Power Efficient Processors Using Multiple Supply Voltages*

Diana Marculescu
Dept. of Electrical and Computer Engineering
Center for Electronic Design Automation
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract -*This paper presents a study of different power metrics for varying microarchitectural configurations and proposes an efficient scheme to reduce the energy requirements of superscalar, out-of-order processors. Specifically, we propose the use of multiple supply voltages at microarchitectural level by exploiting the difference in latencies of different pipeline stages or modules. The proposed scheme is a simple and efficient way of reducing the energy requirements by up to 27%, without affecting the effective performance of the processor.*

1 Introduction

In recent years, power dissipation has become a critical design concern not only for designers of battery powered or wireless electronics, but also in the case of high-performance microprocessors, multimedia and digital signal processors or high-speed networking. While it is generally agreed that tools for power estimation and optimization do exist for hardware specifications at different levels of abstraction (circuit, gate, register-transfer or behavioral) [1], less has been done in the area of power analysis or optimization at microarchitecture, architecture or system level [2]. Having tools that are able to quantify the effect of different performance or power optimization schemes for a piece of code running on a given processor is of extreme importance for computer architects and compiler engineers who can characterize different architecture styles not only in terms of their performance, but also in terms of the corresponding energy efficiency. This will also enable the fine tuning of any existing energy/performance trade-offs.

Relevant previous work comes from several different areas such as software power modeling for general purpose, embedded or Digital Signal Processing (DSP) applications, compilation techniques for power optimized software, pipeline scheduling for low power, and energy efficient memory systems. Specifically, [3] proposes a *per-instruction base power model* that can be easily used to find an aggregate power estimate for a sequence of instructions. The *inter-instruction overhead* due to changes in circuit state are also taken into account, but their effect is found to be negligible. The approach presented in [4] targets instruction scheduling to reduce circuit state overhead. The proposed *cold scheduling* technique reduces the switching activity in the control path by reordering the instructions based on the inter-instruction energy overhead. For the special case of real-time systems, an approach for reducing energy via scheduling, while still meeting the deadlines, has been presented in [5]. In [6], the case of DSP applications is addressed. There, while the same type of models as in [3] have been developed, the inter-instruction effects turn out to be much more prevalent, thus making possible to develop instruction scheduling techniques that target power minimization. In [7] a more efficient model with space complexity linear in the instruction set size has been presented.

More recently, [8] and [9] present two more advanced microarchitectural power simulators. In [8], a very accurate parameterized power simulator (within 10% when compared to three different high-end microprocessors) is presented, as well as some interesting trade-offs between energy and performance under varying microarchitecture settings. In [9], the case of datapath dominated architectures is considered, as well as an analysis of the impact of compiler optimizations and memory design on power consumption values. In the area of energy optimization, the authors of [10] present an architectural enhancement to reduce the extra work or energy in the pipeline of a superscalar processor due to mispredicted branches, without significant loss in performance. In [11] a technique for reducing the average power consumption for the pipeline structure is presented. Other approaches target techniques for energy efficient memory systems, such as the use of selective cache ways [12], filter-caches [13] or code compression [14]. Several approaches in the most recent literature on power efficient architecture point out that different microarchitectural settings do provide different values for energy per instruction [25] or energy delay product [8]. It has also been observed that there exists a wide variation from one application to another, as well as between different parts of the same application [26] both in terms of the inherent parallelism and in terms of the necessary resources to sustain a given performance level.

In this paper we address the problem of power optimization of processors using *multiple supply voltages*. While the same technique has been proposed for the case of high level synthesis for low power [16,17,18], its use has not been explored in the case of low power microprocessors. Since the dynamic component of energy consumption is quadratically dependent on the supply voltage ($E \propto CV_{dd}^2$), moderate reductions in the supply voltage will produce great savings in the power consumption. However, the price that needs to be paid is higher delay: $D \propto V_{dd}/(V_{dd}-V_T)^{\alpha}$ where V_T is the threshold voltage and α is strongly dependent on the mobility degradation of electrons in transistors (with typical value between 1 and 2 [15]). However, as we shall see later, it is possible to get power savings without performance penalty since some pipeline stages may dictate the operating clock frequency (such as I-cache or D-cache accesses, register file accesses, wake-up and selection logic for out-of-order processors), whereas others may be off of the critical path (such as ALUs, etc.). As pointed out in [27,28,29], the most important in determining the clock speed are the cache and register file cycle times and the scheduling logic (dependence check, wake-up and select, bypass logic). However, as we will see in the sequel, there exists a lot of variation among different microarchitectural configurations as far as the latency of each of the modules is concerned. This variation can be exploited to reduce the power consumption by selectively applying lower voltages to pipeline stages or modules that are guaranteed to have a lower latency than the critical one. A similar problem has been addressed in [30] where the problem of joint optimization of performance and clock rate has been addressed and Complexity-Adaptive Processors (CAPs) are proposed. CAPs provide many different performance/clock rate trade-off points within a single hardware implementation, but do so while maintaining the high clock rate of a fixed architecture.

As pointed out in [27], when characterizing the performance of modern processors, the *CPI* (Cycles per Instruction) is only one of the two parameters that needs to be considered, the second one being the actual cycle time. Indeed, it is shown that with increasing complexities of modern architectures (wider issue rates, larger instruction windows, etc.) the *CPI* does decrease, but also the critical path delay increases at a higher rate than *CPI* decreases. Thus, the product $CPI*T_{cycle}$ is a more accurate measure for characterizing the performance of modern processors that use a significant amount of dynamic scheduling and out-of-order execution to achieve high levels of

.*This research was supported in part by NSF CAREER Award MIP-0084479.

performance. In the case of power consumption, most researchers have concentrated on the problem of estimation or optimization of *energy per committed instruction* (EPI) or *energy per cycle* (EPC) [8,9,25,26]. While in the case of embedded computer systems with tight power budgets some performance may be sacrificed for lowering the power consumption, in the case of high performance processors this is not desirable and solutions that jointly address the problem of low power and high performance are needed. To this end, we propose the *energy delay product per instruction* (EDPPI) defined as $EPI*CPI*T_{cycle}$ as a measure that characterizes both the performance and power efficiency of a given architecture. Such a measure can identify microarchitectural configurations that keep the power consumption to a minimum, without significantly affecting the performance. In addition to classical metrics (such as *EPC* and *EPI*), we use this measure to assess the efficiency of our power optimization technique and to compare different configurations as far as power consumption is concerned.

To validate our results, we use a microarchitecture-level power simulator developed recently [8]. The underlying tool is SimpleScalar, a widely-used detailed performance simulator [20], augmented with information about both dynamic and static power consumption. As the experimental results show, significant savings can be obtained in both energy and power consumption, ranging from 10% to 27%, depending on the microarchitectural settings of the processor.

The paper is organized as follows: in Section 2 we present the rationale for using multiple supply voltages at microarchitectural level. In Section 3, we present our assumptions and basic approach for reducing power consumption. Section 4 shows our experimental results for the proposed technique applied on a subset of SpecInt95 benchmarks. We conclude in Section 5 with some final remarks.

2 Delay variation with microarchitectural settings

In what follows, we consider a typical superscalar configuration, based on the reservation station model (Fig.1). This structure is used in modern processors like Pentium Pro and PowerPC 604. The main difference between this structure and the one used in other processors (like MIPS R10000, DEC 21264, HP PA-8000) is that the reorder buffer holds speculative values and the register file holds only committed, non-speculative data, whereas for the second case, both speculative and non-speculative values are in the register file. However, the wake-up, select and bypass logic are common to both types of architectures and, as pointed out in [27], their complexity increases significantly with increasing issue widths and window sizes.

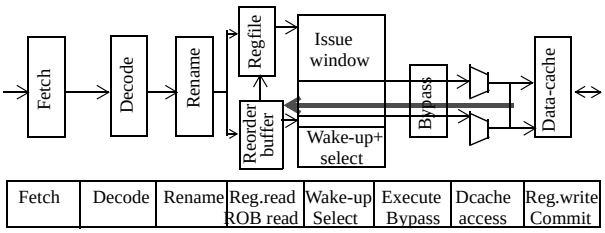


Fig.1 The reservation station model

We review in the following the analytical delay models that have been previously proposed for the most critical modules or stages in a typical out-of-order superscalar processor. As it will be seen in the sequel, the latency that dictates the clock period of the processor is highly dependent on microarchitectural features that are used to increase performance, such as issue width and instruction window size.

Analytical delay models for the dynamic scheduler

This mainly concerns the dispatch and issue logic and basically determines the amount of parallelism that can be exploited. The main issue with these structures is the fact that they rely on broadcast operations on long wires and thus their delay will not scale with shrinking device sizes [27]. In addition, their complexity and delay is quadratically dependent on the *issue width* (*IW*) and *instruction window size* (*WS*).

Rename logic

This structure is responsible for mapping logical registers into physical registers and uses a map table and dependency check logic that detects true dependencies (*read after write* or *RAW* dependencies), that is, whether a current source register is one of the previous destination registers that has been already renamed in a previous clock cycle. The complexity of the rename logic increases quadratically with *IW* due to dependency check logic. As shown in [27], the delay of the renaming logic is determined by the map table since the dependency checking is done in parallel and typically is faster than accessing the map table. The map table is assumed to have a CAM (content addressable memory) structure. The delay of the renaming logic has the form [27]:

$$Delay_{rename} = a_0 + a_1 IW + a_2 IW^2 \quad (1)$$

where a_0, a_1, and a_2 are technology dependent constants. *Wake-up logic*

The wake-up logic is responsible for updating source dependencies of instructions in the window waiting for their source operands to become available. It relies on broadcasting produced results to all instructions in the instruction window, and thus it's complexity is a function of the windows size (*WS*). The underlying structure is again a CAM and the delay for accessing an entry has the form:

$$Delay_{wake-up} = b_0 + (b_1 + b_2 IW)WS + (b_3 + b_4 IW + b_5 IW^2)WS^2 \quad (2)$$

As it can be seen, as larger windows and wider issue widths are used, the delay increases quadratically. However, the gain in performance (i.e., decrease in *CPI*) increases at a much lower rate and tapers off after a while. Again, b_0-b_5 are technology dependent constants.

Selection logic

The selection logic selects instructions to be executed from the pool of ready instructions in the issue window. Since more instructions than available resources may be ready for execution, some sort of policy for selecting a subset of them is needed, such as *oldest ready first*. For a scheme that assumes selection is done based on the position of the ready instruction in the issue window (as it is done in HP PA-8000), the delay of the selection logic increases with the window size as follows:

$$Delay_{select} = c_0 + c_1 \log_4(WS) \quad (3)$$

Assuming a 0.35 μm technology, with V_{dd} = 2.5V and V_T = 0.67V, the following variation with issue width and window size is observed for the delay of various structures of dynamic scheduler (Table 1). Some of the delay values have been reported in [27] whereas other were obtained using the models in (1)-(3). As it can be seen, the wake-up and select logic are the bottleneck since they have to be completed in the same cycle. Also, the corresponding delay grows fast with the increasing *WS* or *IW.*

Analytic delay models for the execute-bypass stage

The data bypass logic is responsible for bypassing result values to subsequent instructions from instructions that have completed execution, but have not yet written their results into the register file. However, the delay of bypass logic may become dominant with shrinking technology sizes and increasing *IW.* This is because the key factor that determines the speed of bypass logic is the delay of the result wires that are used to transmit bypassed values. The wire length is proportional to *IW* and to the height of functional units available for execution during each clock cycle. It is also a function of the layout used; for example, a clustered layout (with bypassing within a group of functional units and

registers) can be more suitable and able to overcome the increasing contribution of bypass logic delay. .

Table 1: Delay of dynamic scheduling logic

WS (Instr.)	IW (Instr./cycle)	Rename delay (ps)	Wake-up delay (ps)	Select (ps)
8	1	625.4	214.4	1112.9
16			218.8	1113.0
32			222.6	1997.5
64			248.5	1997.4
8	2	649.4	255.8	1112.9
16			263.7	1113.0
32			274.1	1997.5
64			308.3	1997.4
8	4	698.5	332.6	1112.9
16			345.4	1113.0
32			366.7	1997.5
64			419	1997.4
8	8	800.8	463.2	1112.9
16			476.9	1113.0
32			510.1	1997.5
64			605.0	1997.4

Table 2: Delay of bypass logic

WS (Instr.)	IW (Instr./cycle)	Bypass delay (ps)
8	1	12.1
16		14.9
32		21.4
64		38.1
8	2	44.1
16		50.4
32		64.4
64		97.6
8	4	168.1
16		184.5
32		219.5
64		298.82
8	8	656.4
16		704.7
32		806.5
64		1030.6

Analytically, for a scheme that does not use clustering, the delay of the bypass logic has the form:

$$Delay_{bypass} = d_0\left(WS(d_1 + d_2 IW) + \sum_{i=1}^{IW} height(FU_i)\right)^2 \quad (4)$$

where the first term account for the height of the register file and the summation accounts for the height of all functional units that are available to execute in parallel during every cycle. Assuming that all functional units are ALUs performing general computations (adding, shifting, logic operations), we get the values in Table 2 in a 0.35μm technology. Thus, when considering the latency of the execute-bypass stage, one has to consider the delay of an ALU plus the bypass logic (all other functional units are considered pipelined). However, with shrinking technologies, the delay of the ALU scales down, but the bypass logic delay will be unchanged since wire delays do not scale with smaller feature sizes.

Delay models for the register file and caches

The register file is accessed in the register and commit stages when either operands have to be read or results are committed into the architected register file. It has been shown in [28] that the register file delay increases with *IW* since the number of ports needed is proportional to the issue width. The model is similar to the one for the CAM structure used in the wake-up logic, but uses bitlines and wordlines instead of taglines and matchlines. For the case of I-cache and D-cache, we consider the model developed in [29] which provides a complete analysis of the access and cycle time as a function of the cache size, block size and associativity. To minimize the access time, the cache hardware organization is determined using *Cacti* tools [29]. The same model has been used in [8] to reorganize caches for minimum access time. We provide in Table 3 the cycle times for a register file of size 32, a direct mapped I-cache of size 16K with a block size matching the fetch rate (varying between 8, 16, 32, 64B) and a 4-way set associative D-cache of size 16K with 32B blocks. The values have been scaled down from the reported values in [28,29] from a 0.5μm (register file) and a 0.8μm (cache) technology to the 0.35μm case considered by us.

Table 3: Delay of register file and caches

IW (Instr./cycle)	Register file (ns)	I-cache (ns)	D-cache (ns)
1	2.2	4.1	4.9
2	2.3	3.6	
4	2.5	3.5	
8	2.9	3.3	

As it can be seen in Tables 1-3, the stages that have the highest latency for the considered configurations are the D-cache, I-cache, bypass and execute, register file, wake-up and select and rename, in this order. However, as pointed out in [27], the wake-up and select logic have to be performed atomically, whereas the delay of all the others can be reduced either by pipelining (e.g., cache accesses) or by clustering (bypass logic). In addition, with shrinking feature sizes, some of the delays will scale differently, depending on the breakdown in terms of gate and wire delay. Nonetheless, it is clear that there exists a wide variation in terms of latencies across different pipeline stages. This variation can be exploited for running slower stages at a lower voltage, while keeping the operating clock frequency the same. The next section examines how this can be achieved.

3 Using multiple supply voltages at microarchitectural level

The problem of using multiple supply voltages for reducing the power requirements of a given design has been studied in the area of high level synthesis for low power [16,17]. It has been pointed out that, while it is theoretically possible to have many supply voltages available, or voltage values that are exactly matched with the latency of each module, such a scenario is too expensive to use in practice. Instead, from a practical point of view, the availability of a small number of supply voltages (typically two or three) is a reasonable assumption.

In the case of low power high level synthesis using multiple supply voltages the slack available in the scheduling of different operations in a dataflow graph (DFG) is exploited so that modules not on the critical path can be run at a lower voltage, possibly over many clock cycles. The DFG is assumed scheduled and allocated and voltages are assigned so that the total energy consumption is optimized under given performance constraints [17]. The problem of both pipelined and non-pipelined DFGs has been addressed and savings of up to 60-70% in some cases have been reported.

In the case of out-of-order superscalar processors, the basic structure is a linear pipeline with some feedback due to bypass logic and reading/writing of registers. Assigning voltages to reduce power consumption can be done since, as shown previously, there exists some slack between the slowest pipeline stages and the rest. In addition, as it will be shown in the experimental part, the optimal energy configuration (defined by either minimum *EPI* or *EDPPI*) is in most of the cases not the one that uses large issue widths or window sizes. Thus, a larger slack between the cache accesses and other stages of the pipeline can become manifest, thus providing more opportunities for running faster stages at a lower voltage. Our goal is to keep everything within the given cycle time, so that there is no performance penalty.

While we do consider assigning different voltages for the purpose of reducing power consumption, we do keep the global clock lines powered at the higher V_{dd}. This will limit the amount of savings that can be achieved since the clock power consumption can be significant (up to 40% [2]). In what follows, we consider the following simplifying assumptions:

- The power and delay overhead due to voltage level converters is negligible;
- The power and delay overhead of the multiple voltage DC-DC converter compared to the single voltage DC-DC converter is negligible.

4 Experimental results

To report our experimental results, we have used the SimpleScalar microarchitecture simulator [20] augmented with information about power consumption (*Wattch* [8]). SimpleScalar performs fast, flexible and accurate simulation of modern processors that implement a derivative of the MIPS-IV architecture [22]. *Wattch* assumes that aggressive clock gating is used to reduce the dynamic power consumption whenever a hardware unit is not in use [23].

Table 4: Processor configurations

Parameter	Value
RUU size	8-64 instructions
LSQ size	4-32 instructions
Fetch queue size	1-8 instructions
Fetch width Decode width Issue width Commit width	1-8 instructions/cycle
Functional units	1- 8 IntALUs, 1 IntMult/Div 1- 8 FPALUs, 1 FPMult/Div
Branch prediction	Bimodal, 2K table
BTB	2K, 4-way
Return-address stack	8 entry
Mispredict penalty	3 cycles
L1 D-cache	16K, 32B blocks, 4-way, LRU 1 cycle latency
L1 I-cache	16K, (8, 16, 32, 64)B blocks, direct-mapped, LRU, 1 cycle latency
L2	256K, 64B blocks, 4-way, LRU 6 cycles latency
Memory	10 cycles latency
DTLB	512K, 4K blocks, 4-way, LRU 10 cycles miss latency
ITLB	256K, 4K blocks, 4-way, LRU 10 cycles miss latency

In addition, the power model used in [8] is *parameterized* so that the power consumption is scaled down according to the number of resources (in the case of multiple functional units) or ports used (for the register file or caches) during a given clock cycle. For the purpose of reporting the results, the simulator was run in the high-performance mode which uses a fairly accurate branch prediction mechanism. We consider the configurations specified in Table 4 for a typical superscalar microprocessor. Some of the microarchitecture settings are the default values in the SimpleScalar [20] simulation environment. Every given value of the fetch rate (1-8) is matched with a corresponding issue, dispatch and commit rate, as well as a corresponding number of available functional units. In addition, for all configurations the I-cache size is 16K, but with varying line size to match the increasing fetch rate.

The purpose of our experimental study is twofold:

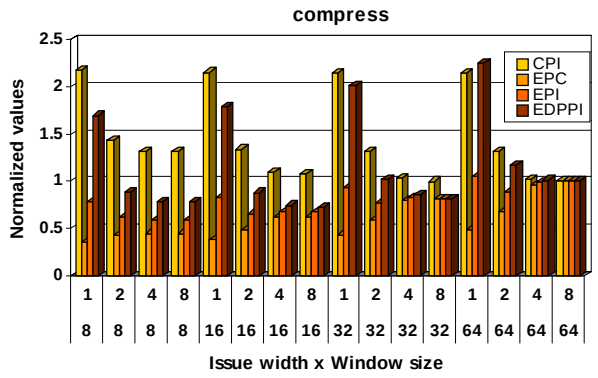
(i) To compare various microarchitectural configurations in terms of power consumption.

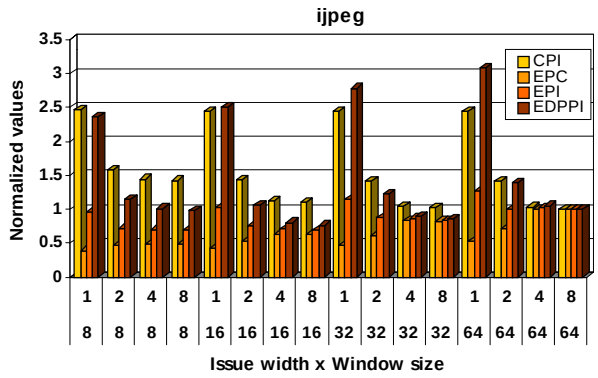
(ii To assess the efficiency of our proposed multiple supply voltage scheme in different architectural settings.

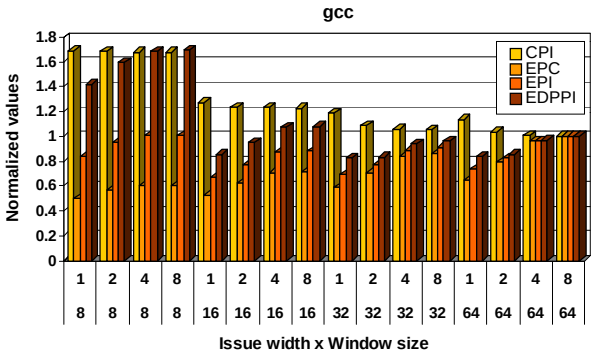
Energy metrics

In the following, we present our results on a subset of the SpecInt95 benchmark suite. In Fig.2-3 we report the normalized values for *CPI* (cycles per instruction), *EPC* (energy per cycle), *EPI* (energy per instruction) and *EDPPI* (energy delay product per instruction). All values have been obtained assuming a 0.35μm technology, with V_{dd} = 2.5V.

In all cases reported, the best performance is obtained when a wider issue rate and/or a large instruction window is used. *CPI* steadily decreases when *IW* is increased, although in some cases (e.g., *gcc*), the dependence on *WS* is more prevalent. However, in most cases, going from a window size of 32 to 64 brings almost no improvement in terms of performance. This actually confirms that increasing complexity comes at an increasing price in both area and delay, but does not provide a significant improvement of performance [27]. On the other hand, the average power consumption (*EPC*) is usually minimized for lower values of *IW* and *WS* but this reduction comes at the price of decreased performance. In fact, the total energy consumed during the execution of a given benchmark may actually increase due to increased idleness of different modules. To characterize the actual energy consumption, the energy per committed instruction (*EPI*) is a more appropriate measure. While in some cases (*compress*, *ijpeg*) *EPI* decreases with higher *IW* and increases with higher values of *WS*, there are cases where *EPI* decreases with increasing *IW* (*gcc*) or oscillates when either *IW* or *WS* are varied (*lisp*). This shows that there is no clear relationship between the microarchitectural settings and the energy needed to accomplish a given task. The energy-optimal configuration is very different from one application to another, as it can also be among different parts of the same application [26]. For example, an issue width of 8 with a window size of 8 minimizes *EPI* for *ijpeg*, whereas the others have varying optimal configurations. If, however, the highest energy reduction with lowest performance penalty is sought, in all cases but one (*gcc*) the optimal configuration (i.e., lowest EDPPI) is *IW* = 8 and *WS* = 16. Although the energy is not minimized in this cases, the penalty in performance is less than in other cases with similar energy savings.







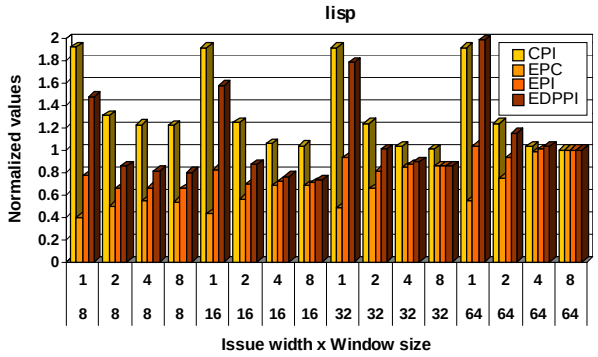
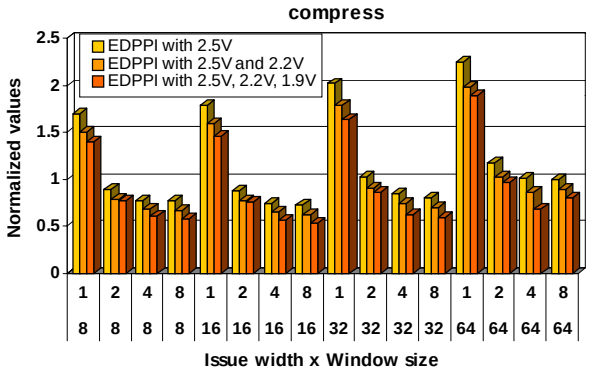
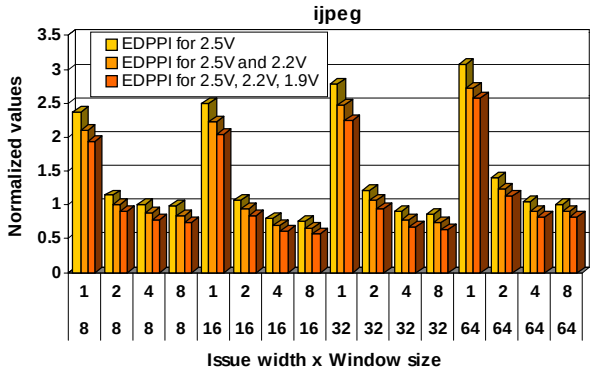
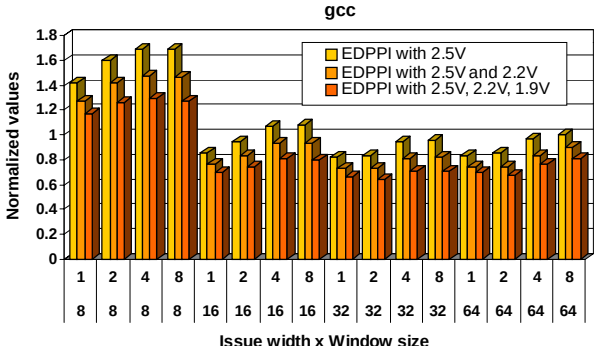


Fig.2 Detailed analysis for some SpecInt95 benchmarks







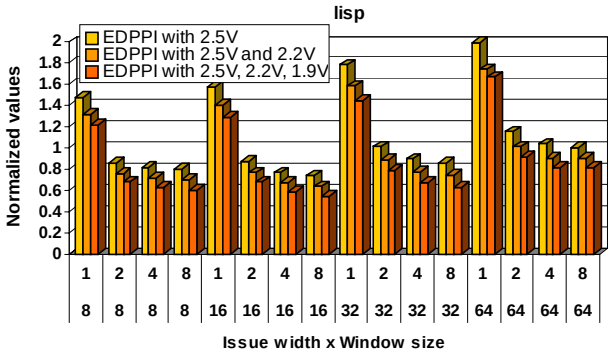


Fig.3 Energy savings using multiple supply voltages

Energy reduction using multiple supply voltages

To this end, we will consider the effect of using multiple voltages assuming that up to three supply voltages are available: V_{dd1} = 2.5V, V_{dd2} = 2.2V, V_{dd3} = 1.9V. We report in Fig.4-5 the savings in *EDPPI* obtained when using multiple supply voltages when compared to the original single voltage scheme. Since changing the voltage does not change the *CPI* or T_{cycle}, the same savings will be obtained for *EPI* or *EPC*. The energy savings obtained using an additional, lower voltage supply of 2.2V reduces the energy consumption by about 10%, whereas adding another voltage supply (1.9V) further reduces power by another 17%. The savings obtained have little variation from one benchmark to another, the main reason residing in the fact that the power breakdown among different stages does not vary much. In addition, since the clock power (up to 40% or more in some cases) is not scaled down, the savings are limited by the different modules that have an available slack that can be exploited. However, these savings come with no performance penalty and the throughput is maintained, while preserving both *CPI* and cycle time T_{cycle}. Also, by choosing different voltage values (lower than what has been considered) it is conceivable that the reduction observed when using this scheme will further increase.

5 Conclusion and discussion

In this paper, we have presented a study of different power metrics for varying microarchitectural configurations and a promising scheme to reduce the energy requirements of superscalar, out-of-order processors. The variation of energy (or energy delay product) per committed instruction with different architectural settings provides an insight into finding energy-optimal settings for a given application. Such a configuration could be found in a run-time environment either for a given application, or on a finer grain, for a given computational kernel belonging to an application. The multiple voltage supply scheme is a simple and efficient way of reducing the energy requirements by up to 27% (for the given set of voltage values) of a processor, with no performance overhead. Since a completely synchronous scheme requires the existence of a global clock with high power overhead, to achieve more significant savings the multiple voltage supply solution could be applied in a globally asynchronous, locally synchronous architecture where the overhead of communication among different modules is minimized. Nonetheless, the paradigm of running slower stages at a lower voltage could be employed in a run-time environment that is able to adjust the voltage and clock frequency dynamically, on a fine grain, to fit the application needs.

References

[1] J. Mermet and W. Nebel, 'Low Power Design in Deep Submicron Electronics,' Kluwer Academic, Norwell, MA, 1997.

[2] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, F. Baez, 'Reducing Power in High-Performance Microprocessors,' in *Proc. ACM/IEEE Design Automation Conference*, pp.732-737, June 1998.

[3] V. Tiwari, S. Malik, and A. Wolfe, 'Power Analysis of Embedded Software: A First Step Toward Software Power Minimization,' in *IEEE Trans. on VLSI Systems*, vol.2, no.4, pp.437-445, April 1994.

[4] C.L. Su, C.-Y. Tsui, and A.M. Despain, 'Saving Power in the Control Path of Embedded Processors,' in *IEEE Design and Test of Computers*, vol.11, no.4, Dec. 1994.

[5] S.T. Cheng, C.M. Chen, J.W. Huang, 'Low-Power Design for Real-Time Systems,' in *Real-Time Systems*, vol.15, no.2, pp.131-148, Sept. 1998.

[6] M.T.-C. Lee, V. Tiwari, S. Malik and M. Fujita, 'Power Analysis and Minimization Techniques for Embedded DSP Software,' in *IEEE Trans. on VLSI Systems*, vol.5, no.1, pp.123-135, Jan. 1997.

[7] B. Klass, D.E. Thomas, H. Schmit, D.E. Nagle, 'Modeling Inter-Instruction Energy Effects in a Digital Signal Processor,' in *Power-Driven Microarchitecture Workshop*, in conjunction with *Intl. Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[8] D. Brooks, V. Tiwari, and M. Martonosi, 'Wattch: A Framework for Architectural-Level Power Analysis and Optimizations,' in *Proc. Intl. Symposium on Computer Architecture*, Vancouver, BC, Canada, June 2000.

[9] N. Vijaykrishnan, M. Kandemir, M.J. Irwin, H.S. Kim, and W. Ye, 'Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower,' in *Proc. Intl. Symposium on Computer Architecture*, Vancouver, BC, Canada, June 2000.

[10] S. Manne, A. Klauser, and D. Grunwald, 'Pipeline Gating: Speculation Control for Energy Reduction,' in *Proc. Intl. Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[11] T.M. Conte, K.N. Menezes, S.W. Sathaye, and M.C. Toburen, 'System-Level Power Consumption Modeling and Trade-off Analysis Techniques for Superscalar Processor Design,' to appear in *IEEE Transactions on VLSI Systems.*

[12] D. Albonesi, 'Selective Cache Ways: On-Demand Cache Resource Allocation,' in *Proc. Intl. Symposium on Microarchitecture (MICRO-32)*, Haifa, Israel, pp.248-259, Nov. 1999.

[13] J. Kin, M. Gupta, and W. Mangione-Smith, 'The Filter Cache: An Energy Efficient Memory Structure,' in *IEEE Micro*, Dec.1997.

[14] H. Lekatsas, J. Henkel, and W. Wolf, 'Code Compression for Low Power Embedded System Design,' in *Proc. ACM/IEEE Design Automation Conference*, Los Angeles, CA, June 2000.

[15] T. Ishihara and H. Yasuura, 'Voltage Scheduling Problem for Dynamically Variable Voltage Processors,' in *Proc. ACM Intl. Symposium on Low Power Electronics and Design*, pp.197-202, Monterey, CA, Aug. 1998.

[16] J.-M. Chang and M. Pedram, 'Energy Minimization Using Multiple Supply Voltages,' in *IEEE Trans. on VLSI Systems*, vol.5, no.4, pp.425-435, Dec. 1997.

[17] S. Raje and M. Sarrafzadeh, 'Variable Voltage Scheduling,' In *Proc. ACM Intl. Symposium on Low Power Design*, pp.9-14, Dana Point, CA, April 1995.

[18] T. Pering, T. Burd, and R. Brodersen, 'Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System,' in *Power-Driven Microarchitecture Workshop*, in conjunction with *Intl. Symposium on Computer Architecture*, Barcelona, Spain, June 1998.

[19] G. Cai and C.H. Lim, 'Architectural Level Power/Performance Optimization and Dynamic Power Estimation,' in *Proc. Intl. Symposium on Microarchitecture (MICRO-32), Cool Chips tutorial*, Haifa, Israel, Nov. 1999.

[20] D. Burger, T.M. Austin, 'The SimpleScalar Tool Set, Version 2.0,' *CSD Technical Report #1342*, University of Wisconsin-Madison, June 1997.

[21] 'Advanced Configuration and Power Interface Specification,' Intel, Microsoft, Toshiba, Revision 1.0b, Feb. 2, 1999, at *http://www.teleport.com/~acpi/DOWNLOADS/ACPIspec10b.pdf*.

[22] C. Price, 'MIPS IV Instruction Set, revision 3.1.,' MIPS Technologies, Inc., Mountain View, CA, Jan. 1995.

[23] S. Gary, 'Low-Power Microprocessor Design,' in *Low Power Design Methodologies* (Eds. J.M. Rabaey and M. Pedram), pp.255-288, Kluwer Academic, Norwell, MA, 1996.

[24] C. Svensson and D. Liu, 'Low Power Circuit Techniques,' in *Low Power Design Methodologies* (Eds. J.M. Rabaey and M. Pedram), pp.37-64, Kluwer Academic, Norwell, MA, 1996.

[25] V. Zyuban, P. Kogge, 'Optimization of High-Performance Super-Scalar Architectures for Energy-Delay Product,' in *Proc. Intl. Symposium on Low Power Electronics and Design*, July 2000, Portofino, Italy.

[26] D. Marculescu, 'Profile-Driven Code Execution for Low Power Dissipation,' in *Proc. Intl. Symposium on Low Power Electronics and Design*, July 2000, Portofino, Italy.

[27] S. Palacharla, N.P. Jouppi, and J.E. Smith, 'Quantifying the Complexity of Superscalar Processors,' CS-TR-1996-1328, Univ. of Wisconsin, Nov. 1996.

[28] K.I. Farkas, N.P. Jouppi, and P. Chow, 'Register File Design Considerations in Dynamically Scheduled Processors,' WRL Research Report 95/10, Digital Equipment Corp., Nov. 1995.

[29] S.J.E. Wilton and N.P. Jouppi, 'An Enhanced Access and Cycle Time Model for On-Chip Caches,' WRL Research Report 93/5, Digital Equipment Corp., July 1994.

[30] D.H. Albonesi, 'Dynamic IPC/Clock Rate Optimization,' in *Proc. Intl. Symposium on Computer Architecture*, June 1998.