Best of Both Latency and Throughput

Ed Grochowski, Ronny Ronen, John Shen, Hong Wang
Intel Labs
Ed, John, Hong: Intel, 2200 Mission College Blvd, Santa Clara, CA 95054, USA
Ronny: Intel Israel, MTM Scientific Industries Center, Haifa 31015, Israel
edward.grochowski@intel.com, ronny.ronen@intel.com,
john.shen@intel.com, hong.wang@intel.com

Abstract

This paper describes the tradeoff between latency performance and throughput performance in a power-constrained environment. We show that the key to achieving both excellent latency performance as well as excellent throughput performance is to dynamically vary the amount of energy expended to process instructions according to the amount of parallelism available in the software. We survey four techniques for achieving variable energy per instruction: voltage/frequency scaling, asymmetric cores, variable-size cores, and speculation control. We estimate the potential range of energies obtainable by each technique and conclude that a combination of asymmetric cores and voltage/frequency scaling offers the most promising approach to designing a chip-level multiprocessor that can achieve both excellent latency performance and excellent throughput performance.

1. Introduction

Computer workloads may be broadly classified into two categories: those that have little inherent parallelism (*scalar*) and those that have significant amounts of parallelism (*parallel*). Typical scalar workloads include software development tools, office productivity suites, and operating system kernel routines. Typical parallel workloads include 3D graphics, media processing, and scientific applications. Scalar workloads may have IPCs in the range of 0.2 to 2 whereas parallel workloads may have IPCs in the range of 4 to several thousand [28]. The latter high IPCs are obtainable through the use of instruction-level parallelism and thread-level parallelism, respectively.

It is desirable to design a microprocessor that can run both scalar and parallel workloads at high performance. Moreover, the same program may contain phases of high parallelism as well as phases of low parallelism. It is therefore desirable to design a microprocessor that can dynamically alter its behavior according to the amount of parallelism available in each phase. The ability to quickly run both phases of high parallelism and phases of low parallelism can reduce the overall run-time of such a program [23].

2. Scalar and Parallel Performance

To achieve high scalar performance, it is necessary to reduce execution latency as much as possible. Microarchitectural techniques to reduce effective latency include speculative execution, branch prediction, and caching. The pursuit of high scalar performance has resulted in large out-of-order, highly speculative, deep pipeline microprocessors such as the Intel® Pentium® 4 processor [18].

To achieve high parallel performance, it is necessary to provide as much execution throughput (bandwidth) as possible. Microarchitectural techniques to increase throughput include wide superscalar, chip-level multiprocessing, and multithreading. High-throughput performance designs include Sony's Emotion Engine (used in the Sony PlayStation* 2) and Sun's Niagara [25, 30].

Two problems arise when trying to build a microprocessor that performs well on both scalar and parallel tasks. The problems are:

- The design techniques needed to achieve short latency are very different from the design techniques needed to achieve high throughput
- Achieving short latency often requires expending large amounts of energy per instruction, whereas

Proceedings of the IEEE International Conference on Computer Design (ICCD'04)
1063-6404/04 $ 20.00 IEEE

achieving high throughput requires minimizing the amount of energy expended per instruction

In this paper, we assume that the first problem may be tackled by utilizing both sets of design techniques (for example, by building a chip-level multiprocessor using an array of high scalar-performance cores). Furthermore, we assume that future process technologies will provide sufficiently large transistor budgets to enable very large chip-level multiprocessors to be built, and that future software will provide enough threads to effectively utilize very large numbers of processors. With these assumptions, performance will be limited by power rather than design effort, die area, or software algorithms. We will thus concentrate on the fundamental problem of the differing energy requirements of scalar and parallel performance.

3. Power and Performance

An examination of microprocessor design trends shows that modern microprocessors are expending large amounts of power for relatively small improvements in scalar performance. A graph of power versus scalar performance for four generations of Intel microprocessors is shown in Figure 1. Both power and performance have been adjusted to factor out improvements due to process technology over time, and all data have been normalized to the i486™ microprocessor.

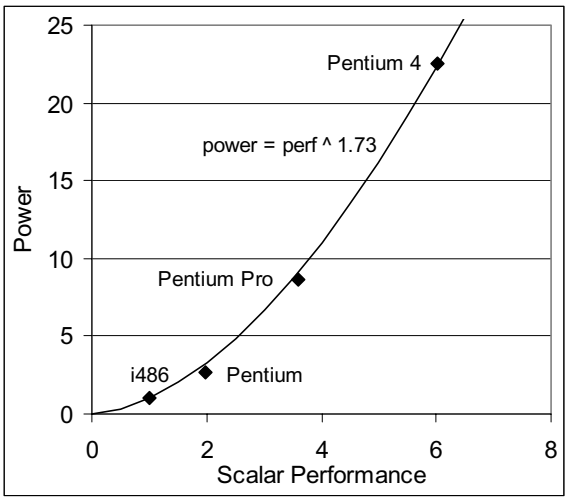

Figure 1: Power versus Scalar Performance

Figure 1 was created by comparing the SpecInt and power ratings of a pair of microprocessors on the same process technology, at the same supply voltage, and at the same point in time. From the raw data in Table 1, we compute the ratio of the performance of each pair of microprocessors (Pentium® to i486, Pentium Pro to Pentium, and Pentium 4 to Pentium III processors), and multiply together the performance ratios to create the graph. We repeat the calculation for power. This method effectively removes the contributions due to process technology, leaving only the contributions due to design. The results are as if all four generations of microprocessors were built on the same process technology. To realize these performance deltas in practice, older microprocessors would need to be given appropriate high-speed memory systems in newer process technologies.

Process	0.8um		0.6um		0.18um	
Uarch	486	P5	P5	P6	P6	NetBurst
Product	486DX2-66	Pentium-66	Pentium-100	Pentium Pro	Pentium III	Pentium 4
Frequency (MHz)	66	66	100	150	1000	2000
Voltage (volts)	5	5	3.3	3.3	1.75	1.75
Performance	39.6 SpecInt 92	77.9 SpecInt 92	3.33 SpecInt 95	6.08 SpecInt 95	405 SpecInt 2K	681 SpecInt 2K
Power (watts)	4.9 typical	13 typical	10.1 peak	33.1 peak	29.0 thermal	75.3 thermal

Table 1: Four Generations of Microprocessors

The steepness of Figure 1 is striking. Relative to the i486 processor, the Pentium 4 processor delivers approximately 6 times more scalar performance (2x the IPC at 3x the frequency), but consumes 23 times more power. The graph implies that high-performance microprocessors are spending roughly 4 units of power for every 1 unit of scalar performance compared to earlier generations. We informally refer to this graph as *climbing an ever-steepening hill.*

The reason for the dramatic increase in power is that the design techniques needed to increase scalar performance tend to result in much more energy being expended per instruction due to higher capacitance toggled to process each instruction. The higher switching capacitance is due to the physically larger layout as well as additional toggles for processing misspeculated instructions. Large die area is a direct consequence of the amount of hardware required to process many in-flight instructions and to recover from misspeculation.

In contrast to scalar performance, throughput performance is highly linear with respect to power. Figure 2 shows what a throughput-oriented microprocessor similar to Piranha [7] would be expected to achieve (considering only the CPU contributions to power).

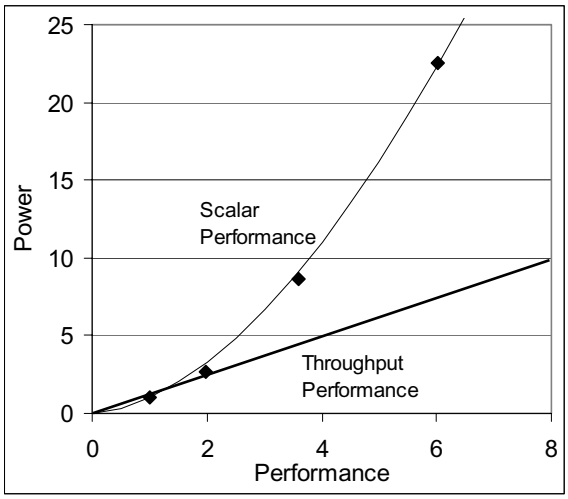

Figure 2: Power versus Throughput Performance

Throughput performance tends to have an almost linear power/performance ratio because replicating a CPU core results in nearly proportional increases to both throughput performance and power.

4. Power Efficiency

Power efficiency is commonly measured in terms of MIPS/watt. The MIPS/watt metric is equivalent to energy per instruction., or more precisely, MIPS/watt is the reciprocal of energy per instruction as follows:

$$\frac{\text{Mips}}{\text{Watt}} = \frac{\frac{\text{Instructions}}{\text{Second}}}{\frac{\text{Joules}}{\text{Second}}} = \frac{\text{Instructions}}{\text{Joule}} \qquad (1)$$

An important property of the energy per instruction metric is that it is independent of the amount of time required to process an instruction. This makes energy per instruction an ideal metric for throughput performance. For latency performance, metrics such as MIPS^2/watt (equivalent to energy•delay) and MIPS^3/watt (equivalent to energy•delay^2) are more appropriate because they assign increasing weight to the amount of time required to process an instruction.

An approximate analysis of a microprocessor's power consumption may be performed by thinking of the microprocessor as a capacitor that is charged or discharged with every instruction processed (for simplicity, we'll ignore leakage current and short-circuit switching current). With this assumption, energy per instruction depends on only two things: the amount of capacitance toggled to process each instruction (from fetch to retirement), and power supply voltage. The well-known formula:

$$E = ½ \bullet C \bullet V^2 \qquad (2)$$

which is normally applied to capacitors, may be applied to microprocessors as well. E is the energy required to process an instruction; C is the amount of capacitance toggled in processing the instruction; and V is the power supply voltage.

A microprocessor must operate within a fixed power budget such as 100 watts. Averaged over some time period, the microprocessor's power consumption cannot exceed the power budget regardless of what the microprocessor or software do. To achieve this objective, modern microprocessors incorporate some form of dynamic thermal management [8, 18]. Similarly, a chip-level multiprocessor is required to regulate (or *throttle*) its activities to stay within a fixed power budget regardless of whether it is delivering 0.2 IPC or 20 IPC. To deliver the best performance, the chip-level multiprocessor must be able to vary its MIPS/watt, or equivalently its energy/instruction, over a 100:1 range in this example.

The key to designing a microprocessor that can achieve both high scalar performance and high throughput performance is to *dynamically vary the amount of energy expended to process each instruction according to the amount of parallelism available in the software.* In other words, if there's little parallelism, a microprocessor should expend all available energy processing a few instructions; and if there's a lot of parallelism, the microprocessor should expend very little energy in processing each instruction. We can formalize this relationship as:

$$P = EPI \bullet IPS \qquad (3)$$

where P is the fixed power budget, EPI is the average energy per retired instruction, and IPS is the aggregate number of instructions retired per second across all CPU cores. We take the goal of maintaining the total multiprocessor chip power at a nearly constant level, which is a simple but reasonable goal for AC line powered equipment. Battery powered devices will require a more sophisticated power management policy.

We now consider several techniques to achieve variable ratios of energy per instruction. For each technique, we quantify the potential range of energy per instruction and the amount of time required to vary energy per instruction. Note that we are quantifying a minimum-to-maximum range of energy per instruction rather than incremental energy per instruction as in

more theoretical work. Of the four techniques we examine, three vary the amount of switching capacitance while one varies power supply voltage.

5. Voltage/Frequency Scaling

Marketed under tradenames such as Intel's SpeedStep® technology [19], AMD's PowerNow!* [3], and Transmeta's LongRun* [14], CMOS voltage/frequency scaling has long been used to achieve different energy per instruction ratios. The basic idea is to vary the microprocessor's power supply voltage and clock frequency in unison according to the performance and power levels desired. Ideal voltage/frequency scaling exhibits a cubic relationship between power and performance according to the well-known equations:

$$P = C \bullet V^2 \bullet F \quad (4)$$

$$F \sim V \quad \text{for small deltas} \quad (5)$$

$$P \sim F^3 \quad (6)$$

$$P \sim V^3 \quad (7)$$

where P is power, C is switching capacitance, V is supply voltage, and F is clock frequency. Note that switching capacitance may be expressed as an activity factor multiplied by a total, constant amount of capacitance.

To maintain a chip-level multiprocessor's total power consumption within a fixed power budget, voltage/frequency scaling may be applied dynamically as follows:

- In phases of low thread parallelism, run a few cores using high supply voltage and high frequency for best scalar performance
- In phases of high thread parallelism, run many cores using low supply voltage and low frequency for best throughput performance

For simplicity, we assume that inactive cores consume no power. To come close to this goal, leakage control techniques such as dynamic sleep transistors and body bias [29] will be required in addition to clock gating. We analyze energy per instruction by estimating the range of possible supply voltages and applying equation (2).

From equation (2), energy per instruction is proportional to the square of the supply voltage. Today's mobile microprocessors operate over a supply voltage range from 1.4 down to 1.0 volts, yielding a 2x reduction in energy per instruction. We estimate that a supply voltage reduction of 50% may be designed for today as a stretch goal, yielding a 4x reduction in energy per instruction. However, in future process technologies, the range of useful voltage/frequency scaling can be expected to decrease.

6. Asymmetric Cores

Recent research has suggested the possibility of designing a single-ISA heterogeneous multi-core microprocessor in which different microarchitectures are used to span a range of performance and power [13, 16, 20, 21, 24]. For our work, we assume a chip-level multiprocessor built from two types of CPU cores, referred to as the large core and small core. The two cores implement the same instruction set architecture, use cache coherency to implement shared memory, and differ only in their microarchitecture. The large core may be an out-of-order, superscalar, deep pipeline machine whereas the small core may be an in-order, scalar, short pipeline machine. The Intel Pentium 4 processor and Intel i486 processor are representative of the two classes of cores. Table 2 presents some estimated parameters.

	Large core	Small core
Microarchitecture	Out-of-order, 128-256 entry ROB	In-order
Width	3-4	1
Pipeline depth	20-30	5
Normalized performance	5-8x	1x
Normalized power	20-50x	1x
Normalized energy/instruction	4-6x	1x

Table 2: Asymmetric Multiprocessor Cores

To illustrate, let's consider a chip-level multiprocessor built with one large core and 25 small cores, with the two types of cores having a 25:1 ratio in power consumption, a 5:1 ratio in scalar performance, and a 5:1 range of energy per instruction. The chip-level multiprocessor would operate as follows:

- In phases of low thread-level parallelism, run the large core for best scalar performance
- In phases of high thread-level parallelism, run multiple small cores for best throughput performance

For simplicity, we assume that inactive cores consume no power. At any instant in time, the microprocessor may run either one large core or 25 small cores. Because the number of available software threads will vary over time, the asymmetric

multiprocessor must be capable of migrating a thread between large and small cores.

In practice, it is desirable to allow a few small cores to run simultaneously with the large core in order to minimize the throughput performance discontinuity at the point of switching off the large core. In the previous example, a discontinuity of 3 units of throughput results from switching off the large core and switching on two small cores. To minimize the percentage of the total throughput lost, we can move the discontinuity to occur with a higher number of running threads by permitting, for example, up to 5 small cores to run simultaneously with the large core. Figure 3 illustrates this concept which necessitates a small increase in power budget compared to the non-overlapping case.

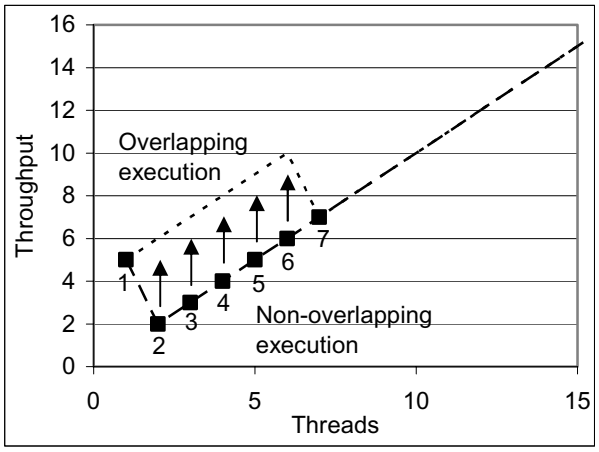

Figure 3: Overlapping Large and Small Cores

Using two types of cores representative of today's microprocessors, a 4:1 range of energy per instruction is achievable as shown in section 3. As future microprocessors continue to deliver even higher levels of scalar performance, the range of possible energy per instruction may be expected to increase to perhaps 6:1.

7. Variable-size Core

Recent research has examined the possibility of building variable-sized schedulers, caches, TLBs, and branch predictors to minimize switching capacitance (and hence energy) when large array sizes aren't needed [1, 2, 6, 10, 11, 15]. In addition to dynamically resizing arrays, it is also possible to design a large core that degrades into a small core by dynamically disabling execution units and even pipestages [5, 12, 17]. These techniques are collectively known as *adaptive processing*. A chip-level multiprocessor would operate as follows:

- In phases of low thread parallelism, run a few cores using all available resources on each core for best scalar performance
- In phases of high thread parallelism, run many cores using fewer resources on each core for best throughput performance

The net effect of reducing array sizes and disabling execution units is to reduce the capacitance toggled per instruction. However, switching capacitance cannot be reduced by as much as designing a smaller core to begin with. While unused execution hardware may be gated off, the physical size of the core does not change, and thus the wire lengths associated with the still active hardware blocks remain longer than in a small core.

We estimate the possible reduction in energy per instruction by examining the floorplan of a large out-of-order microprocessor and determining how many blocks can be turned off to convert the processor into a small in-order machine (keeping in mind that the blocks cannot be physically moved). We then quantify the percentage of CPU core area turned off, which approximates the reduction in switching capacitance. From equation (2), energy per instruction is directly proportional to the amount of switching capacitance.

A rough estimate is that up to 50% of the switching capacitance may be turned off, resulting in a 1-2x reduction in energy per instruction. Previously published results tend to be on the low end of this range because most research has concentrated on minimizing scalar performance loss rather than energy per instruction, and has considered turning off only a few portions of the CPU core. Achieving a 2x reduction in energy per instruction will mandate the use of leakage control techniques such as dynamic sleep transistors and body bias in addition to clock gating.

8. Speculation Control

Various forms of speculation control have been proposed to reduce energy wasted due to misspeculated instructions, for example, instructions following a mispredicted branch [4, 9, 22, 26]. The additional energy results from capacitance toggled to process a misspeculated instruction. While the results of the misspeculated instruction may be discarded, the energy has already been spent. This energy may be accounted for by charging it to the next correctly-speculated (retired) instruction.

Pipeline gating is a technique to avoid filling the pipeline with instructions that are likely to be discarded due to one or more low-confidence branch predictions. *Simultaneous multithreading* is also a form of speculation control since the pipeline is filled with instructions from several threads that are more likely to be retired than instructions from a single thread [27]. This is due to the shorter effective distance between fetch and branch resolution as seen by each thread. Using speculation control, a chip-level multiprocessor would operate as follows:

- In phases of low thread parallelism, run a few cores using as much speculation as possible for best scalar performance
- In phases of high thread parallelism, run many cores using little speculation on each core for best throughput performance

There exists some overlap between the variable-size core technique and speculation control because reducing the number of scheduler and reorder buffer entries also reduces the number of instructions that may be speculated. The size of other CPU resources such as caches, TLBs, and branch predictors do not have as great an effect on the amount of speculation possible.

Figure 4 shows a histogram of the percentage of instructions discarded due to misspeculation in a large out-of-order microprocessor on a suite of 57 traces from the 12 SpecInt2K benchmarks. On average, misspeculation accounts for 37% of all instructions fetched, 23% of all instructions decoded, and 19% of all instructions executed. The variation in the data is large: on some traces, 65% of all instructions fetched may be discarded due to misspeculation, whereas on other traces, the percentage may be as small as 4%. Aragon [4] estimates that 28% of the power consumed in a typical microprocessor is attributable to misspeculation on benchmarks with high branch misprediction rates. Thus, if we were to completely eliminate all misspeculation, we would expect up to a 1.4x reduction in energy per instruction. Note that applications that can be parallelized usually do not suffer from high misspeculation.

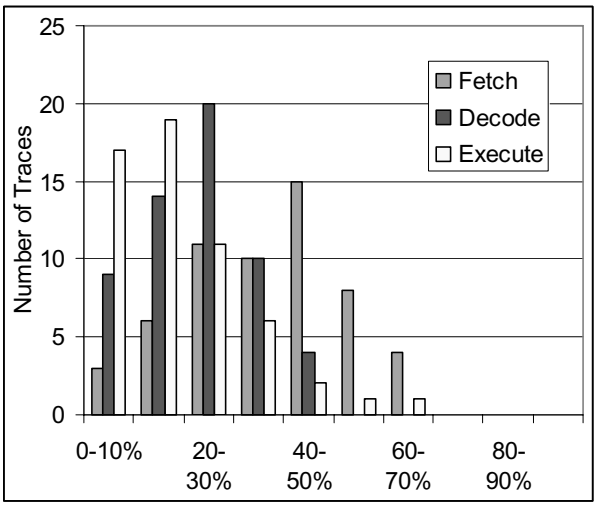

Figure 4: Misspeculated Instructions

9. Comparison

A comparison of the four techniques for varying energy per instruction is shown in Table 3.

Method	EPI Range	Time to Vary EPI
Voltage frequency scaling	1:2 to 1:4	100us (ramp Vcc)
Asymmetric cores	1:4 to 1:6	10us (migrate 256KB L2 cache)
Variable-size core	1:1 to 1:2	1us (fill 32KB L1 cache)
Speculation control	1:1 to 1:1.4	10ns (pipeline latency)

Table 3: Comparison of Energy/Instruction Techniques

While we have evaluated each technique independently, some techniques may be used in combination to further extend the range of energy per instruction. For example, voltage/frequency scaling may be used in combination with asymmetric cores to vary energy per instruction over a 24:1 range.

Table 3 also presents an order-of-magnitude estimate of the amount of time required to vary energy per instruction over the specified range. The amount of time varies enormously between the four techniques. Slower techniques will require longer program phases to realize the benefits.

10. An Energy Per Instruction Throttle

For each of the techniques we've examined, on-chip hardware regulates the operation of the multiprocessor

to maintain total chip power within a fixed power budget. The regulator hardware satisfies equation (3) by varying the amount of energy per instruction in inverse proportion to the aggregate number of instructions retired per second. In response to an over power situation, the regulator takes one or more of the following actions:

- Lower voltage and frequency (in the case of voltage/frequency scaling)
- Migrate threads from large cores to small cores (in the case of asymmetric cores)
- Reduce the capacity of processor resources (in the case of variable-size cores)
- Reduce the amount of speculation (in the case of speculation control)

In each case, software views the processor as a large symmetrical chip-level multiprocessor, albeit with the unusual property that existing threads become slower as the software asks hardware to run more threads simultaneously, even though net throughput increases. With this approach, software written for today's shared-memory multiprocessor programming model will continue to run without modification. Future work may examine the possibility of using software to manage energy policies.

11. Conclusion

In this paper, we have shown that the key to achieving both high scalar performance and high throughput performance is to dynamically vary the amount of energy expended to process each instruction according to the amount of parallelism available. We've surveyed four approaches to achieving a range of energy per instruction and shown that the most promising direction is a combination of asymmetric cores and voltage/frequency scaling. Developing the microarchitecture and software model for such a multiprocessor are areas of future work.

12. Acknowledgements

The authors would like to acknowledge the contributions of Murali Annavaram, Bryan Black, Shekhar Borkar, Ned Brekelbaum, Doug Carmean, Glenn Hinton, Doron Orenstein, Jeff Rupley, and Gad Sheaffer.

13. References

[1] Albonesi, D.H.; "Selective cache ways: on-demand cache resource allocation", Proceedings. 32nd Annual International Symposium on Microarchitecture, 1999.

[2] Albonesi, D.H.; Balasubramonian, R.; Dropsho, S.G.; Dwarkadas, S.; Friedman, E.G.; Huang, M.C.; Kursun, V.; Magklis, G.; Scott, M.L.; Semeraro, G.; Bose, P.; Buyuktosunoglu, A.; Cook, P.W.; Schuster, S.E.; "Dynamically tuning processor resources with adaptive processing", Computer , Volume 36, Issue 12, Dec. 2003.

[3] AMD PowerNow! Technology, http://www.amd.com/us-en/assets/content_type/DownloadableAssets/Power_Now2.pdf

[4] Aragon, J.L.; Gonzalez, J.; Gonzalez, A.; "Power-aware control speculation through selective throttling", Proceedings The Ninth International Symposium on High-Performance Computer Architecture, 2003.

[5] Bahar, R.I.; Manne, S.; "Power and energy reduction via pipeline balancing ",Proceedings 28th Annual International Symposium on Computer Architecture, 2001.

[6] Baniasadi, A.; Moshovos, A.; "Branch predictor prediction: a power-aware branch predictor for high-performance processors", Proceedings IEEE International Conference on Computer Design, 2002.

[7] Barroso, L.A.; Gharachorloo, K.; McNamara, R.; Nowatzyk, A.; Qadeer, S.; Sano, B.; Smith, S.; Stets, R.; Verghese, B.; "Piranha: a scalable architecture based on single-chip multiprocessing", Proceedings of the 27th International Symposium on Computer Architecture, 2000.

[8] Brooks, D.; Martonosi, M.; "Dynamic thermal management for high-performance microprocessors", The Seventh International Symposium on High-Performance Computer Architecture, 2001.

[9] Buyuktosunoglu, A.; Karkhanis, T.; Albonesi, D.H.; Bose, P.; "Energy efficient co-adaptive instruction fetch and issue", Proceedings 30th Annual International Symposium on Computer Architecture, 2003.

[10] Delaluz, V.; Kandemir, M.; Sivasubramaniam, A.; Irwin, M.J.; Vijaykrishnan, N.; "Reducing dTLB energy through dynamic resizing", Proceedings 21st International Conference on Computer Design, 2003.

[11] Dhodapkar, A.S.; Smith, J.E.; "Managing multi-configuration hardware via dynamic working set analysis", Proceedings 29th Annual International Symposium on Computer Architecture, 2002.

[12] Efthymiou, A.; Garside, J.D.; "Adaptive pipeline depth control for processor power-management", Proceedings IEEE International Conference on Computer Design, 2002.

[13] Figueiredo, R.J.O.; Fortes, J.A.B.; "Impact of heterogeneity on DSM performance", Proceedings Sixth

International Symposium on High-Performance Computer Architecture, 2000.

[14] Fleischmann, M. “LongRun Power Management”, http://www.transmeta.com/pdfs/paper_mfleischmann_17jan01.pdf, 2001

[15] Folegnani, D.; Gonzalez, A.; “Energy-effective issue logic”, Proceedings 28th Annual International Symposium on Computer Architecture, 2001.

[16] Gebotys, C.H.; Gebotys, R.J.; "Power minimization in heterogeneous processing", Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, 1996.

[17] Gonzalez, J.; Gonzalez, A.; “Dynamic cluster resizing”, Proceedings 21st International Conference on Computer Design, 2003.

[18] Intel® Pentium® 4 Processor in the 423-pin Package at 1.30 GHz, 1.40 GHz, 1.50 GHz, 1.60 GHz, 1.70 GHz and 1.80 GHz Datasheet, 2001. Pages 78-79.

[19] Intel® Pentium® M Processor Datasheet, April 2004. Pages 13-14.

[20] Kumar, R.; Farkas, K.I.; Jouppi, N.P.; Ranganathan, P.; Tullsen, D.M, “Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction”, Proceedings 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003.

[21] Kumar, R.; Tullsen, D.; Ranganathan, P.; Jouppi, N.; Farkas, K. "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance". Proceedings 31st International Symposium on Computer Architecture, 2004

[22] Manne, S.; Klauser, A.; Grunwald, D.; “Pipeline gating: speculation control for energy reduction”, Proceedings the 25th Annual International Symposium on Computer Architecture, 1998.

[23] Menasce, D.; Almeida, V.; "Cost-performance analysis of heterogeneity in supercomputer architectures", Proceedings of Supercomputing '90.

[24] Morad, T. Y.; Weiser, U.; Kolodny, A.; "ACCMP - Asymmetric Chip Multi-Processing", CCIT Technical Report #488, http://www.ee.technion.ac.il/morad/publications/accmptr.pdf, June 2004.

[25] Oka, M.; Suzuoki, M.; “Designing and programming the emotion engine”, IEEE Micro, Volume 19, Issue 6, Nov-Dec 1999.

[26] Parikh, D.; Skadron, K.; Zhang, Y.; Barcella, M.; Stan, M.R.; “Power issues related to branch prediction”, Proceedings Eighth International Symposium on High-Performance Computer Architecture, 2002.

[27] Seng, J.S.; Tullsen, D.M.; Cai, G.Z.N.; “Power-sensitive multithreaded architecture”, Proceedings IEEE International Conference on Computer Design, 2000.

[28] TOP500 Supercomputer Sites List for November 2003, http://www.top500.org/list/2003/11.

[29] Tschanz, J.; Narendra, S.; Yibin Ye; Bloechel, B.; Borkar, S.; Vivek De; “Dynamic-sleep transistor and body bias for active leakage power control of microprocessors”, Digest of Technical Papers, IEEE International Solid-State Circuits Conference, 2003.

[30] Turner, V; Willard, C; "Sun's Throughput Computing Strategy to Create a Quantum Change in Server Performance", http://www.sun.com/processors/whitepapers/idc_whitepaper.pdf, February 2004.