RC23351 (W0409-168) September 28, 2004
Computer Science

IBM Research Report

Characterizing the Impact of Different Memory-Intensity Levels

Ramakrishna Kotla
University of Texas at Austin

Anirudh Devgan, Soraya Ghiasi, Freeman Rawson, Tom Keller
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Characterizing the Impact of Different Memory-Intensity Levels

Ramakrishna Kotla, Anirudh Devgan, Soraya Ghiasi, Freeman Rawson, and Tom Keller
University of Texas at Austin, IBM Austin Research Laboratory

Abstract

Applications on today's high-end processors typically make varying load demands over time. A single application may have many different phases during its lifetime, and workload mixes show interleaved phases. This work examines and uses the differences between *memory*- and *CPU-intensive phases* to reduce power. Today's processors provide resources that are underutilized during memory-intensive phases, consuming power while producing little incremental gain in performance. This work examines a deployed system consisting of identical cores with a goal of running them at different effective frequencies. The initial goal is to find the appropriate frequency at which to run each phase.

This paper demonstrates that memory intensity directly affects the throughput of applications. The results indicate that simple metrics such as IPC (instructions per cycle) cannot be used to determine what frequency to run a phase. Instead, one identifies phases through the use of performance counters which directly monitor memory behavior. Memory-intensive phases can then be run on a slower core without incurring significant performance penalties. The key result of the paper is the introduction of a very simple, online model that uses the performance counter data to predict the performance of a program phase at any particular frequency setting. The information from this model allows a scheduler to decide which core to use to execute the program phase. Using a sophisticated power model for the processor family shows that this approach significantly reduces power consumption. The model was evaluated using a subset of SPECCPU and the SPECjbb and TPC-W benchmarks. It predicts performance with an average error of less than 10%. The power modeling shows that memory-intensive benchmarks achieve about a 50% power reduction at a performance loss of less than 15% when run at 80% of nominal frequency.

1. Introduction

In the past, processor design had a single primary design focus – performance. While this focus has led to faster processors and higher performance, it has introduced two significant, related problems – power and the over-provisioning of systems [1]. Systems are designed for peak performance, with the result that many resources are often under-utilized, resulting in wasted power. With the relative importance of these problems increasing, power is now a first-level design constraint. Researchers have focused on two different approaches to managing power. The first approach attacks power directly through the introduction of low-power components including low-leakage devices and SOI technologies. The second approach instead focuses on over-provisioned systems by exploiting workload variability.

Prior work has found that even a single application is composed of different phases [12]. Modern processor design and research has tried to take advantage of the variability in processor workloads. Examining an application at different granularities exposes different types of variable behavior which can be exploited to reduce power consumption. At the circuit level, clock gating has been introduced into commercial processors to eliminate unnecessary dynamic power consumption in unused components on a cycle-by-cycle basis. Long-lived phases can be detected and exploited by the operating system. Frequency and voltage scaling are common operating system level approaches for addressing workload variability [2]. Complexity-effective designs have been introduced at the microarchitecture level to exploit intermediate-length phases leading to a variety of different techniques including variably sized instruction windows, self-balancing pipelines, and memory systems which use low power modes to save power in unused banks.

Most complexity-effective designs respond to varying demands for the core and memory subsystems by reconfiguring components to meet current demands. For example, an application in a low instruction-level parallelism (ILP) phase may result in a system with small instruction window and a narrower pipeline, while a high ILP phase requires a larger instruction window and wider pipeline. The core dynamically responds these phase changes by reducing and expanding the number of available resources. An alternative to dynamically adjusting the core features is to have a number of fixed designs, any of which may be used at a given time. If done on a component-by-component basis, this approach greatly expands the area consumed by the circuits. The area requirements are even larger if this is done on a core-level basis, but such a design opens up the possibility of running on multiple cores simultaneously.

A complexity-effective approach using heterogeneous, or asymmetric, multi-core designs can be used instead. Historically, designers have used heterogeneous processors to provide highly specialized functions such as cryptographic processing, network address translation and I/O-related functions. Generally, when systems use heterogeneous processors, they use them as coprocessors of some type of general-purpose processing engine. Unlike these designs, but like some recent work ([3], [4], [5]), this effort uses processors that have the same instruction architecture but offer different implementation characteristics including different performance and power levels. This work is a significant contribution beyond the earlier work in this area in the following ways.

- It demonstrates that there is an opportunity to use multiple processors with heterogeneous implementation characteristics but a homogeneous instruction-set architecture. The opportunity arises from the fact that the performance of memory-intensive programs saturates, so that increasing frequency does not yield additional performance.
- The heterogeneity of the processors derives solely from voltage and frequency settings. Memory-intensive programs suffer little performance penalty from running on slower processors.
- It employs a standard, separately developed, methodology [6] to estimate peak power, given the voltage, frequency and technology characteristics of the processors being used.
- It develops a simple but accurate model of the impact of processor frequency and memory-intensity on application performance that allows a system to predict, based on the behavior of the workload as measured by performance counters, what frequency yields the best

performance at the lowest power. This avoids the use of sampling to determine what core should run the work. The model is valid because core heterogeneity arises only from frequency and voltage differences and not from other factors. A scheduler can apply the model to select the proper core on which to run a particular program or phase.

- All of the experimental work is on real, commercially available hardware.

2. Related Work

This work draws inspiration from a number of disparate areas. The eventual goal is a heterogeneous design which incorporates voltage and frequency scaling, a memory-aware operating system, and the use of performance counters to detect phase behavior which, in turn, is used to guide job selection and placement. Relevant prior work in each of these areas is discussed in more detail below.

2.1. Dynamic Voltage and Frequency Scaling

Transmeta's LongRun [7] and Intel's Demand Based Switching [8] respond to changes in demands, but do so on an application-unaware basis. In either approach, an increase in CPU utilization leads to an increase in voltage and frequency while a decrease in utilizations leads to a corresponding decrease in voltage and frequency. Flautner and Mudge [2] explored the use of dynamic voltage and frequency scaling in the Linux operating system with a focus on average power and total energy consumption. Their Vertigo system dynamically uses multiple performance-setting algorithms to reduce energy. One algorithm responds to local, short-lived information while the other responds to a more global view of the world. The combination of algorithms provides an energy savings over LongRun on their experimental setup. However, Vertigo requires instrumentation of certain system components, such as the X server, to track changes in workload demand.

This work differs by responding to easily observed changes in memory subsystem demands. Voltage and frequency scaling are performed only when the memory subsystem indicates there are a large number of memory stalls in the current phase. During CPU-intensive phases, as indicated by low numbers of memory references, the core is run at full voltage and frequency.

2.2. Heterogeneous Cores

Prior work on single ISA, heterogeneous cores fall into two distinct categories. The first uses a processor family which may be run at the same frequency, while the second category uses a processor family which cannot be run at the same frequency. Both approaches take advantage of advances in processor design to provide heterogeneity.

Single frequency heterogeneous cores have been studied by Kumar, et al. ([3], [4], [5]). Their work uses different generations of the Alpha processor family all scaled into the current technology generation. Since Alpha processors have similar complexities, as measured in fan-out-of-four inverter loads, it is possible to scale processors to the same technology and run them at the same frequency. The goal of the work is to minimize energy consumption while maintaining performance. Early work considered only a single application on a single core and examined different metrics including energy per committed instruction and energy-delay product for identifying the most suitable core. The authors examine both dynamic and static core assignment. In later work, the authors extend these ideas to support multiple cores running simultaneously. Sampling is used to identify the best-suited core. In contrast, this paper predicts performance to find the appropriate core.

Ghiasi and Grunwald ([9], [10], [11]) explored single-ISA, heterogeneous cores of different frequencies. Their work uses different generations of the Intel x86 processor family scaled into the current technology generation. Intel’s x86 processor line has undergone significant changes in complexity over time, leading to designs which cannot be clocked at the same frequency even after scaling. Their work focuses on the thermal characteristics of a heterogeneous system and how they can be exploited to reduce and address thermal emergencies. Applications are run simultaneously on multiple cores and an operating system level component is introduced to monitor and direct applications to the appropriate job queues.

In contrast, this work uses a single generation in IBM’s PowerPC processor family, but cores are run at different frequencies. It also differs from prior work by using a commercial product and direct evaluation of techniques, rather than relying on simulation.

2.3. Phase Detection and Performance Counters

Phase detection is important to any real world system which is designed to take advantage of the variability of applications. Sherwood, et al., [12] provides the most detailed analysis of phase detection, but their work uses offline phase analysis. They use different performance-related metrics and correlate them to different phases of a program. Dhodapkar and Smith [13] compared the use of working set signatures, basic block vectors and conditional branch counters and illustrated the tradeoffs between identifying stable phases and phase length. Dynamic phase detection is more prone to error and is not as well studied at the operating system level. The simple performance model developed here can be used for dynamic phase detection.

Most recent work in the area has been on identifying threads to run simultaneously on multi-threaded systems via the use of core metrics accessible through performance counters. Snavely and Carter [14], used sampling of a subset of possible application combinations to determine which applications can be run together with minimal impact on the Tera MTA. Snavely differs from most work in this area by performing experiments on commercial hardware. Kumar, et al, [5] also detects and responds to changes in behavior through the use of sampling phases. Ghiasi [11] used temperature-defined phases, relying on simulated temperatures rather than performance counters to detect phases.

3. Performance Model

On a system with heterogeneous cores, the operating system needs a simple and accurate model that allows it to schedule application threads to the appropriate cores and to set frequency and voltage on machines that allow them to change dynamically. Such a model is also useful to system designers in determining what frequency and voltage settings to offer.

3.1. Motivation

The intuition behind this work is that programs obtain a limited benefit from increasing processor performance. Once a certain level of processor performance is reached, increasing performance offers very little incremental benefit. This *performance saturation* is due to the fact that memory is much slower than the processor, so that at some point the speed of the program is bounded by the speed of the memory. The ratio of memory-intensive to CPU-intensive work in the program determines the saturation point. The saturation point is illustrated in Figure 1 for a number of ratios.

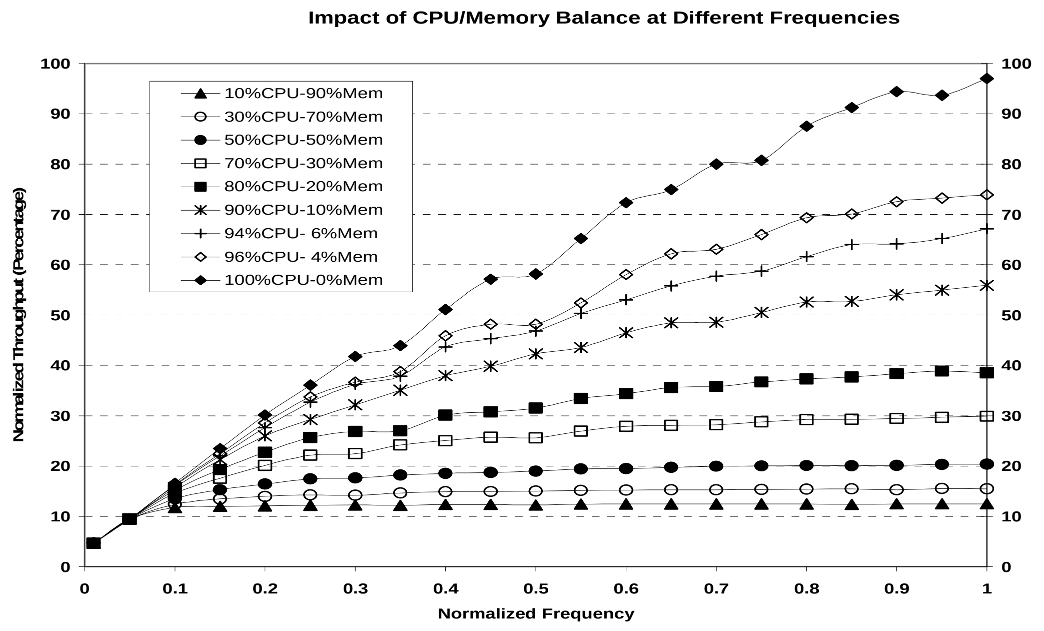


Figure 1: Performance saturation with a synthetic benchmark.

3.2. Examples

A synthetic benchmark, described in Section 4.5, illustrates the possible benefits of using heterogeneous cores with frequencies determined by the memory demands of an application. It was run using a phase length of 2 seconds. Phase 1 has an L1 hit rate of 100% and is considered “CPU-intensive”. Phase 2 has an L1 hit rate of only 50% and is “memory-intensive”. Figure 2 suggests the performance counters tracking memory can be used to identify phases. The memory-intensive phases are readily identified by their much higher levels of L3 and memory references while the CPU-intensive ones appear as high periods of low memory activity and high IPC. The curves nicely track the periodicity of the benchmark, and the presence of information about all of the levels of the memory hierarchy allow the predictive model to estimate accurately the level of memory intensity of a phase. The application throughput shown in the figure lags the phase being reported since it is calculated at the end of each phase. Using the power calculation methodology of Section 4.3, Figure 3 shows the reduction of cumulative energy by adapting the frequency of the processor used to the phases of the synthetic benchmark.

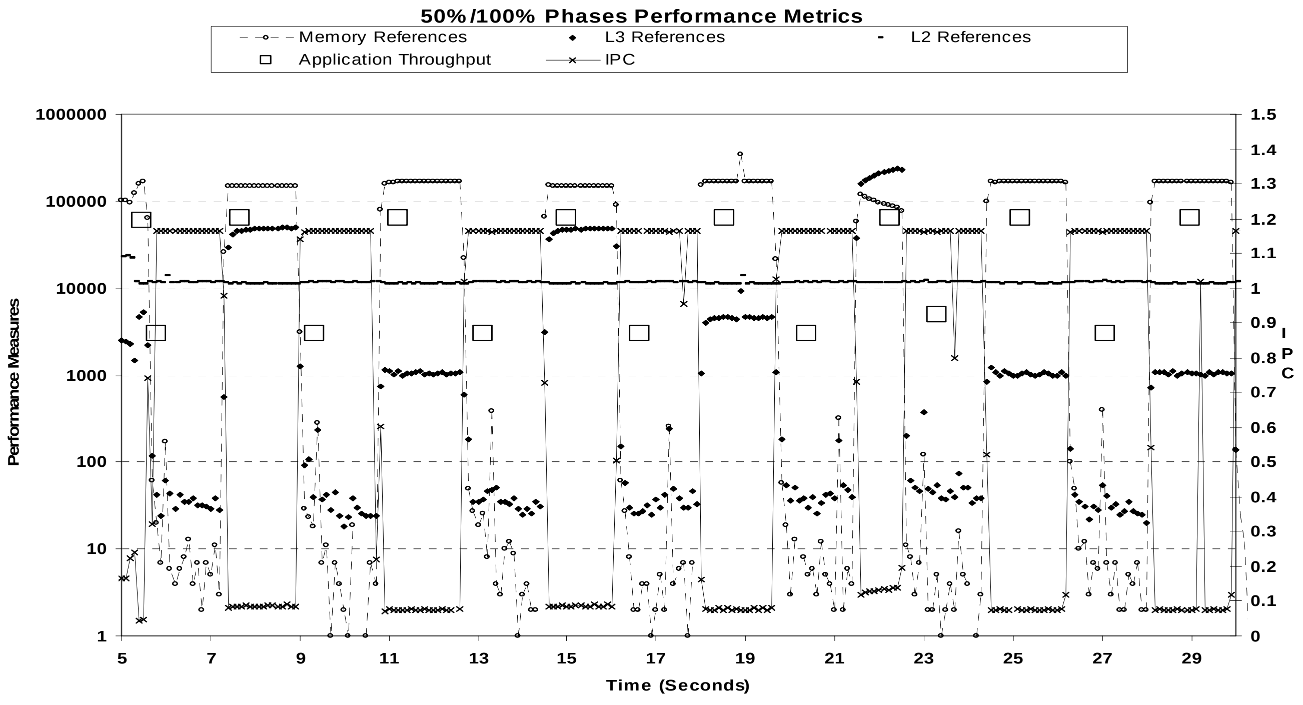


Figure 2: Performance counter results for the phased, synthetic benchmark.

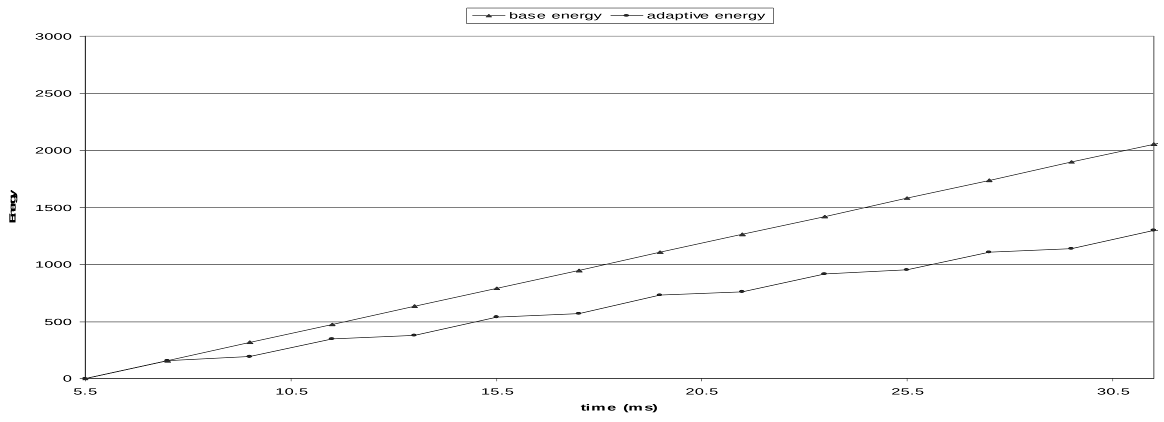


Figure 3: Cumulative energy with and without adaptation to the performance requirements of the phases.

In order to develop a model to predict the target effective frequency, data was collected for a range of different effective frequencies, and at each frequency, counter data was gathered for a range of L1 hit rates. The observed IPC versus the effective frequency is shown in Figure 4. It indicates reducing the frequency is an effective way to mask the latencies in cycles of memory accesses.

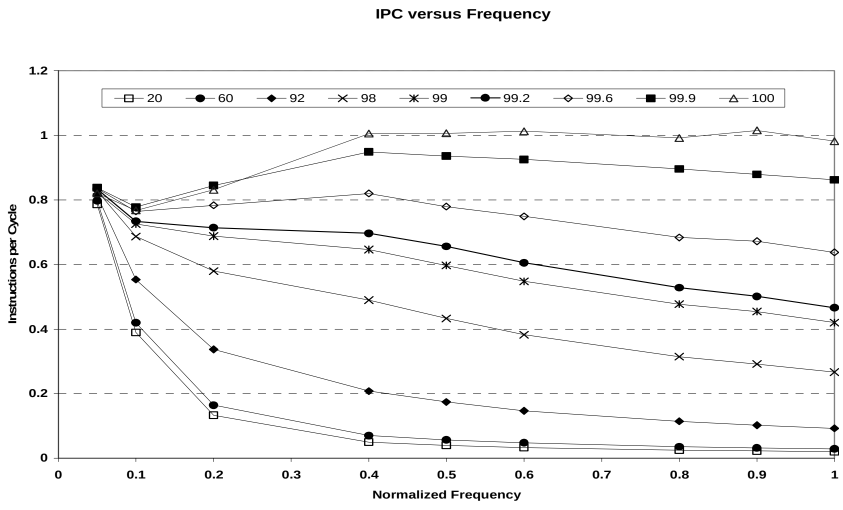


Figure 4: Instructions per cycle versus frequency at different memory intensity levels.

These results suggested that IPC and frequency alone would be sufficient to predict the target effective frequency. However, it is easy to construct a counter example. A simple synthetic workload with a critical path of long-latency floating point instructions can be constructed to target any given IPC. An example is shown in Figure 5.

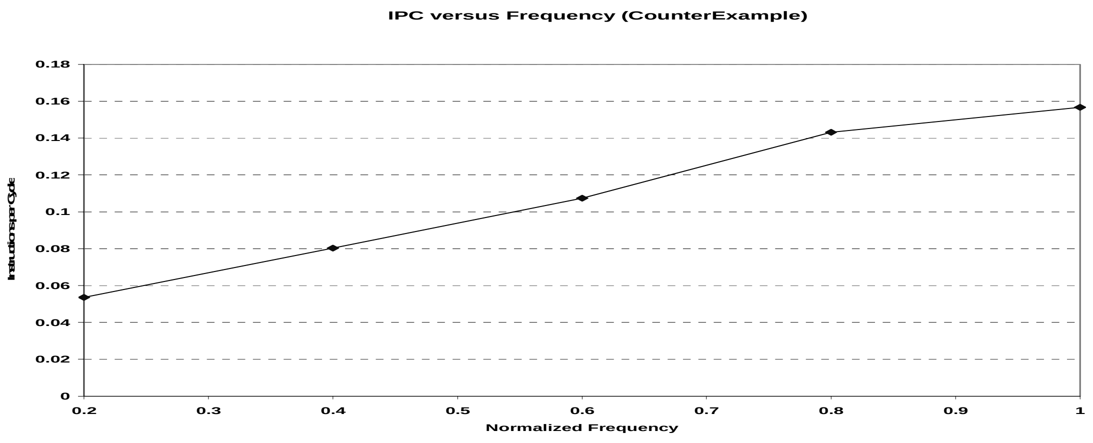


Figure 5: IPC versus normalized frequency for a low-IPC but CPU-intensive program.

3.3. Initial Model

The counter example indicates that a more complicated scheme is necessary. Instead, a model of IPC was developed and broken down into frequency-dependent and frequency-independent components. In the following, α is the IPC of a perfect machine with infinite L1 caches and no stalls. In other words, α takes into account the ILP of a program and the hardware resources available to extract it. Since α cannot be determined accurately, for programs with an IPC less than 1, it is taken to be 1, and for those with an IPC greater than 1, it is the IPC at the nominal frequency.

$$IPC \equiv \frac{Instructions}{Cycles} = \frac{Instr}{C_{stall} + C_{inst}}$$

$$= \frac{Instr}{\frac{Instr}{\alpha} + C_{branch\_stalls} + \ldots + C_{pipeline\_stalls} + (C_{L2\_stalls} + C_{L3\_stalls} + C_{mem\_stalls})}$$

$$= \frac{1}{\frac{1}{\alpha} + \frac{1}{Instr}(N_{L2}C_{L2\_stalls} + N_{L3}C_{L3\_stalls} + N_{mem}C_{mem\_stalls}) + \frac{C_{other\_stalls}}{Instr}}$$

$$= \frac{1}{\frac{1}{\alpha} + \frac{1}{Instr}(N_{L2}T_{L2\_stalls} + N_{L3}T_{L3\_stalls} + N_{mem}T_{mem\_stalls})\,f + \frac{C_{other\_stalls}}{Instr}}$$

Here each N_x is a count of the number of occurrences as provided by the performance counters, each C_x is the number of processor cycles per event and each T_x is the time consumed by each event. Here T_x is pre-determined for the particular processor. The equation assumes that the C_x values are all truly constant. In reality, this is not true and is a source of error, but in practice it does yield a good approximation. This point is discussed further in Section 5. Taking the reciprocals gives cycles per instruction (CPI) in the following equation. This version of the equation is used later to calculate the percentage of memory stall cycles for the program given that its CPI at 100 % frequency is known.

$$CPI = CPI_{inst} + CPI_{mem\_stalls} + CPI_{other\_stalls} = \frac{1}{\alpha} + \frac{N_{L2}T_{L2\_stalls} + N_{L3}T_{L3\_stalls} + N_{mem}T_{mem\_stalls}}{Instr} f + \frac{C_{other\_stalls}}{Instr}$$

The equation asymptotically reaches the following CPU-intensive and memory-intensive forms. Figure 4 gives a graphical representation of these approximations.

$$IPC_{cpu\_intensive} \approx \frac{1}{\frac{1}{\alpha} + \frac{1}{Instr}(C_{branch} + \ldots + C_{pipeline})} \approx \alpha$$

$$IPC_{memory\_intensive} \approx \frac{Instr}{(N_{L2}T_{L2} + N_{L3}T_{L3} + N_{mem}T_{mem})\, f}$$

At any given frequency, this equation can be used to predict the IPC at another frequency given the number of misses at the various levels in the memory hierarchy as well as the actual time it takes to service a miss. This provides a mechanism for identifying the optimal frequency at which to run a given phase with minimal performance loss. As expected, the more memory-intensive a phase is, as indicated by the memory subsystem performance counters, the more feasible it is to lower the frequency (and voltage) to save power without impacting the performance and the better that the phase fits onto a slower core.

3.4. Handling Non-Constant Latencies

The model developed in the previous section assumes that the latencies to the caches and memories are constant. Due to prefetching and other effects, this is not true, and there are easily detectable cases, as shown in Section 5.1, where the error from using the nominal latencies is significant. Since the latencies are assumed to be program- but not frequency-dependent, by making an additional linearity assumption, it is possible to determine the latencies empirically. However, this does require the measurement of the IPC of the program at two different frequencies. The IPC equation in the previous section may be written as a linear equation of two variables in the form $af + b = 1/IPC$.

By taking two measurements of IPC at different frequencies, one gets two instances of the equation that can then be solved to get *a* and *b*, which are then used to do the predictions. Intuitively, *a* / (*a* + *b*) is the fraction of the cycles or the CPI due to cache and memory stalls. The experiments presented here use both models, trying the initial one first, and then if it yields too great an error, turning to the second.

4. Methodology

4.1. Power Calculations

The equation $P = CV_{dd}^2 f + \beta V_{dd}^2$ gives the power as a function of the frequency and voltage. Here C is the capacitance, f is the frequency, V_{dd} is the supply voltage and β is process- and temperature-dependent. The first term is the active power while the second is the static, due primarily to leakage.In reality, systems usually scale frequency and voltage separately, subject to the constraint that running at a given frequency typically requires at least a certain voltage. However, treating frequency and voltage as independent quantities makes analysis difficult. Due to the relationship between the two quantities, one reasonable simplification is to take the voltage to be a function of the frequency, that is, $V_{dd} = V(f)$. Here the value returned by V(f) is the lowest possible voltage for the part at the specified frequency.

4.2. Experimental Platform

The experiments described in this paper were performed on an IBM PowerPC-based pSeries P630 [15] system consisting of 4 1GHz Power4+ cores operating at a core voltage of 1.3 volts. Each core has a private 32 KB L1 instruction cache and a 64 KB L1 data cache. Two adjacent cores share a unified 1.44 MB data cache, resulting in two L2 caches, each shared by two cores. In addition, the adjacent cores share a 32 MB L3 cache, resulting in two L3 caches, each shared by two cores. The machine has 4 GB of main memory. Using experimentation, it was determined that the nominal latency to the L1 cache is 4 to 5 processor cycles, the nominal latency to the L2 cache is 11 to 14 cycles, the nominal latency to the L3 is 100 to 125 cycles and that to memory is 500 cycles. Although these values agree reasonably well with other reported values for the same hardware, they are, in fact, dependent on the way in which the measurement program accesses the caches and memory, and, as will become apparent later, the variability affects the predictive model developed by this research.

The experimental platform runs Gentoo Linux with a 2.6.7 kernel. The kernel has been modified to support CPU throttling. The underlying hardware provides mechanisms for throttling the pipeline via dispatch, fetch or commit throttling. The hardware does not currently support direct frequency scaling, so throttling is used to mimic the effects of frequency scaling. All experiments are performed using fetch throttling. Experimental data indicate that this provides a good approximation to frequency scaling, even though the remainder of the pipeline continues processing during fetch-throttled cycles. Throttling can be used to cover the entire range from 0% frequency to 100% frequency. This work assumes throttling yields the same power and performance results that using different frequencies for the processors would. In other words, if f_{eff} is the effective frequency, $f_{nominal}$ is the nominal frequency, which is 1.0 GHz for the experimental platform used, and throttle is the throttling percentage, expressed as a decimal, then f_{eff} = throttle x $f_{nominal}$.

The Power4 processor provides a number of performance counters, accessible either in user or privileged state, which can monitor memory performance. The operating system or a user program can read the performance counters, and the implementations used here read them periodically, with 10 and 100 millisecond being typical sampling intervals. Different experiments use somewhat different programs to sample and record the counters, depending on the nature of the experiment. In many experiments, to avoid interference from daemon processes, the experimental platform sets the priority of the workload of interest to one for a high priority real-time job and the processor affinity to a particular core.

4.3. Calculating Power

The experiments in this study rely on the Lava power-estimation tool [6]. Lava is driven primarily by the voltage and frequency used and the implementation technology: it is not activity-based. However, since the Power4 parts do not exhibit very much power variation across activity levels, there is no need for activity-based calculations. Lava is a circuit-level tool, so this work uses it to determine the shape of the power versus voltage and frequency curves for a particular technology and then determines where on the curve, in terms of relative power, the particular design points fall. The primary metric is peak power as calculated at the selected frequencies, assuming that at each frequency setting, one uses the minimum possible voltage. Voltage is treated as a well-defined function of frequency to simplify the analysis by reducing it to the consideration of a single independent variable. Actual hardware generally runs a range of frequencies at each voltage for a fictitious, but not unlikely, implementation used in this similar to the real hardware used elsewhere in this study.

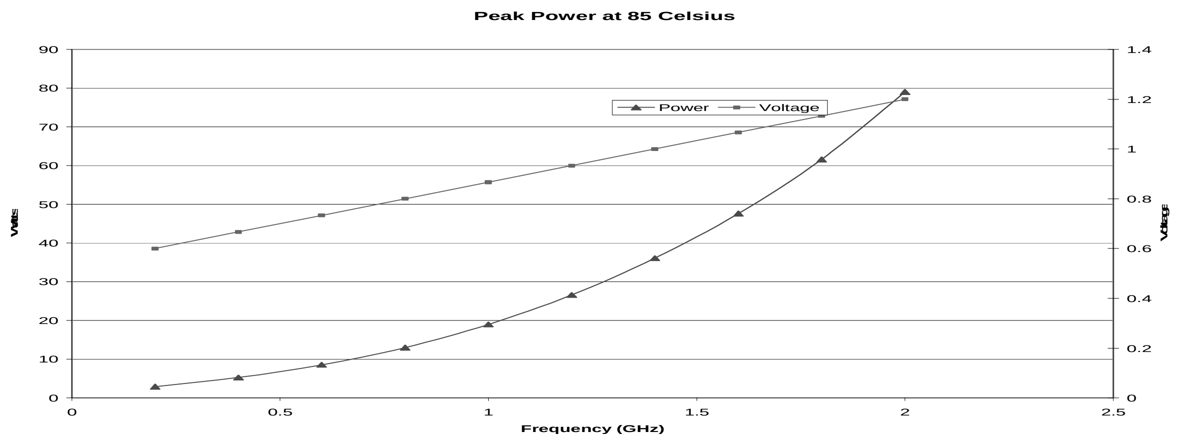


Figure 6: Typical power and voltage curves as calculated by Lava.

4.4. Metrics

In this paper, performance is reported in terms of throughput. The synthetic microbenchmark used here reports performance results that are easily interpreted as throughput numbers. For other tests, benchmark-specific throughput is reported where appropriate. Power is reported in terms of relative power. The base for power is a core at full frequency for the duration of the experimental run.

4.5. Benchmarks

This work is a preliminary study analyzing the feasibility of using differences in memory demands to reduce power. As such, much of this work has been done with a synthetic benchmark that allows one to measure the performance variability of a program with an adjustable ratio of processor-intensive to memory-intensive operations. The synthetic benchmark is a single-threaded program that accepts a parameter that determines the ratio of memory-intensive to CPU-intensive as well as the length of phases. It currently supports two (2) phases, but the phases may be of different lengths and different memory-to-CPU intensity. It is constructed so that a miss in the L1 is highly likely to result in a memory access due to the large memory footprint. The program reports its performance in terms of throughput.

To verify the generality of the results, additional experiments were done using a subset of SPEC CPU2000 benchmarks -- gzip, mcf, and art – chosen for their contrasting memory behavior [16], and with two commercial workloads – SPECjbb and TPC-W [17] implemented using Apache 2.0.49, PHP 3.4.6 and MySQL 4.0.28 [17]. The web server and the data base engine are both run on the measured system. The predictive model was applied to the analysis of all of these benchmarks, but the results reported here are averaged values since there is currently no implementation of an adaptive scheduler using the model.

5. Experimental Results

5.1. SPECCPU

The first of the SPECCPU benchmarks, gzip, is CPU-intensive, and the initial predictive model yields reasonable results for it. Figure 7 shows the measured and expected performance results over the range of normalized frequencies. Memory stalls account for about 8% of total cycles. The model predicts the application performance (IPS) with an average error of 2.4% and a maximum error of 12%. A 58% reduction in power by reducing frequency at minimum voltage from 100 to 80% comes at the cost of 20% performance degradation.

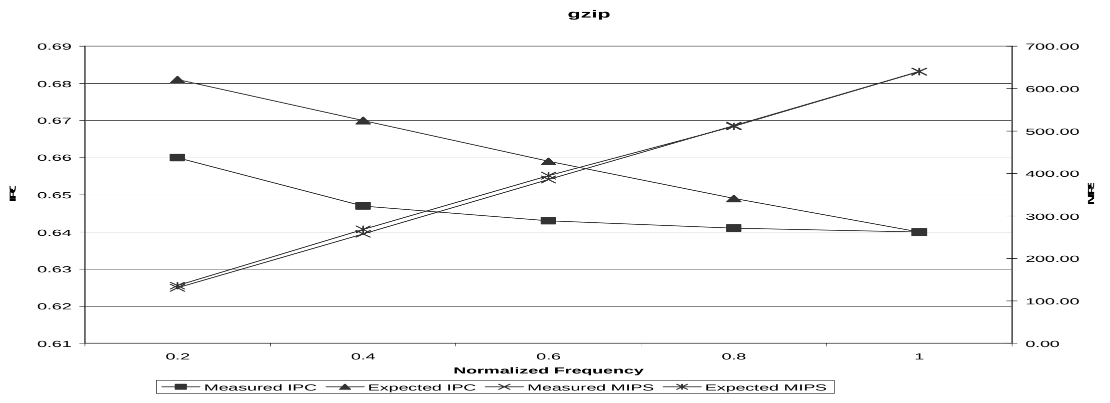
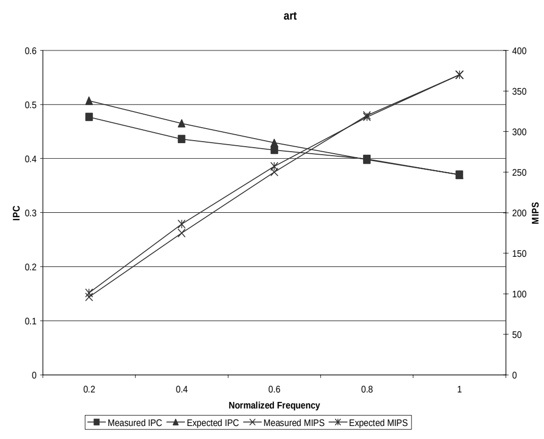


Figure 7: gzip measured and expected performance versus normalized frequency.

Since it accesses memory in such a way that the processor benefits from prefetching, the art benchmark, whose results are shown in Figure 8, demonstrates that it is essential to handle non-nominal latencies.



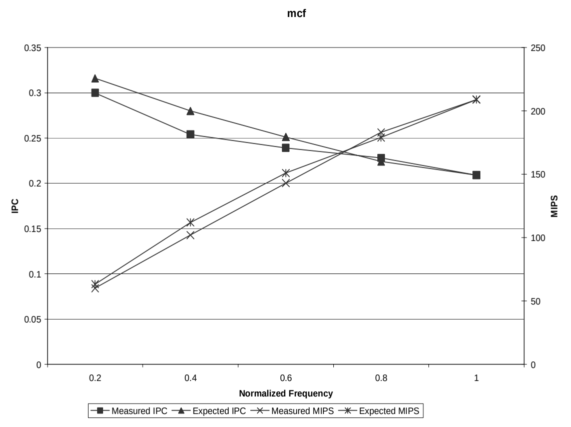


Figure 8: art and mcf measured and expected performance versus normalized frequency.

In this case, *a* is 0.9 and *b* is 1.79, so that cache and memory stalls account for about 33% of the total cycles. The predictive model has an average error of 4.53% and a maximum error of 7% versus the measured instructions per second. However, using the original model with constant latencies yields errors in excess of 20%. A 55% reduction in power at minimum voltage by reducing frequency 20% costs a reduction of 13.5% in application performance.

Similarly, mcf, also shown in Figure 8, is similarly moderately memory-intensive. It has *a* = 2.03 and *b* = 2.75. Memory stalls account for 42% of total cycles. The predictive model yields a result with an average error of 6.2% and a maximum error of 12%. There is a 54% reduction in power by reducing frequency from 100% to 80% at the cost of 12.5% performance degradation.

These results show that running the memory-intensive SPECCPU benchmarks, art and mcf, on a lower frequency processor can reduce power significantly with only a nominal loss in performance. They also demonstrate the accuracy of the model in predicting the performance of both the CPU-intensive and the memory-intensive benchmarks.

5.2. SPECjbb

The SPECjbb benchmark measures the performance of a system's processors and memory using a Java implementation of the processing portion of the TPC-C. Figure 9 shows the results obtained for it.

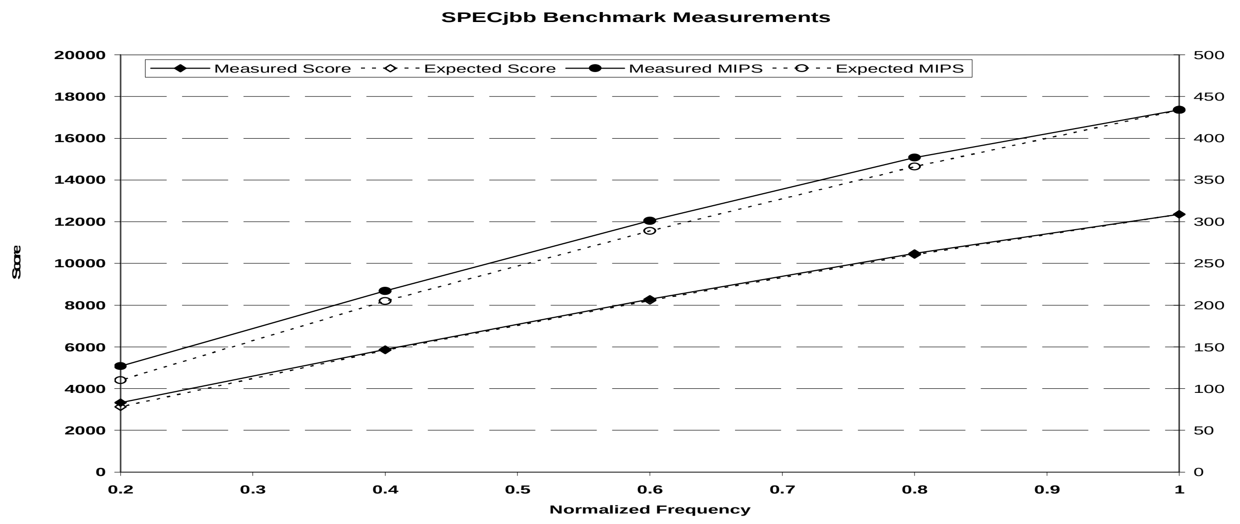


Figure 9: SPECjbb measured and expected results.

SPECjbb is a moderately CPU intensive workload with 25% of total cycles are due to memory stalls. The model predicts its throughput score at various frequencies within 6% of measured performance

5.3. TPC-W

The TPC-W experiment uses the implementation of TPC-W described in [17]. Figure 10 plots the reported performance of the benchmark versus the normalized frequency as well as the performance projected by the predictor. Both the MIPS and the TPC-W throughput metric of WIPS are shown. One interesting feature of the TPC-W is that it exhibits a relatively constant IPC across its execution, and it is an example of a benchmark that is limited by CPU frequency rather than by memory access latency. Here there is very little opportunity to make good use of heterogeneous cores using only a very straightforward scheduling scheme.

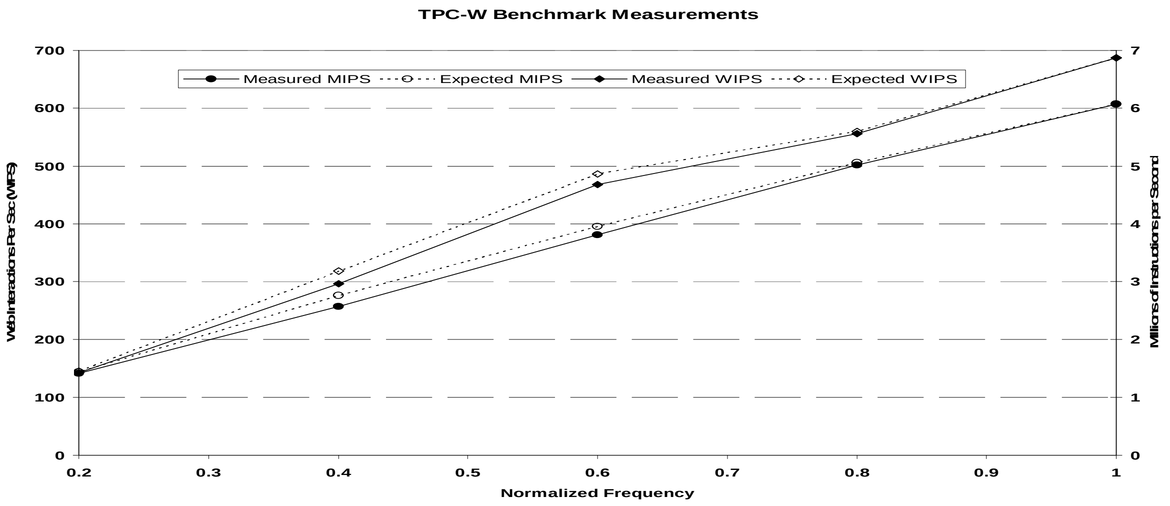


Figure 10: TPC-W performance versus frequency.

6. Conclusions

This study demonstrates the presence of performance saturation due to cache and memory latencies: this saturation limits the benefits of additional processor frequency and higher power consumption and creates an opportunity for the use of heterogeneous cores to limit dramatically power consumption without significantly reducing performance. When heterogeneity arises from differences in frequency and voltage, there are simple predictive models based on memory-related performance counter information that allow a scheduler to select which core should run what phase of a program or permits either the designer or the system itself to determine what voltage and frequency settings to use for its processors. An even simpler IPC-based predictor is insufficient due to the existence of low IPC programs whose performance does scale with processor frequency. The existence of the predictors eliminates the need to run a program phase on all cores to sample its performance in order to determine the proper assignment and offers the possibility of developing an adaptive scheduler.

Although this work indicates the value of heterogeneity based on frequency and voltage settings and the existence of a simple predictor for the performance impact of such heterogeneity, this is a preliminary study. But even this initial investigation shows the promise of power reduction at limited performance cost through the use of processor cores that differ in their operating frequencies and voltages.

7. Acknowledgment

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCH3039004.

8. References

[1] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Mike Kistler and Tom W. Keller, “Energy Management for Commercial Servers”, Computer, volume 36, number 12, December, 2004, pages 39-48.

[2] K. Flautner and T. Mudge, “Vertigo: Automatic performance-setting for Linux”, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI’02), December, 2002, pages 105-116.

[3] Rakesh Kumar, Keith Farkas, Norman P. Jouppi, Partha Ranganathan, and Dean M. Tullsen, “A Multi-Core Approach to Addressing the Energy-Complexity Problem in Microprocessors”, Workshop on Complexity-Effective Design, 2003.

[4] Rakesh Kumar, Keith Farkas, Norman P. Jouppi, Partha Ranganathan, and Dean M. Tullsen, “Single-IA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction”, Proceedings of the 36th International Symposium on Microarchitecture, December, 2003.

[5] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, Keith I. Farkas, “Single –ISA Heterogeneneous Multi-Core Architectures for Multithreaded Workload Performance”, Proceedings of the 31st International Symposium on Computer Architecture, June, 2004.

[6] Anirudh Devgan, “LAVA: Leakage Avoidance and Analysis”, IBM User’s Guide, 2004.

[7] Transmeta Corporation, “Transmeta LongRun Dynamic Power/Thermal Management”, http://www.transmeta.com/crusoe/longrun.html.

[8] Deva Bodas, “New Server Power-Management Technologies Address Power and Cooling Challenges”, Technology@Intel, http://www.intel.com/update/contents/sv09031.htm.

[9] S. Ghiasi and D. Grunwald, “Aide de Camp: Asymmetric Dual Core Design for Power and Energy Reduction”, Technical Report CU-CS-964-03, Department of Computer Science, University of Colorado, Boulder, May, 2003.

[10] S. Ghiasi and D. Grunwald, “Thermal Management with Asymmetric Dual Core Designs”, Technical Report CU-CS-965-03, Department of Computer Science, University of Colorado, Boulder, May, September, 2003.

[11] S. Ghiasi, “Aide de Camp: Asymmetric Multi-Core Design for Dynamic Thermal Management”, Ph. D. thesis, Department of Computer Science, University of Colorado, Boulder, July, 2004.

[12] Timothy Sherwood, Erez Perelman, Greg Hamerly and Brad Calder, “Automatically Characterizing Large Scale Program Behavior”, Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X) , October, 2002, pages 45-57.

[13] Ashutosh S. Dhodapkar and James E. Smith, “Comparing Program Phase Detection Techniques”, 36th Annual International Symposium on Microarchitecture (Micro-36),December, 2003.

[14] Allan Snavely and Larry Carter, ``Symbiotic Jobscheduling on the Tera MTA", Workshop on Multi-Threaded Execution, Architecture and Compilation (MTEAC'00), January, 2000.

[15] Guilian Anselmi, Derrick Daines, Stephen Lutz, Marcelo Okano, Wolfgang Seiwald, Dave Williams and Scott Vetter, pSeries 630 Models 6C4 and 6E4 Technical Overview and Introduction, IBM Corporation, December, 2003.

[16] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D.C. Burger, S.W. Keckler, and C.R. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," *Proceedings of the 30th International Symposium on Computer Architecture (ISCA-30),* June, 2003.
[17] Cristiana Amza, Anupam Chanda, Alan L. Cox, Sameh Elnikety, Romer Gil, Karthick Rajamani, Willy Zwaenepoel, Emmanuel Cecchet and Julie Marguerite, Specification and Implementation of Dynamic Web Site Benchmarks, IEEE 5th Annual Workshop on Workload Characterization (WWC-5), November, 2002.