CSR: Medium: Collaborative Research: Architecture and System Support for Power-Agile Computing

NSF Proposal
15 Oct 2013

Co-PI: Geoffrey Challen (*University at Buffalo*), Co-PI: Mark Hempstead (*Drexel University*)

As energy management on energy-constrained devices continues to challenge researchers and frustrate users, device designs are addressing the problem by integrating more hardware components that can trade off energy and performance. Dynamic voltage-and-frequency scaling (DVFS) allows CPUs and memory to trade off speed and energy, buffering and polling rates allow radios to trade off latency and energy, and screen refresh rates allow displays to trade off quality and energy. And as the Dark Silicon utilization wall forces systems to choose what parts of the CPU to operate, the already-large configuration space will explode. This proposal refers to the emerging class of devices integrating multiple energy-proportional components as *power-agile*, reflecting their potential ability to adaptively reallocate energy usage between components to improve performance and save energy. But as energy-management features proliferate, new interfaces enabling coordination between applications, the operating system (OS), and hardware are urgently needed to realize the potential energy and performance benefits.

Intellectual Merit: Our proposal describes a new architecture for power-agile systems with both novel interfaces that cleanly separate energy management responsibilities and a new approach to energy allocation driven by differences in hardware energy efficiency. Applications use *resource requests* to allocate energy between hardware components, making their resource needs explicit. The OS manages energy by using the application's priority to determine an *inefficiency allocation*, which controls how much extra energy the application can consume to improve performance. Hardware energy usage is controlled through per-component *energy constraints*, which facilitate OS control while allowing components to maximize constrained performance. By improving energy coordination and allocation, the system achieves energy-efficient performance not currently possible.

Enabling power agility requires research at both the application-OS and OS-hardware boundary. At the application-OS boundary we will invent a new interface allowing applications to allocate energy between components that expose energy-efficiency tradeoffs. This interface requires new ways to describe energy balance between components, language support for programmer annotations to guide the tuning process, and libraries of algorithms to support unannotated applications. Resource request traces will also help inform hardware design and consumer purchases in exciting and novel ways. At the OS-hardware boundary, a new energy management interface must be invented allowing the operating system to effectively set energy constraints, and new ways to support this interface in hardware must be explored.

While resource requests allow applications to adjust energy balance between components, the OS must remain in control of total energy usage in order to prioritize energy between applications and over time. Because the energy efficiency of many hardware components changes along with their energy-performance settings, we propose to investigate inefficiency as an novel energy allocation mechanism. This approach has the potential to address many of the limitations of previous attempts at OS energy management.

Broader Impact: The proposed broader impact activities will excite the next generation of computer scientists about power-agile design and build a shared knowledge and development base within the energy management community. First, a publicly-available component energy usage database will be established and seeded with both device and component energy measurements and workloads developed during the project. This will meet a critical need in the energy management community, as reliable numbers for components are difficult to obtain and few appropriate workloads are available. Standardizing and sharing this information will accelerate research in this area. Second, support for energy-proportional components and Dark Silicon features will be added to the popular `gem5` simulator and made available for other researchers to use. Finally, a new graduate course on power-agile computing will be developed and taught by the co-PIs.

Keywords: (1) **energy management**; (2) **distributed systems**; (3) **mobile systems**; (4) **smartphones**.

INTELLECTUAL MERIT

Energy-constrained smartphones are proliferating rapidly, with the International Data Corporation (IDC) reporting that one new unit for every 30 people on earth was shipped by manufacturers during the second quarter of 2013 alone [30]. Energy management on these devices, however, remains a major challenge, with consumers citing battery lifetime as their top concern with today's smartphones [53].

As a result, energy-constrained devices are integrating multiple hardware components presenting energy-performance tradeoffs: CPUs and memory that can scale voltage and frequency to save energy as they slow down; radios that can tune their polling rates and idle timeouts to save energy by increasing latency; and screens that can dim or reduce refresh rates to save energy by reducing quality. Multiple energy-performance knobs create the potential for applications to choose the right balance of component settings to maintain acceptable performance while saving as much energy as possible. We refer to this ability as *power agility*[1]. Figure 1 shows how a power-agile device would operate, with the application allocating energy between components while the OS controls overall usage. The energy saved by power agility can improve device lifetimes or accelerate application performance.

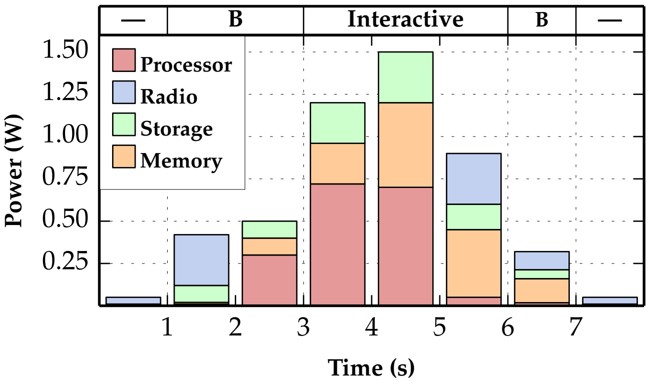


Figure 1: **Example of power agility.** Numbered labels below describe the energy usage graph above.

0. When idle components consume minimal power.
1. Radio and storage power consumption increase as a background task begins receiving and storing data.
2. Once data is received, power shifts to the processor, storage and memory as processing begins.
3. When the user begins interacting with the application, the OS boosts its priority, allowing it to consume more energy. The balance between components, however, remains the same as the application requested previously.
4. The application continues rebalancing energy usage between components.
5. When the applications begins offloading data, radio and storage usage increase, storage usage decreases.
6. As the interactive session completes, the applications priority and total energy allocation are reduced. Again, balance between components is preserved.
7. The background task completes, idling the phone.

Unfortunately, today we are a long way from achieving power agility. While tuning single components such as CPUs using dynamic voltage and frequency scaling (DVFS) has been studied for a decade [31, 18, 32, 34], OSes continue to use simple but ineffective approaches that neither coordinate with applications nor isolate energy usage between them [33, 25]. When multiple energy-proportional components are present, they are usually tuned independently, without considering cross-component interactions. Tuning is done without application input, forcing the OS to guess application performance requirements [21]. And controlling energy usage is complicated by the fact that energy is treated as a side effect resulting from the interaction between performance settings and application workloads, forcing the OS to reverse engineer settings that achieve the desired energy usage. As a result, today's operating systems cannot achieve power agility, and Dark Silicon [17] will further expose this deficiency, since including more functionality than can be activated forces the system to make even more energy-performance tradeoffs.

Our proposal outlines a novel hardware-software system architecture capable of achieving power agility and improving energy management on today's and tomorrow's energy-constrained devices. Section 1 identifies problems with existing approaches to energy management that prevent them from achieving power agility and motivate our approach. In Sections 2 and 3 we describe the two core components of our proposed solution: a novel way of prioritizing energy usage using *inefficiency* and new *interfaces* enabling coordination between the application, OS, and hardware. Section 4 describes the new definitions of power agility and energy proportionality needed to evaluate power-agile systems. We compare and contrast our approach with previous work in Section 5. Section 6 describes the project's broader impacts and how they will benefit society, and Section 7 concludes with a project plan and deliverables.

[1]We use "power agility" rather than "energy agility" to reflect the near-instantaneous adaptation power agility describes.

1 — MOTIVATION

Our key insight is that the way that many hardware components scale energy and performance produces differences in *energy efficiency*. Processors using DVFS, for example, use less energy per instruction when running at a lower frequency and supply voltage; future DVFS memory chips will behave similarly. Radios are more efficient at receiving packets when polling slowly, at the price of increased latency. Table 1 shows that many components found in smartphones can make one or more of these tradeoffs between energy efficiency and performance.

Component	Attribute	Method
Processor	Speed	DVFS.
Memory	Bandwidth	DVFS.
Radio	Latency	Tuning polling intervals and opportunistic timeouts.
	Bandwidth	Through protocol choice, antenna count, or encoding selection.
Screen	Quality	Adjusting brightness and refresh rate.
Audio	Quality	Via sampling rate.

Table 1: **Hardware component exposing an efficiency tradeoff.** Most components found in smartphones present efficiency tradeoffs between one or more performance attributes.

Despite this hardware reality, existing software approaches fail to consider or manage efficiency. Below we discuss limitations of current energy management approaches and use them to motivate our novel approach which allocates inefficiency through coordination between applications, the OS, and hardware.

1.1 — Limitations of Frequency Governors and Rate Limiting

To adaptively manage energy usage, Linux provides multiple *frequency governors*, each with a different algorithm for adjusting CPU frequency at runtime. A representative and relevant example is the "interactive" governor used by Android smartphones. It monitors CPU utilization periodically, and if the utilization over a previous window exceeds a threshold it immediately increases the CPU frequency to its fastest setting. Once the utilization falls below the threshold, the governor will slowly reduce CPU frequency.

The most serious flaw with frequency governors is their failure to prioritize energy usage between tasks. Once the utilization threshold is breached, all runnable tasks will be able to use the CPU at top speed, regardless of their priority. We have verified on Android that a single lowest-priority task is sufficient to trigger this behavior. Because DVFS CPUs become less efficient as they run faster, granting low-priority tasks access to faster frequencies robs energy from higher-priority tasks. This behavior is a legacy of old CPUs that transitioned slowly between voltage domains, requiring 100K to 10M clock cycles to change settings. As a result of this overhead, governors limit the number of transitions by adapting slowly, too slowly to apply per-task settings. On Android, the minimum time between frequency changes is 80 ms, while the maximum task length is 6 ms. In contrast, future processors with on-chip voltage regulators will switch frequencies in 10s of cycles [35], making the overhead of transitions minimal and enabling rapid adaptation. Governors also do not consider cross-component tradeoffs, making them unsuitable for devices with multiple energy-proportional hardware components.

Multiple research OSes have used *rate limiting* to manage energy usage [22, 57, 69, 40]. Although terminology and mechanisms differ, the basic principle is the same. For each task, the OS maintains an amount of energy the task may consume. If the task exhausts its allocation, it must wait for more energy before running again. The OS can assign rates to processes reflecting their priority, and use global rate control to target a desired system lifetime. Unfortunately, rate limiting also has multiple serious flaws.

First, merely time-shifting energy usage into the future does not reduce overall energy usage on meaningful time scales. At best, it represents a tradeoff between performance and short-term energy usage. At worst, it may actually cause tasks to slow down *and* consume more energy. To see why, consider task running on an energy rate-limited OS. It starts executing, runs out of energy and must wait, continues executing, and repeats this cycle until the task completes. When it does, it has consumed at least the same amount of energy it would have consumed if it had not been periodically stopped, and, if the components it used had high idle power, it may have consumed more energy by preventing the components from entering low-power sleep states. If the task had been a browser trying to load a web page for a waiting user, then even the best-case scenario of sacrificing performance—and wasting users' time—without achieving meaningful energy savings is not acceptable.

Second, effectively assigning rates to tasks is a difficult problem that rate-limiting OSes usually ignore. Consider a video-conferencing client and a text chat client. If they are both well-written, the video-conferencing client will inherently require a higher rate than the chat client to achieve acceptable performance, due to its much heavier use of the network and CPU. But rate-limiting OSes have no way of distinguishing between these two applications in order to assign rates properly. If rate assignment is done using OS priority levels, then at the same priority level the video-conferencing client will lack energy while the chat client is allocated too much. This difficulty makes assigning rates a poor way to prioritize energy usage.

Finally, like frequency governors, rate-limiting fails to recognize differences in hardware energy efficiency that can cause misallocation of energy between tasks with different priorities. If low-priority tasks are allowed to run at the fastest and least-efficient CPU frequency setting, even if they are rate limited they will still consume extra energy that should be reserved for higher-priority tasks. Without controlling the efficiency of hardware components, rate-limiting again achieves both reduced performance and higher energy usage. Making a task slower does not necessarily make it run more efficiently.

1.2 — Limitations of Determining Task Deadlines

Another group of research OSes and energy-management approaches try to minimize energy usage while meeting task-specific *deadlines*, potentially after introducing some bounded performance degradation or slack [39, 13, 6, 10, 19, 55, 20]. While reducing application requirements to deadlines is appealingly simple, it is also a gross oversimplification. Many applications have tasks that do not have strict deadlines, that can be postponed, or can trade off quality for energy usage when needed; deadline assignment runs in to many of the same problems as rate-limiting when trying to determine what deadlines to assign to tasks.

Deadline-driven approaches also force the OS to do error-prone guesswork to determine what energy efficiency tradeoffs to make, instead of leveraging application knowledge. On today's systems, that consist of multiple components that can be tuned independently, these guesses are more and more likely to be wrong. A background synchronization task knows when its performance is dependent on the network, CPU, or memory, but cannot communicate this to the OS. All the OS knows is that it has missed a deadline, and has to try and determine what hardware component to accelerate. Making the wrong choice wastes energy without improving performance.

1.3 — Limitations of Performance-Based Hardware Control

Finally, every approach that attempts to manage energy usage must confront the fact that current hardware interfaces are not designed to facilitate this task. Because the energy consumed by a hardware component over a fixed time interval depends both on the task and the component's performance settings, different tasks running over the same time interval with the same performance settings can consume different amounts of energy. Because hardware components expose their energy efficiency tradeoffs through performance settings, managing energy requires the OS to predict the amount of energy a task will consume before it runs; this is impossible. Because energy is treated as a side effect, rather than a control, there is no way for the OS to ask hardware to maximize performance within some energy constraint, a fundamental requirement for energy-management approaches on energy-constrained devices.

1.4 — Achieving Power Agility

In the remainder of our proposal, we present novel solutions to each of the problems identified above that make power agility possible. To address the inability of frequency governors and rate limiting to allocate and prioritize energy within the OS, we introduce the concept of *inefficiency*, described next in Section 2. Because inefficiency directly reflects differences in hardware component efficiency, it ensures that if tasks run more slowly they also save energy, allowing effective allocation and prioritization. Second, we introduce the *novel interfaces* between the application and OS and OS and hardware required to take the guesswork out of understanding application requirements and communicating them to hardware components. These interfaces are described in Section 3. Together they accomplish two things: they ensure that applications can communicate their performance requirements to the OS (Section 3.1), and that the OS can bound component energy usage while allowing hardware to maximize performance (Section 3.2).

2 — Allocating Inefficiency

Frequency governors, rate limiting, and other OS energy management efforts are limited by by their failure to differentiate between several components of energy usage that each produce different energy management problems. To produce a more useful taxonomy, we divide the total energy required to execute a task on a device, E_{total}, into three components: $E_{total} = E_{min} + E_{extra}$, and $E_{extra} = E_{perf} + E_{waste}$.

- E_{min} represents the minimum amount of energy the task required to complete on this device. Once the choice is made to execute this task, this much energy will eventually be consumed.
- E_{perf} measures the amount of extra energy the task consumed that improved performance by running hardware components at faster and less energy-efficient settings.
- E_{waste} measures extra energy consumed that did not improve performance. If $E_{waste} > 0$, it implies that equivalent performance could have been achieved while consuming only $E_{total} - E_{waste}$.

We further define inefficiency as E_{total}/E_{min}. An inefficiency of 1.0 indicates the most efficiency execution possible, while one of 3.0 indicates that three times more energy was used to execute the task than was strictly necessary[2]. The division of E_{total} into E_{min}, E_{perf} and E_{waste} produces three distinct and separable energy management challenges, each requiring a different approach:

2.1 — Energy waste reduction:

The most obvious goal is to eliminate E_{waste}, since doing so will not affect performance and only reduce energy usage or increase battery life. Energy waste occurs when components are tuned incorrectly for a given workload. For example, for a CPU-bound task, energy waste might occur if the memory was set to run fast and inefficiently while the CPU was set to run slow and efficiently. Because the the extra energy consumed by the memory does not significantly improve performance, it is wasted. Eliminating energy waste requires correctly dividing the energy available to the task between components that expose efficiency tradeoffs. We describe our new interface that enables applications to rebalance energy usage by making *resource requests* in Section 3.1, as well as the tuning libraries and code annotations guiding this process. This interface takes the OS guesswork out of improving energy-constrained application performance.

2.2 — Energy scheduling:

Because E_{min} is a direct result of task scheduling we refer to managing E_{min} as *energy scheduling*. Once a task is allowed to run, there is no way to stop it from consuming E_{min}. While it can be time-shifted, there are only two ways to reduce E_{min}: rewrite applications or improve hardware. While the OS is not in a position to rewrite applications, effective prioritization of energy usage does provide an incentive for applications developers to produce more energy-efficient applications. However, our power-agile architecture does facilitate hardware improvements by controlling hardware components using energy constraints, as described in Section 3.2. This approach allows hardware to appear faster as it becomes more energy efficient without requiring significant changes in how the OS manages energy, while also allowing hardware components to manage their own energy usage on timescales that are not achievable by the OS.

2.3 — Inefficiency allocation:

Finally, the OS must allocate E_{extra} between different tasks, which we call *inefficiency allocation*. For each task, the OS maintains an inefficiency constraint that bounds the amount of extra energy the task may use. A low-priority task may have a inefficiency constraint of close to 1.0, forcing it to run as efficiently as possible, whereas a higher-priority task may be allocated more E_{extra} to improve its performance. Prioritizing E_{extra} between tasks prioritizes energy usage without preventing tasks from running. Instead, we require that low-priority tasks run efficiently, while allowing higher-priority tasks to run inefficiently. Our architecture allows applications to control the *balance* of energy between components to meet their own performance requirements, while allowing the OS to maintain *control* over the total amount of energy used. Allocating inefficiency also ensures that the only reason tasks slow down is to save energy.

[2]Because batteries on energy-constrained devices are typically shared by all hardware components, OS inefficiency allocation considers the inefficiency of the entire device. Inefficiency levels for individual components can also be considered separately, which we return to in Section 3.

2.3.1 — Prioritizing inefficiency between tasks: On power agile architectures the OS uses task energy constraints to prioritizing energy usage between tasks. Given that task priorities represent a well-established way of guiding OS resource allocation, we propose to experiment with mapping them directly to inefficiency constraints. In this way we can leverage all of the existing work on priority determination and interactivity detection to help manage energy usage along with other OS resources. Tasks may request that the OS notify them of their assigned inefficiency constraint at any time through a new system call, and the OS will signal a task when its inefficiency constraint changes.

Allocating inefficiency also addresses another problem faced by rate limiting systems: energy hoarding. Many rate limiting systems mitigate their difficulties correctly assigning task energy rates by allowing tasks to keep unused portions of their allocations for later use. This causes its own set of complications. If the assigned rate is too high, the task can hoard energy that other tasks cannot use. Tasks need not be malicious to do so: overly-conservative approaches to energy planning that reserve more than the task needs create the same problem. Rate limiting systems have proposed a number of ad-hoc mechanisms to address this problem, including leaky buckets—which require their own rates—and making energy allocations revocable at any time—which defeats the purpose of saving energy for the future.

When allocating inefficiency, we can avoid these problems. The simplest approach is to not return unused inefficiency allocations, encouraging tasks to use their entire allocation and run as fast as their inefficiency constraint allows. But because allocating inefficiency does not allow one task to prevent another from running, we can also safely allow tasks to save inefficiency if they believe that their current energy constraint is too generous for their needs. We will investigate which of these approaches is superior.

2.3.2 — Allocating inefficiency over time: Many previous OS attempts at managing energy have had achieving a certain system lifetime as their goal. To achieve a target lifetime, in addition to allocating energy between tasks, they also allocate energy over time. While we believe that these attempts are misguided if they only time-shift tasks—particularly interactive ones—we can also influence system lifetime by setting inefficiency constraints. On power-agile systems, the multiplier between task priorities and task inefficiency constraints controls the rate at which we deplete E_{extra} and, as a result, the balance between performance and system lifetime. The higher the multiplier, the faster the device will consume E_{extra} and the quicker it will exhaust its battery; the lower, the longer the device will last. We plan to explore ways of determining the correct multiplier by observing user behavior.

Note that, in contrast with previous approaches, we cannot meet a target lifetime by controlling inefficiency since this does not control E_{min}. However, on the smartphone platforms we are proposing to investigate, we believe that this is a natural result. When subject to heavy use, smartphone batteries will die more quickly, because the user also plays a direct role in energy allocation.

2.3.3 — Interaction with scheduling: By separating identifying inefficiency and separating E_{min} from E_{extra}, our power-agile architecture should be able to avoid most interactions with the OS scheduler. Tasks can run whenever the scheduler chooses them, but with the inefficiency constraint assigned to them.

While we would like to avoid interactions with scheduling entirely, there are two cases where it is required and beneficial. The first, is when components impose significant transition costs to change energy-performance settings, such as a delay required to migrate a task between big and little cores. The second is when tuning components that have inherently shared performance features, such as memory, which is certain to occur on multi-core systems but can also happen on single-core devices. In both cases, it may make sense to alter the order tasks are run to amortize hardware transition costs or overlap tasks with similar hardware requirements. We will investigate new ways to do this while preserving as much of the existing Linux scheduling algorithm as possible.

3 — NOVEL INTERFACES

Achieving power agility also requires novel interfaces between applications, the OS, and hardware. The new interfaces in our design create a natural separation of concerns between these three energy management stakeholders by making allocation of inefficiency between components explicit and simplifying the task of controlling hardware energy usage. Figure 2 provides an overview of our architecture and the two new proposed interfaces discussed in this section.

Applications request a balance of energy usage across components exposing efficiency tradeoffs by passing a *resource request* to the OS (Section 3.1). In Figure 2, the unmodified application has requested a fast CPU and memory but slow network and storage, whereas the modified application has requested a fast network but slow CPU, memory and storage. Resource requests allow applications to configure the device explicitly and free the OS from the difficult task of guessing task performance requirements and cross-component dependencies. We describe how applications can use programmer annotations, preexisting traces, compiler support, feedback-driven tuning libraries, or a mixture of these approaches to make effective resource requests.

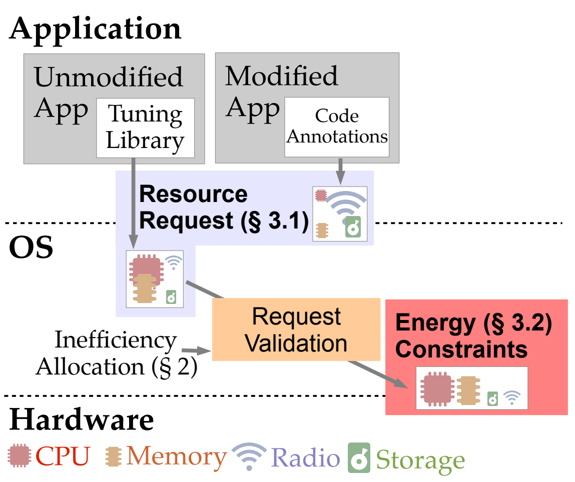


Figure 2: **Our power agile architecture.** The size of the components in the resource requests represent the balance between component energy usage requested by the application.

The OS maintains control over energy usage by validating that the resource request does not exceed each application's target *inefficiency*, as described in Section 2. If the resource request is valid, it is communicated to hardware in the form of per-component *energy constraints* (Section 3.2). Energy constraints allow the OS to control hardware energy usage explicitly and free the OS from having to estimate energy usage based on performance settings. They also provide the guidance that hardware components on energy-constrained systems need in order to determine how to maximize their own performance.

3.1 — Application-OS Interface

Operating multiple components that can each trade off performance for energy efficiency requires explicit application control due to three characteristics of application energy-usage on energy-constrained devices. First, usage is distributed across multiple components. Figure 3 breaks down energy consumption from 77 PHONELAB smartphones, showing that energy usage is well-distributed, with the display, CPU, 3G and Wifi radios all accounting for significant fractions. Usage also shows a great deal of interuser variation resulting from variations in application usage patterns. Since different applications use these components differently, they will want to tune them differently to manage overall device inefficiency.

Second, application performance needs vary over time. Figure 4 shows cycle-per-instruction (CPI) for the `gzip` SPEC2006 benchmark, showing how even this simple application goes through several CPU- and memory-bound phases during execution. More complex interactive applications will have even more temporally varying resource needs. Due to this dynamic behavior, applications will want to reallocate energy usage between components at runtime to achieve good performance.

Third, when multiple energy-proportional components are used by an application, significant energy waste can occur if they are tuned improperly. Figure 5 shows performance and energy usage curves for a simulated smartphone with both CPU and memory capable of DVFS[3]. We compare two SPEC2000 [7] benchmarks: *eon*, which is CPU-bound, and *art*, which is memory-bound. For both benchmarks, the combination of CPU and memory frequency scaling produces an order-of-magnitude range in both the achieved speedup—S, computed with respect to the slowest execution and shown with shading—and the device inefficiency—I, computed as described in Section 2 and shown with contours. However, at each desired speedup only a single pair of settings—shown by the white line—eliminate E_{waste}. All other settings consume energy without improving performance. For example, when running at a speedup of 3, `art` can consume between 20 and 50 times as much energy as its most efficient $E_{total} = E_{min}$ execution, showing both that slow does not mean efficient and the potential for energy waste.

3.1.1 — Resource requests: Eliminating E_{waste} requires a way for tasks to explicitly allocate device inefficiency between hardware components and adjust these allocations as their resource requirements change.

[3]Server designs are already incorporating DVFS memory to address the rising energy contribution of the memory subsystem, and we expect next-generation smartphones to follow shortly.

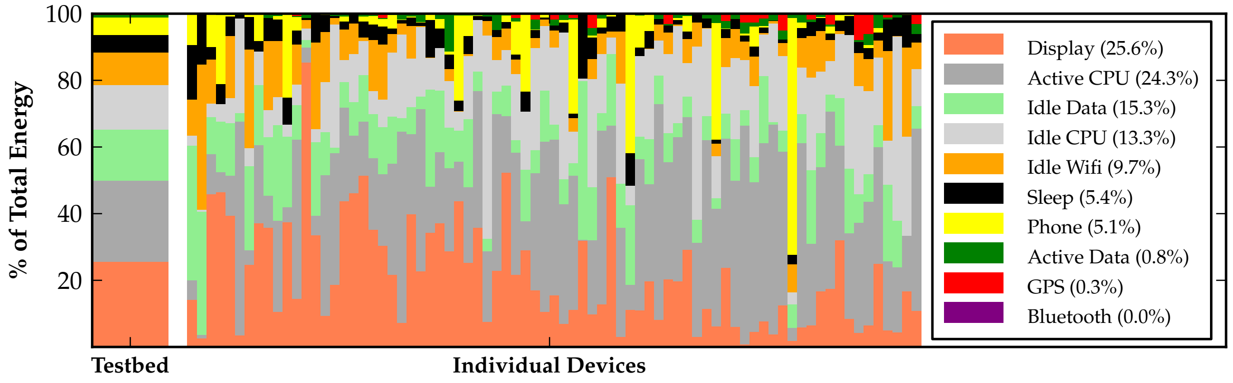


Figure 3: **Per-component energy breakdown for 77 real smartphones collected on PhoneLab.** Energy usage is spread across multiple components and varies significantly between users.

To enable power agility, we propose to invent a new OS interface allowing tasks to make runtime *resource requests*. The format of resource requests must be specific enough to identify particular hardware energy-performance features tasks would like to use, such as initiating a computational sprint [54], general enough to be used on a variety of power-agile devices, and easy for the OS to use to validate that the task's resource request does not exceed its inefficiency allocation. We propose to invent a resource request format that can be used at the application-OS interfaces that meets these criteria and is flexible enough to incorporate the new features emerging on Dark Silicon designs.

One promising resource request format we propose to investigate divides the task's inefficiency allocation into per-component inefficiency allocations. These bound the energy used by each component in the same way that the device inefficiency bounds the energy used by the entire device. For example, an CPU inefficiency allocation of 2.0 indicates that the CPU should not use more than twice the amount of energy that was required to perform the computation. Section 3.2 describes how and wy we also propose to implement inefficiency allocation at the OS-hardware interface.

Resource requests in this format also have the benefit of being straightforward for the OS to validate, because the device inefficiency can be computed as a linear combination of component inefficiencies, each scaled by the component's overall contribution to the total energy usage of the device. Given a device consisting of a set of components $\mathbb{C} = (c^1, c^2, \ldots)$, with each component c^a set to inefficiency level $i^a_{component}$ and consuming e^a in its most efficient state, then the total device inefficiency I_{device} can be estimated as $I_{device} = \sum_a i^a_{component} \cdot e^a$. Requests for invalid sets of resources will fail and the task will be required to try again. When lowering a task's inefficiency allocation, the task will be asked to reallocate inefficiency by making a new resource request. If this request is invalid, the task will be forced to run at a device inefficiency of 1.0 until it makes a valid resource request.

3.1.2 — Making effective resource requests: While power-agile architectures require applications to explicitly manage energy usage by making resource requests, there are multiple ways to accomplish this, including tuning libraries and code annotations. We will explore both of these options.

Adaptive tuning libraries transparently observe application performance to make resource requests at runtime and provide a way to support unmodified binaries. We propose to develop a variety of different tuning algorithms suited to different types of applications or different user expectations. Existing applications can use a default algorithm or an algorithm selected by their developer. It may even be useful to expose this choice to the user, allowing them to assign each application a labeled algorithm at install time and change them during operation, similar to the user-driven adaptation proposed by "application modes" [46].

One example tuning algorithm we propose to evaluate uses a guided search that climbs a performance gradient. At each time step, it tests the sensitivity of the application performance to a shift in component inefficiency allocation, using performance feedback from the OS to help select components to target. For example, if the application has been spending most of its time waiting on the network, the algorithm may try increasing the inefficiency allocated to network latency and reduce the amount allocated to the CPU

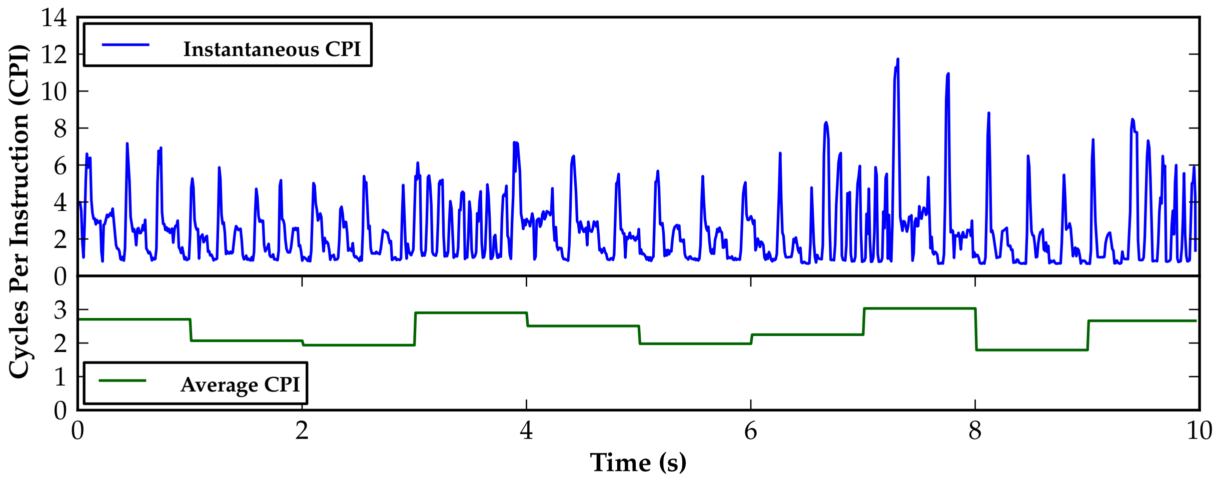


Figure 4: **Example of application transitions.** CPI for `gzip` run as part of the SPEC2006 benchmark suite.

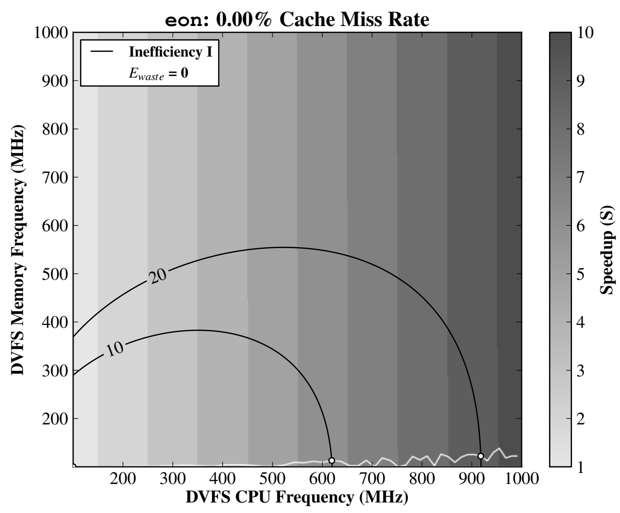
and memory. If performance improves, the algorithm will continue tuning from that point; otherwise, it will backtrack. This process is repeated at regular intervals.

While tuning libraries are an effective way to support legacy applications, they suffer from many of the same limitations as attempts to observe application deadlines from below the OS interface. Therefore, we expect developers to leverage their knowledge to make more effective resource requests. We propose to add support to Android Java for device description annotations, allowing programmers to request different mixes of component energy-performance settings directly. For example, an annotation can alert the OS to the beginning of a computationally-intensive code path, during which radio and storage component performance can be reduced to allow the CPU and memory to run faster. Compilers or offline profiling tools may also automatically add explicit annotations to binaries, similar to previous techniques used for CPU DVFS [67]. Annotations may also be combined with tuning support, providing a way to quickly override the default behavior and allow more rapid adaptation.

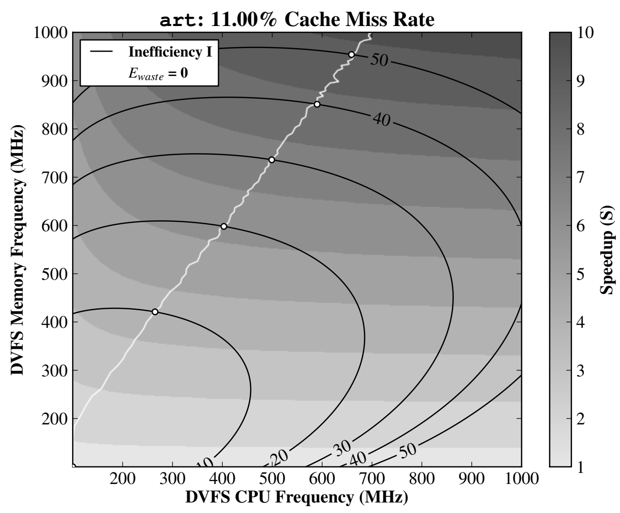
3.1.3 — Resource request traces: Whether the device adaptation is done by a tuning library or direct annotations, the output is time series of resource requests. These traces contain information about application performance requirements that can be used to improve tuning algorithms, evaluate new hardware designs, and even to help smartphone users evalutae new models when upgrading. Once we have completed implementing the resource request interface and adaptive tuning libraries, we will instrument the Android platform used by smartphones on the PhoneLab smartphone testbed to log the resource request traces made by smartphone applications used by PhoneLab participants. Our primary reason for collecting these traces is to use them to evaluate the effectiveness of our tuning algorithms, but we also propose to experiment with using them during smartphone selection, which we describe in more detail in Section 6.3.

3.2 — OS-Hardware Interface

At the hardware interface, instead of treating energy usage as a side-effect of performance settings, we use *energy constraints* to explicitly bound component energy usage. This approach has several advantages. First, it allows the OS to bound energy consumption directly, making it straightforward to ensure that tasks run within their inefficiency allocation. To accomplish this, we propose to investigate using component inefficiency as the mechanism to communicate energy constraints to hardware. Since our proposed resource request format already uses component inefficiencies to allow applications to express cross-component balance, after validating the resource request the OS simply communicates these energy constraint settings to the relevant hardware components. Second, it allows hardware components to maximize performance under the energy constraint without OS involvement. Power-agile systems will be able to integrate new hardware energy management features with minimal alterations to the format of resource requests, OS inefficiency allocation, or application tuning algorithms. This will be increasingly important as new hard-



(a) eon, CPU bound.



(b) art, memory bound.

Figure 5: **Synthetic inefficiency and performance curves for a device with DVFS CPU and memory.** To represent the four-dimensional space, shading is used to indicate speedup S and the labeled contours identify inefficiency I. The white curve identifies the optimal frequency settings for any desired inefficiency allocation. For both benchmarks, an order of magnitude performance and energy usage difference can be achieved, but most settings are not optimal.

ware components are already beginning to include advanced features such as computational sprinting [54], memory DVFS [11] and 3G tail adaptation [68]. As Dark Silicon promises to expand this space even further, a better energy management interface between the OS and hardware is needed.

3.2.1 — Energy constraints using inefficiency: When a hardware component is assigned an energy constraint, it begins trying to maximize performance under that constraint and continues until the constraint is changed. Since the energy consumed depends on workload characteristics that hardware components cannot predict, meeting the constraint is an adaptive process and components will tend to converge to it over time. We propose to invent the algorithms necessary to perform this adaptation, focusing on rapid convergence which will allow energy constraints to be changed often without incurring significant error.

In order to bound hardware energy usage using inefficiency, each component must continuously estimate E_{min}. For active portions of energy consumption, such as the energy used by each CPU instruction, a simple running total may be sufficient. Idle energy is somewhat more complex to account for because it depends on the amount of time required to execute a task, which will vary based on the inefficiency constraint the hardware component is assigned, requiring a scaling factor to estimate correctly. We will experiment with novel techniques to estimate inefficiency in components simulated by `gem5`.

While components should not exceed the energy constraint, they may not always be able to meet it; it might be too high relative to the task's actual energy usage. For example, if a task does not use the memory, than any inefficiency setting much larger than 1.0 will be difficult for the memory to meet, even if it it completes the work is it assigned as quickly and inefficiently as possible. Since this feedback can indicate that energy is not correctly allocated between multiple components, we will experiment with allowing hardware to provide information to the OS about the inefficiency that is actually achieved during a period of time.

3.2.2 — Supporting legacy hardware: While we believe that energy constraints are the right interface for energy management on future hardware components, we also want to enable power agility on existing devices. To do this, we propose to implement energy constraints in open source device drivers for existing smartphone hardware components. These drivers will have to determine the performance settings required to meet the inefficiency constraint and monitor energy usage in the way appropriate the component that they support. This temporary solution allows us to bring power agility to existing devices.

4 — Measuring Agility

To evaluate power-agile systems, we must define power agility. And because power agility relies on capabilities of the device's individual hardware components, evaluating power agile systems also requires an empirical definition of energy proportionality. Surprisingly, despite widespread interest in energy-proportional systems, a precise definition for this term does not exist. Below we first define power agility and then provide an operational definition of energy proportionality.

4.1 — Defining Power Agility

We define power agility as the ability of applications, the OS, and hardware to collaborate to manage energy usage on devices consisting of multiple energy-proportional components. Given the battery-powered nature of the mobile devices we are targeting, we use energy as our constraint and performance as our maximization goal[4]. In general, the energy and performance of a task depend on the device's hardware capabilities and how they are used. Given a device consisting of a set of components $\mathbb{C} = (c_1, c_2, \ldots, c_i)$, with each component c_i presenting one or more power-performance levels $c_{i,1}, c_{i,2}, \ldots, c_{i,j}$, we define a tuning strategy $S = ((t_0, c_{i,j}), (t_1, c_{k,l}), \ldots)$ as the time series of component power-performance levels that are used during the execution of the task, with $(t_i, c_{j,k})$ indicating that component c_j was set to state k at time t_i. Then given a task T, energy constraint E_{max}, and performance measure P, the most power-agile tuning strategy is the one that maximizes P within the energy constraint. Similarly, the most power-agile device for this task is the one where the most power-agile tuning strategy produces the best performance. Our definition allows us to measure power agility while varying both device capabilities and tuning strategies.

4.2 — Defining Energy Proportionality

The degree to which a device can achieve power agility depends on the energy proportionality of its hardware components. As Table 1 shows, multiple components are performance proportional and many can trade off multiple performance aspects with energy usage, but specific components achieve this in different ways, to different extents, and with differing overheads. A memory chip might decrease energy usage by reducing either capacity or bandwidth. A CPU might only be able to scale frequency and voltage through a limited range. A heterogeneous multi-core architecture might offer a low-power core but impose a transition cost to activate it. Despite the richness of the power-performance design space, energy-proportional components are produced without considering how they will integrate into a complete device.

Evaluating energy-proportional components requires overcoming a basic challenge: a precise empirical definition of energy proportionality must be formulated and tested. This definition will allow us to evaluate our efforts and guide design of future energy-proportional components, power-agile architectures, and power agility algorithms. More specifically, we seek a definition of energy proportionality allowing device architects to choose components with useful energy-saving features and energy-performance knobs.

To motivate our definition, we introduce two candidate hypothetical energy-proportional memory chips. Figure 6 displays the relationship between energy consumption and performance for each. We note that:

1. **Performance has multiple dimensions.** The banked memory chip achieves proportionality by powering down transistors, trading off capacity for energy. The frequency-scaled chip achieves proportionality by reducing transistor voltage, trading off latency for energy.
2. **The range over which components are proportional varies.** The banked-memory chip is energy-proportional from 125 to 500 mW; the frequency-scaled chip from 250 to 500 mW.
3. **The smoothness with which components trade power for performance varies.** The banked-memory chip requires costly power-gating circuitry, meaning that it shuts down memory only in large 256 MB chunks. In contrast, the frequency-scaled chip can tune its operating frequency smoothly.

The first observation motivates us to split the energy-proportionality metric along performance axes appropriate for each component. Many common axes will likely emerge driven by transistor fundamentals:

[4]Energy management approaches aimed at servers frequently use performance as the constraint while trying to minimize energy usage, which is more appropriate for wall-powered machines frequently operating under service-level agreements. While our focus is on mobile devices, our definition can easily be reframed for performance-constrained systems.

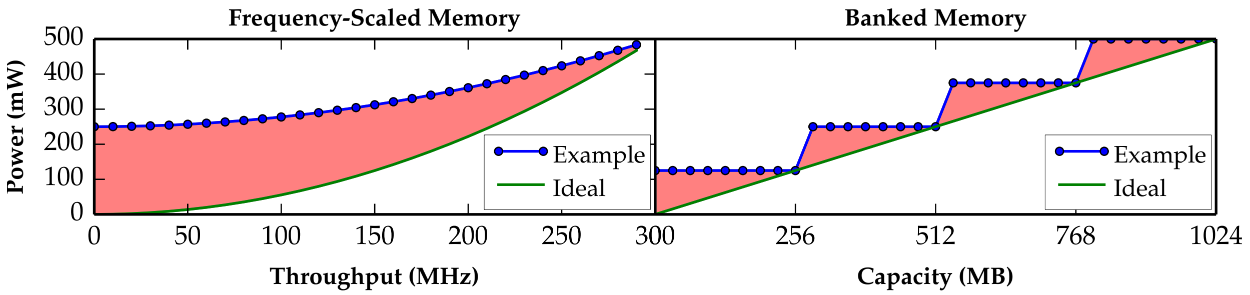


Figure 6: **Example power-performance curves for two 1GB energy-proportional memory chips.** Red filled areas show the degree of dis-proportionality with respect to an ideal component, as discussed in Section 4.2.

memory and networking components have both latency and throughput axes, where output devices have quality axes. After separating performance attributes, we measure the proportionality of a real component by comparing it to an ideal component. For the banked memory chip, the idealized component could disable individual bytes of memory. For the frequency-scaled chip, the idealized component would smoothly scale to zero as usage decreased. While ideal components are not achievable—both of these hypothetical memory chips ignore circuit realities—they serve as useful benchmarks.

Finally, given a single performance parameter P, the measured relationship between energy and performance for a candidate component C $E_C(P)$, and the idealized relationship between energy and performance for an idealized component I $E_I(P)$, we define the disproportionality D as: $D = \int_P E_C(P)\mathrm{d}P - E_{Ideal}(P)\mathrm{d}P$. Figure 6 graphically illustrates the disproportionality calculation for the two candidate memory chips.

5 — Related Work

We are not aware of a similar cross-layer approach to energy management that engages applications, the OS, and hardware, but many recent advances are complementary to our work and can be incorporated into power-agile architectures. Examples include work on energy-aware management [49], sleeping during idle periods to conserve power [47], reducing storage energy consumption through dynamic consolidation [64], and delivering energy-proportional computing using non-energy-proportional components [62]. Our own previous work on achieving power-agility through heterogeneity [8] is also complementary, although we anticipate that the continuing addition of energy-management features to single components will make heterogeneity-based adaptation less necessary on future devices.

5.1 — Single-Component Scaling

While tuning single components is not sufficient to provide power agility, research into single component scaling can inform the design of hardware controllers that attempt to maximize performance under an energy constraint. Several techniques for homogeneous multi-core processors have been studied that use predictors such as instructions-per-cycle (IPC) to decide when to use DVFS or migrate threads [56, 4, 32, 31]. These hardware-only approaches are not aware of the effect on other threads and application-level performance. Heterogeneous multicore processors containing cores with different energy-performance tradeoffs pose a challenge to scheduling. The PIE model captures the amount of instruction level parallelism and memory parallelism in a workload and predicts if it will perform better on a big or little core [63].

As the contribution of memory to overall system energy usage has increased, multiple systems have tried to control memory energy consumption [60, 70, 16, 37, 48, 45]. Many efforts focus on adapting to various active and low power states of memory based on application memory performance requirements, while some [11, 14, 15] scale memory frequency. MemScale [15] scales voltage for memory channels and the memory controller but scales only frequency for DRAM devices. A separate approach [11] scales voltage and frequency through a control algorithm that runs periodically to choose the best settings.

5.2 — Multi-Component Scaling

Integrated power management of CPU and memory are considered in recent works [39, 13, 6, 10, 19, 55, 20]. Gupta et al. [39] explore micro-architectural adaptations and idle low-power states of memory to conserve energy within performance degradation bounds. Xiaobo et al. [19] evaluate gains in energy consumption using DVS for CPU and low power states of memory.

One approach scaling both CPU and memory is CoScale [13], which implements DVFS for memory channels and the memory controller and FS for DRAM devices. Its control algorithm profiles application performance by reading performance counters at the beginning of every scheduling quantum. Using a greedy heuristic, CoScale attempts to choose frequency settings for both the CPU and memory that minimum energy usage without violating a performance bound. While CoScale advances the state-of-the-art by considering multiple components during tuning, it suffers from the same problems with application visibility and hardware control as many other systems.

5.3 — Systems and Software Approaches

Responding to the energy limitations of battery-powered mobile and embedded devices, researchers in the mobile systems and sensor networking communities have proposed energy-aware operating systems and decentralized energy management strategies. networks [9, 36, 65, 59, 44]. Systems such as Odyssey [22], PowerScope [23], Currentcy [69], Pixie [41], Eon [59], Levels [36], and Cinder [57], have addressed measuring or adapting to energy variations on battery-powered devices using various kinds of rate-limiting and guided application adaptation. None of these systems have overcome the problems we have identified with rate limiting or enabled the kind of cross-component adaptation power agility requires.

FAWN [2] and Amdahl-balanced blades [61] represent single systems where cross-component energy usage and performance have been balanced for a single application, not through online adaptation, but through static hardware choices. They show the potential of power agility to save energy but require workloads with fixed energy balance requirements. Other systems, such as Quanto [24] and Carat [52] provide new ways of profiling and measuring application energy usage but not not attempt to control it.

6 — Broader Impacts

The broader impact components of our proposal (1) enhance research infrastructure through structured dissemination of research results; (2) advance discovery and understanding by promoting teaching and learning; and (3) contribute to shared educational infrastructure.

6.1 — Component Power Database

Component energy usage information is difficult to obtain. Because datasheets rarely present useful power numbers, determining the energy proportionality of a particular component requires acquiring and measuring it under appropriate workloads. This is a time-consuming task and prone to repetition when results are not distributed. Unfortunately, power measurements—no matter how thorough—are not considered research results and frequently not disseminated. To address this problem, we will establish a shared database of component power measurements, seeded with output of our measurements of component energy proportionality. Aggregating this information will greatly accelerate research in this area by eliminating the duplication of effort necessary to understand component power consumption. Multiple groups will have a place to share power measurements gathered using common experimental procedures under identical workloads, enabling "apples to apples" component comparisons. We will contribute our own power measurements and workloads to the database, and advertise its availability to spur adoption.

6.2 — `gem5` Development

The team will conduct many of its simulation studies using the `gem5` full system simulator and will have to make significant changes to `gem5`; these changes will be released to the large `gem5` user base. The simulation environment is used widely by the research community and industry, including by ARM, Intel, IBM, HP, and AMD [5]. There is no existing simulation environment that supports cross-layer power management from the application, OS to hardware. System designers and researchers will benefit from the hardware component models; interface and control methodologies; driver and operating system kernels that will be contributed by the research team. The team is already in active collaboration with researchers at ARM, former colleagues of co-PI Hempstead, who regularly contribute to `gem5`.

6.3 — Improving Smartphone Selection and Design

The resource request traces described in Section 3.1.3 that emerge as an output of tuning and code annotation are primarily used by the OS to balance energy usage between components. However, these traces are also valuable in two other ways: helping users choose new phones and companies design better devices.

Today's consumers face more and more choices in the smartphone market, but lack ways to determine which devices are the best fit for their applications and usage patterns. By aggregating their resource request traces and matching them against the energy-saving features of smartphone models under consideration, users can be guided toward smartphones with necessary features and without unnecessary ones, and where the applications they use will perform well. Aggregated across multiple users, resource request traces can help hardware vendors determine the need for new energy-saving features and identify performance bottlenecks, leading to better phone designs.

6.4 — Curriculum Development Activities

Beginning in Year 2 the co-PIs will co-teach a joint graduate course entitled "Power Agile Computing: Architecture Design and System Support". The course will focus on the hardware-software co-design issues relevant to power-agile architectures. Enrolled students will undertake research projects spanning the hardware-software divide in interinstitution groups.

Classroom time will be divided between presenting and discussing relevant related work and modules specifically addressing relevant research outputs of this project. Video conferencing will be used to link the separate classrooms. In Year 2, the `gem5` simulator with integrated component power models and a set of appropriate workloads will be available, and the course will address resource requests and tuning algorithms. Year 3 will feature similar tools but focus on hardware energy constraints. Finally, in Year 4 the project will focus on integrating and evaluating the overall power agile system. This course will also develop two essential engineering skills. First, it will build understanding between the hardware and software communities. Second, it will teach students the tools and management skills necessary to manage remote collaborations, critical in the increasingly global technology landscape.

The instructors will be able to leverage the power-aware material designed by co-PI Hempstead for his course at Drexel: "ECEC 623: Advanced Computer Architecture". Using the textbook by Kaxiras and Martonosi and current research papers, the course introduces students to mechanisms of power consumption and power-management techniques [34]. Students complete independent projects which have included low-power circuit design, power measurement of desktop and mobile systems, and learning-based power models for GPUs.

6.5 — Diversity and Outreach

When admitting graduate students, the co-PIs are inspired by the approach of the CCC/CRA NSF-funded CI Fellows program, which improved representation of women and traditionally-underrepresented minorities increased simply by reviewing those applications first. The co-PIs are copying this approach when admitting graduate students and recruiting undergraduates. Over the past year, seven women have contributed to our research groups: Rizwana Begum (Drexel), a PhD candidate on this project; Maria Gonzalez (Drexel), an undergraduate REU student; Vinu Charanya and Anuja Raval (University at Buffalo), Masters students; Sonali Batra and Anudipa Maiti (University at Buffalo), PhD students; and Na Gong (University at Buffalo), a PhD student who contributed directly to preliminary work on this project. We have now expanded our efforts to recruit females and underrepresented minorities earlier in their careers.

Both Drexel University and the University at Buffalo, with their large population of undergraduates from underrepresented groups, are excellent places to cultivate future researchers. This year, co-PI Hempstead is serving as a mentor for a Liberty Scholar, an underprivileged student from Philadelphia receiving a full scholarship from Drexel. As research has shown, it is through personal contact, exposure to challenging problems and mentoring that undergraduates come to consider research as a promising career [1, 12].

6.5.1 — Undergraduate Research: The co-PIs are committed to involving undergraduate students in research. Since arriving at Drexel three years ago, co-PI Hempstead has involved four undergraduate students in research resulting in a publication [51]. In the past year, Co-PI Challen has involved two talented undergraduates in the development of PHONELAB and is overseeing research by a third on new interactive instructional techniques. Co-PI Hempstead has mentored undergraduate project teams—including a freshmen team and a senior design team—working on power-agile design projects. As a general philosophy, we integrate undergraduates into our research team by inviting them to participate in weekly paper discussions and giving them development tasks appropriate to their training.

7 — QUALIFICATIONS, DELIVERABLES, AND PLAN

In this section we describe why we are qualified to undertake this project, list project deliverables, and present a plan outlining how we will complete the required tasks within four years.

7.1 — Qualifications and Prior Support

Co-PIs Challen and Hempstead are young investigators that bring both a history of successful collaboration and complementary experience above and below the OS-hardware interface to the project.

7.1.1 — Geoffrey Challen wrote a dissertation on energy management for embedded sensor systems. Along with co-PI Hempstead, he developed PowerTOSSIM [58], an augmented version of the TinyOS [29] simulator TOSSIM [38] enabling application power profiling. Challen's work on Lance, IDEA and Peloton addressed energy consumption at the network—rather than node—level. Lance [66] showed that data-intensive sensor network applications must consider both the cost and value of information when collecting data, and proposed a novel optimization heuristic enabling near-optimal online performance. IDEA [9] demonstrated that a network-wide energy coordination layer could facilitate energy optimizations impossible for a single node to perform alone. Peloton [65] proposed a distributed operating system for coordinated resource management built on state sharing, a distributed energy ticket abstraction, and neighborhood ticket management.

Challen's recent work focuses on smartphone usage characterization using data collected on PHONELAB [50]. His group has designed and deployed software to operate the testbed, and a usage characterization experiment collecting a variety of useful data in order to begin active public experimentation. His current and prior NSF projects include:

1. **PhoneLab: A Programmable Participatory Smartphone Testbed** *(CI-ADDO-NEW-1205656, $1.3M, 06/01/2012–05/31/2015)*—Co-PI Challen leads the PHONELAB project along with co-investigators from the University at Buffalo. PHONELAB is a programmable smartphone testbed providing the power, scale, and realism required to evaluate mobile systems research. Consisting of 288 smartphones, PHONELAB opens for public experimentation in October, 2013.
2. **Travel Support for SenSys 2010** *(CNS-NeTS, $15,000, 10/01/2010–09/30/2011.* Co-PI Challen distributed NSF funding supporting student travel to SenSys'10.

7.1.2 — Mark Hempstead has experience with wireless sensor networks (WSNs), high performance microprocessors, and low power circuit design. Hempstead developed a new system architecture for WSNs incorporating application accelerators for regular tasks and power gating circuits to reduce leakage power [27]. A prototype was manufactured in 130nm CMOS and measured to generate a detailed power model across a wide voltage-frequency range [26, 28]. Measurements revealed a need for a new memory structure called the accelerator store, that dynamically allocates memory to accelerators and automatically manages leakage power [42, 43]. Recent work has leveraged the circuit design experience in Hempstead's group at Drexel to investigate structures for register renaming and file gating in modern processors [3].

Hempstead's recent work has focused on applying heterogeneous and accelerator-based computing to more general purpose domains. His group has developed a custom accelerator for ultra-sound image processing and evaluated it against CPU and GPU implementations in a recent CASES paper [51]. With co-PI Challen, Hempstead's group has also looked at using heterogeneous components to create an energy-proportional system [8]. His current and prior NSF projects include:

1. **AfterBurner: Efficient Performance Scaling via Post-Retirement Processing** *(CCF-1017654, $119K, 09/01/2010–01/01/2012)*—Hempstead lead the AfterBurner project with Professor Amir Roth from the University of Pennsylvania. AfterBurner targets regions of low instruction-level parallelism (ILP) through techniques such as speculative retirement and selective re-execution with the aim to increase the throughput of modern microprocessors. Recent work has demonstrated how reference counting in the rename stage of a out-of-order microprocessor can enable power efficient techniques, such as register file power gating, checkpointing, and move elimination [3]. Support from this project has resulted in two additional publications [8, 51].

7.2 — Deliverables

Our main project deliverable will be Agile Android, a complete power-agile smartphone platform for Android devices running on both real hardware and an augmented `gem5` simulator. Agile Android provides a platform to experiment with the novel interfaces, algorithms, and mechanisms we have described, but requires changes to Linux, the Android platform, and the ability to experiment with hardware modifications which we propose to accomplish by using a cycle-accurate hardware simulator.

Inefficiency allocation will be added to the Linux kernel, along with the resource request interface between applications and the OS; and the energy constraint interface between the OS and simulated hardware. To support legacy hardware and allow us to experiment on real smartphones, we will implement energy constraints for existing hardware components within their device drivers, as described previously.

Support for resource request annotations will be added to the Android platform, and several popular open-source apps included in the platform, including the browser and email client, will be annotated to make resource requests intelligently as they transition between different execution phases. We will also provide a tuning library that implements several different algorithms for generating resource requests.

We will use the popular `gem5` simulator to enable rapid prototyping of our changes at the hardware interface, including energy constraints and the hardware support they require. To experiment with emerging hardware energy management features not yet found on real devices, we will implement several in `gem5`.

7.3 — Plan

How the co-PIs will manage an effective inter-institution collaboration is described in the separate collaboration plan, including how research tasks will be divided between the systems team at the University at Buffalo and the hardware team at Drexel University. Below we present a preliminary task plan describing how research and broader impact tasks will be scheduled over four years.

7.3.1 — Year 1: We will begin by investigating novel interfaces (§ 3) through implementing and testing prototypes. We will also need to determine the format of resource requests (§ 3.1.1), and begin exploring algorithms for our tuning library (§ 3.1.2). In the first year we will also initiate our public component power database (§ 6.1) and begin entering measurements.

7.3.2 — Year 2: During the second year we will focus on designing effective algorithms to allocate efficiency and prioritize allocations between tasks (§ 2), while continuing work on the interfaces begun in the first year. We will also add inefficiency interfaces to several existing device drivers allowing experimentation on real hardware (§ 3.2.2), allowing us to gather resource request traces on PhoneLab (§ 3.1.3). We will also offer the joint graduate course for the first time (§ 6.4).

7.3.3 — Year 3: In the third year work will continue on inefficiency allocation while beginning implementing support for hardware energy constraints (§ 3.2) and `gem5` support for advanced energy-saving features (§ 6.2). We will also begin testing our definitions of power agility and energy proportionality (§ 4) so that they are ready to evaluate the complete system. Finally, we will use the resource request traces gathered in the previous year both to continue improving our resource request algorithms and improving smartphone selection and design (§ 6.3).

7.3.4 — Year 4: Finally, we will integrate all of the innovations into the complete Agile Android platform and perform end-to-end testing and evaluation. We also reserve time in final project year to complete any delayed tasks and ensure that our results are widely disseminated.

REFERENCES

[1] J. Adair, M. Reyes, M. Anderson-Rowland, and D. Kouris. Workshops vs. tutoring: how asu's minority engineering program is changing the way engineering students learn. In *Frontiers in Education Conference, 2001. 31st Annual*, volume 2, pages T4G –7–T4G–11 vol.2, 2001.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 1–14, New York, NY, USA, 2009. ACM.

[3] S. Battle, D. Hilton, M. Hempstead, and A. Roth. Flexible, low-power register management using reference counting. *To appear in High Performance Computer Architecture, 2012. HPCA 2012. IEEE 18th International Symposium*, 2012.

[4] A. Bhattacharjee and M. Martonosi. Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors. In *International Symposium on Computer Architecture (ISCA)*, June 2009.

[5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[6] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, pages 318–329. IEEE Computer Society, 2008.

[7] J. Cantlin and M. Hill. Cache Performance for SPEC CPU2000 Benchmarks. `http://research.cs.wisc.edu/multifacet/misc/spec2000cache-data/`.

[8] G. Challen and M. Hempstead. The case for power-agile computing. In *Proc. of the 13th Workshop on Hot Topics in Operating Systems (HotOS-XIII)*, May 2011.

[9] G. Challen, J. Waterman, and M. Welsh. IDEA: Integrated Distributed Energy Management for Wireless Sensor Networks. In *Proc. of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010)*, June 2010.

[10] M. Chen, X. Wang, and X. Li. Coordinating processor and main memory for efficientserver power control. In *Proceedings of the international conference on Supercomputing*, pages 130–140. ACM, 2011.

[11] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 31–40. ACM, 2011.

[12] A. Davis. Capturing young womens' imagination. *Women in Engineering Magazine, IEEE*, 3(2):26 –41, dec. 2009.

[13] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *The 45th Annual IEEE/ACM International Symposium on Microarchitecture, 2012*, 2012.

[14] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini. Multiscale: memory system dvfs with multiple memory controllers. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 297–302. ACM, 2012.

[15] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: active low-power modes for main memory. *ACM SIGPLAN Notices*, 46(3):225–238, 2011.

[16] B. Diniz, D. Guedes, W. Meira Jr, and R. Bianchini. Limiting the power consumption of main memory. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 290–301. ACM, 2007.

[17] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)*, June 2011.

[18] K. J. N. et al. A 32-bit powerpc system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling. *IEEE Journal of Solid-State Circuits*, 37(11):1441–1447, Nov 2002.

[19] X. Fan, C. S. Ellis, and A. R. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In *Power-Aware Computer Systems*, pages 164–179. Springer, 2005.

[20] W. Felter, K. Rajamani, T. Keller, and C. Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 293–302. ACM, 2005.

[21] K. Flautner and T. Mudge. Vertigo: automatic performance-setting for linux. In *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, pages 105–116, New York, NY, USA, 2002. ACM.

[22] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. *SIGOPS Oper. Syst. Rev.*, 33(5):48–63, 1999.

[23] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[24] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking energy in networked embedded systems. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2008.

[25] A. Grover. Modern system power management. *Queue*, 1(7):66–72, Oct. 2003.

[26] M. Hempstead, D. Brooks, and G. Wei. An accelerator-based wireless sensor network processor in 130 nm cmos. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 1(2):193 –202, june 2011.

[27] M. Hempstead, N. Tripathi, P. Mauro, G.-Y. Wei, and D. Brooks. An ultra low power system architecture for sensor network applications. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, ISCA '05, pages 208–219, Washington, DC, USA, 2005. IEEE Computer Society.

[28] M. Hempstead, G.-Y. Wei, and D. Brooks. An Accelerator-based Wireless Sensor Network Processor in 130nm CMOS (Invited paper). In *ACM/IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Grenoble, France, October 2009.

[29] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, Nov. 2000.

[30] International Data Corporation. Worldwide Mobile Phone Growth Expected to Drop to 1.4Despite Continued Growth Of Smartphones. `http://goo.gl/ROLYO`.

[31] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. Phases: Duration Predictions and Applications to DVFS. *IEEE Micro*, 2005.

[32] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *International Symposium on Microarchitecture (MICRO)*, December 2006.

[33] T. K, M. M. Dsouza, and G. Varaprasad. Low power techniques for an android based phone. *SIGARCH Comput. Archit. News*, 39(2):26–35, Aug. 2011.

[34] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers, 2008.

[35] W. Kim, D. Brooks, and G.-Y. Wei. A fully-integrated 3-level dc-dc converter for nanosecond-scale dvfs. *Solid-State Circuits, IEEE Journal of*, 47(1):206–219, 2012.

[36] A. Lachenmann, P. J. Marron, D. Minder, and K. Rothermer. Meeting lifetime goals with energy levels. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[37] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power aware page allocation. *ACM SIGPLAN Notices*, 35(11):105–116, 2000.

[38] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[39] X. Li, R. Gupta, S. V. Adve, and Y. Zhou. Cross-component energy management: Joint adaptation of processor and memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(3):14, 2007.

[40] K. Lorincz, B. rong Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource aware programming in the pixie os. In *Proc. of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, pages 211–224, New York, NY, USA, 2008. ACM.

[41] K. Lorincz, B. rong Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource Aware Programming in the Pixie OS. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, November 2008.

[42] M. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks. The accelerator store framework for high-performance, low-power accelerator-based systems. *Computer Architecture Letters*, 9(2):53 –56, feb. 2010.

[43] M. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks. The accelerator store: A shared memory framework for accelerator-based systems. *To appear in ACM Transactions on Architecture and Code Optimization (TACO)*, 2012.

[44] G. Mainland, D. C. Parkes, and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In *Symposium on Networked Systems (NSDI'05)*, May 2005.

[45] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz. Towards energy-proportional datacenter memory with mobile dram. In *Proceedings of the 39th International Symposium on Computer Architecture*, pages 37–48. IEEE Press, 2012.

[46] M. Martins and R. Fonseca. Application modes: a narrow interface for end-user power management in mobile devices. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[47] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. *SIGPLAN Not.*, 44:205–216, March 2009.

[48] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[49] M. Milenkovic, E. Castro-Leon, and J. R. Blakley. Power-aware management in cloud data centers. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 668–673, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen. Phonelab: A large programmable smartphone testbed. In *Proc. of the 1st International Workshop on Sensing and Big Data Mining (SenseMine 2013)*, November 2013.

[51] S. Nilakantan, S. Annangi, N. Gulati, K. Sangaiah, and M. Hempstead. Evaluation of an Accelerator Architecture for Speckle Reducing Anisotropic Diffusion. In *ACM/IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, Taipei, Taiwan, October 2011.

[52] A. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *SenSys '13: Proceedings of the 11th international conference on Embedded networked sensor systems*, New York, NY, USA, 2013. ACM.

[53] R. Punzalan. Smartphone Battery Life a Critical Factor for Customer Satisfaction . `http://www.brighthand.com/default.asp?newsID=18721`.

[54] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. Wenisch, and M. Martin. Computational sprinting. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1–12, 2012.

[55] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 48–59. ACM, 2008.

[56] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread Motion: Fine-Grained Power Management for Multi-Core Systems. In *International Symposium on Computer Architecture (ISCA)*, June 2009.

[57] S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Apprehending joule thieves with cinder. In *MobiHeld '09: Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 49–54, New York, NY, USA, 2009. ACM.

[58] V. Shnayder, M. Hempstead, B. rong Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.

[59] J. Sorber, A. Kostadinov, M. Brennan, M. Garber, M. Corner, and E. D. Berger. Eon: A Language and Runtime System for Perpetual Systems. In *ACM Conference on Embedded Networked Sensor Systems (SenSys'07)*, November 2007.

[60] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis. Micro-pages: increasing dram efficiency with locality-aware data placement. In *ACM Sigplan Notices*, volume 45, pages 219–230. ACM, 2010.

[61] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White. Low-power amdahl-balanced blades for data intensive computing. *SIGOPS Oper. Syst. Rev.*, 44:71–75, March 2010.

[62] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, pages 2–2, Berkeley, CA, USA, 2008. USENIX Association.

[63] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 213–224, Washington, DC, USA, 2012. IEEE Computer Society.

[64] A. Verma, R. Koller, L. Useche, and R. Rangaswami. Srcmap: energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 20–20, Berkeley, CA, USA, 2010. USENIX Association.

[65] J. Waterman, G. W. Challen, and M. Welsh. Peloton: Coordinated resource management for sensor networks. In *Proc. of the 12th Workshop on Hot Topics in Operating Systems (HotOS-XII)*, May 2009.

[66] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: Optimizing high-resolution signal collection in wireless sensor networks. In *Proc. of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys'08)*, Raleigh, NC, USA, November 2008.

[67] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. A dynamic compilation framework for controlling microprocessor energy and performance. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 271–282, Washington, DC, USA, 2005. IEEE Computer Society.

[68] F. Xu, Y. Liu, T. Moscibroda, R. Chandra, L. Jin, Y. Zhang, and Q. Li. Optimizing background email sync on smartphones. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, MobiSys '13, pages 55–68, New York, NY, USA, 2013. ACM.

[69] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Currentcy: a unifying abstraction for expressing energy management policies. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 4–4, Berkeley, CA, USA, 2003. USENIX Association.

[70] H. Zheng, J. Lin, Z. Zhang, and Z. Zhu. Decoupled dimm: building high-bandwidth memory system using low-speed dram devices. *ACM SIGARCH Computer Architecture News*, 37(3):255–266, 2009.