Evaluation of the Intel® Core™ i7 Turbo Boost feature

James Charles, Preet Jassi, Ananth Narayan S, Abbas Sadat and Alexandra Fedorova

***Abstract*—The Intel® Core™ i7 processor code named *Nehalem* has a novel feature called Turbo Boost which dynamically varies the frequencies of the processor's cores. The frequency of a core is determined by core temperature, the number of active cores, the estimated power and the estimated current consumption. We perform an extensive analysis of the Turbo Boost technology to characterize its behavior in varying workload conditions. In particular, we analyze how the activation of Turbo Boost is affected by inherent properties of applications (i.e., their rate of memory accesses) and by the overall load imposed on the processor. Furthermore, we analyze the capability of Turbo Boost to mitigate Amdahl's law by accelerating sequential phases of parallel applications. Finally, we estimate the impact of the Turbo Boost technology on the overall energy consumption. We found that Turbo Boost can provide (on average) up to a 6% reduction in execution time but can result in an increase in energy consumption up to 16%. Our results also indicate that Turbo Boost sets the processor to operate at maximum frequency (where it has the potential to provide the maximum gain in performance) when the mapping of threads to hardware contexts is sub-optimal.**

I. Introduction

The latest multi-core processor from Intel code named Nehalem [9] has a unique feature called Turbo Boost Technology [10]. With Turbo Boost, the processor opportunistically increases the frequency of the cores based on the core temperature, the number of active cores, the estimated current consumption, and the estimated power consumption. Normally, the Core i7 processor can operate at frequencies between 1.5 GHz and 3.2 GHz (the maximum non-Turbo Boost frequency or the base frequency) in frequency steps of 133.33 MHz. With Turbo Boost enabled, the processor can increase the frequency of cores two further levels to 3.3 GHz and then 3.4 GHz. We refer to the first frequency above the base frequency as the *lower Turbo Boost frequency* (3.3 GHz) and to the maximum frequency as the *higher Turbo Boost frequency* (3.4 GHz). If multiple physical cores are active, only the lower Turbo Boost frequency is available.

Turbo Boost is made possible by a processor feature named power gating. Traditionally, an idle processor core consumes zero active power while still dissipating static power due to leakage current. Power gating aims to cut the leakage current as well, thereby further reducing the power consumption of the idle core. The extra power headroom available can be diverted to the active cores to increase their voltage and frequency without violating the power, voltage, and thermal envelope.

James Charles {jac27@cs.sfu.ca}, Preet Jassi {preetj@cs.sfu.ca}, Ananth Narayan S {ans6@cs.sfu.ca}, Abbas Sadat {sas21@cs.sfu.ca}, and Alexandra Fedorova {fedorova@cs.sfu.ca} are with the School of Computing Science, Simon Fraser University, Canada.

Turbo Boost Technology essentially makes the Nehalem a dynamically *asymmetric* multi-core processor (AMP); cores use the same instruction set but their frequency can vary independently and dynamically at runtime.

We perform a detailed evaluation of the Turbo Boost feature with the following goals:

1) To understand how Turbo Boost behaves depending on the properties of the application such as its degree of CPU or memory intensity,
2) To find how system load, specifically the number of threads running concurrently, affects when and how often Turbo Boost gets engaged, and finally,
3) To determine how scheduling decisions that distribute load in a processor affect the potential performance improvements offered by Turbo Boost.

To this end, we select benchmark applications from the SPEC CPU2006 benchmark suite with diverse qualities (integer versus floating point applications, memory-intensive versus computationally-intensive applications). We run benchmarks individually and in groups while monitoring system performance with and without the Turbo Boost feature. The results of our study will be useful to both CPU designers as they demonstrate the benefits and costs of Turbo Boost technology, and to software designers as they will provide insight into the benefits of this technology for applications.

Prior work has shown that such a processor configuration offers higher performance per watt in most situations when compared with symmetric multi-core processors [12], and a great deal of other work has analyzed the performance, versatility, and energy-efficiency of AMP systems either theoretically or through simulation [2], [8], [12], [15], [18].

Prior work from Intel [2] has shown that such a processor can be leveraged to mitigate Amdahl's law for parallel applications with sequential phases. Amdahl's law states that the speedup of a parallel application is limited by its sequential component. A typical parallel application might divide a computational task into many threads of execution executing in parallel, and then aggregate the results using only a single thread. This division of work results in an execution pattern where parallel phases of execution are interspersed with sequential "bottleneck" phases. A dynamically asymmetric processor can accelerate such bottleneck phases while staying within its energy budget.

When a program enters a sequential phase, the processor would automatically turn off idle cores and boost the frequency on the active core. When the program returns to the parallel phase, all the cores would be activated, but the frequency of each core would be reduced. The benefits of such an architecture are demonstrated by Annavaram et al. [2]. They observe performance improvements of as much as 50% relative to symmetric systems using a comparable energy budget. Nehalem, with its Turbo Boost feature has the potential to mitigate Amdahl's law for parallel applications with sequential phases, therefore we evaluated this capability using several parallel applications from the PARSEC [5] and BLAST [1] benchmark suites.

Our results demonstrate that Turbo Boost increases performance of applications by up to 6%, but the benefit depends on the type of application and on the processor load. Memory-intensive applications (i.e., those with a high rate of requests to main memory) in general experience smaller performance improvements than CPU-intensive applications. Turbo Boost is engaged less often when a large number of cores is busy as opposed to when the number of busy cores is small. Interestingly, Turbo Boost engages more frequently when the mapping of threads to cores is not optimal with respect to resource contention: that is, given two thread mappings, the assignment with greater contention for shared resources is also the one where the Turbo Boost feature will be activated more frequently. As to mitigating Amdahl's law, we found that while Turbo Boost does respond to transitions into sequential phases by boosting the processor frequency, the frequency increase is not large enough to deliver benefits similar to those demonstrated in previous work.

The rest of the paper is organized as follows. We discuss our experimental methodology in Section II. We discuss our experimental configuration and results in Section III. We evaluate energy consumption in Section IV, and summarize our conclusions in Section VI.

II. Methodology

We run four sets of experiments for this study: the *Isolation Tests*, the *Paired Benchmark Tests*, the *Saturation Tests*, and the *Multi-Threaded Tests*.

A. Isolation Tests

In this set of experiments we run individual applications from the SPEC CPU2006 suite with Turbo enabled and with Turbo disabled, and measure the performance improvements from Turbo Boost. According to prior work, applications differ in their *sensitivity* to the changes in frequency (i.e., how much their performance improves as the processor frequency is increased) [15]. The sensitivity is determined by the application's *CPU-intensity* or *memory-intensity*. *CPU-intensive* applications are those that spend most of their time executing instructions on the CPU and have a low last level cache (LLC) miss rate. Conversely, *memory-intensive* applications experience a high LLC miss rate and thus spend more time waiting for data to be fetched from memory. As a result of spending more time on the CPU, CPU-intensive applications are more sensitive to changes in CPU frequency than memory-intensive applications.

Applications can be categorized as CPU-intensive or memory-intensive by examining their LLC miss rate. We characterized all the applications in the SPEC CPU2006 benchmark suite by running each in isolation on a Nehalem processor and measuring the LLC miss rate (in this case, the L3 cache miss rate). From this, we were able to classify applications according to the categories given in Table I. In the isolation tests we analyze whether there is a relationship between the speedup derived from the Turbo Boost feature and the application's LLC miss rate.

TABLE I
Application Categories

Identifier	Memory performance	Calculation Type
MF	Memory-intensive	Floating point
MI	Memory-intensive	Integer
CF	CPU-intensive	Floating point
CI	CPU-intensive	Integer

B. Paired Benchmark Tests

We run pairs of benchmarks to determine if the processor could still make effective use of the Turbo Boost feature with more than one application running in the system. This provides insight into the effects of running different types of applications with each other, and also into the interplay between Turbo Boost and contention for shared resources when multiple applications are running concurrently.

From the SPEC CPU2006 suite, we choose two groups pf applications, with four applications each. We then run pairs of benchmarks within each set on the hardware contexts of the same physical core and on different physical cores, with and without Turbo Boost enabled. Our goal is to analyze how Turbo Boost engages in these different configurations.

C. Saturation Tests

The Nehalem processor has four cores, each with two thread contexts (see Section III). The saturation tests are designed to identify whether Turbo Boost activates while all threads contexts are busy. To do this, we saturate all of the cores with applications of various types and execute them with and without Turbo Boost enabled.

We saturate the system using three different loads: (1) a CPU-intensive load where an instance of a CPU-intensive application is bound to each logical processor, (2) a corresponding memory-intensive load, (3) a mixed load, with four CPU-intensive and four memory-intensive applications.

The saturation tests show if there is a relationship between the type of the load and the corresponding performance improvements from Turbo Boost. We expect that Turbo Boost will activate less frequently under the CPU-intensive load, because this load will cause the chip to operate at a higher temperature compared to a memory-intensive workload.

D. Multi-Threaded Tests

As described in Section I, dynamic AMP processors such as Nehalem have the potential to mitigate Amdahl's law for parallel applications with sequential phases. To test if Turbo Boost is responsive to phase changes in applications and, more significantly, if it can engage to accelerate the sequential phases of parallel code, we perform multi-threaded tests.

We execute multi-threaded applications drawn from the PARSEC [5] and BLAST [1] benchmark suites with and without Turbo Boost enabled. We monitor the frequency and utilization of each core during the execution. If all but one cores have 0% utilization, the application is deemed to be in a sequential phase. Likewise, parallel phases can be clearly seen when several (potentially all) cores are active. The multi-threaded applications are executed such that they use up to eight threads to match the number of thread contexts available. From the time series of history data, we can determine whether a particular core is operating at one of the Turbo Boost frequencies. Over the course of a benchmark, this data reveals how Turbo Boost responds to changes in CPU utilization as well as how Turbo Boost augments the performance of multi-threaded workloads.

III. Experimental Setup and Results

The experiments are executed on an Intel Core i7 965 (Extreme Edition) with 3GB DDR3 RAM, running the Linux 2.6.27 kernel (Gentoo distribution). The Core i7 965 is a quad core processor with 2 simultaneous multi-threading (SMT) contexts per core. This provides for 8 logical cores. Figure 1 shows the physical layout of the cores on the Nehalem processor. The highest non-Turbo frequency of the Core i7 is 3.2 GHz. The two supported Turbo Boost frequencies are 3.3 GHz and 3.4 GHz. Core frequency was obtained by implementing the frequency calculation algorithm described in [10]. This algorithm can be summarized with these steps:

1) The base operating ratio is obtained by reading the `PLATFORM_INFO` Model Specific Register (MSR). This is multiplied by the bus clock frequency (133.33 MHz) to obtain the base operating frequency.
2) The Fixed Architectural Performance Monitor counters are enabled. Fixed Counter 1 counts the number of core cycles while the core is not in a halted state (`CPU_CLK_UNHALTED.CORE`). Fixed Counter 2 counts the number of reference cycles when the core is not in a halted state (`CPU_CLK_UNHALTED.REF`).
3) The two counters are read at regular intervals and the number of unhalted core cycles and unhalted reference cycles that have expired since the last iteration are obtained. The core frequency is calculated as $F_{current} =$ Base Operating Frequency * (Unhalted Core cycles / Unhalted Reference Cycles). This is repeated for each core.

Core temperature is obtained by reading the `IA32_THERM_STATUS` MSR. Both temperature and frequency are measured on a per-physical-core basis. For all the experiments, applications are executed four times: the first run is discarded and results from the remaining runs are averaged. The standard deviation of the measurements was negligible.

A. Isolation tests

We run all SPEC CPU2006 benchmarks on a single core in isolation with and without Turbo Boost. The Turbo Boost frequency scaling algorithm takes into account the number of active cores when determining the frequency of a core. Thus, we expect that the active core will spend the majority of its time at the higher Turbo frequency as only one core is active. Furthermore, we expect that CPU-intensive applications will obtain a greater speedup compared to the memory-intensive applications as changes in clock frequency alter the performance of CPU-intensive applications more than memory-intensive applications—that is, CPU-intensive applications are more *sensitive* to changes in the clock frequency compared to memory-intensive applications.

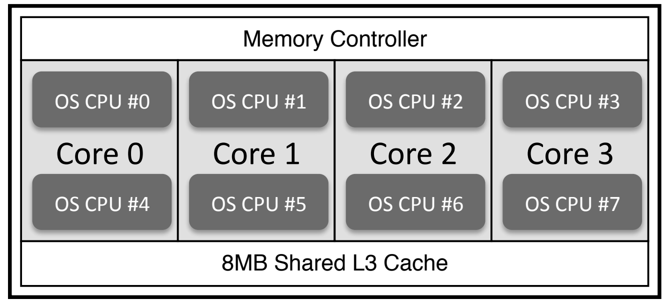


Fig. 1. Nehalem Layout

Figure 2 captures the percentage reduction in execution time seen per benchmark against the last level cache (LLC) miss rate, which, as explained earlier, determines the memory intensity of applications. The figure shows that, as expected, applications with a higher cache miss rate receive a smaller speedup due to the increase in frequency. The only outlier to this trend is MCF which exhibited close to 4% speedup despite having a high LLC miss rate.

When the benchmarks run in isolation, they spend at least 80% of execution time at the higher Turbo frequency but execute almost entirely at the Turbo frequencies. Once again, this behavior is expected. Figure 3 shows the distribution of the time spent at the different frequencies for all the SPEC CPU2006 benchmarks.

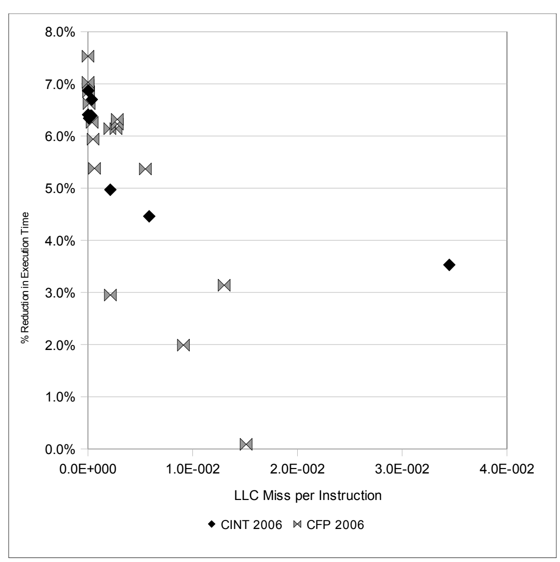


Fig. 2. Percentage Speedup versus LLC Miss rate

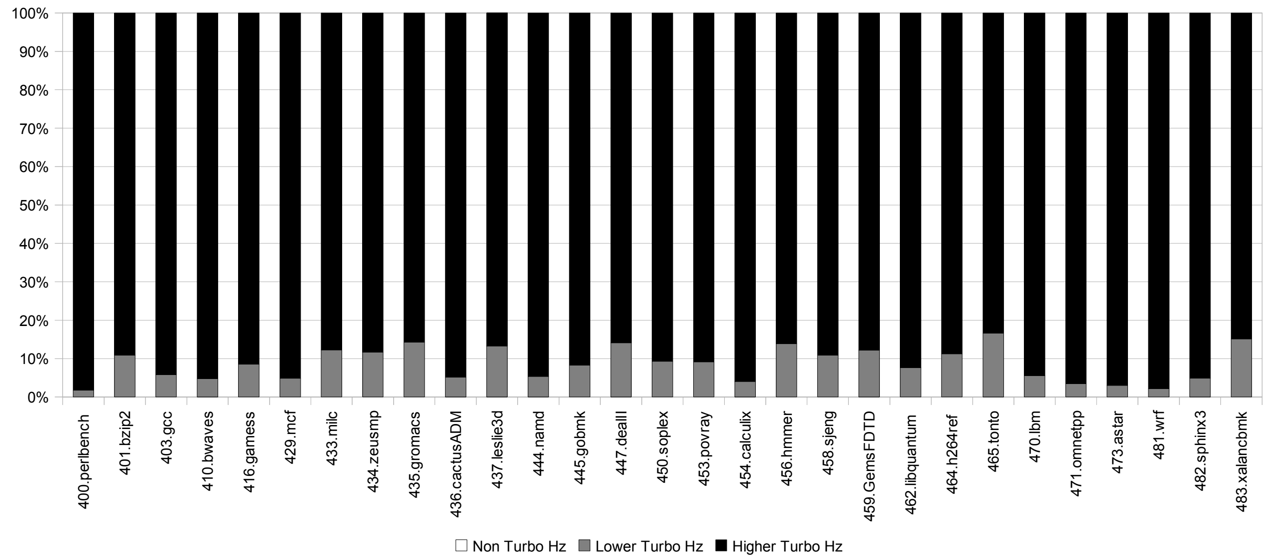


Fig. 3. The distribution of time spent at various frequencies for all SPEC CPU2006 benchmarks running in isolation

Finally, we analyze the speedup from the Turbo feature according to the application type. We classify all these applications according to the categories given in Table I. Applications with an LLC miss rate below the median miss rate are considered CPU-intensive. Those above and including the median are considered memory-intensive. Table II shows the average speedup for each class resulting from Turbo Boost. The CPU-intensive benchmarks receive a greater speedup in comparison to the memory-intensive benchmarks.

TABLE II
ISOLATION RESULTS

Benchmark Class	Speedup
MF	4.5%
MI	4.3%
CF	6.9%
CI	6.5%

B. Paired Benchmarks Tests

For this set of experiments, we select a subset of the SPEC CPU2006 benchmarks and construct two sets. We restrict ourselves to a subset of SPEC CPU2006 applications to keep the number of experiments feasible. We pick two sets of applications with each set containing four applications, one of each category MF, MI, CF, and CI (Table I). Within each category, applications were selected randomly. The two sets of application are shown in Table III.

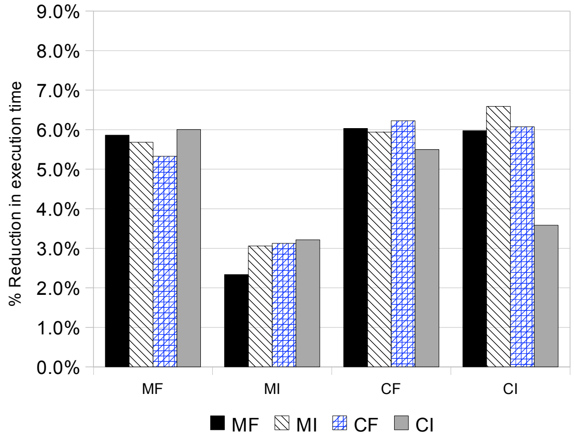
For each application set, we run all possible pairs of the four applications using one pair per experiment. First, the applications in a pair are executed affinitized on the same physical core; then the applications in the pair are affinitized to different physical cores. We repeat each experiment with and without Turbo Boost enabled. For each test, one application is identified as the principal application and the second as the interfering application. The interfering application is restarted if it completes prior to the principal application. Between successive executions of the principal application, a two minute idle time is introduced. The idle time allows for the processor to cool and reach a steady temperature.

TABLE III
BENCHMARK SETS FOR PAIRED BENCHMARK TESTS

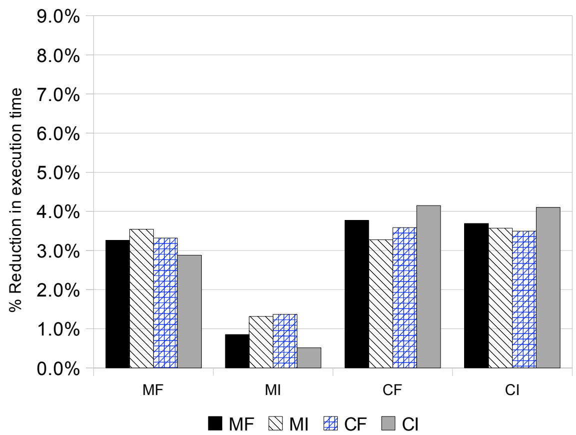
Classification	Set 1	Set 2
MF	Leslie3D	Namd
MI	Omnetpp	Astar
CF	Povray	Bwaves
CI	H264	Hmmer

Figure 4 and Figure 5 show the percentage speedup due to enabling Turbo Boost for Set 1 and Set 2 respectively. The principal application is on the abcissa of the graph while the interfering application is denoted by the shading of the bars. Thus, each bar shows the percent speedup of the principal application when paired with an interfering application. Figures 4(a) and 5(a) shows the percent speedup due to Turbo Boost when the application are assigned to the same core. Figures 4(b) and 5(b) capture the percent speedup due to Turbo Boost when the application are assigned to the different cores.

To analyze how the effect of Turbo is determined by the type of application, we average the speedups resulting from Turbo Boost across the different categories of benchmarks namely CPU-intensive (C) and memory-intensive (M). Table IV shows the average increase in performance for the various combinations of benchmark classes as well as the average *degradation* of performance resulting from scheduling the benchmark in the respective configuration. The degradation is calculated by normalizing the execution time of the principal application by the execution time of the principal application when it is run in isolation (with Turbo Boost enabled in both cases). A degradation of 1.0 implies that the application completed in the same time when executed standalone and

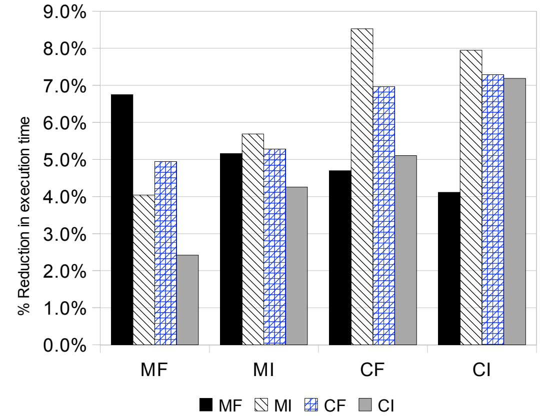


(a) Speedup for Set 1 due to Turbo (Same Core)

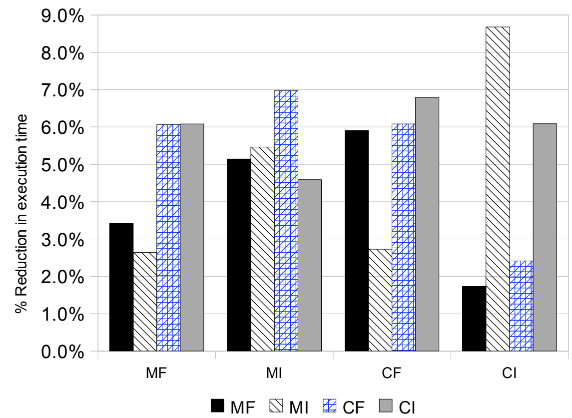


(b) Speedup for Set 1 due to Turbo (Different Cores)

Fig. 4. Speedup due to Turbo - Set 1



(a) Speedup for Set 2 due to Turbo (Same Core)



(b) Speedup for Set 2 due to Turbo (Different Cores)

Fig. 5. Speedup due to Turbo - Set 2

with interference.

Table IV shows that the greatest speedup from Turbo Boost is seen when two CPU-intensive benchmarks are bound to the same core while the smallest speedup from Turbo Boost is when two memory-intensive benchmarks are bound to different cores. Additionally, Table IV shows that Turbo Boost provides a greater performance gain when the applications are run on the same core compared to when they are run on different cores. However, the degradation is significantly larger when the benchmarks are bound to the same core compared to when they are bound to different cores. Therefore, in the case of executing two benchmarks, Turbo Boost provides the maximum gains in performance when the degradation resulting from scheduling is the largest. In other words, in the case of executing two benchmarks, Turbo Boost provides higher gains in performance when the mapping of threads to cores is sub-optimal in terms of contention.

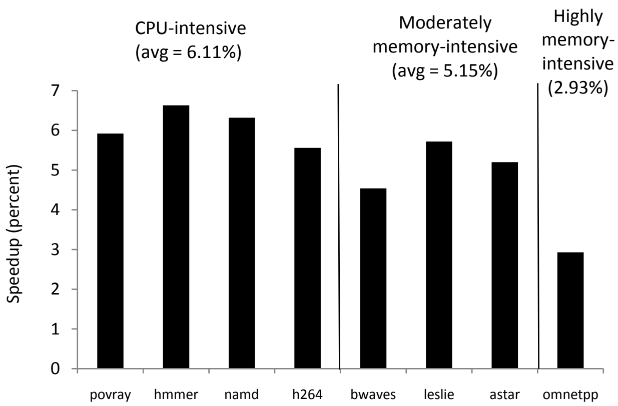
Figures 4(a)-5(b) show that applications experience varying amounts of speedup when executed with different co-runners. For example, the speedup obtained by *omnetpp*, which has the highest LLC miss rate in Set 1 is much lower compared to the rest of the applications in its set. However, we do not see such a clear behaviour exhibited by *astar* which exhibits the highest LLC miss rate in Set 2.

TABLE IV
AVERAGE DEGRADATION AND AVERAGE SPEEDUP

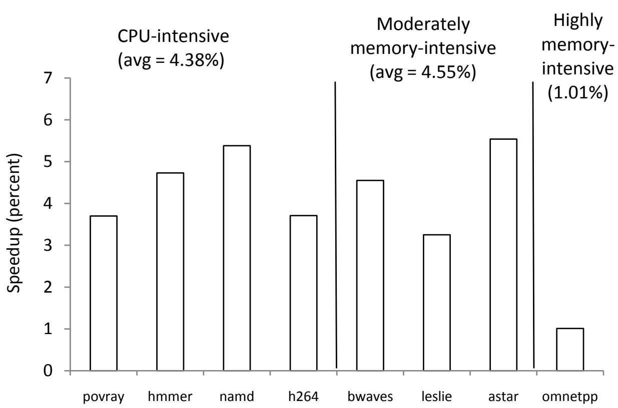
	Same Core		Different Core	
	Set 1			
	Degradation	Speedup	Degradation	Speedup
CC	1.6	5.4%	1.0	3.8%
CM	1.4	5.3%	1.0	2.8%
MM	1.4	4.2%	1.1	2.2%
	Set 2			
	Degradation	Speedup	Degradation	Speedup
CC	1.3	6.6%	1.0	5.3%
CM	1.3	4.4%	1.0	5.3%
MM	1.3	5.4%	1.0	4.2%

Our initial classification of applications (Table I) does not provide sufficient information to analyze these results. To explain this data, we use a more fine-grained classification (based on LLC miss rate) and group the applications into three clusters: CPU-intensive, moderately memory-intensive and highly memory-intensive. We visually determine the cluster to which an application belongs using the clustering seen in Figure 2. We plot the average speedup and effective frequency seen by the applications in Figures 6 and 7 respectively.

Figures 6(a) and 6(b) show the percentage speedup experienced by individual applications when executed with the

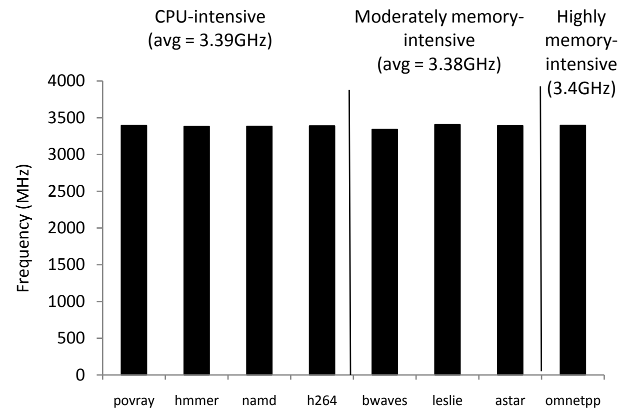


(a) Speedup on Same Core Config

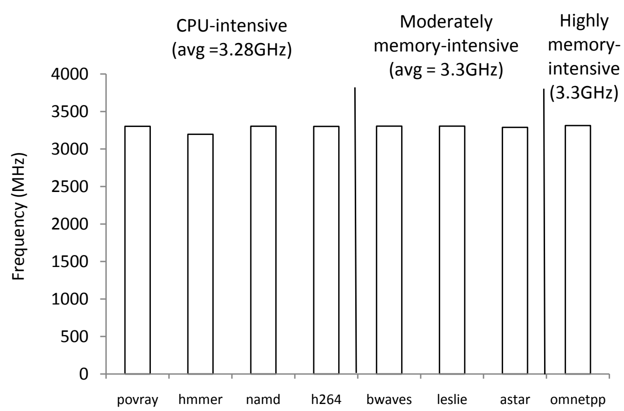


(b) Speedup on Diff Core Config

Fig. 6. Average application speedup



(a) Frequency on Same Core Config



(b) Frequency on Diff Core Config

Fig. 7. Effective frequency

interfering application on the same core (henceforth referred to as same-core config) and different cores (henceforth referred to as different-core config). The speedup shown in the figure is calculated by averaging the speedup seen across all co-run pairs. As expected, the highly memory-intensive applications experience the least speedup. Figures 7(a) and 7(b) show the effective frequency that the applications execute at. We calculate the effective frequency using the percentage of execution time spent by the application at each frequency. These figures show that the frequencies are 2-3% lower in the different-core configs. When only one core is active (as in the same-core config) the processor is able to run at a higher frequency than when two cores are active (as in the different-core config).

In the different-core config, the moderately memory-intensive applications experience roughly the same average speedup as the CPU-intensive applications. This is due to the fact that the moderately memory-intensive applications run at a relatively higher frequency in the different-core config than the CPU-intensive applications. Looking at the relative change in the effective frequency between the same-core config and different-core config, we see that the moderately memory-intensive applications experience a smaller frequency degradation than the CPU-intensive applications. For the moderately memory-intensive applications, the frequency drops by 2.4% between the same-core and different-core config. For the CPU-intensive applications, the drop is 3.4%. The CPU-intensive applications experience a lower Turbo speedup relative to moderately memory-intensive applications because they execute at lower frequencies.

We undertake a regression analysis to identify the correlation between application properties and the speedup they exhibit in the various configurations. Speedup is the dependent variable; application type (Floating Point vs. Integer), LLC miss rate, temperature, and effective frequency are the independent variables. When the differences in the degree of memory-intensity are large, the LLC largely determines the speedup from Turbo. However, when these differences are smaller (as between the CPU-intensive and moderately memory-intensive applications), the frequency at which the applications run is the main determinant. We also analyzed whether the properties of the application determine the frequency at which the processor runs. We found low correlation between the application characteristics that we measured (LLC, Floating Point or Integer type) and the effective frequency. However, we believe that further investigation is required to understand the effect of application properties on frequency.

C. Saturation Tests

We saturate all of the cores so that there is a benchmark application bound to each SMT context on the processor. There are four configurations, each used with and without Turbo Boost. In the first configuration, all eight cores are loaded with the same CPU-intensive benchmark (denoted by CC CC CC CC). In the second configuration, the eight cores are loaded with the same memory-intensive benchmark (denoted by MM MM MM MM). Finally, in the last two configurations, we load the SMT contexts with four CPU-intensive (C) benchmarks and four memory-intensive (M) benchmarks in MM MM CC CC and CM CM CM CM assignments. In the MM MM CC CC assignment, each memory-intensive application shares a core with another memory-intensive application, while in the CM CM CM CM assignment, memory-intensive applications share cores with CPU-intensive applications. We run the saturation tests on two different sets of benchmarks where Set 1 is composed of the CPU-intensive *povray* and the memory-intensive *leslie3d* benchmarks while Set 2 is composed of the CPU-intensive *h264* and the memory-intensive *omnetpp* applications. We need to note here that the Set 1 and Set 2 are different from the ones used for the paired benchmark tests described in Section III-B.

The Turbo Boost frequency scaling algorithm takes into account the number of active cores and core temperature when determining the frequencies of the cores. For the saturation tests we expect that the cores would spend the majority of their execution time at non-Turbo frequencies. This is because all of the cores are active. Additionally, we expect that the processor temperature will be high due to all of the cores being active. This will also cause the frequency scaling algorithm to lower the frequencies of the cores.

Figure 8 shows the frequency distribution of the cores when the saturation tests are executed. The results of the saturation tests show that in all configurations, the cores spent the majority of their time at the lower Turbo frequency. This is against our prediction that the cores would operate mostly at non-Turbo Boost frequencies. In all configurations, Core 2 (the third physical core) operates at the higher Turbo frequency for a portion of time, but all cores spend the majority of their time (at least 90%) at the lower Turbo frequency and some time at other non-Turbo frequencies. Additionally, the behaviour of the cores is not affected by the types of applications that were bound to them – cores running memory-intensive benchmarks did not operate at higher frequencies compared to the cores running CPU-intensive benchmarks. At the same time, we see that a core that has two CPU-intensive applications bound to it spends more time at non-Turbo frequencies compared to the other configurations.

Table V shows the execution times of the benchmarks with and without Turbo Boost and Table VI shows the resulting speedup from enabling Turbo Boost for the different configurations. The MM MM CC CC configuration yields the largest speedup when Turbo is enabled compared to all other configurations. The CC CC CC CC configuration also receives a significant speedup. In addition to the execution times, the table also captures the normalized average speedup (marked N.Avg). We normalize the execution time of each benchmark to its execution time in the MM MM CC CC configuration. The resultant values are then averaged to obtain the normalized average speedup.

The normalized average speedups shown in Table V reveal that applications execute in lesser time in the CM CM CM CM configuration compared to the MM MM CC CC configuration (for both sets of applications and irrespective of Turbo Boost) and is also the optimal assignment of the four memory-intensive and four CPU-intensive benchmarks. Applications in Set 1 get a 10% speedup in the CM CM CM CM configuration (without Turbo Boost) and 6% (with Turbo Boost) compared to the MM MM CC CC configuration. For Set 2, the speedups are 8% and 5% respectively. The second set of applications have a lower LLC miss profile, so isolating the memory intensive applications from each other (as in the CM CM CM CM) configuration is less important.

TABLE V
EXECUTION TIMES (IN SECONDS)

	Without Turbo			With Turbo		
	Set 1					
	C	M	N.Avg	C	M	N.Avg
CC CC CC CC	408.2	–	–	392.3	–	–
MM MM CC CC	414.2	1176.0	1	396.7	1116.0	1
CM CM CM CM	408.2	1065.5	0.9	393.7	1034.0	0.94
MM MM MM MM	–	1400.0	–	–	1394.7	–
	Set 2					
	C	M	N.Avg	C	M	N.Avg
CC CC CC CC	149.8	–	–	142.8	–	–
MM MM CC CC	154.0	662.0	1	145.4	650.3	1
CM CM CM CM	120.5	679.5	0.92	116.3	673.8	0.95
MM MM MM MM	–	831.1	–	–	824.8	–

Interestingly, the speedups of the configurations in Table VI reveal that the MM MM CC CC configuration receives a greater speedup from Turbo Boost compared to the CM CM CM CM configuration. This difference in the speedup due to Turbo Boost versus the speedup due to optimal thread scheduling carries an important implication for designers of thread schedulers for systems like Nehalem: attempts to maximize the activation of the Turbo Boost feature must be carefully weighed against the possible effects of sub-optimal scheduling.

Finally we note that the speedup for memory-intensive applications is smaller in the MM MM MM MM configuration relative to mixed CM configurations, because each memory-intensive application experiences a higher contention for the shared cache in the MM MM MM MM configuration relative to mixed CM configurations. As a result, memory-intensive applications become even less sensitive to changes in CPU frequency and experience a smaller speedup from Turbo Boost.

D. Multi-threaded Workloads

A selection of benchmarks from the PARSEC benchmark suite [5] as well as a small set of queries (tblastx) for the

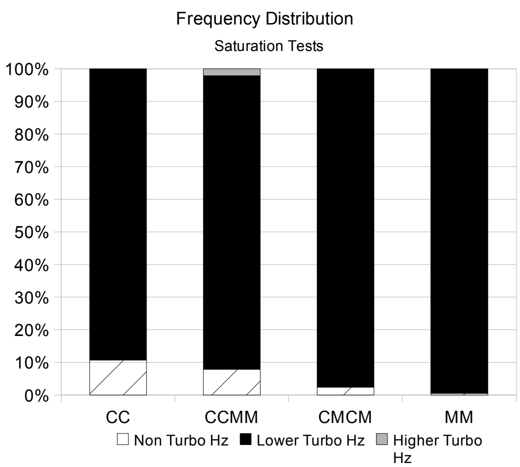


Fig. 8. Distribution of time spent at various frequencies for saturation tests (Set 1)

TABLE VI
SPEEDUP FOR SATURATION TEST 1

Configuration	% Speedup	
	Set 1	
	CPU-Intensive	Memory-Intensive
CC CC CC CC	4.0%	–
MM MM CC CC	4.4%	5.4%
CM CM CM CM	3.6%	3.0%
MM MM MM MM	–	0.3%
	Set 2	
	CPU-Intensive	Memory-Intensive
CC CC CC CC	4.9%	–
MM MM CC CC	5.9%	1.8%
CM CM CM CM	3.6%	0.8%
MM MM MM MM	–	0.7%

BLAST bio-informatics tool [1] are executed with Turbo enabled; each is spawned so as to have up to eight running threads. Based on prior characterization of these benchmarks [1], [5] we can roughly assess the extent to which these benchmarks are parallelized. Table VII shows this classification as well as the impact of Turbo Boost on the execution time of these benchmarks; the applications certainly benefit from Turbo. It is more interesting, however, to look at the CPU utilization over time for these benchmarks to understand from where these performance gains are coming. It is not possible to show all the data here so instead we will highlight data from the BLAST bio-informatics sequencing tool which show an exemplary range of behaviours.

TABLE VII
SPEEDUP DUE TO TURBO FOR MULTI-THREADED BENCHMARKS SUITES

Benchmark	Degree of Parallelization	Speedup
Blast	limited by serial sections	4.5%
Blackscholes	highly parallel	4.8%
Ferret	highly parallel	5.4%
Swaptions	highly parallel	4.2%
X264	limited by serial sections	2.7%

Figure 9 shows the diverse range of behaviour during the execution of BLAST. While all the cores start at 100% utilization and run at the lower Turbo frequency, Core 0 drops to 50% utilization while the other cores drop to 0% utilization. 50% utilization on a single core is actually indicative of a sequential phase. Because there are two thread contexts on each physical core, the maximum core utilization during a sequential phase is 50%. Core 0 transitions to the higher Turbo frequency during this sequential portion. We observe that among the four physical cores, physical Core 1 spends largest percentage of time executing sequential code portions. Yet, the frequency on Core 1 does not reach the higher Turbo frequency during all the sequential phases; nor does it stay at higher Turbo frequency for the entire duration of the sequential phase. We also observe that the other cores continue to operate at the lower Turbo frequency despite having 0% utilization. We expected the frequency of idle cores to reduce to 1.5 GHz during the sequential phases but the *on-demand* governor on Linux is not aggressively making this adjustment.

From these results, we can conclude that Turbo Boost is sensitive to changes in load which enables it to accelerate sequential phases of the code. However, software power manager is not aggressive enough at reducing the frequency of idle cores to enable frequent and extended activation of Turbo. Furthermore, the speedups from acceleration of sequential phases on fast cores are smaller than that reported by the previous study [2] because the Turbo Boost frequency increase (relative to the maximum non-Turbo frequency) is much smaller than the frequency differential in that study.

IV. ENERGY CONSIDERATIONS

When Turbo Boost is enabled applications experience a boost in performance. However, what is the cost of this performance improvement? To answer this question, we derive a power metric using the time spent at the different frequencies when Turbo Boost is enabled. We derive such a metric as we do not have the equipment to measure power consumption. The metric is motivated by a simple power model. Note that we call the metric the *energy consumption* of the processor; however, this value is not the actual energy consumption.

Processor power consumption is given by

$$P = \alpha C(V + I_l)Vf \qquad (1)$$

where α is the activity factor, C is the capacitance, V is the supply voltage, f is the operating frequency, and I_l is leakage current.

We conservatively assume that C and I_l are constant, and we set α to 1. We define the power of the processor when operating at the base frequency of 3.2 GHz, that is, the highest frequency that the processor operates at when Turbo Boost is disabled.

$$P(f_{base}) = \alpha C(V + I_l)Vf = C(V + I_l)Vf_{base} \qquad (2)$$

Intel processors require 50mV additional voltage to operate 133.33 MHz faster [16]. Then, the power at Turbo Boost frequencies can be obtained by the equation :

$$P(f) = \frac{(V' + I_l)V'f}{(V + I_l)Vf_{base}} \qquad (3)$$

where $V' = V + k * .05$, k being number of steps above f_{base}.

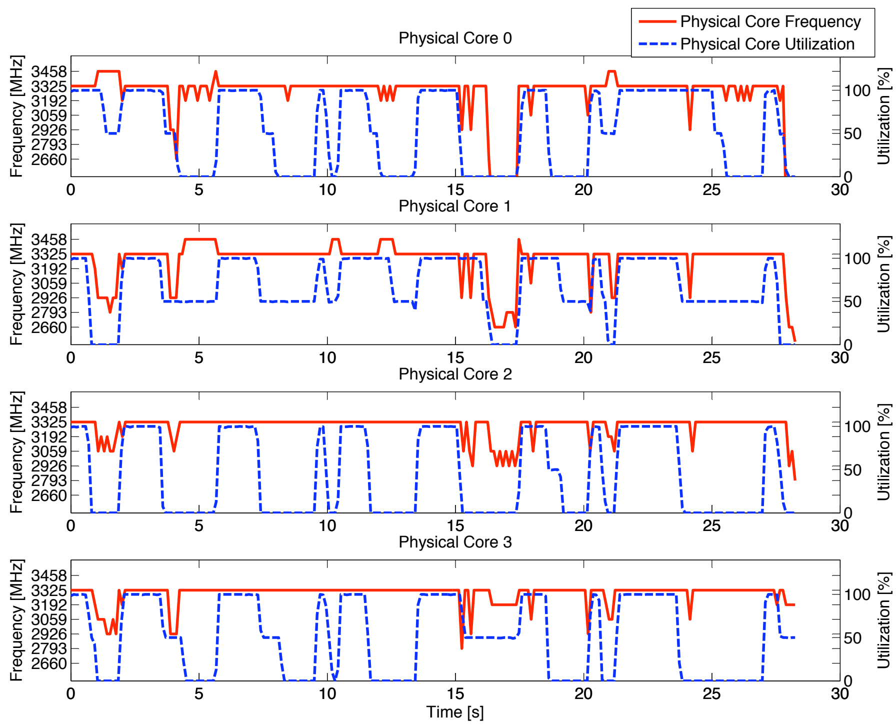


Fig. 9. Physical core frequency and utilization during a single execution of the BLAST bio-informatics benchmark. Clear sequential phases are seen during execution.

TABLE VIII
PERCENT INCREASE IN ENERGY IN ISOLATION TESTS FROM ENABLING TURBO BOOST

Benchmark	Cost
MF	13.9%
MI	13.7%
CF	13.9%
CI	14.6%

Energy is given as $Power * Time$. Therefore, to obtain the energy we multiply the power at the different frequencies by the time spent by the application at the various frequencies. For example, the time spent at the base frequency when Turbo Boost is disabled is 100% and the power of the processor is 1, therefore, the total energy consumption is 100 units. These assumptions are reasonable as we are not interested in the exact value of the energy that is consumed but rather in the energy consumption relative to the base frequency. We use abstract units instead of Watts to emphasize that this is a modeled value and not a measured value. To obtain the total energy consumption across the processor, we sum up the power consumption for each individual core as determined by Equation 3 multiplied by the time spent at the various frequencies.

TABLE IX
PERCENT INCREASE (AVERAGE) IN ENERGY IN PAIRED TESTS FROM ENABLING TURBO BOOST

	Same Core	Different Core
CC	15.7%	10.6%
CM	15.9%	11.2%
MM	16.6%	11.3%

TABLE X
PERCENT INCREASE (AVERAGE) IN ENERGY IN SATURATION FROM ENABLING TURBO BOOST

	Set 1	Set 2
CC CC CC CC	5.1%	9.0%
CC CC MM MM	12.3%	11.6%
CM CM CM CM	9.0%	9.9%
MM MM MM MM	9.4%	8.9%

Tables VIII, IX, and X show the percent increase in energy resulting from enabling Turbo Boost for isolation tests, paired tests, and saturation tests respectively. The increase can be attributed to the increase in the voltage which has a quadratic effect on power consumption and is also the dominant factor in Equation 1. In the isolation tests applications spend a large percentage of their execution time at the higher Turbo Boost frequency which accounts higher increase in the modeled energy. This observation can also be made in the paired execution scenarios—the same-core configuration (where the processor operates mostly at the higher Turbo Boost frequency) shows a higher modeled energy value compared to the different-core config. In the saturation tests, the CM CM CM CM configuration completes in lesser time, and also does not spend time in the highest Turbo Frequency (Figure 8). Consequently, it shows a lower energy metric compared to the MM MM CC CC configuration.

V. RELATED WORK

The release of a new processor triggers performance measurement activity in the hardware hobbyist and research community. Tuck et al. [19] studied Intel Hyper Threading (HT) technology when the first HT processors were released. Keeton et al. [11] characterized the performance of the quad core Pentium processor using OLTP workload. Our work is similar in spirit to both these works—it is an attempt to understand the attributes of a new processor feature. More recently, Barker et al. [3] investigated a pre-release version of the Nehalem architecture. Their work compares the performance of this architecture against the Intel® Tigerton and AMD® Barcelona processors (both x86_64, quad core processors) using scientific computing workloads. They specifically focus on measuring and comparing the NUMA performance of Nehalem against Barcelona and Tigerton, and highlight the excellent performance of Nehalem's memory architecture. In their study, they disable the Turbo Boost feature for their workload execution.

The focus on Nehalem's capability to accelerate sequential phases of parallel applications is inspired by the work of Annavaram et al. [2] as discussed in Section I. We have shown that Nehalem certainly accelerates sequential phases of parallel applications, but the frequency improvements delivered by Turbo Boost are smaller than those projected from running sequential phases on "fast" cores of AMP architectures proposed in previous studies [2], [8], [18].

VI. Conclusion

Turbo Boost Technology opportunistically boosts the frequencies of the cores on the multi-core Core i7 processor. Our isolation, paired and saturation tests showed that Turbo Boost can provide on average up to a 6% reduction in execution time. Turbo Boost Technology had the most impact on performance when the scheduling was not optimal; however, in all cases, Turbo Boost enhanced performance. Turbo Boost also resulted in a significant increase in energy consumption because the processor requires a higher voltage to operate at Turbo Boost frequencies. However, current processors also support low power sleep states where they consume very little power. Disks, memory and other platform components can also be big contributors to platform power consumption. When we consider the total platform power, it could be beneficial to execute with Turbo Boost, complete work faster, and save platform power by placing the CPU and other platform components (DIMMs, Hard Disk Drives, NICs, etc) in low-power idle state. Further investigation is necessary to ascertain our hypothesis and measure the extent of power savings. Finally, Turbo Boost exhibits the potential to accelerate sequential sections in multi-threaded code which improves performance of many parallel applications—an important attribute now and in the future.

VII. Acknowledgments

We would like to thank Martin Dixon, Jeremy Shrall and Konrad Lai from Intel for the help that they provided during the course of this work and for their generous donation of the Nehalem machine on which this work has been performed.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *J Mol Biol*, 215(3):403–410, October 1990.

[2] Murali Annavaram, Ed Grochowski, and John Shen. Mitigating Amdahl's Law through EPI Throttling. In *ISCA '05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 298–309, Washington, DC, USA, 2005. IEEE Computer Society.

[3] Kevin Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Mike Lang, Scott Pakin, and Jose C. Sancho. A Performance Evaluation of the Nehalem Quad-core Processor for Scientific Computing. *Parallel Processing Letters Special Issue*, 18(4), December 2008.

[4] M. Becchi and P. Crowley. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd conference on Computing frontiers*, pages 29–40. ACM New York, NY, USA, 2006.

[5] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[6] Standard Performance Evaluation Corporation. SPEC 2006.

[7] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: selecting representative program-input pairs. pages 83–94, 2002.

[8] Mark D. Hill and Michael R. Marty. Amdahl's Law in the Multicore Era. *Computer*, 41(7):33–38, 2008.

[9] Intel® Corporation. First the tick, now the tock: Next generation Intel® microarchitecture (Nehalem). Whitepaper, Intel Corporation, April 2008.

[10] Intel® Corporation. Intel® Turbo Boost Technology in Intel® Core™ Microarchitecture (Nehalem) Based Processors. Whitepaper, Intel® Corporation, November 2008.

[11] Kimberly Keeton, David A. Patterson, Yong Qiang He, Roger C. Raphael, and Walter E. Baker. Performance Characterization of the Quad Pentium Pro SMP Using OLTP Workloads. Technical Report UCB/CSD-98-1001, EECS Department, University of California, Berkeley, Apr 1998.

[12] R. Kumar, KI Farkas, NP Jouppi, P. Ranganathan, and DM Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 81–92, 2003.

[13] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *ISCA '07: Proceedings of the 34th annual International Symposium on Computer Architecture*, pages 412–423, New York, NY, USA, 2007. ACM.

[14] D. Shelepov and A. Fedorova. Scheduling on Heterogeneous Multicore Processors Using Architectural Signatures. In *Proceedings of the Workshop on the Interaction between Operating Systems and Computer Architecture, in conjunction with ISCA*, 2008.

[15] Daniel Shelepov, Juan Carlos Saez, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. HASS: A Scheduler for Heterogeneous Multicore Systems. *ACM Operating Systems Review, Special Issue on the Interaction among the OS, Compilers, and Multicore Processors*, 43(2), 2009.

[16] Jeremy Shrall and Martin Dixon. Personal Communication.

[17] Allan Snavely and Dean M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *ASPLOS-IX: Proceedings of the ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 234–244, New York, NY, USA, 2000. ACM.

[18] M. Aater Suleman, Onur Mutlu, Moinuddin K. Qureshi, and Yale N. Patt. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS '09: Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 253–264, New York, NY, USA, 2009. ACM.

[19] Nathan Tuck and Dean M. Tullsen. Initial Observations of the Simultaneous Multithreading Pentium 4 Processor. In *PACT '03: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, page 26, Washington, DC, USA, 2003. IEEE Computer Society.