Impact of Thread and Frequency Scaling on Performance and Energy in Modern Multicores: A Measurement-based Study

Armen Dzhagaryan
Electrical and Computer Engineering
The University of Alabama in Huntsville
301 Sparkman Dr, Huntsville, AL 35899
aad0002@uah.edu

Aleksandar Milenković
Electrical and Computer Engineering
The University of Alabama in Huntsville
301 Sparkman Dr, Huntsville, AL 35899
milenka@uah.edu

ABSTRACT
Modern microprocessors integrate a growing number of components on a single chip, such as processor cores, graphics processors, on-chip interconnects, shared caches, memory controllers, and I/O interfaces. An ever-increasing complexity and the number of components present new challenges to software developers interested in finding operating points that strike an optimal balance between performance and energy consumed. In this paper we analyze the impact of thread scaling and frequency scaling on performance and energy in modern multicores. By exploiting recent additions to microprocessors that support energy estimation and power management, we measure execution times and energy consumed on an Intel Xeon 1240 v2 microprocessor when running the PARSEC benchmark suite. We conduct a number of experiments by varying the number of threads, $1 \le N \le 16$, and processor clock frequency, $1.6 \le F \le 3.4$ GHz. We find that the maximum performance is achieved when the number of threads matches or slightly exceeds the number of logical processors ($8 \le N \le 12$) and the clock frequency is at maximum ($F = 3.4$ GHz). The minimum energy is consumed when the processor clock frequency is in range $2.0 \le F \le 2.4$ GHz. Finally, we find that the best performance at minimal energy is achieved when $8 \le N \le 12$ and $2.8 \le F \le 3.1$ GHz.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Measurement techniques.

General Terms
Measurement, Performance, Experimentation.

Keywords
Energy-efficiency, Power Profiling.

1. INTRODUCTION
Modern microprocessors have evolved into complex system-on-a-chip (SoC) designs that integrate a growing number of general-purpose processor cores, graphics processors, memory controllers, and other components on a single silicon die. They are often referred to as multicore processors or just multicores. Multicores continue to evolve by integrating an ever increasing number of new architectural structures and features aimed at achieving high-performance. Such features include vector processing, tight integration with graphics, hardware acceleration of various time-critical operations, and deep caching hierarchies. In such conditions, exploiting the performance of modern multicores require intimate knowledge of underlying architecture. Software developers thus rely on modern software tools for software tuning and optimization to fully harness the capabilities of modern multicores.

In conditions when power and thermal envelopes of multicores are bounded, increasing levels of integration and core clock frequencies pose new power and energy challenges. To address these challenges, the focus has shifted to highly energy-efficient designs. In addition, a number of new power management schemes and power states have been introduced. They are geared toward achieving the maximum performance when needed and preserving power and energy when the maximum performance is not needed. For example, various components including processor cores can be selectively turned on or off to match workload, the processor core clocks can be adjusted "on-the-fly" in order to reduce power or temperature of the die (dynamic frequency scaling, DFS), and the power supply can also be adjusted "on-the-fly" (dynamic voltage scaling, DVS). Modern operating systems include utilities that allow users to set clock frequency or choose a particular power scheme that fits their application needs [1].

Modern Intel microprocessors, starting with SandyBridge architecture, include the Running Average Power Limit (RAPL) interface [2]. Although the RAPL interface is designed to limit power usage on a chip while ensuring maximum performance, this interface supports power and energy measurement capabilities. An on-chip circuitry estimates energy usage based on a model driven by: (a) architectural event counters from all components, (b) temperature readings on the die, and (c) current leakage models. Estimates are available to users in a model-specific register (MSR), updated in the order of milliseconds. Energy estimates offered by RAPL have been validated by Intel and closely follow actual energy used [3]. A number of tools have been introduced [4]–[8] to allow software developers to estimate power and energy of running programs.

When executing a parallel program on a multicore machine we can vary a number of parameters, such as the number of threads and the processor clock frequencies. These parameters have a direct impact on performance and energy consumed. What is the optimal number of threads for a given program on a given ma-

ACM SE '14, March 28 - 29 2014, Kennesaw, GA, USA

http://dx.doi.org/10.1145/2638404.2638473

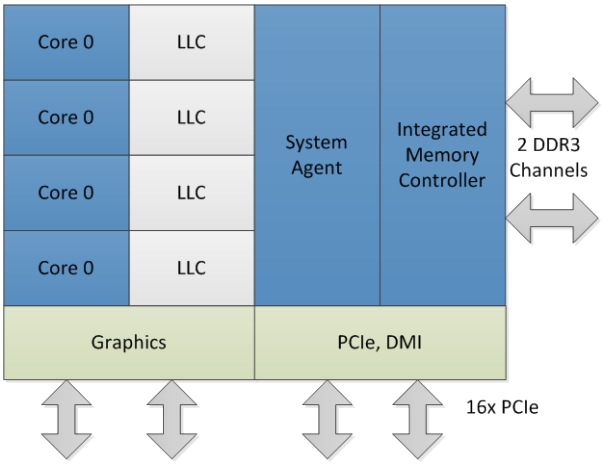

Figure 1. Block diagram of an Intel Xeon E3-1240 v2.

chine if we want to achieve the maximum performance? What is the optimal clock frequency if we want to minimize energy consumed? To answer these questions we perform a measurement based study that explores the impact of thread and frequency scaling on performance and energy consumed in modern multicores.

In this paper we present the results of our study performed on an Intel's Xeon E3-1240 v2 microprocessor when running parallel programs from the PARSEC benchmark suite [9]. We measure benchmark execution times and energy consumed using LIKWID tool suite, while varying the number of threads from N = 1 to N = 16 and processor clock frequency from a maximum F = 3.4 GHz to a minimum F = 1.6 GHz. Section 2 describes our experimental methodology, including measuring setup, benchmarks, experiments, and metrics.

We find that majority of Parsec benchmarks achieve the maximum performance when the number of threads matches or slightly exceeds the number of logical processors on a test machine ($8 \le N \le 12$) and when the clock frequency is at maximum F = 3.4 GHz. We find that the minimum energy is consumed when clock frequency is in range $2.0 \le F \le 2.4$ GHz for almost all benchmarks. Finally, we find that an optimal performance-energy ratio is achieved when $8 \le N \le 12$ and $2.8 \le F \le 3.1$ GHz. Section 3 describes the detailed results of our experimental evaluation, and Section 4 concludes the paper.

2. EXPERIMENTAL METHODOLOGY

In this section we describe our experimental methodology, including hardware and software setup (2.1), benchmarks (2.2), and experiments and metrics (2.3).

2.1 Hardware and Software Setup

Our experimental setup includes a Dell PowerEdge T110 II server with a single Intel Xeon E3-1240 v2 processor and 16 Gbytes of memory. The Xeon E3-1240 v2 processor consists of a single monolithic die with four 2-way threaded physical processor cores for a total of 8 logical processor cores, a shared 8 Mbytes L3/LLC cache memory, an integrated memory controller, PCI and DMI interfaces, a graphics processor, and a system agent (Figure 1). The system agent encompasses a module responsible for power management called the Package Control Unit (PCU). The PCU connects to individual processor cores and other functional blocks via power management agents that collect information about power consumption and junction temperature. The PCU runs firmware that constantly monitors power and thermal conditions and performs various power-management functions, e.g., turn on or off a processor core or portions of the LLC cache or dynamically scale voltage and frequency.

The server runs the CentOS 6.3 operating system with 2.6.32 Linux kernel. To change the processor clock frequency we use a set of utilities called *cpufrequtils*. The *cpufreq-info* utility allows a user to inspect the current clock frequency setup on each of the processor cores. The *cpufreq-set* utility allows a user to set the minimum, the maximum, and the current clock frequency, as well as a governor that specifies a power scheme for each processor core. The power schemes, such as Performance, Powersave, Userspace, Ondemand, and Conservative [1], dictate to the Linux kernel how to dynamically adjust the processor frequencies. For example, the Performance governor forces processor cores to constantly run at the maximum allowed frequency. The Xeon E3-1240 v2 processor supports a total of 18 frequencies, ranging from the minimum 1.6 GHz to the maximum 3.4 GHz. In our experiments we force all processor cores to run at a specific clock frequency, which remains fixed during a benchmark run.

Table 1. PARSEC Benchmark suite.

	Benchmark	*Application Domain*	*Parallelism Model*	*Working Set*	*Communication*
1	blackscholes	Computational finance application	data-parallel	small	low
2	bodytrack	Computer vision application	pipeline	medium	medium
3	canneal	Electronic Design Automation (EDA) kernel	data-parallel	huge	high
4	dedup	Enterprise storage kernel	pipeline	huge	high
5	facesim	Computer animation application	data-parallel	large	medium
6	ferret	Similarity Search application	pipeline	huge	high
7	fluidanimate	Computer animation application	data-parallel	large	medium
8	freqmine	Data mining application	data-parallel	huge	medium
9	raytrace	Computer animation application	data-parallel	medium	high
10	streamcluster	Machine learning application	data-parallel	medium	medium
11	swaptions	Computational finance application	data-parallel	medium	low
12	vips	Media application	data-parallel	medium	medium
13	x264	Media application	pipeline	medium	high

2.2 Benchmarks

The Princeton Application Repository for Shared-Memory Computers (PARSEC) [9] is a benchmark suite composed of a diverse set of multithreaded programs. The suite focuses on emerging workloads and was designed to be representative of the next-generation shared-memory programs for chip-multiprocessors. It is frequently used in research as well as in performance measurements on real machines.

Table 1 gives the PARSEC benchmark names, as well as additional information including application domain, parallelism mode, working set size, and communication intensity between threads. The PARSEC benchmarks cover a wide range of computer tasks such as financial analysis (1, 11), computer vision (2), engineering (3), enterprise storage (4), animation (5, 7 and 9), similarity search (6), data mining (8), machine learning (10), and media processing (12, 13). Benchmarks vary in type of parallelization model (data-parallel or pipelined), working set (ranging from small to huge), and communication intensity (varying between low and high). For each benchmark several data input sets are available, including *Test*, *Simdev*, *Simsmall*, *Simmedium*, *Simlarg*e, and *Native* (with *Test* being the smallest and *Native* being the largest input set). In our experiments we use PARSEC 3.0 benchmarks compiled using the gcc 4.4.7 compiler. The *Native* input set is used as the workload.

2.3 Experiments and Metrics

To evaluate the impact of the thread and frequency scaling on performance and energy efficiency, we conduct a number of experiments while varying the number of threads (from N = 1 to N = 16) and the clock frequency (F = 3.4, 3.3, 3.1, 3.0, 2.8, 2.6, 2.5. 2.4, 2.2, 2.1, 2.0, 1.9, 1.7, 1.6 GHz). Using the *cpufrequtils* the clock frequency is set to a chosen value, and the benchmarks are run under the likwid-powermeter tool [6]–[8].

To illustrate the running of PARSEC benchmarks, an example run script for *blackscholes* is shown in Figure 2. The outer loop sets the number of threads (N) and the inner loop defines a number of runs for each experiment (set to 3 for each benchmark/thread combination). The command line specifies the benchmark executable (*blackscholes*), its input (in_10M.txt) and output (prices.txt). The *likwid-powermeter* tool captures the benchmark run time (or wall-clock execution time, ET) and the energy consumed in joules (E). getResults is a script based on the *grep* and *awk* Linux utilities that filters the output report to extract and record the benchmark execution time and the energy consumed. Similar scripts are prepared for all Parsec benchmarks.

```
1.  for N in {1..16} # the number of threads
2.  do
3.     echo "$N" | tee -a $path/likwid.txt;
4.     for i in {1..3}
5.     do
6.       likwid-powermeter $path2exe/blackscholes
         $N in_10M.txt prices.txt | getResults >>
         $path/likwid.txt
7.     done
8.  done
```

Figure 2. Run script for blackscholes.

3. RESULTS

3.1 Performance

Execution times of benchmarks are affected by both the number of threads and processor frequency. We can expect that the execution time scales linearly with the processor clock cycles, though some deviations caused by benchmark characteristics (e.g., frequency of memory references, data locality, and others) are possible. In addition, synchronization among multiple threads and data communication can impact the performance of multi-threaded programs.

First, we consider the impact of thread scaling. Table 2 shows the execution times for all PARSEC benchmarks, as a function of the number of threads, ranging from N = 1 to N = 16, when the processor clock is set to the maximum, F = 3.4 GHz. Figure 3 shows the speedup calculated as the ratio between the execution time for a single-threaded run and the execution time with N threads, $1 \le N \le 16$, ET(1)/ET(N). The performance increases as we increase the number of threads up to a point. Some benchmarks such as *freqmine*, *swaptions*, and *x264* scale very well, achieving maximum speedups of 6.7, 6.8, and 7.2, respectively. Somewhat surprisingly, several benchmarks achieve their maximum speedups when the number of threads exceeds the number of available threads on our machine (N = 8). Scaling beyond N = 12 threads yields no significant performance benefits, and often results in a slight performance degradation (e.g., in *bodytrack*, *freqmine*, *raytrace*, *streamcluster*, *swaptions*).

How does performance scale with clock frequency? To answer

Table 2. Execution time (ET) in seconds as a function of the number of threads (F = 3.4GHz).

Benchmark	Number of Threads															
	N=1	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11	N=12	N=13	N=14	N=15	N=16
blackscholes	150.9	84.5	62.3	52.4	50.6	47.3	44.8	41.7	42.7	42.3	42.7	42.6	42.2	42.1	42.0	41.6
bodytrack	126.6	67.3	47.2	41.0	37.8	35.7	33.5	32.9	32.7	25.1	25.3	25.7	26.0	26.2	26.7	32.4
canneal	193.8	123.0	98.7	91.7	94.8	87.8	82.8	78.0	90.0	62.5	61.4	60.7	60.7	61.0	62.1	84.7
dedup	21.2	12.6	10.9	10.4	14.2	10.5	9.6	9.6	13.5	11.7	11.7	12.4	13.4	13.5	13.8	13.9
facesim	338.1	175.7	124.4	114.5	NA	105.5	0.0	92.7	NA	NA	NA	NA	NA	NA	NA	113.7
ferret	270.7	137.5	95.9	78.2	70.5	66.7	64.7	64.1	63.5	45.5	45.5	45.8	46.1	46.4	46.8	62.6
fluidanimate	280.1	148.5	NA	91.4	NA	NA	NA	68.6	NA	NA	NA	NA	NA	NA	NA	76.1
freqmine	404.4	203.0	150.3	130.2	99.4	95.3	89.2	85.5	86.7	60.5	61.8	62.4	62.4	63.3	63.5	87.1
raytrace	211.5	137.7	111.7	100.7	97.3	94.3	91.7	90.2	90.4	63.4	63.5	63.6	64.1	64.9	64.7	89.6
streamcluster	357.1	186.1	129.7	108.5	105.0	91.5	81.8	74.5	112.5	76.8	75.6	75.5	75.5	76.4	77.7	107.1
swaptions	240.1	120.1	80.7	61.9	62.7	57.6	51.7	47.1	49.9	35.5	36.1	35.9	37.1	36.7	37.1	48.5
vips	93.5	48.3	35.5	31.1	28.4	25.5	25.5	25.0	23.8	20.9	19.7	20.5	21.8	20.4	20.5	23.6
x264	109.7	40.6	28.3	23.2	23.7	21.4	19.3	18.1	18.0	15.3	15.6	15.9	16.2	16.5	16.8	17.8

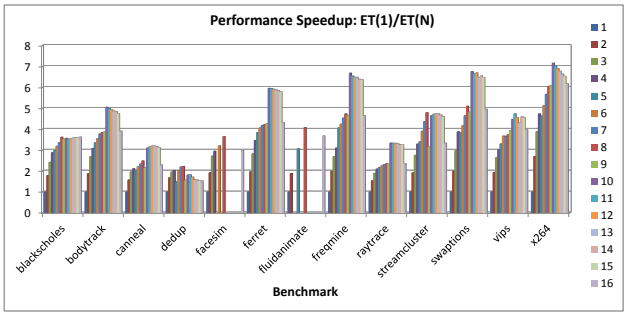

Figure 3. Performance speedup: normalized execution times as a function of the number of threads (N=1-16).

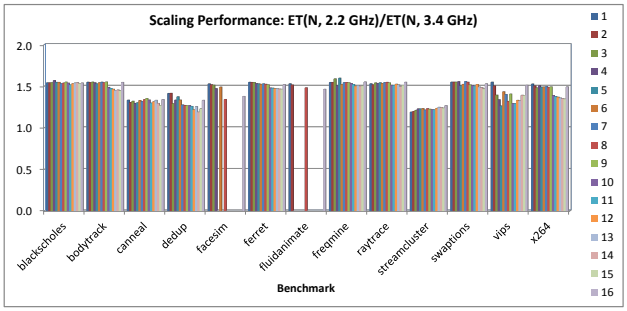

Figure 4. Performance scaling with frequency: ET(N, 2.2GHz)/ET(N, 3.4GHz).

this question, we compare benchmark execution times at two clock frequencies. Figure 4 shows the ratio of the benchmark execution times for two distinct clock frequencies of F = 2.2 and F = 3.4 GHz, ET(2.2GHz)/ET(3.4GHz). The ratio of these two frequencies is 3.4/2.2=1.54, and we expect to see a similar ratio between the benchmark execution times. Indeed, the results from Figure 4 show that many benchmarks exhibit linear scaling independent of the number of threads, e.g., *blackscholes*, *bodytrack*, *freqmine*, *raytrace*, and *swaptions*. Three notable exceptions are *canneal*, *dedup*, and *streamcluster*. For these three benchmarks the slowdown due to running at lower frequency is less than expected 1.54 times: ~1.3 times for *canneal* and 1.2 times for *streamcluster*. To explain this behavior, we take a closer look at characteristics of these benchmarks, especially their cache memory behavior.

Cache memories in modern processors play a critical role in achieving high performance. In benchmarks where the number of of cache misses is relatively high, the processor clock frequency scaling may result in non-linear effects. For example, let as assume that the processor is waiting on a cache miss to be serviced from main memory, which may take hundreds of processor clock cycles. Let us assume that it takes 100 ns to satisfy a cache miss. If a processor is running at F = 3.4 GHz, the stall time corresponds to 340 clock cycles. However, when the processor is running at F = 2.2 GHz, the stall time corresponds to 220 processor clock cycles. Thus, the penalty due to cache misses expressed in clock cycles is relatively higher when the processor is running at a higher clock frequency. We expect these benchmarks to have a significant percentage of cache misses.

To confirm this observation we profile the PARSEC benchmarks using the *perf* tool [10]. We measure an event called "cache misses" that captures the number of memory accesses that cannot be served by any of the cache memories (L1, L2, and L3 caches).

Table 3. Number of cache misses per 1K executed instructions as a function of the number of threads.

Misses/1K ins	Number of Threads				
	1	2	4	8	16
blackscholes	0.093	0.092	0.093	0.090	0.091
bodytrack	0.029	0.030	0.031	0.032	0.033
canneal	9.003	9.053	9.162	8.949	9.034
dedup	0.790	0.449	0.461	0.503	0.660
facesim	0.534	0.536	0.712	1.065	0.796
ferret	1.104	1.144	1.191	1.222	1.257
fluidanimate	0.763	0.825	0.999	1.128	1.193
freqmine	0.091	0.112	0.141	0.174	0.195
raytrace	0.294	0.305	0.310	0.335	0.354
streamcluster	14.954	14.914	14.865	14.879	9.697
swaptions	0.000	0.000	0.000	0.000	0.000
vips	0.047	0.050	0.065	0.215	0.295
x264	0.748	1.416	2.124	2.450	2.374

Table 3 shows the number of cache misses per 1,000 (1K) executed instructions. Two benchmarks, *canneal* and *streamcluster* show a very high number of misses, thus confirming our hypothesis. However, cache misses cannot quite explain behavior of *dedup* – its number of cache misses is not that high, though it still scales non-linearly with the clock frequency. Unlike other benchmarks, *dedup* spends a significant portion of its execution time in input/output operations as it is witnessed by a relatively large number of page faults. Again, the relative impact of these stalls on the benchmark execution time is higher when the processor is running at higher clock frequencies.

3.2 Energy

Similarly to execution time, the total energy a processor spends on a benchmark execution is affected by both the number of threads and processor frequency. In this section, we analyze the impact of these two parameters on the total energy.

Table 4 shows the total energy in Joules for all benchmarks as a function of the number of threads (N = 1 to N = 16) for a fixed processor frequency of F = 3.4 GHz. Figure 5 shows energy savings as a function of the number of threads, calculated as the ratio of the energy consumed by a benchmark run as a single-threaded application, E(1), and the energy consumed by a benchmark run with N threads, E(N).

The total energy is directly proportional to the benchmark execution time and power, E = ET*P. As the number of threads increases ($1 < N \leq 12$), the execution time decreases, and thus the total energy tends to go down. On the other side, multiple threads running in parallel result in an increase in the current drawn by the processor. This, in turn, results in an increased power and consequently the total energy. Which of these two trends prevails is an open question and depends on benchmark characteristics as well as on processor design. Our results indicate that the parallel execution improves overall energy efficiency for all benchmarks. The energy efficiency is improved over 1.5 times for almost all benchmarks with a sufficiently large number of threads (N = 8 to N = 12). The largest energy savings over a single-threaded execution is recorded for x264 when N = 12, and it reaches a factor of 2.2. Typically, the best energy efficiency is achieved when the number of threads is $8 \leq N \leq 12$, which corresponds to or slightly exceeds the number of logical processor cores. For some bench-

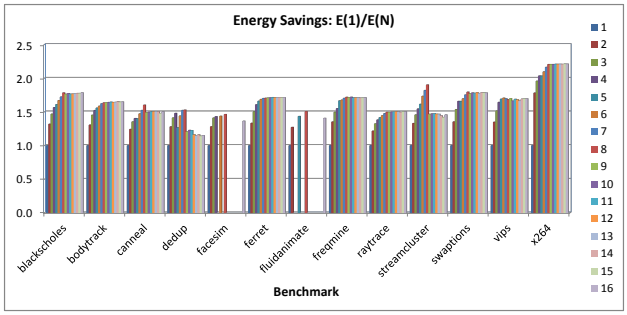

Figure 5. Energy savings as a function of the number of threads: E(1, 3.4GHz)/E(N, 3.4GHz).

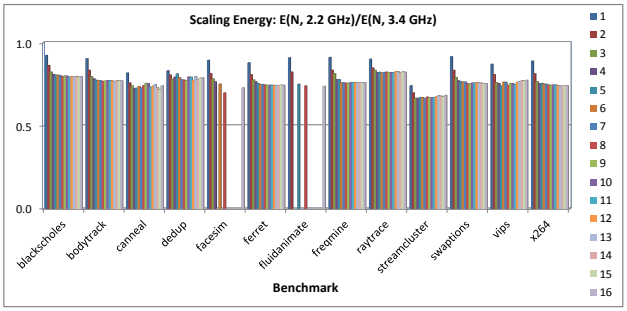

Figure 6. Energy scaling as a function of frequency: E(N, 2.2GHz)/E(N, 3.4GHz).

marks the total energy savings tend to decrease if we keep increasing the number of threads beyond N = 8. The *dedup* and *streamcluster* benchmarks show this type of behavior. On the other hand, benchmarks such as *blackscholes*, *bodytrack*, *ferret*, *freqmine*, *raytrace,* and *x264* plateau at a certain level of energy savings.

How does frequency scaling impacts the total energy? By lowering the processor clock frequency, the benchmark execution times will increase. On the other side, the current drawn by the microprocessor will decrease, resulting in a lower power during program execution.

To analyze the impact of frequency scaling we consider the total energy consumed by the Parsec benchmarks when the clock frequency is F = 2.2 GHz and compare it with the total energy when the clock frequency is F = 3.4 GHz. Figure 6 shows the ratio of energies E(N, 2.2)/E(N, 3.4 GHz), when $1 \le N \le 16$. We can see that running at the lower clock frequency reduces the the total energy for a fixed number of threads. For N = 1, the energy when running at F = 2.2 GHz is between 0.74 (*streamcluster*) and 0.92 (*blackscholes*) of the energy when running at F = 3.4 GHz. For N = 8, the energy when running at F = 2.2 GHz is in range between 0.67 (*streamcluster*) and 0.80 (*blackscholes*) of the energy when running at F = 3.4 GHz. Thus, the energy is reduced when running at F = 2.2 GHz, regardless of the number of threads. In general, the savings tend to increase with an increase in the number of threads. Expectedly, the energy savings are relatively larger in benchmarks where performance slowdown due to running at a lower frequency is smaller (*canneal*, *streamcluster*). Similar observations can be made for other clock frequencies, though the range of energy savings varies with the clock frequency. Lowering clock frequency to the minimum F=1.6 GHz does not yield the best energy savings. For example, *x264* with N>1 requires the least energy when running at 2.0 GHz and the most when running at 3.4 GHz.

3.3 Energy-Time

So far we have discussed performance and energy as a function of the number of threads and processor frequency. The best performance is achieved when running at the highest clock frequency with the number of threads typically in range between N = 8 and N = 12. The best energy efficiency is typically achieved for the clock frequencies in range of F = 2.0 and F = 2.2 GHz. Depending on user needs, we may opt to optimize for either performance or energy. An interesting question arises when one wants to optimize for both the performance and energy. What are N and F that will provide the best performance at minimal energy?

Figure 7 shows an energy-time plot for *x264*. The y-axis represents the total energy in Joules, and the x-axis represents the execution time in seconds. We analyze benchmark runs when N = 4, N = 8, N = 12, and N = 16 for all frequencies supported on our machine. We can see that *x264* achieves the best performance when N = 12. For N = 12, lowering the processor clock frequency

Table 4. Energy (E) in Joules as a function of the number of threads (F=3.4 GHz).

Benchmark	Number of Threads															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
blackscholes	1992	1511	1357	1273	1238	1193	1157	1118	1130	1124	1129	1128	1124	1123	1122	1117
bodytrack	1804	1382	1242	1185	1158	1138	1113	1103	1101	1099	1096	1099	1098	1093	1097	1095
canneal	2499	2013	1847	1781	1785	1699	1641	1561	1676	1670	1665	1662	1660	1663	1691	1664
dedup	334	261	236	226	263	232	219	218	277	272	273	288	291	287	292	290
facesim	5045	3941	3578	3528	NA	3513	NA	3451	NA	NA	NA	NA	NA	NA	NA	3703
ferret	4162	3123	2771	2590	2508	2468	2448	2443	2437	2433	2428	2431	2430	2430	2431	2429
fluidanimate	3890	3060	NA	2713	NA	NA	NA	2584	NA	NA	NA	NA	NA	NA	NA	2768
freqmine	5651	4181	3778	3647	3394	3371	3319	3290	3298	3288	3306	3306	3304	3311	3307	3303
raytrace	2972	2444	2246	2152	2099	2058	2015	1988	1989	1984	1982	1978	1980	1989	1979	1979
streamcluster	4801	3612	3298	3108	2975	2771	2639	2526	3277	3263	3257	3272	3277	3328	3386	3299
swaptions	3280	2425	2136	1979	1979	1934	1872	1828	1853	1840	1845	1837	1848	1837	1839	1842
vips	1404	1042	926	855	831	824	828	835	827	845	832	836	844	830	828	829
x264	1608	903	821	790	791	768	744	729	729	729	728	728	728	729	727	728

from 3.4 GHz to 2.8 GHz increases execution time from 15.9 s to 18.1 s (or ~13% degradation) and decreases energy from 728.5 to 589.5 Joules (or ~19 % energy savings). Thus, an operating point of (N = 12, F = 2.8 GHz) is more desirable than (N = 12, F = 3.4 GHz).

Figure 8 shows the ratio of the energy-time product for single-threaded execution at maximum clock frequency (N=1, F=3.4) and the energy-time product when N and F are varied. We summarize this metric for all benchmarks using a geometric mean. When all PARSEC benchmarks are considered together, the optimal operating point when both energy and execution time are considered equally is at (N = 10, F = 2.8 GHz) or (N = 10, F = 3.1 GHz).

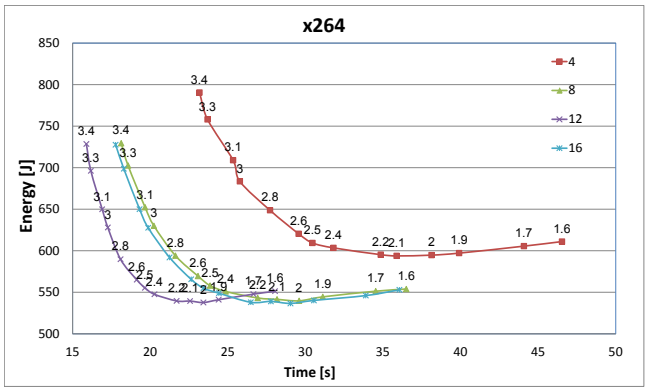

Figure 7. Energy-Time for x264.

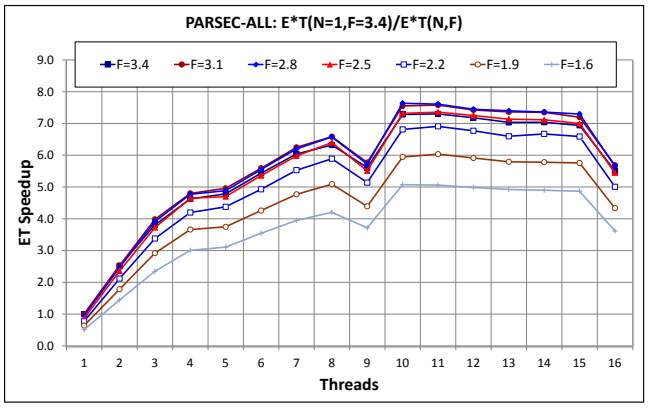

Figure 8. Energy-Time speedup for PARSEC.

4. CONCLUSIONS

Growing complexity of modern multicores that integrate a number of processor cores, hardware accelerators, on-chip interconnect, and cache hierarchies, poses a number of challenges to software developers who are responsible for tuning and optimization of applications. In addition to performance, power and energy used by an application become critical in many application uses. A number of new power management schemes and tools are introduced to support software developers to estimate power and energy.

In conditions when a user can vary a number of parameters such as processor core clock frequency and the number of threads in a parallel benchmark, several practical questions arise: what is the optimal number of threads to achieve the best performance, what is the optimal clock frequency to achieve the best energy-efficiency, and how these two can be optimized together.

In this paper we presented the results of our measurement-based experimental study aimed at quantifying the impact of frequency scaling and thread scaling on performance and energy. We measured execution time and energy consumed by an Intel Xeon 1240 v2 microprocessor while executing the Parsec benchmarks. We find that maximum performance is achieved when the number of threads matches or slightly exceeds the number of logical processors ($8 \le N \le 12$) and the clock frequency is at maximum (F = 3.4 GHz). The minimum energy is consumed when clock frequency is in range $2.0 \le F \le 2.4$ GHz. Finally, we find that an optimal Energy-Performance ratio is achieved when $8 \le N \le 12$ and $2.8 \le F \le 3.0$ GHz.

5. ACKNOWLEDGMENTS

This work has been supported in part by National Science Foundation grants CNS-1205439 and CNS-1217470.

6. REFERENCES

[1] "CPU Frequency Scaling - ArchWiki." [Online]. Available: https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling. [Accessed: 19-Dec-2013].

[2] "Intel® 64 and IA-32 Architectures Software Developer Manuals," *Intel*. [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html. [Accessed: 21-Dec-2013].

[3] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20 –27, Apr. 2012.

[4] "PAPI." [Online]. Available: http://icl.cs.utk.edu/papi/index.html. [Accessed: 20-Dec-2013].

[5] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring Energy and Power with PAPI," in *2012 41st International Conference on Parallel Processing Workshops (ICPPW)*, 2012, pp. 262–268.

[6] "likwid - Lightweight performance tools - Google Project Hosting." [Online]. Available: http://code.google.com/p/likwid/. [Accessed: 08-Dec-2013].

[7] J. Treibig, G. Hager, and G. Wellein, "LIKWID: Lightweight Performance Tools," *arXiv:1104.4874 [cs]*, pp. 207–216, Sep. 2010.

[8] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments," *arXiv:1004.4431 [cs]*, Apr. 2010.

[9] C. Bienia, "Benchmarking Modern Multiprocessors," Princeton University, Princeton, NJ, USA, 2011.

[10] "Perf Wiki." [Online]. Available: https://perf.wiki.kernel.org/index.php/Main_Page. [Accessed: 07-Dec-2013].