SWEEP: Evaluating Computer System Energy Efficiency using Synthetic Workloads

Kristof Du Bois Tim Schaeps Stijn Polfliet Frederick Ryckbosch Lieven Eeckhout
ELIS Department, Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
{kristof.dubois, tim.schaeps, stijn.polfliet, frederick.ryckbosch, lieven.eeckhout}@UGent.be

ABSTRACT

Energy efficiency is a key design concern in contemporary processor and system design, in the embedded domain as well as in the enterprise domain. The focus on energy efficiency has led to a number of power benchmarking methods recently. For example, EEMBC released EnergyBench and SPEC released SPECpower to quantify a system's energy efficiency; also academics have proposed power benchmarks, such as JouleSort. A major limitation for each of these proposals is that they are tied to a specific benchmark, and hence, they provide limited insight with respect to why one system may be more energy-efficient than another.

This paper proposes SWEEP, Synthetic Workloads for Energy Efficiency and Performance evaluation, a framework for generating synthetic workloads with specific behavioral characteristics. We employ SWEEP to generate a wide range of synthetic workloads while varying the instruction mix, ILP, memory access patterns, and I/O-intensiveness; and we use SWEEP to evaluate the energy efficiency of commercial computer systems across the workload space and learn about how the energy efficiency of a computer system is tied to its workload's characteristics.

This paper also presents the Energy-Delay Diagram (EDD), a novel method for visualizing energy efficiency. The EDD clearly illustrates the energy versus performance trade-off, and provides more intuitive insight than the traditionally used EDP and ED^2P metrics.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: Modeling of computer architecture; C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*

General Terms

Performance, Measurement, Experimentation

Keywords

Energy-efficiency, workload characterization and generation

HiPEAC 2011 Heraklion, Crete, Greece.

1. INTRODUCTION

Energy efficiency has emerged as a primary design concern across the entire compute range, from low-end embedded systems to high-end servers and datacenters. Embedded systems are typically battery-operated and higher energy efficiency translates into greater user satisfaction through extended battery autonomy. Improving energy efficiency in servers and datacenters reduces the operational cost by reducing the electricity bill for powering the servers as well as for cooling them down. Moreover, there is an environmental concern as well. Improving the energy efficiency of computer systems is key to reduce carbon dioxide emissions by the IT industry.

Architects are well aware of the need for energy-efficient computer systems, and therefore, people have proposed benchmarks and benchmarking methodologies for evaluating energy efficiency, which should ultimately lead to more energy-efficient designs. The Embedded Microprocessor Benchmarking Consortium (EEMBC) released EnergyBench which provides data on the amount of energy a processor consumes while running a performance benchmark [6]. Recently, SPEC, the Standard Performance Evaluation Corporation, launched SPECpower_ssj2008, a benchmark for evaluating the power and performance characteristics of computer servers [12]. Rivoire et al. [16] propose JouleSort, a sort benchmark aimed at evaluating the energy efficiency of a wide range of computer systems from servers to embedded systems.

Although these approaches offer valuable insight in the energy efficiency of a computer system, they have limited flexibility. The benchmarks are rigid and cannot be altered to reflect different workload behaviors. In particular, EEMBC's EnergyBench is tied to the EEMBC performance benchmarks; the SPEC power benchmark is a Java server workload that generates and completes a mix of transactions; JouleSort implements a sort algorithm. These benchmarks are unable to explore the energy efficiency of computer systems across the workload space. In other words, the numbers produced by these approaches may be limited in scope — they are tied to these specific workloads — and it is hard to generalize towards other types of workloads, i.e., a computer system that is energy-efficient for the power benchmark does not necessarily imply that it is energy-efficient for other workloads.

This paper proposes SWEEP (Synthetic Workloads for Energy Efficiency and Performance evaluation), a framework for generating synthetic workloads with specific workload characteristics. SWEEP can generate compute-intensive workloads, memory-intensive workloads, I/O-intensive workloads, and any mix thereof. In particular, SWEEP enables its users to configure the workload's characteristics by setting the ratio of integer versus floating-point instructions, the inter-instruction dependencies, memory access patterns, disk I/O access patterns, etc. SWEEP provides a unique op-

portunity to its users: it allows for exploring the energy efficiency and performance of computer systems by 'sweeping' across the workload space. Using SWEEP we generate a range of synthetic workloads with very different characteristics and run these workloads on two real hardware systems, a low-end system (Intel Atom) as well as a high-end system (AMD Quad-Core Opteron), and evaluate their energy efficiency across different workload behaviors. A preliminary validation using the PARSEC benchmarks and a number of I/O-intensive applications reveals that SWEEP can generate workloads that exhibit a similar performance-energy trade-off as real applications.

We make the following contributions in this paper:

- We propose using synthetic workloads for evaluating the energy efficiency of computer systems. In contrast to current power benchmarking practice which uses specific benchmarks, this paper proposes a framework, called SWEEP, for generating synthetic workloads with workload characteristics of interest. SWEEP can generate synthetic benchmarks that are compute-intensive, memory-intensive, I/O-intensive, or any mix thereof. By varying the workload characteristics, the SWEEP end user can sweep across the workload space and gain insight in how energy efficiency and performance of a computer system relates to workload characteristics.

- We propose the Energy-Delay Diagram (EDD), a novel visualization method to summarize a computer system's energy consumption and performance relative to a reference machine. EDD clearly illustrates the trade-off in performance versus energy, and provides more insight than the traditional energy-delay-product (EDP) and energy-delay-square-product (ED^2P) metrics.

- Using a wide range of synthetic workloads with very different characteristics, we evaluate the energy efficiency of two real hardware systems, a low-end Intel Atom based machine and a high-end AMD Quad-Core Opteron system. We conclude that for I/O-intensive workloads the low-end machine tends to be more energy-efficient, i.e., it consumes much less energy while achieving similar performance; however, the opposite is true for compute-intensive workloads for which the high-end machine tends to be more energy-efficient: performance is much better and it consumes less or similar total energy. For memory-intensive workloads, there is a trade-off between both.

This paper is organized as follows. We first revisit prior work in Section 2. Section 3 presents the SWEEP framework and discusses how we generate a synthetic workload from an abstract workload model. Section 4 proposes the Energy-Delay Diagram for evaluating a computer system's energy efficiency. After detailing our experimental setup (Section 5), we then use the SWEEP framework to evaluate the energy efficiency of two real hardware platforms using EDDs in Section 6. We compare the energy efficiency characteristics of real-life applications and benchmarks against the synthetic workloads and we conclude that the synthetic workloads exhibit a performance versus energy trade-off that resembles the real workloads (Section 7). Finally, we conclude in Section 8.

2. PRIOR WORK

2.1 Power benchmarks

Given the growing importance of energy efficiency, interest has grown in power benchmarking methods. In the embedded domain for example, EEMBC has released EnergyBench [6], a method for reporting processor energy consumption when running embedded performance benchmarks.

For the server enterprise domain, SPEC recently released SPECpower_ssj2008 [12] which is a system-level, server-side Java workload that quantifies energy efficiency under varying loads. SPECpower generates and completes a mix of transactions and the reported throughput is the number of transactions completed per second over a fixed period of time; the workload considers 11 levels of load. Energy efficiency is quantified as the average number of transactions completed per unit of time per Watt.

Rivoire et al. [16] present JouleSort, a sort benchmark that reads its input from a file and writes its output to a file on a non-volatile device. There are three scale categories with 10GB, 100GB and 1TB records, and the benchmark aims at covering multiple domains, from embedded, to mobile, as well as to the server domain. The energy efficiency metric is the total energy consumed by the sort benchmark.

SWEEP is very different in its approach. SWEEP generates synthetic workloads with tunable workload characteristics, which allows for understanding the relationship between energy efficiency of a computer system with respect to workload behavior. The prior power benchmarking proposals are tied to specific benchmarks; SWEEP on the other hand, can generate a range of workload behaviors. Our results, which will be presented later in this paper, in fact indicate that whether one machine is more energy-efficient compared to another machine is closely tied to its workload: for one workload, system A may be more energy-efficient, whereas for another workload, system B may be more energy-efficient. SWEEP can also be used across multiple domains, from embedded to enterprise.

2.2 Synthetic benchmarks

Synthetic benchmarks such as Whetstone [4] and Dhrystone [18] are manually crafted benchmarks that aimed at representing real workloads. Manually building benchmarks though is both tedious and time-consuming. Whetstone and Dhrystone have become less relevant as they no longer represent current workloads.

Statistical simulation [5] collects program characteristics from a program execution and subsequently generates a synthetic trace from it which is then simulated on a simple, statistical trace-driven processor simulator. The important advantage of statistical simulation is that the dynamic instruction count of a synthetic trace is several orders of magnitude smaller than for today's industry-standard benchmarks, making it a useful simulation speedup technique for quickly identifying a region of interest in a large design space during the processor design cycle.

Recent work proposed automated synthetic benchmark generation [1, 8, 10] which builds on the statistical simulation approach but generates a synthetic benchmark rather than a synthetic trace, which allows for running the synthetic workload on an execution-driven simulator as well as on real hardware. Joshi et al. [11] take the idea of synthetic benchmark generation one step further and leverage the synthetic workload generation approach to generate stressmarks or power viruses. They use a genetic algorithm to search the workload space to identify those workload characteristics that maximize average power consumption, peak power consumption, temperature, dI/dt, etc.

This work in statistical simulation and synthetic workload generation has traditionally focused on CPU-intensive workloads, and does not include memory-intensive and/or I/O-intensive behavior. SWEEP on the other hand allows for generating synthetic I/O-intensive and memory-intensive workloads.

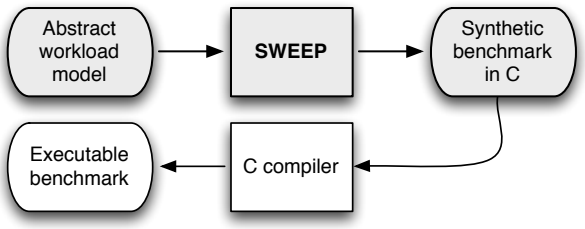

Figure 1: High-level view on the SWEEP framework.

Synthetic benchmarks have been developed to evaluate specific aspects of a computer system. For example, the STREAM benchmark seeks at quantifying a computer system's sustainable memory bandwidth using simple vector kernels [13]. IOzone [15] is a filesystem benchmark and generates a variety of file operations. Vasudevan et al. [17] use a set of microbenchmarks to evaluate the energy-efficiency of FAWN (Fast Array of Wimpy Nodes) computing clusters. Gamut, formerly called sstress [14], interleaves the execution of a compute-intensive loop with periods of idleness to match a target CPU utilization, and it also offers possibilities for generating memory-intensive and disk-intensive workloads. SWEEP can generate more diverse workload behaviors in a more flexible way than these specific synthetic (micro)benchmarks.

2.3 Energy efficiency metrics

Metrics are at the foundation of experimental research and development. Adequate metrics are absolutely crucial to steer research and development in the right direction. There exist a number of metrics for quantifying a computer system's energy efficiency. Two commonly used energy efficiency metrics are energy-delay product (EDP) and energy-delay-square product (ED^2P) [3, 7]. A major limitation of these metrics is that they combine energy consumption and performance in a single metric, which complicates understanding because in many cases there is a trade-off in performance versus energy, and these metrics may not always capture this trade-off in a comprehensive way, as we will discuss later in more detail. We instead propose EDD, the energy-delay diagram, which visualizes energy consumption versus performance in an insightful way.

Rivoire et al. [16] use total energy consumption as their energy efficiency metric. The winner is the system with the minimum total energy use. While this may be adequate for some workloads, e.g., batch-style background and throughput processes, it is not for performance-critical and latency-sensitive applications such as interactive applications, real-time applications, commercial applications (e.g., web servers, OLTP), etc. Energy usage by itself may be misleading as an energy-efficiency metric because it does not account for the energy versus performance trade-off. For example, a system that consumes marginally less energy than another system while yielding substantially less good performance is still considered the winner. The EDD instead captures the energy versus performance trade-off.

3. SWEEP

3.1 High-level overview

Figure 1 presents a high-level overview of the SWEEP framework. The end user specifies a set of desired workload characteristics in the abstract workload model from which the SWEEP framework then generates a synthetic workload. The abstract workload is specified in XML format which allows for easily configuring the synthetic workload. SWEEP's output is the synthetic workload, a C program, which is subsequently compiled and run on a simulator or on real hardware.

The concept of the SWEEP framework is such that the workload generator considers a number of building blocks, with each building block representing a different type of behavior. In particular, there is a building block to represent a linear sequence of code (a basic block), a loop, a thread, an access sequence to a data structure in memory, an access sequence to a data structure stored on disk. These building blocks can be configured at will in terms of their length, their characteristics (e.g., instruction mix, amount of ILP), memory reference locality, etc. For example, the basic block building block specifies the number of instructions, their types and inter-instruction dependencies; the loop building block specifies how many times the loop needs to be iterated; an access sequence to memory specifies the data structure that is to be traversed (array, linked list, tree) and how it is to be traversed.

The SWEEP framework is modular in the sense that it allows for combining these building blocks at will. This allows for building a synthetic workload of interest. For example, one could build a multi-threaded synthetic workload with extensive locking in order to evaluate a particular synchronization primitive. Or, one could build a workload with extensive I/O operations in order to evaluate a system's I/O performance. In this work, we will use the framework to synthesize workloads that are compute-intensive, memory-intensive or I/O-intensive for evaluating a computer system's energy efficiency across these three major classes of workload behaviors.

3.2 The SWEEP building blocks

There are five building block types in total, which we briefly discuss now.

3.2.1 Basic block

The 'basic block' building block represents a linear sequence of instructions and is an atomic unit of work. The basic block can be configured through a number of parameters, such as the number of instructions in the basic block, their types (integer or floating-point) and their inter-instruction dependencies. The latter determines the amount of instruction-level parallelism (ILP) in the program. The inter-instruction dependency distance is defined as the number of dynamically executed instructions between writing a data value and reading it. Hence, a large inter-instruction dependency distance implies high ILP, and a small dependency distance implies low ILP.

An additional parameter specifies the probability for the basic block to be executed. This is useful for generating conditional control flow in the synthetic workload (e.g., if-then-else statement).

3.2.2 Loop

The 'loop' building block specifies that the enclosed building blocks need to be iterated a number of times. The number of iterations is to be set by the SWEEP end user. The loop building block can include other loop building blocks which allows for building nested loops of any depth. Also, it can include basic blocks that the loop will iterate on, and if the basic blocks have conditional execution probabilities associated with it, then the generator will generate conditional control flow within the loop (e.g., if-then-else statements with a hard-to-predict branch within the loop).

3.2.3 Memory

The 'memory' building block specifies a memory-intensive program sequence. The main attribute specifies the data structure and its size that is to be accessed; there are three options: an array, a

linked list and a binary tree. Also, there are a number of possible access patterns. For the array, one can have a sequential, strided or random access pattern; for the linked list, the only access pattern is to sequentially traverse the linked list; for the tree data structure, the end user has the ability to select a breadth-first or depth-first access pattern. These access patterns can be either reads or writes, and are initiated within a loop.

3.2.4 Multi-threading

There are three building blocks related to multi-threaded execution. (1) The 'thread' building block initiates a thread in the synthetic workload. An attribute of the thread building block is whether the data structures accessed within the enclosed memory building blocks are private (access by the given thread only) or global (accessed by all threads). (2) The 'thread group' building block can be used inside the loop building block and allows for initiating parallel work done by threads that join (barrier synchronization) before proceeding to the next iteration. (3) The 'mutex' building block specifies that the enclosed building blocks are part of a critical section and thus need synchronization using locks.

3.2.5 Input/Output

Finally, the 'I/O' building block initiates reads and writes to a file stored on disk. There are three attributes: (1) the size of the file, (2) the access pattern (sequential, strided or random), and (3) whether the file is to be read or written. In order to fully stress the disk, there is an option to eliminate the buffering by the operating system and disk.

4. ENERGY-DELAY DIAGRAM

As mentioned in the introduction, we use the SWEEP framework to generate different flavors of workload behaviors in order to evaluate a computer system's energy efficiency. Now, quantifying energy efficiency is by itself a non-trivial issue. Traditionally, two metrics are being used for evaluating a computer system's energy efficiency, namely energy-delay product (EDP) and energy-delay-square product (ED^2P). EDP is defined as the total energy consumed to execute a unit of work multiplied by the execution time; ED^2P is defined as energy multiplied by the square of the execution time — hence, ED^2P puts more emphasis on performance than EDP. EDP and ED^2P are appealing because they quantify energy efficiency by a single number. However, evaluating a computer system's energy efficiency by a single metric may be misleading or at least it may complicate understanding the energy versus performance trade-off.

The Energy-Delay Diagram (EDD) visualizes the energy versus performance trade-off in an intuitive way, see Figure 2. The vertical axis shows the logarithm of the ratio of the energy consumption on the target machine relative to the reference machine:

$$y = \log_2\left(\frac{Energy_{target}}{Energy_{reference}}\right). \quad (1)$$

The horizontal axis shows the logarithm of the ratio of the execution time on the target machine relative to the reference machine:

$$x = \log_2\left(\frac{Time_{target}}{Time_{reference}}\right). \quad (2)$$

The origin of the EDD represents the reference machine. The first quadrant (I) represents cases in which the reference machine is more energy-efficient than the target machine, i.e., the reference machine consumes less energy and execution time is shorter. The third quadrant (III) represents the opposite situation: the target machine is more energy-efficient than the reference machine, i.e., the target machine consumes less energy and yields better performance. The second (II) and fourth (IV) quadrants represent trade-offs. For example, in quadrant II, the reference machine yields better performance at the cost of consuming more energy; in quadrant IV, we have the dual situation: the target machine yields better performance at the cost of consuming more energy. An important feature of the EDD is that, because it uses the logarithm of the energy and performance ratios, the EDP and ED^2P metrics can be visualized as straight lines in the EDD. The EDP line, which denotes points where the target and the reference machines are equally energy-efficient according to the EDP metric, is shown as the anti-bisector in Figure 2; the ED^2P line is shown as well.

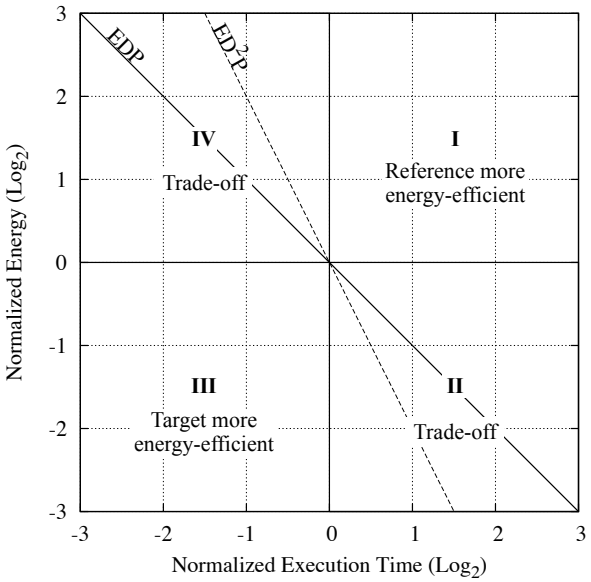

Figure 2: Energy-Delay Diagram.

The EDD visualizes the energy efficiency trade-off in an intuitive way. For example, a target system that is equally energy-efficient as the reference machine according to the EDP metric will appear on the anti-bisector. If the target system appears in quadrant II (on the EDP line), this means that the target system consumes less energy at the cost of a proportional loss in performance; if it appears in quadrant IV, this means that the target system consumes more energy at the benefit of a proportional performance gain. As another example, a target system appearing above the EDP line in quadrant II, implies that the reference system is more energy-efficient than the target system according to the EDP metric; however, the EDD shows that there is a trade-off: the reference system consumes less energy, but this comes at a performance hit (however, the performance hit is relatively small compared to the reduction in energy). In other words, the EDD clearly illustrates the trade-off in energy consumption versus performance.

Use case #1: Comparing machines for a fixed workload.

One possible use case for EDDs is to visualize the energy and performance trade-off of computer systems. For example, plotting different machines in the EDD enables a quick and intuitive competitor analysis in terms of the energy efficiency of computer systems for a given benchmark or a set of benchmarks. Figure 3 shows an illustrative EDD with four machines, A (the reference machine),

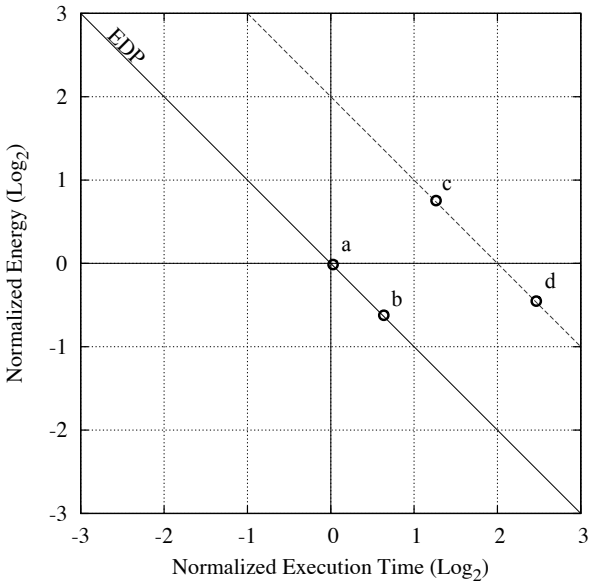

Figure 3: Comparing machines' energy efficiency using the EDD.

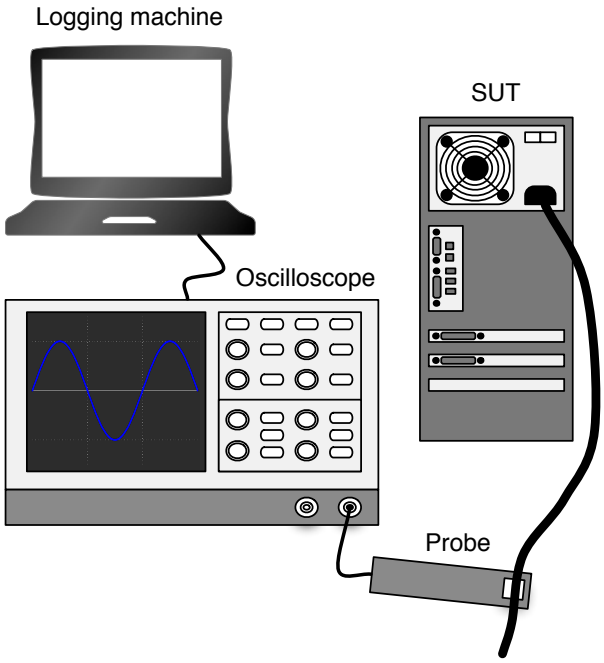

Figure 4: Runtime power monitoring setup.

B, C and D. Machine B achieves the same EDP as machine A as they lie on the anti-bisector EDP line. Also, C and D achieve the same EDP as they lie on a straight line parallel with the EDP line. This is a key feature of the EDD: design points that achieve the same EDP lie on a straight line parallel with the EDP line — this is a result of representing the logarithm of energy and execution time on the vertical and horizontal axes.

In this same example, machine C is less energy-efficient than both A and B: energy consumption is higher and performance is lower. Machine D on the other hand represents a trade-off relative to A: D consumes less energy than A at the cost of delivering worse performance.

Use case #2:
Comparing machines across workloads.

Another use case, which we will explore further in this paper, is to consider two computer systems (a reference and a target machine) and a range of workloads, and then provide data points for each of the workloads in the EDD. This enables exploring whether the energy efficiency of one system compared to another is subject to the workload. And given the SWEEP framework, this will enable us to explore how energy efficiency of a computer system relates to workload characteristics.

5. EXPERIMENTAL SETUP

Figure 4 illustrates our runtime power monitoring setup. The probe (Tektronix TCP 202) of an oscilloscope (Tektronix TDS 7104) is connected to the power cord of the System Under Test (SUT). The probe measures the current flowing through the power cord which enables measuring the total power consumed by the SUT. The oscilloscope is connected to a logging machine, which allows for post-processing the experiment data. This setup is similar to the one used by others [9].

We consider two SUTs in our experiments, a low-end Intel Atom machine and a high-end AMD Quad-Core Opteron server, see Table 1. The Intel Atom processor is a dual-core processor. Each core is an in-order SMT core with two thread contexts. The cache hierarchy is private to each core. The AMD Opteron is a quad-core processor. Each core is a superscalar out-of-order core (without SMT). The L1 and L2 caches are private and the L3 cache is shared among the cores. Both machines have a comparable 7200 rpm hard disk. The Thermal Design Power (TDP) is very different: the TDP for the Intel Atom is rated to be 8 Watt whereas the TDP for the AMD Opteron is rated to be 95 Watt.

6. REAL SYSTEM EVALUATION

We now exploit the unique property offered by SWEEP to 'sweep' the workload space and gain insight in how the energy efficiency of a computer system is affected by the characteristics of its workload. We systematically vary workload characteristics in the abstract workload, generate synthetic workloads, and run these synthetics on both of our SUTs. In the EDDs to follow, we consider the high-end AMD Opteron server as the reference machine. We organize the discussion along three major flavors of workload types: CPU-intensive, memory-intensive and I/O-intensive workloads.

6.1 CPU-intensive workloads

The first synthetic workload that we generate is a compute-intensive workload. It involves a limited number of memory accesses (and all memory accesses are cache hits), and performs no disk I/O. The workload consists of floating-point operations and the workload characteristic that we vary here is the inter-instruction dependency distance. An inter-instruction dependency distance of one means that an instruction is dependent on the instruction before it in the dynamic instruction stream. In other words, the synthetic workload involves a long chain of dependent instructions, and hence, there is no ILP. Increasing the inter-instruction dependency distance increases the opportunities for exploiting ILP and hence, performance improves.

Figure 5 shows the EDD for the CPU-intensive workloads for a varying inter-instruction dependency distance (see the legend). A

Low-end Intel Atom machine	
CPU	1.6GHz Intel Atom 330, two cores, two SMT threads per core 56KB private L1, 512KB private L2 TDP: 8 Watt
Memory	DDR2-800, 2GB
Disk	WD Scorpio Blue 7200 rpm
Power supply	Antec Trio 550 (85% efficiency)
High-end AMD Quad-Core Opteron	
CPU	2GHz Quad-core AMD Opteron 2350 Barcelona 128KB private L1, 512KB private L2, 2MB shared L3 TDP: 95 Watt
Memory	DDR2-667, 4GB
Disk	Samsung SATA 7200 rpm
Power supply	Antec EA 380D Green (80% efficiency)

Table 1: The Systems Under Test considered in this paper: a low-end and a high-end machine.

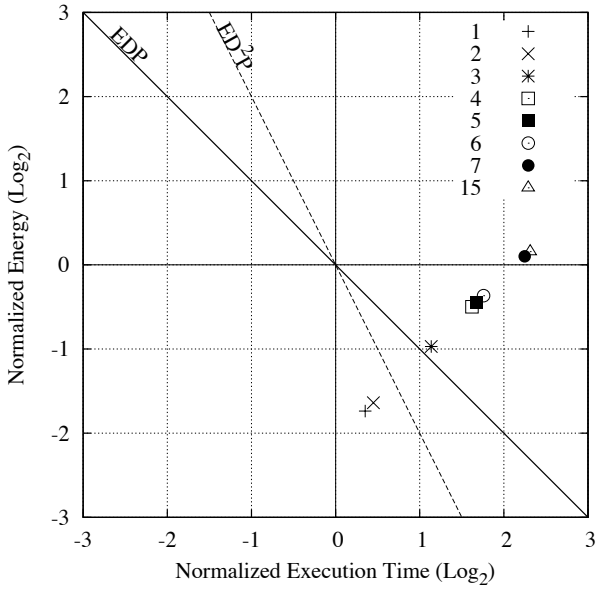

Figure 5: EDD for a CPU-intensive workload with varying inter-instruction dependency distance (see legend).

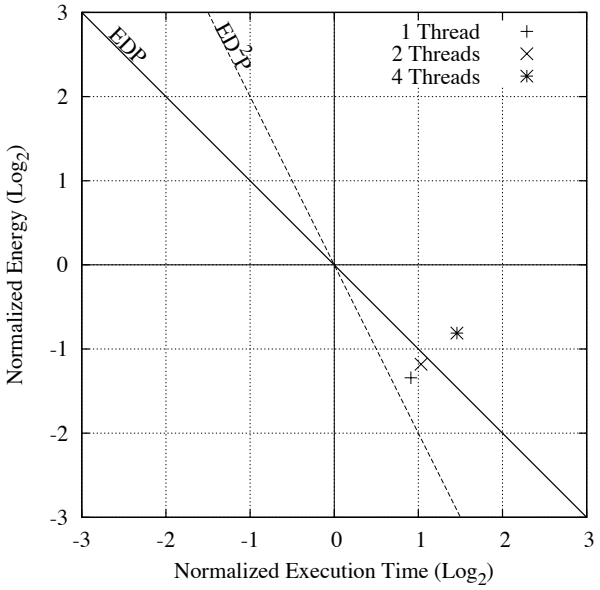

Figure 6: EDD for memory-intensive, multi-threaded workloads.

workload with no or limited ILP (i.e., a short inter-instruction dependency distance of 1 or 2) is more energy-efficiently run on the low-end machine than on the high-end machine, according to both the EDP and ED^{2}P metrics: the points corresponding to an inter-instruction dependency distance of 1 and 2 lie under the EDP and ED^{2}P lines. At higher degrees of ILP, the high-end server is more energy-efficient: the points lie above the EDP and ED^{2}P lines. And for high degrees of ILP (inter-instruction dependency distance of 7 and higher), the high-end machine clearly is the most energy-efficient machine: it consumes less total energy and execution time is shorter. This result can be explained by the fact that the high-end machine is a superscalar out-of-order processor which can better exploit the available ILP than the low-end in-order processor can. Clearly, for the high-end processor and workloads with high levels of ILP, the shorter execution time outweighs the higher power consumption of the processor, which ultimately leads to an overall reduction in the total amount of energy consumed. The interesting observation is that compute-intensive, high-ILP workloads are more energy-efficiently run on high-end processors, i.e., high-end processors yield better performance at lower total energy use.

6.2 Memory-intensive workloads

Our next experiment considers memory-intensive multi-threaded workloads. These workloads access a 150MB binary tree, and each thread accesses a private tree. All threads perform a breadth-first tree search and read all the values along the tree (over 85% of the instructions are loads). The IPC (for a single thread) on the high-end AMD Opteron processor is fairly low, namely 0.24. The reason is twofold: relatively high cache miss rates in the L2 and L3 caches, and low branch prediction accuracy. Figure 6 shows that both machines are comparable in terms of their energy-efficiency according to the EDP metric (for one thread and two threads). For four threads, the high-end quad-core processor is more energy-efficient compared to the low-end dual-core (two-way SMT per core) processor. The reason is the more aggressive memory hierarchy of the high-end processor (more on-chip cache space and more memory bandwidth) along with the fact that each thread on the high-end machine runs on a private core. On the other hand, the low-end ma-

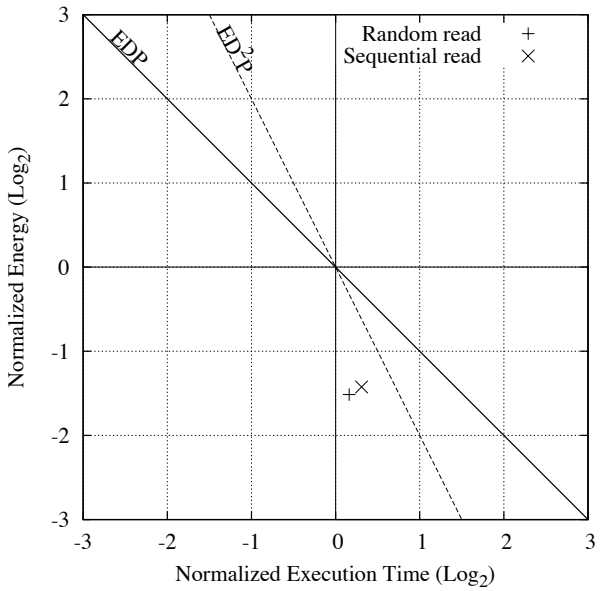

Figure 7: EDD for I/O-intensive workloads.

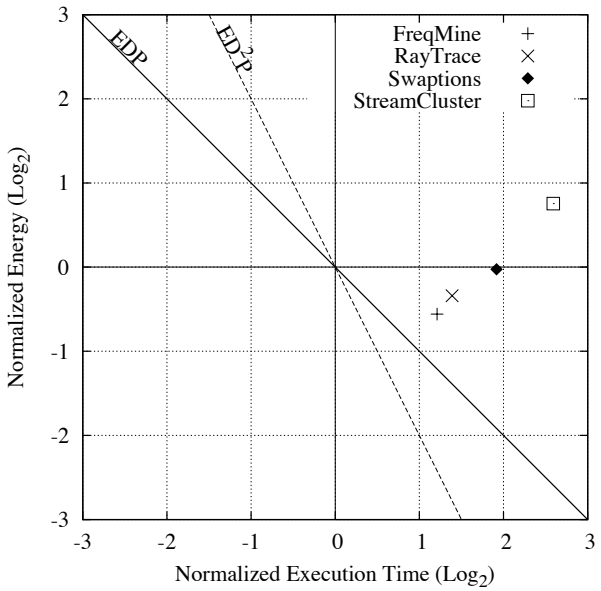

Figure 8: The EDD considering some of the PARSEC benchmarks.

chine's memory hierarchy is less aggressive, and two SMT threads per core share many of the resources. The memory system performance advantage of the high-end processor outweighs the additional energy consumed by the additional cores. The important observation here is that there is a trade-off in energy versus performance for memory-intensive workloads: the high-end processor yields better performance at the cost of consuming more energy; the low-end processor on the other hand consumes less total energy but performance is worse.

6.3 I/O-intensive workloads

For I/O-intensive workloads that read randomly or sequentially through a 12GB file, the low-end processor tends to be more energy-efficient than the high-end processor according to both the EDP and ED^{2}P metrics, see Figure 7. The low-end processor yields slightly less performance than the high-end processor, however, it consumes much less energy. The reason is that the processor is waiting for the disk to return while it is consuming power, and since the high-end processor is consuming more power than the low-end processor, the end result is that the low-end processor is more energy-efficient for this type of workloads. This also explains why the low-end processor is relatively more energy-efficient for the random read access pattern than for the sequential read pattern, i.e., the random access patterns introduces even more wait time for the processor than the sequential access pattern does.

7. REAL-LIFE APPLICATIONS

So far, we have considered synthetic workloads only. We now consider real applications (both benchmarks and GNU programs) and we evaluate whether the real applications lie in a region that is comparable to the region covered by the synthetics. In other words, we want to do some preliminary validation to gain confidence with respect to whether the synthetic workloads generate a performance versus energy trade-off that somehow relates to real application behavior. It is not our intent to validate that SWEEP can generate synthetic workloads that can serve as proxies for real-life applications, rather we want to evaluate whether the conclusions we obtained in the previous section using synthetics hold true when considering real applications.

Our first set of applications is taken from the multi-threaded PARSEC benchmark suite [2]. We consider four benchmarks, freqmine, raytrace, swaptions and streamcluster, see Figure 8; each benchmark runs four threads. The high-end machine is clearly more energy-efficient than the low-end machine for these benchmarks. Especially for streamcluster, the high-end machine yields better performance and consumes less energy; for swaptions, the high-end machine yields better performance at the same energy as the low-end machine. For the other benchmarks, freqmine and raytrace, there is a trade-off, however, the high-end machine is more energy-efficient according to both the EDP and ED^{2}P metrics. This result suggests that the PARSEC benchmarks are primarily CPU-intensive, exhibit substantial ILP and have limited memory requirements.

Our second set of applications comprises well-known GNU tools, namely tar and gzip. The tar tool creates an archive and is I/O-intensive: it reads a number of files and writes them in an archive, the tarfile. The second tool combines tar with gzip: it tars a number of files and then compresses it in a gzipped tarfile. We consider two compression levels here: 1 and 5 (5 means higher compression than 1). Figure 9 shows the EDD for the tar and gzip applications. Interestingly, the low-end machine is more energy-efficient for the tar workload, whereas the high-end machine is more energy-efficient for the tar+gzip workload. The reason for this difference is that the tar workload involves I/O operations almost exclusively, whereas the tar+gzip workload also involves substantial CPU-intensive operations during compression. This is further explained by the observation that the high-end machine is even more energy-efficient for gzip's CPU-intensive compression level 5 than for compression level 1.

8. CONCLUSION

This paper proposed SWEEP, a framework for generating synthetic workloads with specific behavioral characteristics. SWEEP can generate compute-intensive, memory-intensive and I/O-intensive workloads, and any mix thereof. SWEEP enables novel capabili-

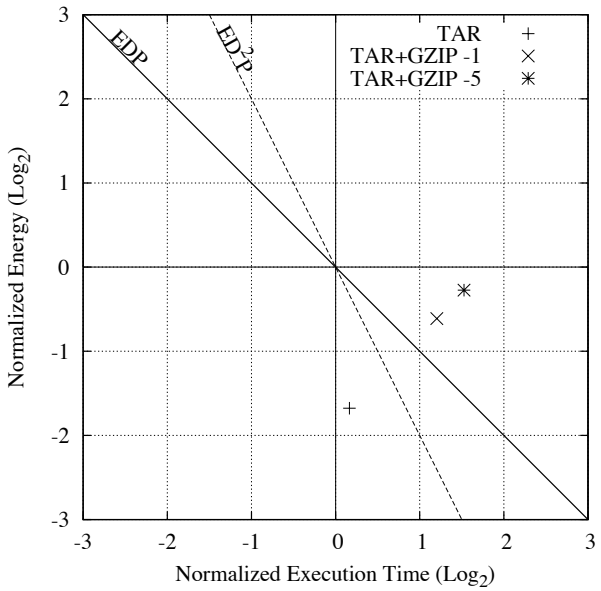

Figure 9: The EDD considering the tar and gzip Linux tools.

ties to study a computer system's energy efficiency. Whereas prior work in power benchmarking is tied to specific benchmarks, such as EnergyBench, SPECpower and JouleSort, SWEEP enables sweeping the workload space and study how energy efficiency is tied to the workload characteristics. We conclude that whether one machine is more energy-efficient than another machine is very much workload dependent. SWEEP is a useful tool to explore these trade-offs.

This paper also presented the Energy-Delay Diagram (EDD), a novel way of visualizing a machine's energy efficiency relative to a reference machine. The EDD represents the trade-off in performance versus energy in a more intuitive way than the traditionally used EDP and ED^2P metrics do.

We believe this paper points towards an interesting avenue of future work. The observation that some workloads are more energy-efficiently run on one machine whereas other workloads are more energy-efficiently run on another machine, suggests that heterogeneous datacenters may be an energy-efficient solution. In a heterogeneous datacenter, workloads would be steered dynamically towards the most energy-efficient server. Given the trend towards cloud computing which suggests many different workloads running in consolidated environments, there may be opportunities for exploiting workload diversity in the datacenter for improving overall energy efficiency and decreasing (operational) cost.

9. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their constructive and insightful feedback. Frederick Ryckbosch is supported through a doctoral fellowship by the Research Foundation–Flanders (FWO). Stijn Polfliet is supported through a doctoral fellowship by the Agency for Innovation by Science and Technology (IWT). Additional support is provided by the FWO projects G.0232.06, G.0255.08, and G.0179.10, and the UGent-BOF projects 01J14407 and 01Z04109.

10. REFERENCES

[1] R. Bell, Jr. and L. K. John. Improved automatic testcase synthesis for performance model validation. In *Proceedings of the 19th ACM International Conference on Supercomputing (ICS)*, pages 111–120, June 2005.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81, Oct. 2008.

[3] D. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, November/December 2000.

[4] H. J. Curnow and B. A. Wichmann. A synthetic benchmark. *The Computer Journal*, 19(1):43–49, 1976.

[5] L. Eeckhout, S. Nussbaum, J. E. Smith, and K. De Bosschere. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro*, 23(5):26–38, Sept/Oct 2003.

[6] Embedded Microprocessor Benchmark Consortium (EEMBC). Energybench v1.0 power/energy benchmarks. http://www.eembc.org/benchmark/power_sl.php.

[7] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sept. 1996.

[8] C. Hsieh and M. Pedram. Micro-processor power estimation using profile-driven program synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1080–1089, Nov. 1998.

[9] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO)*, pages 93–104, Dec. 2003.

[10] A. M. Joshi, L. Eeckhout, R. Bell, Jr., and L. K. John. Distilling the essence of proprietary workloads into miniature benchmarks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 5(2), Aug. 2008.

[11] A. M. Joshi, L. Eeckhout, L. K. John, and C. Isen. Automated microprocessor stressmark generation. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 229–239, Feb. 2008.

[12] K.-D. Lange. Identifying shades of green: The SPECpower benchmarks. *IEEE Computer*, 42(3):95–97, Mar. 2009.

[13] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream/. University of Virginia.

[14] J. Moore, J. Chase, K. Farkas, and P. Ranganathan. Data center workload monitoring, analysis and emulation. In *Proceedings of the Eighth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW), held in conjunction with HPCA*, Feb. 2005.

[15] W. D. Norcott. IOzone filesystem benchmark. http://www.iozone.org/.

[16] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A balanced energy-efficiency benchmark. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 365–374, June 2007.

[17] V. Vasudevan, D. Andersen, M. Kaminsky, L. Tan, J. Franklin, and I. Moraru. Energy-efficient cluster computing with FAWN: Workloads and implications. In *Proceedings of the First International Conference on Energy-Efficient Computing and Networking (e-Energy)*, Apr. 2010.

[18] R. P. Weicker. Dhrystone: A synthetic systems programming benchmark. *Communications of the ACM*, 27(10):1013–1030, Oct. 1984.