LOW-POWER PROCESSOR DESIGN

Ricardo E. Gonzalez

Technical Report No. CSL-TR-97-726

June 1997

This research has been supported by ARPA contract FBI-92-194.

LOW-POWER PROCESSOR DESIGN

Ricardo E. Gonzalez

Technical Report: CSL-TR-97-726

June 1997

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
William Gates Computer Science Building, A-408
Stanford, Ca 94305-9040
<pubs@shasta.stanford.edu>

Abstract

Power has become an important aspect in the design of general purpose processors. This thesis explores how design tradeoffs affect the power and performance of the processor. Scaling the technology is an attractive way to improve the energy efficiency of the processor. In a scaled technology a processor would dissipate less power for the same performance or higher performance for the same power. Some micro-architectural changes, such as pipelining and caching, can significantly improve efficiency. Unfortunately many other architectural tradeoffs leave efficiency unchanged. This is because a large fraction of the energy is dissipated in essential functions and is unaffected by the internal organization of the processor.

Another attractive technique for reducing power dissipation is scaling the supply and threshold voltages. Unfortunately this makes the processor more sensitive to variations in process and operating conditions. Design margins must increase to guarantee operation, which reduces the efficiency of the processor. One way to shrink these design margins is to use feedback control to regulate the supply and threshold voltages thus reducing the design margins. Adaptive techniques can also be used to dynamically trade excess performance for lower power. This results in lower average power and therefore longer battery life. Improvements are limited, however, by the energy dissipation of the rest of the system.

Key Words and Phrases: processor design, processor architecture, low-power CMOS circuits, supply and threshold scaling.

Acknowledgments

I would never have completed this document without the support, encouragement and guidance of many people. I cannot possibly list everyone that helped make my stay at Stanford a fulfilling experience; I will mention just a few.

First and foremost I would like to thank Mark Horowitz, my principal advisor, for his support, guidance, and encouragement. Working with him for 7 years was a privilege. His ability to understand my ideas before I did and his continuous pursuit of knowledge were always a source of inspiration.

I would also like to thank Kunle Olukotun and Simon Wong for their support an guidance as members of my reading committee. I am also thankful to Bruce Wooley and Arogyaswami Paulraj for participating in my oral examination committee. Noe Lozano did much to convince me to pursue a Ph.D. and arranged the financial support for me to get started.

My family also gave their unwavering support to this enterprise. My parents never stopped asking how my research was progressing. And, more important, never accepted "It's going OK" for an answer. This dissertation is dedicated to them.

My friends and colleagues made my life at Stanford very enjoyable. This document now before you is due to them. They encouraged me, helped me spend time away from my workstation, and were an unending source of interesting conversations.

Finally, I would like to thank all the members of the Stanford Cycling Club especially our coach, Art Walker, for making the past few years of my life so painfully memorable.

This research was supported in part by the Advanced Research Projects Agency under contract J-FBI-92-194. by the School of Engineering at Stanford, and by the Intel Foundation.

To my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past five years there has been an explosive growth in the demand for portable computation and communication devices, from portable telephones to sophisticated portable multimedia terminals [1]. This interest in portable devices has fueled the development of low-power signal processors and algorithms, as well as the development of low-power general purpose processors. In the digital signal processing area, the results of this attention to power are quite remarkable. Designers have been able to reduce the energy requirements of particular functions, such as video compression, by several orders of magnitude [2], [3]. This reduction has come as a result of focusing on the power dissipation at all levels of the design process, from algorithm design to the detailed implementation. In the general purpose processor area, however, there has been little work done to understand how to design energy efficient processors. This thesis is a start at bridging this gap and explores power and performance tradeoffs in the design and implementation of energy-efficient processors.

Performance of processors has been growing at an exponential rate, doubling every 18 to 24 months, as is shown in Figure 1.1. The bad news is that the power dissipated by these processors has also been growing exponentially, as is shown in Figure 1.2. Although the rate of growth of power is perhaps not quite as fast as the performance curve, it still has led to processors which dissipated more than 50W [4]. Such high power levels make cooling these processors difficult and expensive. If this trend continues processors will soon dissipate hundreds of watts, which is unacceptable in most systems. Thus there is great interest in understanding how to continue increasing performance without also increasing power dissipation.

For portable applications the problem is even more severe since battery life depends on the power dissipation. Lithium-ion batteries have an energy density of approximately 100Wh/Kg, the highest available today [5]. To operate a 50W processor for 4 hours requires a 2Kg battery, hardly a portable device. To address this problem processors manufacturers have introduced a variety of low-power chips. The problem with these processors is that they tend to have poor performance, as is shown in Figure 1.3. This

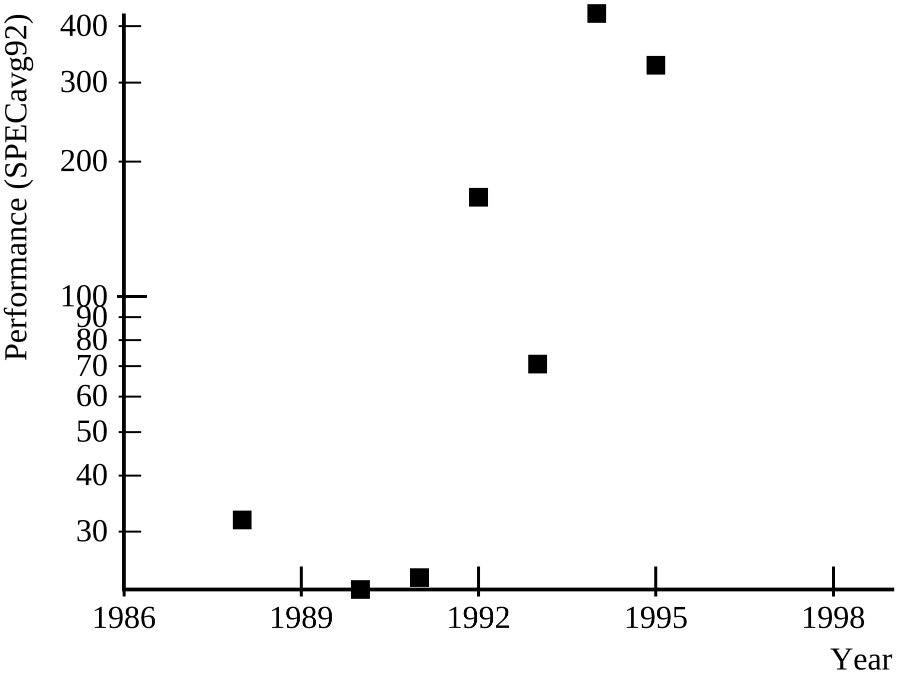


Figure 1.1: Evolution of processor performance.

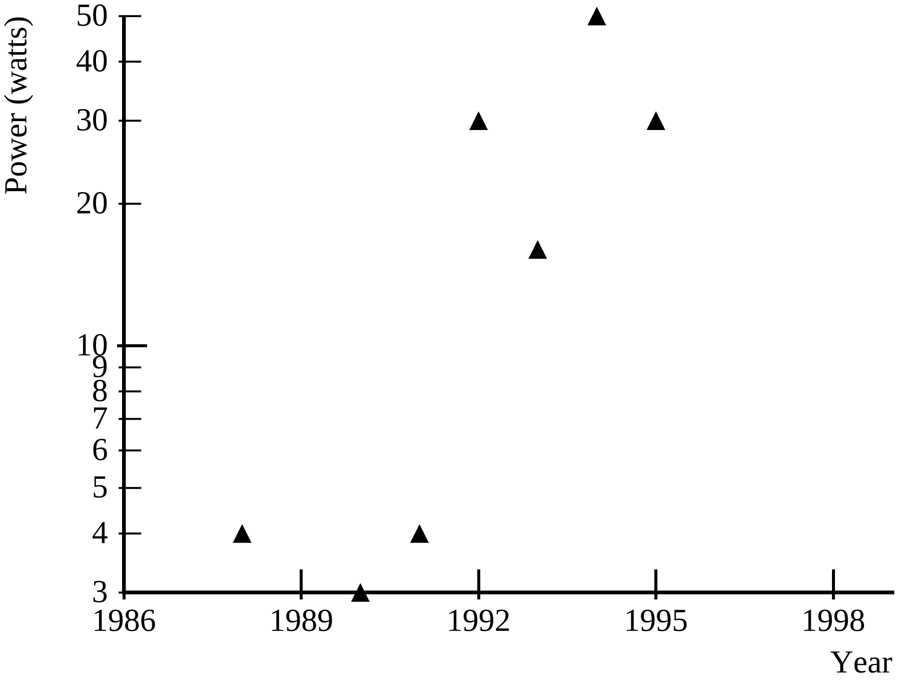


Figure 1.2: Evolution of processor power.

figure plots on the Y-axis performance, measured as the average of SPECint92 and SPECfp92 [6], and on the X-axis energy, measured as watt/SPEC.

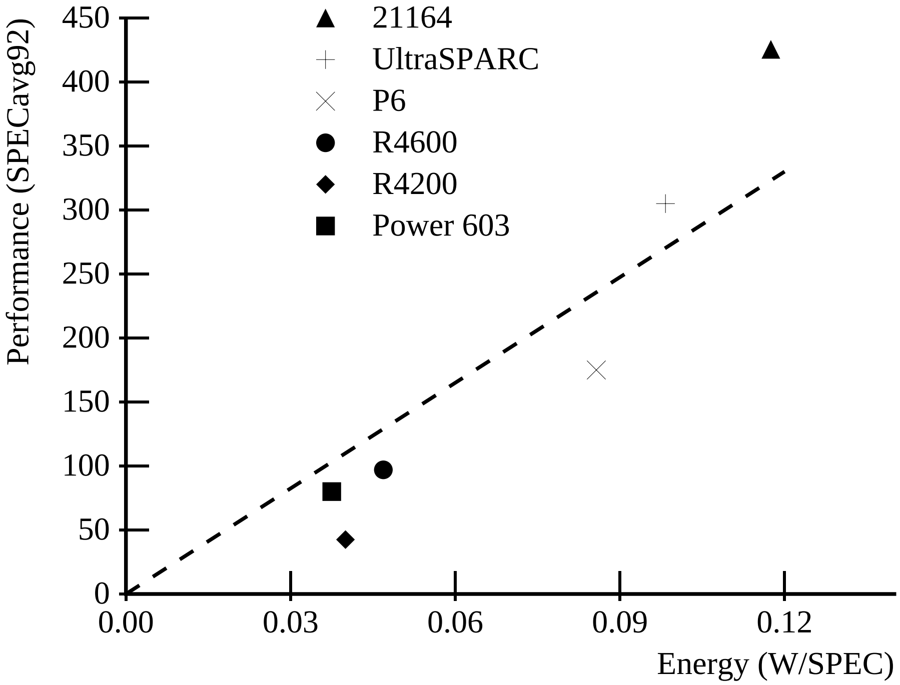


Figure 1.3: Performance and energy of processors.

In order to compare processor designs that have different performance and power one needs a measure of “goodness”. If two processors have the same performance or the same power, then it is trivial to choose which is better—users prefer higher performance for the same power level or the lower power one if they have the same performance. But processor designs rarely have the same performance. In particular when determining whether to add a particular feature designers need to know whether it will make the processor more desirable. Chapter 2 introduces the energy-delay product, or EDP for short, as a measure of “goodness” for low-power designs. This chapter also describes the most common low-power techniques and explores how they affect the energy-delay product of CMOS circuits.

Chapter 2 will show that exploiting parallelism is one important technique enabling the reduction of the energy-delay of a circuit. Thus Chapter 3 explores how micro-architectural choices, which change the amount of parallelism the processor exploits, affect the efficiency of the processor. Since both the performance and energy dissipation of modern processors depend heavily on the design of the memory hierarchy, one must look not only at the processor itself, but also explore how the design of the memory hierarchy affects the overall efficiency of the system. Using three idealized processor models Chapter 3 shows micro-architectural changes do not significantly improve the efficiency of the processor system. The processor's efficiency is set by a few circuits elements: memories and clocks.

Since memories and clocking circuits are critical components of every digital system, much work already has been done to reduce the energy requirements. A different approach to reduce the energy dissipation of clocks and memories is to change the technology by scaling the supply voltage and the threshold voltage of transistors. Chapter 4 explores tradeoffs in scaling the supply and threshold voltage of CMOS circuits. A simple mathematical model of the EDP predicts large gains in efficiency by scaling the supply and threshold voltages, especially if transistors are velocity saturated. If there is uncertainty in the supply and threshold, however, the gains in EDP are much smaller. Furthermore, to achieve these modest gains it may be necessary to give up large (3X) factors of performance. These tradeoffs are discussed in more detail in Chapter 4 along with a promising method to reduce the effect of variations by using adaptive techniques to control the supply and threshold.

Finally, Chapter 5 summarizes the contributions of this dissertation and proposes areas for future research.

Chapter 2

Energy-Delay Product

During the past few years many different techniques for reducing the power dissipation of CMOS circuits have been proposed, but relatively little work has been done to compare the benefits and costs of these different techniques. This chapter provides a review of these techniques and compares the effect they have on power and speed of the circuit. The rest of this thesis will investigate how the most promising of these techniques affect general purpose processors in particular.

This chapter begins by giving a brief description of the sources of energy dissipation in CMOS circuits, since it is important to understand this topic before addressing the question of how to reduce the energy dissipation. It then describes different metrics that can be used to compare designs. An attractive possibility is to represent every design as a point in the performance-energy plane. For CMOS, since energy and performance are highly correlated, it is often enough to compare the energy-delay product, or EDP for short.

2.1 Energy Dissipation in CMOS Circuits

There are three sources of energy dissipation in CMOS circuits; dynamic energy, static energy, and short-circuit energy. A simple CMOS gate consists of two transistors, represented as a resistor and a switch, connected to a fixed output load capacitance and a constant voltage source, as shown in Figure 2.1. Dynamic energy is due to the charging and discharging of the load capacitance. If the output node is originally at ground and assuming that it swings full rail, then an amount of energy equal to CV^2 is drawn from the voltage source on a low to high transition. Of this amount, $1/2CV^2$ is dissipated in the p-transistor to charge the load capacitance and $1/2CV^2$ is stored in the capacitor itself. The stored energy is dissipated in the n-transistor to discharge the load. Thus $1/2CV^2$ is dissipated on each transition. The circuit only dissipates dynamic energy when it is active or switching. If the output node remains at a fixed voltage level, then no energy is dissipated. Most nodes in CMOS circuits transition only infrequently; therefore the energy per cycle is usually written as,

$$E = \frac{nCV^2}{2} \tag{2.1}$$

where n is the number of transitions during the period of interest. If the circuit is synchronous and clocked at a frequency f then the average power can be written as,

$$P = aCV^2 f \tag{2.2}$$

where a is the probability of a transition at the output node divided by 2. If a node transitions every cycle then a=0.5.

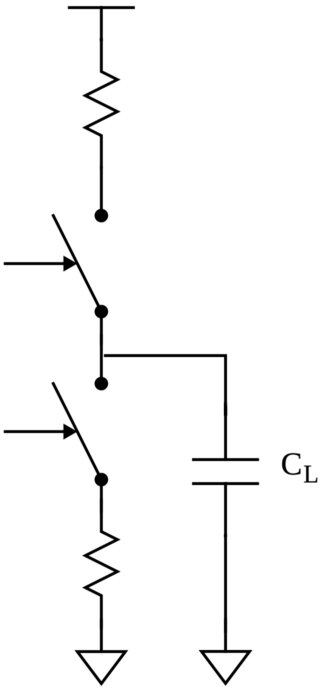


Figure 2.1: CMOS inverter.

Static energy is due to resistive paths between the supply and ground. The two main sources of static energy are analog or analog-like circuits which require constant current sources, and leakage current. Although there is some leakage current through the reverse biased diode between the source/drain and the bulk, the more important component is leakage through the channel when the transistor is nominally off [7]. The leakage current density (current per µm of gate width) is proportional to $e^{-V_{th}/\gamma V_t}$, where V_{th} is the threshold voltage of the transistor, V_t is the thermal voltage, and γ is a constant slightly larger than 1. Static energy is important because it can limit the energy dissipation when the circuit is idle or in standby mode and there is no dynamic energy dissipation.

Short-circuit energy is due to both transistors being on simultaneously while the gate switches. Troutman [8] and Chatterjee [9] provide good descriptions of short-circuit current in CMOS circuits. As these papers show this component is usually small and therefore will be ignored for the remainder of this thesis.

For most CMOS circuits in today's technologies dynamic energy dissipation dominates. For example, in a 0.6μm technology with V_{th}=0.9V leakage current is 4-5 orders of magnitude smaller than dynamic current (for one inverter in a 31 element ring oscillator). That is, only when the circuit is idle for 99.99% of the time does leakage current become an important consideration. As the amount of on-chip transistor width increases or the threshold voltage of the transistors decreases, leakage current becomes more significant. Chapter 4 explores in more detail how the energy efficiency of CMOS circuits changes as the threshold voltage changes.

In order to reduce the energy dissipation it is necessary to reduce one or more of α, C, or V. The next section describes and compares how some often proposed low-power design techniques attempt to reduce these quantities and how they effect the power and performance of CMOS circuits.

2.2 Low-Power Metrics

When optimizing a design for low power it is necessary to have a metric that can be used to compare different alternatives. The most obvious choice is power, measured in watts. Power is the rate of energy use, or $P=dE/dT$. A more useful definition, however, is average power, or the energy spent to perform a particular operation divided by the time taken to perform the operation $P_{avg}=E_{op}/T_{op}$. How to define the operation of interest is arbitrary and depends on what is being compared. In the case of a processor, it could be the energy to run a benchmark to completion, or the energy to execute an instruction—as long as all processors compared execute the same instructions.

Power is important for two reasons. The first is that it determines what kind of package can be used for the chip. For example, a small plastic package, the cheapest form of packaging, can only dissipate a few watts. A processor which dissipates more than that will have to be sold in a more expensive package. The second reason power is important is because it limits how long the system battery will last. But power as a metric of "goodness" of low-power designs has some drawbacks. The most important drawback is that power is proportional to the operation rate, so one can reduce the power by slowing down the system. In CMOS circuits this is very easy to do, one simply reduces the clock frequency.

Regardless of what definition of an operation one uses, the basic problem with power remains, that power decreases simply by extending the time required to complete an operation. Power, therefore, is only a good metric to compare processors that have similar performance levels. If two processors can perform computation at the same rate, then clearly whichever dissipates less power is more desirable. If the processors run at different rates the slower processor will almost always be lower power.

An alternative metric is the energy per operation, measured in jules/op, or its inverse, measured in SPEC/watt or MIPS/watt. This metric does not depend on the time taken to perform the operation, since running the processor at half the frequency means you need to accumulate the power for twice as long. The problem with this metric is that, from Equation (2.1) the energy per operation can be made smaller by lowering the supply voltage. However, the supply voltage also affects the speed or performance of the basic CMOS gates, with lower supplies increasing the delay per operation. Thus low energy solutions might (and often do) run very slowly.

Another alternative is to use both metrics, energy and speed. Rather than representing designs by a single number they are represented as a point in the performance-energy plane, as in Figure 1.3 and Figure 2.2. Given some requirements, such as minimum performance or maximum energy, one can determine which is the best solution available. The problem with this representation is how to compare designs which have different performance or energy levels. Without additional requirements, such as area or cost, there is no way to decide which solution is better.

From an optimization standpoint one possible metric is the product of energy and delay, measured in jules-sec, or its inverse, measured in SPEC^2/watt. Optimizing the energy-delay product will prevent the designer from trading off a large amount of performance for a small savings in energy, or vice versa. As will be described later the energy-delay is also an attractive metric for other reasons. In Figure 2.2 the EDP corresponds to the inverse of the slope of a line that connects a design point to the origin. Thus finding a solution with a low EDP corresponds to finding a solution which lies on a steeper line.

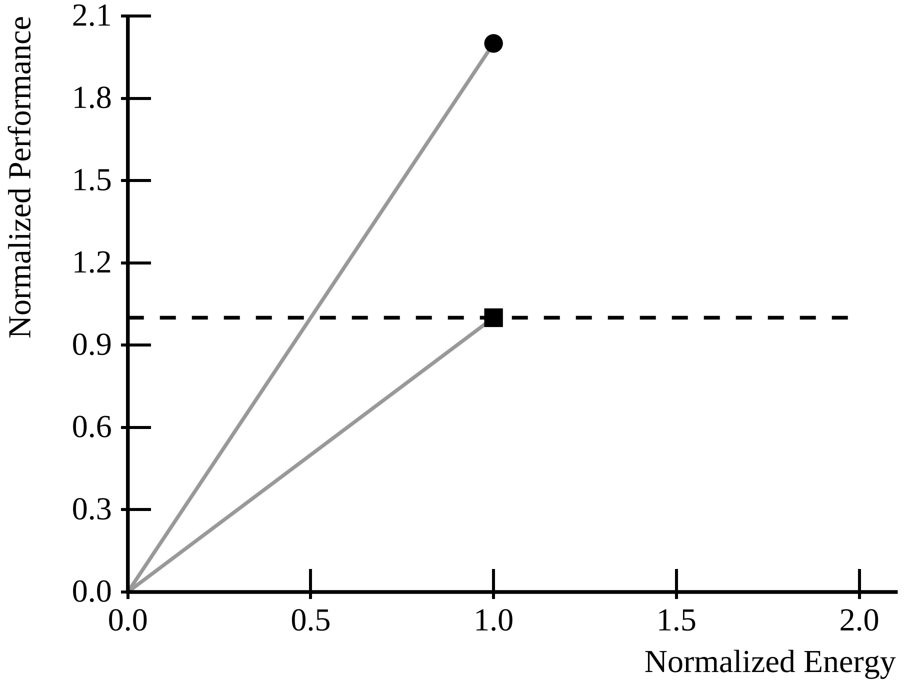


Figure 2.2: Performance-energy plane.

2.3 Low-Power Design Techniques

From Equation (2.1) one simple way to reduce the energy per operation is to lower the power-supply voltage. However, since both capacitance and threshold voltage are constant, the speed of the basic gates will also decrease with this voltage scaling. The delay of a CMOS gate can be modeled as the time required to discharge the output capacitance by the transistor current, $T_g = CV/I$. Using the current model presented by [10] this gives,

$$T_g = K\frac{V}{(V - V_{th})^{\alpha}} \tag{2.3}$$

where α is the velocity saturation coefficient and K is a technology specific constant. When transistors are not velocity saturated α=2.0 and the equation reduces to the quadratic model for transistor current. As transistors become more velocity saturated α decreases towards one. For typical 0.25μm technologies α=1.3-1.5.

Figure 2.3 plots the normalized speed of operation versus the energy per operation of a CMOS gate as the supply voltage is scaled. The speed and energy were found by simulating an inverter in a 0.6μm technology using HSPICE. The threshold voltage is held constant in this example. At large voltages reducing the supply reduces the energy for a modest change in delay. This causes the curve to bend over. At voltages near the device threshold, small supply changes cause a large change in delay for a modest change in energy. But from V=1.5V_{th} to V=6V_{th} changes in energy and delay cancel each other and the curve approaches a straight line, which corresponds to a constant energy-delay product. Over this region the EDP remains within a factor of 2. In this case scaling the supply voltage reduces the power dissipation but at the expense of the speed of the gates. Looking at Equation (2.3), we see that one way to gain back the performance lost is to scale the threshold voltage. But this increases the leakage current. Chapter 4 explores the tradeoff between power and performance from scaling the supply and threshold voltages. It will be shown, however, that when leakage power is a very small fraction of the total power, as is the case in most technologies today, scaling the supply voltage does not significantly affect the EDP.

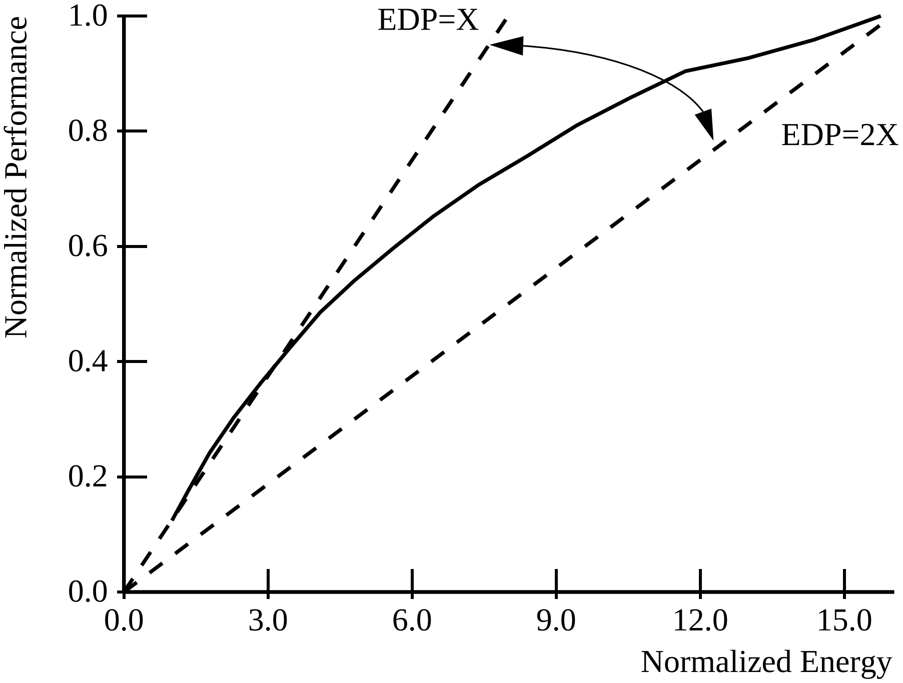


Figure 2.3: Variation in performance and energy with supply voltage.

Another technique to reduce the energy per operation is to reduce the size of all transistors in the gate. This reduces the capacitance that needs to be switched when one of the input switches. Unfortunately it also decreases the current drive of the gate, making it slower. This can be partly compensated by making the next gate smaller. At some point, however, the load of the gate will no longer be dominated by the input capacitance of the following gates, but rather by the capacitance of the interconnect between gates.

Figure 2.4 graphs the normalized energy per operation versus the speed of operation of a CMOS gate as the percentage of loading that is due to gate capacitance varies from 20% to 80%. The diffusion capacitance also depends on transistor width and therefore the percent of loading that depends linearly on the transistor width is larger than shown in the figure. The dotted line indicates the point at which 80% of loading is proportional to transistor width. The load will be mostly wire capacitance for small transistor, and will be mostly gate capacitance for large devices. Continuing to increase the transistor sizes gives very small gains in performance for a large energy cost. This causes the curve to be almost flat. But for the points shown in the figure the difference in EDP is a factor of 2.5.

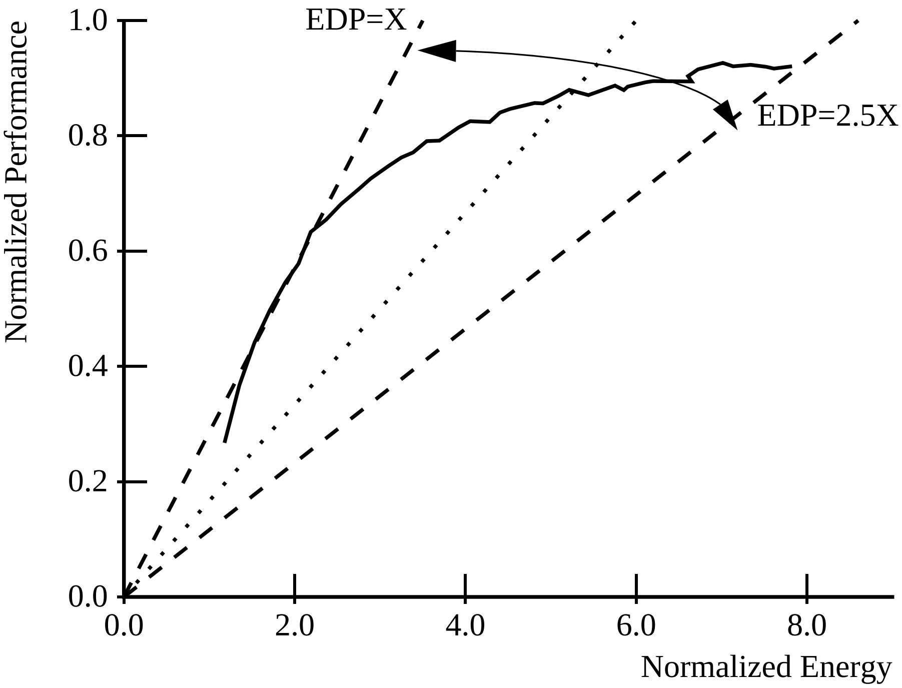


Figure 2.4: Variation in performance and energy with transistor sizing.

Clearly, real circuits are more complex. The gate and wire capacitance is different for every gate, nodes transition at different frequencies, and not all gates are on the critical path. While this problem is difficult to solve precisely, the basic tradeoff remains the same.

Sizing the transistors allows the designer to tradeoff speed for power. At either extreme (very large or very small transistors) the tradeoffs become poor.

Often one wants to optimize two, or perhaps more, variables simultaneously. For example one may want to find the optimal supply voltage and transistor size. One way to visualize the data is to plot contours of energy, delay, and perhaps energy-delay, on the supply voltage *vs* transistor size plane. Figure 2.5 is an example of such a plot, and gives contours of inverse relative EDP versus transistor size and supply voltage. The Y-axis shows the percent of the total load that is due to gate capacitance, and the X-axis shows the supply voltage. For the range of supply voltage and gate loading shown in this figure there is a local minimum in the EDP at V=1.7V and 40% gate loading. The relative EDP is the EDP normalized to the minimum value. The figure plots contours of the inverse of this metric. Thus the value at the minima is 1 and decreases as one moves away from the minima. The contour labeled 0.5 has twice the minimum energy-delay. The advantage of representing the data this way is that it is much easier to understand how the variables of interest (energy, delay, energy-delay) change with the optimization variables (supply voltage, transistor sizing). If the data were plotted in the energy *vs.* performance plane then understanding whether decreasing the supply voltage is a win is harder. In Chapter 4 this technique is used to visualize how energy and delay change over a broad range of supply and threshold voltages.

The reason the energy-delay product can be used to represent the performance-energy plane is that these low-power techniques allow the designer to trade excess performance for lower power. In Figure 2-2, for example, using voltage scaling the performance of the design marked with a circle can be reduced until it just meets the performance target. At that point it dissipates less energy than the design marked with a box so it is clearly a better choice. Moving the design to a steeper line, that is reducing the energy-delay, will result in a lower energy solution for the same performance. Using the EDP relies on the assumption that it is possible to almost linearly trade performance and energy. If this were not true, then one would again have to resort to the performance-energy plane. Fortunately this condition is true for most circuits today. For the reminder of this thesis, except for Chapter 4, the EDP is often used to compare different design alternatives. Scaling the threshold voltage can make this condition no longer true. In that case the performance and energy are used.

Adiabatic or charge-recovery circuits are another method that allow a designer to trade energy for performance [11], [12]. These circuits resonate the load capacitance with an

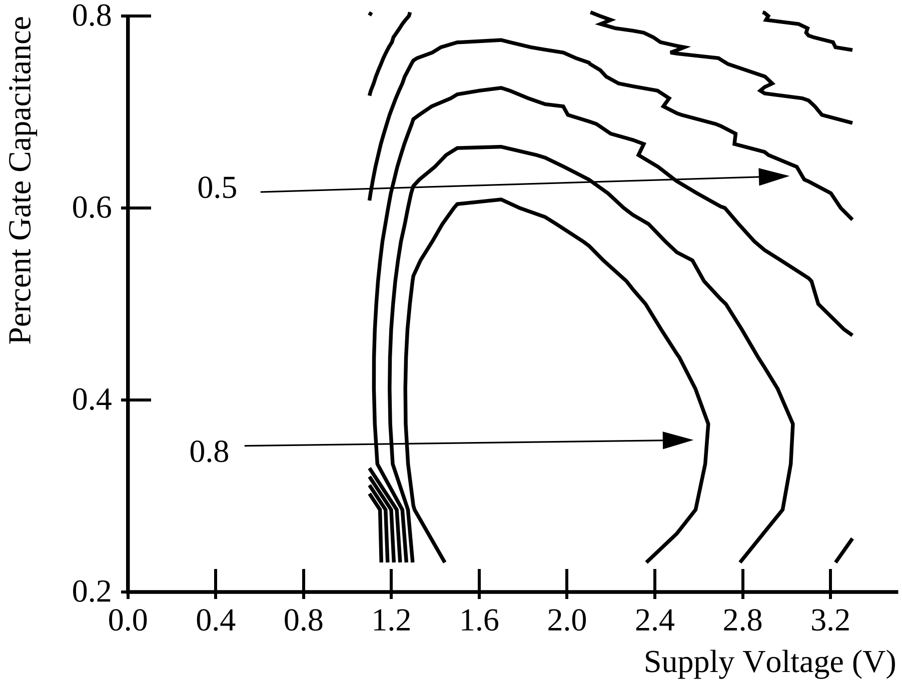


Figure 2.5: EDP contours versus transistor size and supply voltage.

inductor, which recovers some of the energy needed to charge the capacitor. The energy loss in switching the load can be reduced to CV^2T/τ, where T is the intrinsic delay of the gate, and τ is the delay set by the LC circuit. By changing the time constant of the LC circuit it is possible to linearly trade speed for switching energy leaving the EDP constant. The problem with these circuits is that for a given performance level they have larger energy requirements than normal CMOS gates [13]. Thus adiabatic circuits become attractive only when one needs to operate at delays beyond the range viable by voltage scaling and transistor sizing standard CMOS.

To increase the energy efficiency of the circuits it is necessary to move to a steeper line on the performance-energy plane. As was described earlier voltage scaling and transistor sizing do not dramatically change the EDP. One of the best techniques to improve the energy-delay product is to improve the technology. Under ideal scaling [14] if the minimum transistor length shrinks by a factor λ, then the energy per operation will shrink by a factor λ^3 and the delay will shrink by λ. Thus the EDP will decrease by λ^4. Therefore a 0.7 shrink of a chip will result in 1/4 the energy-delay. At the same performance the processor will dissipate 1/4 the power. Unfortunately most technologies do not scale ideally [15]. Often the threshold voltage does not scale linearly with λ. Furthermore, the performance of the processor, as will be shown in Chapter 3, depends heavily on the performance of the external memory system, which also does not scale linearly with the technology. Even if the threshold voltage does not scale, the reduced capacitance improves both the energy and the delay, so their product scales at least as λ^2.

Another way to improve the energy-delay product is to avoid wasting energy—avoid causing unneeded node transitions. One common approach to solve this problem is to make sure that idle blocks do not use any power. In CMOS circuits this can be accomplished by disabling the clock of that block therefore preventing any more transitions in the block. This technique is called selective activation because only blocks which produce a useful result are activated. The key to selective activation is to control objects that dissipate a significant amount of power. Chapter 3 explores in more detail what power savings are possible in a processor by using selective activation, and which blocks are likely candidates for selective activation.

Processors designers have used selective activation at many levels in order to reduce the power dissipation. Banked caches—only part of the memory array is activated on every access [16]—are common in low-power processors, such as the MIPS **R4200** [17] and **R4600** [18], the **PowerPC 603e** [19], and the **StrongArm-110** [20]. Some low-power processors, such as the **PowerPC 603e**, also contain control logic which determines, on a cycle by cycle basis, which functional units need to be activated.

Another common technique to reduce wasted power is to slow down the clock frequency when the processor is idle. All of the low-power processors listed before have a reduced power mode where the clocking frequency is reduced to a fixed multiple of the system clock. In some cases the processor clock can be stopped completely. Reducing the clock frequency when idle will, on average, extend battery life. But it will not make the processor more efficient *per se*. Reducing transitions while the processor is active can give some power savings but, as will be shown in Chapter 3, these savings are limited to less than a factor of two or three. In order to obtain larger savings it is necessary to look at the problem from the system level.

Re-thinking an algorithm can give orders of magnitude improvement in energy-delay. Accessing memories, for example, is relatively expensive in energy terms, while performing computation is relatively cheap. For many applications it is possible to reduce the energy dissipation by performing more computation in order to reduce the number of memory accesses [21]. In many cases rethinking the algorithm gives an improvement in both energy and delay, which means the energy-delay decreases as the square of the gains.

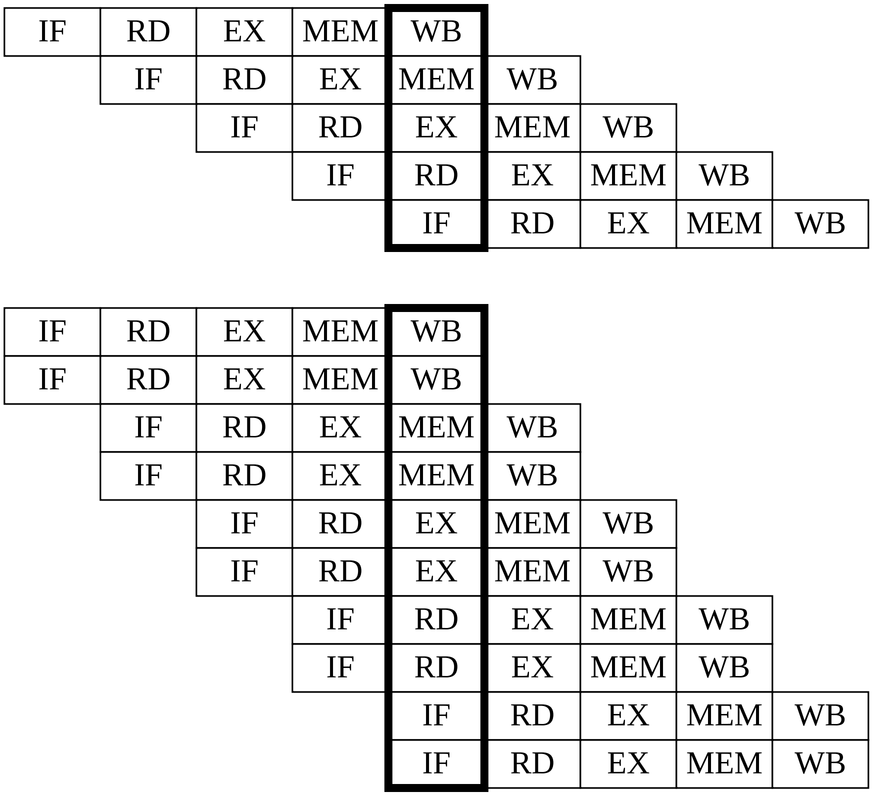
If an algorithm requires two operations in series—each of which requires one unit of time and one unit of energy—then changing the algorithm so that it only requires one operation will decrease the EDP by a factor of four, a factor of two from the energy and a factor of two from the delay. Tsern and Gordon [22], [3] estimate that redefining the DSP compression algorithm provided at least an order of magnitude reduction in the energy-delay product.

Unfortunately designers of general purpose processors cannot re-think the algorithm. That is something only the system programmer can do. Fortunately, however, redefining the algorithm is not the only way to attack the problem at a system level.

One can improve the energy-delay product by reducing either the energy or the delay per operation. One way to reduce the effective delay without affecting the energy—to a first approximation—is to exploit parallelism. For example, if one functional unit can execute one operation every cycle, then adding a second functional unit in parallel will double the performance, to two operations per cycle, but leave the energy per operation unchanged. This is because it now takes roughly twice the energy to compute two results. The EDP is cut by half. Ideally the energy-delay goes down with *N*, the degree of parallelism. In reality, however, there is always some energy and delay overhead to distribute the operands and to collect the results. Furthermore, the amount of parallelism may be limited—the second functional unit may be idle part of the time—so the actual improvements in both performance and EDP will be smaller than *N*. However, exploiting parallelism is a very powerful general technique to increase the energy efficiency of circuits.

The improvements in energy-delay is independent of how the parallelism is extracted (pipelining, parallel machine, etc.) although the overhead factors will be different. For DSP applications with a large amount of parallelism, the performance gains allow the resulting systems to run at very low supply voltage, use small transistor sizes, and still meet their performance targets [23].

Exploiting parallelism has been a common way to increase the performance of processors. In the late 80’s RISC processors popularized pipelining as a way of exploiting parallelism. Pipelining overlaps the execution of instructions, so that even if an instruction takes more than one cycle to execute, it appears as if the processor can execute one instruction per cycle. During the early 90’s super-scalar machines, which can execute more than one instruction per cycle, became popular. These machines exploit parallelism both by pipelining and by executing more than one instruction per cycle, as is shown in Figure 2.6 In the past few years the maximum issue size of processors has increased, to try to exploit more parallelism. Unfortunately the parallelism available in typical programs is very limited [24], [25], [26]. Researchers have found that for realistic machines the cycles per instruction (CPI) is greater than 0.5. This means on average less than two instructions start execution on every cycle. Trying to extract additional parallelism often results in decreased performance because the added hardware increases the cycle time. Therefore even if the amount of parallelism available is larger [27] the performance gains are limited.



IF	RD	EX	MEM	WB				
	IF	RD	EX	MEM	WB			
		IF	RD	EX	MEM	WB		
			IF	RD	EX	MEM	WB	
				IF	RD	EX	MEM	WB

IF	RD	EX	MEM	WB				
IF	RD	EX	MEM	WB				
	IF	RD	EX	MEM	WB			
	IF	RD	EX	MEM	WB			
		IF	RD	EX	MEM	WB		
		IF	RD	EX	MEM	WB		
			IF	RD	EX	MEM	WB	
			IF	RD	EX	MEM	WB	
				IF	RD	EX	MEM	WB
				IF	RD	EX	MEM	WB

Figure 2.6: Scalar and super-scalar processors pipeline diagrams.

One surprising fact is that parallelism does not seem to greatly affect the energy-delay of processors. Table 2.1 shows more information for the same processors as in Figure 1.3. The SPEC^2/W metric, which is inversely proportional to the energy-delay, does not seem to correlate with the maximum issue size of the machine, as previous paragraphs suggest. This result is rather surprising since designers of signal processing chips have used parallelism to dramatically improve the efficiency of their circuits. The next chapter explores the effectiveness of exploiting parallelism in improving the energy-delay product of processors, and explains why the results are modest.

	Dec 21164	UltraSPARC	P6	R4600	R4200	Power 603
SPECavg92	426.00	305.00	175.00	97.00	42.50	80.00
Power	50.00	30.00	15.00	5.500	1.80	3.00
SPEC^2/W	3629.50	3100.83	2041.70	1710.33	1003.47	2133.33
L_{min}	0.50	0.45	0.60	0.64	0.64	0.50
SPEC^2/Wλ^2	4480.90	3100.83	3629.69	3459.51	2029.73	2633.74
Max Issue	4	4	3(3)	1	1	2

Table 2.1: Current processors.

Chapter 3

Micro-architectural Tradeoffs

This chapter explores how the micro-architecture of a processor changes its energy efficiency. One common technique to speed-up processors is to overlap the execution of instructions, i.e. to exploit the parallelism available at the instruction level. This can be accomplished by simply pipelining the execution of instructions, or more aggressively by executing more than one instruction at once. The next section investigates the tradeoffs in energy-delay of exploiting instruction level parallelism (ILP) using idealized models of pipelined and super-scalar processors. The results show that pipelining gives a large boost in efficiency while super-scalar issue only gives a small improvement. Then Section 3.2 explores how far current implementations are from these inherent limits by looking at two processor implementations, a simple RISC machine and TORCH—a super-scalar processor designed at Stanford. In Section 3.3 the energy dissipation of these realistic machines is categorized into four components. The section shows that most of the energy dissipation is in clocking and storage circuits.

Simply increasing the rate at which the processor can execute instructions is not enough. Often the memory system cannot provide enough instructions and data for the processor to operate on. That is, the performance is limited by the memory system. Furthermore, the memory system dissipates a considerable amount of power, sometimes more than the processor itself. Section 3.4 explores how the design of the memory hierarchy affects the energy-delay product of the system as a whole.

Finally the last section of this chapter investigates how more sophisticated architectural features, such as non-blocking caches and out-of-order issue, will affect the energy-delay of processors.

3.1Lower Bound on Energy and Energy-Delay

Although there have been several papers that have presented dramatic reductions in the power dissipation of processors, there has been no work to investigate what is the lower bound on the energy and energy-delay product. Therefore one cannot determine if the reductions accomplished represent a large fraction of the potential improvements. This section investigates what these bounds are by looking at three idealized machines: an unpipelined processor, a simple pipelined RISC processor, and a super-scalar processor.

All processors fundamentally perform the same operations, they sequence through the steps shown in Figure 3.1. They fetch instructions from a cache, use that information to determine which operands to fetch from a register file, then either operate on this data, or use it to generate an address to fetch from the data cache, and finally store the result. These operations are sequenced using a global clock signal. High performance processors have sophisticated architectural features that improve the instruction fetch and data cache bandwidth and exploit more instruction level parallelism. But they must still perform these basic operations for each instruction. In a real processor, of course, there is much more overhead. For example, often many functional units are run in parallel and only the “correct” result is used. In addition considerable logic is needed to control the pipeline during stalls and exceptions. However, since this overhead depends on the implementation and in some sense is not essential to the functionality of the processor it will be neglected. Most of the operations outlined above require either reading or writing one of the on-chip memories—instruction and data caches, and the register file. Only one of the steps requires computation. So it will be aggressively assumed that the energy cost of performing computation is zero, and also that communication costs within the datapath is also zero. Only the energy needed to read and write memories, and, where required, the energy to clock essential storage elements, such as pipeline latches is considered. Furthermore these ideal machines only dissipate energy when it is absolutely necessary. Most processors perform some operations speculatively, in the hope that the result is useful. These ideal machines have perfect knowledge so there is never a need for speculative operations or unwanted transitions, such as glitches.

3.1.1 Simulation Methodology

To estimate the energy dissipation of these idealized machines a simple trace based method is used. In these machines all the energy is dissipated in memories or storage elements, and is independent of the data values. The energy required to read a memory is independent of the data value read. The number of memory accesses is computed from the instruction trace. The product of the energy per operation times the number of operation of each type gives the total energy dissipation. The clock energy is found by determining the capacitance of a single latch and multiplying by the minimum number of latches to maintain the processor state and ensure correct operation.

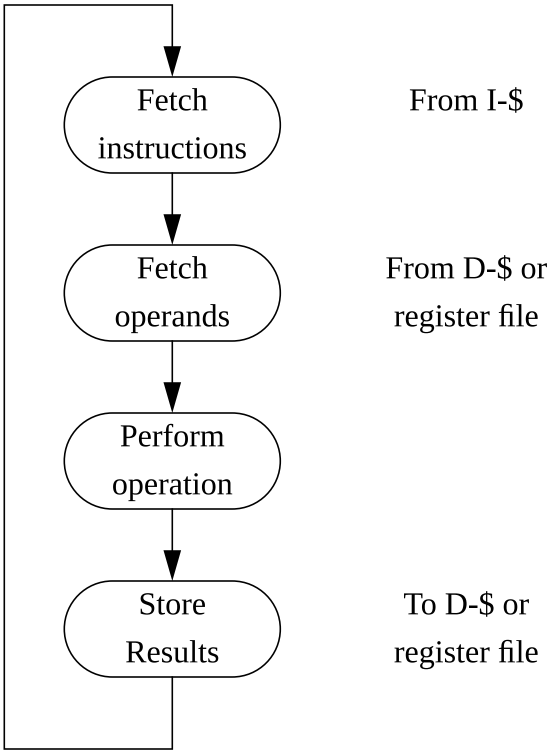


Figure 3.1: Basic processor operation.

The memory power model developed by Amrutur (described briefly in Appendix B) was used to estimate the power of the caches. The memory model was developed for a 0.8µm process, so this process is used throughout this chapter for the simulations. The power of the register file was estimated by using a modified version of Amrutur's models to account for the difference in size and the number of read/write ports needed. The power model of the register file takes into account the number of active ports so the power requirements depend on the number of operands being read and written.

The latency of the functional units only depends on the particular implementation and is independent of the processor micro-architecture. The latency of the functional units can be characterized by the number of equivalent fanout-of-four inverters which have approximately the same delay[1]. The cycle time of the processor, if it is pipelined, can be measured using the same units. For current processors the cycle time is in the order of 30 fanout-of-fours. Chapter 4 will explain in more detail why using the equivalent number of fanout-of-four inverter delays is a good metric. The latencies of the functional units, in inverter delays, are assumed to be as follows: branch 60 (2 cycles), ALU 90 (3 cycles), and load/store 120 (4 cycles).

[1] A fanout-of-four inverter drives four equally-sized inverters.

The energy dissipation depends also on the benchmark being run, since it determines how many times the on-chip memories must be accessed. Due to the limited simulation speed (only about 1 Hz when simulating the complete machines described in the next section) only two benchmarks from the SPEC benchmark suite were used: compress and jpeg. The first is typical of integer programs that have many unpredictable branches. The second is typical of multimedia programs which are becoming popular. Due also to the slow simulation speed the benchmarks were run with reduced inputs. The benchmarks were compiled using the TORCH optimizing C compiler or the MIPS C compiler, depending on the target machine.

3.1.2 Machine Models

The simple machine is the most basic compute element, similar to DLX [28]. The processor consists of a state machine that fetches an instruction, fetches the operands, performs the operation and stores the result. The processor is not pipelined so each instruction completes before the next one is fetched. Since this machine is not pipelined no energy is dissipated in clocking. Only the energy required to read and write the caches and register file is taken into consideration. This implementation should have the lowest average energy per instruction; however, since it does not take advantage of the available parallelism, it will have relatively poor performance and high energy-delay product.

To improve performance one can create a pipelined processor that is similar to the previous one, except that it uses pipelining to exploit instruction level parallelism. That is, more than one instruction is executed concurrently. Thus the change in energy-delay product of this machine compared to the ideal will be due entirely to pipelining. This machine uses the traditional MIPS or DLX [28] 5-stage pipeline shown in Figure 2.6. In a pipelined machine, because multiple instructions are being executed simultaneously, it is necessary to have some synchronization or storage elements, such as flip-flops or latches, in the datapath portion of the machine to ensure that data from different instructions do not corrupt each other. Therefore in addition to the energy required to read and write memories the energy required to clock the latches in the pipeline and the PC chain is also taken into account.

The ideal super-scalar processor is similar to the pipelined machine, except that it can execute a maximum of two-instructions per cycle. One can classify super-scalar processors into two groups, statically scheduled and dynamically scheduled. Statically scheduled processors rely on the compiler to determine an instruction execution sequence that guarantees program correctness. The compiler must ensure that before an instruction is executed all of its source operands are available. Dynamically scheduled processors rely on on-the-fly dependency analysis to determine which instructions can be executed concurrently. In these machines the hardware ensures that when an instruction is executed its source operands are available. For this idealized machine how the ILP is found is not important, since the energy dissipated in the control logic is not taken into account. What matters is how much parallelism is exploited. Therefore the experiments were done using an aggressive static scheduler from TORCH [29] which gives comparable performance to a dynamic scheduled machine. Since most integer programs do not have enough parallelism to fill both issue slots regularly, this machine supports some conditional execution of instructions as directed by the compiler. This machine is an idealization of the TORCH machine that is described in a later section. The difference in energy-delay product between this machine and the previous one will be due to the ability to exploit more parallelism by executing two instructions in parallel each cycle.

3.1.3 Comparison of Energy and Energy-Delay Product

Figure 3.2 shows the total energy required to complete a benchmark on the three machines versus their performance. All numbers in this section have been normalized to the corresponding value for the super-scalar processor. As expected the unpipelined machine has the lowest energy dissipation, and the super-scalar machine has the highest. The unpipelined machine, however, has relatively poor performance. One can instead build a pipeline processor, which dissipates slightly more energy but has much better performance. Thus it has a much lower energy-delay—about 2X lower than the unpipelined machine. Super-scalar issue only gives a small improvement in energy-delay product. The performance gain is smaller and the energy cost is higher. In this simple model, a processor with a wider parallel issue would be of limited utility for reducing the EDP. The basic problem is that the amount of instruction level parallelism (in the integer benchmarks used) is limited, so the effective parallelism is small. Yet increasing the peak issue rate increases the energy of every operation, even in this ideal machine because the energy of the memory fetches and clocks will increase. Furthermore, the speedup of the super-scalar processor is limited by Amdahl’s law. All three processors spend the same number of cycles waiting for data from memory because the memory hierarchy is the same for all three. As the performance of the processor increases the stall time becomes a larger fraction of the total execution time, so that the corresponding decrease in energy-delay product will be smaller. As will be shown in the following section, the overhead in a super-scalar machine is actually worse than that of a simple machine, and overwhelms the modest gain in energy-delay product gained from parallelism.

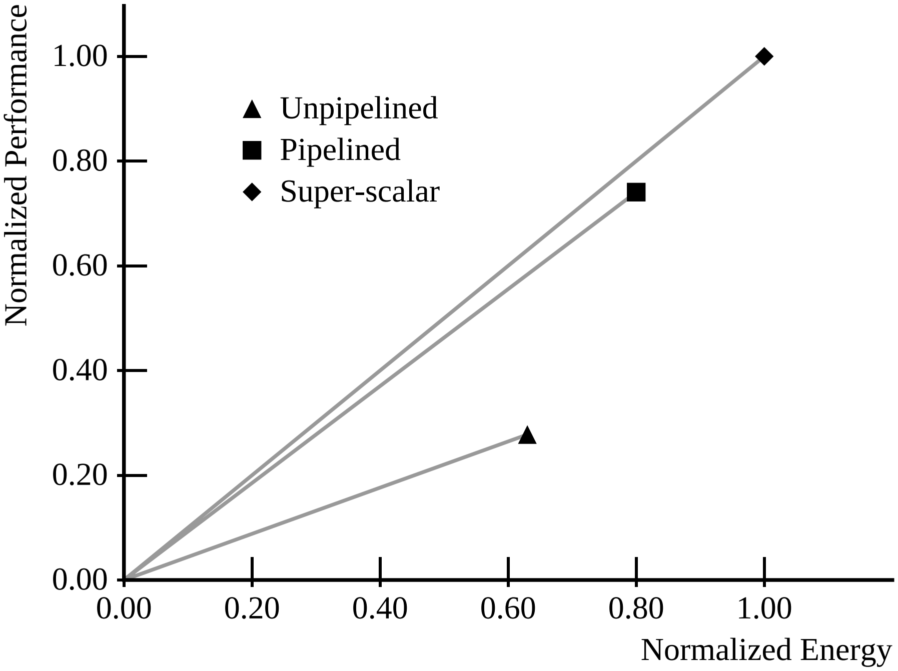


Figure 3.2: Normalized performance and energy of idealized machines.

3.2 Processor Energy-Delay Product

To see how real machines compare with the ideal machines described in the previous section, this section looks at two processors designed at Stanford, a simple RISC machine and a super-scalar processor called TORCH. For the real processors all sources of overhead were considered, including control, communication, and all architecturally defined state. This is unlike the idealized machines where attention was focused only on the essential operations. As has been shown before [17], [30], [19], [31], some simple optimizations can yield significant gain in energy dissipation. So these optimizations, which are described in Section 3.2.3, were performed on our processors as well.

The simulation framework for evaluating these machines is given next, and is followed by a brief description of the two machines we simulated. Having described the simulation setup the energy and delay results are presented and then are compared to the idealized machines presented earlier.

3.2.1 Simulation Methodology

CMOS gates dissipate energy only when their outputs transition. Thus the energy dissipation over time depends on the particular input sequence chosen. For a processor this implies that the energy to run a benchmark depends on the data used by the program. This makes simulating the real machines hard, since it is necessary to model the machines in great level of detail.

In order to estimate the total energy dissipation it is necessary to know how many times each node transitioned, and what the total capacitance of that node is. Fortunately a complete RTL (Verilog) description of both processors under investigation was available. The model includes the main datapaths, instruction and data caches, co-processor 0 (system or privileged state), TLB, and external interface; although it does not include a floating point unit. The model can stall, take exceptions and interrupts, or suffer cache misses. Specialized CAD tools were developed to estimate the total capacitance at each node in the model. The operation of the CAD tools is described in Appendix A. A commercial Verilog simulator is used to run a benchmark on the processor model. During the simulation a special tool accumulates the number of transitions on every node. The total energy dissipation is proportional to the sum over all nodes of the transition count times the capacitance.

The on-chip memories are modeled by assuming a certain energy dissipation every time the memory is read or written. For example, whenever the instruction cache access signal is enabled all of the access energy is added to the total. No toggle or capacitance information is kept for the internal nodes of the memories. The processors implemented contain a TLB which is also modeled using a modified version of Amrutur’s model. For the TLB it is important to model the power dissipation of the match lines. On an access they are all pre-charged high and all but one transition low. The power of the TLB was 150mW at 100MHz.

To ensure accurate comparisons with the idealized machines the same benchmarks were run on these processors. See Section 3.1.1.

3.2.2 Machine Models

The simple RISC processor is similar to the original MIPS R3000 described by Kane [32], except that it includes on-chip caches. The processor has the same five-stage pipelined that was shown in Figure 2.6. This processor is similar to the ideal pipelined machine except that it accounts for all the energy required to complete the instruction and it includes all the overhead associated with the architecture, such as the TLB and coprocessor 0. The difference in energy-delay product between this processor and the pipelined ideal machine represents a limit on possible improvements in efficiency by focusing only on the logic required to execute an instruction.

TORCH [29] is a statically scheduled two-way super-scalar processor, which uses the same five stage pipeline that was shown in Figure 2.6. TORCH supports conditional execution of instruction directed by the compiler. The processor includes shadow state that can buffer the results of these conditional instructions until they are committed. To improve the code density a special NOP bit is used to indicate that an instruction should be held for a cycle before it is executed. This is important in programs that do not have enough parallelism to keep both datapaths busy because it improves the instruction cache performance and reduces the number of cache accesses needed to complete a program. Instructions are encoded in a 40-bit word which are packed in main memory using a special format that improves the instruction fetch efficiency. Instructions are unpacked as they are written in the first level instruction cache.

TORCH is a good example of a low-power super-scalar processor since it should have a smaller overhead than a dynamically scheduled machine. And because TORCH has similar performance as a dynamically scheduled machine [29] TORCH should have lower energy-delay. Section 3.5 examines in more detail the energy cost of out-of-order issue.

3.2.3 Energy Optimizations

The additional energy associated with a particular architectural feature can be divided into overhead and waste. Overhead is that part of the energy that cannot be eliminated with careful design. For example, adding additional functional units will increase the average wire length thus increasing the overhead of the architecture. Other sources of energy can simply be designed away. For example, in a pipelined design clock gating can be used to eliminate unwanted transition of the clock during interlock cycles. A good implementation will reduce waste as much as possible. However, the designer must carefully weigh the gains in power dissipation versus the cost in complexity or cycle time. If a particular optimization causes a large increase in cycle time then dissipating slightly more energy but running at a higher frequency may be better.

As this section will show eliminating all of the waste is difficult because much of it is scattered among a large number of functional units. The key to effectively reducing energy waste is to find points that have large leverage such that a small design change will produce large energy savings. Thus nodes which have a large amount of capacitance, such as clocks and buses, are prime target for selective activation. Furthermore large structures, such as memories are also good candidates since they dissipate a large amount of energy and can be easily controlled. Most memories have a read enable and write enable signals. Following are a few techniques that were used to reduce waste in the implementations.

Clock gating can be used to eliminate transitions that have no effect. In these implementations latches in the datapath are qualified when the instruction does not produce a result or when the processor is stalled waiting for data from the outside world. In TORCH densely coded NOP's improve the code density and reduce the number of instruction cache accesses. However, they introduce extra instructions into the pipeline which can cause spurious transitions. Using clock gating can eliminate these spurious transitions. Also, all latches that are not in the main execution datapath are qualified. These latches tend to be enabled only infrequently, and clocking them every cycle is inefficient. For example, latches in the address translation datapath only need to be enabled on a load or store instruction, or during an instruction cache miss.

Most of the latches in the control sections are not qualified since this can introduce clock skew. Control sections are automatically synthesized and therefore it is not possible to have very fine control over the implementation of clock gating. However, the power dissipated in the control sections is only a small fraction of the total power, and the clock power is only a fraction of that. Some control signals are qualified to prevent spurious transitions in the datapath buses. If an instruction has been dynamically NOPed then preventing the datapath latches from opening is not sufficient. If the control signals to the datapath change, this can cause spurious transitions on some of the datapath buses, which can cause significant energy consumption. Since buses have large amounts of capacitance spurious transitions can dissipate significant amounts of power.

The use of clock gating saved approximately one third (33%) of the clock power or close to 15% of the total power. Most of the latches in datapath blocks have an enable signal. To further reduce the energy it would be necessary to either make the enabling signal more restrictive or qualify latches in the control sections. In either case further improvements would require a larger effort and provide smaller returns.

Selective activation of the large macro-cells was used to reduce power dissipation. Obviously the most important step was to eliminate accesses to the caches and the register file when the machine is stalled. Also important was to eliminate accesses to the instruction cache when an instruction is dynamically NOPed. In this case the instruction has already been read on the previous cycle so there is no need to re-read the cache. By doing so approximately 8% of the power dissipation was saved.

Speculative operations are commonly used to improve the performance of a microprocessor. For example two source operands are read from the register file before the instruction is decoded. This reduces the critical path but consumes extra energy, since one or both operands may not be needed. One simple way to eliminate the waste is to pre-decode the instruction as they are fetched from the second-level cache, and store a few extra bits in the instruction cache. Each bit would indicate whether a particular operand should be fetched from the register file. Since the energy required to perform a cache access is not a strong function of the number of bits accessed the cost will be small.

The designers of the DEC 21164 chip used a similar idea [4]. The second level cache is 96KB 3-way set-associative. To reduce the power dissipation the cache tags are accessed first. Once it is known in which bank, if any, the data is resident in, only that bank is accessed. This reduces the number of accesses from 4 to 2, reducing the power dissipation. However, the first-level cache refill penalty increases by one cycle.

The idea of reducing speculative operations should not be taken too far. In TORCH, as in most microprocessors, computing the next program counter (PC) is one of the critical paths. The machine must determine whether a branch is taken, and then perform an addition to either compute the branch target or to increment the current PC. To remove the adder from the critical path, TORCH performs both additions in parallel, while the outcome of the branch is determined, and then uses a multiplexer to select one of the two results. The energy saved by eliminating the extra addition is negligible, while the cycle time penalty is large. Thus designers should carefully consider the tradeoff when eliminating speculative operations.

Figure 3.3 shows the energy saved using the simple optimizations presented in this section. The bar labeled “Original” refers to the un-optimized processor. In “Clock” only clock qualification is enabled. In “Selective” clock qualification and selective activation of the caches and register file are enabled. The numbers have been normalized to the energy of the fully optimized design. Thus simple optimizations can save almost 20%-25% of the total power. However further gains would be harder to come by because—as will be shown later—the energy is dissipated in a variety of units, none of which accounts for a significant fraction of the total energy.

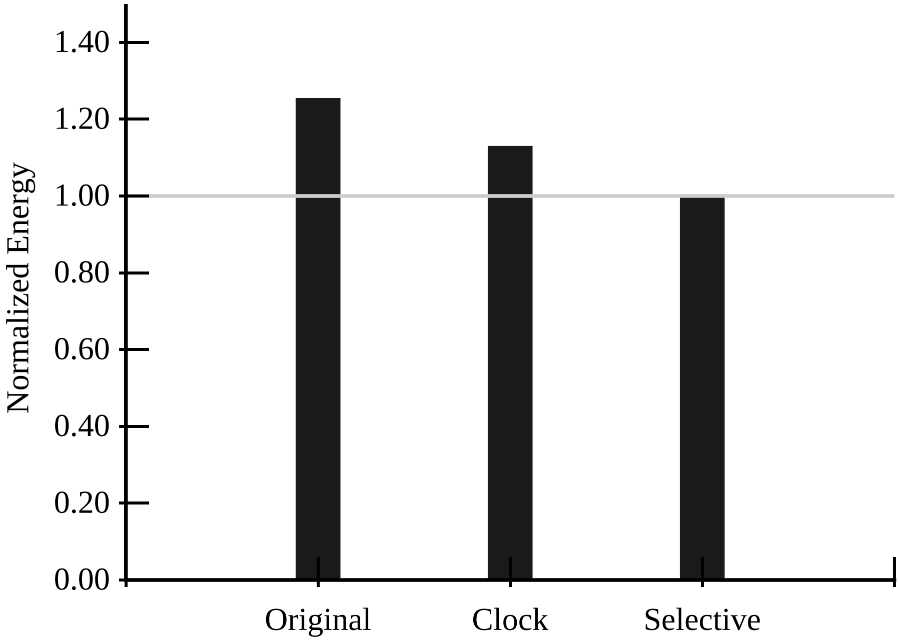


Figure 3.3: Reduction in energy from simple optimizations.

3.2.4 Energy and Energy-Delay Results

Figure 3.4 shows the energy and performance for the ideal super-scalar machine, the simple RISC machine, and TORCH. All figures in this sections have been normalized to the corresponding value for TORCH. As expected TORCH requires the most energy to execute a program, closely followed by the RISC processor. The difference in energy between the ideal machine and TORCH represents an upper bound on potential future improvements from eliminating waste. Since TORCH and the ideal super-scalar machine have identical execution models they have the same execution time. The super-scalar processors have a 1.35X speedup when compared to the single issue machine. With an ideal memory system the speedup would have been 1.6X. This highlights the importance of the external memory system, even in low-power applications. This topic is discussed in more detail in Section 3.4. TORCH and the simple RISC processor have nearly the same efficiency. Super-scalar issue provides a small gain in performance. Unfortunately it also increases the energy cost of all instructions. The net result is no significant improvement in energy-delay product.

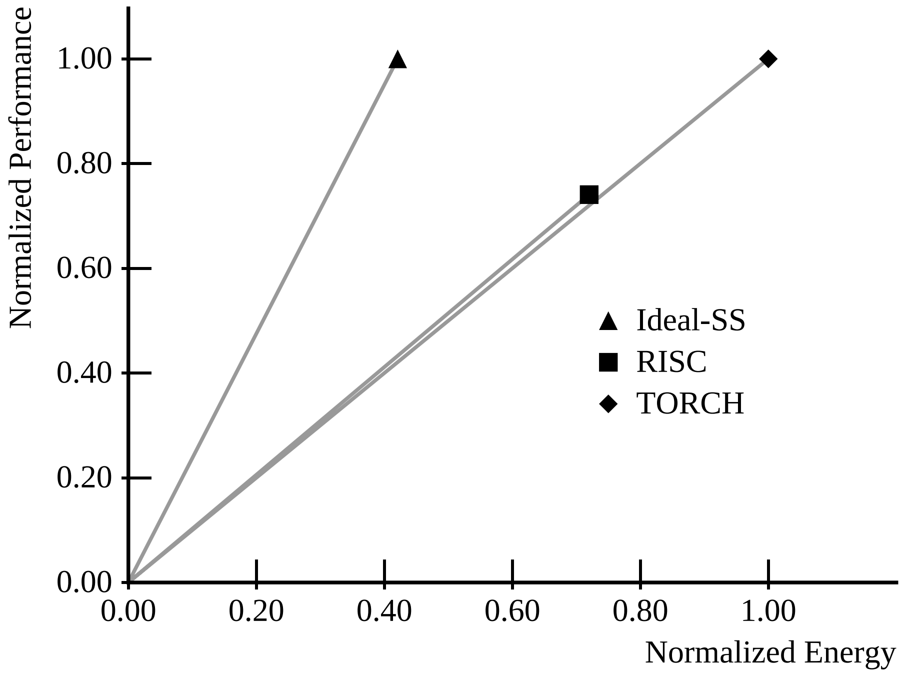


Figure 3.4: Normalized energy and performance for RISC and TORCH.

We expect the power dissipation of the super-scalar processor core to be about 1W-2W at 100MHz in a 0.8µm technology.

The ideal super-scalar processor is roughly twice as efficient as the real machine. This means that no important sources of energy dissipation were overlooked in the idealized machines. In order to understand how to improve the efficiency of TORCH the next section looks at where the excess energy is dissipated.

3.3 Energy Breakdown

In this section the energy dissipation is broken down into four categories. The first is energy dissipated in reading and writing memories, such as the caches, and the register file. The second category is energy dissipated in the clock network. The third is energy dissipated in global interconnect nodes, such as the instruction bus, operand buses, and so on. The final category comprises everything not in one of the previous categories, and includes all control and data logic.

Figure 3.5 shows the energy breakdown for three of the five machines simulated. The figures have been normalized to the total energy dissipation of TORCH. The energy dissipated in the on-chip memories is almost the same in the ideal machines and in our implementations. This means that the optimizations to reduce the number of unnecessary cache accesses were successful. The clock power in the real machines is slightly higher since there is a large amount of state that was not considered in the ideal machines. Although each storage element dissipates very little energy, the sum total represents a significant fraction. Global interconnect and computation each represent about 15% of the total energy. The most important point in this graph is that further improvements in energy-delay product will be hard to come by because the energy is dissipated in many small units. To reduce the energy significantly one would have to reduce the energy of many of these small units.

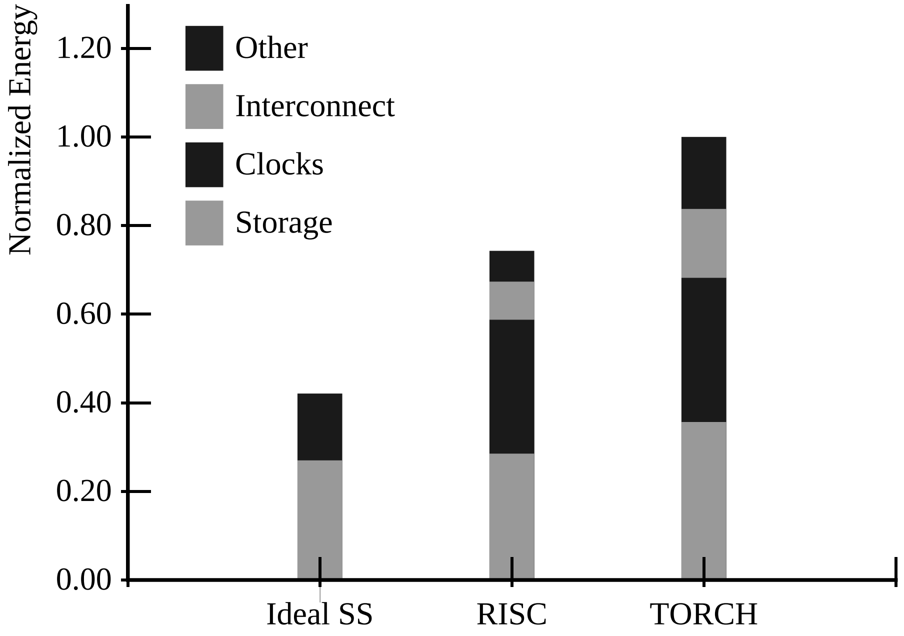


Figure 3.5: Energy breakdown for RISC and TORCH processors.

This section ignored the energy required to communicate with the external memory system and the energy dissipated in the external memories. Since such a large percentage of the energy is dissipated in the on-chip memories it is only natural to wonder how the energy and delay of the processor depend on the organization of the external memory hierarchy. The next section explores these tradeoffs.

3.4 Memory Hierarchy Design

The number of different ways to organize the memory system is almost unlimited, which explains the large number of papers related to memory hierarchy design. Przybylski [33] presents an excellent description of memory hierarchy design for high performance. This section extends that analysis to cover power as well as performance. This section describes the typical architecture of processor systems, followed by a description of the simulation methodology used. Then Section 3.4.3 explores how different memory system parameters affect the performance and power of the system.

3.4.1 System architecture

Processor systems are generally organized as shown in Figure 3.6. A fast bus connects the processor, second level cache (L2) and the cache controller (CC), although increasingly the cache controller is integrated with the processor. This bus usually operates at a high frequency (~100MHz), and requires special termination in order to guarantee signal integrity. Therefore the power dissipated to drive the bus can be significant, even if the total capacitance is not large. The processor, cache controller, and main memory are also connected to the main processor bus. Other chips, such as the I/O controller and memory controller that need to read or write main memory are also connected to the processor bus. The processor bus operates at only a fraction of the speed of the L2 bus. Due to the large loading on the main processor bus the energy required to drive it is also considerable.

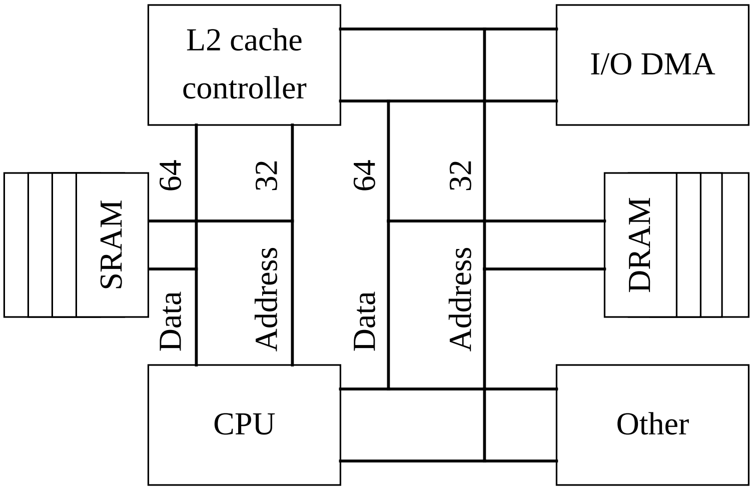


Figure 3.6: Architecture of processor system.

Normally the first level cache (L1) is on the processor itself. Some systems contain only one level of caching. In this case the L2 cache and the L2 bus disappear. If present, the second-level cache can be on- or off-chip. An external L2 cache is built using standard SRAM parts. The tag portion of the cache can be built using SRAM's or can be integrated with the processor. Main memory is built using standard DRAM chips. The caches are assumed to be write-back in order to reduce accesses to lower-levels.

3.4.2 Simulation Methodology

In this section the processor is modeled as dissipating a fixed amount of energy per instruction executed. As was discussed earlier most of the energy is dissipated in essential functions that all instructions must perform, so differences in the energy dissipation of instructions are small. It is also assumed the processor on average executes 1.45 instruction every cycle. This corresponds to 0.688 cycles per instruction (CPI)—same as TORCH with a perfect memory system. The actual CPI of the processor is the base CPI plus the average CPI contribution due to the memory hierarchy.

A trace-driven simulator [34] was used to find the miss rate of the SPECint92 benchmarks on a particular memory organization. The timing model presented by Wilton and Jouppi [35] is used to find the access time of the on-chip caches. It is assumed that the cycle time of the processor is the maximum of a base number (7ns) or the access time of the L1 cache. For most processors accessing the instruction and data caches are two of the critical paths. Increasing the size of the caches decreases the maximum operating frequency. Only for very small sized caches is the cycle time limited by other logic on the processor. The model presented by Amrutur [16] and described in Appendix B is used to find the energy dissipation of on-chip caches. The access time of external components (L2 and main memory) is set by the access time of the chips, plus some overhead to do processing.

Table 3.1 and Table 3.2 describe important characteristics of the SRAM and DRAM chips used to built L2 and main memory. The simulated DRAM's are extremely fast in order to make the simulations conservative. Since DRAM power is specified as a constant I_{DD} current while the part is active reducing the access and cycle time reduces the energy dissipation per access. Furthermore, it reduces the processor performance degradation that is due to the memory system. One very important consideration is how the DRAM chips are accessed. To build a 32-bit wide memory, it is possible to access 4 8-bit wide chips or 8 4-bit wide chips (ignoring parity for this example). Part of the energy dissipated in accessing a DRAM is fixed overhead, and the remainder is linearly dependent on the number of bits read. The overhead is due, among other things, to DRAM's reading thousands of bits during the row access (RAS). These bits are held in the sense amplifiers and are then driven off-chip during column access (CAS). Increasing the access width will amortize the overhead over a larger number of bits, reducing the energy dissipation. The increasing size of DRAM chips also encourages designers to use wider DRAM's because it allows the user to add main memory in smaller increments.

Manufacturer	Cypress
Size	1Mbit (64K x 18)
Active I_{DD} current	265mA
Standby I_{DD} current	20mA
Access width	18 bits (16 bits data, 2 bit parity)
Pad capacitance	5pF
Access time	10ns
Miss time	6 cycles

Table 3.1: Summary of SRAM characteristics.

In Table 3.1 miss time refers to the number of processor cycles from the moment the L1 cache detects a miss until the second level cache returns the first word. The transfer time depends on the line size.

Manufacturer	NEC
Size	16Mbit (1M x 16)
RAS time	60ns
CAS time	40ns
Active I_{DD} current	170mA
Refresh I_{DD} current	170mA
Standby I_{DD} current	110µA
Access width	16 bits
Pad capacitance	5pF
Access time	100ns
Miss time	19 cycles

Table 3.2: Summary of DRAM characteristics.

In Table 3.2 miss time refers to the number of processor cycles from the moment the processor detects a miss until it receives the first word from the DRAM.

The results shown below ignore the energy dissipated in the DRAM refresh. Although the I_{DD} current during refresh is comparable to the active I_{DD} current, time spent doing refresh is much smaller than the time spent in active mode. The refresh power is in the order of 10mW, which is much smaller than the access power. One needs to worry about the refresh energy only when the processor is idle for extended periods of time but the DRAM's are not in standby mode.

For these simulations it was assumed the L2 bus uses series termination such that the energy dissipation can be modeled as $1/2CV^2$. Using series-parallel termination would add 10mW-40mW of power to the bus, due to short-circuit paths between supply and ground. Using parallel termination would add a fraction of that power, depending on how often the driver pulls against the resistor. Assuming a random distribution half the cycles would have static current, so the additional power would be 5mW-20mW. For comparison the dynamic power of a 50pF capacitor at 100MHz is in the order of 15mW, assuming random distribution of inputs. The power dissipation of the processor bus is also modeled as $1/2CV^2$. The capacitance of the bus depends on how many chips are connected to it because pads are a significant source of capacitance. For these simulation it is assumed the processor, all the memory chips, and two more devices are attached to the bus. The capacitance of the bus also depends on the physical length of bus. PCB traces have a capacitance of approximately 1.25pF/cm of length. The bus is assumed to be 20cm long. The total capacitance of the bus is 125pF.

3.4.3 Energy and Performance Tradeoffs

Figure 3.7 shows the average energy per instruction dissipated in the external memory hierarchy broken up into the different components, as the combined (instruction plus data) size of the first level cache increases. The energy dissipated in the processor is also shown. The instruction and data caches are of equal size. There is no second level cache. The main point to notice is that DRAM's dissipate large amounts of energy per access. Therefore small caches are inefficient because main memory must be accessed often. As the size of the first level cache increases the DRAM's are accessed less frequently. It is not until the first level cache is relatively large (32K) that the energy dissipation of the DRAM's becomes comparable to that of the first level cache. Having a small cache does not only affect the energy dissipation, it also affects the performance. As the size of the first level cache increases, the miss rate decreases. Figure 3.8 shows a plot of the energy-delay product for the same memory hierarchy. Most of the change in energy-delay comes from a lower CPI due to a lower miss rate. Ideally as the cache size increases the energy-delay should decrease quadratically. But this ignores the fact that the cycle time of the processor changes as the L1 cache size changes. If the cache is small increasing the size increases the performance because the performance gain due to lower miss rate is larger than the performance lost due to slower cycle time. As the miss rate curve flattens out, increasing the cache size results in decreased performance. The other drawback of increasing the cache size is that the die area and on-chip power increase. For many applications die area and power are tightly constrained—for example power must be less than 1W-1.5W if the processor is to be sold in a plastic package. In this case system efficiency must be traded off for lower processor cost.

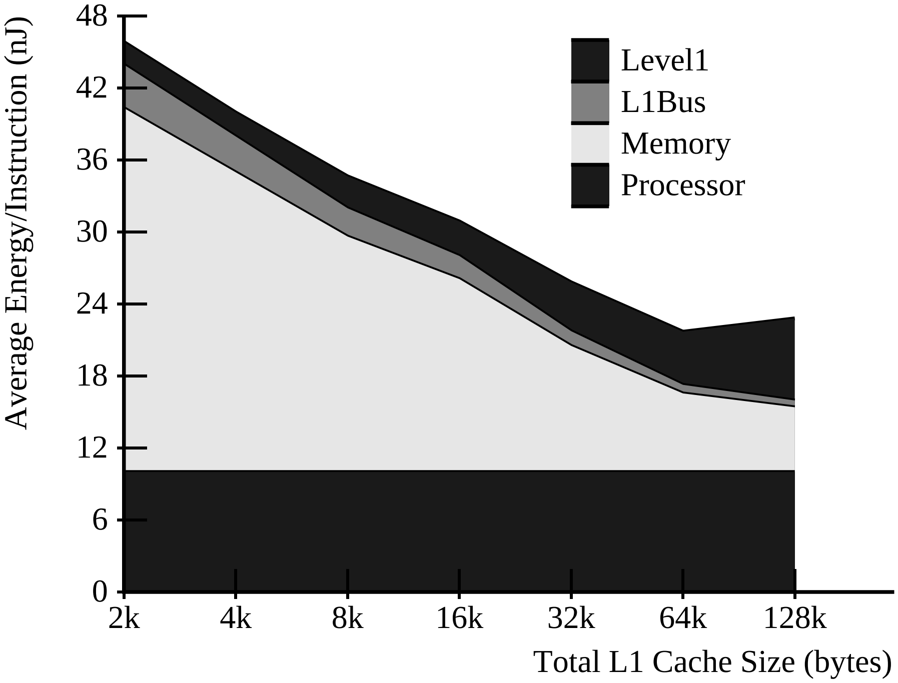


Figure 3.7: Energy breakdown for single level hierarchy.

One way to improve the efficiency of processor would be to integrate the DRAM and the processor on the same die. This eliminates some of the power dissipation in the processor bus and increase the performance of the processor. Ideally this would eliminate the L1Bus component in Figure 3.7. But since the cost of driving on-chip wires from the DRAM to the processor is not negligible the gains will be smaller. Having the processor and DRAM on the same die would also allow wider buses between them, which would in turn reduce

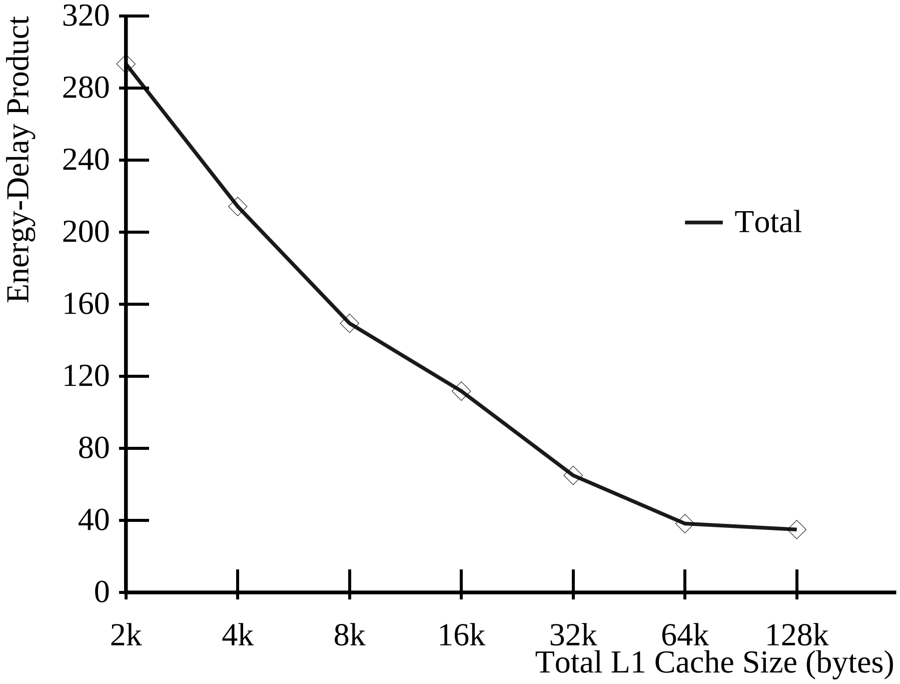


Figure 3.8: Energy-delay product for single level hierarchy.

the cache refill time. The data transfer time could potentially shrink from 8 cycles (32B line and 32b bus) to 1 cycle (256b interface). If the processors waits for the entire transfer to complete before re-starting this reduces the average CPI by 0.1. Increasing the size of the databus would also increase the efficiency of the DRAM. As was stated earlier in a DRAM hundreds or thousands of bits are accessed on every read. In conventional DRAM's most of these bits are unused and therefore the energy spent reading them is wasted. If the bus from the memory is much wider then this overhead will decrease. Quantifying this reduction is hard since it depends on the internal architecture of the DRAM. Another alternative is to latch the data read from the DRAM core in the sense amplifiers, similar to how Rambus DRAM's work [36]. Most of a processor's main memory requests are for a complete L2 cache line. The entire cache line could be read once and held on the sense amps while it is driven to the processor.

There are other cache parameters that affect the power and performance of the processor. Figure 3.9 shows how the energy-delay changes as the degree of associativity of the cache increases. For very small cache sizes increasing the associativity decreases the miss rate improving the EDP. As the size of the cache increases the energy cost of higher associativity—having to access multiple banks simultaneously—overcomes the decreasing miss rate advantage. So for large cache sizes direct mapped caches are the most efficient. One way to improve the efficiency is to use column associative caches [37]. Column associative caches use a hash-rehash algorithm. The cache first tries to find that data using the normal hash function. If the data is not found, then it will try using a rehash function. How to define the rehash function is arbitrary, but Agarwal suggest using bit-flipping. This algorithm eliminates the need to access two cache banks in parallel, and eliminates the tag comparator and data multiplexer from the critical path of the cache. This technique works well because data can most often be found using the first hash function. Only very infrequently is it necessary to rehash into the cache. Thus the performance and energy overhead are small. Agarwal has found the performance of column associative caches to be almost identical to that of 2-way set-associative caches. Our own simulations show the energy dissipation is similar to that of a direct mapped cache.

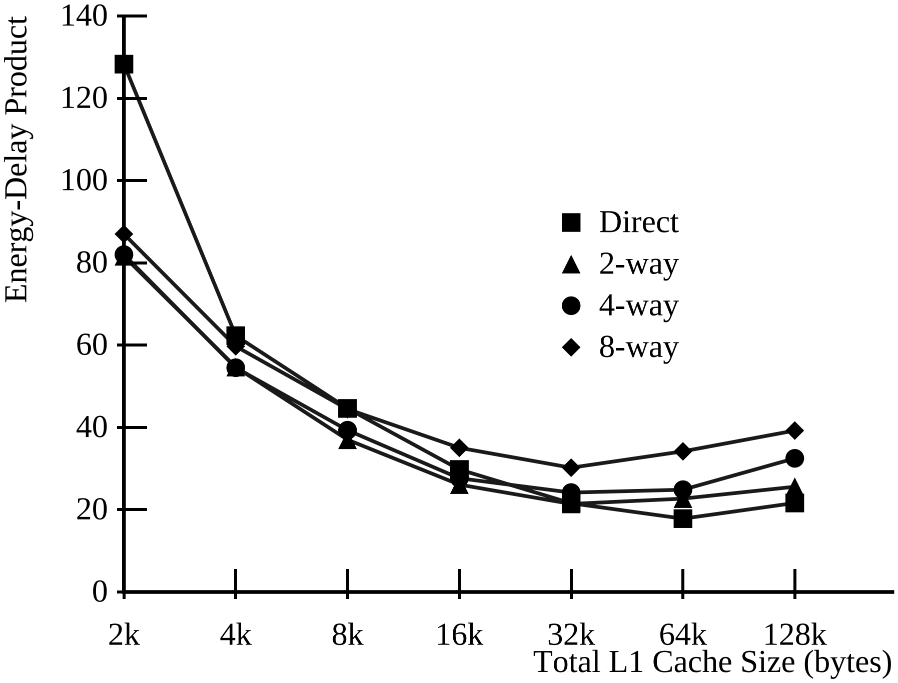


Figure 3.9: Energy-delay versus associativity for single level hierarchy.

Figure 3.10 shows how the energy-delay changes as the line size of the cache increases. Although there is some improvement due to larger lines—because larger lines have smaller miss rates—the effect of line size in the efficiency of the system is small.

Increasing the L1 cache size has a large effect on performance and therefore on energy-delay. Another way to increase the performance and reduce the energy is to add a second-

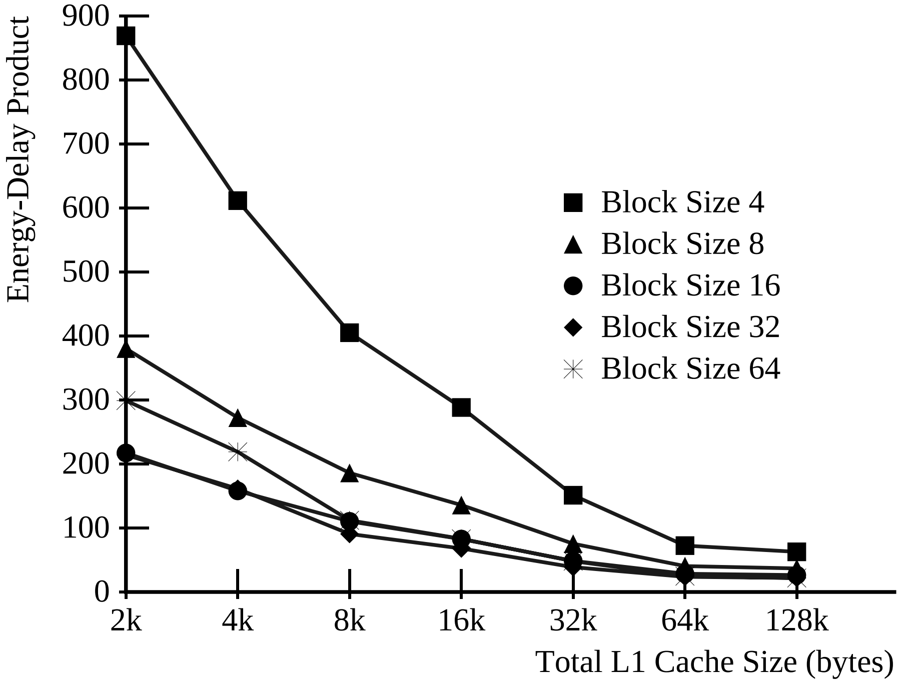


Figure 3.10: Energy-delay versus line size for single level hierarchy.

level cache. Ideally the L2 cache should be implemented on the processor die in order to reduce miss penalty by making the cache access as fast as possible. Adding a second-level cache instead of using very large first level caches has several advantages as described by Jouppi and Wilton [38]. The two most important advantages are first, the access time and access energy of the first-level caches are smaller. This increases the frequency of the processor and reduces the average energy per instruction. And second, because the access time of the L2 cache is not as critical it can have higher associativity. To reduce the energy dissipation it is possible to first access the tag arrays, determine in which bank, if any, the data resides in and then access only that bank, as it is done in the **21164** chip from DEC. Figure 3.11 shows the energy-delay product for a system with 128Kbyte 4-way set-associative on-chip L2 cache as the combined first level cache size increases. As a reference the EDP of a system with single-level caching is shown as a dashed line. The change in energy-delay as the size of L1 caches changes is much smaller. The L2 cache filters most of the misses that would otherwise have gone to main memory. Increasing the size of the L2 cache increases the performance slightly but in our model it also increases the energy so the change in EDP is almost negligible. For benchmarks with large data-sets having a larger L2 cache would be advantageous. The energy of main memory is fixed because the number and frequency of accesses to main memory depend only on the characteristics of the L2 cache. The energy dissipated in the L1 cache remains unchanged if compared to a system with a single level of caching. The L2 cache does dissipate some additional energy, but less than the energy that would be dissipated by the DRAM's if the L2 cache was not there.

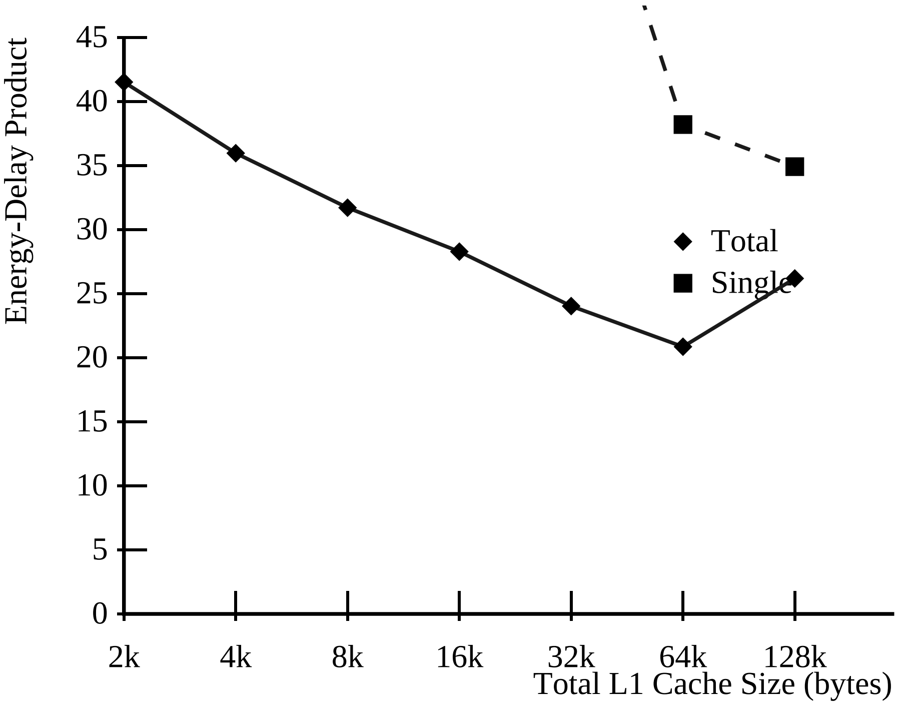


Figure 3.11: Energy-delay for two-level on-chip cache hierarchy.

In most processors it is not possible to have so much on-chip memory due to die size limitations. In this case the L2 cache can be implemented with commercial SRAM chips. The main reason having an L2 cache is advantageous is that the access energy of SRAM chips is lower than that of DRAM chips. Figure 3.12 shows the average energy per instruction for such a system broken down into its consisting components. As a reference the average energy per instruction for a single level hierarchy is shown as a dashed line. The L2 cache is held fixed at 512KB and 4-way set associative. There is an advantage to having an external L2 cache when the first-level caches are small. If the L1 caches are large (32KB or more each) then a single level hierarchy is more efficient. Figure 3.13 shows the energy-delay product for the all three scenarios: single level, two level hierarchy with L2 on-chip, and two level hierarchy with L2 off-chip. Adding as much on-chip cache as possible makes the system more efficient. This on-chip RAM should be split into two smaller L1 caches and a larger unified L2 cache. When the amount of on-chip cache is limited then using a large off-chip L2 cache is advantageous.

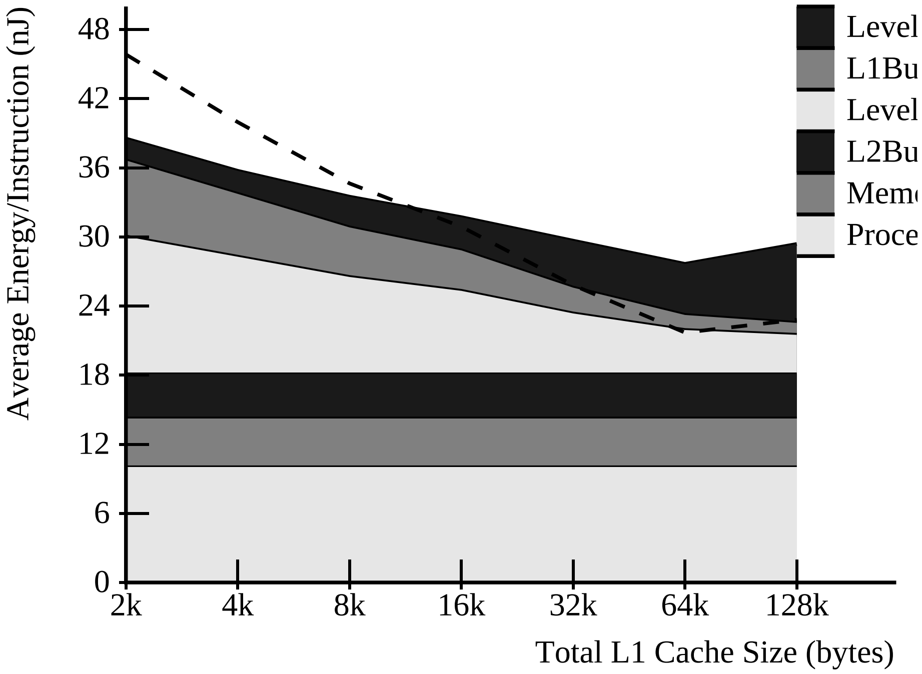


Figure 3.12: Energy breakdown for two-level hierarchy.

In summary exploiting locality is important not only for performance but also for energy reasons. This is a general principle that can be applied to any low-power design. Accessing a smaller local memory is more efficient than accessing a larger memory so long as the data is used often enough that the cost of maintaining the consistency between the two memories is small.

The task of designing a high performance memory system has become more difficult during the past 10 years because the speed of processors has increased at a much higher rate than the speed of memories. Latency to main memory—measured in processor cycles—has increased from tens to hundreds of cycles. The next section looks at some techniques that attempt to hide this latency by increasing the complexity of the processor.

3.5 Future Directions

The time for a processor to execute a benchmark is given by the product of three quantities, the average number of cycles per instruction (CPI), the cycle time (seconds per cycle), and the number of instructions. The number of instructions is set by the particular instruction set architecture (ISA). Since most current processors designs implement an

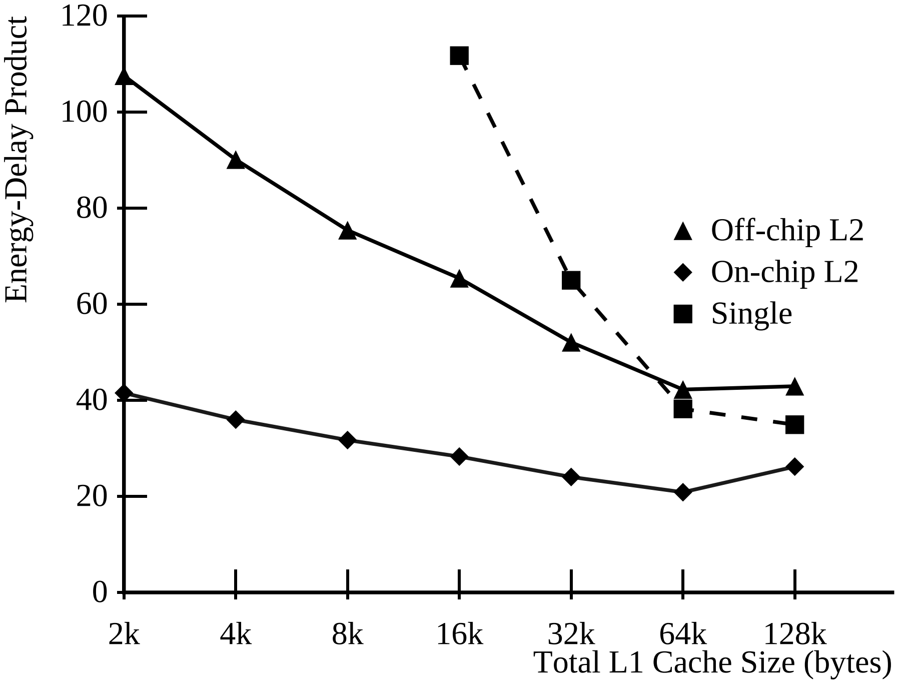


Figure 3.13: Comparison of three memory hierarchies.

existing ISA the number of instructions is fixed. To improve the performance designers usually try to minimize the CPI and the cycle time.

The CPI of a processor has two basic components. The first, CPI_{base}, is the average number of cycles required to execute an instruction assuming an ideal memory hierarchy. The second component, CPI_{mem}, is due to the memory system. For example, with an ideal memory system TORCH has a speedup of 1.6X over the simple RISC machine. With a realistic memory system, however, TORCH has a speedup of only 1.35X over the simple RISC processor. Roughly 30% of the TORCH execution time is spent stalled waiting for the memory system. This is because whenever there is a data cache miss the processor stalls until the data is fetched from the second level cache, or from main memory in the case of a second-level cache miss[2]. There are two ways to reduce the stall time. The first is to design a more aggressive memory system. As the previous section described, much effort has already been spent to improve the performance of the memory system. Future gains from further tweaks will probably be small. The second option is to attempt to hide some of the latency of load instructions by having the processor continue to execute instructions while it waits for data from the second level cache.

[2] Instruction cache misses are a major performance bottleneck for some applications. In this case, however, there is not much one can do other than make the instruction cache as large as possible.

In early RISC processors when an instruction missed in the cache the whole processor would stall until the data arrived. This approach is attractive because the processor does not need to explicitly track dependencies between instructions. A simple approach to increase performance is for the cache to stall the processor only when the requested data is used, or when another memory request is made. Instructions which have no data dependencies and which do not access the cache can be executed. Unfortunately the number of instructions that meet these criteria is small so the performance benefit is small. But the energy cost of this feature is also small. To implement this approach only a couple of comparators are required. Each launching instruction compares its source registers with the destination register of the load that missed[3] and on a match the processor stalls. One optimization is to allow the cache to service other requests while it waits for data from the second level cache, normally called a hit-under-miss cache. The cache stalls only when a second instruction misses. This optimization is conceptually simple but requires more control complexity in the cache, since more events can occur simultaneously. A further optimization is to allow the cache to have multiple outstanding requests, which is called a non-blocking cache. More than one instruction can be waiting for data from the second level cache. The processor can continue to execute instructions as long as it does not encounter a data dependency. A valid (or presence) flag for every register is required. When a load misses in the cache the valid flag for the load’s destination register is cleared. If an issuing instruction depends on register whose valid flag is cleared the machine stalls. But the basic problem with these approaches is that the processor stalls as soon as it encounters an instruction that depends on the load that missed.

To further improve performance processors can change the execution order of instructions, a feature known as out-of-order issue. Program semantics require that instructions be executed in program sequence. Instruction number *i* must be executed before instruction *i+1*, even if there are no explicit data or control dependencies between the two instructions. Thus in an in-order machine if instruction *i* causes a cache miss and instruction *i+1* depends on instruction *i* then the processor must stall because subsequent instructions cannot be executed. An out-of-order machine, on the other hand, may speculatively try to execute instruction *i+2* if it has no dependencies with instructions *i* or *i+1*. The instructions are executed speculatively because the architectural state of the machine cannot change until all previous instructions have executed in order to maintain program semantics. In the simple example above the processor must buffer the results of instruction *i+2* until instructions *i* and *i+1* complete. The hardware to perform

[3] The instruction must also compare its destination register to guarantee output-dependencies.

dependency analysis and result buffering is complex and can have a significant impact on the cycle time and power dissipation of the processor.

Several papers have looked at the benefit of out-of-order issue machines and found that the performance benefit is modest. Wilson and Olukotun [39] estimate the performance benefit of a non-blocking cache on an aggressive out-of-order processor to be between 1%-18% for typical integer benchmarks. The **MIPS** R10000 processor uses both out-of-order issue and a non-blocking cache to improve performance. According to the designers [40] the non-blocking cache reduces the average cycles per instruction (CPI) by 10%-20% on benchmarks that do not have very large amounts of parallelism. The change in CPI does not take into account the degradation in cycle time due to the complexity of the micro-architecture. The actual performance gain would be smaller than the 20% figure quoted. Farkas [41] also concluded that out-of-order issue and non-blocking caches provide a 20%-40% performance improvement. Other researchers [42] have published similar results.

Quantifying exactly the energy cost of out-of-order issue is difficult, since it is hard to attribute overhead to a particular architectural feature. Part of the cost, for example, is due to the issue width of the machine (usually 4 or more instructions per cycle). The energy cost of building such a wide machine is higher than the results presented earlier in this chapter. In this section it is assumed that out-of-order machines add four basic operations that need to be performed per instruction; rename, wakeup, select, and reorder [43]. A brief description of these four operations is given below. For a more complete description of out-of-order machines see [28] or [44].

Rename is a mechanism to eliminate anti- and output-dependencies. It translates a logical register number into a physical register number. Each live variable receives a different physical register number thereby eliminating all but true dependencies. In addition, the rename logic must check for true dependencies between instructions being renamed in parallel. There are two ways to implement the rename logic. One is to use a RAM array which is accessed using the logical register designator. The array contains the physical register number which maps to that logical register. In addition the processor must keep a list of free physical registers which is implemented as a FIFO queue. The energy cost is set by the read/write energy of the narrow (6- or 7-bit) register file. The second scheme uses a CAM array which has one entry per physical register. Each entry contains the logical register that maps to that physical register and a valid bit which indicates if the mapping is valid. In a CAM all the word-lines are pre-charged high and all but one discharge every cycle. Thus the energy of the latter technique is likely to be higher.

Wakeup logic tracks dependencies of instructions in the issue window and wakes up the instructions when all their source operands are available. The wakeup logic can be implemented as an independent unit—as in the MIPS R10000 [40] and the Intel Pentium-Pro [45]—or as part of the reorder mechanism—as in the HP-PA 8000 [47]. The cost of the latter method is describe below.

Selection logic chooses which instructions to execute amongst all ready instructions. It is usually implemented as a two-stage priority encoder. The first stage selects one instructions from a small group, usually four instructions. The second stage selects one instruction from each of the oldest groups. The selection logic can be simplified even more if issue restrictions are placed on instructions. This technique can reduce the number of priority encoders needed. This logic is also often implemented as part of the reorder mechanism, which is described next.

The **reorder** buffer (RB) ensures that instructions update architectural state in-order. The basic reorder structure is a FIFO queue. Each instruction occupies one entry in the FIFO. Instructions are placed in the queue in program order and are retired in program order. An instruction is not allowed to modify architectural state until all previous instructions have completed therefore guaranteeing program semantics. The processor must also recognize when load and stores to the same memory location are performed out-of-order. If, for example, a store writes memory location *i* and a later load reads the same location the instructions must be executed in order. One simple way to solve the problem is to guarantee that loads never bypass stores and stores remain in order. More complicated designs [46] compare the cache index of loads and stores and hold stores which match the index of store waiting to execute.

Adding comparators to each reorder buffer entry allows the wake-up logic to be incorporated as part of the RB. Each instruction must compare its source operands with the destination register of launching instructions. On a match the dependency is cleared. For a 4-issue MIPS machine each entry requires 8 5-bit comparators. If the cache indices of loads and store are also compared then an additional 15-20 1-bit comparator are needed. The selection logic can be implemented as a tree of priority encoders along the side of the reorder buffer. The remainder of this section gives a simple power estimate for implementing this functions.

The power dissipation of these four functions can be estimated by finding the number and length of wires required to implement the logic. Because so many long wires are needed most of the power is due to wire capacitance. For the power estimates presented below a 0.6µm technology with 3.3V supply running at 200MHz is assumed. This is a more advanced technology than the 0.8µm technology used elsewhere in this section. This technology makes the processor more efficient—lower EDP—but also higher power since it operates at a higher frequency—200MHz instead of 100MHz. The change in EDP due to technology scaling is a factor of 3.2. Therefore the processor core described earlier operating at 200MHz would dissipated 0.6W-1.3W in this technology.

The cost of the wake-up logic is the energy needed to broadcast the destination tags of all launching instructions (usually done as true/complement pairs) plus the energy of the comparators. For a 4-issue machine this requires 1W at 200MHz if the wake-up logic is implemented as part of a 56-entry reorder buffer.

The energy cost of the reorder buffer can be broken down into five main components. First, energy is required to drive the source tags of new instructions from the fetch unit into the RB (up to 4 instructions per cycle). Second, energy is required to drive the results of executed instructions from the execution units back into the RB, (again up to 4 instructions per cycle). Third, energy is required to bypass results being held in the RB to executing instructions when there is a dependency. Fourth, energy is required to drive the results of instructions being retired from the RB to the register file. Fifth, energy is needed to clock storage elements and pre-charge dynamic circuits. Clock power can be minimized by only clocking entries which are being updated. The power requirements are 0.4W, 2.2W, 1.9W, 1.1W, and 0.36W respectively. The resulting power for the reorder buffer is 6.0W. Adding comparators to ensure dependent loads do not overtake stores would add an additional 1.2W. This power does not take into account the cost of building a wider machine. Because the cost of a wider machine grows quadratically the energy per instruction is now higher.

Even ignoring control overhead, which is not trivial, the added power due to out-of-order issue and non-blocking caches appears to be larger than the 20%-40% performance gain they provide. These architectural features will therefore most likely not appear in low-power processors unless memory latency, measured in processor cycles, continues to increase. Then these features would become more attractive.

3.6 Summary

Exploiting parallelism gives limited improvements in energy-delay mainly because the amount of parallelism in common integer benchmarks is small. Thus pipelining gives a 2X improvement in EDP, but most processors do this already. Super-scalar issue, on the other hand does not significantly alter the energy-delay. If the mix of applications that are typically run on processors changes, then having a wider issue machine may be more attractive.

Although it is possible to reduce the energy requirements by careful design, this energy gain will be a one-time improvement. It was shown that by using easy-to-implement optimizations the energy can be reduced by approximately 25%. Further improvements would require much more sophisticated optimizations since the remaining energy is dissipated in many small units none of which account for a significant fraction of the total energy dissipation.

The memory system accounts for a large fraction of the total system power, sometimes more than the processor itself. Optimizing the memory hierarchy for performance, however, will in general give a solution which is close to optimal for efficiency. Ideally the processor should have a large first level cache, or a small first-level cache and a large (>128KB) unified second-level cache on-chip. If it is not possible to have large amounts of memory on-chip then adding an external second-level cache is advantageous.

It was shown that improving the efficiency of the processor purely by architectural means is hard. Sophisticated architectural features, such as non-blocking caches and out-of-order execution require large structures which dissipate large amounts of power and provide modest performance gains. Furthermore, for a simpler processor clocking and memories account for a significant fraction of the energy, both on- and off-chip. In order to significantly reduce the energy of the processor it is necessary to reduce the energy of these two components. Other have looked at the low-level circuit architecture of SRAM’s and DRAM’s [16], [48], [49], and the design of storage elements [50]. Researchers published some results indicating gains in efficiency. The next chapter, however, pursues an alternative approach, scaling the supply and threshold voltages.

Chapter 4

Supply and Threshold Scaling

One common technique for reducing the power dissipation of processors is to scale the power supply voltage. This technique is attractive because it reduces the power of all the circuits on the chip, including memories and clocks. For CMOS circuits the cost of lower supply voltage is lower performance. Scaling the threshold voltage can limit this performance loss somewhat but results in increased static power dissipation. Burr *et. al.* [51], [52] have shown that if one optimizes CMOS circuits for minimum energy then operating in the sub-threshold region is advantageous. This chapter explores the effect of scaling the supply and threshold voltages on the energy and delay of CMOS circuits.

The next section extends the first order model of energy and delay of CMOS circuits presented in Chapter 2 to include leakage power. Using this model one can find the optimal operating point, the value of supply and threshold voltage for which the product of energy and delay is minimum, as well as how this optimal operating point will change as circuit and process parameters change. Section 4.3 extends the model to take into account the uncertainty in the value of the supply and threshold voltage. Then Section 4.4 presents some promising adaptive techniques to help deal with the uncertainty inherent in CMOS circuits. The last section describes how these techniques can also be used to trade excess performance for lower power.

4.1 Energy and Delay Model

As described in Chapter 2, the two main sources of power dissipation in a CMOS gate are static current, which results from resistive paths between power supply and ground, and dynamic power, which results from switching capacitive loads between different voltage levels. In Chapter 2 the energy per operation for a single CMOS gate was given as,

$$E = aCV^2 \qquad (4.1)$$

where *a* is the probability of transition of the output node. For a processor, the total dynamic energy is simply the sum of the dynamic energy of all the gates.

Chapter 2 ignored leakage power since it is usually a small fraction of the total power. When scaling the threshold voltage, however, it is important to take leakage into account. The leakage current for a CMOS gate can be written as,

$$I_l = WI_s e^{-\frac{V_{th}}{V_o}} \tag{4.2}$$

where W is the effective transistor width[1] of the cell, I_s is the zero-threshold leakage current, and V_o is the sub-threshold slope. This model ignores the dependence of I_l on drain voltage, and also the leakage current in the reverse biased diodes. The leakage current for the entire chip is simply the sum of the leakage currents of all the gates.

The total energy per operation for the processor as a whole can thus be written as,

$$E = \sum_i a_i C_i V^2 + \sum_i W_i I_s e^{-\frac{V_{th}}{V_o}} V T_c \tag{4.3}$$

where T_c is the cycle time and *i* is an index that runs over all gates in the circuit. Each CMOS gate dissipates static energy throughout the cycle, but dissipates dynamic energy for a short period of time while it switches.

Notice that this equation can be rewritten as the total transistor width times the average energy dissipated per micron of width,

$$E = \sum_i W_i \left(\frac{\sum a_i C_i}{\sum_i W_i} V^2 + I_s e^{-\frac{V_{th}}{V_o}} V T_c \right) \tag{4.4}$$

and is very similar to the energy consumed by a single inverter (with the “correct” average activity *a* and load *C*) so optimizing the energy of this average inverter will yield an

[1] Transistor width that contributes to the leakage current.

optimal operating point for the processor. In fact the optimal point remains unchanged if the equation is normalized by the width of this average inverter, yielding the average energy consumed per micron of transistor width,

$$E_{avg} = C_{avg}V^2 + I_s e^{-\frac{V_{th}}{V_o}} VT_c \tag{4.5}$$

where C_{avg} is the average capacitance switched every cycle per micron of transistor width. This parameter is different for every design, depending on the types of circuits used. For the StrongArm-110 processor from DEC C_{avg}=0.2fF [53]. Since caches—which have very low activity factors—occupy about 50% of the area of this chip one would expect other designs to have larger values of *Ceff*. A value of 1fF was assumed to make lower voltages seem more attractive. Later on it will be shown the location of the optimal point is highly insensitive to the value of *Ceff*.

A similar technique can be used to model the minimum operating cycle time, or critical path, of the processor. The critical path normally goes through a variety of gates, each with a different delay. Fortunately changes in supply voltage, temperature, and threshold voltage affect all gates in nearly the same way so delay of any gate remains roughly proportional to the delay of an inverter, as is shown in Figure 4.1. This figure shows the delay of different circuit elements divided by the delay of an inverter. Solid lines show the delay at high temperature (125˚C), dashed and doted lines show the delay at lower temperatures (25˚C and -25˚C respectively). Since the lines are flat (ratio does not change with supply voltage) the delay of most circuits track the delay of a standard inverter. Thus the critical path can be normalized by dividing the cycle-time by the delay of a standard inverter (T_g). A standard inverter is one which drives four equally sized inverters. The cycle time divided by the delay of a standard inverter is called the logic depth (L_d), since it represents how many inverters are in a ring oscillator which has the same frequency as the maximum operating frequency of the chip. For modern microprocessors the logic depth is usually around 30 standard inverters. The cycle time is then just L_d T_g.

The delay of an inverter was describe in Chapter 2 as,

$$T_g = K\frac{V}{(V - V_{th})^{\alpha}} \tag{4.6}$$

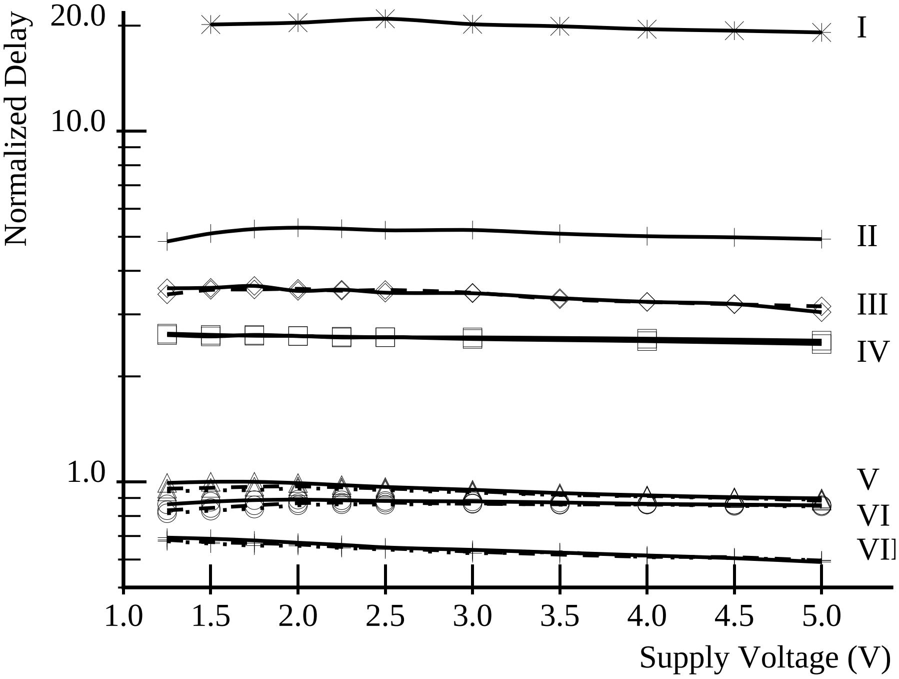


Figure 4.1: Delay of circuit blocks divided by the delay of standard inverter.

Key	Description
I	I-Cache
II	Carry chain
III	Regenerative carry chain
IV	Post-charge logic
V	Stacked inverter
VI	Nor gate
I	Nand gate

Table 4.1: Circuit element description.

where K is a proportionality constant specific to a given technology. Remember the α power accounts for the fact that the transistors may be velocity saturated. If transistors are not velocity saturated then α is 2 and as transistors become more velocity saturated α approaches 1. For a 0.25μm technology α is likely to be 1.3-1.5.

Combining Equation (4.5) with Equation (4.6) the energy-delay product for the average inverter can be written as,

$$EDP = \frac{K^2 I_s L_d V^3}{(V - V_{th})^{\alpha}} \left(K_2 + \frac{e^{-\frac{V_{th}}{V_o}}}{(V - V_{th})^{\alpha}} \right) \tag{4.7}$$

where K_2 is a constant for the given technology and is given by,

$$K_2 = \frac{C_{eff}}{I_s K L_d} \tag{4.8}$$

To find the optimal supply and threshold voltage one needs to differentiate Equation (4.7) with respect to V and V_{th} and set the equations to zero. Solving for V and V_{th} one gets,

$$V = \frac{3V_{th}}{3 - \alpha} + \frac{3\alpha}{3 - \alpha} V_o \tag{4.9}$$

and,

$$e^{-n} = \frac{K_2 \left[\frac{\alpha}{3 - \alpha} (n + 3) V_o) \right]^{\alpha} (3 - \alpha)}{(n + 2\alpha - 3)} \tag{4.10}$$

or

$$n = -\alpha \ln\left(\frac{\alpha}{3 - \alpha} (n + 3) V_o \right) - \ln (K_2) + \ln (n + 2\alpha - 3) \tag{4.11}$$

where

$$n = \frac{V_{th}}{V_o} \tag{4.12}$$

Although it is not possible to find a closed form solution for n, Equation (4.9) and Equation (4.11) can be solved numerically. For the parameters given in Table 4.2 the optimal operating voltage and threshold voltage are quite low, V=254mV and V_{th}=119mV, and only weakly depend on most of the technology parameters. The strongest dependence is on the velocity saturation parameter, α, since the optimal V_{th} is proportional to it. As transistors become more velocity saturated V_{th} at the optimal point decreases, and the optimal operating voltages decreases as well. The threshold depends logarithmically on the other technology parameters ($Ceff$, L_d) present in the K_2 constant. For every order of magnitude change in the effective switched capacitance, due to a change in activity or capacitance, the optimal V_{th} changes by about 70mV. The logic depth is not likely to change by more than an order of magnitude so its influence on the optimal V_{th} is small. Since order-of-magnitude changes in the other technology parameters is also unlikely, their effect on the threshold voltage is likely to be small.

Variable	Value
$Ceff$	1.0fF
K	155E-6
I_s	1mA
α	1.3
L_d	30
V_o	1.3KT/q

Table 4.2: Process and circuit parameters for a 0.25μm technology.

This simple analysis indicated that for advanced technologies, there might be a potential energy saving by reducing both V and V_{th} to relatively small values. To determine the magnitude of the savings it is necessary to determine the steepness of the EDP surface near the minima. One way to do this is to plot contours of constant EDP versus V and V_{th} by numerically solving Equation (4.5) and Equation (4.6). However, sub-threshold current and second-order terms were ignored in the formulation of these equations. This made the equations simple to manipulate algebraically. When finding numerical solutions the complexity of the equations is not as important. Therefore the following equations were used in the simulations.

$$I_l = I_s e^{\frac{(V_{gs} - V_{th})}{\gamma V_o}} \left(1 - e^{-\frac{V_{ds}}{\gamma V_o}} \right) \qquad (4.13)$$

If $V > V_{th}$

$$I = K(V - V_{th})^{\alpha} + I_s\left(1 - e^{-\frac{V_{ds}}{\gamma V_o}}\right) \tag{4.14}$$

else

$$I = I_l \tag{4.15}$$

$$T_g = \frac{CV}{I} \tag{4.16}$$

$$E = aCV^2 + I_s e^{-\frac{V_{th}}{\gamma V_o}}\left(1 - e^{-\frac{V_{ds}}{\gamma V_o}}\right) V L_d T_g \tag{4.17}$$

If the transistors are not velocity saturated (α=2), the EDP surface is relatively flat, as is shown in Figure 4.2[2]. This figure shows contours of the inverse of the relative energy-delay product. The relative energy-delay product can be found by normalizing to the value of the EDP at the optimal point. Thus any point on the curve labeled 0.5 has an EDP value twice that of the minimum. The optimal point is at V=533mV and V_{th}=127mV. But at V=1V and V_{th}=300mV the EDP of the circuit has increased by less than a factor of 2. Thus the benefit of operating at the optimal point is small.

When transistors are velocity saturated, however, the EDP surface is much steeper, which means that the contour lines are much closer together. Figure 4.3 shows contours similar to those of Figure 4.2 but when α=1.3. At the same operating point as before, V=1V and V_{th}=300mV, the EDP of the circuit is 4 times that of the optimal. Thus the model as described so far predicts that very low supply and threshold voltages would be beneficial for reducing power.

[2] The small “kinks” in the curves near the border of the sub-threshold region are artifacts of the model and can be ignored. The current is slightly non-monotonic as the transistor switches from the sub-threshold to the active region.

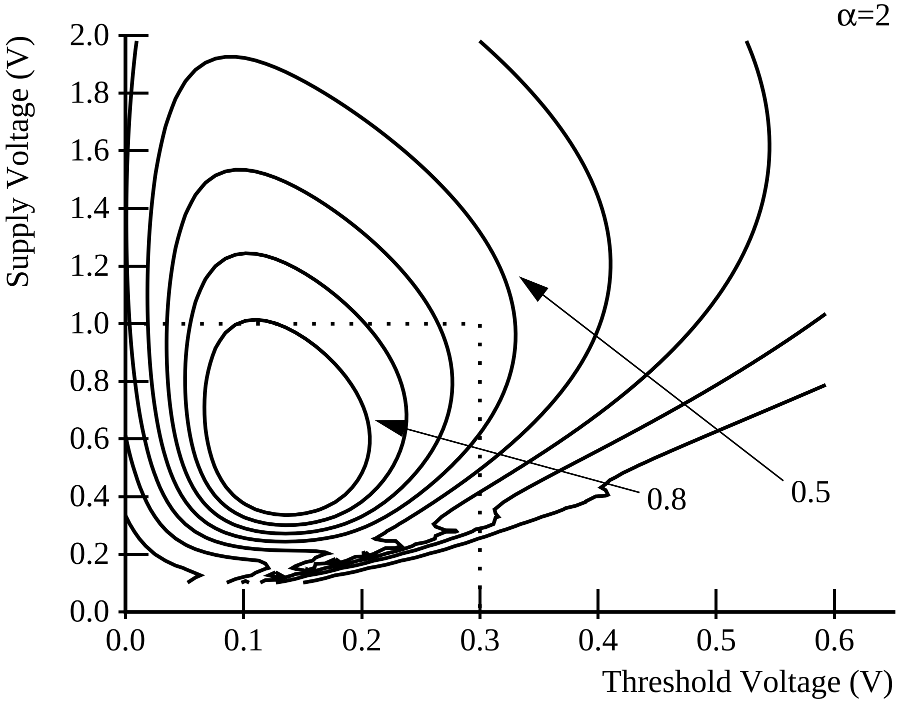


Figure 4.2: EDP contours without velocity saturation.

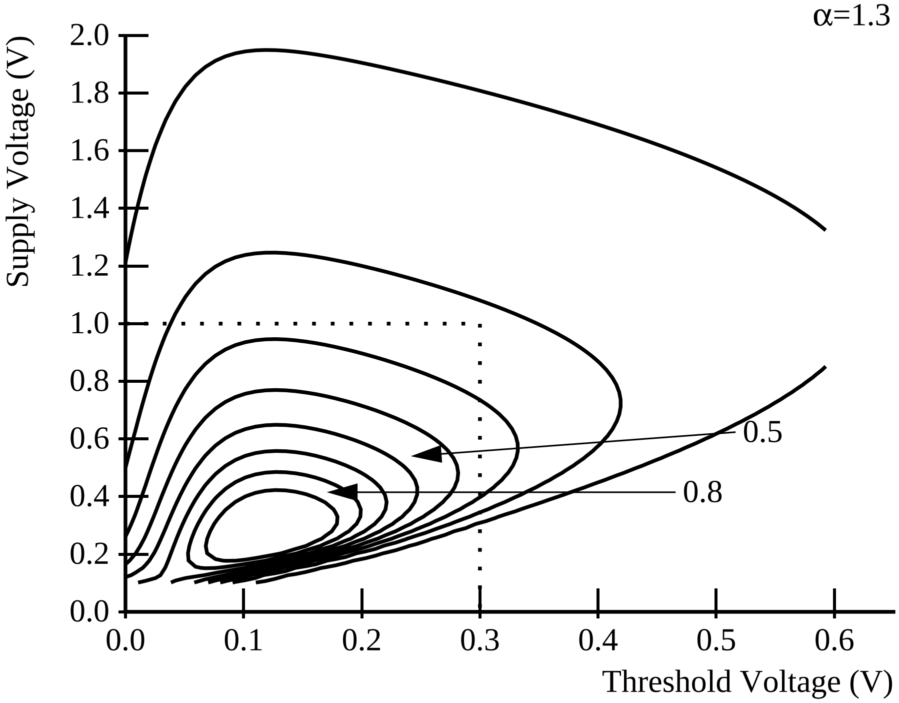


Figure 4.3: EDP contours with velocity saturation.

Most circuits must meet specific performance targets, so it is important to look at the actual performance in addition to the energy-delay product. The dashed lines in Figure 4.4

show contours of constant performance. Performance was normalized to that of the performance contour that runs approximately through the optimal point. If the circuit must operate somewhere along the topmost performance contour, then the designer could reduce power by more than a factor of 3 without changing performance by moving from using a 2V supply and 0.5V V_{th} to using a 0.8V supply and 0.1V V_{th}. But this point is still not at the optimal energy-delay product. To further improve the energy efficiency requires reducing the supply with constant threshold voltage which will make the gates slower. The gates at the optimal point are 2.7X slower than the highest performance curve. If the logic can be changed to reduce the levels of logic in the design, allowing the use of these slower cells, there is about another factor of 3 reduction in EDP.

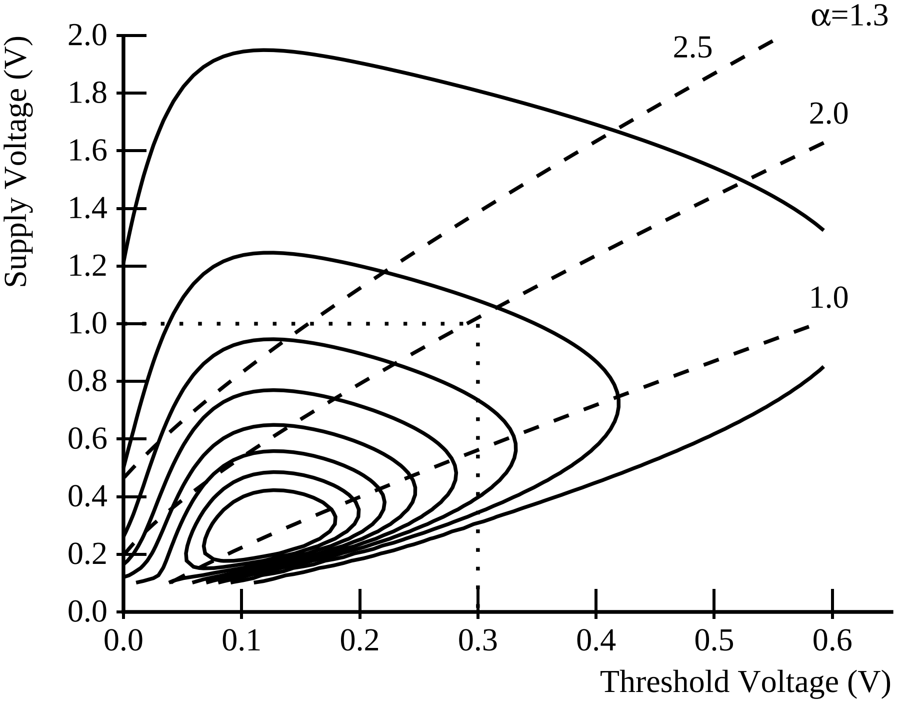


Figure 4.4: EDP and performance contours with velocity saturation.

While the reduced gate performance is one factor that is keeping supply voltages from dropping too quickly, this simple analysis indicates that sub-micron technologies with low threshold voltages should be very attractive for low-power applications. By simply moving to a low V_{th} process, a designer could reduce the supply voltage and power without requiring a major change of the design, since the gate speed would remain constant. If the design could be changed to use slower basic gates, further power savings would be possible. Unfortunately the next two sections show that some of this advantage is illusory; when sleep modes and variation in voltages are taken into account the advantage of operating at these very low voltages is greatly diminished.

4.2 Sleep Mode

There are some applications where the processor must be idle for extended periods of time. During this time it is not always possible to shut-off the power supply since the processor must maintain state. One can reduce the dynamic power by simply reducing or completely stopping the clock signal. However, during this period leakage power remains constant. In recent processor implementations [20] [54] the idle power has been limited by the leakage current of the transistors. One can estimate the ratio of active to idle power by finding the ratio of leakage to total power in the circuit. Figure 4.5 shows the ratio of leakage power to total power. When the ratio of active to idle power must be a factor of 1000 then the threshold voltage will be limited to be at least 300mV or so. This high V_{th} will limit voltage scaling and thus limit the EDP of the circuit.

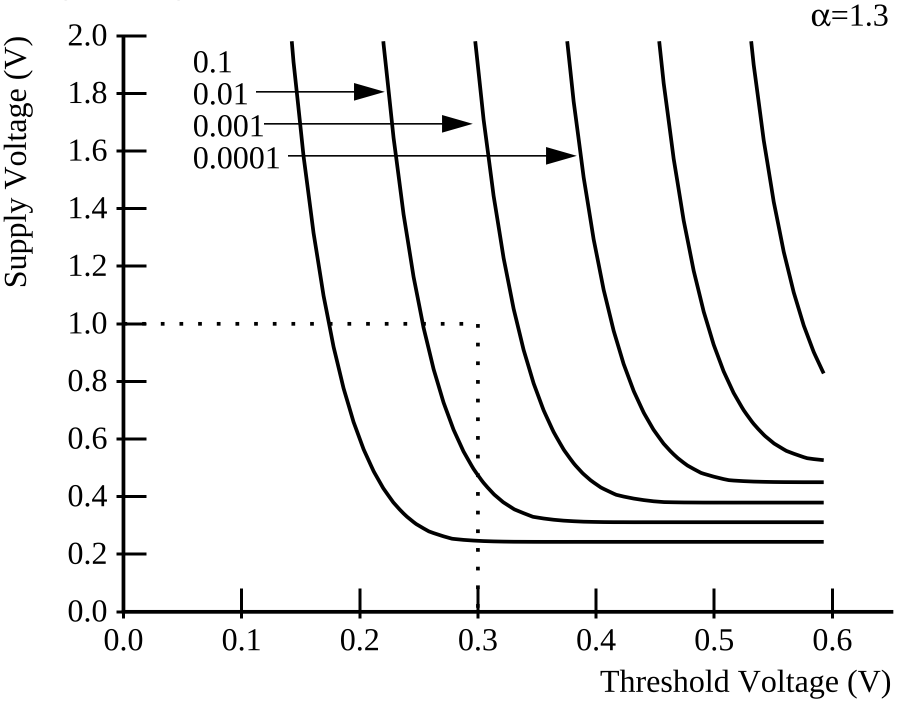


Figure 4.5: Ratio of leakage to total power.

In order to get around this constraint, the circuit would need a mechanism to change the effective V_{th} of its transistors. It would use the lower value during normal operation, and use the higher values during sleep modes. Some papers have proposed using high threshold power switch devices for this function [55], [56], [57] while others have proposed direct control of V_{th} [58], [59]. For some applications this is a good idea, even if the cost of switching from active to sleep mode is not negligible. This transition requires charging and discharging a very large capacitor—similar in magnitude to the total gate capacitance of the processor. This capacitance is either the well capacitance or the gate capacitance of the switch device. If the processor is in sleep mode only for a brief period of time then the energy cost of adjusting V_{th} can be larger than the energy saved due to lower leakage current. Figure 4.6 plots contours of time at which energy due to leakage equal the energy due to charging the source-to-bulk capacitance to adjust the threshold. The X-axis shows the original threshold voltage and the Y-axis shows the source-to-bulk potential. The change in threshold is modeled using the HSPICE level 39 V_{th} equation [60], which is reproduced below,

$$\Delta V_{th} = K_3\sqrt{\Phi + V_{sb}} - K_4(\Phi + V_{sb}) - V_{th} \qquad (4.18)$$

The values used for the constants are shown in Table 4.3. In Figure 4.6 each line represents an order of magnitude change in time. The bulk capacitance is assumed to be 10fF/μ of gate. The dashed and solid lines show contours for t=1ms for the cases when the capacitance is 20fF/μ and 2fF/μ respectively. If V_{th}=0.2V then reverse biasing the source-to-bulk junction by 800mv once a millisecond dissipates about the same energy as the leakage current during the same time. When the user is typing, for example, the processor must be active about once every ten millisecond. For this application it makes sense to adjust the threshold voltage.

Variable	Value
K_3	0.8555
K_4	155E-6
Φ	0.8

Table 4.3: Additional process parameters for 0.25μm technology.

While this adaptive threshold control also requires overhead in design time, area, and performance, it might be needed to deal with the more significant problem caused by process and operating point variations, which is described in the next section.

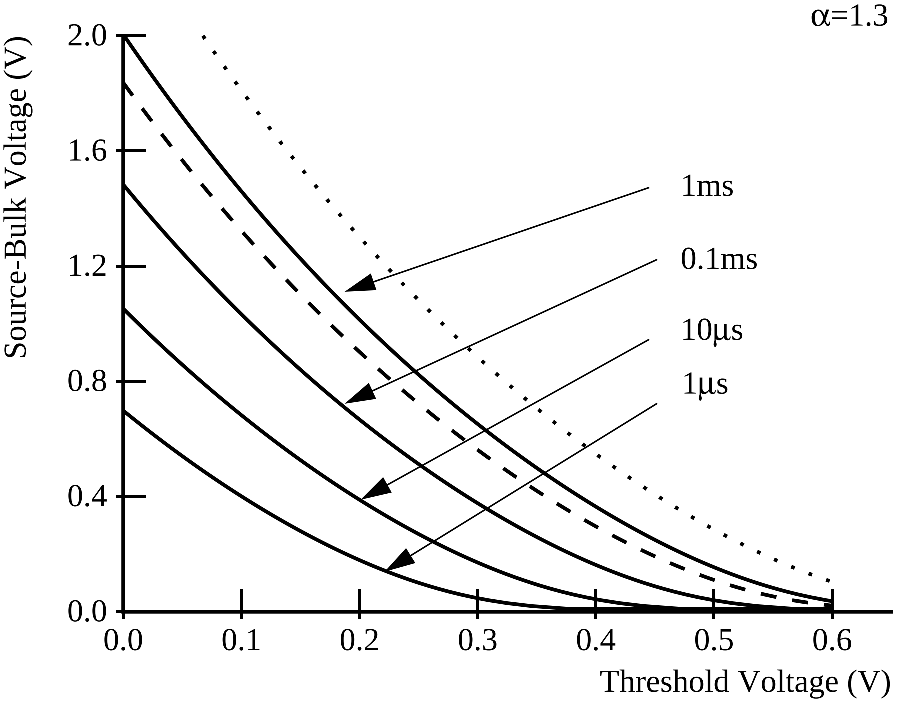


Figure 4.6: Minimum time for threshold adjustment.

4.3 Process and Operating Point Variations

The previous analysis assumed that the supply and threshold voltages were fixed, although in reality, both have small variations. In this section new energy-delay curves which take into account the effect of these variations are derived. It is first necessary to define the energy-delay product when variations are present. Then energy-delay curves for the same technology used in the previous section are derived. This section also presents the results from a real 0.25μm technology by using HSPICE to generate the power and delay numbers for the average inverter.

So far it has been assumed that the supply and threshold voltages can be controlled perfectly. For real processors there is always uncertainty in the value of V and V_{th}. There are two main sources of uncertainty. The first is that the processor must work over a range of operating conditions. The supply voltage is normally specified to be within 10% of a nominal value. The operating temperature can be anywhere between 25˚C and 100˚C. Since the threshold voltage changes by -0.8mV/˚C [60], the variation in T introduces approximately 60mV of uncertainty in V_{th}. The second source of uncertainty is the variability in V_{th} due to the manufacturing process. The threshold voltage of transistors within a chip, or a single transistor across chips, varies randomly. Variability due to manufacturing can be modeled by assuming V_{th} is a random variable with Gaussian probability density function (PDF) and a standard deviation of 35mV [61], [62], [63]. This gives a 3σ of about ±100mV.

Introducing uncertainty into V and V_{th} causes the delay and energy to be spread out over a range. The solid line in Figure 4.7 shows how the energy and delay of the processor vary as V_{th} varies. In this example supply voltage and temperature are fixed. Every design is specified to have a maximum energy and delay. That is, the processor is guaranteed not to dissipate more energy than the maximum, and guaranteed to meet or exceed the minimum performance. These limits correspond to vertical and horizontal dashed lines in Figure 4.7. The processor is guaranteed to lie somewhere below and to the left of these two lines. If V_{th} is a fixed quantity, then setting the processor specifications is easy. All parts will have the same energy and the same delay. In Figure 4.7 the star represents this point for some value of V_{th}. If V_{th} varies over a range then the specifications need to be relaxed, or else few parts will meet the targets. This corresponds to shifting the target lines up and right in Figure 4.7, as shown by the dotted lines. Reducing the area below and to the left of the target or cut-off lines improves the energy and delay specifications, but reduces the number of parts that meet the specifications. Where to draw the cut-off lines is somewhat arbitrary. The important point is that even though the parts will fall somewhere along the solid line, the processor will be sold as if it operated at the intersection of the energy and delay cut-off lines, marked with a triangle. Thus the energy-delay is defined as the product of the worst-case energy and the worst-case delay of the circuit, even though no part can have both the worst-case energy and worst-case delay simultaneously.

Since V_{th} is modeled as a normal random variable both energy and delay will follow some kind of distribution. One possible cut-off line is at the mean of the energy and the mean of delay. However, at this point a relatively small number of chips would meet both spec limits. The mean plus one standard deviation is used instead. In addition to the uncertainty due to V_{th} variations one must also account for the variations due to V and T. Therefore the equations are solved at four process corners (low V low T, high V low T, high V high T, low V high T). The mean and standard deviation of the energy and delay are found at each process corner. The EDP then is the product of the worst case energy at any of the four corners times the worst case delay at any of the four corners.

Figure 4.8 shows contours of the inverse of the relative EDP when there is uncertainty in V, V_{th}, and T. In this case the EDP surface is again relatively flat even though transistors

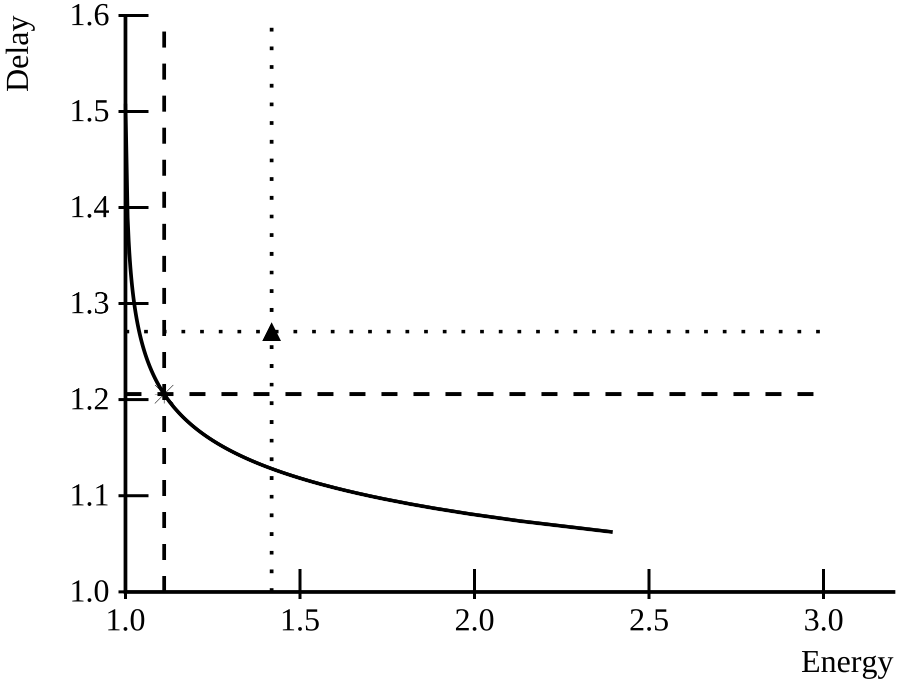


Figure 4.7: Variation in energy and delay.

are velocity saturated (α=1.3). The optimal operating point has moved to approximately *V*=0.5V and Vt=200mV. The surface becomes flat because at lower supply and threshold voltages the delay and the energy become more sensitive to variations in *V* and V_{th}. Thus operating at higher voltages is beneficial. The largest change in V_{th} is due to the temperature variation, which introduces approximately 60mV of uncertainty in V_{th}.

The effect of uncertainty then is to reduce the overall efficiency of the circuit. Although on average the circuit will operate at much higher efficiency, it can only be guaranteed to work for the worst-case conditions. The penalty for this worst-case operation becomes severe for low-voltage, low-threshold operating conditions. This is shown in Figure 4.9 which gives the ratio of the EDP without uncertainty to the EDP with uncertainty. At operating voltages above 1V, and V_{th} above 0.2V the effect of the variations is modest, less than 40% reduction. But the cost of the variations is around a factor of 4 at the previously found optimal point, *V*=250mV, V_{th}= 120mV. This high cost at low operating voltages is what flattens out the curves.

So far simple models have been used to approximate the energy and delay of the average CMOS inverter. Using this model allows one to understand how the EDP depends on the different circuit and process parameters, but the model's simplicity raises questions about its accuracy. In order to ensure that the models are correct, they were compared with the

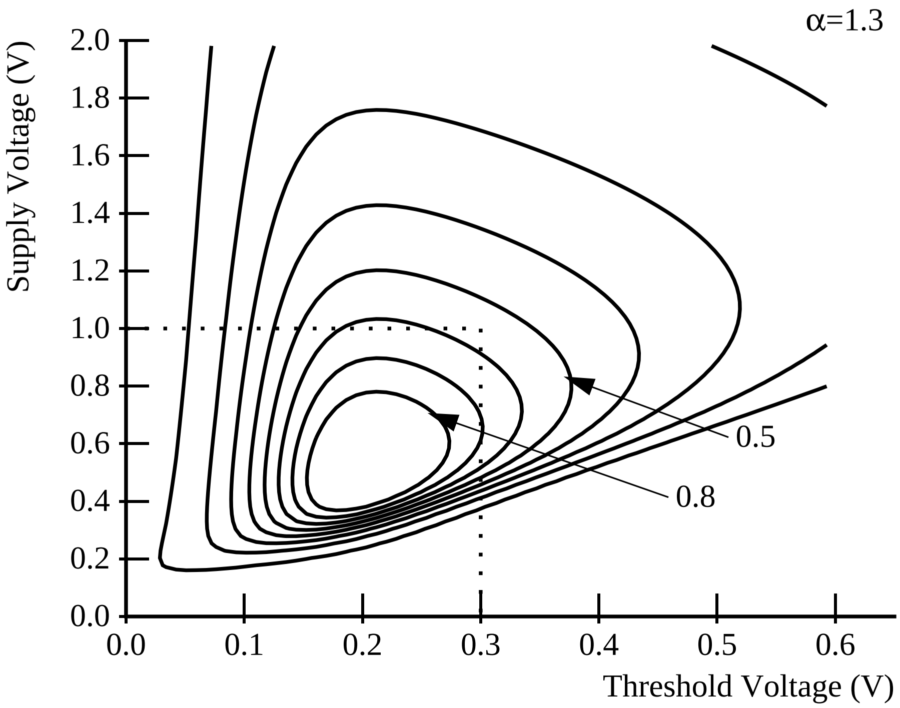


Figure 4.8: EDP contours with uncertainty.

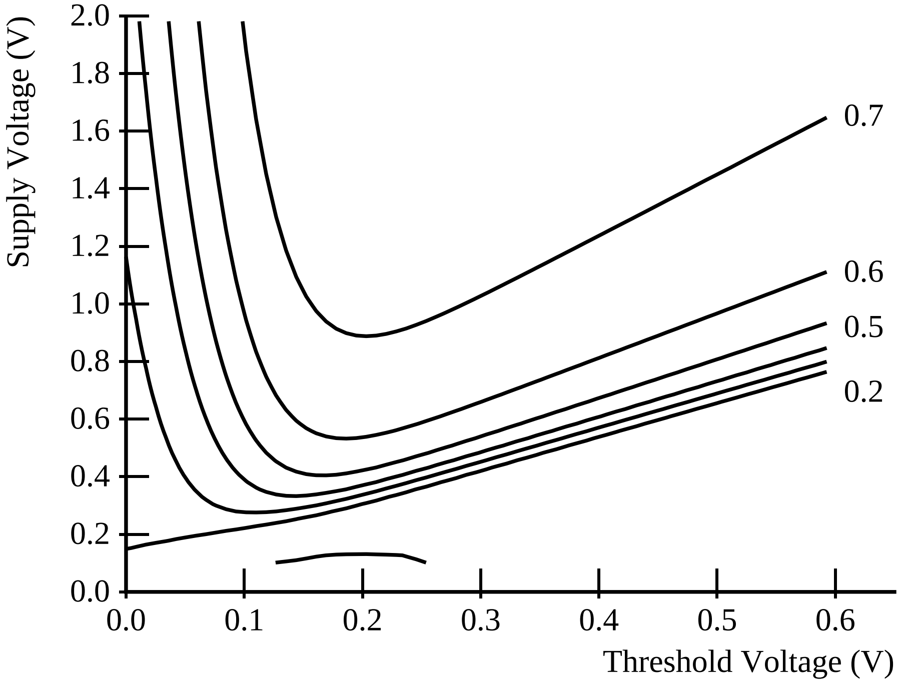


Figure 4.9: Ratio of EDP without and with uncertainty.

results from running HSPICE. Using HSPICE a chain of inverters was simulated to find the delay and energy per μm of gate width, as the supply and threshold voltage were swept. Level 37 models of a next generation 0.25µm process from Texas Instruments [64] were used for the simulations. The nominal threshold voltage is V_{th}=0.4V. V_{th} was adjusted by modifying the VT0 parameter of the HSPICE models. The results of the simulations are shown in Figure 4.10 The figure shows contours of the inverse of the relative EDP. The simulations consider variations in V and T.

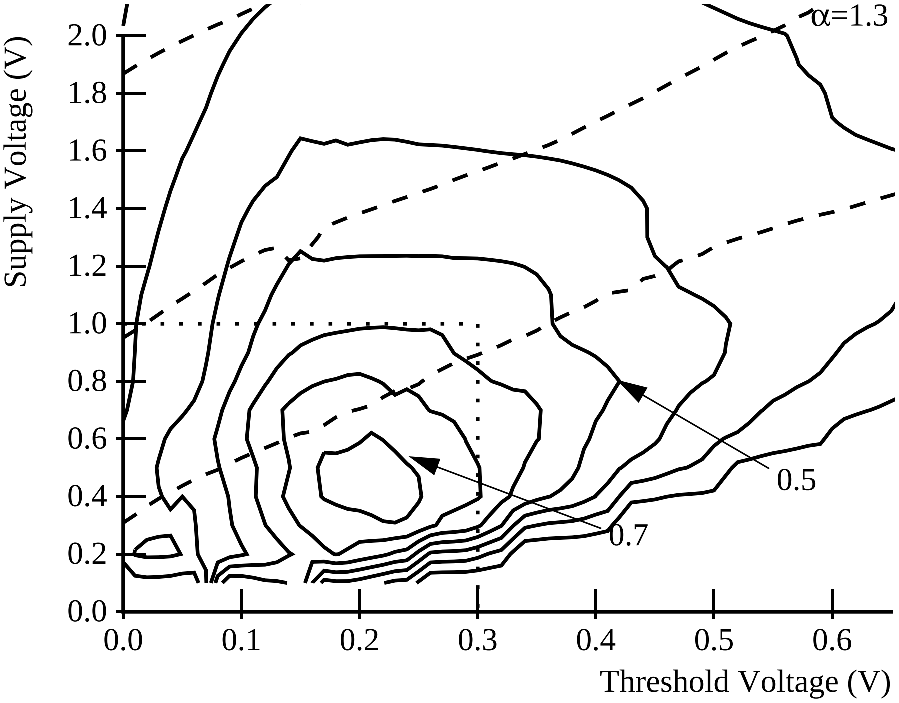


Figure 4.10: EDP contours using HSPICE models.

The contour lines are strikingly similar to previous figures. The EDP surface is marginally steeper (the first contour is at 0.7 not at 0.8 as in Figure 4.8). This is not surprising, however, since the HSPICE simulations do not consider uncertainty in V_{th}. One should note, however, that absolute values of V_{th} are hard to compare across processes, because there is no common definition of threshold voltage. The same chain of inverters was also simulated using BSIM2 models of the HP/MOSIS CMOS14B process. This is a 0.6µm process with nominal V_{th}=0.9V. V_{th} was adjusted by modifying the flat-band voltage parameter *VFB*. The most striking difference with the curves shown in Figure 4.8 is that the optimal operating point occurs at a much higher threshold voltage (0.5V vs. 0.2V). Most of this difference is a result of HSPICE using a different definition of V_{th}, which accounts for about 250mV of the shift. The remaining difference is because of drain-induced barrier lowering (DIBL) [8] which lowers the effective threshold voltage at higher supply voltages. Thus for high supply voltages the effective V_{th} is lower than shown in Figure 4.10.

The dashed lines in Figure 4.10 show contours of constant performance. Thus if a circuit is operating at the nominal point for this process (V=2.0V, V_{th}=0.4V) the range of EDP one can reach is limited. In order to approach the minimum one must be willing to give up a factor of 2X in gate performance, but this gives a factor of 6X reduction in the energy of the operation and 12X reduction in power.

Uncertainty in process and operating conditions incur a large cost in both performance and energy. Therefore the next section investigates how to improve the energy-delay of a processor by using adaptive techniques to reduce this uncertainty.

4.4 Adaptive Techniques

Adaptive techniques can be used to tighten the specifications therefore improving the energy-delay product. In Figure 4.7 this corresponds to being able to guarantee that the processor will be in a narrower region closer to the point marked with a star. Even if the processor specifications do not change, adaptive techniques can be used to, on average, trade excess performance for lower power. Although the processor must be designed to operate correctly under worst-case conditions, it rarely does so. On average the processor has excess performance which can be traded for lower energy. In Figure 4.7 this corresponds to moving along the curve towards the doted horizontal line. Although the performance of the processor is not better, on average the battery will last longer. This section first looks at the sources of uncertainty and then reviews some of the proposed methods for controlling supply and threshold voltages.

The three variables that introduce most uncertainty in energy and delay are supply voltage, threshold voltage, and carrier mobility (μ_c). Figure 4.11 shows how the energy and delay vary as a function of these three variables. Nominal parameters are V=0.5V, V_{th}=0.2V, and T=27˚C. The delay and energy of the circuits are plotted at three different temperatures T=0˚C, T=27˚C, and T=100˚C, shown with dotted, solid, and dashed lines respectively. The delay and energy are also plotted for three different supply voltages -10%, nominal, and +10%. Since V_{th} is a normal random variable at each operating corner 100 sample points are plotted assuming σ=35mV. Unfortunately a large portion of the variation in energy and delay is due to temperature changes and therefore cannot be compensated by adjusting the supply or threshold voltage. As the temperature increases the circuit requires both more energy and more time to operate correctly. The remaining uncertainty can be eliminated by adjusting either the supply, the threshold voltage, or both. The specifications for the processor, represented as the dotted horizontal and vertical lines in Figure 4.11, should be set as close to the origin as possible but must intersect all of the curves in Figure 4.11. At the point where the horizontal and vertical dotted lines meet the delay specification has improved by a factor of 1.7 and the energy specification by a factor of 2.4 compared to the worst case. This corresponds to a factor of 4 in EDP. This technique is only effective at low voltage operation. For large V and V_{th} uncertainty is dominated by temperature changes so controlling V and V_{th} will not significantly reduce the variation in energy and delay.

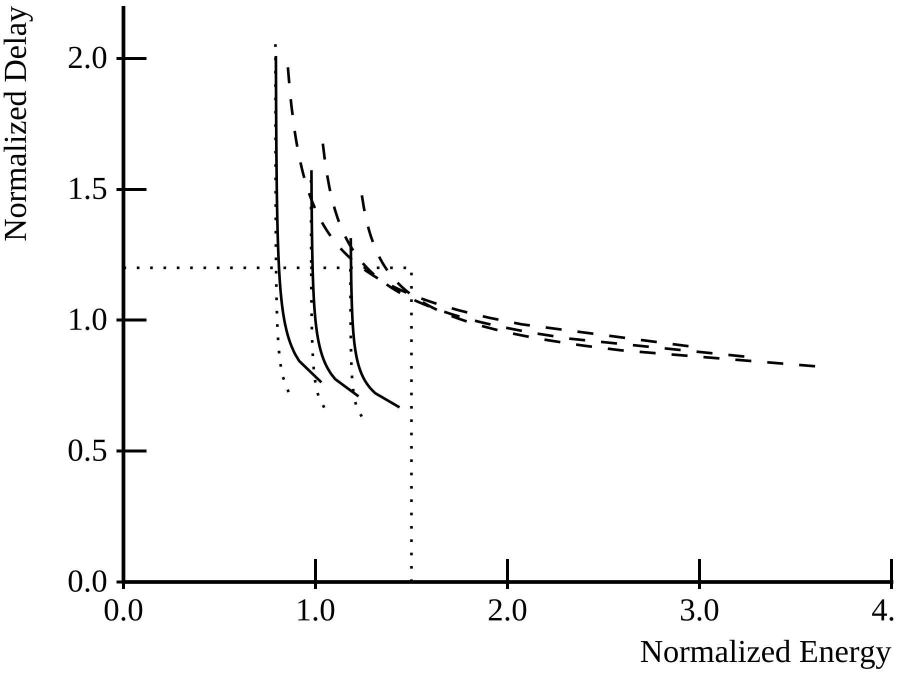


Figure 4.11: Energy and delay variations with operating conditions.

It is not necessary to change the specifications of the processor to reduce the energy dissipation. Adaptive techniques can be used to trade excess performance for lower power. If during operation the processor has more performance than specified then one can reduce its performance by reducing the supply voltage or increase the threshold voltage and save energy. From the nominal to the worst-case point in Figure 4.11 there is a factor of 2 in delay which can be traded for lower power. This corresponds to a voltage reduction from V=0.5V to V=0.4V—a factor of 1.5 in dynamic power. This technique gives lower benefits than tightening the specifications but is less risky to implement. If the feedback loop does not work correctly the processor will meet spec but dissipate higher average power. With the previous technique the processor could run slower than advertised. The rest of this section describes how to implement the circuitry to dynamically adjust the supply or threshold voltage.

Some papers have proposed measuring delay or performance in the feedback loop [65], [66], [67] while others have proposed measuring transistor current [68]. From a system viewpoint the first of the two is preferable since it guarantees that the processor is operating synchronously with the rest of the system[3]. Measuring the delay of the critical path is hard, but it can be approximated using the delay of a replica circuit, such as a ring oscillator, implemented on the same die as the processor. The delay of the replica circuit is matched to the period of an external reference signal by adjusting the control variable, either supply or threshold voltage. This replica circuit needs to operate slower than the processor for four reasons. First the delay of CMOS gates normalized to the delay of an inverter varies slightly over process and operating conditions (see Figure 4.1). This will require a margin in the order of 15% (±7.5%). Second some critical paths may be wire-*RC* limited. As the operating conditions change the delay of these paths will change relative to the delay of paths limited by gate loading. The margin needed depends on the operating range but will likely be in the order of 5%-10% (±5%). Third the power supply may suffer small “instantaneous” glitches due to *dI/dt* noise, also known as ground bounce. Feedback control cannot eliminate these glitches since they have a much smaller time constant than the feedback loop itself. In a well design system ground bounce is 10% of the supply voltage (±5%). Finally there may be a slight mismatch in the frequency comparison. In the implementations described later this is due to the ripple in the supply voltage introduced by the buck converter. This translates into a slow variation in processor speed over time. This variation is likely to be small (± 2.5%). Because of these four constraints the processor must operate a total of 20%-30% faster than absolutely necessary to ensure correct operation under all conditions.

One way to adjust the power supply is to use a buck converter [66], [67]. An important parameter in this architecture is the efficiency of the power supply since other costs are usually small. For example, the power of the controller, if implemented on the same die as the processor, should be small since it requires much less logic than the processor. Furthermore no additional circuitry is required to communicate with the outside world.

[3] For some circuits, such as dynamic nodes or memories, control of the leakage current relative to the wire, gate and diffusion capacitance is important.

Most modern processors incorporate this circuits anyway because the core normally runs at a lower voltage than the interface. The efficiency of the power supply is set mainly by the effective resistance of the buck converter (the resistance of the inductor and power transistors) as well as the energy dissipated in switching the power MOSFET's. The energy efficiency of the power MOSFET's is set by the technology (g_m/C_g and C_d) and cannot be changed by the designer. There is also a small cost to adjust the output voltage but this is probably negligible.

Adaptive control can increase the efficiency of the processor by either tightening the specifications or by trading excess performance for lower power. It was shown that since CMOS chips must be designed for worst-case conditions, which rarely arise, most often processors have excess performance. Tightening the processor specifications gives bigger gains in EDP, but requires more confidence in the circuit implementation. Furthermore the latter technique can be used to trade any excess performance. Even when the processor is active the required performance is a small fraction of the peak performance. If adaptive control is used then this excess performance can also be traded for lower power, as the next section explains.

4.5 Adaptive Power Management

Users regularly buy a processor system based on the peak performance specification. Some applications, such a 3D rendering, require this peak performance. But many other applications rarely, if at all, require so much performance. Very often when the user is interacting with the computer the required performance is very low. A fast typist, for example, presses a key once every ten milliseconds. The performance of the processor could be several orders of magnitude smaller and the user would not notice the difference. This is what we call excess performance. For a typical laptop system the processor is only active about 20% of the time. This section explores how adaptive control can be used to trade this excess performance for lower energy.

There are two basic ways in which the performance of the processor can be reduced. First the processor can run at full speed, perform all the computation it needs, and then shut itself off until the next event, such as a key press or mouse movement, wakes it up. The other alternative is to slow down the processor such that is just finishes processing event i when event i+1 arrives. This can be accomplished by either scaling the operating frequency leaving the supply voltage unchanged or scaling both the operating frequency and the supply voltage.

In a system without static power adjusting the supply voltage is the most efficient. Reducing the operating frequency with constant supply and shutting off the processor achieve the same power savings. Figure 4.12 shows the difference in power dissipation of a CMOS gate for the case where frequency is scaled with a fixed power supply, the case where supply voltage is adjusted to match the required frequency, and the difference in power between the two. All curves are normalized to the peak power and frequency.

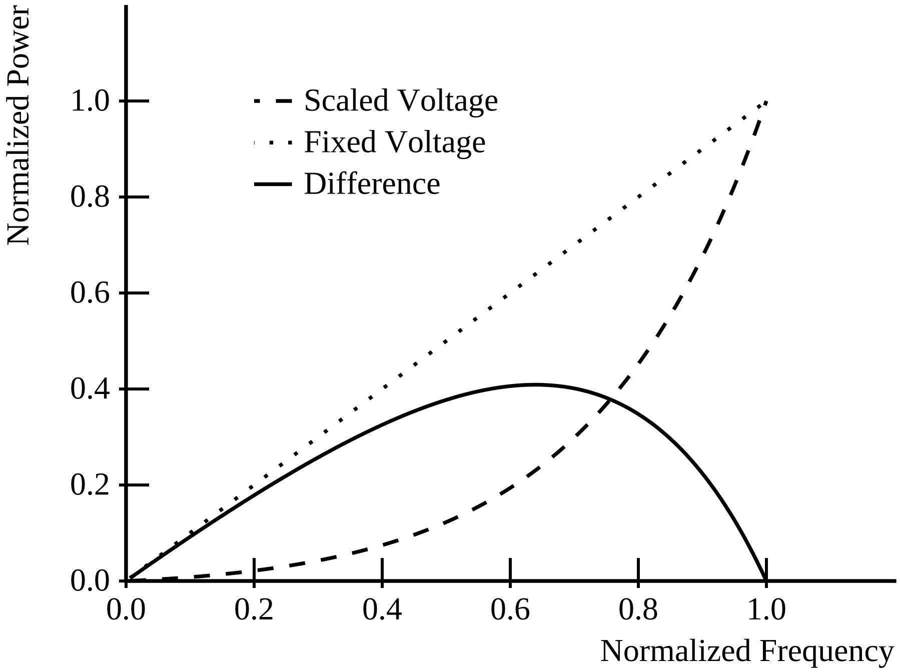


Figure 4.12: Power versus performance for fixed and variable supply.

In a system with significant static power reducing the speed of operation is not as effective. The previous section assumed static power could be controlled by adjusting the threshold voltage but some sources of static power do not depend on V_{th}. When static power remains constant regardless of the speed of operation, it makes more sense to operate as quickly as possible and then shut-off the system. Figure 4.13 shows the power breakdown for a Pentium 90 based laptop motherboard for 6 different operating modes [69]. The figure does not include the power dissipation in other components such as the LCD screen and the hard disk. Table 4.4 describes the operating modes. The power dissipation is broken up into 6 categories, which are described in Table 4.5. In this system it is not possible to scale the supply voltage of the second level cache (Cache), display controller (VGA), and other essential functions (Essential). Therefore the power dissipation of these components, which account for 20% of the total, remains constant regardless of the speed of operation. As mentioned earlier a processor is active about 20% of the time, or the processor requires about one fifth the performance. With an adaptive power supply processor power goes down by a factor of 50 and system power by a factor of 5. With frequency scaling processor power decreases by a factor of 5 and system power by a factor of 2.5. If the processor shuts off system power goes down also by a factor of 5. When the processor is idle for longer periods of time shutting it off is the most efficient alternative.

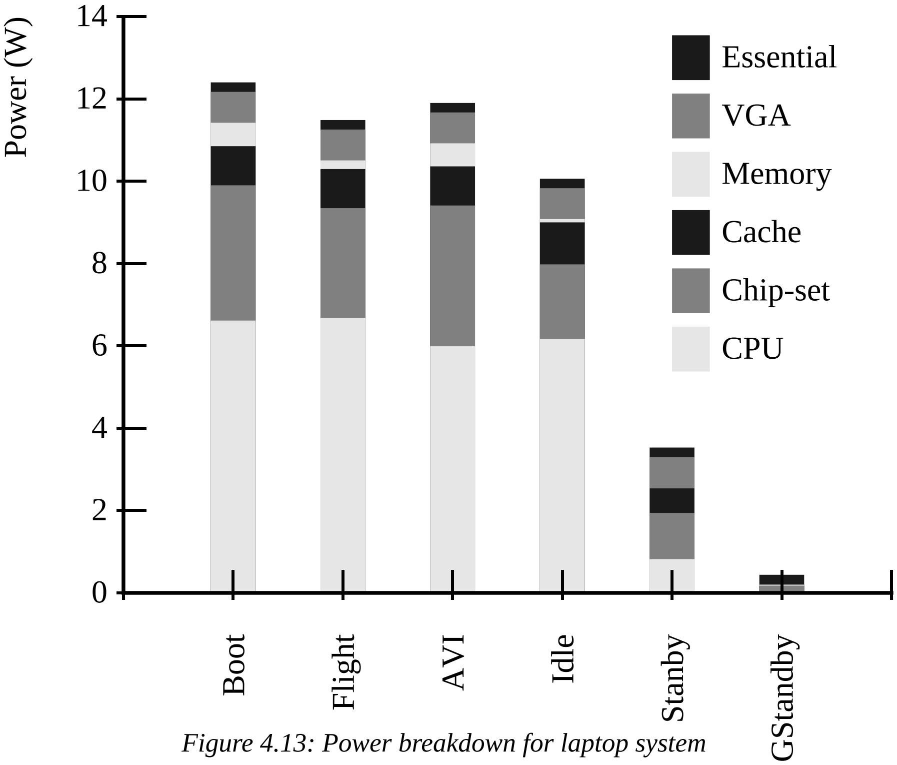


Figure 4.13: Power breakdown for laptop system

In this model reducing the power dissipation of the processor further gives little benefit because system power is limited by static power. One way to reduce system power is to also scale the supply voltage of other components, such as the second level cache. Scaling the voltage of the cache is possible because its frequency of operation depends on that of the processor. The display adapter, on the other hand, must operate at a fixed frequency and is therefore not amenable to supply voltage scaling. This optimization would make shutting off the processor only attractive at a utilization lower than 20%. The exact gains depend on how the shape of the delay versus power curves of the cache or other components whose supply voltage is being scaled.

Mode	Description
Boot	Booting Windows 3.11
Flight	Flight Simulator 4.0
AVI	Playing an AVI file
Idle	Windows 3.11 idle
Standby	Processor and interface chips enter sleep mode. Screen remains active, and L2 cache maintains state
GStandby	Complete system enters sleep mode. Only essential components and DRAM refresh remain

Table 4.4: Operating modes.

Component	Description
CPU	Processor power
Chip-set	Power of interface chips between CPU and system (PCI bus)
Cache	L2 Cache SRAM power
Memory	Main memory DRAM power
VGA	Power of display controller
Essential	Essential system functions (oscillator, real-time-clock, floppy disk).

Table 4.5: Power breakdown categories.

4.6 Summary

Chapter 3 showed a large fraction of the energy dissipation of processors is in intrinsic elements, clocks and memories. Others have looked at how to improve the efficiency of these circuit blocks by changing their architecture. This chapter explored instead how the energy, delay, and energy-delay vary as the supply and threshold change.

Using a first order model of energy and delay of CMOS circuits values of supply and threshold voltage which minimize the energy-delay were found. This operating point and the shape of the EDP surface near the minima are a strong function of how velocity saturated the transistors are. If transistors are not velocity saturated then the EDP surface is relatively flat. As transistors become more velocity saturated the EDP surface becomes steeper and the optimal point moves closer to the origin. For a 0.25µm technology, this analysis yielded a supply of 250mV and V_{th} of 120mV. One difficulty with operating at this point is that the speed of each gate is modest, forcing the designers to reduce the levels of logic in the processor to maintain performance. If this was the only issue, designers would use technologies with scaled thresholds and simply operate them at higher voltages (1V vs. 250mV) to recoup the lost gate speed. The main difficulty with these low voltage operating points is dealing with the variations caused by changes in operating conditions and threshold voltages.

While low operating voltages looked very attractive for low-power operation, they are very sensitive to both manufacturing variations and operating point changes. If one needs to provide margins in the processor to ensure it will meet certain speed and power requirements, the advantage of using technologies with very low threshold voltages disappears. When variations are considered the optimal point moves to higher voltages, and the whole energy-delay surface becomes flatter. Even when there is uncertainty, however, scaling the supply and threshold voltage can give significant gains in energy (9X) at some cost in delay (3X). In order to achieve the large potential gains of operating at low voltages, the circuit needs to use adaptive control to regulate the energy and delay.

Adaptive technique allow the designer to guarantee that the processor will be in a narrower performance and energy region. This resulted in a 4X gain in energy-delay. Adaptive techniques can also be used to trade excess performance for lower power. The source of excess performance can be either operating conditions better than worst-case or reduced performance requirements from the user. In the earlier case adaptive control gave a 1.5X gain in EDP. In the latter case adaptive control doubled battery life.

Chapter 5

Conclusions

Since the late 80’s higher performance has been the most important driving force behind processor evolution. For the past 10 or 15 years designers have doubled processor performance every 18 to 24 months. Unfortunately designers paid little or no attention to power. The result is large and growing power levels in processors. This thesis explored how some common design tradeoffs affect the energy and delay of a processor. Better understanding of these tradeoff is required if designers are to continue to increase performance but limit the power dissipation.

Unfortunately for processors energy and performance seem to be strongly related. Many techniques that reduce the energy dissipation also reduce the performance of the processor. There are several reasons for this correlation. The first is that for CMOS circuits common low-power optimizations simply trade energy and delay. Scaling the supply and the threshold voltage can reduce the energy dissipation by a factor of 9, but at the cost of slower basic elements. Second, because much of the power dissipation in processors is in basic operations, that all processors must perform, the internal organization of the processor does not have a large impact on the energy dissipation. This means that higher level optimizations, which have been highly effective in DSP applications, do not help. Exploiting parallelism does not improve the energy efficiency of a processor because of the limited parallelism available in typical programs run today. Only if the typical program mix changes will wider issue machines become attractive. Another high-level technique, rethinking the algorithm, can not be used by processor designers. Only the system programmer can change the algorithm.

There are a few techniques, however, which do decrease the energy for a given performance. One common feature of many these techniques is that they improve both the energy and the delay therefore increasing the efficiency at least by the square of the gains. The most promising of these techniques is to improve the technology. Scaling the dimensions of transistors and wires gives linear improvement in delay and cubic improvement in energy. Another effective technique is to exploit data locality. The farther the processor must go to fetch data the more energy and delay required. Large caches and register files can significantly increase the performance while at the same time slightly decreasing the energy requirements. Unfortunately most processors use a cache already, so future improvements are likely to be small.

Scaling the supply and threshold voltage can provide large gains in energy dissipation, at the cost of some performance lost. Another problem with low voltage operation is that the circuit is much more sensitive to variations in nominal values of V and V_{th}. Some of the efficiency gain due to scaling is lost because of this uncertainty. One way to limit the efficiency lost is to use adaptive techniques. A promising approach is to use adaptive control to trade excess performance, due to design margins, for lower power.

The excess performance need not be due only to design margins. The operating system should regulate the performance of the processor to reduce the power dissipation. This technique is attractive because the processor of a portable system is active only for about 20% of the time. Using an adaptive power supply the power dissipation of the processor could be reduced to 2% of the peak power.

Therefore over the next few years the efficiency of processors is likely to continue to improve. New more advanced technologies will allow the same computation to be performed with less energy. Shrinking the technology will also allow the same computation to be done faster, which will increase the power dissipation. Further scaling of the supply and threshold voltages will also increase the efficiency of processors, although at the expense of some performance. If designers are willing to give up a small factor in performance then large gains in energy are possible. Some of the efficiency lost due to uncertainty may be gained back as adaptive techniques to control the supply voltage become common. Simple pipelined machines are the most attractive because they represent a good tradeoff of performance and energy.

5.1 Future Work

This thesis presents several avenues for reducing the power dissipation of processors but does not describe a prototype. An excellent opportunity exists for incorporating the ideas presented to realize how small the power dissipation of the processor can be. This prototype should probably include an adaptive power supply and perhaps adaptive control of the threshold voltage. Further work is needed, however, to really understand the different sources of uncertainty and how they can be managed. Also incorporating adaptive control will answer the question of how much margin is needed to ensure correct operation in all conditions.

There are also many opportunities for future research investigating the energy efficiency of the whole system. The user buys a computer, not just a processor. In Chapter 4 it was shown the processor is only a small fraction of the total power. As the power dissipation of the processor decreases reducing the power dissipation of other components will become even more important. Some sources of power dissipation, for example the chip-set interfacing the processor to the system bus, can be attacked with techniques similar to those presented here. Other sources of power, for example the LCD controller and driver, may require novel techniques.

Finally, research is needed to understand how to dynamically manage the power dissipation of the whole system. Systems today waste a lot of energy because the system operates at a fixed performance. Ideally the operating system must determine what performance level is appropriate for the user and how much power should be dissipated in each of the different components. Under some conditions letting the interface chips run very slowly may not degrade system performance significantly but may allow lower power operation. Research is needed to understand not only how the operating system makes these decisions, but also how to build a system with dynamically varying characteristics.

Chapter 6

Bibliography

[1] S. Narayanaswamy, S. Seshan, E. Amir, et al., “A low-power, lightweight unit to provide ubiquitous information access application and network support for InfoPad”, *IEEE Personal Communications*, pp. 4–17, Apr. 1996.

[2] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power CMOS digital design”, *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–483, Apr. 1992.

[3] B. Gordon, *Low power video compression for portable applications using sub-band Coding*, PhD thesis, Stanford University, June 1996.

[4] W. J. Bowhill, R. L. Allmon, S. L Bell, et al., “A 300MHz 64b quad-issue CMOS RISC microprocessor”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1995, vol. 38, pp. 182–183.

[5] D. Linden, Ed., *Handbook of Batteries*, McGraw-Hill, New York, 2nd edition, 1995.

[6] K. M. Dixit, “New CPU benchmark suites from SPEC”, in *COMPCON Spring*, Feb. 1992, pp. 305–10.

[7] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, Reading, MA, 2nd edition, 1993.

[8] R. R. Troutman, “Drain-induced barrier lowering”, *IEEE Journal of Solid-State Circuits*, vol. 14, pp. 383, Apr. 1979.

[9] A. Chatterjee, M. Nandakumar, and I. Chen, “An investigation of the impact of technology scaling on power wasted as short current in low voltage CMOS”, in *International Symposium on low power electronics and design*. IEEE/ACM, Aug. 1996, pp. 145–150.

[10] T. Sakurai and A. R. Newton, “Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas”, *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–593, Apr. 1990.

[11] S. G. Younis and T. F. Knight, “Practical implementation of charge recovering asymptotically zero power CMOS”, in *Integrated Systems,* 545 Technology Square, Cambridge Massachusetts, MA 02139, Oct. 1993, M.I.T. Artificial Intelligence Laboratory, pp. 234–250, MIT Press.

[12] J. Koller and W. Athas, “Adiabatic switching, low energy computing, and the physics of storing and erasing information”, in *Proceedings of Physics of Computation Workshop*, Oct. 1992.

[13] T. Indermaur and M. Horowitz, “Evaluation of charge recovery circuits and adiabatic switching for low power CMOS design”, in *Symposium on Low Power Electronics*. IEEE, Oct. 1994, vol. 1, pp. 102–103.

[14] R. Dennard et al., “Design of ion implanted MOSFET’s with very small dimensions”, in *IEEE Journal of Solid-State Circuits*. IEEE, 1974, vol. 9, pp. 256–267.

[15] Z. Chen, J. Shott, J. Burr, et al., “CMOS technology scaling for low voltage low power applications”, in *IEEE International Symposium on Low Power Electronics*. ACM/IEEE, Oct. 1994, vol. 1, pp. 56–7.

[16] B. S. Amrutur and M. Horowitz, “Techniques to reduce power in fast wide memories”, in *IEEE International Symposium on Low Power Electronics*. IEEE, Oct. 1994, vol. 1, pp. 92–93.

[17] N. K. Yeung, Y.-H. Sutu, T. Yi-Feng-Su, et al., “The design of a 55 SPECInt92 RISC processor under 2W”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1994, vol. 37, pp. 206–207.

[18] Integrated Device Technology, Inc., *Orion Product Overview*, Mar. 1994.

[19] D. Pham, M. Alexander, A. Anzpe, et al., “A 3.0W 75 SPECInt92 85 SPECfp92 superscalar RISC microprocessor”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1994, vol. 37, pp. 212–213.

[20] J. Montanaro, R. T. Witek, et al., “A 160MHz 32b 0.5W CMOS RISC microprocessor”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1996, vol. 39, pp. 215–215.

[21] B. Gordon and T. H.-Y. Meng, “A low power subband decoder”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb 1995, vol. 38.

[22] E. K. Tsern, *Pyramid Vector Quantization*, PhD thesis, Stanford, June 1996.

[23] A. Chandrakasan, A. Burstein, and R. W. Brodersen, “A low power chipset for portable multimedia applications”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1994, vol. 37, pp. 82,83.

[24] N. P. Jouppi and D. Wall, “Available instruction-level parallelism for superscalar and superpipelined machines”, in *International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM/IEEE, Apr. 1989, pp. 272–282.

[25] M. D. Smith, M. Johnson, and M. Horowitz, “Limits on multiple instruction issue”, in *International Symposium on Computer Architecture*, Boston, MA, Apr. 1989, ACM/IEEE, pp. 290–302.

[26] D. Wall, “Limits of instruction-level parallelism”, in *International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, Ca, Apr. 1991, ACM/IEEE, pp. 176–188.

[27] M. Butler, T.-Y. Patt, M. Alsup, H. Scales, and M. Shebanow, “Single instruction issue stream parallelism is greater than two”, in *International Symposium on Computer Architecture*, Toronto, Canada, May 1991, ACM/IEEE, pp. 276–286.

[28] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, CA, first edition, 1990.

[29] M. D. Smith, *Support for Speculative Execution in High-Performance Processors*, PhD thesis, Stanford University, Nov. 1992.

[30] R. Bechade, R. Flaker, B. Kauffman, et al., “A 32b 66MHz 1.8W microprocessor”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1994, vol. 37, pp. 208–209.

[31] T. Biggs, J. Dunning, R. Eisele, et al., “A 1Watt 68040-compatible microprocessor”, in *IEEE International Symposium on Low Power Electronics*. ACM/IEEE, Oct. 1994, vol. 1, pp. 12–13.

[32] G. Kane, *MIPS RISC Architecture*, Prentice Hall, Engelwood Cliffs, NJ 07632, 1988.

[33] S. Przybylski, *Performance-directed memory hierarchy design*, PhD thesis, Stanford University, Sept. 1988.

[34] M. D. Smith, “Tracing with Pixie”, Tech. Rep. CSL-TR-91-497, Stanford University, Nov. 1991.

[35] S. J. E. Wilton and N. P. Jouppi, “An enhanced access and cycle time model for on-chip caches”, Research Report 93/5, DEC Western Research Lab, 250 University Ave., Palo Alto, Ca, 94301, June 1994.

[36] J. A. Gasbarro, “The Rambus memory system”, in *International Workshop on Memory Technology, Design and Testing*. IEEE, Aug. 1995, pp. 94–96.

[37] A. Agarwal and S. D. Pudar, “Column-associative caches: A technique for reducing the miss rate of direct-mapped caches”, in *International Symposium on Computer Architecture*, San Diego, Ca, May 1993, ACM/IEEE, vol. 21, pp. 179–90.

[38] N. P. Jouppi, P. Boyle, J. Dion, M. J. Doherty, et al., “A 300MHz 115W 32b bipolar ECL microprocessor with on-chip caches”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1993, vol. 36, pp. 84–85.

[39] K. Wilson and K. Olukotun, “High performance cache architectures to support dynamic superscalar processors”, Tech. Rep. CSL-TR-95-682, Stanford, June 1995.

[40] A. Ahi, Y.-C. Chen, R. Conrad, et al., “R10000 superscalar processor”, in *Hot Chips VII*. IEEE, Aug. 1995, pp. 227–238.

[41] K. I. Farkas, N. P. Jouppi, and P. Chow, “How useful are non-blocking loads, stream buffers, and speculative execution in multiple issue processors?”, Research Report 94.8, DEC Western Research Lab, 250 University Ave., Palo Alto, Ca, 94301, Dec. 1994.

[42] T. A. Conte, “Tradeoffs in processor/memory interfaces for superscalar processors”, in *International Symposium on Microarchitecture*, Portland, Or., 1992.

[43] S. Palacharla, N. P. Jouppi, and J. E. Smith, “Quantifying the complexity of superscalar processors”, Tech. Rep. CS-TR-96-1328, University of Wisconsin, Madison, Nov. 1996.

[44] W. M. Johnson, *Super-Scalar Processor Design*, PhD thesis, Stanford University, June 1989.

[45] R. P. Colwell and R. L. Steck, “A 0.6mum BiCMOS processor with dynamic execution”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1995, vol. 38, pp. 176–177.

[46] N. B. Gaddis, J. R. Butler, A. Kumar, and W. J. Queen, “A 56-entry instruction reorder buffer”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1994, vol. 39, pp. 212–213.

[47] N. Gaddis and J. Lutz, “A 64-b quad-issue CMOS RISC microprocessor”, *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1697–1702, Nov. 1996.

[48] W. F. Ellis, E. Adler, and H. L. Kalter, “A 2.5-V 16-Mb DRAM in 0.5-µm CMOS technology”, in *IEEE International Symposium on Low Power Electronics*. ACM/IEEE, Oct. 1994, vol. 1, pp. 88–9.

[49] K. Itoh, K. Sasaki, and Y. Nakagome, “Trends in low-power RAM circuit technologies”, in *IEEE International Symposium on Low Power Electronics*. ACM/IEEE, Oct. 1994, vol. 1, pp. 84–7.

[50] U. Ko and P. T. Balsara, “High performance, energy efficient master-slave flip-flop circuits”, in *IEEE International Symposium on Low Power Electronics*. ACM/IEEE, Oct. 1995, vol. 2, pp. 16–17.

[51] J. B. Burr, “Cryogenic ultra low power CMOS”, in *IEEE International Symposium on Low Power Electronics*. IEEE, Oct. 1995, p. 9.4.

[52] J. B. Burr and A. M. Peterson, “Energy considerations in multichip-module based multiprocessors”, in *IEEE International Conference on Computer Design*, 1991, pp. 593–600.

[53] D. Dobberphul, “Personal communication”, Personal E-mail.

[54] V. Kaenel, D. Aebischer, C. Piquet, and E. Dijkstra, “A 320MHz 1.5mW at 1.35V CMOS PLL for microprocessor clock generation”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1996, vol. 39, pp. 132–133.

[55] M. Horiguchi, T. Sakata, and K. Itoh, “Switched-source-impedance CMOS circuit for low standby subthreshold current giga-scale LSI’s”, in *VLSI Circuits Symposium*, May 1993, pp. 47–8.

[56] D. Takashima, S. Watanabe, H. Nakano, et al., “Standby/active mode logic for sub-1-V operating ULSI memory”, *IEEE Journal of Solid-State Circuits*, pp. 441–7, Apr. 1994.

[57] S. Mutoh, S. Shigematsu, Y. Matsuya, et al., “A 1V multi-threshold Voltage CMOS DSP with an efficient power management technique for mobile phone applications”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1996, vol. 39, pp. 168,169.

[58] T. Kobayashi and T. Sakurai, “Self-adjusting threshold voltage scheme SATS”, in *Custom Integrated Circuits Conference*. IEEE, May 1994, pp. 271–274.

[59] K. Seta, H. Hara, T. Kuroda, et al., “50% active-power saving without speed degradation using standby power reduction (SPR) circuit”, in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1995, vol. 38, pp. 318,319.

[60] Meta-Software, Inc., *HSPICE User's Manual*, Feb. 1996.

[61] A. J. Strojwas, M. Quarantelli, J. Borel, et al., "Manufacturability of low power CMOS technology solutions", in *International Symposium on low power electronics and design*. IEEE/ACM, Aug. 1996, pp. 225–232.

[62] X. Tang, V. K. De, and J. D. Meindle, "Effects of random MOSFET parameter fluctuations on total power consumption", in *International Symposium on low power electronics and design*. IEEE/ACM, Aug. 1996, pp. 233–237.

[63] M. Eisele, J. Berthold, D. Schmitt-Landsiedel, et al., "The impact of intra-die device parameter variations on path delays and the design for yield of low voltage digital circuits", in *International Symposium on low power electronics and design*. IEEE/ACM, Aug. 1996, pp. 237–242.

[64] M. Nandakumar, A. Chatterjee, M. Rodder, et al., "A device design study of 0.25µm gate length CMOS for 1V low power applications", in *IEEE International Symposium on Low Power Electronics*. ACM/IEEE, Oct. 1995, vol. 2, pp. 80–1.

[65] L. S. Nielsen, C. Niessen, J. Sparso, et al., "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage", *IEEE Transactions on Very Large Scale Integration*, pp. 391–7, Dec. 1994.

[66] G.-Y. Wei and M. Horowitz, "A low power switching power supply for self-clocked systems", in *International Symposium on low power electronics and design*. IEEE/ACM, Aug. 1996, pp. 313–317.

[67] W. Namgoong, M. Yu, and T. Meng, "A high efficiency variable-voltage CMOS dynamic DC-DC switching regulator", in *IEEE International Solid-State Circuits Conference*. IEEE, Feb. 1997, vol. 40, pp. 380–81.

[68] V. R. Kaenel, M. D. Pardoen, E. Dijkstra, and E. A. Vittoz, "Automatic adjustment of threshold and supply voltages for minimum power consumption in CMOS digital circuits", in *IEEE International Symposium on Low Power Electronics*. IEEE, Oct. 1994, pp. 78,79.

[69] “Intel 430MX PCIset power measurement analysis”, Tech. Rep., Intel, Chandler, Az, Jul. 1996.

[70] B. W. Kernighan and S. Lin, “An efficient heuristics procedure for partitioning graphs”, *Bell Systems Technical Journal*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[71] S. Sangiovanni-Vincentelli, *Design Systems for VLSI circuits*, chapter Automatic layout of integrated circuits, pp. 113–195, Martinus Nijhoff, 1987.

[72] M. Pedram and B. Preas, “Interconnection length estimation for optimized standard cell layouts”, in *IEEE International Conference on Computer-Aided Design*, Nov. 1989, pp. 390–3.

Appendix A

Capacitance Estimation

In order to estimate the energy dissipation of the Verilog models it was necessary to develop CAD tools which estimate the total capacitance of each net. The capacitance of a node has three components, diffusion capacitance—area and perimeter capacitance of the drain, gate capacitance—input capacitance of the next gate, and interconnect capacitance—capacitance of the wires between gates. Estimating the diffusion and gate capacitance is easy. It can be found from the size and type of the gates attached to the net. The wire capacitance, on the other hand, depends on where the cells are placed and how they are interconnected together. Since wire capacitance is assumed to be constant per unit length (0.2fF/μm), the problem reduces to estimating how long each net is.

Wire length estimation is done in two steps. First wire lengths are estimated for each block, one at a time. Then, using a manually generated floor-plan, the length of global nets is estimated. The total length of any net is the sum of these two components. A different algorithm is used to compute wire lengths within a block depending on the block type. There are three kinds of blocks, standard cell blocks, datapath blocks, and macro-cell blocks.

Estimating the wire capacitance in macro-cells, like SRAM's, is easy because the structure is so regular. A few basic cells are replicated in a geometric pattern, so by estimating the wire length in one cell a very good estimate of the total wire length can be found. Estimating the wire capacitance of datapath blocks is slightly more complicated. Datapath blocks are highly regular but the ordering of the different cells is not specified in the Verilog model. The wire estimation tool first uses the Kernighan-Lin [70], [71] graph partitioning heuristic to determine a pseudo-optimal placing of the cells. The program tries to find the cell placement with the least number of tracks. It then estimates the length of control wires based on the number of bits in the datapath and the bit-pitch. The bit-pitch is set by how many tracks are required to connect all the cells. The wire estimation tool then finds the length of data wires based on the connectivity and the width (or height) of the cells.

To estimate the length of wires in standard cell blocks an algorithm given by Pedram was used [72]. The algorithm first computes the neighborhood population of each net (how many cells are two hops away) and then translates this number into a wire length based on the number and average size of the cells in the control section.

Appendix B

Memory Power Model

Amrutur [16] developed a low-power memory that uses hierarchical word lines, pulsed word line to limit bit line swing, and a small swing local bus. Based on that implementation he developed a power model for memories which is describe in this appendix.

In order to reduce power dissipation the storage array is broken-up into banks or blocks, only one of which is active per access. A simple representation of the memory is shown in Figure B.1. The decoders select one row to be read or written and drive a global word line. A block select signal is and-ed with the global word line to generate a local word line. The contents of the addressed row are driven on the bit lines and then sensed by the sense amplifiers at the bottom of the block. From there the data is driven onto the I/O bus.

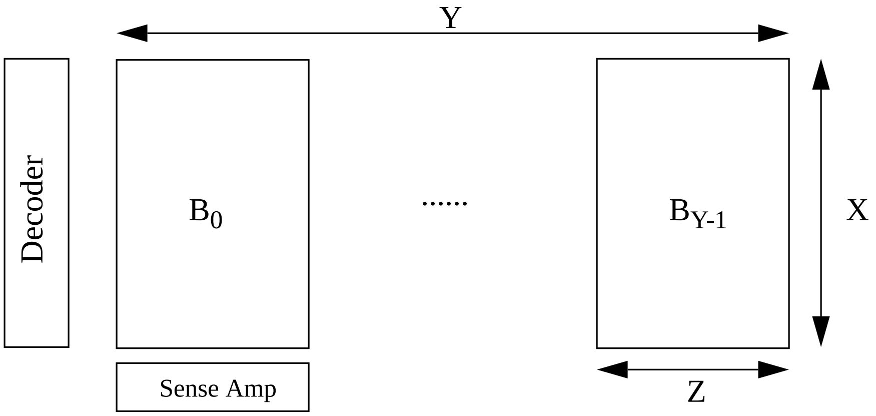


Figure B.1: Cache power model.

Amrutur characterized the power dissipation of different components in a memory by three parameters, the number of rows (X), the number of banks (Y), and the number of columns in a bank (Z). He found that most of the power dissipation was due to the wire capacitance. For a given cell size the length of the wires is proportional to the three parameters given. For example, the length of the bit lines depends on the number rows. The power to pre-charge the bit lines also depends on how many bit lines there are. Given the dimensions X, Y and Z, the power consumed to access the array is as follows,

$$P = K_x X + K_y Y + K_z Z + K_{xz} XZ + K_{yz} YZ + K_{yzz} YZ^2 + K_{\ln(xy)} \ln(XY) \qquad (6.2)$$

The power coefficients for Amrutur’s memory running at 100MHz are given in Table B.1. Where two values are given the first is for reads the second for writes. The bit lines swing is larger during writes and therefore writes dissipate more power.

Parameter	Description	Value (W)
K_x	X decoder, block select signal	3.035e-4
K_y	global sense signal, write enable signal	3.125e-4
K_z	local word line, memory cells, local sense amps, global sense amps, write drivers	6.297e-4/9.344e-4
K_{xz}	pre-charge bit lines, swing bit lines	2.075e-6/1.001e-5
K_{yz}	Y decoder, global word line	4.941e-6
K_{yzz}	pre-charge local bus, swing local bus	1.465e-6
$K_{\ln(xy)}$	address buffer	2.885e-3

Table B.1 Cache model parameters.

For a 16KByte cache, where X=256, Y=8, and Z=64, the total power dissipation is 227mW for reads and 377mW for writes.