The Case for Power-Agile Computing

Geoffrey Challen
*MIT, SUNY Buffalo**

Mark Hempstead
Drexel University

1 Introduction

Battery-powered devices are trapped by trends. More powerful performance requires more power, and while battery technologies slowly improve [17] users want more capable devices with longer battery lifetimes [19]. A way to escape this trap leverages power-proportional hardware architectures [5] that scale power consumption to perform when needed and draw little power when idle. Because most components are tuned to operate efficiently within a narrow power-performance range, we expect future power-proportional architectures to be *heterogeneous*, featuring multiple different processors, memory chips, storage devices and radios, each with different power-performance tradeoffs. Heterogeneity produces devices with fluid characteristics: phones that sprint like desktops and sleep like sensor nodes.

Today's devices already incorporate multiple processors, storage devices and radios with different power-performance characteristics. Researchers have proposed operating system designs that acknowledge this heterogeneity [6], performance- or power-driven component combinations [11, 4], approaches harnessing the efficiency of a particular set of components for certain tasks [1, 16], and systems organized into multiple power-performance tiers [15]. Inspired by these efforts, we coin the term *power agility* to describe a system's ability to efficiently operate a heterogeneous power-proportional device, balancing performance and power consumption.

Given increasingly heterogeneous devices, power agility requires not merely adjusting individual components but activating and deactivating them to react to changing demand. The idle phone in my pocket consumes less power than the one using GPS to route me to my destination, and while the mapping application wants the high-power radio, the game prefers a faster processor. So while power-proportional *hardware* allows the device to sprint and sleep, power-agile *software* guides it correctly between states. Recent microarchitectural advances attempt to mask hardware heterogeneity from the operating system [14], but we consider these a mistake. Only the operating system has the system-wide visibility and application information to achieve power agility.

This paper outlines the principles of power-agile computing. To begin, we design a heterogeneous power-proportional device to illustrate the size and diversity of the state space inherent to these architectures. Next, we present a scenario demonstrating our device responding to changes in demand. Using this scenario, we develop a set of challenges inherent to power-agile operation and discuss approaches to overcoming them.

2 Example Architecture

To begin, we assemble a device combining two general-purpose processors[1] (P1 and P2), two memory chips (M1 and M2), two storage devices (S1 and S2) and two radios (R1 and R2). Table 1 describes each component.

The relationship between power and performance varies for each component. Processors may transition smoothly over a restricted power envelope using dynamic voltage and frequency scaling (DVFS), but cannot scale to zero without losing state. Memory (DRAM) has a constant refresh cost that scales roughly with capacity plus additional power draw corresponding to the rate of reads and writes. Storage devices differ based on whether or not they include spinning components. Flash drives do not and scale approximately with usage but are limited in size. Radios exhibit wide power-performance variation because their usage depends both on the hardware and the protocol. 802.11 clients can enter power-saving mode (PSM) which uses base station buffering to save power. Bluetooth has limited range but lower power consumption balanced between both sides of the link.

*Starting 7/1/2011.

[1] Distinguished from task-specific processors like GPUs or DSPs.

ID	Name	mW	Performance
P1	ARM Cortex-M4[1] [3]	0.9[2] 15.6	75 MHz 300 MHz
P2	ARM Cortex-A9 [2]	23.5[2] 400.	415 MHz 830 MHz
M1	**32 MB** ISSI SDRAM [9]	81. 108.	Refresh only 166 MHz
M2	**1 GB** Micron "Slow" DDR2	322.[4] 482.	Refresh only 266 MHz
S1	**2 GB** MicroSD Card	20.[5] 100.[5]	Idle 25 MBps[6]
S2	**64 GB** OCZ SSD	200.[7] 1000.[7]	Idle 5.5 MBps[7]
R1	**250 kbps** TI CC2540 BLE	6.7[3] 66.3[9]	10% duty cycle[8] Receive mode[9]
R2	**11 Mbps** Marvell 802.11bg	30.9[3] 309.3[10]	10% duty cycle[8] Idle mode[10]

[1] Capable of running a subset of the full P2 instruction set.
[2] Optimistic estimate based on an optimistic estimate of DVFS providing 1:5 performance and 1:17 power scaling [7].
[3] Estimated based on scaled full-power performance.
[4] Estimated based on Micron leakage numbers.
[5] Estimated due to lack of publicly-available datasheets.
[6] Maximum achievable.
[7] Measured by Tom's Hardware [18].
[8] Duty cycling shifts power usage from the receiver to the sender, which has to remain online (as in 802.11 PSM) or send longer packets (as in 802.15.4 Low-Power Listening [12]).
[9] Receive-only in high-sensitivity mode. Transmit is similar.
[10] Transmit and receive vary so usage is workload-dependent.

Table 1: **Performance and power consumption of selected hardware components.** We assume voltage gating can reduce the usage of disabled components to near zero [13]. The 10 notes reflect the challenge in obtaining these numbers, as most data sheets omit this information.

We define a *component ensemble* as the components currently active, constraining the set of valid ensembles to include only those that can support the device operating system. For our example, these include (a) one or both processors, (b) one or both memory chips[2], (c) neither, one or both storage devices and (d) neither, one or both radios. By switching between components our device can operate across a wide power range. It its lowest-power ensemble, the device has a 75 MHz CPU, 32 MB of RAM, and draws 82[3] mW and is roughly-equivalent to a embedded sensor node. In its highest-power ensemble the device has multiple cores, over 1 GB of RAM, over 320 GB of storage, Wi-Fi and Bluetooth. Consuming almost 2.5 W, it is similar to emerging smartphones.

[2]While many low-power processors come with small amounts of integrated memory, we have conservatively chosen to require 32 MB of RAM in order to run embedded versions of Linux. It is conceivable that our candidate device could enter an active sleep state with a microkernel capable of fitting in the processor's onboard RAM.

[3]Actual power consumption would be higher due to system buses, memory controllers, and other components of a complete architecture.

This device can activate *144 valid component ensembles*[4]. Figure 1 shows the composition and power envelope of each, and motivates two observations. First, there are many valid ensembles and wide usage variation even in an architecture with only two components per class. Incorporating more components would produce even more options. Second, at any power level there are many diverse ensembles the device can use: a fast processor, small memory chip, and slow disk; a slow processor, large memory chip, and fast radio; etc. These differ not in their total power consumption but in how they perform and distribute power across components, and while some ensembles may seem too weird to be useful they may suit certain applications. Finally, while it may seem best to avoid inefficient ensembles—those achieving low utilization and a low active- to idle-power ratio—given the speed of temporal changes in demand and the overhead of ensemble transitions we expect devices to spend some time at the low end of ensemble power envelopes.

3 Challenges

To illustrate how a power-agile device might operate we imagine a phone performing a background task that is interrupted by an interactive session. Figure 2 shows how overall and per-component power allocations change to respond to the needs of the two applications. We refer to this scenario throughout the rest of this section as we examine the challenges inherent to power-agile computing. These are related to five roles that the operating system plays while operating power-agile hardware: measuring (3.1) and predicting (3.2) performance; and selecting (3.3), preparing (3.4) and executing (3.5) ensemble transitions. Throughout we demonstrate how traditional scheduling and resource-allocation problems are complicated by the flexible nature of the underlying hardware.

3.1 Measuring Efficiency

Determining performance differences between ensembles requires application metrics weighting both power and performance such as the energy-delay product (EDP), commonly used in circuit design [10]. The EDP is defined as EDP $= E\Delta$ where E measures the energy consumed during some time quantum and Δ measures a application-specific performance characteristic such as the time necessary to process a block of data or respond to user input. The system tries to minimize the EDP for each application. Controlling the strength of the performance component using an exponential—EDP $= E\Delta^n$—allows applications to weight their preference for performance or efficiency. In our scenario,

[4]3 processor choices $\times$ 3 memory choices $\times$ 4 storage choices $\times$ 4 radio choices.

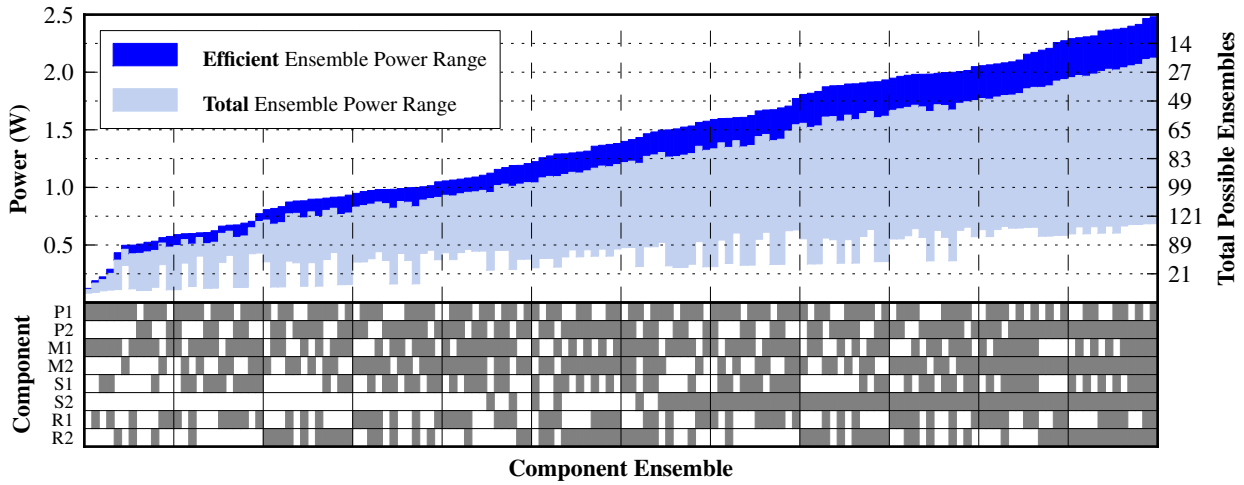


Figure 1: **Power envelopes of all 144 example device component ensembles.** Ensembles are sorted by increasing maximum power draw. For each ensemble, the bottom shows which components are active and the top displays the power envelope. The top 20% of the envelope—the most efficient operating range—is drawn in dark blue. The right axis counts the total number of ensembles that might draw that much power: e.g., there are 121 ensembles that could consume 0.75 W, depending on the workload.

the interactive application uses $E\Delta^2$ causing the system to activate high-performance ensembles; the background task uses $E\sqrt{\Delta}$, causing the system to remain in lower-power states. Ensuring that applications choose appropriate exponents and balancing between applications at run-time are challenges inherent to this approach.

3.2 Predicting Ensemble Performance

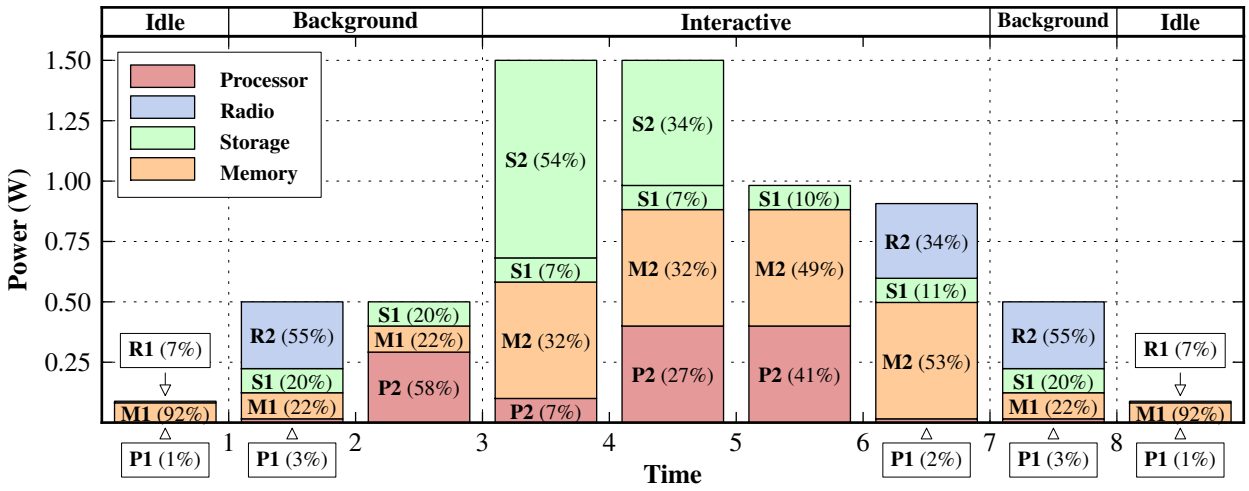
Given the size of the ensemble state space, predicting ensemble performance is a key part of transitions. Assuming an application with preferred EDP $E\Delta^n$, both E and Δ will vary across ensembles: E with the cost and utilization of system components, and Δ with performance. The direct way to determine power-performance is to run the application on many ensembles, but given the number of states and transition cost this is infeasible online. However, offline experimentation could produce binary annotations. Another approach is to have executables include hints about performance characteristics important to various stages. Before transmitting a large amount of information, a hint would alert the system to the need for a high-bandwidth radio. Hints have the advantage being portable across devices, but they require programmer support and the system must ensure that applications do not abuse them to gain unfair access to resources.

When running unannotated binaries or mixtures of applications with performance dependencies, the system may need to estimate the impact of ensemble changes before performing them. In some cases, the currently running ensemble can be *artificially* constrained to estimate how performance might change after a component change. For example, when moving from **M2** to **M1** at t = 7 in the scenario, the system might be concerned about the impact of this transition on the usage of **S1**. If disabling the large memory chip causes **S1** usage to increase dramatically, the system will fail to achieve the intended power reduction. To uncover a link between memory size and disk usage, the operating system can artificially limit the amount of memory in use by trimming pages from **M2**. It may do this in a smooth fashion until it is using only roughly the same amount of the larger chip as the smaller chip size, and then, assuming no serious component relationships have been uncovered, initiate the transition. This strategy is more applicable to transitions that attempt to trim power by disabling components, but this is also when it is most useful, as it allows the operating system to discover relationships between component usage that might negate power reductions.

3.3 Selecting Component Ensembles

Scheduling ensemble transitions relies on the capabilities already presented—metrics for evaluating performance and predicting performance across ensembles. When running a single application the system can respond directly to its estimated performance, weighting efficiency improvements against ensemble transition costs.

Running multiple applications creates new challenges. First, there is the question of how to assign performance metrics to applications. In our scenario the background task would complete faster if it were allowed to use the higher exponent used by the interactive application. The goal is to assign the most efficient metric to the application that produces acceptable performance, and doing so is likely to require user feedback.



0 When idle **P1** and **M1** are idled and **R1** operates at low duty cycle.

1 Receiving data over **R1** the phone initiates a background task. The device activates **R2** to rapidly receive data and **S1** to store it.

2 As the phone begins processing the task it activates **P2** and disables **R2**.

3 The user removes the phone from their pocket and begins interacting with an application, which activates **M2** and retrieves data from **S2**.

4 As the interactive application continues energy usage shifts from **S2** to **P2**.

5 When the interactive application is finished with **S2** it is disabled.

6 As the interactive session completes, the phone offloads data using **R2** driven by **P1**.

7 Background processing resumes in the same ensemble it was using previously.

8 The background task completes, idling the phone.

Figure 2: **Scenario.** The figure and table describe the scenario referred to throughout Section 3. Bars indicate the total energy consumed, broken down and labeled by component. The table describes what is happening at each time step.

Choosing the correct ensemble for both applications is the next challenge. If their performance requirements are aligned, then an ensemble may exist that works well for both. Applications differing in their performance requirements complicate the process. If the system has sufficient energy it may choose to operate a combination of both ideal ensembles, but this produces inefficiency as the set of distinct resources needed by one application is idled while the other runs.

The simplest approach is to transition between the ideal ensembles while increasing both application's time quanta sufficient to amortize the transition cost. In many cases, however, we expect that this will lead to unacceptable interactive performance. A second possible approach is to pick an ensemble that produces acceptable—but not ideal—performance for both applications, potentially weighted towards the application with higher priority. Another option is to select an ensemble optimized for one application while allocating resources within that ensemble in favor of the other. For example, given one application that requires a high-speed disk and another than needs a large memory chip, we can choose to use the large memory chip and a slower disk allocating a large portion of the memory to a buffer cache to improve performance for the I/O-bound application.

3.4 Preparing Ensemble Transitions

Because ensemble transitions are both important and costly, the operating system should prepare the system to minimize their overhead. Preparation is particularly important in the memory and storage hierarchy, where the location of data has a significant impact on component transitions. Preparation also requires the system forecast future application demand and ensemble dwell times.

Consider an example transition that activates a larger memory chip with superior performance. If the system will be in that ensemble for a significant length of time, all applications will benefit from having data relocated from the smaller to the larger chip. This also allows the smaller chip to be shut off to save power. However, if and when the device wants to disable the larger memory chip in order to shift power toward some other necessary component, the amount of data stored in the larger memory bank creates a high overhead for this transition.

If the system predicts brief use of the larger memory bank, it may try several strategies to reduce the eventual transition overhead. First, if the transition is due to a particular application, it may continue to operate the smaller chip for other applications while allocating new pages on the larger component. Once the memory-hungry ap-

plication is finished with these pages, they can be discarded and the memory disabled without migrating data. Another approach is to copy accessed pages on demand but mirror writes to both memory banks to minimize the eventual transition cost. Assuming that the smaller chip is never shut off—possible if consumes little power—the physical address space may be configured to always mirror a portion to both chips when the larger bank is active. The operating system may try to allocate memory from the mirrored portion of the address space for pages that have long expected lifetimes, are used by applications that prefer more power-efficient states, or based on explicit application requests. These pages will benefit from better performance when the larger bank is active while never requiring migration.

3.5 Executing Ensemble Transitions

Ensemble transitions tailor the device to application demands but may require complex or expensive component transitions. The Advanced Configuration and Power Interface (ACPI) specification [8] standardizes per-component and overall power states but does not consider component transitions. Below we outline for each component class, the complexity and cost of transitions and a brief description of how to perform one:

- **Processor:** Difficulty: *high*, Cost: *medium*. Transitioning between processors, even ones with highly-compatible instruction sets, requires migrating process state, correcting for processor differences, and potentially reloading new process executables enabling or disabling certain instructions.
- **Memory:** Difficulty: *medium*, Cost: *high*. Moving to a smaller chip requires migrating some pages to the new memory area while flushing others to the backing store, along with kernel adjustments to its own memory footprint. Transitioning to a larger chip requires migrating data.
- **Storage:** Difficulty: *low*, Cost: *low*. Disabling requires writing out dirty buffers. Enabling will cause a performance dip while caches fill.
- **Radio:** Difficulty: *medium*, Cost: *medium*. Disabling requires flushing any outstanding buffers, closing connections and potentially coordinating with the receiver to move together to a new radio technology. Enabling may require association—potentially costly, depending on the protocol—and a delay while link parameters necessary for efficient operation can be determined.

4 Summary

Power-proportional heterogeneous devices require system support to continuously balance performance and power efficiency, an ability we call *power agility*. On today's ubiquitous battery-powered devices power agility is critical in order to continue improving performance while delivering acceptable battery lifetime.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant # CCF-1017654.

References

[1] ANDERSEN, D. G., FRANKLIN, J., KAMINSKY, M., PHANISHAYEE, A., TAN, L., AND VASUDEVAN, V. Fawn: A fast array of wimpy nodes. In SOSP'09.

[2] ARM. Cortex-A9 Processor. http://bit.ly/bHOomu.

[3] ARM. Cortex-M4 Processor. http://bit.ly/hs7dQ7.

[4] BALASUBRAMANIAN, A., MAHAJAN, R., AND VENKATARAMANI, A. Augmenting mobile 3G using WiFi. In MobiSys'10.

[5] BARROSO, L. A., AND HÖLZLE, U. The case for energy-proportional computing. Computer 40 (December 2007), 33–37.

[6] BAUMANN, A., BARHAM, P., DAGAND, P.-E., HARRIS, T., ISAACS, R., PETER, S., ROSCOE, T., SCHÜPBACH, A., AND SINGHANIA, A. The multikernel: a new os architecture for scalable multicore systems. In SOSP'09.

[7] ET AL., K. J. N. 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 (Nov 2002), 1441–1447.

[8] HEWLETT-PACKARD, INTEL, MICROSOFT, PHOENIX TECHNOLOGIES, AND TOSHIBA. Advanced configuration and power interface. http://www.acpi.info/.

[9] ISSI. Is42vm32100c advanced information. http://bit.ly/eDqr3O.

[10] MARTIN, A. J., NYSTRÖM, M., AND PÉNZES, P. I. ET2: A metric for time and energy efficiency of computation. Tech. rep., 2001.

[11] MOGUL, J. C., ARGOLLO, E., SHAH, M., AND FARABOSCHI, P. Operating system support for NVM+DRAM hybrid main memory. In HotOS XII.

[12] MOSS, D., HUI, J., LEVIS, P., AND CHOI, J. I. Tinyos extension proposal TEP-126. http://bit.ly/gCEEZ5.

[13] POWELL, M., YANG, S.-H., FALSAFI, B., ROY, K., AND VIJAYKUMAR, T. Gated-vdd: A circuit technique to reduce leakage in deep-submicron cache memories. In ISLPED 2000.

[14] RANGAN, K., POWELL, M., WEI, G.-Y., , AND BROOKS, D. Achieving uniform performance and maximizing throughput in the presence of heterogeneity. In HPCA-17.

[15] SORBER, J., BANERJEE, N., CORNER, M. D., AND ROLLINS, S. Turducken: Hierarchical power management for mobile devices. In MobiSys'05.

[16] SZALAY, A. S., BELL, G. C., HUANG, H. H., TERZIS, A., AND WHITE, A. Low-power amdahl-balanced blades for data intensive computing. In HotPower'09.

[17] THE ECONOMIST. In search of the perfect battery. http://econ.st/h9jzzk.

[18] TOM'S HARDWARE. Flash ssd update: More results, answers. http://bit.ly/ylLgD.

[19] WALKO, J. What do cell phone users want? Better batteries! http://bit.ly/gefaF1.