by the ACPI standard and used by operating systems to scale the frequency and voltage of processors. Higher P-state numbers represent slower processor speeds. Thus, P0 is the highest-performance state, with P1 to Pn being successively lower-performance states. Similarly, an H-state is assigned to each type of core in the core group. A high-performance big core corresponds to a lower numbered H-state, while a low-power small core corresponds to a higher-numbered H-state. Thus, a core group exposes a set of H-states (H_0, . . ., H_n) which are controlled by an *H-state controller* module. Depending on the state transition logic and the resultant H-state, a *core switcher* transparently migrates the execution to the appropriate core. In this manner, applications perceive only homogeneous cores with

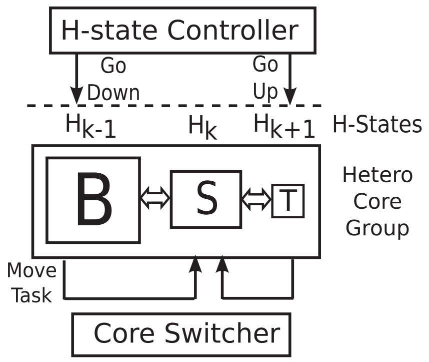


Fig. 4. A core group consisting of three heterogeneous cores: a big (B), a small (S), and a tiny (T) core exposed as three H-states.

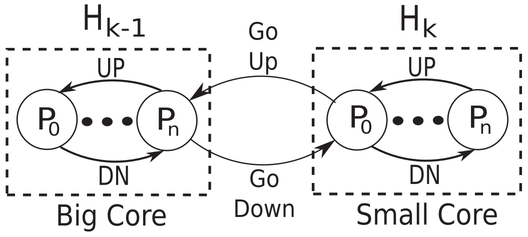


Fig. 5. H-state and P-state transition state machines. H-state determine the core for execution, while P-states determine the frequency on that core.

larger dynamic power/performance range than any of the individual cores.

4.1.2. H-state controller

H-states on a core group are controlled by the H-state controller, in a manner similar to frequency scaling operations performed by a CPU governor. A CPU governor is a kernel module that changes core P-states based on a policy. In a similar manner, the H-state controller performs H-state scaling operations. However, instead of changing voltage and frequency as in the case of P-states, a change in H-state causes the execution to move to a different core. The functions of the H-state controller and of the traditional P-state governor complement each other. For example, Fig. 5 shows the combined P-state and H-state transition diagram for a two-core heterogeneous core group. Here, H_k corresponds to the small core, and H_{k-1} corresponds to the big core. P-state changes within a core are performed by the P-state governor, while cross-core migrations are governed by the H-state controller.

CPU governors available in current operating systems (e.g., the ondemand governor in Linux [8]) dynamically change CPU frequency in response to CPU load (utilization). However, CPU load alone is not sufficient to drive H-state scaling operations, which also require determining whether a bigger or smaller core is more suitable for execution. Previous work on heterogeneous processor scheduling [4–6] has identified application IPC (instructions-per-cycle) as a key metric to select the right core for execution. Therefore, HeteroMates uses a combination of CPU load and application IPC to form the H-state transition logic shown in Fig. 6.

The intuition behind the scaling algorithm can be explained as follows. An application with high CPU load but low IPC is likely to perform equally well on both big and small cores due to its low IPC requirements, which can easily be met on a small core. Applications with high IPC but small CPU load under-utilize the big core. Moving such applications to a smaller core results in higher utilization of the small core, but without a significant penalty in application performance. When both of these conditions are violated, the application is likely to gain performance by executing on a bigger core.

The H-state controller monitors application IPC and CPU load at periodic intervals and compares them with pre-defined thresholds

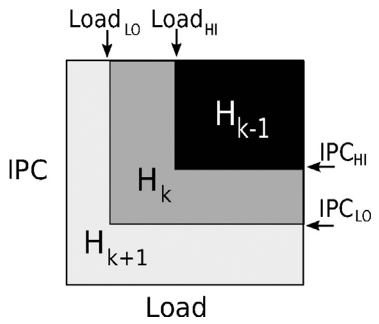


Fig. 6. H-state scaling operations in response to application IPC and CPU load.

to determine the resultant state (see Algorithm 1). If both the IPC and load are above thresholds IPC_{HI} and $Load_{HI}$, respectively, the core group is scaled up, i.e., moved to a higher-performance or lower numbered state (H_{cur-1}). If either IPC or load are lower than thresholds IPC_{LO} and $Load_{LO}$, the H-state is scaled down to a lower-performance state (H_{cur+1}). For values in between these thresholds, no H-state change is performed. These thresholds are defined for each type of core in the system. By setting different values for these thresholds, different policies can be enforced. For example, low values of thresholds force the execution to big cores, and thus prefer performance over power. Similarly, a policy having thresholds with high values picks smaller cores more often.

Algorithm 1 *(H-state controller heuristic).*

```
if $IPC > IPC_{HI}$ AND $Load > Load_{HI}$ then
    // Scale up
    $H_{next} = H_{cur-1}$
end
else if $IPC < IPC_{LO}$ OR $Load < Load_{LO}$ then
    // Scale down
    $H_{next} = H_{cur+1}$
end
else
    // No change
    $H_{next} = H_{cur}$
end
```

An H-state change operation causes the execution to switch to a different core. This switching overhead could be substantial due to migration latency and loss of private cached data if such changes are very frequent. In response, we use *history counters* to dampen core switching frequency. A switch is performed only after a certain number of consecutive identical H-state change requests are received. The history counter is a simple integer counter associated with each core group, which is incremented whenever consecutive intervals generate the same requests and reset otherwise.

4.1.3. Advantages

The design of HeteroMates offers multiple advantages. First, H-state interface decouples heterogeneity from scheduling such that the scheduler need not deal with performance/power differences among cores. Instead, a separate H-state driver handles this transparently to the scheduler. Second, H-states can be implemented either in hardware, firmware, operating system, or even hypervisors, allowing a broader applicability. As an architectural solution, it provides a way to completely hide heterogeneity from the operating system, which is critical to support legacy software and applications. Further, core groups provide a unified mechanism to easily accommodate a variable number of heterogeneous cores by adding an H-state for each type of core. Finally, core groups can be useful when TDP (thermal-design-point) limits may constraint the number of cores that can be active simultaneously. As transistor density on modern processors keep increasing, such TDP limits are proving to be a critical design constraint in the form of *dark silicon* [7].

4.2. Multicore groups

The core group abstraction can be extended to multicore systems using *multicore groups*. This section describes how multicore groups can exploit dark silicon using over-provisioned heterogeneous processors.

4.2.1. Exploiting over-provisioning

Due to the presence of dark silicon [7], future platforms are likely to be constrained by CPU power rather than die area. This presents interesting choices for platform designers to utilize such excess die area. For example, Fig. 7 shows several processor design choices containing homogeneous and heterogeneous cores for a fixed power budget (equivalent to CPU area in the figure). One design option is to use only homogeneous cores, while staying below the platform power budget. Thus, it may contain few big cores or many smaller cores. However, these different configurations are suited for different types of applications. For example, the big core can deliver high responsiveness for single-threaded user-facing applications, while multiple small cores are more suited for applications with parallelism. Therefore, heterogeneous cores can be used to extract the benefits of both types of cores. However, fixed CPU power budget constraints require either slower or fewer processors to be used in the heterogeneous configuration to stay within budget ceiling when compared to homogeneous options, thus, comprising performance for certain applications.

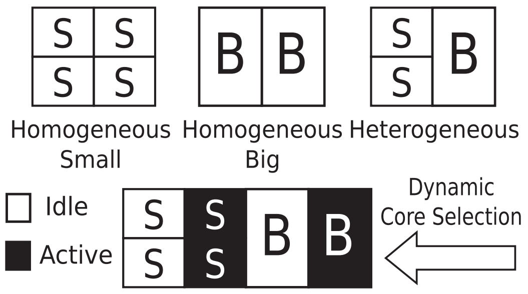


Fig. 7. Over-provisioned platform can be morphed into different configurations by dynamic core selection.

In comparison, the over-provisioned configuration (bottom figure) exploits dark silicon by employing heterogeneous cores consisting of many low-powered smaller cores and high-performance bigger cores. Since the total power consumption of all the cores exceeds the TDP limits of the platform, it requires dynamic core selection mechanisms to activate a subset of the cores such that they conform to the budget specifications. Such an *over-provisioned heterogeneous processor* can be used to provide the goodness of various configurations shown, by opportunistically using the right set of cores for application execution depending on application behavior and user preferences. By matching execution resources to application needs, it maximizes system performance/energy-efficiency under power-constrained conditions.

4.2.2. Platform reconfiguration

Various platform configuration that satisfy the budget requirements can be pre-configured or dynamically created during execution, which are exposed using *multicore-states* (M-states). As different H-states represent various cores within a core group, an M-state is similarly assigned to each platform configuration. For example, a state M0 could represent a configuration consisting of fewer high-performance big cores, while a configuration consisting of many low-power small cores corresponds to a different state (M1) which are controlled by an *M-state controller* module. Depending on the state transition logic and the resultant state, a *task switcher* transparently migrates the execution to the appropriate cores. M-state switching operations are controlled by the M-state controller. A change in M-state causes the execution to move to a different set of cores. The controller needs to take platform power constraints, application power consumption, and thread behavior into account to select the optimal execution environment.

4.3. Implementation

HeteroMates is implemented for the Linux kernel. The current implementation of core groups considers systems involving pairs of heterogeneous cores. H-states are implemented by customizing the P-state tables on each core to expose two P-states corresponding to each core in a pair. H-state changes work in lock-step on both of these cores to avoid conflicting operations. An H-state change causes execution to switch cores instead of performing DVFS. Our current implementation does not consider traditional voltage and frequency scaling. This is because there is substantial previous work on DVFS [9–12], which can be used to perform P-state scaling in addition to H-state transitions.

The H-state controller is implemented as a kernel module which runs on each active core as a kernel thread. It periodically (40 ms) reads various hardware performance monitoring counters (PMCs), applies models, and performs any H-state changes depending on the policy and thresholds chosen. The overhead of running models is measured to be small (approximately 2% increase in core active and 5% increase in package active residency). The core switcher is implemented in the OS kernel by changing the runqueue pointer for the tasks in the source runqueue to point to the destination runqueue. The overhead of this operation is minimal when runqueue length is not large, which we have observed as being the case for the typical client workloads used in our experiments. We note that similar functionality can be provided by hardware, to further reduce overheads. Also, only active cores are made available for scheduling to the Linux CFS scheduler. Inactive cores are put into an offline mode using a lightweight mechanism. A value of three is used for history counters.

For the M-state solution, this paper considers designs where the system is configured to operate in one of the many built-in M-states that are under budget limits. Further, a simple heuristic based upon the thread-level-parallelism in the application is used for analysis as described in Section 7.2.1. As part of our future work, we are exploring designs which dynamically compose various platform configurations based on the power profile and thread behavior of running applications.

5. Beyond core:uncore

The dynamic power/performance range offered by a platform consisting of heterogeneous cores can be strongly affected by the *uncore* subsystem present on modern multicore processors.

5.1. 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 Fig. 8. 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.

5.2. Idle state coordination

Modern multicore processors contain core idle states (C-states) to progressively turn off components in order to conserve power.

Table 1
Core and package idle state coordination.

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

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.

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.

5.3. Impact of uncore

Fig. 9 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 are 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 [13]. 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 [14,9].

Energy consumption for the big core and small core execution for such platforms can be modeled using Eqs. (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.

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

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

To understand the impact of uncore power, the evaluation in Section 7 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 such as memory controllers or cache For example, fewer memory channels, memory controllers, or a smaller cache can be used with a slow

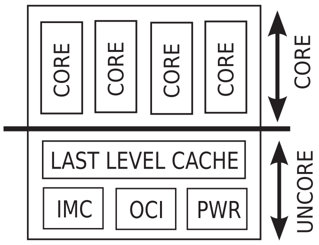


Fig. 8. Core and uncore in multicore processors.

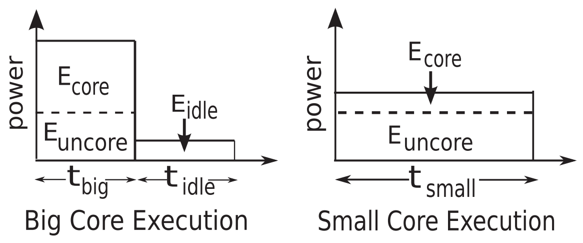


Fig. 9. Effect of uncore power on the dynamic power range of heterogeneous cores.

small core that imposes smaller resource requirement on the cache and memory subsystem. units are turned off or powered down as we move to the small core. Hence, in this case, the uncore power scales along with core power when a workload moves to a different core.

5.4. Uncore-aware operation

As discussed above, the energy-efficiency of a platform is not only determined by the type of core used for execution, but also by the power consumption of the shared uncore subsystem. Workloads for which execution on a bigger core provides both higher performance and better energy-efficiency due to improved performance scaling, should always be run on big cores as small core degrades both performance and efficiency. HeteroMates addresses this issue by adding the *energy override* condition in Eq. (3) to the heuristic described in Section 4.1.2. If the energy consumption of the current H-state (H_{cur}) is greater than the energy consumption of the next higher state (H_{cur-1}), a scale up operation is performed to move the execution to the bigger core.

$$\text{If} \quad \frac{Energy(H_{cur-1})}{Energy(H_{cur})} < 1 \quad \text{then} \quad H_{next} = H_{cur-1} \tag{3}$$

For energy-aware operation, Eq. (3) requires the energy consumption of the application to be estimated on a different core (H-state). This task can be divided into two components: processor power prediction and application behavior (e.g., execution time, IPC) prediction. CPU power visibility to the operating system is becoming increasingly important, with multiple CPU vendors providing hardware counters to measure the power of different components on the platform [15]. Further, previous work has developed light-weight models to accurately predict per-core power using existing performance events [16]. Using a similar approach, this work also uses power models, described in Section 6.3, to obtain core and uncore power consumption.

In order to understand the impact of a core transition on application behavior, hardware assistance can be provided. For example, HeteroScouts [17] proposes hardware performance counters to predict workload behavior on a remote core (after-transition) from the parameters available on the local core (before-transition). Due to unavailability of such counters in current processors, simple prediction models are developed using experimental data. The following section provides details of the modeling methodology.

5.5. Remote behavior prediction

To model the relationship between application IPC on a big and a small core in our experimental platform (see Fig. 11), the client workloads in Table 2 and SPEC CINT2006 benchmarks are executed on both types of cores. Fig. 10 plots the obtained $IPC_{scaling}$ data, defined as the ratio of the big core IPC and the small core IPC, as

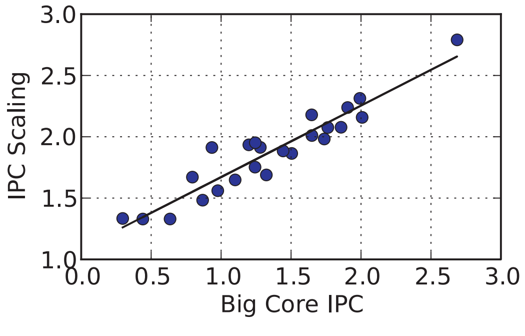


Fig. 10. Modeling IPC scaling as a function of IPC.

a function of the IPC on the big core. As evident from the figure, a linear curve fits the data well, with the resultant model given by the equations below.

$$IPC_{scaling} = 0.6 * IPC_{big} + 1.01 \tag{4}$$

$$IPC_{scaling} = 1.31 * IPC_{small} + 0.94 \tag{5}$$

The impact of IPC scaling on the execution time of an application is workload dependent. CPU-bound workloads show a proportional relationship between IPC scaling and execution-time scaling. However, this does not hold true for many client workloads with significant idle phases, e.g., media and graphics workloads. For such workloads, execution time is not affected by the core performance. Instead, a change in core performance translates into change in core idle state residency. These conditions are modeled by applying the scaling function to the product of core active state (R_{active}) residency and execution time (t), as shown in Eq. (6). The equation was experimentally verified using all of the client workloads in Table 2 as majority of the workloads closely follow the modeled relationship. In the online model, sampling interval is substituted for the execution time.

$$(R^{small}_{active} * t_{small}) = IPC_{scaling} * (R^{big}_{active} * t_{big}) \tag{6}$$

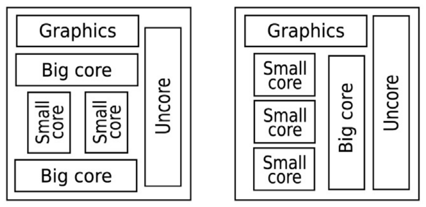
Further, the change in core idle residency (R_{idle}) impacts package idle state (U_{idle}) residency in an application dependent manner. Applications for which the package becomes idle as soon as the core becomes idle, show a strong correlation between core and package idle states. On the other hand for multi-threaded applications and graphics-intensive applications, a core's idle state does not necessarily translate to the package idle state since the package can still be busy due to activity in another core or the graphics processor. Such applications show a weak or negligible correlation between core and package idle states. These two scenarios are modeled in Eq. (7) where a difference of 20% between U_{idle} and R_{idle} is assumed as an indicator of weak correlation. For such cases, U_{idle} is assumed to be the same irrespective of the type of core used for execution.

$$U^{small}_{idle} = \begin{cases} U^{big}_{idle} & \text{if} \;\; U^{big}_{idle} \ll R^{big}_{idle}, \\ R^{small}_{idle} & \text{otherwise} \end{cases} \tag{7}$$

Using the models presented above and the power models described later in Section 6.3, an application's relative energy consumption on two different H-states can be obtained. These values are used to perform energy override operations as defined earlier by Eq. (3).

Table 2
Client workload summary.

Workload	Description	Metric
7zip	Text file compression using archiver	Time
applaunch	Application launch operation	Latency
bodytrack	Computer vision kernel	Time
browse	Web-page rendering (browser)	Latency
c-ray	Image generation using ray-tracer	Time
canvas	HTML5 canvasing tests (browser)	FPS
eclipse	Java IDE performance tests	Time
filescan	File-system read/write operations	Time
gmagick	Batch resizing of images	Time
grayscale	Image filtering operation (browser)	Latency
gtkperf	GTK GUI performance tests	Latency
javascript	Scripting operations (browser)	Latency
lightsmark	3D graphics rendering tests	FPS
mplayer	H/W accelerated video playback	FPS
mytube	Streaming video playback (browser)	FPS
openarena	3D first-person-shooter game demo	FPS
palbum	Photo-album slide show (browser)	Latency
strike	2D shooting game demo (browser)	FPS
x264	Media file conversion using encoder	Time
zoom	Image zoom operation (browser)	Latency



	Big cores (freq)	Small cores (freq)
M0	1 (2.4 GHz)	0
M1	1 (1.8 GHz)	1(1.8 GHz)
M2	0	3 (1.8 GHz)

(a) S-B core groups (b) Platform configuration and corresponding M-states

Fig. 11. Experimental heterogeneous platform.

6. Experimental evaluation

6.1. Experimental platform

Our experimental platform consists of a quad-core Intel i7-2600 client processor. To create heterogeneity, we use an Intel proprietary tool to de-feature and emulate the performance of low-powered small cores for a subset of the cores [4]. The block diagrams of two platform configurations used for analysis are shown in Fig. 11. An on-die graphics processor is used to accelerate graphics workloads. All of the cores share an LLC of size 8 MB. All the workloads are run using Linux kernel 3.0 and automated. Browser workloads are run using Google Chrome 15.0.

The configuration shown in Fig. 11(a) is used for forming two core groups consisting of one big and one small core each, both operating at a frequency of 2.6 GHz. Similarly, the platform configuration in Fig. 11(b), consisting of one big core, three small cores, is used to form a multicore group with three M-states as shown by the table in the figure (on right). M0 consists of a single big core (1B) running at a frequency of 2.4 GHz, while M2 state uses three small cores (3S) each configured to run at 1.8 GHz. Similarly, M1 contains one big and one small core, but the big core is throttled to run at 1.8 GHz to stay within power limits (1B*,1S).

6.2. Client workloads

To assess the viability of using heterogeneity for client systems, a diverse set of real-world applications are chosen which are typical of modern end-user devices since prior server-centric research on heterogeneous processors [4–6] 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 which include browsing, gaming, media, etc., and relevant performance metrics which are different from server workloads.

6.3. 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 power consumption.

6.3.1. Core power

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

$$P_{core} = R_{active} * P^{core}_{active} + R_{idle} * P^{core}_{idle} \tag{8}$$

$$P^{core}_{active} = C_{dyn} * V^2 * f \tag{9}$$

Here, R_{active} and R_{idle} denote core active and idle state residencies (%), and P^{core}_{active} and P_{idle} 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 Eq. (10), as shown and validated by other researchers [18]. Similarly, Eq. (11) models the capacitance for a small core having three-times smaller area than the big core.

$$C_{big} = 0.499 * ipc_{big} + 0.841 \tag{10}$$

$$C_{small} = 0.472 * ipc_{small} + 0.176 \tag{11}$$

6.3.2. Uncore power

Similar to core power, uncore power can be modeled using package idle state residencies (U_x) as shown in Eq. (12).

$$P_{uncore} = U_{active} * P^{uncore}_{active} + U_{idle} * P^{uncore}_{idle} \tag{12}$$

$$P^{uncore}_{active} = P_{wake} + P_{activity} * LLC_{rate} \tag{13}$$

Further, uncore active power (P^{uncore}_{active}) is modeled as a function of the LLC activity in Eq. (13) 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). 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.

7. Results

Experimental results evaluating H-state and M-state solutions are presented in Sections 7.1 and 7.2, respectively.

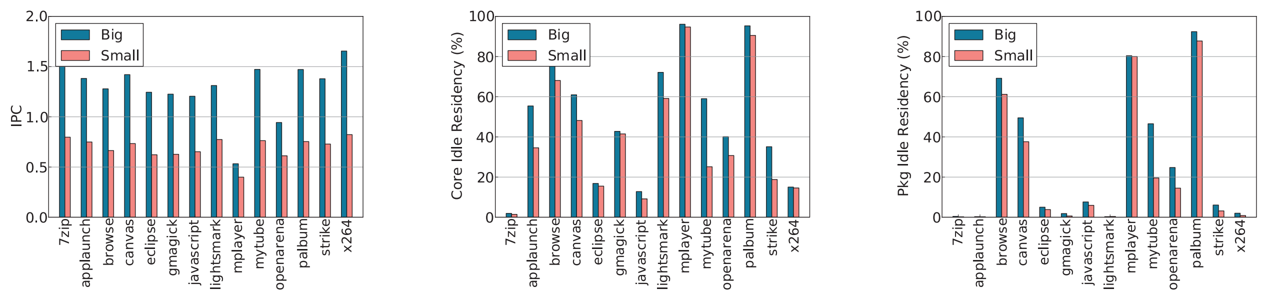
7.1. H-state evaluation

7.1.1. Methodology

Two different policies, performance-driven and power-driven, are used for evaluation. This is done by choosing different threshold values, obtained after experimenting with several combinations of thresholds. Table 3 summarizes the various thresholds used to cater to these policies. For a paired-core system, small cores can only perform scale up operations and not scale down, therefore, only HI thresholds are relevant for small cores. Similarly, only LO thresholds are relevant for the big cores. The first performance-driven policy favors performance over power by using big cores

Table 3
Thresholds for performance- and power-driven policies.

	Small core		Big core	
	IPC_{HI}	$Load_{HI}$	IPC_{LO}	$Load_{LO}$
Performance-driven	0.5	70%	0.8	40%
Power-driven	0.7	80%	1.25	50%



(a) Instructions-per-cycle (b) Core idle residency (c) Package idle residency

Fig. 12. A comparison of the behavior of several client workloads on big vs. small cores.

for execution in an aggressive manner. This is achieved by choosing smaller thresholds in the table. The power-driven policy, on the other hand, focuses on power by choosing bigger thresholds and forcing the execution to small cores more often. The evaluation is carried out by comparing the performance and energy consumption of the performance-driven policy with only big core execution and of the power-driven policy with just small core execution. These two comparison points provide us a perspective of the advantage of using heterogeneous cores over homogeneous configurations.

7.1.2. *Client workload characterization*

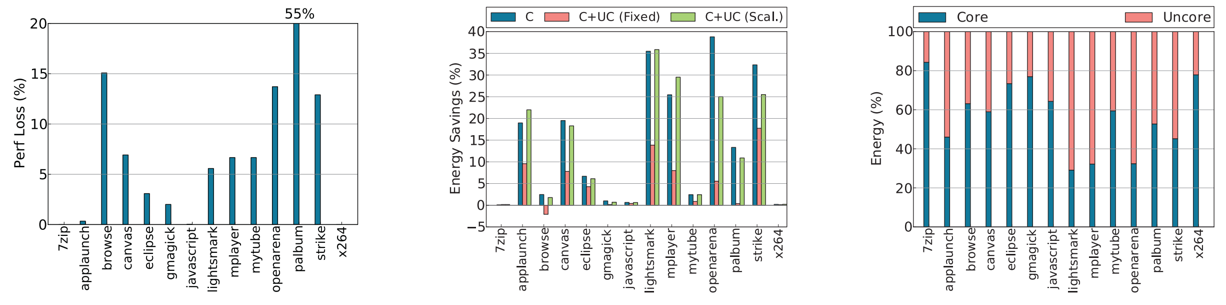
The results shown in Fig. 12 provide a comparison of the behavior of various client applications on heterogeneous cores. Specifically, they compare average IPC (instructions-per-cycle), core idle residency, and package idle state residency for all of the workloads in Table 2 for big and small core execution. As evident from Fig. 12(a), most of the applications observe a significant decrease in their IPC when running on the small core as compared to the big core. This reduction in IPC results in the small core being active for longer durations, thereby causing a decrease in core and package idle residency (see Fig. 12(b) and (c)). Further, many applications are seen to have almost negligible package idle residency. These applications either heavily use the graphics processor (e.g., openarena, lightsmark), or they always keep one of the CPU cores busy (e.g., 7zip, gmagick, x264), and do not allow the uncore to enter into an idle state.

7.1.3. *Performance-driven policy*

Fig. 13 provides results comparing the performance and energy consumption of the performance-driven policy with execution on big cores. Specifically, Fig. 13(a) shows performance loss (%) with respect to the maximum performance achievable by using big cores for the entire execution, and Fig. 13(b) shows corresponding energy savings by using small cores for partial execution when big core is not energy-efficient. Performance is measured based upon the metrics in Table 2, with inverse of latency as the metric for latency-oriented workloads. As evident from the figures, this policy is able to achieve performance within 15% of the big core performance for all the workloads except browse and palbum. This high performance loss for these two workloads is due to their bursty nature, i.e., these applications exhibit sudden bursts of high activity during page-rendering. HeteroMates uses history counters to dampen core switching frequency, which requires multiple consecutive state change requests to be received before actually making the change. Due to this reason, these bursty applications observe a short delay before they are moved to the big core which incurs a higher performance degradation. However, the absolute increase in the latency for these applications may not be user-perceivable.

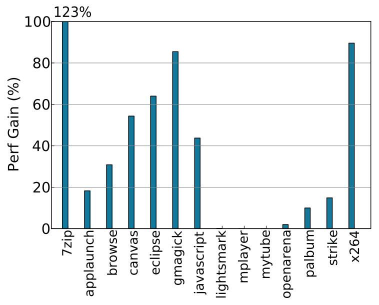
Fig. 13(b) shows corresponding energy savings results for three scenarios: core-only savings (C), SoC-wide savings (C + UC) with a fixed uncore, and SoC-wide savings with a scalable uncore. As seen from the figure, it is able to save significant energy for several applications with a small performance degradation. Workload openarena achieves highest gains with 39% core energy savings. However, these savings are strongly affected when the power consumption of the uncore is taken into account. On the other hand, when a scalable uncore is used, these savings increase and become comparable (25%) to core-only energy savings.

To elaborate on the importance of uncore power in total SoC power, Fig. 13(c) shows the distribution of core and uncore energy consumption for various applications. Core energy component dominates for CPU-intensive applications like 7zip, eclipse, gmagick, and x264, while uncore component is significant for other applications including lightsmark, mplayer, and openarena. These results highlight the growing importance of uncore power in the processor power consumption and motivate the need for a scalable uncore design when seeking to obtain large gains from heterogeneous multicores.

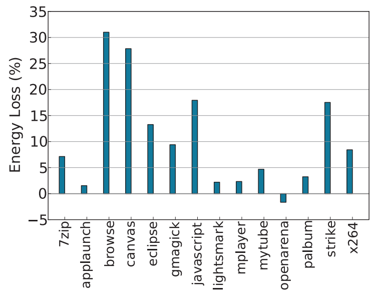


(a) Performance loss wrt. big cores (b) Energy savings wrt. big cores (c) Core and uncore energy distribution

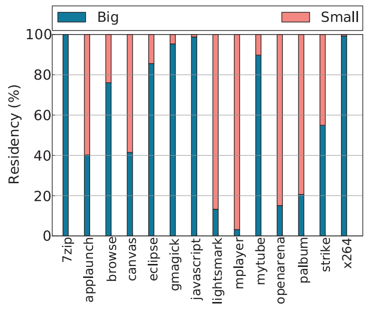
Fig. 13. Comparison of performance-driven policy with big core execution.



(a) Performance gain wrt. small cores



(b) Energy loss wrt. small cores



(c) Core residency

Fig. 14. Comparison of power-driven policy with small core execution.

7.1.4. *Power-driven policy*

Results for the power-driven policy are presented in Fig. 14, where Fig. 14(a) and (b), respectively, show performance gain and energy loss (SoC-wide) in comparison to small-core-only execution. As results show, this policy is able to achieve significant performance gains for many applications by selectively using big cores. Further, it is able to do so with only a small to moderate increase in energy consumption. For example, the browse and canvas workloads observe the highest increases in energy consumption of 31% and 28%, respectively, while most of the other applications show a smaller increase. However, these two applications also show a 31% and 54% performance gain for the increased energy consumption due to their usage of big cores. We note that some applications like lightsmark, mplayer, and openarena exhibit negligible performance improvement due to poor scalability.

Results in Fig. 14(c) show the percentage residency on big and small cores for all of the applications. Different applications exhibit different degrees of big and small core usage. For example, applications like 7zip, eclipse, and x264 with good performance scalability spend the majority of their execution on big cores. On the other hand, applications like lightsmark, mplayer, and palbum remain on small cores for a significant portion of their execution time. Other applications like applaunch, canvas, and strike make use of both types of cores during their execution. To illustrate this further, the big and small core usage profiles of the applaunch and strike workloads are shown in Fig. 15. The applaunch workload launches and executes a series of graphics-intensive applications. The launch operation is CPU-intensive and performs better on a big-core, while the execution phase is accelerated using the on-die graphics processor and a small core provides comparable performance to the big core at a lower power. Therefore, this workload transits between big and small cores during launch and execution phases (see Fig. 15(a)). Similarly, Fig. 15(b) shows the execution profile for the strike gaming workload. This workload exhibits several phases with high activity (e.g., bots shooting) when big cores are used and phases with low activity (e.g., bots aiming and moving) when small cores may suffice. In this manner, the appropriate core is used depending on the activity.

7.2. *M-state evaluation*

7.2.1. *Methodology*

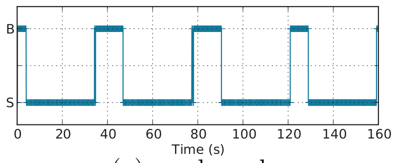
Experimental evaluation and analysis for multicore groups are carried out as the steps summarized below.

- Each workload is first individually evaluated on each of the three M-state configurations in Fig. 11(b).
- Using the data collected in the previous step for each M-state, we perform an analysis for each workload to obtain its performance for a thread-level-parallelism-aware controller (TLP) that assigns state M0 for single-threaded applications and M2 for applications having multiple threads. Since many applications use helper threads which do minimal work, only threads with CPU load larger than 10% are used for accounting in the heuristic. Our analysis assumes the use of a fixed M-state for entire application run. The implementation and evaluation of a dynamic switching algorithm is part of our future work.
- Similarly, we obtain results for a 'static oracle' controller (ORCL) that selects the M-state with maximum performance among the three M-states. This state corresponds to the highest performance that can be achieved on the over-provisioned platform by selecting that M-state for each workload.
- Based upon the analysis methodology described and the power models presented in Section 6.3, we compare performance improvement and energy savings provided by three under-provisioned platforms corresponding to each M-state and the over-provisioned configuration with the two controller heuristics.

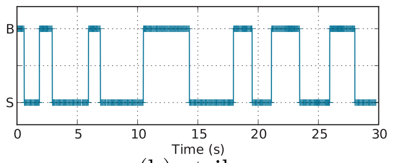
7.2.2. *Results*

Fig. 16 shows experimental results for the three M-state configurations described in Section 6.1. Results presented are mean values over three runs.

Specifically, Fig. 16(a) shows relative performance in each M-state (normalized to the minimum performance state) for all of the client workloads. As evident from the figure, various applications have affinity toward different configurations.



(a) applaunch



(b) strike

Fig. 15. Big (B) and small (S) core usage profile (*x*-axis: time (s)).

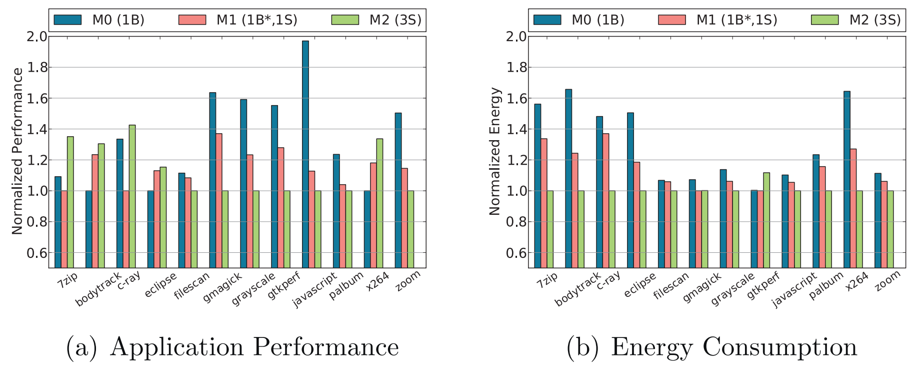


Fig. 16. Experimental results for three platform configurations: M0 (1B), M1 (1B*1S), and M2 (3S).

Some applications (e.g., gmagick, javascript, zoom, etc.) perform better in state M0, while other applications like 7zip, c-ray, x264, etc. exhibit higher performance in state M2. Thus, different M-states can be used to improve the performance of these applications.

The corresponding change in energy consumption for these configurations is shown in Fig. 16(b). Interestingly, the increase in energy consumption in state M0 for several applications like gmagick, gtkperf, and javascript is small in comparison to the corresponding performance gain. On the other hand, M2 configuration provides both higher performance and lower energy consumption for 7zip, bodytrack, c-ray, and x264 applications. Both of these observations can be explained by the 'race-to-idle' phenomenon. In the former case, a big core in state M0 consumes higher power but finishes execution quickly to enter a low-power idle state. Thus, the increase in energy consumption is small. Similarly, improved performance and thus lower active time in state M2 for other applications causes the core and uncore subsystem to sleep longer and save energy.

Finally, Table 4 provides performance and energy comparison for three under-provisioned (UP) platform configurations corresponding to each M-state and an over-provisioned heterogeneous platform (OP). Results for the OP configuration are based upon two heuristics, TLP-aware (TLP) and static oracle (ORCL), as described in Section 7.2.1. The table shows average performance and energy consumption for all of the workloads in Table 2. Results demonstrate that the big core (M0) provides small increase in performance (1.10x) for a larger increase in energy consumed (1.29x) when compared to all-small configuration (M2). The corresponding values for state M1 are observed to be 1.01x and 1.14x.

In comparison, the OP platform shows a performance improvement of 1.25x and 1.30x at the cost of an increased energy consumption of 1.04x and 1.05x for TLP and ORCL controllers. Thus, an OP platform can provide a performance boost for mobile devices at the cost of a small increase in energy consumption, as demonstrated by the results. Further, the TLP-aware controller is able to perform comparatively to the static-oracle controller. Our current heuristic takes only application parallelism into account, thus information regarding the behavior of individual application threads and their interaction can be used to enhance the switching heuristic.

Table 4
Results summary: an over-provisioned (OP) platform provides significant performance gain in comparison to under-provisioned (UP) configurations.

Configuration	Power	Performance	Energy
M0 (1B)	UP	1.0x	1.0x
M1 (1B*,1S)	UP	1.10x	1.29x
M2 (3S)	UP	1.01x	1.14x
TLP (1B,3S)	OP	1.25x	1.04x
ORCL (1B,3S)	OP	1.30x	1.05x

7.3. Summary

In summary, the results presented bring us to the following conclusions:

- Client applications behave significantly differ from traditional server-centric workloads.
- Heterogeneous cores can enable both higher-performance and lower-power modes than homogeneous processor configurations.
- Performance-driven H-state controller results in significant energy-gain for several workloads with small performance-loss in comparison to a big-core-only system. Similarly, a power-driven policy provides performance boost for many applications with a small increase in energy consumption.
- Uncore subsystem is a significant and even dominating contributor to total energy consumption for many workloads.
- Energy savings from the use of small cores are severely affected by the uncore power, with a scalable uncore resulting in higher gains.
- Heterogeneous cores enable dynamic platform reconfiguration for power-constrained over-provisioned multicore systems.
- A TLP-aware M-state controller provides significant performance gains over static multicore configurations.

8. Related work

Heterogeneous chip multiprocessors (CMPs) have been proposed to achieve higher energy-efficiency than symmetric multicore processors. Using a mix of big and small cores, different phases within an application can be mapped to the core which can run them most efficiently [5,19,20]. Similarly, heterogeneous cores can be used to improve the performance of parallel applications by speeding up sequential phases within the application [3,21]. Studies have been performed to analyze the impact of performance asymmetry on several server workloads [22] and JAVA virtual machine services [23]. Researchers have also developed appropriate scheduling algorithms for operating systems [4,6,24–27] and virtual machine monitors [28,29] to efficiently run applications on heterogeneous cores. However, earlier work exposed the core heterogeneity to systems software requiring changes to the software stack to deal with heterogeneous cores. Also, it focused on server-centric workloads for evaluation.

In comparison, HeteroMates proposes core group abstraction to expose multiple heterogeneous cores as a single execution unit and thus reduce software complexity for wider adoption of heterogeneous systems. Our work targets client devices where energy is a premium resource, with diverse application behavior and performance metrics. In addition, previous work either maintained a fixed heterogeneous CPU configuration or did not impose power constraints on the platform to optimize core usage. On the other hand, HeteroMates exploits over-provisioned CPU resources using heterogeneous cores by dynamically changing between different M-states. Similar arguments have been made for distributing a thread's computation across various cores on an over-provisioned multicore, but with a focus on maximizing core reuse [30]. Further, conservation cores exploit dark silicon by using specialized processors to reduce per-computation power [31]. However, HeteroMates uses single-ISA heterogeneous cores to improve performance of diverse mobile applications by dynamically selecting a different set of cores and does not require any language tool-chain support to make use of heterogeneous cores.

There is also substantial previous work on dynamic voltage and frequency scaling (DVFS). Several techniques have been developed to dynamically select appropriate voltage and frequency for maximum efficiency [9–12]. However, others have questioned the effectiveness of DVFS on modern processors [14,32]. In this context, we extend the existing DVFS mechanisms to go beyond homogeneous cores and support core heterogeneity to enable a wide dynamic power/performance range on these client devices. We also highlight the significance of uncore power in total SoC power and motivate the need for a scalable uncore for exploiting maximum gains from heterogeneous CMPs.

9. Conclusions

This paper presents the *HeteroMates* solution in order to provide a wide dynamic power/performance range on client devices. It exploits dark silicon and core heterogeneity to enable both high-performance and power-savings modes while also being energy-efficient. Core group abstraction is proposed to mitigate challenges associated with heterogeneity which groups together a small number of heterogeneous cores to form a single execution unit. Cores within a core group are exposed as multiple heterogeneity (H) states. H-state transitions are governed by an H-state controller, while a core switcher transparently migrates the task to the appropriate core depending on the resultant H-state. Core group abstraction is extended to power-constrained multicore systems using multicore groups. A power-constrained platform is allowed to operate in different multicore configurations, exposed as M-states, by using the right set of heterogeneous cores which cater to the needs of running applications and fit the power envelope. In addition, it also highlight the growing importance of uncore power in total SoC power consumption and the need for a scalable uncore design to completely realize the intended gains. Using a diverse mix of client applications and an experimental heterogeneous platform, we show that heterogeneous CMPs can be used to provide a superior solution for client devices by providing significant performance and power improvements.

Acknowledgement

This work was partially supported by Intel Science and Technology Center for Embedded Computing (ISTC-EC) funding.

References

[1] NVIDIA, Variable SMP: a multi-core CPU architecture for low power and high performance, White paper, 2011.
[2] P. Greenhalgh, Big.LITTLE processing with ARM CortexTM-A15 & Cortex-A7, White paper, ARM, September 2011.
[3] M.D. Hill, M.R. Marty, Amdahl's law in the multicore era, Computer 41 (7) (2008) 33–38.
[4] D. Koufaty, D. Reddy, S. Hahn, Bias scheduling in heterogeneous multi-core architectures, in: Proceedings of the 5th European Conference on Computer systems, EuroSys'10, ACM, Paris, France, 2010, pp. 125–138.
[5] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, D.M. Tullsen, Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction, in: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, MICRO36, IEEE Computer Society, Washington, DC, 2003, p. 81.
[6] J.C. Saez, M. Prieto, A. Fedorova, S. Blagodurov, A comprehensive scheduler for asymmetric multicore systems, in: Proceedings of the 5th European conference on Computer systems, EuroSys'10, ACM, Paris, France, 2010, pp. 139–152.
[7] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, D. Burger, Dark silicon and the end of multicore scaling, in: Proceedings of the 38th annual international symposium on Computer architecture, ISCA'11, ACM, San Jose, CA, 2011, pp. 365–376.
[8] V. Pallipadi, A. Starikovskiy, The ondemand governor: past, present and future, Linux Symposium 2 (2006) 223–238.
[9] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, R. Rajkumar, Critical power slope: understanding the runtime effects of frequency scaling, in: Proceedings of the 16th international conference on Supercomputing, ICS'02, ACM, New York, NY, 2002, pp. 35–44.
[10] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, F. Rawson, Application-aware power management, in: 2006 IEEE International Symposium on Workload Characterization, 2006, pp. 39–48.
[11] D.C. Snowdon, E. Le Sueur, S.M. Petters, G. Heiser, Koala: a platform for OS-level power management, in: Proceedings of the 4th ACM European conference on Computer systems, EuroSys'09, ACM, Nuremberg, Germany, 2009, pp. 289–302.
[12] A. Weissel, F. Bellosa, Process cruise control: event-driven clock scaling for dynamic power management, in: Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES'02, ACM, Grenoble, France, 2002, pp. 238–246.
[13] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, G. Srinivasa, The forgotten 'uncore': on the energy-efficiency of heterogeneous cores, in: Proceedings of the 2012 USENIX Conference on Annual Technical Conference, USENIX ATC'12, USENIX Association, Boston, MA, 2012, p. 34.
[14] G. Dhiman, K.K. Pusukuri, T. Rosing, Analysis of dynamic voltage scaling for system level energy management, in: Proceedings of the 2008 Conference on Power aware Computing and Systems, HotPower'08, USENIX Association, San Diego, CA, 2008, p. 9.
[15] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. Rubio, F. Rawson, J. Carter, Architecting for power management: the IBM POWER7 approach, in: 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), 2010, pp. 1–11.
[16] B. Goel, S. McKee, R. Gioiosa, K. Singh, M. Bhadauria, M. Cesati, Portable scalable per-core power estimation for intelligent resource management, in: 2010 International Green Computing Conference, 2010, pp. 135–146.
[17] S. Srinivasan, R. Iyer, L. Zhao, R. Illikkal, HeteroScouts: hardware assist for OS scheduling in heterogeneous CMPs, SIGMETRICS: Performance Evaluation Review 39 (2011) 341–342.
[18] V. Spiliopoulos, S. Kaxiras, G. Keramidas, Green governors: a framework for continuously adaptive DVFS, in: 2011 International Green Computing Conference and Workshops (IGCC), 2011, pp. 1–8.
[19] A. Fedorova, J.C. Saez, D. Shelepov, M. Prieto, Maximizing power efficiency with asymmetric multicore systems, Communications of the ACM 52 (12) (2009) 48–57.
[20] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, K.I. Farkas, Single-ISA heterogeneous multi-core architectures for multithreaded workload performance, in: Proceedings of the 31st annual international symposium on Computer architecture, ISCA'04, IEEE Computer Society, Washington, DC, 2004, p. 64.
[21] M.A. Suleman, O. Mutlu, M.K. Qureshi, Y.N. Patt, Accelerating critical section execution with asymmetric multi-core architectures, in: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'09, ACM, Washington, DC, 2009, pp. 253–264.
[22] S. Balakrishnan, R. Rajwar, M. Upton, K. Lai, The impact of performance asymmetry in emerging multicore architectures, in: Proceedings of the 32nd International Symposium on Computer Architecture, 2005 (ISCA'05), 2005, pp. 506–517.
[23] T. Cao, S.M. Blackburn, T. Gao, K.S. McKinley, The yin and yang of power and performance for asymmetric hardware and managed software, in: Proceedings of the 39th International Symposium on Computer Architecture, ISCA'12, IEEE Press, Portland, OR, 2012, pp. 225–236.
[24] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, J. Emer, Scheduling heterogeneous multi-cores through performance impact estimation (pie), in: 2012 39th Annual International Symposium on Computer Architecture (ISCA), 2012, pp. 213–224.
[25] N.B. Lakshminarayana, J. Lee, H. Kim, Age based scheduling for asymmetric multiprocessors, in: Proceedings of the Conference on High Performance Computing Networking Storage and Analysis, SC'09, ACM, Portland, OR, 2009, pp. 25:1–25:12.
[26] T. Li, D. Baumberger, D.A. Koufaty, S. Hahn, Efficient operating system scheduling for performance-asymmetric multi-core architectures, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC'07, ACM, Reno, NV, 2007, pp. 53:1–53:11.
[27] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, S. Hahn, Operating system support for overlapping-ISA heterogeneous multi-core architectures, in: 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), 2010, pp. 1–12.
[28] V. Kazempour, A. Kamali, A. Fedorova, AASH: an asymmetry-aware scheduler for hypervisors, in: Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'10, ACM, Pittsburgh, PA, 2010, pp. 85–96.

[29] Y. Kwon, C. Kim, S. Maeng, J. Huh, Virtualizing performance asymmetric multi-core systems, in: Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA'11, ACM, San Jose, CA, 2011, pp. 45–56.
[30] K. Chakraborty, P.M. Wells, G.S. Sohi, A case for an over-provisioned multicore system: energy efficient processing of multithreaded programs, Tech. Rep. CS-TR-2007-1607, University of Wisconsin-Madison, August 2007.
[31] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, M.B. Taylor, Conservation cores: reducing the energy of mature computations, in: Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems, ASPLOS'10, ACM, Pittsburgh, PA, 2010, pp. 205–218.
[32] D.G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, V. Vasudevan, Fawn: a fast array of wimpy nodes, in: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, SOSP'09, ACM, Big Sky, MT, 2009, pp. 1–14.

Vishal Gupta is a Ph.D. candidate in the College of Computing at Georgia Institute of Technology, Atlanta. He received his M.S. in computer science from the University of North Carolina at Chapel Hill in 2008, and a B.Tech. from the Indian Institute of Technology (IIT) Madras, India in 2006. He is a recipient of Intel PhD fellowship award for the year 2012-2013. His research interests include operating systems, virtualization, and distributed systems with a focus on systems software targeting future heterogeneous multi-core architectures in his recent research activities.

Paul Brett is a senior software engineer in the Operating System Research group at Intel Labs where his focus has ranged from Operating System development for the PlanetLab distributed testbed to studying the design, modeling and implementation of heterogeneous multi-core architectures. His professional interests include virtualization, distributed systems, and system software. Paul has a first-class honors degree in systems engineering from the Open University, UK.

David A. Koufaty was born in Venezuela in 1966. He received the B.S. and M.S. in computer science from the Universidad Simon Bolivar in Caracas in 1988 and 1991, respectively, and a Ph.D. from the University of Illinois at Urbana-Champaign in 1997. After completing his Ph.D. he joined Intel Corporation. Between 1997 and 2004 he was part of the microarchitecture and performance team responsible for designing multiple server and desktop processors and was a key developer of Hyper-Threading Technology. Since 2005 he has been with Intel Labs as a research scientist in the Systems Architecture Lab, his most recent research focusing on heterogeneous architectures. His primary interests are in architecture, system software and performance analysis.

Dheeraj Reddy is a research scientist at Intel Labs working on heterogeneous architectures and operating systems. His professional interests include operating systems, processor micro-architecture, data networking and high-performance simulation and emulation. He graduated with a Ph.D. from Georgia Institute of Technology in 2007.

Scott Hahn is a Principal Engineer in Intel Labs. Scott joined Intel in 1994 and spent much of his career working on various aspects of computer networking. Scott started working at Intel in the Supercomputer Systems Division where he was responsible for developing Intel's IP over ATM solution for Intel's TeraFLOP super computer and has been active in defining several network management technologies including chairing an IETF working group and co-authoring several Internet standards-track RFCs. Scott joint Intel Labs in 2005 and has most recently been focused on various aspects of heterogeneous architectures.

Karsten Schwan is a Regents' professor in the College of Computing at the Georgia Institute of Technology. He is also the Director of the Center for Experimental Research in Computer Systems (CERCS). He received his M.S. and Ph.D. degrees from Carnegie-Mellon University in Pittsburgh, Pennsylvania. He established the PArallel, Real-time Systems (PARTS) Laboratory at the Ohio State University. At Georgia Tech, his work ranges from topics in operating and communication systems, middleware, parallel and distributed applications, and high performance computing.

Ganapati Srinivasa is a senior principal engineer with the Intel Architecture Group (IAG) at Intel Corporation. Currently, he is leading research activities on heterogeneous computing for energy and power efficiency. He graduated from Indian Institute of Science in 1983 from School of Automation. He has worked at WIPRO in the performance computing and aerospace areas; at Texas Instruments-India lead their Design Automation software development; at Microsoft member of team developing OS. In 93, he joined Intel where he led the efforts on VLSI CAD and created the Area Routing concepts for Intel's microprocessor design and led Xeon Architecture work for the past 10 years.