©IEEE 2010. This is the author's version of the work. It is posted here for your personal use. Not for redistribution.
The definitive Version of Record was published in: Proceedings of the 1st International Green Computing Conference (IGCC'10), http://dx.doi.org/10.1109/GREENCOMP.2010.5598316.

Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86-64 Processors

Daniel Molka, Daniel Hackenberg, Robert Schöne and Matthias S. Müller
Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden – 01062 Dresden, Germany
Email: {daniel.molka, daniel.hackenberg, robert.schoene, matthias.mueller}@tu-dresden.de

***Abstract*—The energy efficiency of computer systems is influenced by many interdependent aspects. To asses the efficiency, typical benchmarks characterized the total power consumption of a computer system under certain domain specific workloads. For example, in case of the SPECPower benchmark the workload is a typical web server specific Java application. The contribution of individual components is usually not considered in this class of benchmarks. The CPU makes the most significant contribution due to both its high peak power consumption and the high variability depending on the workload. Correlations of workload and energy consumption of parts of the processors are usually done with simulations rather than actual measurements. This is mainly a consequence of the limited time resolution of power meters that is usually orders of magnitude too low to observe variations in the time scale of microarchitectural events. Furthermore, it is usually not possible to solely measure power consumption of processors as they are supplied by multiple power lines that are not easily accessible and are often shared with other components. In this paper we present benchmarks and a measurement methodology that compensate for the time resolution of our power meter by applying a constant and well-defined workload to the system. Using this experimental setup we analyze x86-64 microarchitectures from AMD and Intel. We furthermore characterize the contribution of individual operations and data transfers to the total power consumption of the Intel system.**

***Keywords*-energy efficiency; power consumption; data transfers; caches;**

I. Introduction

While the instruction set of x86 processors is well documented, details about the microarchitectural implementation are hardly available. The different building blocks such as cache levels with their individual size and organization as well as processor pipelines, arithmetic logic units (ALU), and floating point units (FPU) are generally known. However, there is usually no information available about the contribution of individual units or typical operations to the power consumption of the processor.

©2010 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

In this paper we present an approach to isolate the energy consumption characteristics of certain basic operations. We show how data transfers and arithmetic operations contribute to the total energy consumption. This is achieved by using workloads that stress only selected processor parts. Furthermore, we control the location of the used data and thereby investigate how expensive data transfers from different cache levels or main memory are. Unlike other approaches this is not limited to the cost in relation to the used time but also to the consumed energy for moving data from a memory location to a register. The microbenchmarks we are using compensate the time resolution of our power meters which is several orders of magnitude lower than is required to observe effects of single instructions. They apply a constant load to certain parts of the processor for sufficiently long runtimes that can be captured with our power meter.

The data gathered for the energy consumption of our workloads can for example be used to compare the energy efficiency of different microarchitectures for certain operations. Re-evaluating the reasonableness of microarchitectural features like simultaneous multithreading (HyperThreading) for certain workloads is conceivable as well, not only with respect to performance implications, but also with respect to energy efficiency. Another possible use case is to create a model for the energy consumption of processors that could for example be used to estimate power consumption without measurement. It may also be possible to help software architects to decide whether it is more efficient to transfer previously calculated results from a certain memory location into the processor core or to recalculate the result.

This information is also crucial to develop well-founded propositions for creating more energy efficient applications and systems. A compiler for example could choose those instructions that consume as little energy as possible, if multiple options are available to correctly execute the given algorithm. A system setup could be optimized in a way that the used programs run at full speed but with some parts of the processor, which would use additional power, deactivated or throttled. Another option would be the avoidance of speculative execution of instructions or the deactivation of power inefficient processor features like prefetching.

The paper is structured as follows: In Section II we introduce our AMD and Intel based test systems, describe our power measurement infrastructure and methodology, and present the benchmarks we use. Section III shows the results for the energy consumption of certain workloads on both systems. In Section IV we use these results to create a model of the power consumption for the Intel system. Section V discusses related work and Section VI concludes this paper.

II. Experimental Setup

A. Test Systems

In this work we examine two dual socket systems with six-core processors from AMD and Intel. The AMD test system is a Sun Fire X4140 with two Opteron 2435 processors (Istanbul microarchitecture). The Intel test system is a software evaluation system equipped with two Xeon X5670 processors (Westmere-EP microarchitecture). Figure 1 shows the system architecture of both test systems, which is very similar. The six cores of each processor have private L1 and L2 caches, whereas the L3 cache is shared by all cores. Point-to-Point interconnects are used for communication between the processors. The memory controllers are integrated into the processors. However, the memory bandwidth of the two systems differs significantly as the Opteron system uses DDR2 memory, whereas the Xeon system supports DDR3 memory.

Both our test systems are equipped with redundant power supplies. However, we removed one power supply of each system during our measurements in order to avoid the additional power consumption of a second power supply. Table I lists further details concerning processor properties and other system components. Because of the many differences between the two systems, the total power consumption should not be compared.

Due to stability problems we limited the QPI frequency of our pre-production Intel machine to 4.8 GT/s. The reduced bandwidth between the cores is not relevant for our measurements as threads are pinned to certain cores and all data is allocated at the local NUMA node. Based on the results of our bandwidth measurements the reduced QPI clock only slightly affects the available bandwidth to local memory. Aside from that both systems use default BIOS settings. This includes enabled HyperThreading on the Intel system and disabled HyperTransport Assist on the AMD system as well as default prefetcher settings on both systems.

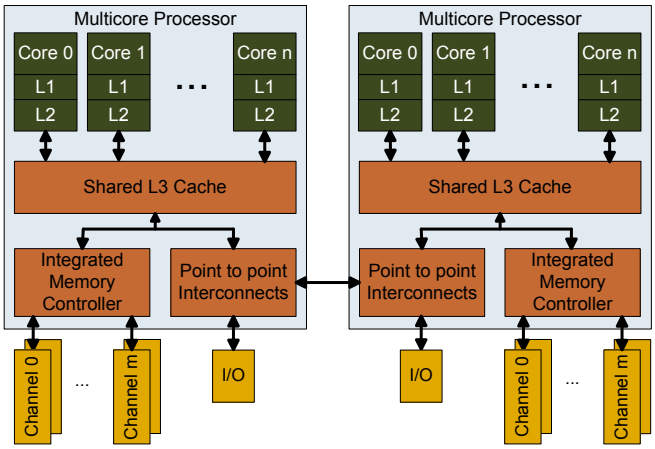


Figure 1. Test system overview

Table I
Hardware Configuration and Processor Details

Vendor	Intel	Sun
System	Software Evaluation System	Sun Fire X4140
Operating system	Linux 2.6.26	Linux 2.6.33
Processor	2x Intel Xeon X5670	2x AMD Opteron 2435
Core clock	2.93 GHz	2.6 GHz
Uncore/NB clock	2.66 GHz	2.2 GHz
TDP	95 W	115 W (75 W ACP)
Interconnect	QPI 4.8 GT/s (19.2 GB/s)	coherent HT3 2.2 GHz (17.6 GB/s)
Codename	Westmere-EP	Istanbul
Technology	32 nm HKMG	45 nm SOI
SSE extensions	SSE, SSE2, SSE3, SSSE3,SSE4.1,SSE4.2	SSE4A
L1 cache	2x 32 KiB per core	2x 64 KiB per core
L2 cache	256 KiB per core	512 KiB per core
L3 cache	12 MiB per chip	6 MiB per chip
IMC channels	3x RDDR3	2x RDDR2
Memory type	PC3-10600R	PC2-5300R
Memory size	12 GiB (6x 2 GiB)	16 GiB (8x 2 GiB)
Chipset	Intel 5520	nVidia MCP55
Disk	500 GB 3.5" SATA 7200 rpm	2x 147 GB, RAID 0 2.5" SAS 10k rpm
Power supply	Ablecom PWS-801P-1R 885W	Delta Energy Systems A221 658W

We run Linux on both test systems and use the sysfs interface of cpufreq to control power management and ensure a fixed clock rate. The userspace governor is activated to obtain manual frequency control and deliberately disable software guided dynamic frequency scaling. Before starting any benchmark a fixed frequency is enforced by selecting the lowest P-state (P0) on the AMD system and the second lowest P-state (P1) on the Intel system. This effectively disables the Intel Turbo Boost technology and therefore ensures that our measurements are not influenced by hardware-controlled dynamic frequency enhancements of individual cores.

B. Power Measurement Methodology

Figure 2 shows the design of our measurement infrastructure. Attached to the test systems is a ZES LMG450 power meter that records the power consumption of both systems at a rate of 10 Hz. A power measurement daemon on a separate computer, the *Collector*, is used to configure the power

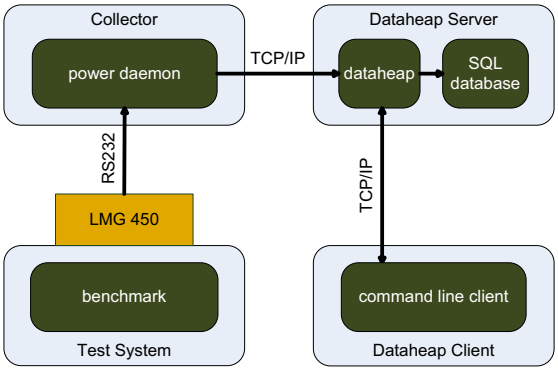


Figure 2. System setup for power measurements using the Dataheap

meter and record the power consumption. The Collector forwards the recorded power values to the so called *Dataheap* Server which stores all measurement data along with their individual timestamps for later analysis. A command line client is used to query the average power consumption for a certain interval from the Dataheap. This client may for example run on the analysts desktop computer to perform an (automated) post mortem merging of benchmark results and power measurement data.

Among the advantages of this setup is that there is no need for a software modification of the workload that runs on the test system besides providing start and end timestamps that are used to retrieve the corresponding power consumption from the Dataheap. Our setup only requires a tight (ntp) time synchronization of the test system and the Collector host. Most importantly, the power measurement does not affect the runtime behavior of the workload on the test system. Moreover, unlike for example the power measurement tool-chain of SPECPower [6], our approach does not require a special network setup to allow the test system to communicate with other hosts that are involved in the power measurement process. The absence of network communication between test system and measurement host makes the process of application specific energy efficiency benchmarking less error-prone.

Our measurement approach does not require any hardware modification of the test system. On systems with redundant power supplies, the power meter can easily be planted without interrupting the operation of the test system. However, the power consumption of the system fans as well as the processor itself depends on the temperature [7] and our coarse-grained approach does not allow to inhibit these effects. We therefore ensure that the power consumption has reached a stable value by running each benchmark for 15 minutes. The power consumption is then determined as the average of a one minute interval starting 90 seconds before the end of each individual benchmark. Our power meter samples at a rate of 10 Hz and we therefore have 600 samples that need to be within a small range. Measurements that include any runaway values within the 60 second interval are repeated in order to eliminate random errors that may be generated by the power meter or by peripherals like network interfaces or hard disks.

We use the benchmarks described in Section II-C to generate specific workloads that allow us to measure the energy consumption of arithmetic operations and data transfers. The benchmarks can be configured to selectively stress certain units of the processor and record the bandwidth of the data transfers during the benchmark as a measure of throughput. The units that can be examined are the processor's ALUs and FPUs as well as the different cache levels and main memory. The contribution of the workload to the energy consumption of the CPU is determined by using the estimated baseline power consumption that we describe in Section II-D. The baseline power compensates for the power consumption of other system components as well as the power required to have the processors in active state (i.e. C0, not power or clock gated). Only the additional power consumed when actually executing instructions is attributed to the respective execution unit.

C. Workloads

In order to characterize certain processor parts with respect to their power consumption we need to apply a constant workload that generates a specific processor load. On today's general purpose processors and their instruction sets it is not possible to stress single units without also using other processor parts. However, it is possible to create different workloads that generate varying pressure for some units while keeping the utilization of all other units constant. This can be done for the ALUs and FPUs by substituting the performed operation on the same chunk of data. Data transfers can be examined by performing the same operation on data from different locations.

The data transfer benchmarks are based on existing open source memory bandwidth benchmarks[9]. They allow to measure the available bandwidth for each cache level and main memory by carefully placing data in the desired location. In principle they work as follows:

```
1) start one thread on each core
2) for all data set sizes
3)  all threads: access data sequentially
4)  all threads: synchronize
5)  all threads: measure t_begin
6)  all threads: access data sequentially
7)  all threads: measure t_end
8)  duration = max(t_end) - min(t_begin)
```

Each thread is restricted to a certain core (line 1) using the `sched_setaffinity()` function in order to avoid thread migration. Furthermore, `numactl --localalloc` is used to start the benchmarks in order to ensure that all threads work on physical memory attached to their home node. Memory for the data is allocated using

2 MiB pages in order to reduce *TLB* misses. The bandwidth is measured for a configurable number of data set sizes (line 2). The first access to the buffer (line 3) is used to bring data into the desired cache level. Thread synchronization, time measurement, and the whole data access (lines 4-7) are implemented in assembly language to rule out any compiler influence and to achieve precise time measurements. We use the `movdqa` instruction to transfer 16 Bytes from memory into an 128 Bit SSE register. The read bandwidth is then derived from the data set size and the measured duration.

For data set sizes that fit into the L3 caches of our test systems the sequential buffer access has a runtime of around 150 microseconds. This is several orders of magnitude below the resolution of our power meters, not to mention our target runtime of 15 minutes. Simply repeating the whole measurement for a certain data set size has the disadvantage that the outer loop iterations interrupt the otherwise constant workload. Increasing the data set size is also not an option as we would only be able to measure main memory accesses in this case. Therefore, the runtime of the measurement routine itself (line 6) has to be increased. This can be achieved by repeatedly accessing the same data. The *least recently used* (LRU) replacement strategy of the caches ensures that data is already evicted back to the original cache level when it is accessed again. However, this "ring buffer" style memory access destroys the coherency state control of the original benchmarks. Therefore, corner cases like writing to exclusive cache lines can not be investigated with the modified benchmarks. After reaching the runtime limit of 15 minutes the benchmark will terminate itself after receiving a SIGTERM signal from an additional thread that sleeps until the time limit expires.

In order to investigate how arithmetic operations contribute to the energy consumption of the CPU we have implemented additional measurement routines. In these routines, the `movdqa` instruction that was originally used for loading data is replaced by instructions that additionally perform arithmetic operations. All these instructions use a two address format and we use them with one memory operand and one register operand. This combines the original load operation and an arithmetic operation without significantly changing the code size. Integer operations as well as single and double precision floating point operations are used to generate varying load for the ALUs and FPUs while keeping the utilization of other units constant. We use packed SSE instructions that perform multiple 32 or 64 Bit operations on each 16 Byte element and also consider alternative load instructions such as `movaps` and `movapd`. Table II lists the names and properties of all the benchmarks.

The three `load` variants move data from a cache or main memory location into registers but perform no operation on it. The `add`, `and`, and `mul` benchmarks perform one operation on each operand. Single precision floating point instructions execute four 32 Bit operations per 16 Byte, integer and double precision floating point instructions perform two 64 Bit operations per 16 Byte. The `m+a_pd` consists of a multiply instruction using a memory operand and a subsequent addition using the result from the register. Thus four 64 Bit floating point operations are performed per 16 Byte. The results of the calculations are not stored back to memory. Therefore, the difference in power consumption between e.g. the `movapd` and the `mulpd` workload can be solely attributed to the double precision floating point multiplication.

Table II
BENCHMARK NOMENCLATURE

notation	instruction	data type / operation
load_pi	movdqa	packed integer load
load_ps	movaps	packed single load
load_pd	movapd	packed double load
and_pi	pand	packed integer and
and_pd	andpd	packed double and
add_pi	paddq	packed integer add
add_ps	addps	packed single add
add_pd	addpd	packed double add
mul_pi	pmuldq	packed integer mul
mul_ps	mulps	packed single mul
mul_pd	mulpd	packed double mul
m+a_pd	mulpd+addpd	packed double sequent mul and add

On Istanbul, floating point SSE multiplication and addition each can only be performed by one of the SIMD units, which is not enough to fully utilize the two 128 Bit loads per cycle that the L1 can handle. The throughput of these operations therefore differs by a factor of two compared to the load operations that fully utilize the L1 interface. This has to be taken into account for the power consumption comparison of the different workloads. Moreover, the `mul_pi` benchmark does not work on the Opteron test system as it does not support SSE4.1.

Special care has to be taken to avoid overflows as the same registers are used repeatedly. This is not an issue for logic operations. For integer operations it can be ignored as execution will continue normally after overflows. However, for floating point addition and multiplication the occurrence of the value *NAN* (not a number) has to be avoided. We achieve this by carefully initializing the buffer with alternating $value$ and $-value$ in case of addition or $value$ and $value^{-1}$ for the multiplication benchmark. Registers are initialized with $value$. This results in the registers altering between $value$ and $value + value$ or $value * value$. We thereby avoid the values 0.0 and 1.0 in the registers at any time as well.

D. Baseline Power Consumption

In Section IV we want to use our results to determine the energy consumption of certain operations, for example memory accesses (measured e.g. in picojoule per byte). We obviously cannot attribute the power consumption of the

whole system to these small operations. Instead we have to specify a *baseline* power consumption. This baseline is defined in our sense as the lowest possible power consumption of the whole system with all processor cores being in C0 state (operating state, CPU fully turned on). This specifically forbids to use the idle power as baseline, as the operating system sends processor cores into deep sleep states during idle phases. On the other hand, deactivating the C-states in the system BIOS lets the operating system run a certain idle loop during phases of inactivity. The power consumption of that loop may not be the lowest achievable baseline.

One way to approximate the baseline power is to implement idle loops that do not perform any data movement and arithmetic operations. We tested empty loops, loops containing only `nop` instructions, loops containing only `pause` instructions, and a loop that uses the `rep` prefix to repeatedly execute the `nop` instruction. The `nop`, `pause`, and `rep nop` loops were measured for different numbers of instructions within a loop (different start values for register ECX in case of `rep nop`). Different jump instructions at the end of the loops were tested as well (unconditional `jmp` and conditional `jne`). OpenMP is used to execute the loops simultaneously on all cores using GOMP_CPU_AFFINITY to avoid unnecessary thread migrations.

Our second approach to estimate the baseline power consumption is to use instructions with a high latency that stall the processor while executing. The `sqrtpd` instruction performed on register operands turned out to result in an energy consumption below all our idle loops on both test systems. Unfortunately, this includes a certain amount of energy required for performing the calculations. The baseline power is therefore assumed to be slightly lower than what was actually measured for the `sqrtpd` loop (see Section III-A).

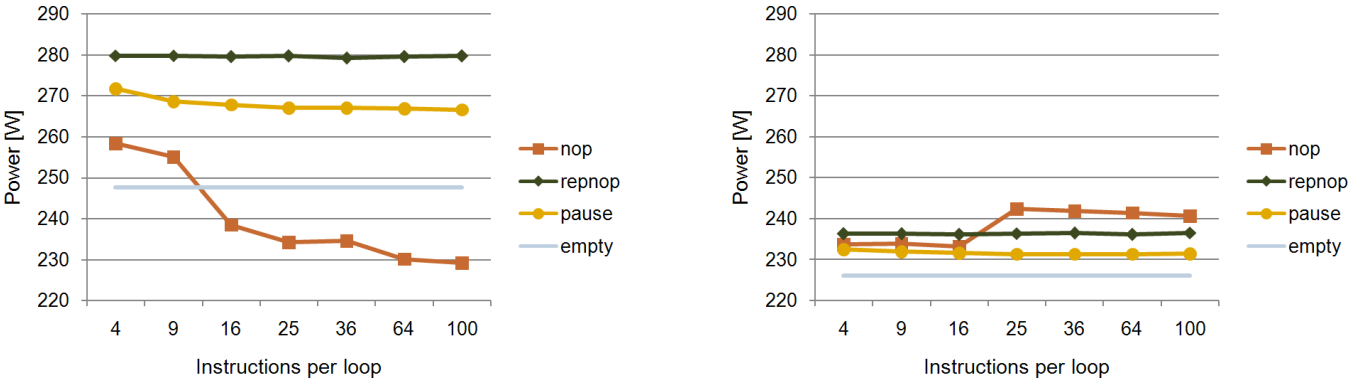
A disadvantage of using a baseline power consumption is that it can only be estimated. The differences between total power consumption and the baseline are therefore error-prone. Furthermore, the baseline approach does not compensate temperature effects like fan speed and leakage power. The effect of fan-speeds and other components could be compensated by measuring processor power separately at the dedicated 12 V power connector. However, the processor leakage would still influence the power estimates and it would still require an estimated baseline for the processor being active idle. Moreover, current platforms use the dedicated 12 V power connector only to supply a part of the processor [10]. Measurements of the 12 V rail would therefore have to be combined with the 3.3 and 5 V rails in order to include all processor components. An isolated power analysis of the CPU is not possible with standard lab equipment.

III. Results

A. Idle loops

The power consumption of idle loops (see Section II-D) is slightly higher if conditional jumps are used. However, the differences are not significant and we only discuss the results using unconditional jumps. Figure 3 shows the results for both test systems. Depending on the implementation of the idle loop its power consumption varies by 14 Watts on the Intel system and by 50 Watts on the AMD system.

The most notable effect on the Intel Xeon processor is the power saving induced by the loop stream detector. This feature allows the processor to disable parts of the pipeline like fetch and decode if it detects a loop. The 9 Watts lower power consumption of small `nop` loops compared to loops with more instructions (see Figure 3b) can be solely attributed to the loop stream detector. The power consumption of the `rep nop` loop is independent of the number of instructions per loop as it uses the value in ECX to adjust the number of iterations and therefore has a fixed code size that can be handled by the loop stream detector. This results in a fairly constant power consumption of 236 Watt for the `rep nop` loop, slightly more than the `nop` loop (233 Watt if only few instructions are in the loop). The `pause` operation within the loop uses about 231 Watt,



(a) AMD Opteron 2435 (Istanbul)

(b) Intel Xeon X5670 (Westmere-EP)

Figure 3. Comparison of the power consumption of different idle loops on our test systems

Table III
IDLE POWER CONSUMPTION OF THE TEST SYSTEMS

System	Intel Xeon X5670	AMD Opteron 2435
Low frequency	133 W (1600 MHz)	156 W (800 MHz)
High frequency	133 W (2933 MHz)	166 W (2600 MHz)
Idle loops	226-242 W	227-279 W
Register `sqrtpd` loop	223 W (2933 MHz)	221 W (2600 MHz)
Estimated baseline	220 W (2933 MHz)	218 W (2600 MHz)

less then any of the `nop` based idle loops. An "empty" loop (containing only the jump instruction) has the lowest power consumption (226 Watts).

The results of our AMD Istanbul test system are very different (see Figure 3a). The `nop` loop uses less power the more the loop is unrolled. In our experiments, the power consumption drops from 256 to 227 Watts. Unlike on the Intel system, the `nop` loop can be more efficient than an empty loop that consumes 247 Watts. We do not observe a similar effect for the `rep nop` and `pause` loops that consume 279 Watt and 267 Watts, respectively, and show only low fluctuation. It is particularly interesting to note that the AMD system does not use the `pause` hint efficiently. In contrast to the definition provided by AMD [1], the `pause` instruction apparently does not reduce the power consumption of a spinning loop.

Table III compares the power consumption of the idle loops with the operating system's idle implementation that utilizes C-states in order to be more energy efficient. The most surprising effect is that the AMD system consumes 11 Watts less if the processor cores are forced into highest P-state (lowest frequency) prior to entering the C-state. On the Intel system this makes no difference. If C-state support is disabled in the BIOS, Linux uses a `rep nop` loop. The power consumption in this case is equal to the results of our `rep nop` implementation. Table III also shows the workload with the lowest power consumption that we were able to measure, a loop that uses the `sqrtpd` instruction to perform calculations on register operands only. We use this loop for our baseline power estimation (see Section III-A) and subtract a small value that accounts for the energy consumed by the FPU and register file activity caused by this operation.

B. Intel Xeon X5670

Figure 4 shows the power consumption of the workloads described in Section II-C on our dual socket Intel Xeon X5670 test system. The bars depicting the bandwidth clearly show the four levels of the memory hierarchy, namely L1, L2, and L3 cache, and main memory. The respective accumulated bandwidth of all 12 cores of the system is 561, 372, 170, and 40 GB/s. There is only one noticeable deviation in case of the `and_pd` instruction that only reaches 528 GB/s in the L1 cache for an unknown reason. Still, the results demonstrate that our workloads can achieve a constant throughput for different instructions that either only transfer data into the processor registers or do a combination of data transfer and double precision floating point arithmetic. All results are well reproducible, the deviation between multiple runs is typically less than 2%.

The power consumption shows a regular pattern that is mostly consistent with our initial assumptions. For each memory location, the `load` instruction is the one with the lowest power consumption, as it only transfers data into the processor registers. The `and`, `add`, and `mul` instructions are increasingly more demanding (in this order). The combination of an `add` and a `mul` instruction (`mulpd+addpd`), that doubles the number of calculations done during a given time or for a given amount of data, has a surprisingly high impact on the power consumption. To put the power consumption into perspective, the lowest value on the secondary vertical axis (220 Watt) is the baseline power that we estimated in Section III-A.

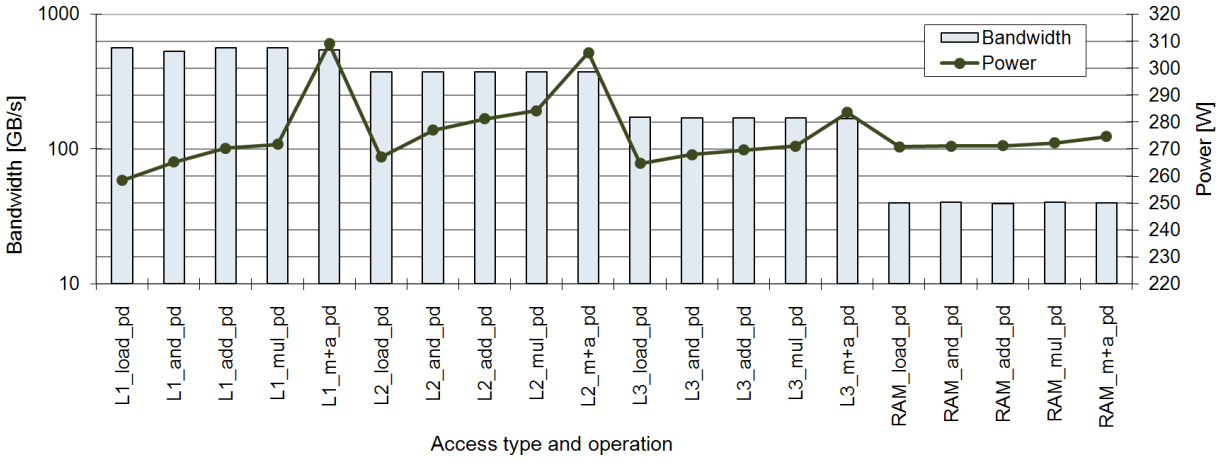


Figure 4. Power consumption and data throughput for different packed double instructions when accessing different memory locations of our Intel Xeon X5670 (Westmere-EP) test system

Figure 5 includes only a subset of the previously shown instructions, namely `load`, `add`, and `mul`. However, in this plot we compare the power consumption of three different data types: packed integer, packed double, and packed single. With one exception, the bandwidth that our workloads achieve is independent from the data type and Figure 5 therefore only shows the results that have been measured with the packed double version. The packed integer version only achieves 528 GB/s in the L1 cache for the `add` and `mul` instructions, similar to what we measured in the `and_pd` case. The L1 `load_pi` bandwidth of 561 GB/s as well as the bandwidth in all other cache levels is identical to what is achieved with packed double and packed single instructions.

The results in Figure 5 consistently show that integer instructions require the most power followed by double and single. The only exception is the L1 `mul` case due to the reduced integer bandwidth. The fact that integer operations consume more power is somewhat unexpected. This might be caused by the chip layout, e.g. longer distance between ALU and register file that results in additional energy needed for transferring data between registers and execution units. It should also be noted that the power consumption of the single precision case is lower than for double precision, although our workload performs calculations on the full SSE register in both cases (either two double precision or four single precision operations).

C. The Impact of HyperThreading on the CPU Power Consumption

In our highly optimized benchmarks, one thread per core can fully utilize the available bandwidth of all cache levels and main memory. The bandwidth can not be improved by using all logical cores of the system (24 threads instead of 12). This allows us to study how HyperThreading affects the power consumption of the processor for cases where the overall throughput is not increased by the additional threads. Figure 6 shows that the power consumption is significantly higher if two threads are used per core. The difference can be as high as 25 Watts in our test system. This indicates that the use of HyperThreading should be reconsidered from an energy efficiency viewpoint as the benefit for the application performance should exceed 10% in order to compensate the additional power consumption. However, we have verified that enabling or disabling HyperThreading in the BIOS of our Intel test system does not have a measurable effect on the system's power consumption as long as only one thread per core is used. Only the system's idle power increases slightly as the scheduler has to manage more CPUs. It is currently not clear if other implementations of simultaneous multi-threading (such as IBM POWER7) suffer from similar power efficiency issues.

D. Comparing AMD Opteron 2435 and Intel Xeon X5670

Figure 7a shows a comparison of the power consumption of our AMD and Intel test systems. The first thing to note in Figure 7a is that the L1 bandwidth on the AMD Opteron differs significantly depending on the workload. The Opteron microarchitecture achieves twice the L1 bandwidth when only loading data (`load_pd`) compared to the `add_pd` and `mul_pd` workload. This explains that the power consumption of the `load_pd` workload exceeds that of `add_pd` and `mul_pd`. The L2, L3, and main memory bandwidth do not depend on the workload. Unlike on the Intel system (see Figure 7b), the `load_pd` workload does not consume less power than `add_pd` and `mul_pd`. Instead, loading data into the registers is apparently the most expensive operation. This unexpected behavior should not be attributed to measuring inaccuracies, as we observe the same trend (`load_pd` highest, `add_pd` lowest, `mul_pd` in between) for L2, L3, and main memory. We suspect that this effect is caused by the same microarchitectural property that also causes the different L1 bandwidths, namely that `load` instructions can be handled by all floating-point pipelines while `add` and `mul` only use one pipeline.

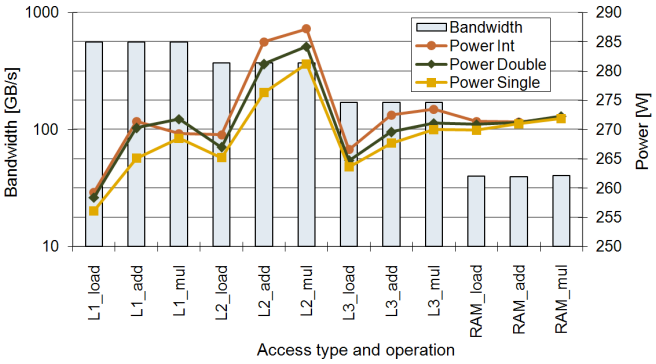


Figure 5. Comparison of the power consumption of SSE instructions (packed integer, double and single) on our Intel Xeon X5670 (Westmere-EP) test system

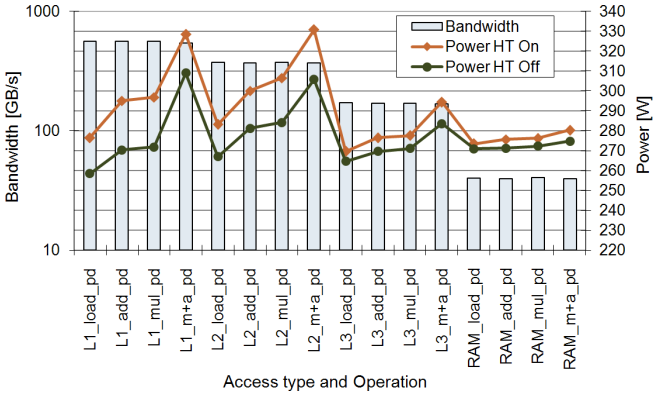
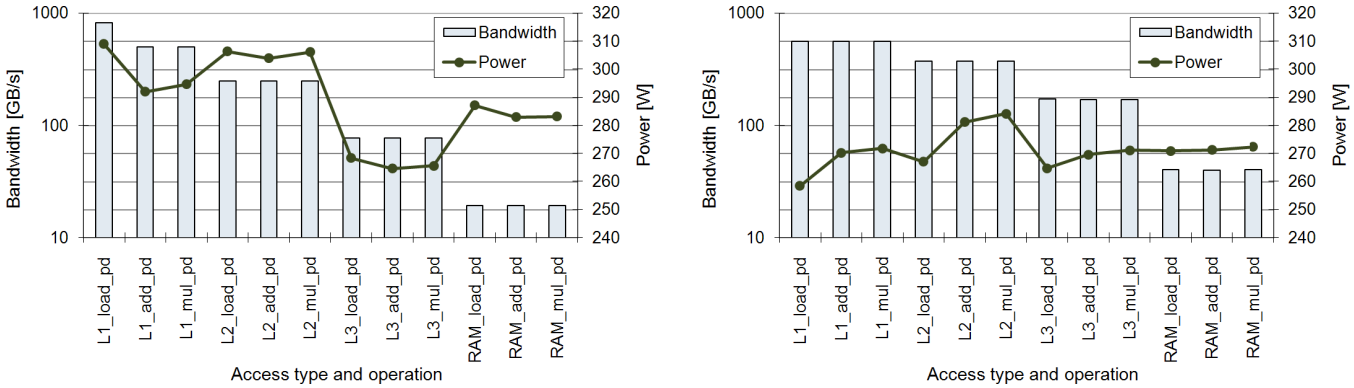


Figure 6. Power consumption of different instructions on our Intel Xeon X5670 test system with HyperThreading enabled and disabled; the data throughput is identical for both cases, only the number of threads differs



(a) AMD Opteron 2435 (Istanbul)

(b) Intel Xeon X5670 (Westmere-EP)

Figure 7. Comparison of the power consumption and data throughput of our AMD and Intel test systems

IV. Modeling Energy Consumption

Besides the general evaluation presented in Section III, our results can be used as a basis to model a CPU's energy consumption. For example, the cost for data transfers from different locations in the memory subsystem into the processor registers can be very interesting, especially when combined with the cost for arithmetic operations. In this Section we therefore describe a simple, strictly linear model that characterizes the energy consumption of the Intel Westmere-EP processor for basic data transfers and arithmetic operations.

One property of the Istanbul microarchitecture is that loads consume more power than arithmetic operations on the same data (see Figure 7a). In order to determine the energy needed for arithmetic operations we would need to subtract the energy required for loading data from the energy consumed during the execution of the arithmetic instructions. This would result in negative values for the energy per operation and thus prevents us from assessing the energy requirements of arithmetic operations for the AMD platform.

A. Data Transfer

Our results show that the single precision load instruction (`movaps`) has the lowest average power consumption for data transfers independent of the memory location (see Figure 5). We therefore use this operation to determine the energy consumption of data transfers from the L1, L2, L3 cache or main memory. The energy consumed per Byte of transferred data (E_{trans}) is derived from the baseline power ($P_{base} = 220\ W$, see Section II-D), the measured power consumption (P_{total}), and the data throughput for the `load_ps` benchmark:

$$E_{trans}\ [\frac{nJ}{Byte}] = \frac{P_{total}\ [W] - P_{base}\ [W]}{bandwidth\ [\frac{GB}{s}]}.$$

Table IV shows that transfers from the L2 or L3 cache are about twice as expensive as from the L1 or L2 cache, respectively. Transferring data from main memory consumes five times as much energy as a L3 cache access. This clearly demonstrates the disadvantage of main memory accesses, not only due to the limited bandwidth, but also with respect to the high energy demand.

Table V compares our two test systems with respect to the energy consumption of data transfers. Due to the limitations of our measurement methodology, these numbers can only be considered to be estimates. However, they do allow to draw some conclusions. The 1.6-fold difference for L1 accesses can most likely be attributed to the different dynamic power consumption properties of the technology nodes used to build the processors (32 nm HKMG for Intel, 45 nm SOI for AMD). Both microarchitectures do not require write-backs when data is loaded from the L1 data cache into the processor registers. However, the different size of the L1 cache as well as other properties like the dual-ported load interface from the core to the

Table IV
Bandwidth and energy consumption of data transfers (using the movaps instruction) from different memory locations on the Intel Xeon X5670 test system

Location	P_{total}	$Bandwidth$	E_{trans}
L1	256.1 W	561.6 GB/s	64 pJ/Byte
L2	265.2 W	372.2 GB/s	121 pJ/Byte
L3	263.6 W	171.6 GB/s	254 pJ/Byte
RAM	269.9 W	39.9 GB/s	1250 pJ/Byte

Table V
Comparison of the energy consumption of data transfers (movaps), AMD Opteron 2435 and Intel Xeon X5670

Location	E_{trans} AMD	E_{trans} Intel	Factor
L1	105 pJ/Byte	64 pJ/Byte	1.6
L2	357 pJ/Byte	121 pJ/Byte	2.9
L3	654 pJ/Byte	254 pJ/Byte	2.6
RAM	3590 pJ/Byte	1250 pJ/Byte	2.8

L1 cache can also influence the result. The L2 and L3 cache show a 2.9-fold and 2.6-fold difference, respectively. The increased difference compared to the L1 case is likely to be an effect of the different cache architectures. Intel's inclusive caches allow accesses to the L2 cache with old L1 cache lines being evicted silently (without write-backs). The same holds true for L3 accesses and silent L1 and L2 evictions. In contrast, AMD's exclusive/non-inclusive design always requires write-backs when unmodified exclusive data needs to be evicted from the L1 or L2 cache. The 2.8-fold difference for main memory accesses is dominated by different bandwidth with the Opteron's DDR2 providing half the throughput compared to the Xeon's DDR3 with a similar power envelope. It allows no further conclusion with respect to the efficiency of both architectures.

B. Arithmetic Operations

We derive a estimate of the energy consumption of arithmetic operations (E_{calc}) based on the baseline power (P_{base}), the measured power consumption (P_{total}) for a certain workload, the data throughput (bandwidth), the number of bytes per arithmetic operation (ω_{op}, see Table VI) for this benchmark, and the energy consumption of data transfers (E_{trans}, see Section IV-A).

$$E_{calc}[\frac{nJ}{op}] = (\frac{P_{total}\ [W] - P_{base}\ [W]}{bandwidth[\frac{GB}{s}]} - E_{trans}\ [\frac{nJ}{Byte}]) * \omega_{op}[\frac{Byte}{op}]$$

Table VI presents the energy consumption for different arithmetic operations. Again, these numbers can only be seen as an estimation that allows a comparison of individual operations as well as a comparison of the energy requirements of calculations and the data transfer estimations presented in Section IV-A. It is interesting to note that a single precision calculation consumes less than half the energy of a double precision calculation. Moreover, the combined double precision addition and multiplication operation turns out to be fairly expensive.

Table VI
ENERGY CONSUMPTION OF ARITHMETIC OPERATIONS ON THE INTEL XEON X5670 TEST SYSTEM

Workload	operations per 16 Byte	ω_{op}	E_{calc}
add_pi	2 (64 Bit)	8 Byte/op	428 pJ/op
mul_pi	2 (64 Bit)	8 Byte/op	476 pJ/op
add_pd	2 (64 Bit)	8 Byte/op	319 pJ/op
mul_pd	2 (64 Bit)	8 Byte/op	387 pJ/op
mul+add_pd	4 (64 Bit)	4 Byte/op	464 pJ/op
add_ps	4 (32 Bit)	4 Byte/op	111 pJ/op
mul_ps	4 (32 Bit)	4 Byte/op	164 pJ/op

C. Limitations

One limitation of the presented approach is the lack of an efficiency estimation of the power supply. As the power supply's efficiency depends on its operating point, it certainly affects the correlation between throughput and energy consumption. Although done carefully, our estimation of the baseline power of the system (power consumption of the system with all cores being idle but not in a C-state) significantly affects all assumptions presented in Section IV. Other systematic errors of our model may be caused by the non-consideration of the branch predictors or other processor features. Another weakness is the missing correlation to the processor temperature.

More importantly, our data initialization routine currently does not consider that the power consumption of data transfers as well as arithmetic operations may be influenced by the data initialization. However, multiple initialization values have been tested for all operations and the one with the highest observed power consumption has been used. A systematic investigation with respect to the population count of the used operands as well as the hamming distance between a value within a register and a new value overwriting may be required to identify the actual worst case.

V. RELATED WORK

Several papers [5][11][14] topic the usage of performance counters to estimate the power consumption for a specific load on x86 processors. This approach, however, is not precise enough to distinguish between all different types of operations. As was described in Section III-B, the type of operation is as important as the used datatype or memory level. These three factors can be hardly measured using performance counters. Other publications focus on the performance for transfers from different memory levels[4], but do not topic the influence on the power consumption.

Tiwari et al. measure the power consumption of the system and the current for CPU and DRAM to estimate the power consumption of single instructions for an Intel 486 DX and a Fujitsu SPARClite 934 [13]. Steinke et al. measure the power consumption for different operations on an ARM7TDMI processor, including predictions about the influence of the set bits in moved data [12]. Chang et al. analyzed the power consumption of the very same processor with respect to the different pipeline stages [2]. The less complex architecture and the option to measure only parts of the system lead to much more accurate conclusions about the used energy for different instructions.

McIntire et. al. present an embedded network sensor system for energy monitoring and management [8]. The proposed platform LEAP allows to monitor the energy consumption of different subsystems. This work monitors the energy consumption of different components (e.g. the processor) per task while we approximate the energy consumption of data transfers and single instructions. Another

approach to determine the application's influence on the power consumption of different components like CPU, memory, disk, and fans has been presented by Ge et al. [3].

VI. Conclusion

We have developed a set of multi-threaded x86-64 specific workloads that specifically stress certain CPU components. The innermost routines have been implemented using assembly to allow complete control over the instruction stream that is executed by the processor cores. Moreover, our workloads allow us to strictly control the location of all data, most importantly in which level of the memory hierarchy source operands are located.

The potential of the presented approach is demonstrated with an in-depth comparison of two state-of-the-art six-core processors from AMD (Istanbul microarchitecture) and Intel (Westmere-EP microarchitecture). We reveal that for example the power consumption of different types of idle loops (e.g. using `nop` or `pause` instructions) differs strongly on the two processors. We can also analyze energy efficiency ramifications of features like Intel's loop stream detector. HyperThreading is another feature that is typically analyzed with respect to its impact on application performance. Our results allow us to quantify the impact of HyperThreading with respect to the power consumption, especially for workloads that do not show any throughput improvements. Fortunately, there is no need to disable the feature completely as it is sufficient to restrict an application that does not benefit from HT to one thread per core using `taskset` or other affinity control mechanisms.

Our detailed analysis of the power consumption of different workloads that either only transfer data into the processor registers or combine this with arithmetic operations also reveals significant differences between our two test systems. We use a basic model to approximate and compare the energy consumption of data transfers from different cache levels and main memory on both microarchitectures. Based on our different ALU and FPU workloads on the Westmere-EP based Intel test system we characterize the energy consumption of arithmetic operations. Moreover, this allows us to compare data transfers and computations with respect to their energy consumption.

In future work we plan to extend our approach in order to measure the energy consumption of data transfers between cores, processors, or even nodes. Adding more sophisticated instructions to our analysis and the energy model may be of interest as well. Moreover, minor hardware modifications of our test systems are needed in order to remove unwanted power consumption influences that are e.g. caused by the system fans. The power consumption model may also be used to make a substantiated selection of hardware performance counters that can be used to estimate the total power consumption of processors in software.

Acknowledgment

The authors would like to thank Intel Germany for providing us with the Westemere-EP test system. We would also like to express our gratitude to Maik Schmidt and Michael Kluge for their continuous contributions to the software environment that we use for our work. This work has been funded by the Bundesministerium für Bildung und Forschung via the Spitzencluster CoolSilicon (BMBF 13N10186) and the research project eeClust (BMBF 01IH08008C).

References

[1] Advanced Micro Devices. *AMD64 Architecture Programmers Manual Volume 3: General-Purpose and System Instructions*, 3.15 edition, November 2009.

[2] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy consumption measurement and analysis: case study of arm7tdmi. In *ISLPED*, pages 185–190, 2000.

[3] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658 –671, may 2010.

[4] Daniel Hackenberg, Daniel Molka, and Wolfgang E. Nagel. Comparing cache architectures and coherency protocols on x86-64 multicore smp systems. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 413–422, New York, NY, USA, 2009. ACM.

[5] Russ Joseph and Margaret Martonosi. Run-time power estimation in high performance microprocessors. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, pages 135–140, New York, NY, USA, 2001. ACM.

[6] Klaus-Dieter Lange. Identifying shades of green: The SPECpower benchmarks. *IEEE Computer*, 42(3):95–97, 2009.

[7] Weiping Liao, Lei He, and K.M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(7):1042 – 1053, july 2005.

[8] Dustin McIntire, Kei Ho, Bernie Yip, Amarjeet Singh, Winston Wu, and William J. Kaiser. The low power energy aware processing (leap)embedded networked sensor system. In *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, pages 449–457, New York, NY, USA, 2006. ACM.

[9] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Matthias S. Müller. Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system. In *PACT '09: Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 261–270, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Michael Schuette. Core i7 power plays. http://www.lostcircuits.com/mambo/index.php?option=com_content&task=view&id=44, December 2008.

[11] Karan Singh, Major Bhadauria, and Sally A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, 2009.

[12] Stefan Steinke, Markus Knauer, Lars Wehmeyer, and Peter Marwedel. An accurate and fine grain instruction-level energy model supporting software optimizations. In *in Proc. Int. Workshop Power & Timing Modeling, Optimization & Simulation (PATMOS)*, 2001.

[13] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing Systems*, 13:223–238, 1996.

[14] Wei Wu, Lingling Jin, Jun Yang, Pu Liu, and Sheldon X.-D. Tan. A systematic method for functional unit power estimation in microprocessors. In *DAC '06: Proceedings of the 43rd annual Design Automation Conference*, pages 554–557, New York, NY, USA, 2006. ACM.