# ViPZonE: Hardware Power Variability-Aware Virtual Memory Management for Energy Savings

Mark Gottscho, *Student Member, IEEE*, Luis A.D. Bathen, *Member, IEEE*, Nikil Dutt, *Fellow, IEEE*, Alex Nicolau, *Member, IEEE*, and Puneet Gupta, *Member, IEEE*

**Abstract**—Hardware variability is predicted to increase dramatically over the coming years as a consequence of continued technology scaling. In this paper, we apply the Underdesigned and Opportunistic Computing (UnO) paradigm by exposing system-level power variability to software to improve energy efficiency. We present ViPZonE, a memory management solution in conjunction with application annotations that opportunistically performs memory allocations to reduce DRAM energy. ViPZonE's components consist of a physical address space with DIMM-aware zones, a modified page allocation routine, and a new virtual memory system call for dynamic allocations from userspace. We implemented ViPZonE in the Linux kernel with GLIBC API support, running on a real x86-64 testbed with significant access power variation in its DDR3 DIMMs. We demonstrate that on our testbed, ViPZonE can save up to 27.80 percent memory energy, with no more than 4.80 percent performance degradation across a set of PARSEC benchmarks tested with respect to the baseline Linux software. Furthermore, through a hypothetical "what-if" extension, we predict that in future non-volatile memory systems which consume almost no idle power, ViPZonE could yield even greater benefits, demonstrating the ability to exploit memory hardware variability through opportunistic software.

**Index Terms**—DRAM, variability, energy-aware systems, main memory, allocation/deallocation strategies, operating systems

## 1 Introduction

Inter-die and intra-die process variations have become significant as a result of continued technology scaling into the deep submicron region [1], [2]. The International Technology Roadmap for Semiconductors (ITRS) predicts that over the next decade, both performance and power consumption variation will increase by up to 66, and 100 percent, respectively [3]. Variations can stem from semiconductor manufacturing processes, ambient conditions, device aging, and in the case of multi-sourced systems, vendors [4].

System design typically assumes a rigid hardware/software interface contract, hiding physical variations from higher layers of abstraction [5]. This is often accomplished through guard-banding, a method that ensures reliable and consistent components over all operation and fabrication corners. However, there are many associated costs from over-design, such as chip area and complexity, power consumption, and performance. Despite considerable hardware variability, the rigid hardware/software contract results in software assuming strict adherence to the hardware specifications. The overheads of guard-banding are reduced, but not eliminated, through the practice of binning, where manufacturers market parts with considerable post-manufacturing variability as different products. For example, manufacturers have resorted to binning processors by operating frequencies to reduce the impact of inter-die variation [5]. However, even with guardbanding, binning, and dynamic voltage and frequency scaling (DVFS), variability is inherently present in any set of manufactured chips. Furthermore, with the emergence of multi-core technology, intra-die variation has also become an issue. To minimize the overheads of guardbanding, recent efforts have shown that exploiting the inherent variation in devices [6], [7] yields significant improvements in overall system performance.

This has led to the notion of the Underdesigned and opportunistic (UnO) computing paradigm [5], depicted in Fig. 1. In UnO systems, design guardbands are reduced while some hardware variations are exposed to a flexible software stack. This allows the system to tune itself to suit the unique characteristics of its hardware that arise from process variations (part variability), aging effects, and environmental factors such as voltage and temperature fluctuations (time variability).

### 1.1 Related Work

There is an abundance of literature highlighting the extent of semiconductor process variability and methods for coping with it; we cite some of the more relevant work here. Recently, there have been several works which noted significant power variability in off-the-shelf parts. Hanson et al. [8] found up to 10 percent power variation across identical Intel Pentium M processors, while Wanner et al. [9] measured over 5× sleep power variation across various Cortex M3 processors. [8] also observed up to 2X active power variation across various DRAMs, while Gottscho et al. [10] found up to 20 percent

- *M. Gottscho and P. Gupta are with the NanoCAD Lab, Department of Electrical Engineering, University of California, Los Angeles, CA 90095. E-mail: mgottscho@ucla.edu, puneet@ee.ucla.edu.*
- *L.A.D. Bathen, N. Dutt, and A. Nicolau are with the Department of Computer Science, University of California, Irvine, Irvine, CA 92617. E-mail: lbathen@uci.edu, {dutt, nicolau}@ics.uci.edu.*

*Manuscript received 9 May 2013; revised 18 Feb. 2014; accepted 18 May 2014. Date of publication 8 June 2014; date of current version 8 Apr. 2015. Recommended for acceptance by S.W. Chung.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TC.2014.2329675*

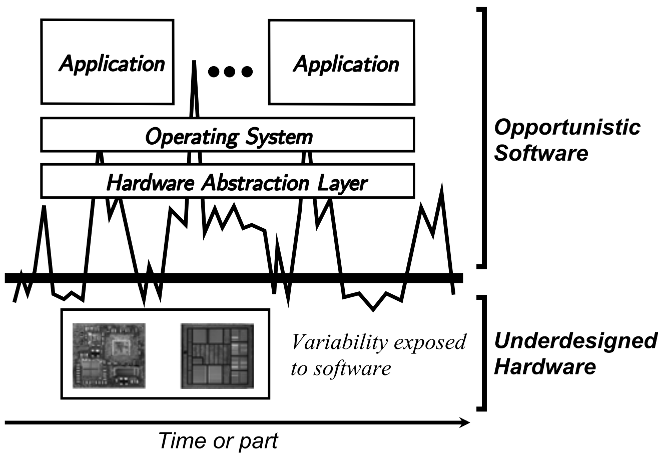


Fig. 1. Underdesigned and opportunistic computing concept adapted from [5], where hardware variability across different parts or over time is deliberately exposed to software layers to improve energy efficiency, performance, cost, etc.

power variation in a set of nineteen commodity 1 GB DDR3 DRAM modules (DIMMs).

Most efforts dealing with variation have focused on mitigating and exploiting it in processors [6], [7], [11], [12] or in on-chip memory [13], [14], [15], [16], [17]. Fewer papers have looked at variability in off-chip, DRAM-based memory subsystems. As off-chip DRAM memory may consume as much power as the processor in a server-class system [18], and is likely to increase for future many-core platforms (e.g., Tilera's TILEPro64 and Intel's single chip cloud computer (SCC)), memory power variations could have a significant impact on the overall system power.

Most works on main memory power management have focused on minimizing accesses to main memory through smart caching schemes and compiler/OS optimizations [19], [20], [21], [22], [23], yet none of these methods have taken memory variability into account. Bathen et al. [24] recently proposed the introduction of a hardware engine to virtualize on-chip and off-chip memory space to exploit the variation in the memory subsystem. These designs require changes to existing memory hardware, incurring additional design cost. As a result, designers should consider software implementations of power and variation-aware schemes whenever possible. Moreover, a variability-aware software layer should be flexible enough to deal with the predicted increase in power variation for current and emerging memory technologies.

### 1.2 Our Contributions

This work extends ViPZonE, an OS-based, pure software memory (DRAM) power variability-aware memory management solution that was first introduced in [25], with a full software implementation that is evaluated on a real hardware testbed. ViPZonE adapts to the power variability inherent in a given set of commodity DRAM memory modules by harnessing disjunct regions of physical address space with different power consumption. Our approach exploits variability in DDR3 memory at the DIMM modular level, but our approach could be adapted to work at finer granularities of memory, if variability data and hardware support are available. Our experimental results across various configurations running PARSEC [26] workloads show an average of 27.80 percent memory energy savings at the cost of no more than a modest 4.80 percent increase in execution time over an unmodified Linux virtual memory allocator.

The key contributions of this work are as follows:

- A detailed description and complete implementation of the ViPZonE scheme originally proposed in [25], including modifications to the Linux kernel and standard C library (GLIBC). These changes allow programmers control of power variability-aware dynamic memory allocation.
- An analysis of DDR3 DRAM channel and rank interleaving advantages and disadvantages using our instrumented x86-64 testbed, and the implications for variability-aware memory systems.
- An evaluation of power, performance, and energy of the ViPZonE implementation using a set of PARSEC benchmarks on our testbed.
- A hypothetical evaluation of the potential benefits of ViPZonE when applied to systems with negligible idle memory power (e.g., emerging non-volatile memory (NVM) technologies).

*ViPZonE is the first OS-level, pure-software, and portable solution to allow programmers to exploit main memory power variation through memory zone partitioning*. The source code is available at [27].

### 1.3 Paper Organization

This paper is organized as follows. We start with background material discussing the DDR3 DRAM memory system architecture, memory interleaving, and the relevant basics of Linux memory management in Section 2. This is followed by a detailed description of the ViPZonE software implementation in Section 3, including the target platform and assumptions, kernel back-end, and the GLIBC front-end. In Section 4, we describe the testbed hardware and configuration, include an analysis of the benefits and drawbacks of memory interleaving for our testbed, and compare the ViPZonE software stack with the vanilla[1] code with memory interleaving disabled. A brief "what-if" study on ViPZonE for emerging non-volatile memories (NVMs) is covered in Section 4.4. We conclude our work and discuss opportunities for future research in Section 5.

## 2 Background

In this section, we provide a brief background on typical memory system architecture, memory interleaving, and vanilla Linux kernel memory management to aid readers in understanding our contributions.

### 2.1 Memory System Architecture

To avoid confusion, we briefly define relevant terms in the memory system. In this work, we use DDR3 DRAM memory technology.

1. In this work, "vanilla" refers to the baseline unmodified software implementation.

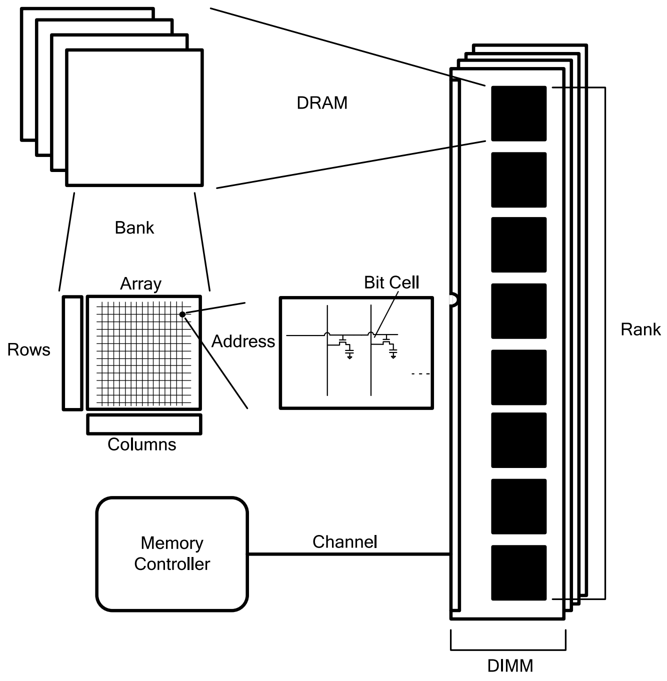


Fig. 2. Components in a typical DDR3 DRAM memory system.

In a typical server, desktop, or notebook system, the memory controller accesses DRAM-based main memory through one or more memory *channels*. Each channel may have one or more *DIMMs*, which is a user-serviceable memory module. Each DIMM may have one or two *ranks* which are typically on opposing sides of the module. Each rank is independently accessible by the memory controller, and is composed of several DRAM devices, typically eight for non-ECC modules. Inside each DRAM are multiple *banks*, where each bank has an independent memory *array* composed of *rows* and *columns*. A memory location is a single combination of DIMM, rank, bank, row, and column in the main memory system, where an access is issued in parallel to all DRAMs, in lockstep, in the selected rank. This organization is depicted in Fig. 2.

For example, when reading a DIMM in the DDR3 standard, a burst of data is sent over a 64-bit wide bus for four cycles (with transfers occurring on both edges of the clock). During each half-cycle, 1 byte is obtained from each of eight DRAMs operating in lockstep, thus composing 64 bits that can be sent over the channel. Note that this is not "interleaving" in the sense that we use throughout the paper, meaning that it is not configurable or software-influenced in any way; it is hard-wired according to the DDR3 standard.

## 2.2 Main Memory Interleaving

Since our scheme as implemented on our testbed requires channel and rank interleaving to be disabled, we include some background material on the benefits and drawbacks of interleaving here.

It is common practice for system designers to employ interleaved access to parallel memories to improve memory throughput, particularly for parallel architectures, e.g. vector or SIMD machines [28]. This is done by mapping adjacent chunks (the size of which is referred to as the *stride*) of addresses to different physical memory devices. Thus, when a program accesses several memory locations with a high degree of spatial (in the linear address space) and temporal locality, the operations are overlapped via parallel access, yielding a speedup.

Many works have explored interleaving performance, generally in the context of vector and array computers, but also with MIMD machines as the number of processors and memory modules scale [29], [30]. While widely used today, interleaving does not necessarily yield improved performance. For example, the technique makes no improvement in access latency, and there is little performance gain when peak memory bandwidth requirements or memory utilization are low (e.g., high arithmetic intensity workloads as defined by the roofline model of computer performance [31]).

Furthermore, interleaving may yield negligible speedup when access patterns do not exhibit high spatial locality (e.g., random or irregular access), and is also capable of *worse* performance when several concurrent accesses have module conflicts as a result of the address stride, number of modules, and interleaving stride [28], [32]. Researchers have come up with techniques to mitigate or avoid this issue, usually through introducing irregular address mappings. For example, the Burroughs Scientific Processor used a 17-module parallel memory and argued that prime numbers of memory modules allowed several common access patterns to perform well [33]. Other approaches suggested skewed or permutation-based interleaving layouts [34], and clever data array organization for application-specific software routines [35].

In our testbed, interleaving prevents the exploitation of any power or performance variability present in the memory system. When striping accesses across different devices, the system runs all the memories at the speed of the slowest device, thus potentially sacrificing performance of faster modules, and preventing opportunistic use of varied power consumption. Interleaving on our testbed is also inflexible: it is statically enabled/disabled (cannot be changed during runtime), and it could also incur power penalties from the prevention of deeper sleep modes on a per-DIMM basis.

In this work, as interleaving and ViPZonE are mutually exclusive on our testbed, we provide an evaluation of power, performance, and energy for different interleaving modes in Section 4.2. We will discuss possible solutions to allow interleaving alongside variability-aware memory management in Section 5.

## 2.3 Vanilla Linux Kernel Memory Management

In order to understand our ViPZonE kernel modifications, we now discuss how the vanilla Linux kernel handles dynamic memory allocations from userspace, in a bottom-up manner. For interested readers, [36], [37] are excellent resources for understanding the Linux kernel, while [38] provides an exceptional amount of detail on the inner workings of the Linux memory management implementation.

### *2.3.1 Physical Memory Zones*

The physical page allocator is at the core of the Linux kernel memory management subsystem. When presented with an allocation request for one or more pages with certain

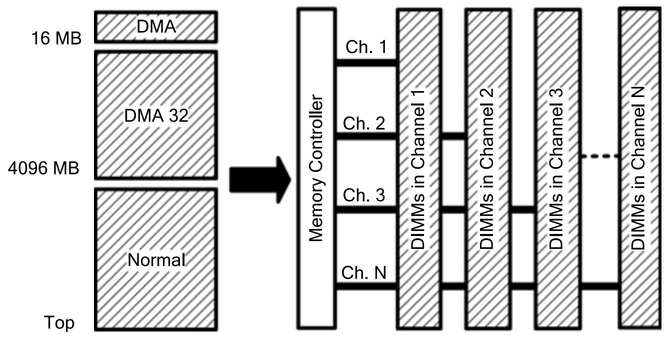


Fig. 3. Vanilla Linux physical address space zoning for x86-64. Zone boundaries do not necessarily fall between DIMM boundaries.

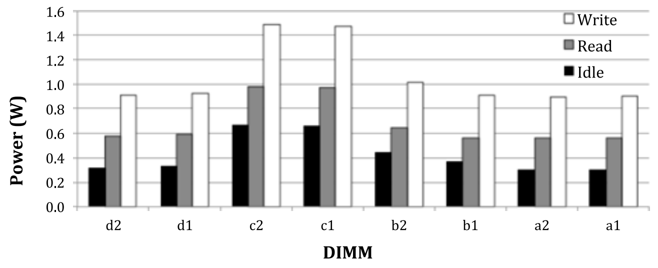


Fig. 4. Measured power variations in eight identically specified off-the-shelf DDR3 DIMMs, using methods from [10]. Letters denote different DIMM models, and numbers indicate instances of a model.

constraints, the system tries to find the most suitable allocation in the least amount of time. The kernel may pass through multiple stages during an allocation attempt, with greater performance penalties as it tries harder to find suitable memory.

The page allocator relies on several important constructs, including, but not limited to: page structures, memory zones, page freelists, and constraint bitmasks [36]. The kernel utilizes several zones to group regions of contiguous physical memory (see Fig. 3), required for legacy hardware support [36]. In direct memory access (DMA), devices talk directly with physical memory, bypassing the CPU. However, many legacy devices can only address the lowest 16 MB[2] and must be able to receive page allocations in this region. The kernel, if configured to support DMA, needs to reserve this space accordingly. There are also newer devices that are capable of addressing up to 4 GB of memory, and the kernel must be able to accommodate these DMA32 devices as well, albeit with more headroom.

The kernel does this by representing these spaces with physically contiguous and adjacent DMA and DMA32 memory zones, each of which tracks pages in its space independently of other zones [36], [39]. This allows for separate bookkeeping for each zone as well, such as low-memory watermarks, buddy system page groups, performance metrics, etc. Thus, if both are supported, the DMA zone occupies the first 16 MB of memory, while the DMA32 zone spans 16 to 4,096 MB. This means that for 64-bit systems with less than 4 GB of memory, all of memory will be in DMA or DMA32-capable zones.

The rest of the memory space not claimed by DMA or DMA32 is left to the "Normal" zone.[3] On x86-64, this will contain all memory above DMA and DMA32. Since the kernel cannot split allocations across multiple zones [36], each allocation must come from a single zone. Thus, each zone maintains its own page freelists, least-recently-used lists, and other metrics for its space.

### 2.3.2 *Physical Page Allocation*

The kernel tries to fulfill page allocation requests in the most suitable zone first, but it can fall back to other zones if required [36], [39]. For example, a user application will typically have its memory allocated in the normal zone. However, if memory there is low, it will try DMA32 next, and DMA only as a last resort. The kernel can also employ other techniques if required and permitted by the allocation constraints (if the request cannot allow I/O, filesystem use, or blocking, they may not apply) [36], [39]. However, the reverse is not true. If a device driver needs DMA space, it must come from the DMA zone or the allocation will fail. For this reason, the kernel does its best to reserve these restricted spaces for these situations [36].

### 2.3.3 *Handling Dynamic Virtual Memory Allocations*

In Linux systems, there are two primary system calls (syscalls) used for applications' dynamic memory allocations. For small needs, such as growing the stack, *sbrk()* is used, which extends the virtual stack space and grabs physical pages to back it as necessary [36]. *sbrk()* is also used by the GLIBC implementation of *malloc()*, as it is quite fast and minimizes fragmentation through the use of memory pooling [40]. For larger requests, *malloc()* usually resorts to the *mmap()* syscall, which is better suited for bigger, longer-lived memory needs, although is slower to allocate (*mmap()* also has other uses, such as shared memory, memory-mapped files, etc.). Both syscalls merely manage the virtual memory space for a process; they do not operate on the physical memory at all. The kernel generally only allocates physical backing pages lazily on use, rather than at allocation time.

## 2.4 Exploiting DRAM Power Variation

Before discussing the ViPZonE implementation, we now briefly discuss the nature of DRAM power variability. In this work, we optimized for variability measured at the DIMM level, as it is the smallest piece of memory that is user-replaceable in a typical desktop, server, or laptop. Fig. 4 depicts the measured power variation in a set of eight identically specified (voltage, clock frequency, timings, capacity, etc.) off-the-shelf DDR3 DIMMs. Each DIMM's write, read, and idle power was characterized using a memory testing routine (more details on the methodology can be found in [10]). These power deviations arise purely from vendor implementations and manufacturing process variation. Note that using DIMMs from different manufacturers in the same system may be common in a situation where there are many memories, and/or when DIMMs need to be replaced over time due to permanent faults (e.g., in

2. In this work, we adhere to conventional memory notation, as opposed to networking and storage notation for capacities. For example, we define 1 GB to be $2^{30}$ bytes of memory, not $10^9$ bytes.

3. The "HighMem" zone present in x86 32-bit systems is not used in the x86-64 Linux implementation.

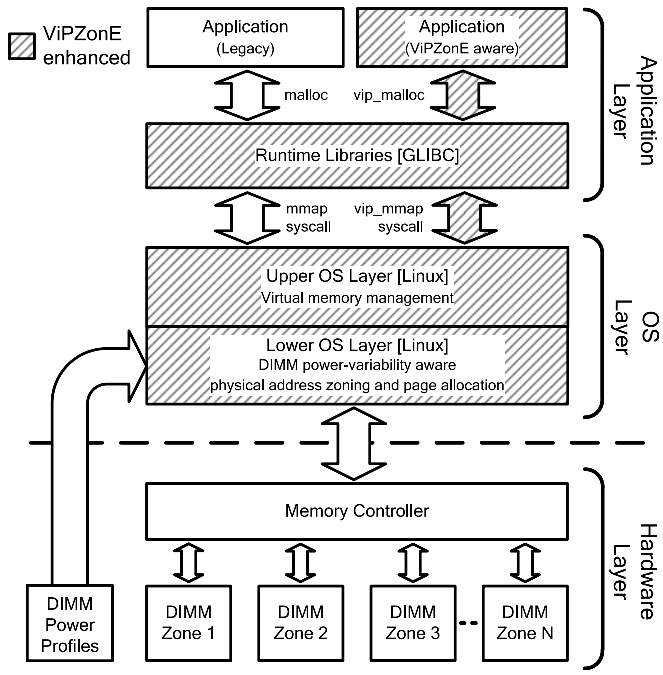


Fig. 5. Layered architecture of ViPZonE.

datacenters). Moreover, the variability among DRAMs is expected to increase in the future [3], especially if variation-aware voltage scaling techniques are used, such as those proposed by [41].

In a variability-aware memory management scheme, the upper-bound on power savings is determined by the extent of power variation across the memories in the system. For example, if we assume interleaving to be disabled, the worst case for power consumption would be when the DIMM with the highest power contains all the data that is accessed, while the rest are idle. The best case would be where all of this data is on the lowest-power DIMM. In a case where data is spread evenly across all DIMMs, no power variation can be exploited and the result is similar to that if interleaving were used.

In a multi-programmed system where only a portion of the physical memory is occupied, then we can intelligently save energy. If most of the memory is occupied and accessed frequently, it is harder to exploit the power variations, but as long as there is some non-uniformity in the way different pages are accessed, it remains possible. We believe that the former scenario is a good case for our study, as any system where the physical memory is fully occupied will suffer from large performance bottlenecks due to disk swapping. When this happens, memory power and performance are no longer a first-order concern. Thus, we believe the latter scenario to be less interesting from the perspective of a system designer when considering power variability-aware memory management.

## 3 VIPZONE IMPLEMENTATION

ViPZonE is composed of several different components in the software stack, depicted in Fig. 5, which work together to achieve power savings in the presence of DIMM variability. We refer to the lower OS layer as the "back-end" and the application layer along with the upper OS layer as the "front-end". These are described in Section 3.2 and Section 3.3, respectively. ViPZonE uses source code annotations at the application level, which work together with a modified GLIBC library to generate special memory allocation requests which indicate the expected use patterns (write/read dominance, and high/low utilization) to the OS.[4] Inside the back-end Linux memory management system, ViPZonE can make intelligent physical allocation decisions with this information to reduce DRAM power consumption. By choosing this approach, we are able to keep overheads in the OS to a minimum, as we place most of the burden of power-aware memory requests to the application programmer. With our approach, no special hardware support is required beyond software-visible power sensors or predetermined power data that is accessible to the kernel.

There are alternative approaches to implementing power variation-aware memory management. One method could avoid requiring a modified GLIBC library and application-level source annotations by having the kernel manage all power variation-aware decisions. However, such an approach would place the burden of smarter physical page allocations on the OS, likely resulting in a significant performance and memory overhead. Furthermore, the kernel would be required to continuously monitor applications' memory accesses with hardware support from the memory controller. Nevertheless, ViPZonE's layered architecture means that implementing alternate memory management strategies could be done without significant changes to the existing framework. We leave the study of these alternative methods to future work.

### 3.1 Target Platform and Assumptions

We target generic x86-64 PC and server platforms that run Linux and have access to two or more DIMMs exhibiting some amount of power variability (ViPZonE cannot have a benefit with uniform memory power consumption). If device-level power variation is available, then this approach could be adapted to finer granularities, depending on the memory architecture. We make the following assumptions:

- ViPZonE's page allocator has prior knowledge of the approximate write and read power of each DIMM (for an identical workload). We could detect off-chip memory power variation, obtained by one of the following methods: (1) embedded power data in each DIMM, measured and set at fabrication time, or (2) through embedded or auxiliary power sensors sampled during the startup procedure.
- As DIMM-to-DIMM power variability is mostly dependent on process variations, and weakly dependent on temperature [10], there is little need for real-time monitoring of memory power use for each module. However, if power variation changes slowly over time (e.g., due to aging and wear-out occuring over time much greater than the uptime of

4. Our scheme does not currently support kernel memory allocations (e.g., *kmalloc()*). As the kernel space is generally a small proportion of overall system memory footprint, and invoked by all applications, we statically place the image and all dynamic kernel allocations in the low power zone.

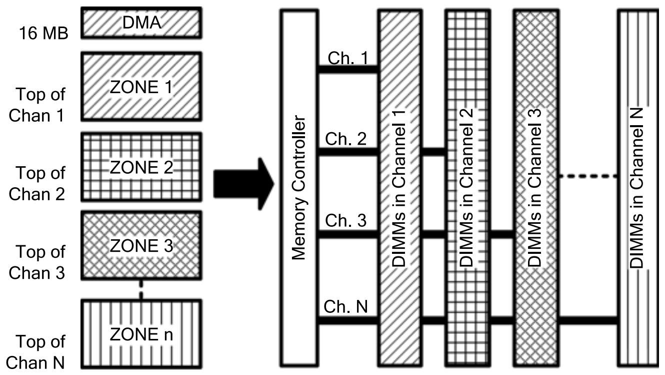


Fig. 6. ViPZonE modifications to Linux physical address space zoning for x86-64. The kernel maintains separate zones which correspond directly to different physical DIMMs.

the system after a single boot), we assume these changes can be detected through power sensors in each module.

- We can perform direct mapping of the address space[5] (e.g., select a single DIMM for each read/write request). This is achieved by disabling rank and channel interleaving on our testbed. We verified the direct mapping of addresses to specific DIMMs. Note that the particular division of DIMMs into ranks and channels is not a primary concern to ViPZonE; the only requirement is that only one DIMM is accessed per address.
- Programmers have a good understanding of the memory access patterns of their applications, obtained by some means, e.g., trace-driven simulations, allowing them to decide what dynamic memory allocations are "high utilization", or write-dominated, etc. Of course, in other scenarios, we do allow for annotation-independent policies.

## 3.2 Back-End: ViPZonE Implementation in the Linux Kernel

We discuss the implementation of ViPZonE in a bottom-up manner, in a similar fashion to the background material presented earlier. Our implementation is based on the Linux 3.2 and GLIBC 2.15 source for x86-64.

### 3.2.1 *Enhancing Physical Memory Zoning to Exploit Power Variability*

In order to support memory variability-awareness, the ViPZonE kernel must be able to distinguish between physical regions of different power consumption. With knowledge of these power profiles, it constructs separate physical address zones corresponding to each region of different power characteristics. The kernel can then serve allocation requests using the suggestions defined by the front-end of ViPZonE (see Section 3.3).

In the ViPZonE kernel for x86-64, we have explicitly removed the Normal and DMA32 zones, while still allowing for DMA32 allocation support. Regular DMA-able space is retained. Zones are added for each physical DIMM in the system (*Zone 1*, *Zone 2*, etc.), with page ranges corresponding to the actual physical space on each DIMM. Allocations requesting DMA32-capable memory are translated to using certain DIMMs that use the equivalent memory space (i.e., addresses under 4 GB). Fig. 6 depicts our modified memory zoning scheme for the back-end. For example, in a system supporting DMA and DMA32, with 8 GB of memory located on four DIMMs ($4 \times 2$ GB), the ViPZonE back-end would divide the memory space into zones as follows (we assume that each DIMM can have different power consumption):

- *Zone DMA*: 0-16 MB, mapped to *DIMM 1*.
- *Zone 1*: 16-2,048 MB, mapped to *DIMM 1*.
- *Zone 2*: 2,048-4,096 MB, mapped to *DIMM 2*.
- *Zone 3*: 4,096-6,144 MB, mapped to *DIMM 3*.
- *Zone 4*: 6,144-8,192 MB, mapped to *DIMM 4*.

### 3.2.2 *Modifying the Physical Page Allocation Algorithm in ViPZonE Linux x86-64*

With zones set up for each DIMM, and knowledge of the relative power consumption of each DIMM, the kernel has the essential tools it needs to make power variability-aware page allocations, whereas the vanilla kernel makes no distinction between modular boundaries. There are many possible physical page allocation policies that could be used in the ViPZonE framework.

The ViPZonE kernel makes a distinction between relative write and read power for each DIMM zone. This is done for the hypothetical case where a module that has the lowest write power may not have the lowest read power, etc. Furthermore, it allows for future applicability to non-volatile memories, such as the MTJ-based family (MRAM, STT-RAM, MeRAM) with large differences in read and write power and energy [42].

The default policy that we implemented is *Low Power First*. The policy is currently configurable at kernel compile-time, but could be changed to work on-the-fly if policies need to be changed at runtime. In this policy, for high-utilization allocations, the kernel tries to get the lowest read or write power zone available. For low-utilization requests, the kernel still tries to fulfill it in a low read or write power zone, as long as some free space is reserved for high-utilization requests. Allocations requiring DMA32-compatible space are restricted to zones less than 4 GB, but otherwise follow the same utilization-priority rules. Finally, legacy DMA requests (less than 16 MB) always are granted in *Zone DMA*. The *Low Power First* policy is depicted in Fig. 7.

For example, in a system with four DIMMs, each with 2 GB of space, the ViPZonE kernel would make allocation decisions as follows:

- *Request for DMA-capable space*: Grant in *Zone DMA*.
- *Request for DMA32-capable space (superset of DMA)*: Restrict possibilities to *Zone 1* or *Zone 2*. If indicated utilization is *low*, choose the lower *write* (if indicated) or *read* (if indicated, or default) power zone, as long as at least *THRESHOLD* free space is available (generally, we choose *THRESHOLD* to be approximately 20 percent of DIMM capacity). If indicated utilization

5. Note that address space layout randomization (ASLR) is not an issue, as ViPZonE deals with physical page placement only, while ASLR modifies the location of virtual pages.

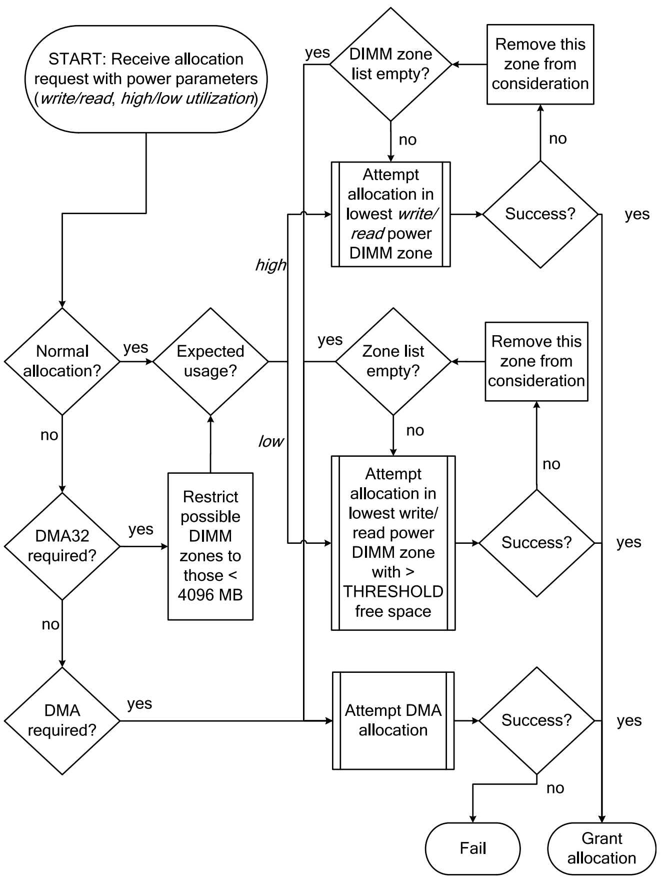

START: Receive allocation request with power parameters (write/read, high/low utilization)
Normal allocation?
yes
no
Expected usage?
high
low
DMA32 required?
yes
no
Restrict possible DIMM zones to those < 4096 MB
DMA required?
yes
DIMM zone list empty?
yes
no
Attempt allocation in lowest write/read power DIMM zone
Success?
yes
no
Remove this zone from consideration
Zone list empty?
Attempt allocation in lowest write/read power DIMM zone with > THRESHOLD free space
Attempt DMA allocation
Fail
Grant allocation

Fig. 7. *Low Power First* default ViPZonE memory allocation policy in Linux x86-64.

is *high*, always choose the lower power (*write/read*) zone if possible. If neither zone has enough free space, the kernel uses the vanilla page reclamation mechanisms or can default to *Zone DMA* as a last resort.

- *Request for Normal space*: Grant in any zone, with the order determined in the same fashion as the above case for DMA32, without the 4 GB address restriction. *Zone DMA* is only used as a last resort.

Alternatively, more sophisticated policies could use a variety of tricks. For example, a kernel which tracks page accesses might employ page migration to promote highly utilized pages to low-power space while demoting infrequently used pages to high power space. However, this would need to be carefully considered, as the performance and power costs of tracking and migrating pages might outweigh the benefits from exploiting power variation. We leave the study of these policies to future work.

### 3.3 Front-End: User API and System Call

The other major component of our ViPZonE implementation is the front-end, in the form of upper layer OS functionality in conjunction with annotations to application code. The front-end allows the programmer to provide hints regarding intended use for memory allocations, so that the kernel can prioritize low power zones for frequently written or read pages. The GNU standard C library (GLIBC) was used to implement our power

TABLE 1
ViPZonE API

| Function | Parameter | Type | Description |
|---|---|---|---|
| *void* vip_malloc* | bytes | size_t | Request size |
| | vip_flags | size_t | Bitmap flag used by ViPZonE back-end page allocator |
| (syscall) *void* vip_mmap* | addr | void * | Address to be mapped, typically NULL (best effort) |
| | len | size_t | Size to be allocated, in bytes |
| | prot | int | Standard *mmap()* protection flags bitwise ORed with *vip_flags* |
| | flags | int | Standard *mmap()* flags |
| | fd | int | Standard *mmap()* file descriptor |
| | pgoff | off_t | Standard *mmap()* page offset |

variation enhanced allocation functions as part of the standard library (source code available at [27]). We briefly describe the methods and their use.

We added two new features to the applications' API, described in Table 1, allowing the programmer to indicate to the virtual memory manager the intended usage. We implemented a new GLIBC function, *vip_malloc()*, as a new call to enable backwards compatibility with all non-ViPZonE applications requiring the use of vanilla *malloc()*. *vip_malloc()* is essentially a wrapper for a new syscall, *vip_mmap()*, that serves as the hook into the ViPZonE kernel. While a pure wrapper for a syscall is not ideal due to performance and fragmentation reasons, we found it sufficient to evaluate our scheme. *vip_malloc()* can be improved further to implement advanced allocation algorithms, such as those in various C libraries' *malloc()* routines.

Because low power memory space is likely to be precious, memory should be released to the OS as soon as possible when an application no longer needs it. As a result, we preferred the use of the *mmap()* syscall over *sbrk()*, which has the heap grow contiguously. With *sbrk()*, it is often the case that memory is not really freed (i.e., usable by the OS). For this reason, a ViPZonE version of the *sbrk()* syscall was not implemented. This also keeps the *vip_malloc()* code as simple as possible for evaluation. We do not expect that it would have a major effect on power or performance.

*vip_malloc()* can be used as a drop-in replacement for *malloc()* in application code, given the ViPZonE GLIBC is available. If the target system is not running a ViPZonE-capable kernel, *vip_malloc()* defaults to calling the vanilla *malloc()*. Custom versions of *free()* and the *munmap()* syscall are not necessary to work with the variability-aware memory manager.

The Linux 3.2 *mmap()* code was used as a template for the development of *vip_mmap()*. Furthermore, the kernel includes ViPZonE helper functions that allow it to pass down the flags from the upper layers in the software stack down to the lower levels, from custom *do_vip_mmap_pgoff()*,

TABLE 2
ViPZonE Supported Flags

| Parameter | Flag | Description |
|---|---|---|
| Dominant Access Type | _VIP_TYP_WRITE | The memory space will have more writes than reads |
| | _VIP_TYP_READ | The memory space will have more reads than writes |
| Relative Utilization | _VIP_LOW_UTIL | Low utilization (prefer low power space if plentiful) |
| | _VIP_HI_UTIL | High utilization (always prefer low power space) |

*vip_mmap_region()* down to the heavily modified physical page allocator routine (*__alloc_pages_nodemask()*). For this purpose, we reserved two unused bits in *vip_mmap()*'s *prot* field to contain the *vip_flags* passed down from the user.

Table 2 shows the sample set of flags supported by *vip_malloc()*, passed down to the ViPZonE back-end kernel to allocate pages from the preferred zone according to the mechanism described earlier in Section 3.2. The flags (*_VIP_TYP_READ*, *_VIP_TYP_WRITE*) tell the allocator that the expected workload is heavily read or write intensive, respectively.[6] If no flags are specified, the defaults of *_VIP_TYP_READ* and *_VIP_UTIL_LOW* are used. We decided to support these flags (only two bits) rather than using a different metric (e.g., measured utilization), since keeping track of page utilization would require higher storage and logic overheads.

An application could use the ViPZonE API to reduce memory power consumption by intelligently using the flags. For example, if a piece of code will use a dynamically-allocated array for heavy read utilization (e.g., an input for matrix multiplication), then it can request memory as follows:

```
retval = vip_malloc(arraySize * elementSize,
_VIP_TYP_READ | _VIP_HI_UTIL);
```

Alternatively, the application could use the syscall directly:

```
retval = vip_mmap(NULL, arraySize *
elementSize, PROT_READ | PROT_WRITE |
_VIP_TYP_READ | _VIP_HI_UTIL,
MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
```

For each *vip_mmap* call, the kernel tries to either expand an existing VM area that will suit the set of flags, or create a new area. When the kernel handles a request for physical pages, it checks the associated VM area, if applicable, and can use the ViPZonE flags passed from user-space to make an informed allocation.

As shown by this example, the necessary programming changes to exploit our variability-aware memory allocation scheme are minimal, provided the application developer has some knowledge about the application's memory access behavior.

6. It is left to the developer to determine what constitutes read or write dominance depending on the semantics of the code. In our implementation, this did not matter, as DIMMs with high write power also had high read power, etc.

## 4 Evaluation

In this section, we demonstrate that ViPZonE is capable of reducing total memory power consumption with negligible performance overheads, thus yielding energy savings for a given benchmark compared to vanilla Linux running on the same hardware. We start with an overview of our testbed and measurement hardware and configurations in Section 4.1. A comparison of the four combinations of memory interleaving modes is in Section 4.2 to quantify the advantages and disadvantages of disabling interleaving to allow variability-aware memory management. In Section 4.3 we compare the power, performance, and energy of ViPZonE software with respect to vanilla Linux as a baseline, using three alternate testbed configurations.

### 4.1 Testbed Setup

We constructed an x86-64 platform that would allow us fine control over hardware configuration for our purposes. Table 3 lists the important hardware components and configuration parameters used in all subsequent evaluations. The motherboard BIOS allowed us to enable and disable channel and rank interleaving independently, as well as adjust all voltages, clock speeds, memory latencies, etc. if necessary. Memory power was measured on a per-DIMM basis using an external data acquisition (DAQ) unit, as shown by the testbed photo in Fig. 8. Data was streamed to a laptop for post-processing.

Tables 3b and 3c list two different CPU and memory configurations that share the same parameters from Table 3a. Unless otherwise specified in the tables, all minor BIOS parameters were left at default values. In the *Fast2* configuration, two DIMMs populated the motherboard with up to 50 percent active total power variation (DIMMs *b1* and *c1* as depicted in Fig. 4 and measured in the same way as in [10]). In the *Slow2* configuration, the memory was set identically but the CPU was underclocked to 1.8 GHz, with only two cores enabled, while Intel TurboBoost and HyperThreading were disabled. None of these configurations specify anything about the channel and rank interleaving modes, which is evaluated in Section 4.2. Our testbed configurations were chosen to represent two flavors of systems, those which may be CPU-bound and those which may be memory-bound in performance.

We used eight benchmarks from the PARSEC suite [26] that are representative of modern parallel computing: *blackscholes*, *bodytrack*, *canneal*, *facesim*, *fluidanimate*, *freqmine*, *raytrace*, and *swaptions*.

### 4.2 Interleaving Analysis

Since disabling interleaving is required for ViPZonE functionality and exploitation of memory variability in our testbed environment, we measured the average memory power, execution time, and total memory energy for different PARSEC benchmarks under both testbed configurations.

TABLE 3
Different ViPZonE Testbed Configurations, Varying CPU Performance on Real Hardware

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| | (a) Common Testbed Configuration | | |
| CPU | Intel Core i7-3820 (Sandy Bridge-E) | CPU supporting features | All enabled (default) |
| Motherboard | Gigabyte X79-UD5 (socket LGA2011) | Linux kernel version | 3.2.1 |
| BIOS version | F8 | GLIBC version (baseline) | 2.15 |
| Storage | SanDisk SDSSDX120GG2 120 GB SSD (SATA 6 Gbps) | DAQ | NI USB-6215 |
| No. DIMMs | 2 | DIMM power sample rate | 1 kHz per DIMM |
| No. memory channels | 2 | Data logging | Second machine |
| DIMM capacity | 2 GB each | Base core voltage | 1.265 V |
| DDR3 data rate | 1333 MT/s | DRAM voltage | 1.5 V |
| | (b) *Fast2* | | |
| No. enabled cores | 4 | TurboBoost | Enabled |
| Nominal core clock | 3.6 GHz | HyperThreading | Enabled |
| | | No. of PARSEC threads | 8 |
| | (c) *Slow2* | | |
| No. enabled cores | 2 | TurboBoost | Disabled |
| Base core clock | 1.8 GHz | HyperThreading | Disabled |
| | | No. of PARSEC threads | 1 |

The available combinations of interleaving were:

- *Cint On, Rint On*. Channel interleaving and rank interleaving are enabled.
- *Cint On, Rint Off*. Channel interleaving is enabled, while rank interleaving is disabled.
- *Cint Off, Rint On*. Channel interleaving is disabled, while rank interleaving is enabled.
- *Cint Off, Rint Off*. Channel interleaving and rank interleaving are disabled.

In the high-end CPU configuration, *Fast2* (results in Figs. 9a, 9c, 9e), the CPU was set for maximum performance, with PARSEC running with eight threads to stress the memory system. For benchmarks with high memory utilization, such as *canneal*, *facesim*, and *fluidanimate*, we found that turning off channel interleaving generally reduced power consumption (Fig. 9a), while total memory energy increased or decreased depending on the application (Fig. 9e) due to degradation in performance from lower main memory throughput (Fig. 9c). Rank interleaving had less impact on power or performance, which suggests that the main throughput bottleneck is the effective bus width rather than the devices. Conversely, for workloads with lower main memory utilization, there was negligible difference in power, performance, and energy. This confirms our intuition that the benefits of interleaving are in the improvement of peak memory throughput, which is only a bottleneck for certain workloads where the CPU is sufficiently fast and/or application memory utilization is high.

In the slower CPU setup, *Slow2* (results in Figs. 9b, 9d, 9f), running only a single workload thread, interleaving generally had little effect on memory power, runtime, and memory energy, because the processor/application were unable to stress the memory system. In these cases, interleaving could be disabled with no detrimental effect to performance. Note that power savings of disabling interleaving could be higher if the memory controller used effective power management on individual idle DIMMs, as opposed to global DIMM power management. A related work on power aware page allocation [19] also required interleaving to be disabled in order to employ effective DIMM power management. As interleaving remains an issue for general DIMM-level power management schemes, further investigation into the inherent tradeoffs and potential solutions is an open research direction.

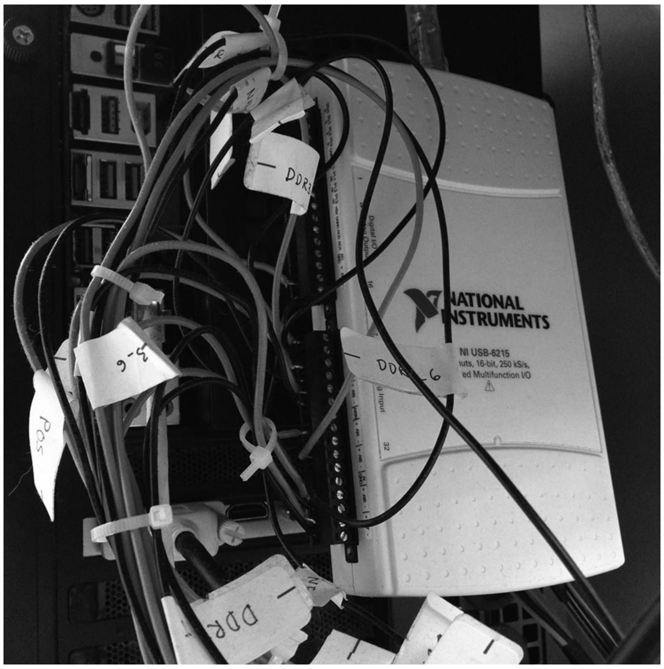
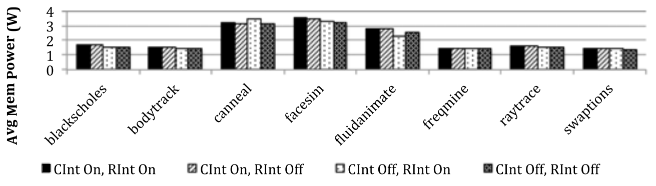


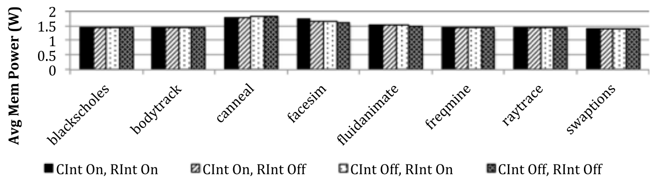
Fig. 8. Testbed photo showing our DAQ mounted on the reverse of the chassis for DIMM power measurement.

## 4.3 ViPZonE Analysis

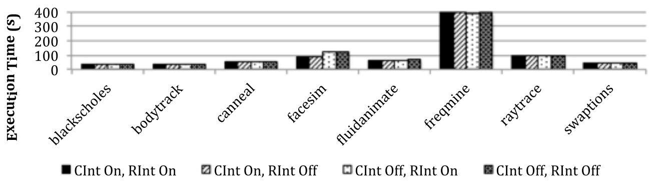
In the evaluation of ViPZonE software, channel and rank interleaving were always disabled, as it was a prerequisite



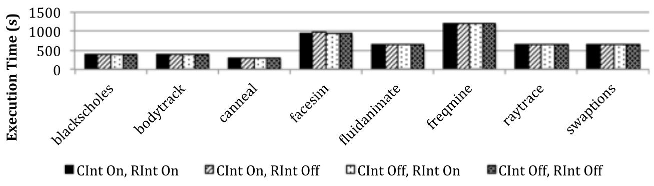
(a) *Fast2* Memory Power



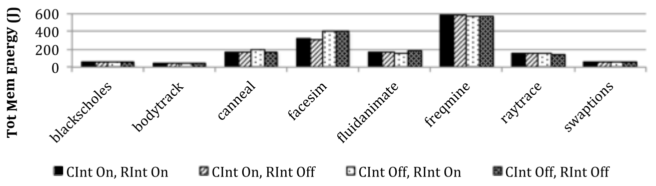
(b) *Slow2* Memory Power



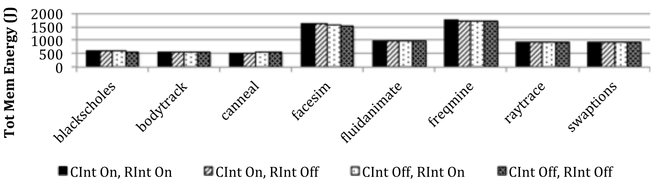
(c) *Fast2* Runtime



(d) *Slow2* Runtime



(e) *Fast2* Memory Energy



(f) *Slow2* Memory Energy

Fig. 9. Evaluation of channel and rank interleaving for both testbed configurations on vanilla Linux.

for functionality. The PARSEC benchmarks were not explicitly annotated (modified) for use with the ViPZonE front-end, although we have tested the full software stack for correct functionality. Instead, we emulated the effects of using all low-power allocation requests by using the default *Low Power First* physical page allocation policy described in Section 3.2.2. ViPZonE was benchmarked with two system configurations, namely *Fast2* (Figs. 10a, 10c, 10e) and *Slow2* (Figs. 10b, 10d, 10f). By using the two-DIMM configurations, we could better exploit memory power variability by including only the highest and lowest power two DIMMs from Fig. 4. Because we cannot harness idle power variability in our scheme, additional inactive DIMMs merely act as a "parasitic" effect on total memory power savings; with more DIMMs in the system, a greater proportion of total memory power/energy is consumed by idle DIMMs.

While ViPZonE does not explicitly deal with idle power management like other related works, it could be supplemented with orthogonal access coalescing methods which exploit deep low-power DRAM states. From the address mapping perspective, the nature of ViPZonE's zone preferences already implicitly allow more DIMMs to enter low-power (or, potentially off) modes by grouping allocations to a subset of the memory.

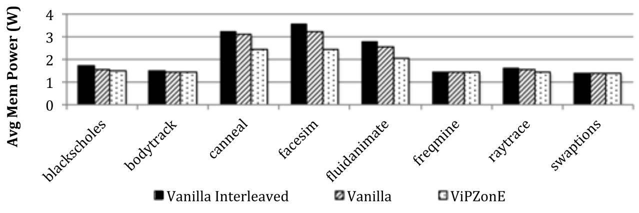
The results from the *Fast2* configuration indicate that ViPZonE can save up to 27.80 percent total memory energy for the *facesim* benchmark with respect to the vanilla software, also with interleaving disabled. Intuitively, our results make sense: benchmarks with higher memory utilization can better exploit active power variability between DIMMs. For this reason, lower-utilization benchmarks such as *blackscholes* gain no benefit from ViPZonE, just as they see no benefit from interleaving (refer to Section 4.2). With the slower CPU configuration, Figs. 10b, 10f indicate that there is a reduced benefit in power and energy from ViPZonE, for the same reasons. The reduced CPU speed, results in less stress being placed on the memory system, resulting in lower average utilization and a higher proportion of total memory power being from idleness.

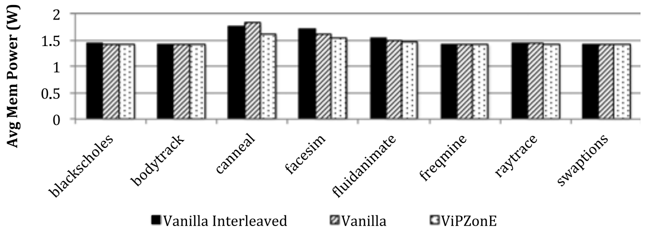
It may initially surprise the reader to note that in some cases, vanilla interleaved roughly matches ViPZonE on the energy metric. This is due to the performance advantage of interleaving. Because we currently have no way to combine ViPZonE with interleaving (although we propose possible solutions in Section 5), we use vanilla without interleaving as the primary baseline for comparison with ViPZonE. In other words, our primary baseline is on equal hardware terms with ViPZonE. The direct comparison with vanilla interleaved was included for fairness, as it merits discussion on whether a variation-aware scheme should be used over a conventional interleaved memory given the current state of DRAM memory organization. Nevertheless, we believe there is significant potential for further work on power variation-aware memory management, especially if there is a solution to allow interleaving simultaneously.

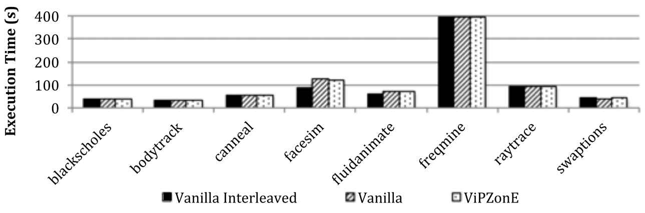
In all cases, ViPZonE running with channel and rank interleaving disabled achieved lower memory energy than the baseline vanilla software, with or without channel and rank interleaving.

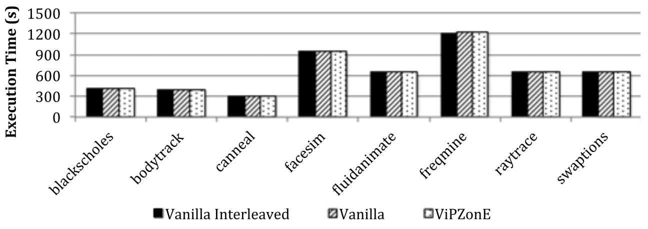
## 4.4 What-if: Expected Benefits with Non-Volatile Memories Exhibiting Ultra-Low Idle Power

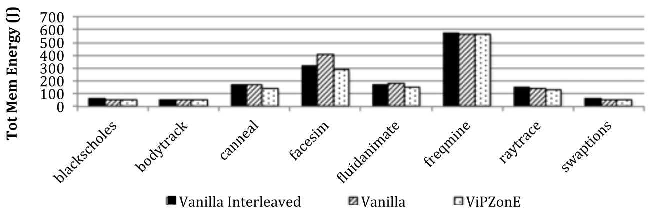
From the results of the ViPZonE comparison on our testbed, we speculate that the benefits of our scheme could be significantly greater with emerging non-volatile memory technologies, such as STT-RAM, PCM, etc. We expect that there are two primary characteristics of NVMs that would make ViPZonE more beneficial: (1) extremely low


(a) *Fast2* Memory Power


(b) *Slow2* Memory Power


(c) *Fast2* Runtime


(d) *Slow2* Runtime


(e) *Fast2* Memory Energy

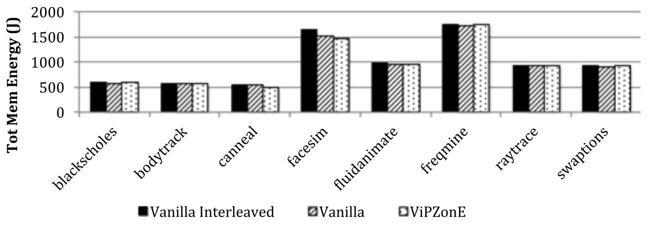
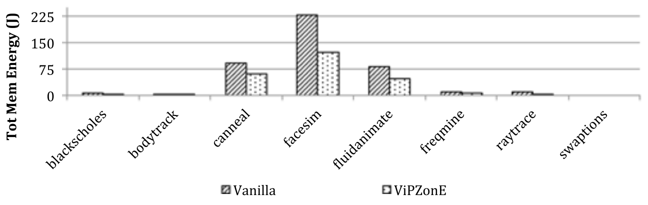

(f) *Slow2* Memory Energy

Fig. 10. ViPZonE vs. vanilla and interleaved (Cint On, Rint On) vanilla Linux.


(a) *Fast2* Memory Energy, idle energy removed

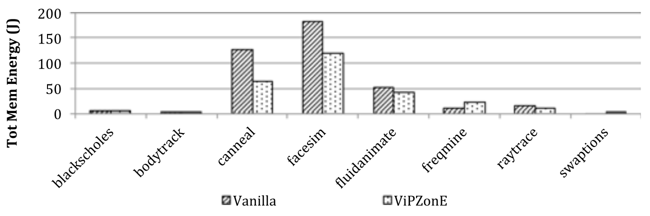

(b) *Slow2* Memory Energy, idle energy removed

Fig. 11. ViPZonE vs. Vanilla Linux, "what-if" evaluation for potential benefits with NVMs (CInt Off, Rint Off).

idle power, thus eliminating its aforementioned parasitic effect on access power variability-aware solutions, and (2) potentially higher process variability with novel devices, leading to higher power variability that can be opportunistically exploited.

Thus, we present the results with the idle power component removed.[7] While this by no means an accurate representation of the realities of non-volatile memories, such as asymmetric write and read power/performance and potential architectural optimizations, this is meant to illustrate how active power variability can be better exploited without the parasitic idle power. The idle power was approximately 1.41 W for the two DIMMs used in the *Fast2* and *Slow2* configurations, specifically DIMMs *b1* and *c1* from Fig. 4. As can been seen in Figs. 11a and 11b, the overall memory energy benefits could increase dramatically, up to 50.69 percent for the *canneal* benchmark. Although these numbers do not realistically represent the results with actual NVMs, as they were derived from our DRAM modules, they present a case for variability-aware solutions for future memories with low idle power and higher power variation.

7. No performance figures are presented, as we did not actually run the system with real non-volatile memories.

## 4.5 Summary of Results

Table 4 summarizes the results from the evaluation of ViPZonE, as well as the theoretical "what-if" study for potential application to NVM-based systems. We expect that with emerging NVMs, the lack of a significant idle power component will result in ViPZonE getting significant energy savings for workloads with a variety of utilizations, even as the number of modules in the system increase. Thus, using variability-aware memory allocation instead of interleaving would likely be a promising option for future systems.

TABLE 4
Summary of ViPZonE Results, with Respect to Vanilla Software with Channel and Rank Interleaving Disabled

| Metric | Value (Benchmark) | Metric | Value (Benchmark) |
|---|---|---|---|
| Max memory power savings, *Fast2* config | 25.13% (*facesim*) | Max memory power savings, *Slow2* config | 11.79% (*canneal*) |
| Max execution time overhead, *Fast2* config | 4.80% (*canneal*) | Max execution time overhead, *Slow2* config | 1.16% (*canneal*) |
| Max memory energy savings, *Fast2* config | 27.80% (*facesim*) | Max memory energy savings, *Slow2* config | 10.77% (*canneal*) |
| Max memory energy savings, *Fast2* config estimated, "NVM" (no idle power) | 46.55% (*facesim*) | Max memory energy savings, *Slow2* config estimated, "NVM" (no idle power) | 50.69% (*canneal*) |

## 5 Conclusion and Future Work

In this work, we implemented and evaluated ViPZonE, a system-level energy-saving scheme that exploits the power variability present in a set of DDR3 DRAM memory modules. ViPZonE is implemented for Linux x86-64 systems, and includes modified physical page allocation routines within the kernel, as well as a new system call. User code can reduce system energy use by using a new variant of *malloc()*, only requiring the ViPZonE kernel and C standard library support. Our experimental results, obtained using an actual hardware testbed, demonstrates up to 27.80 percent energy savings with no more than 4.80 percent performance degradation for certain PARSEC workloads compared to standard Linux software. A brief "what-if" study suggested that our approach could yield greatly improved benefits using emerging non-volatile memory technology that consume no idle power, notwithstanding potentially higher power variability compared to DRAMs. As our approach requires that no channel or rank interleaving be used, we also included a comparison of four different interleaving configurations on our testbed to evaluate the impact of interleaving on realistic workloads.

The lack of interleaving support in the current implementation of ViPZonE is its primary drawback. It is a general problem facing DIMM-level power management schemes, and we believe finding good tradeoffs remains an open research question. We do not claim that ViPZonE is the best solution for all applications and systems. Rather, we think that it is an interesting demonstration of a novel memory management concept in a realistic setting, and motivates further research in this space.

There are several opportunities for further research with ViPZonE. First, given the ability to co-design hardware and software, it might be possible to combine the benefits of interleaving for performance while exploiting power variation for energy savings. We can imagine a few ways this could be done. One solution would use a modified memory controller that interleaves different groups of DIMMs independently. This compromise would allow for performance and potential energy savings somewhere between the current interleaving vs. ViPZonE scenario, but would still be a static design-time or boot-time decision. This could be useful in systems that already have clustered memory, such as non-uniform memory access (NUMA) architectures.

Alternatively, hypothetical systems with disparate memory device technologies side-by-side (e.g., a hybrid DRAM-PCM memory as in [43]) may discourage interleaving across device types due to different access power, latency, read/write asymmetry, and data volatility. In this case, interleaving could still be used within each cluster of homogeneous memory technology, and each such cluster could be used as a single zone for ViPZonE. The result would be ViPZonE becoming heterogeneity-aware as a generalization of variability-awareness.

A more radical idea which may allow the full benefits of interleaving alongside ViPZonE would likely require a redesign of the DIMM organization to allow individual DIMMs, where each rank is multi-ported, to occupy multiple channels. However, the major issue we forsee with this is a much higher memory cost due to the multiplied pin requirements.

Aside from enabling interleaving alongside variation-aware memory management, ViPZonE could potentially be improved on the software side. Adding compiler support could take some of the burden off the programmer while expanding the scope to include static program data. Variability-aware page migration schemes might yield further improvements in energy efficiency by augmenting our static allocation-time hints. Our approach could likely be complemented by several other power-aware methods mentioned in Section 1.1. A simulation study of ViPZonE with detailed models of non-volatile memories could give a better idea of the benefits in the future, where power and delay variation are likely to be higher and there is negligible idle power.

We believe ViPZonE makes an effective case for further research into the Underdesigned and Opportunistic computing paradigm with the goal of improving energy efficiency of systems, while lowering design cost, improving yield, and recovering lost performance due to conventional guardbanding techniques.

## Acknowledgments

This work was supported jointly by US National and Science Foundation (NSF) Variability Expedition Grant Numbers CCF-1029783 and CCF-1029030.

## References

[1] K. Bowman, S. Duvall, and J. Meindl, "Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration," *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 183–190, Feb. 2002.
[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proc. Des. Autom. Conf.*, 2003, pp. 338–342.
[3] The international technology roadmap for semiconductors. (2012). [Online]. Available: http://www.itrs.net

[4] N. Dutt, P. Gupta, A. Nicolau, L. Bathen, and M. Gottscho, "Variability-aware memory management for nanoscale computing," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2013, pp. 125–132.

[5] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. Rosing, M. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 32, no. 1, pp. 8–23, Jan. 2013.

[6] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for MPSoC," in *Proc. Int. Conf. Comput.-Aided Des.*, 2007, pp. 598–603.

[7] J. Sartori, A. Pant, R. Kumar, and P. Gupta, "Variation-aware speed binning of multi-core processors," in *Proc. Int. Symp. Qual. Electron. Des.*, 2010, pp. 307–314.

[8] H. Hanson, K. Rajamani, J. Rubio, S. Ghiasi, and F. Rawson, "Benchmarking for power and performance," in *Proc. SPEC Benchmark Workshop*, 2007.

[9] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava, "A case for opportunistic embedded sensing in presence of hardware power variability," in *Proc. Workshop Power Aware Comput. Syst.*, 2010.

[10] M. Gottscho, A. Kagalwalla, and P. Gupta, "Power variability in contemporary DRAMs," *IEEE Embedded Syst. Lett.*, vol. 4, no. 2, pp. 37–40, Jun. 2012.

[11] A. Pant, P. Gupta, and M. van der Schaar, "Software adaptation in quality sensitive applications to deal with hardware variability," in *Proc. Great Lakes Symp. VLSI*, 2010, pp. 85–90.

[12] L. Wanner, R. Balani, S. Zahedi, C. Apte, P. Gupta, and M. Srivastava, "Variability-aware duty cycle scheduling in long running embedded sensing systems," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–6.

[13] K. Meng and R. Joseph, "Process variation aware cache leakage management," in *Proc. Int. Symp. Low Power Electron. Des.*, 2006, pp. 262–267.

[14] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks, "Process variation tolerant 3T1D-based cache architectures," in *Proc. Int. Symp. Microarchit.*, 2007, pp. 15–26.

[15] M. Mutyam, F. Wang, R. Krishnan, V. Narayanan, M. Kandemir, Y. Xie, and M. Irwin, "Process-variation-aware adaptive cache architecture and management," *IEEE Trans. Comput.*, vol. 58, no. 7, pp. 865–877, Jul. 2009.

[16] A. Sasan, H. Homayoun, A. Eltawil, and F. Kurdahi, "A fault tolerant cache architecture for sub 500mV operation: Resizable data composer cache (RDC-cache)," in *Proc. Int. Conf. Compilers, Archit., Synth. Embedded Syst.*, 2009, pp. 251–260.

[17] L. Bathen and N. Dutt, "E-RoC: Embedded RAIDs-on-Chip for low power distributed dynamically managed reliable memories," in *Proc. Des., Autom., Test Eur. Conf. Exhib.*, 2011, pp. 1–6.

[18] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, "Energy management for commercial servers," *IEEE Comput.*, vol. 36, no. 12, pp. 39–48, Dec. 2003.

[19] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power aware page allocation," in *Proc. Int. Conf. Arch. Support Program. Lang. Oper. Syst.*, 2000, pp. 105–116.

[20] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Scheduler-based DRAM energy management," in *Proc. Des. Autom. Conf.*, 2002, pp. 697–702.

[21] P. Zhou, V. Pandey, J. Sundaresan, A. Raghuraman, Y. Zhou, and S. Kumar, "Dynamic tracking of page miss ratio curve for memory management," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2004, pp. 177–188.

[22] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu, "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency," in *Proc. Int. Symp. Microarchitecture*, 2008, pp. 210–221.

[23] J. H. Ahn, J. Leverich, R. Schreiber, and N. P. Jouppi, "Multicore DIMM: An energy efficient memory module with independently controlled DRAMs," *IEEE Comput. Archit. Lett.*, vol. 8, no. 1, pp. 5–8, Jan. 2008.

[24] L. Bathen, N. Dutt, A. Nicolau, and P. Gupta, "VaMV: variabzility-aware memory virtualization," in *Proc. Conf. Des., Autom., Test Eur.*, 2012, pp. 284–287.

[25] L. Bathen, M. Gottscho, N. Dutt, P. Gupta, and A. Nicolau, "ViPZonE: OS-level memory variability-driven physical address zoning for energy savings," in *Proc. Int. Conf. Hardware/Softw. Codesign Syst. Synth.*, 2012, pp. 33-42.

[26] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compilation Tech.*, 2008, pp. 72–81.

[27] M. Gottscho and L. Bathen. (2013). ViPZonE source code [Online]. Available: http://github.com/nanocad-lab/

[28] H. S. Stone, *High-Performance Computer Architecture*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1993.

[29] G. J. Burnett and E. G. Coffman Jr., "A study of interleaved memory systems," in *Proc. Spring Joint Comput. Conf.*, 1970, pp. 467–474.

[30] B. R. Rau, "Interleaved memory bandwidth in a model of a multiprocessor computer system," *IEEE Trans. Comput.*, vol. C-100, no. 9, pp. 678–681, Sept. 1979.

[31] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed. San Mateo, CA, USA: Morgan Kaufmann, 2012.

[32] P. Budnik and D. J. Kuck, "The organization and use of parallel memories," *IEEE Trans. Comput.*, vol. C-20, no. 12, pp. 1566–1569, Dec. 1971.

[33] D. J. Kuck and R. Stokes, "The Burroughs scientific processor (BSP)," *IEEE Trans. Comput.*, vol. C-31, no. 5, pp. 363–376, May 1982.

[34] G. Sohi, "High-bandwidth interleaved memories for vector processors-a simulation study," *IEEE Trans. Comput.*, vol. 42, no. 1, pp. 34–44, Jan. 1993.

[35] M. Breternitz, and J. P. Shen, "Organization of array data for concurrent memory access," in *Proc. Annu. Workshop Microprogram. Microarchit.*, 1988, pp. 97–99.

[36] R. Love, *Linux Kernel Development*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Education, 2010.

[37] D. Bovet, M. Cesati, and A. Oram, *Understanding the Linux Kernel*. Sebastopol, CA, USA: O'Reilly & Assoc., 2002.

[38] M. Gorman, *Understanding the Linux Virtual Memory Manager*, 2007.

[39] The linux kernel 3.2. (2012) [Online]. Available: www.kernel.org

[40] GLIBC, the GNU c library. (2012). [Online]. Available: http://www.gnu.org/s/libc/

[41] H.-W. Lee, K.-H. Kim, Y.-K. Choi, J.-H. Sohn, N.-K. Park, K.-W. Kim, C. Kim, Y.-J. Choi, and B.-T. Chung, "A 1.6V 1.4 Gbp/s/pin consumer DRAM with self-dynamic voltage scaling technique in 44nm CMOS technology," *IEEE J. Solid-State Circuits*, vol. 47, no. 1, pp. 131–140, Jan. 2012.

[42] K. L. Wang, J. G. Alzate, and P. Khalili Amiri, "Low-power non-volatile spintronic memory: STT-RAM and beyond," *J. Phys. D: Appl. Phys.*, vol. 46, no. 7, 2013.

[43] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Proc. Des. Autom. Conf.*, 2009, pp. 464–469.

**Mark Gottscho** (S) received the BS degree in 2011 and the MS degree with Great Distinction in 2014 from University of California, Los Angeles (UCLA), where he is currently working toward the PhD degree in electrical engineering, advised by Dr. Puneet Gupta. His research interests include energy-aware systems, hardware/software codesign, memory, and variability. He received an Honorable Mention for the National Science Foundation Graduate Research Fellowship Program in 2014. He is a student member of the IEEE.

**Luis A.D. Bathen** (M) received the PhD degree in information and computer science from the University of California, Irvine, in 2012. He is currently working at the US Department of Defense as an information assurance scientist. His current research interests include embedded systems, information assurance/cyber, storage systems, and cloud computing. He is a member of the IEEE.

**Nikil Dutt** (F) received the PhD degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, in 1989. He is currently a Chancellor's professor of CS, EECS, and cognitive sciences at the University of California, Irvine. His current research interests include embedded systems, electronic design automation, computer architecture, systems software, formal methods, and brain-inspired architectures and computing. He is an ACM distinguished scientist and an IFIP Silver Core Awardee. He has served as the editor-in-chief of the *ACM Transactions on Design Automation of Electronic Systems*. He currently serves as an associate editor for the *ACM Transactions on Embedded Computing Systems* and the *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. He is a fellow of the IEEE.

**Alex Nicolau** (M) received the PhD degree from Yale University, New Haven, CT, in 1984. He is currently a professor with the Department of Computer Science and the Department of Electrical Engineering and Computer Science, University of California, Irvine. His current research interests include systems, electronic design automation, computer architecture, systems software, parallel processing, and high-performance computing. He has authored or coauthored more than 300 peer-reviewed articles and multiple books. He serves as the editor-in-chief of IJPP and he was/is on the steering/program committees of the ACM International Conference on Supercomputing, the Symposium on Principles and Practice of Parallel Programming, the IEEE International Conference on Parallel Architectures and Compilation Technology, MICRO, and the Workshop on Languages and Compilers for Parallel Computing. He is a member of the IEEE.

**Puneet Gupta** (S'02-M'08) received the PhD degree from the University of California, San Diego, in 2007. He cofounded Blaze DFM, Inc., Sunnyvale, CA, in 2004. He is currently an associate professor with the Department of Electrical Engineering, University of California, Los Angeles. He currently leads the IMPACT+ Center (impact.ee.ucla.edu) and has served on several conference (DAC, ICCAD, ASPDAC, etc.) program committees. His research interests include optimizing design-manufacturing and hardware-software interfaces for lowered costs and improved power/performance of integrated circuits and systems. He received the National Science Foundation CAREER Award, the ACM/SIGDA Outstanding New Faculty Award, IBM Faculty Award, and SRC Inventor Recognition Award. He has published more than 100 papers and holds 15 US patents. He is a member of the IEEE.