Energy-efficient Cluster Computing with FAWN: Workloads and Implications

Vijay Vasudevan, David Andersen, Michael Kaminsky*
Lawrence Tan, Jason Franklin, Iulian Moraru

Carnegie Mellon University, *Intel Labs Pittsburgh

Abstract

This paper presents the architecture and motivation for a cluster-based, many-core computing architecture for energy-efficient, data-intensive computing. FAWN, a Fast Array of Wimpy Nodes, consists of a large number of slower but efficient nodes coupled with low-power storage. We present the computing trends that motivate a FAWN-like approach, for CPU, memory, and storage. We follow with a set of microbenchmarks to explore under what workloads these "wimpy nodes" perform well (or perform poorly). We conclude with an outline of the longer-term implications of FAWN that lead us to select a tightly integrated stacked chip-and-memory architecture for future FAWN development.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Distributed Systems*; D.4.2 [**Operating Systems**]: Storage Management; D.4.8 [**Operating Systems**]: Performance—*Measurements*

General Terms

Performance, Experimentation, Measurement

Keywords

Design, Energy Efficiency, Performance, Measurement, Cluster Computing, Flash

1. INTRODUCTION

Power is becoming an increasingly large financial and scaling burden for computing and society. The power draw of large data centers is a growing fraction of their cost—up to 50% of the three-year total cost of owning a computer —to the point that companies such as Microsoft, Google, and Yahoo! have built new data centers close to large and cost-efficient hydroelectric power sources [12]. Datacenter density is limited by their ability to supply and cool 10–20 kW of power per rack and up to 10–20 MW per datacenter [16]. Future datacenters may require as much as 200 MW [16], and today, datacenters are being constructed with dedicated electrical substations to feed them. While power constraints have pushed the processor industry toward multi-core architectures, energy-efficient alternatives to traditional disk and DRAM-based cluster architectures have been slow to emerge.

As an energy-efficient alternative for data-intensive computing, we present a cluster architecture called a *Fast Array of Wimpy Nodes*, or FAWN. A FAWN consists of a large number of slower but efficient nodes that each draw only a few watts of power, coupled with low-power storage. We have explored prototype FAWN nodes ranging from five-year old, 500MHz embedded devices using CompactFlash storage, to more modern Intel Atom-based nodes with fast solid-state drives.

In this paper, we describe the long-lasting, fundamental trends in the scaling of computation and energy that suggest that the FAWN approach will become dominant for increasing classes of workloads. First, as we show in §2, slower processors are more efficient: they use fewer joules of energy per instruction than higher speed processors. Second, dynamic power scaling techniques are less effective than reducing a cluster's peak power consumption. After examining CPU scaling trends, we similarly examine the same scaling questions for both memory capacity/speed and for storage.

We then summarize our experience with real FAWN architectures for a variety of workloads: seek-bound, I/O-throughput bound, memory-bound, and CPU-bound. FAWN can be several times more efficient than traditional systems for I/O-bound workloads, and on par with or more efficient for many memory and CPU-limited applications (§3). We also highlight where FAWN nodes are less energy-efficient than traditional systems.

We conclude with a future roadmap for FAWN-like hardware, exploring two diverging paths for constructing low-GHz, manycore systems for data-intensive applications.

2. COMPUTING TRENDS

The FAWN approach to building *well-matched* cluster systems has the potential to achieve high performance and be fundamentally more energy-efficient than conventional architectures for serving massive-scale I/O and data-intensive workloads. We measure system performance in work done per second and measure energy-efficiency in work done per Joule (equivalently, performance per Watt). FAWN is inspired by several fundamental trends:

Increasing CPU-I/O Gap: Over the last several decades, the gap between CPU performance and I/O bandwidth has continually

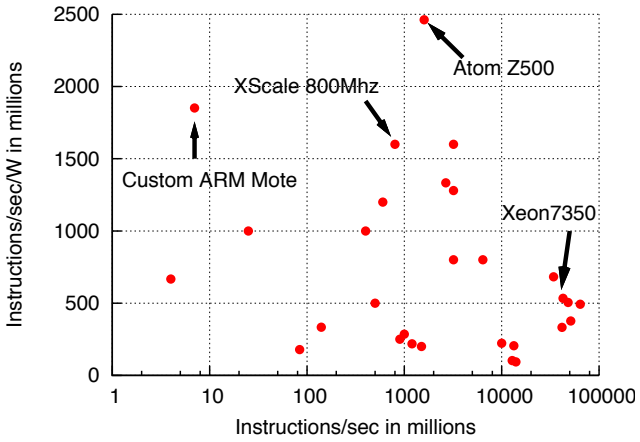

Figure 1: Max speed (MIPS) vs. Instruction efficiency (MIPS/W) in log-log scale. Numbers gathered from publicly-available spec sheets and manufacturer product websites.

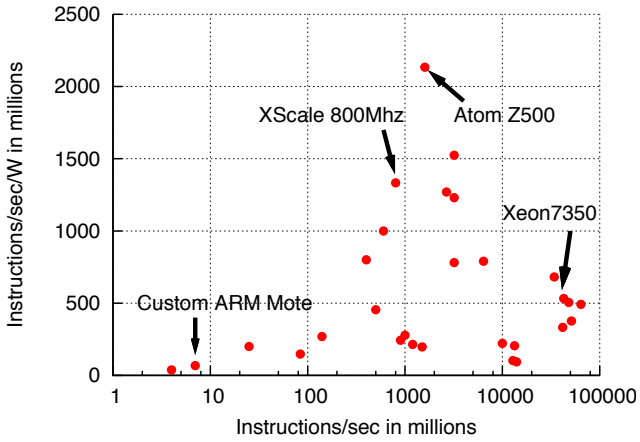

Figure 2: Processor efficiency when adding fixed 0.1W system overhead.

grown. For data-intensive computing workloads, storage, network, and memory bandwidth bottlenecks often cause low CPU utilization.

FAWN Approach: To efficiently run I/O-bound data-intensive, computationally simple applications, FAWN uses wimpy processors selected to reduce I/O-induced idle cycles while maintaining high performance. The reduced processor speed then benefits from a second trend:

CPU power consumption grows super-linearly with speed. Operating processors at higher frequency requires more energy, and techniques to mask the CPU-memory bottleneck come at the cost of energy efficiency. Branch prediction, speculative execution, out-of-order execution and increasing the amount of on-chip caching all require additional processor die area; modern processors dedicate as much as half their die to L2/3 caches [14]. These techniques do not increase the speed of basic computations, but do increase power consumption, making faster CPUs less energy efficient.

FAWN Approach: A FAWN cluster's slower CPUs dedicate more transistors to basic operations. These CPUs execute significantly more *instructions per Joule* than their faster counterparts (Figure 1): multi-GHz superscalar quad-core processors can execute approximately 100 million instructions per Joule, assuming all cores are active and avoid stalls or mispredictions. Lower-frequency in-order CPUs, in contrast, can provide over 1 billion instructions per Joule—an order of magnitude more efficient while still running at $1/3$rd the frequency.

Worse yet, running fast processors below their full capacity draws a disproportionate amount of power:

Dynamic power scaling on traditional systems is surprisingly inefficient. A primary energy-saving benefit of dynamic voltage and frequency scaling (DVFS) was its ability to reduce voltage as it reduced frequency [29], but modern CPUs already operate near minimum voltage at the highest frequencies.

Even if processor energy was completely proportional to load, non-CPU components such as memory, motherboards, and power supplies have begun to dominate energy consumption [4], requiring that all components be scaled back with demand. As a result, running a modern, DVFS-enabled system at 20% of its capacity may still consume over 50% of its peak power [27]. Despite improved power scaling technology, systems remain most energy-efficient when operating at peak utilization. Given the difficulty of scaling all system components, we must therefore consider "constant factors" for power when calculating a system's instruction efficiency. Figure 2 plots processor efficiency when adding a fixed 0.1W cost for system components such as Ethernet. Because powering 10Mbps Ethernet dwarfs the power consumption of the tiny sensor-type processors that consume only micro-Watts of power, their efficiency drops significantly. The best operating point exists in the middle of the curve, where the fixed costs are amortized while still providing energy efficiency.

Newer techniques aim for energy proportionality by turning machines off and using VM consolidation, but the practicality of these techniques is still being explored. Many large-scale systems often operate below 50% utilization, but opportunities to go into deep sleep states are few and far between [4], while "wake-up" or VM migration penalties can make these techniques less energy-efficient. Also, VM migration may not apply for some applications, e.g., if datasets are held entirely in DRAM to guarantee fast response times.

Even if techniques for dynamically scaling below peak power were effective, operating below peak power capacity has one more drawback:

Peak power consumption limits data center density. Data centers must be provisioned for a system's maximum power draw. This requires investment in infrastructure, including worst-case cooling requirements, provisioning of batteries for backup systems on power failure, and proper gauge power cables. FAWN significantly reduces maximum power draw in comparison to traditional cluster systems that provide equivalent performance, thereby reducing infrastructure cost, reducing the need for massive overprovisioning, and removing one limit to the achievable density of data centers.

Finally, energy proportionality alone is not a panacea: systems ideally should be both proportional *and* efficient at 100% load. In this paper, we show that there is significant room to improve energy efficiency, and the FAWN approach provides a simple way to do so.

2.1 Memory trends

The previous section examined the trends that cause CPU power to increase drastically with an increase in sequential execution speed. In pursuit of a balanced system, one must ask the same question of memory and storage as well.

Understanding DRAM power draw. DRAM has, at a high level, three major categories of power draw:

Idle/Refresh power draw: DRAM stores bits in capacitors; the charge in those capacitors leaks away and must be periodically refreshed (the act of reading the DRAM cells implicitly refreshes the contents). As a result, simply storing data in DRAM requires non-negligible power.

Precharge and read power: The power consumed inside the DRAM chip. When reading a few bits of data from DRAM, a larger line of cells is actually precharged and read by the sense amplifiers. As a result, random accesses to small amounts of data in DRAM are less power-efficient than large sequential reads.

Memory bus power: A significant fraction of the total memory system power draw—perhaps up to 40%—is required for transmitting read data over the memory bus back to the CPU or DRAM controller.

Design tradeoffs: Designers can somewhat improve the efficiency of DRAM (in bits read per joule) by clocking it more slowly, for the same reasons mentioned for CPUs. In addition, both DRAM access latency and power grow with the distance between the CPU (or memory controller) and the DRAM: without additional amplifiers, latency increases quadratically with trace length, and power increases at least linearly. This effect creates an intriguing tension for system designers: Increasing the amount of memory per CPU simultaneously increases the power cost to access a bit of data. The reasons for this are several: To add more memory to a system, desktops and servers use a bus-based topology that can handle a larger number of DRAM chips; these buses have longer traces and lose signal with each additional tap. In contrast, the low-power DRAM used in embedded systems (cellphones, etc.), LPDDR, uses a point-to-point topology with shorter traces, limiting the number of memory chips that can be connected to a single CPU, and reducing substantially the power needed to access that memory.

2.2 Storage Power Trends

The energy draw of magnetic platter-based storage is related to several device characteristics, such as storage bit density, capacity, throughput, and latency. Spinning the platter at faster speeds will improve throughput and seek times, but requires more power because of the additional rotational energy and air resistance. Capacity increases follow bit density improvements and also increase with larger platter sizes, but air resistance increases quadratically with larger platter sizes, so larger platters also require more power to operate.

Figure 3 demonstrates this tradeoff by plotting the efficiency versus speed for several modern hard drives, including enterprise, mobile, desktop, and "Green" products.[1] The fastest drives spin at between 10-15K RPM, but they have a relatively low energy efficiency as measured by MB per Joule of max sustained sequential data transfer. The 2.5" disk drives are nearly always more energy efficient than the 3.5" disk drives. The most efficient drives are 2.5" disk drives running at 5400 RPM. Energy efficiency therefore comes at the cost of per-device storage capacity for magnetic hard drives.

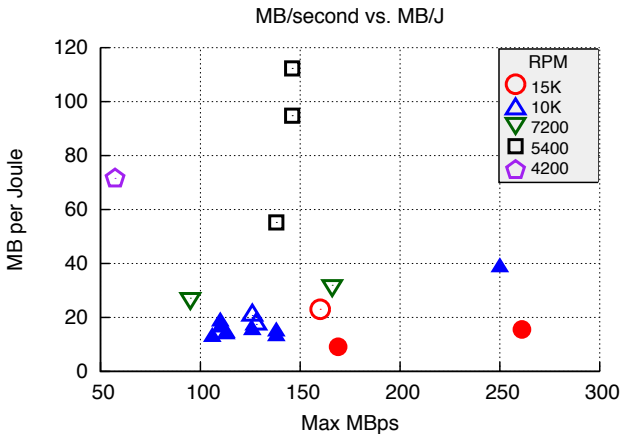

Figure 3: Power increases with rotational speed and platter size. Solid shapes are 3.5" disks and outlines are 2.5" disks. Speed and power numbers acquired from product specification sheets.

Our preliminary investigations into flash storage power trends indicate that the number of IOPS provided by the device scales roughly linearly with the power consumed by the device, likely because these devices increase performance through chip parallelism instead of by increasing the speed of a single component.

3. WORKLOADS

In this section, we describe under what conditions a FAWN architecture can provide superior energy efficiency, and where traditional architectures can be as efficient, or in some cases, more energy-efficient than low-power systems.

3.1 Metrics

Evaluating large systems using only performance metrics such as throughput or latency is slowly falling out of favor as energy and space constraints inform the design of modern large scale systems. There are several metrics for energy efficiency, but the one we focus on is "work done per Joule" of energy, or equivalently, "performance per Watt."

Low-power VLSI designs have alternatively looked at the "energy-delay product," which multiplies the amount of energy to do an amount of work with the time it takes to do that amount of work. This penalizes solutions that reduce the amount of energy by reducing performance for energy efficiency gains. Others have gone further by proposing using "energy $delay^2$" to further penalize solutions that simply reduce voltage at the expense of performance.

[1] The figure uses MB/s data from vendor spec sheets, which are often best-case outer-track numbers. The absolute numbers are therefore somewhat higher than what one would expect in typical use, but the relative performance comparison is likely accurate.

However, for large-scale cluster computing applications that are consuming a significant fraction of energy in datacenters worldwide, "work done per Joule" is an appropriate metric. This metric relies on being able to parallelize workloads, which is often explicitly provided by data-intensive computing models such as MapReduce [8] that harness data-parallelism.

More specifically, when the amount of work is fixed but parallelizable, one can use a larger number of slower machines yet still finish the work in the same amount of time—for example, ten nodes running at one-tenth the speed of a traditional node. If the aggregate power used by those ten nodes is less than that used by the traditional node, then the ten-node solution is more energy-efficient.

3.2 Taxonomy

We begin with a broad classification of the types of workloads found in data-intensive computing whose solution requires large-scale datacenter deployments:

1. I/O-bound workloads
2. Memory/CPU-bound workloads
3. Latency-sensitive, but non-parallelizable workloads
4. Large, memory-hungry workloads

The first of these workloads, I/O-bound workloads, have running times that are determined primarily by the speed of the I/O devices (typically disks for data-intensive workloads). I/O-bound workloads can be either seek- or scan-bound, and represent the low-hanging fruit for the FAWN approach, as described in our earlier work [2]. The second category includes CPU and memory-bound workloads, where the running time is limited by the speed of the CPU or memory system.

The last two categories represent workloads where the FAWN approach may be less useful. Latency-sensitive workloads require fast responses times to provide, for example, an acceptable user-experience; anything too slow (e.g., more than 50ms) impairs the quality of service unacceptably. Finally, large, memory-hungry workloads frequently access data that can reside within the memory of traditional servers (on the order of a few to 10s of gigabytes per machine today). As we describe in Section 3.5.2, the data structure created in `grep` when searching for millions of short phrases requires several gigabytes of memory and is accessed randomly. This causes frequent swapping on FAWN nodes with limited memory, but fits entirely in DRAM on modern servers.

3.3 I/O-bound workloads

Our prior work proposed the Fast Array of Wimpy Nodes (FAWN) architecture, which uses a large number of "wimpy" nodes that act as data storage/retrieval nodes [2]. These nodes use energy-efficient, low-power processors combined with low-power storage and a small amount of DRAM. We compare FAWN-type systems with traditional architectures to understand which system is more energy-efficient in terms of work done per Joule. For all subsequent experiments, we use a "Watts Up?" power meter that logs power draw at the wall socket once per second [28]. We sum the number of Joules during the course of each experiment to compute energy efficiency values, and report the average power draw during the course of the experiment where appropriate.

System / Storage	QPS	Watts	Queries / Joule
Embedded Systems			
Alix3c2 / Sandisk(CF)	1298	3.75	346
Modern Systems			
Server i7 / Fusion-io	61494	194	317.0
Desktop i7 / X25-E (x6)	59448	98.0	606.6
Atom* / X25-E	10760	22.3	482.5

Table 1: Query performance and efficiency for different machine configurations. The Atom node is a prototype.

The first workload, small-key value lookup, examines exact key-value queries at large scale such as those seen in `memcached` and Amazon's Dynamo [9]. The second workload class examines unstructured text mining queries similar to those expressed by simple tools such as `grep`ing through a massive dataset, or by more complex frameworks such as Hadoop and MapReduce.

3.3.1 *Key-value lookup*

Table 1 presents the exploration that we began in our previous work. It shows the rate at which various node configurations can service requests for random key-value pairs (1 KB values) from an on-disk dataset, via the network. The best embedded system (Alix3c2) using CompactFlash (CF) storage was six times more power-efficient (in queries/joule) than even the low-power desktop node with a 2008-era SATA-based flash device.

The low-power server market has expanded dramatically within the last year. We have since benchmarked several modern systems to understand which platform can provide the highest queries per Joule for persistent key-value storage. We have included in our comparisons three different systems that all use modern flash devices. At the high-end server level (Server i7), we use a dual-socket quad-core, rackmount Intel Core i7 (Nehalem) processor system with 16 GB of DRAM and an 80 GB Fusion-io ioDrive on a PCI-e interface. To approximate a modern low-power server, we used a prototype Intel "Pineview" Atom-based system with two 1.8GHz cores, 2 GB of DRAM and an Intel X25-E SATA-based SSD. Unfortunately, production versions of this system were not available at the time we conducted this research: The prototype had only a 100 Mbps Ethernet, which limited its performance, and the motherboard used low-efficiency voltage converters, which increased its power consumption. Between these extremes, we configured a "desktop" Core i7-based system with a single quad-core Core i7 860, 2 GB of DRAM, and 6 X25-E SATA drives. We attempted to balance this system by adding two SATA PCI-e cards because the motherboard supported only 4 SATA ports. We also reduced the power of this system by replacing the 40 W graphics card with a PCI card, and removed several extra DRAM chips for this particular experiment; through these efforts we reduced the desktop idle power to 45 W.

Table 1 shows that both the high-end server and desktop system could serve about 60,000 1 KB queries per second from flash (queries and responses are over the network); the server's power draw was 194 W averaged over the length of the experiment, whereas the desktop's was far less at 98 W. Thus, the desktop system was twice as energy-efficient as the server machine. In contrast,

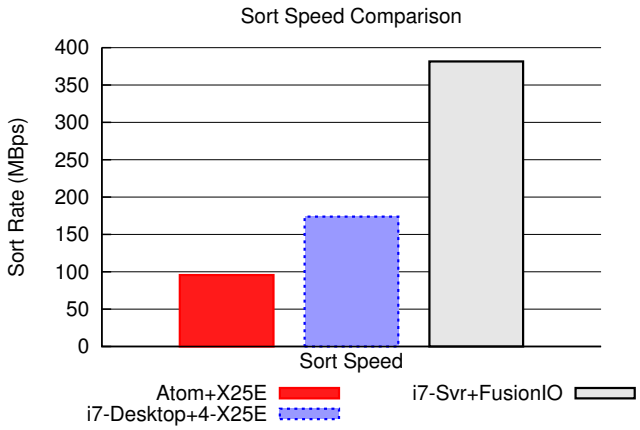

Figure 4: Sort speed on three modern architectures.

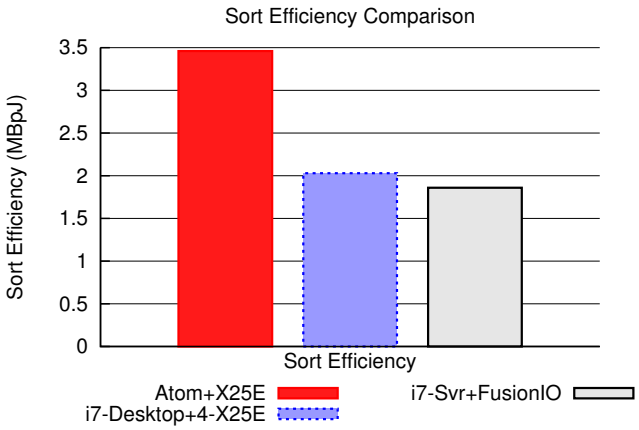

Figure 5: Sort efficiency; Atom - 28 W, i7-Desktop - 87 W, i7-server: 205 W.

the Atom system could only provide 10,760 queries per second because it was limited by the 100 Mbps Ethernet. Despite drawing only 22.3 W, its limited performance placed its energy efficiency in between the other two systems.

There are two interesting observations to be made about these results. First, we note that the 60,000 queries/sec that both the server and the desktop provided is below saturation of the storage devices: The Fusion-io can provide 100,000 4 KB random reads per second and each X25-E can theoretically provide 35,000 4 KB random reads based on filesystem benchmarking tools such as iozone [15] and fio [1]. Understanding this disparity is a topic of ongoing work. However, we note that when all values are retrieved from the filesystem buffer cache and avoid going to the device driver, the i7 systems can saturate a 1 Gbps network with requests, suggesting that the problem is specific to the I/O interface between our software and the flash devices–e.g., the onboard SATA controller on the i7-Desktop may be unable to handle a higher request rate, requiring that the X25-Es connect through enterprise-level external PCI-e HBA cards instead.

Some of the performance bottlenecks may be fixed through software optimization while others may be more fundamentally related to the required processing or hardware architecture of the individual systems. *None* of the modern systems above are perfectly balanced in their use of CPU, memory and I/O, so we cannot make a strong conclusion about which platform will eventually be the most energy-efficient once any software bottlenecks are removed. But the main takeaway is that the lower-power systems (Atom and Desktop i7) are currently significantly more energy-efficient than traditional server architectures, and understanding the bottlenecks of each system should inform the design of future energy-efficient and balanced platforms for persistent key-value storage.

3.3.2 Scan-bound Workloads

The next set of workloads that we examine are scan-bound workloads, which involve large sequential reads (instead of small random reads as in the seek-bound workloads described above). This section presents a sort scan-bound workload and energy efficiency on both traditional and FAWN nodes.

NSort: We evaluate sort using NSort [20], a commercial sort software product used by the authors of the JouleSort benchmark [23] to evaluate their hardware systems; NSort is currently the software used by the winner of the JouleSort competition. We tried to optimize the parameters to provide the highest performance by both specifying the optimal number of parallel sort threads for each architecture and trying to evenly distribute I/O load across multiple flash devices when appropriate. We follow the benchmark regarding the structure of the records to be sorted: 90-byte values with 10-byte keys. As one indicator that the tuning was at least modestly effective, the sort efficiency on *all three* platforms we tested is 1.7–4x higher than the 2009 JouleSort winner.

We present the results for a 10 GB sort (randomly permuted records) in Figure 4 and Figure 5. The Atom-based FAWN with two X25-Es can sort at 95 MB/sec while consuming 2760 Joules, 27.6 W averaged during the run, making it the most energy-efficient of all three architectures. The i7-Desktop system with 4 X25-Es could sort at nearly twice that speed, but consumed 87 W on average, so its efficiency was lower. Finally, the Core i7 server with a Fusion IO could sort at 381 MB/sec but consumed 205 W average during the run, so its efficiency closely matched the i7 Desktop system.

The i7 processors were not 100% utilized during the sort benchmark, suggesting that there should be capability to add more I/O to the system. However, we discovered other architectural limits in the process. For example, the DMI interface bandwidth on the i7-Desktop is 10 Gbps per direction, so placing additional X25-Es did not improve performance. This highlights the importance of developing balanced systems: excess processing capability is wasted if the hardware design is unable to saturate both the CPU and I/O simultaneously, reducing energy efficiency compared to more balanced systems. Newer i7 designs interface the CPU directly with the PCI-e bus, so PCI-e based flash devices may benefit from this better balance.

3.4 Memory/CPU-bound workloads

In the previous section, we discussed workloads whose working sets were large enough to require access to disks or flash, and that the computations on that data are simple enough to make the workload I/O-bound. In this section, we explore some worst-case workloads designed to be *more* energy-efficient on traditional, high-power, high-speed systems than low-power, low-speed systems.

3.4.1 Memory-bound

Workload description: We created a synthetic memory-bound benchmark that takes advantage of out-of-order execution and large caches. This benchmark repeatedly performs a matrix transpose multiplication, reading the matrix and vector data from memory and writing the result to memory. We chose matrix transpose specifically to have poor locality. The matrix data is in row-major format, which means that the transpose operation cannot sequentially stream data from memory. Each column of the matrix is physically separated in memory, requiring strided access and incurring more frequent cache evictions when the matrix does not fit entirely in cache.

The vector multiplications are data-independent to benefit from instruction reordering and pipelining, further biasing the workload in favor of modern high-speed, complex processors. We ran the benchmark with various input matrix sizes. We estimate the metric of performance, FLOPS (floating point operations per second) as the number of multiply operations performed, though we note that this workload is more memory-intensive than CPU-intensive.[2]

Evaluation hardware: In this experiment, we compared only the i7-Desktop to our Atom chipset; the i7-Server's large fixed costs make it less efficient than the i7-Desktop in all cases. The i7-Desktop operates 4 cores at a max of 2.8GHz, though we used the Linux CPU ondemand scheduler to choose the appropriate speed for each workload. The i7 860 has a 32 KB L1 cache and a 256 KB L2 cache *per core*, and also has an 8 MB L3 cache shared across all 4 cores. We enabled two-way Hyper-threading (Simultaneous Multi-Threading) so that the system exposed 8 "processors" to the operating system. Finally, we removed all but one X25-E and one 2 GB DRAM DIMM to further reduce power. At idle, the power consumed by the machine was 40 W and at full load would reach 130 W.

The Atom's processor cores each have a 24 KB L1 data cache and a 512 KB L2 cache. Two-way hyper-threading was enabled, exposing 4 "processors" to the OS. At idle, the Atom system consumed 18 W and at full load would reach 29 W.

Results: Figure 6 shows the energy efficiency (in KFLOPS/W) of our matrix multiply benchmark as a function of the size of the matrix being multiplied. When the matrix fits in the L1 data cache of both the i7-Desktop and the Atom, the Atom is roughly twice as efficient as the i7-Desktop. As the matrix size exceeds the L1 data cache, most memory accesses hit in L2 cache, and the efficiency drops by nearly a factor of two for both systems, with the Atom retaining higher energy efficiency.

The i7-Desktop's efficiency drops even further as the matrix size exhausts the 256 KB of L2 cache per core and accesses hit in L3.

[2]Comparing the FLOPS numbers here to those found in other CPU-intensive benchmarks such as in the Green500 competition will underestimate the actual computational capabilities of the platforms we measured, because this benchmark primarily measures memory I/O, not floating point operations.

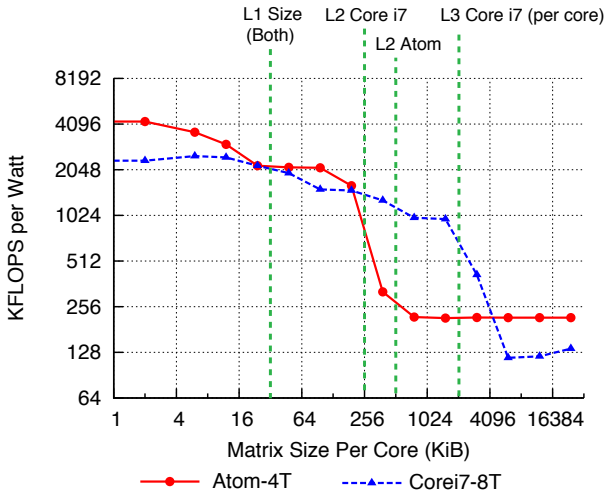

Figure 6: Efficiency vs. Matrix Size. Green vertical lines show cache sizes of each processor.

As the matrix size overflows the L2 cache on the Atom, most accesses then fall back to DRAM and efficiency remains flat thereafter. Meanwhile, the matrix size fits within the 8 MB L3 cache of the i7. Once the matrix grows large enough, most of its accesses then fall back to DRAM, and its energy efficiency drops below that of the Atom.

The main takeaway of this experiment is that when the working set fits in the same caches of each architecture, the Atom is up to twice as energy-efficient as the i7-Desktop. However, when the workload fits in the L2/L3 cache of the i7-Desktop but exhausts the Atom's on-die cache, the i7-Desktop is considerably more efficient, sometimes by a factor of four.

In other words, workloads that are cache-resident on a traditional system but not on a FAWN can be more efficient on the traditional system simply because of the amount of cache available on traditional systems.

The above experiment used OpenMP to run multiple threads simultaneously, eight threads on the i7-Desktop and four threads on the Atom. Running multiple threads is required to fully tax the CPU and memory systems of each node. We also ran the same experiment with one thread, to see how efficiency scales with load. Figure 7 shows that with one thread, the i7-Desktop is more efficient regardless of the size of the matrix.

This can be explained by *fixed power costs*. The i7-Desktop running one thread consumed 70 W (versus 40 W at idle), and the Atom running one thread consumed 20 W (versus 18 W at idle). The Atom platform we evaluated therefore has a large cost of not operating at full capacity. Its energy-proportionality is much worse than that of the i7-Desktop. Because the Atom was, at best, only twice as energy efficient as the i7-Desktop for this worst-case workload at 100% load, the inefficient chipset's power overhead domi-

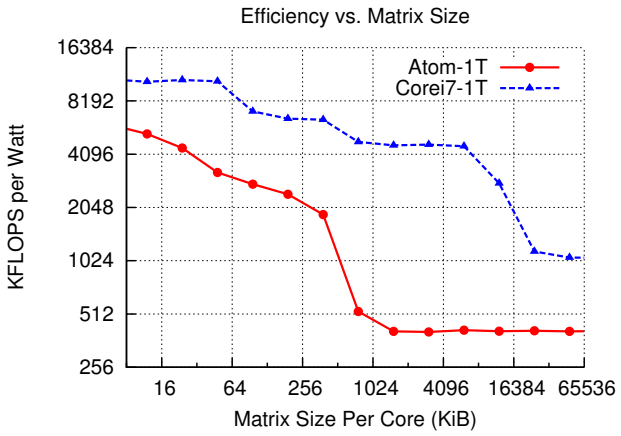

Figure 7: Efficiency vs. Matrix Size, Single Thread

Workload	i7-Desktop	Atom
SHA-1		
MB/s	360	107
Watts	75	19.1
MB/J	4.8	5.6
SHA-1 multi-process		
MB/s	1187	259
Watts	117	20.7
MB/J	10.1	12.51
RSA		
Sign/s	8748	1173.5
Verify/s	170248	21279.0
Watts	124	21.0
Sign/J	70.6	55.9
Verify/J	1373	1013

Table 2: Encryption Speed and Efficiency

nates the CPU power and reduces the energy efficiency at low-load significantly.[3]

3.4.2 *CPU-bound*

The matrix multiplication workload above requires frequent memory accesses per computation. Next, we look at a CPU-intensive task: cryptography. Table 2 shows several assembly-optimized OpenSSL speed benchmarks on the i7-Desktop and Atom systems described above. On SHA-1 workloads, we find that the Atom-based platform is slightly more efficient in terms of work done per Joule than the i7-Desktop architecture, and for RSA sign/verify, the reverse is true.

This flip in efficiency appears to be due to the optimization choices made in the assembly code versions of the algorithms. The OpenSSL "C" implementations of both SHA-1 and RSA are both more efficient on the Atom; we hypothesize that the asm version is tuned for high-performance CPUs. The SHA-1 assembly implementation, in contrast, was recently changed to use instructions that also work well on the Atom, and so its efficiency again exceeds that of the i7-Desktop. These results suggest that, first, CPU-bound operations can be as or more efficient on low-power processors, and second, they underscore that nothing comes for free: code must sometimes be tweaked, or even rewritten, to run well on these different architectures.

3.5 Limitations

FAWN and other low-power manycore cluster architectures may be unsuited for some datacenter workloads. These workloads can be broadly classified into two categories: latency-sensitive, non-parallelizable workloads and memory-hungry workloads.

[3]In the case of our particular system, many of the fixed energy costs are due to non-"server" components: the GPU and video display circuitry, extra USB ports, and so on. Some components, however, such as the Ethernet port, cannot be eliminated. These same factors preclude the use of extremely low-power CPUs, as discussed in Section 2.

3.5.1 *Latency-sensitive, non-parallelizable*

As mentioned previously, the FAWN approach of reducing speed for increased energy efficiency relies on the ability to parallelize workloads into smaller discrete chunks, using more nodes in parallel to meet performance goals; this is also known as the scale-out approach. Unfortunately, not all workloads in data-intensive computing are currently amenable to this type of parallelism.

Consider a workload that requires encrypting a 64 MB chunk of data within 1 second, and assume that a traditional node can optimally encrypt at 100 MB/sec and a wimpy node at 20 MB/sec. If the encryption cannot be parallelized, the wimpy node will not encrypt data fast enough to meet the strict deadline of 1 second, whereas the traditional node would succeed. Note that if the fastest system available was insufficient to meet a particular latency deadline, parallelizing the workload here would no longer be optional for either architecture. Thus, the move to many-core architectures (with individual core speed reaching a plateau) poses a similar challenge of requiring application parallelism.[4]

3.5.2 *Memory-hungry workloads*

Workloads that demand large amounts of memory per process are another difficult target for FAWN architectures. We examined a workload derived from a machine learning application that takes a massive-data approach to semi-supervised, automated learning of word classification. The problem reduces to counting the number of times each phrase, from a set of thousands to millions of phrases, occurs in a massive corpus of sentences extracted from the Web. Our results are promising but challenging. FAWN converts a formerly I/O-bound problem into a memory size-bound problem, which requires algorithmic and implementation attention to work well. The Alix3c2 wimpies can `grep` for a single pattern at 25 MB/sec, close to the maximum rate the CF can provide. How-

[4]Indeed, this challenge is apparent to the designers of next-generation cryptographic algorithms: Several of the entrants to the NIST SHA-3 secure hash competition include a hash-tree mode for fast, parallel cryptographic hashing. The need for parallel core algorithms continues to grow as multi- and many-core approaches find increased success. We believe this general need for parallel algorithms will help make the "wimpy" manycore approach even more feasible.

ever, searching for thousands or millions of phrases with the naive Aho-Corasick algorithm in `grep` requires building a DFA data structure that requires several gigabytes of memory. Although this structure fit in the memory of conventional architectures equipped with 8–16 GB of DRAM, it quickly exhausted the 256 MB of DRAM on each individual wimpy node.

To enable this search to function on a node with tight memory constraints, we optimized the search using a rolling hash function and large bloom filter to provide a one-sided error grep (false positive but no false negatives) that achieves roughly twice the energy efficiency (bytes per second per Watt) as a conventional node [19]. As an added benefit, this technique also increased the performance of the search on the conventional node by improving cache performance.

However, this improved efficiency came at the cost of considerable implementation effort. Our experience suggests that efficiently using wimpy nodes for some scan-based workloads will require the development of easy-to-use frameworks that provide common, heavily-optimized data reduction operations (e.g., grep, multi-word grep, etc.) as primitives. This represents an exciting avenue of future work: while speeding up hardware is difficult, programmers have long excelled at finding ways to optimize CPU-bound problems.

An interesting consequence of this optimization was that the same techniques to allow the problem to fit in DRAM on a wimpy node drastically improved cache performance on more conventional architectures: We were able to apply the techniques we developed to double the speed of virus scanning on desktop machines [7].

4. IMPLICATIONS AND OUTLOOK

In Section 2, we outlined several power scaling trends for modern computer systems. Our workload evaluation in the previous section suggested that these trends hold for CPU in real systems—and that, as a result, using slower ("wimpy") processors represents an opportunity to reduce the total power needed to solve a problem if that problem can be solved at a higher degree of parallelism.

In this section, we draw upon the memory scaling trends we discussed to present a vision for a future FAWN system: Individual "nodes" consisting of a single CPU chip with a modest number of relatively low-frequency cores, with a small amount of DRAM stacked on top of it, connected to a shared interconnect. This architecture is depicted in Figure 8. The reasons for such a choice are several:

Many, many cores. The first consequence of the scaling trends is clear: A future energy-efficient system for data-intensive workloads will have many, many cores, operating at quite modest frequencies. The limits of this architecture will be the degree to which algorithms can be parallelized (and/or load-balanced), and the static power draw imposed by CPU leakage currents and any hardware whose power draw does not decrease as the size and frequency of the cores decrease.

However, the move to many-core does not imply that individual chips must have modest capability. Indeed, both Intel and startups such as Tilera have demonstrated prototypes with 48–100 cores on a single chip. Such a design has the advantage of being able to cheaply interconnect cores on the same chip, but suffers from limited off-chip IO and memory bandwidth compared to the amount of CPU on chip.

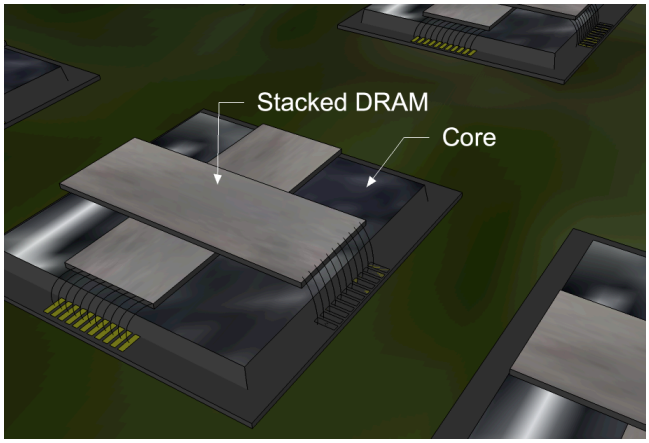

Figure 8: Future FAWN roadmap: Many-core, low-frequency chip with stacked DRAM per core.

Less memory, stacked. We chose a stacked DRAM approach because it provides three key advantages: Higher DRAM bandwidth, lower DRAM latency (perhaps half the latency of a traditional DIMM bus architecture) and lower DRAM power draw. The disadvantage is the limited amount of memory available per chip. Using the leading edge of today's DRAM technologies, an 8Gbit DRAM chip could be stacked on top of a small processor; 1GB of DRAM for a single or dual-core Atom is at the low end of an acceptable amount of memory for many workloads. From the matrix multiplication workload in the previous section, we expect that this decision will result in a similar efficiency "flip-flop": Workloads that fit in memory on a single wimpy node with 1GB of DRAM would run much more efficiently than they would on a comparable large node, but the wimpy would be less efficient for the range of problems that exceed 1GB but are small enough to fit in DRAM on a more conventional server.

However, the challenges posed by this architecture raise several issues:

Optimization back in vogue. Software efficiency was once a community focus: ekeing every last drop of performance or resource from a system was a laudable goal. With the rapid growth of data-intensive computing and a reliance on Moore's law, today's developers are less likely to optimize resource utilization, instead focusing on scalability at the detriment of node efficiency [3]. Instead, the focus has been on scalability, reliability, managability, and programmability of clusters. With a FAWN-like architecture, each node has fewer resources, making the job of the programmers harder. Our prior work has shown that the limited amount of memory per node has required the design of new algorithms [19] and careful balance of performance and memory footprint for in-memory hashtables [2]. These difficulties are compounded by the higher expected node count in FAWN architectures—not only does resource utilization become more important, these architectures will *further* stress scalability, reliability, and managability.

Heterogeneity. The existence of problems for which conventional server architectures still reign suggests that clusters must embrace heterogeneity in computing resources. Today's large-scale systems already must deal with heterogeneity because of arbitrary node failures and cluster purchasing schedules, but the existence of more energy-efficient, slower nodes will require that application and infrastructure software treat them as first-class resources with energy metrics playing a larger role in resource allocation decisions.

Metrics. We have so far evaluated energy efficiency in work done per Joule, which combines performance and power together as the only metrics. However, energy's impact on data-intensive computing is more broad—recent work has shown that platforms such as the Atom have other externalities, such as increased variability and latency, which affects service level agreements and other such quality of service metrics [21]. A focus of our ongoing work is to reduce these latencies and variability without microarchitectural redesigns, and also to devise metrics to properly capture and quantify these more difficult externalities.

5. RELATED WORK

FAWN follows in a long tradition of ensuring that systems are balanced in the presence of scaling challenges and of designing systems to cope with the performance challenges imposed by hardware architectures.

System Architectures: JouleSort [23] is a recent energy efficiency benchmark; its authors developed a SATA disk-based "balanced" system coupled with a low-power (34 W) CPU that significantly out-performed prior systems in terms of records sorted per joule. The results from this earlier work match our own in finding that a low-power CPU is easier to balance against I/O to achieve efficient sorting performance.

More recently, several projects have begun using low-power processors for datacenter workloads to reduce energy consumption [6, 18, 10, 26, 13, 17]. The Gordon [6] hardware architecture argues for pairing an array of flash chips and DRAM with low-power CPUs for low-power data intensive computing. A primary focus of their work is on developing a Flash Translation Layer suitable for pairing a single CPU with several raw flash chips. Simulations on general system traces indicate that this pairing can provide improved energy efficiency. CEMS [13], AmdahlBlades [26], and Microblades [17] also leverage low-cost, low-power commodity components as a building block for datacenter systems, similarly arguing that this architecture can provide the highest work done per dollar and work done per joule. Microsoft has recently begun exploring the use of a large cluster of low-power systems called Marlowe [18]. This work focuses on taking advantage of the very low-power sleep states provided by this chipset (between 2–4 W) to turn off machines and migrate workloads during idle periods and low utilization, initially targeting the Hotmail service. We believe these advantages would also translate well to FAWN, where a lull in the use of a FAWN cluster would provide the opportunity to significantly reduce average energy consumption in addition to the already-reduced peak energy consumption that FAWN provides. Dell recently begun shipping VIA Nano-based servers consuming 20–30 W each for large webhosting services [10].

Considerable prior work has examined ways to tackle the "memory wall." The Intelligent RAM (IRAM) project combined CPUs and memory into a single unit, with a particular focus on energy efficiency [5]. An IRAM-based CPU could use a quarter of the power of a conventional system to serve the same workload, reducing total system energy consumption to 40%. FAWN takes a thematically similar view—placing smaller processors very near flash—but with a significantly different realization. Notably, our vision for a future FAWN with stacked DRAM grows closer to the IRAM vision, though avoiding the embedded DRAM that plagued the IRAM implementation. Similar efforts, such as the Active Disk project [22], focused on harnessing computation close to disks. Schlosser et al. proposed obtaining similar benefits from coupling MEMS with CPUs [24].

Sleeping: A final set of research examines how and when to put machines to sleep. Broadly speaking, these approaches examine the CPU, the disk, and the entire machine. We believe that the FAWN approach compliments them well. Because of the data-intensive focus of FAWN, we focus on several schemes for sleeping disks: Hibernator [30], for instance, focuses on large but low-rate OLTP database workloads (a few hundred queries/sec). Ganesh et al. proposed using a log-structured filesystem so that a striping system could perfectly predict which disks must be awake for writing [11]. Finally, Pergamum [25] used nodes much like our wimpy nodes to attach to spun-down disks for archival storage purposes, noting that the wimpy nodes consume much less power when asleep. The system achieved low power, though its throughput was limited by the wimpy nodes' Ethernet.

6. CONCLUSION

This paper presented the computing trends that motivate our Fast Array of Wimpy Nodes (FAWN) architecture, focusing on the continually increasing CPU-Memory and CPU-I/O gap and the super-linear increase in power vs. single-component speed. Our evaluation of a variety of workloads, from worst-case seek-bound I/O workloads to pure CPU or memory benchmarks, suggests that overall, lower frequency nodes are substantially more energy efficient than more conventional high-performance CPUs. The exceptions lie in problems that cannot be parallelized or whose working set size cannot be split to fit in the cache or memory available to the smaller nodes. These trends point to a realistic, but difficult, path for energy efficient computing: Accepting tight constraints on per-node performance, cache, and memory capacity, together with using algorithms that scale to an order of magnitude more processing elements. While many data-intensive workloads may fit this model nearly out-of-the-box, others may require substantial algorithmic and implementation changes.

Acknowledgments

We would like to thank Ken Mai for his help with understanding memory trends. Amar Phanishayee contributed extensively to the design and implementation of the FANW-KV key value system used for several of the experiments described in Section 3.3. We would also like to thank Kanat Tangwongsan for his aid in developing the matrix multiply benchmark used in this work. Finally, we would like to thank Michael Kozuch for his extensive insights and feedback on our work. This work was supported in part by gifts from Network Appliance, Google, and Intel Corporation, and by grant CNS-0619525 from the National Science Foundation.

References

[1] Flexible I/O Tester. http://freshmeat.net/projects/fio/.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A fast array of wimpy nodes. In *Proc. 22nd ACM Symposium on Operating Systems Principles (SOSP)*, Big Sky, MT, Oct. 2009.

[3] E. Anderson and J. Tucek. Efficiency matters! In *Proc. HotStorage*, Big Sky, MT, Oct. 2009.

[4] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

[5] W. Bowman, N. Cardwell, C. Kozyrakis, C. Romer, and H. Wang. Evaluation of existing architectures in IRAM systems. In *Workshop on Mixing Logic and DRAM, 24th International Symposium on Computer Architecture*, June 1997.

[6] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, Mar. 2009.

[7] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen. SplitScreen: Enabling efficient, distributed malware detection. In *Proc. 7th USENIX NSDI*, San Jose, CA, Apr. 2010.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. 6th USENIX OSDI*, San Francisco, CA, Dec. 2004.

[9] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. 21st ACM Symposium on Operating Systems Principles (SOSP)*, Stevenson, WA, Oct. 2007.

[10] Dell XS11-VX8. Dell fortuna. http://www1.euro.dell.com/content/topics/topic.aspx/emea/corporate/pressoffice/2009/uk/en/2009_05_20_brk_000, 2009.

[11] L. Ganesh, H. Weatherspoon, M. Balakrishnan, and K. Birman. Optimizing power consumption in large scale storage systems. In *Proc. HotOS XI*, San Diego, CA, May 2007.

[12] K. Gray. Port deal with Google to create jobs. The Dalles Chronicle, http://www.gorgebusiness.com/2005/google.htm, Feb. 2005.

[13] J. Hamilton. Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for Internet scale services. http://mvdirona.com/jrh/TalksAndPapers/JamesHamilton_CEMS.pdf, 2009.

[14] Intel. Penryn Press Release. http://www.intel.com/pressroom/archive/releases/20070328fact.htm.

[15] Iozone. Filesystem Benchmark. http://www.iozone.org.

[16] R. H. Katz. Tech titans building boom. IEEE Spectrum, Feb. 2009.

[17] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *International Symposium on Computer Architecture (ISCA)*, Beijing, China, June 2008.

[18] Microsoft Marlowe. Peering into future of cloud computing. http://research.microsoft.com/en-us/news/features/ccf-022409.aspx, 2009.

[19] I. Moraru and D. G. Andersen. Fast cache for your text: Accelerating exact pattern matching with feed-forward bloom filters. Technical Report CMU-CS-09-159, Department of Computer Science, Carnegie Mellon University, Nov. 2009.

[20] C. Nyberg and C. Koester. Ordinal Technology - NSort home page. http://www.ordinal.com, 2007.

[21] V. J. Reddi, B. Lee, T. Chilimbi, and K. Vaid. Web search using small cores: Quantifying the price of efficiency. Technical Report MSR-TR-2009-105, Microsoft Research, Aug. 2009.

[22] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagle. Active disks for large-scale data processing. *IEEE Computer*, 34(6):68–74, June 2001.

[23] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A balanced energy-efficient benchmark. In *Proc. ACM SIGMOD*, Beijing, China, June 2007.

[24] S. W. Schlosser, J. L. Griffin, D. F. Nagle, and G. R. Ganger. Filling the memory access gap: A case for on-chip magnetic storage. Technical Report CMU-CS-99-174, Carnegie Mellon University, Nov. 1999.

[25] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *Proc. USENIX Conference on File and Storage Technologies*, San Jose, CA, Feb. 2008.

[26] A. Szalay, G. Bell, A. Terzis, A. White, and J. Vandenberg. Low power Amdahl blades for data intensive computing, 2009.

[27] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems – optimizing the ensemble. In *Proc. HotPower*, San Diego, CA, Dec. 2008.

[28] WattsUp. .NET Power Meter. http://wattsupmeters.com.

[29] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. 1st USENIX OSDI*, pages 13–23, Monterey, CA, Nov. 1994.

[30] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping disk arrays sleep through the winter. In *Proc. 20th ACM Symposium on Operating Systems Principles (SOSP)*, Brighton, UK, Oct. 2005.