

# Energy Dissipation In General Purpose Microprocessors

Ricardo Gonzalez and Mark Horowitz

**Abstract**—In this paper we investigate possible ways to improve the energy efficiency of a general purpose microprocessor. We show that the energy of a processor depends on its performance, so we chose the energy-delay product to compare different processors. To improve the energy-delay product we explore methods of reducing energy consumption that do not lead to performance loss (i.e., wasted energy), and explore methods to reduce delay by exploiting instruction level parallelism. We found that careful design reduced the energy dissipation by almost 25%. Pipelining can give approximately a 2× improvement in energy-delay product. Superscalar issue, however, does not improve the energy-delay product any further since the overhead required offsets the gains in performance. Further improvements will be hard to come by since a large fraction of the energy (50–80%) is dissipated in the clock network and the on-chip memories. Thus, the efficiency of processors will depend more on the technology being used and the algorithm chosen by the programmer than the micro-architecture.

## I. INTRODUCTION

THE interest in lowering the power of a processor has grown dramatically over the past few years. This interest in power dissipation is fueled partly by the high power levels (greater than 50 W [4]) of today's state-of-the-art processors, as well as the growing market for portable computation devices. The result of this attention to power is an increasing diverse space of processors that a user can choose between, from simple machines with modest performance and power consumption under one Watt to processors with sophisticated architectural features and power dissipation in the tens of Watts. Yet for processors, power and performance seem to be strongly correlated. Low-power invariably means lower performance. This correlation is different from some published results for signal processing applications, where power has been reduced by three orders of magnitude while maintaining the same performance [6], [9]. This paper will show that the reason for the slow progress in processors is fundamental—barring a radical new architecture, the energy-delay product of a processor will be roughly set by the energy-delay product of the underlying technology.

The next section describes how the power and delay of the underlying CMOS technology used for processors are related, and shows how it is easy to trade increased delay for reduced energy. Thus the energy needed to complete a program (in

Jules/instruction or its inverse SPEC/W) is not a good metric to compare two designs. Instead, one needs to consider both energy and delay simultaneously, which is the reason we use the energy-delay product (in Jules/SPEC or its inverse SPEC<sup>2</sup>/W) in the rest of this paper.

Section III then looks at a lower bound on the energy and energy-delay product for a processor by investigating a number of ideal machines, including an unpipelined machine, a simple RISC machine, and a superscalar machine. In these machines only the energy of data storage is accounted for, which includes essential latches, cache accesses, and register file accesses. Since the control and computation are free, these machines form a lower bound on what can be achieved in any real machine.

Then, Section IV looks at two real processors that we have designed, a simple RISC machine and a superscalar processor called TORCH. Starting from an unoptimized design, conventional optimizations can be used to save about 25% of the machine power. After these optimizations, we show that the real machines are within a factor of two from the ideal machine given in the previous section. Since this power is distributed in a number of small units, reducing the overhead will be fairly difficult.

## II. ENERGY-DELAY PRODUCT

Until recently, performance was the single most important feature of a microprocessor. Today, however, designers have become more concerned with the power dissipation, and in some cases low power is one of the key design goals. This has led to an increasing diversity in the processors available. Comparing processors across this wide spectrum is difficult, and we need to have a suitable metric for energy efficiency. Table I shows several possible metrics for a few of the processors available today [4], [5], [8], [12], [15], and [19]. Power is not a good metric to compare these processors since it is proportional to the clock frequency. By simply reducing the clock speed we can reduce the power dissipated in any of these processors. While the power decreases, the processor does not really become “better.”

Another possible metric is energy, measured in Jules/Instruction or its inverse SPEC/W. While better than power, this metric also has problems. It is proportional to  $cv^2$  so one can reduce the energy per instruction by reducing the supply voltage or decreasing the capacitance—with smaller transistors. Both of these changes increase the delay of the circuits, so we would expect the lowest energy processor to also have very low performance, as shown in Table I. Since

Manuscript received June 21, 1995; revised March 22, 1996. This research was supported by the Advanced Research Projects Agency under contract J-FBI-92-194.

The authors are with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA.

Publisher Item Identifier S 0018-9200(96)06477-3.

TABLE I  
CURRENT MICROPROCESSORS

	Dec 21164	UltraSPARC	P6	R4600	R4200	Power 603
SPECint92	346.00	250.00	200.00	110.00	55.00	75.00
SPECfp92	506.00	360.00	150.00	84.00	30.00	85.00
Average	426.00	305.00	175.00	97.00	42.50	80.00
Frequency	300.00	167.00	133.00	150.00	80.00	80.00
Power	50.00	30.00	15.00	5.500	1.80	3.00
SPEC/W	8.51	10.17	8.75	17.64	25.00	26.67
SPEC <sup>2</sup> /W	3621.01	3100.83	2041.70	1710.33	1003.47	2133.33
L <sub>min</sub>	0.50	0.45	0.60	0.64	0.64	0.50
SPEC <sup>2</sup> /Wλ <sup>2</sup>	4470.38	3100.83	3629.69	3459.51	2029.73	2633.74

we usually want minimum power at a given performance level, or more performance for the same power, we need to consider both quantities simultaneously. The simplest way to do so is by taking the product of energy and delay (in Jules/SPEC or its inverse SPEC<sup>2</sup>/W). Although there is a very wide distribution in both performance and power among processors—as shown in Table I—in terms of energy-delay product, all processor are remarkably close to one another.

To improve the energy-delay product of a processor we must either increase its performance or reduce its energy dissipation without adversely affecting the other quantity. Many of the common low-power design techniques do not reduce the energy-delay product, they simply allow the designer to trade-off performance for energy [11]. By reducing the supply voltage, for example, one can reduce the energy and the performance by almost one order of magnitude with only small changes in the energy-delay product.

An effective way to reduce the energy-delay product is to shrink the technology. If the scaling factor is  $\lambda$ , under ideal scaling conditions [9] the energy-delay product scales with  $\lambda^4$ . However, most technologies do not scale ideally since the threshold voltage often does not scale [7]. Moreover, the performance of a processor is also limited by the external memory system. So we would expect the energy-delay product to scale somewhere between  $\lambda^2$  and  $\lambda^3$ . In Table I we also show the energy-delay product scaled by  $\lambda^2$ , under the row labeled “SPEC<sup>2</sup>/Wλ<sup>2</sup>.” The spread between high-performance and low-power processors becomes even narrower. Thus, the energy efficiency of a processor is highly dependent on the efficiency of the underlying technology. As technology scales, the energy delay product of processors will continue to improve. Since the improvement should be approximately constant for all processors, we will ignore technology scaling for the remainder of this paper.

Once technology scaling has been factored in, the energy-delay of the processors shown in Table I are within a factor of two from each other, while the performance and energy vary by more than an order of magnitude. To understand why processors are so similar, the next section investigates what are the intrinsic factors that set the energy and energy-delay product for a processor. This simplified model indicates that

architectural changes like superscalar issue have only a modest effect on a processor’s energy-delay product.

### III. LOWER 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, and therefore we cannot determine if the reductions accomplished represent a large fraction of the potential improvements. In this section we will investigate what these bounds are by looking at three idealized machines: an unpipelined processor, a simple pipelined RISC processor, and a superscalar processor.

All processors fundamentally perform the same operations. 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.<sup>1</sup> 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, and there is considerable logic that 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, we will neglect it. 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 we will aggressively assume that the energy cost of performing computation is zero, and we will also assume that communication costs within the datapath are also zero. We will only consider the energy needed to read and write memories, and, where required, the energy to clock storage elements, such

<sup>1</sup> Program semantics require that instruction  $i$  be executed before instruction  $i + 1$ . However, if the second instruction does not depend upon the first, they can both be executed simultaneously. We call this instruction level parallelism.

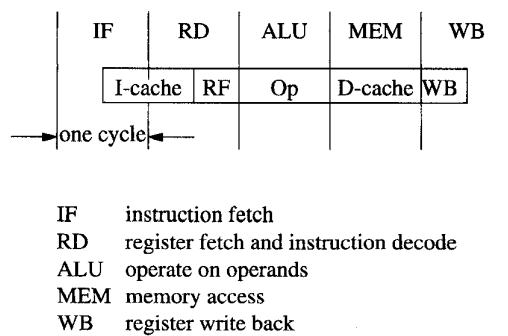


Fig. 1. Pipeline diagram.

as pipeline latches. Furthermore, these machines only dissipate energy that is absolutely necessary. Most processors perform some operations speculatively, in the hope that the result is useful. Our ideal machines have perfect knowledge so there is never a need for speculative operations. And finally, there are no unwanted transitions, such as glitches. We use a trace-based simulation to estimate the energy and delay of these idealized machines. For a more detailed description of our simulation methodology please refer to the Appendix.

#### A. Machine Models

The simple machine is the most basic compute element, similar to DLX [10]. The processor consists of a simple state machine that fetches an instruction, fetches the operands, performs the operation, and stores the result. The processor is not pipelined so we model the execution time by assuming control transfer instructions (CTI) take two cycles, ALU operations take three cycles, and cache operations take four cycles. Since this machine is not pipelined, we assume 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 we created a pipelined processor that is similar to the previous one, except that we use pipelining to exploit instruction level parallelism (ILP). 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. We use the traditional MIPS or DLX [10] five-stage pipeline shown in Fig. 1. Since it is not possible to build a pipelined machine without some kind of storage element, in addition to the energy required to read and write memories, we also take into account the energy required to clock the latches in the pipeline and the PC chain.

The ideal superscalar processor is similar to the pipelined machine, except that it can execute a maximum of two instructions per cycle. Since we are not concerned with the energy dissipated in the control logic, how the instruction level parallelism is found is only significant in that it sets the amount of parallelism exploited. We use an aggressive static scheduler from TORCH [16] which gives comparable performance to a dynamic scheduled machine. Since most

integer programs do not have enough parallelism to fill both issue slots regularly, we assume that 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 execute two instructions in parallel.

#### B. Comparison of Energy and Energy-Delay Product

Fig. 2 shows the total energy required to complete a benchmark. All numbers in this section have been normalized to the corresponding value for the superscalar processor. As we would expect, the unpipelined machine has the lowest energy dissipation, and the superscalar machine has the highest. The execution time, shown in Fig. 3, is reversed. The superscalar processor has the highest performance, while the unpipelined machine has the lowest performance. In the figure we have explicitly shown the time spent stalled due to the memory system. Fig. 4 shows the energy-delay product. Pipelining provides a big boost in performance for very little energy cost, so it gives almost a  $2\times$  improvement in energy-delay product. Superscalar issue, on the other hand, only gives a small improvement in energy-delay product. The performance gain is small and the energy cost is higher. In this simple model, a processor with a wider parallel issue would be of limited utility. The basic problem is that the amount of instruction level parallelism (in the integer benchmarks that we use) is limited,<sup>2</sup> 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 superscalar processor is limited by Amdahl's law. 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 we will show in the following section, the overhead in a superscalar machine is actually worse than that of a simple machine, and overwhelms the modest gain in energy-delay product gotten from parallelism.

## IV. CURRENT IMPLEMENTATIONS

To see how real machines compare with the ideal machines described in the previous section, this section looks at two processors that we have designed, a simple RISC machine and a superscalar processor we call TORCH. In the three idealized machines presented earlier we focused only on the essential operations and ignored all sources of overhead. For the real processors we will include all sources of overhead. This simulation method is also described in the Appendix.

As has been shown before [2], [3], [15], and [19], some simple optimizations can yield significant gain in energy-dissipation. With careful design it is possible to reduce the

<sup>2</sup>There have been several reports on the amount of ILP available in typical integer programs [13], [16]–[18]. These studies have found that for realistic machines the ILP is severely limited and in most cases smaller than two. Even for very aggressive machines the average number of instructions issued per cycles is less than two.

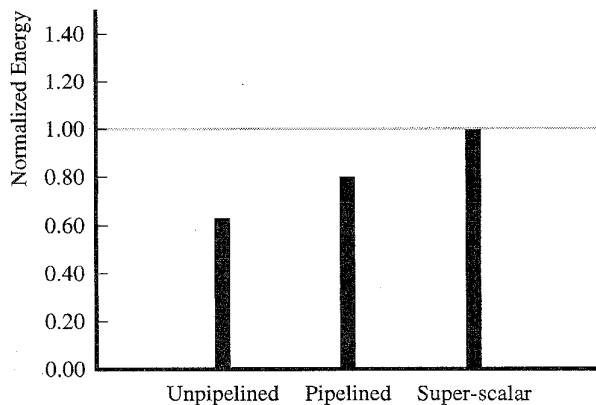


Fig. 2. Normalized energy of ideal machines.

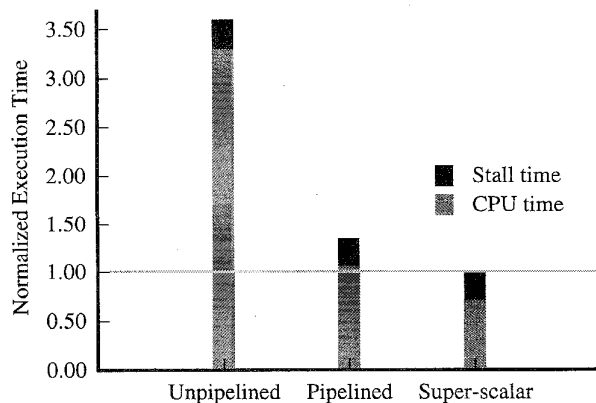


Fig. 3. Normalized execution time of ideal machines.

energy waste considerably without much change to the critical path. We will briefly discuss some of the optimizations used in our designs and what improvements we observed. We then compare the results from the optimized real machines to the ideal machines of the previous section.

#### A. Machine Models

The simple RISC processor is similar to the original MIPS R3000 described by Kane [14], except that it includes on-chip caches. The processor has the same five-stage pipeline that was shown in Fig. 1. 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 co-processor 0. The difference in energy-delay product between this processor and the pipelined ideal machine represents a limit on possible improvements in efficiency if we only focus on the logic required to execute an instruction.

TORCH [16] is a statically scheduled two-way superscalar processor, which uses the same five-stage pipeline that was shown in Fig. 1. 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

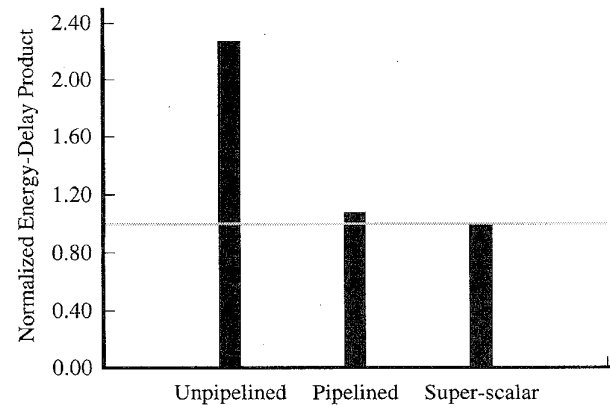


Fig. 4. Normalized energy-delay product of ideal machines.

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-b 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.

We use TORCH as an example because of its smaller overhead compared to that of an aggressive dynamically scheduled processor. Due to a larger overhead, such a processor would have a higher energy per instruction but equivalent performance [16], and thus have a higher energy-delay product.

#### B. Energy Optimizations

The additional energy associated with a particular architectural feature can be divided into overhead and waste. We consider overhead 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 cost. 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. Following are a few techniques that we used to reduce waste in our implementations.

Clock gating can be used to eliminate transitions that should never have happened. In our implementation we qualify latches in the datapath when the instruction does not produce a result. 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 we can eliminate these spurious transitions. We also qualify all latches that are not in the main execution datapath. These latches tend to be

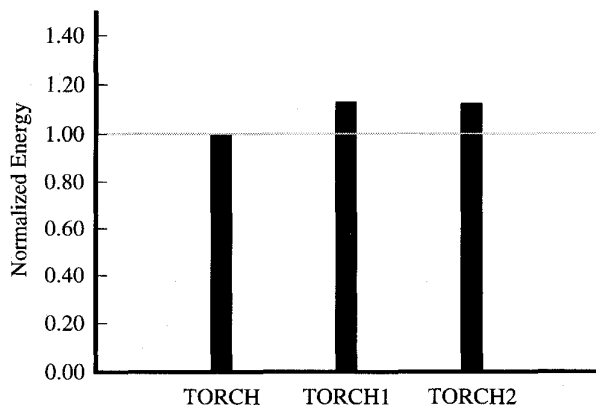


Fig. 5. Reduction in energy from simple optimizations.

enabled only infrequently. 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 we do not 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. We do qualify some control signals 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.

Through the use of clock gating we 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 we would need 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.

We use selective activation of the large macrocells 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 reread the cache. By doing so we saved approximately 8% of the total power.

Speculative operations are commonly used to improve the performance of microprocessors. 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.

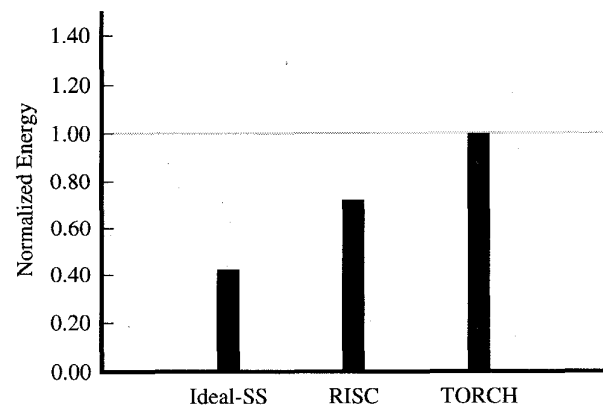


Fig. 6. Normalized energy for ideal superscalar, simple RISC, and TORCH processors.

The designers of the DEC 21 164 chip used a similar idea [4]. The second level cache is 96 kB three-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 four to two, 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 PC is usually 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, we perform both additions in parallel, while the outcome of the branch is determined, and then use a multiplexer to select one of the two results. The energy we save by eliminating the extra addition is negligible, while the cycle time penalty is large. Thus, designers should carefully consider the trade-off when eliminating speculative operations.

Fig. 5 shows the energy saved using the simple optimizations presented in this section. TORCH refers to the fully optimized processor. In TORCH1 we have disabled selective activation of the instruction cache and always speculatively read the register file. In TORCH2 we have removed all qualification from the clocks. We normalized to the energy of TORCH. Thus, simple optimizations can save almost 20–25% of the total power. However, further gains would be harder to come by because—as we will show later—the energy is dissipated in a variety of units, none of which accounts for a significant fraction of the total energy.

### C. Comparison to Ideal Machines

Fig. 6 shows the energy for the ideal superscalar machine, the simple RISC machine, and TORCH. All figures in this section 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 potential future improvements. Since we

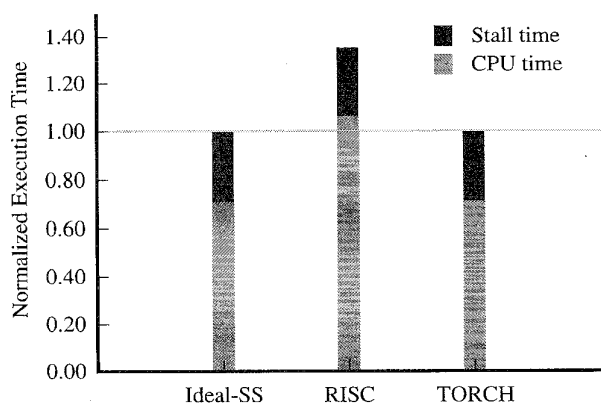


Fig. 7. Normalized execution time for ideal super-scalar, simple RISC, and TORCH processors.

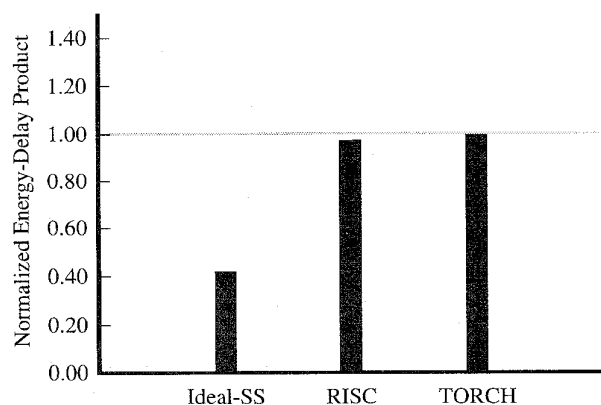


Fig. 8. Normalized energy-delay product for ideal super-scalar, simple RISC, and TORCH processors.

were very aggressive in our simulations, we would expect the gains to be even smaller than this.

Fig. 7 shows the total execution time for the same three machines. Since TORCH and the ideal superscalar machine have identical execution models, they have the same execution time. The superscalar processors have a  $1.35\times$  speedup when compared to the single issue machine. With an ideal memory system, the speedup would have been  $1.6\times$ . This highlights the importance of the external memory system, even in low-power applications.

Finally, Fig. 8 shows the energy-delay product for all three machines. As expected from the two previous figures, 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.

The ideal super-scalar processor is roughly twice as efficient as the real machine. Since we want to understand how to make TORCH approach the ideal processor, we look at where the extra energy is dissipated.

#### D. Energy Breakdown

We chose to divide the energy dissipation into four categories. The first is energy dissipated in reading and writing

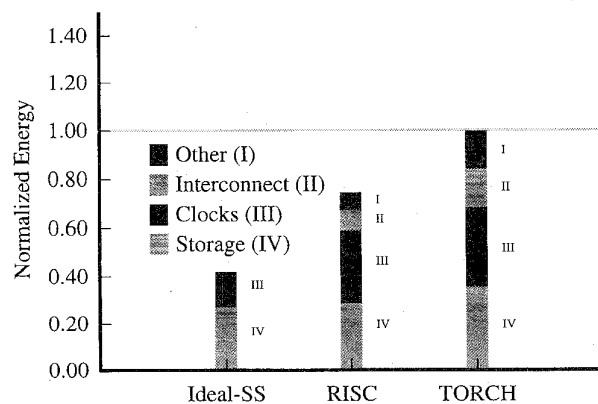


Fig. 9. Energy breakdown for ideal superscalar, simple RISC, and TORCH processors.

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. The final category comprises everything not in one of the previous categories.

Fig. 9 shows the energy breakdown for three of the five machines we 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 as in our implementations. This means that we were successful in reducing the number of unnecessary cache accesses. The clock power in the real machines is slightly higher since there is a large amount of state that we did not consider 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.

#### V. CONCLUSION

We have shown that the performance and energy of a processor are related. Over a wide spectrum of performance and energy, processors seem to have quite similar energy-delay products. We have shown there are two main reasons for this. First, all processors perform similar functions. Two of the most important operations, accessing on-chip memories and clocking, account for a large (50–80%) fraction of the total energy dissipation. Thus, doubling the energy dissipated in the combinational logic will only slightly change the overall energy dissipation. Second, only pipelining provides a large boost in performance for a low cost in energy. Other architectural features do not seem to significantly change the energy-delay product. Given that most processors today are pipelined, we would expect all of them to have a similar energy-delay product.

Also, although it is possible to reduce the energy requirements by careful design, this will only provide a one-time improvement. We have shown that 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.

#### APPENDIX SIMULATION METHODOLOGY

There are two sources of energy dissipation in CMOS circuits, static energy which results from resistive paths from the power supply to ground, and dynamic energy, which results from switching capacitive loads between two voltage levels. Static energy dissipation exists whenever the chip is powered on and usually represents a small fraction of the total energy dissipated, unless the chip is idle for large periods of time. Ignoring static energy dissipation, we can estimate the energy dissipation as

$$E = \sum_i \frac{n_i}{2} c_i V^2 \quad (1)$$

where  $n_i$  is the number of times the  $i$ th node toggles, and  $c_i$  is the capacitive load of that node. Thus, to compute the energy dissipation, we need to accurately estimate the capacitance and the toggle count for each node.

Estimating the energy dissipation of some macrocell blocks, like SRAMS, is easier because most of the energy in these blocks is dissipated in pre-charge/discharge differential structures. Since one of the two signals in a differential pair is guaranteed to discharge on every access, the energy dissipated is independent of the actual data values read or written. Thus, we can approximate the energy dissipated in the SRAMS by simply counting the number of read accesses and the number of write accesses and then multiplying by the energy dissipated per access [1].

To estimate the energy dissipation of the idealized machines, we use a simple trace-based method. In these machines all the energy is dissipated in the memories or the storage elements, so it is only necessary to compute the toggle count and capacitive load of the clock. Given a trace of all of the instructions executed, we can compute the number of accesses to each memory and the total energy dissipated. We find the clock energy by determining the capacitance of a single latch and multiplying by the minimum number of latches to maintain the processor state.

To estimate the energy dissipation of the simple RISC machine and TORCH, we must accurately determine the toggle count and the capacitance for each of the nodes. We have developed a complete Verilog model of TORCH which includes the main datapath, instruction and data caches, co-processor 0, TLB, and external interface; although, it does not include a floating point unit. The model can stall, take exceptions, or take external interrupts. The model can also run in MIPS mode in which case it behaves like the simple RISC machine. We use this model to estimate both the toggle counts and the capacitance of each node.

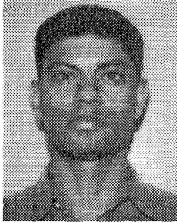
To determine the toggle count we run a simulated program on the Verilog model. We have written several routines which

use Verilog's programming language interface (PLI) to gather this information. The PLI routines also traverse the network to determine the total capacitance of each node. The gate and diffusion capacitance is annotated in each of our library cells. We have written several programs which allow us to estimate the wire capacitance for both standard cell and datapath blocks. We estimate the length of global wires by generating a floorplan of the chip and then extracting the layout.

Since the performance and the energy dissipation of the processor depend on the test being run, we decided to use *compress* and *jpeg*, from the SPEC benchmark suite. The first is representative of traditional integer programs normally run on a processor. The second program is more representative of multimedia type applications. The results for both applications are so similar, both in terms of performance and energy, that we show the results for *compress only*. We compile the benchmarks using the TORCH optimizing C compiler or the MIPS C compiler, depending on the target machine.

#### REFERENCES

- [1] B. S. Amrutur and M. Horowitz, "Techniques to reduce power in fast wide memories," in *Symp. Low Power Electr.*, IEEE, Oct. 1994, pp. 92-93.
- [2] R. Bechade *et al.*, "A 32 b 66 MHz 1.8 W microprocessor," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1994, pp. 208-209.
- [3] T. Biggs *et al.*, "A 1 Watt 68040-compatible microprocessor," in *Symp. Low Power Electr.*, IEEE Solid-State Circuits Council, Oct. 1994, vol. 1, pp. 12-13.
- [4] W. J. Bowhill *et al.*, "A 300 MHz 64 b quad-issue CMOS RISC microprocessor," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1995, pp. 182-183.
- [5] A. Chamas *et al.*, "A 64 b microprocessor with multimedia support," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1995, pp. 178-179.
- [6] A. P. Chandrakasan, S. S. Brodersen, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473-483, Apr. 1992.
- [7] Z. Chen, J. Shott, J. Burr, and J. D. Plummer, "CMOS technology scaling for low voltage low power applications," in *Symp. Low Power Electr.*, Oct. 1994, pp. 56-57.
- [8] R. P. Colwell and R. L. Steck, "A 0.6  $\mu$ m BiCMOS processor with dynamic execution," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1995, pp. 176-177.
- [9] B. M. Gordon and T. H.-Y. Meng, "A low power subband video decoder architecture," in *Int. Conf. Acoustics, Speech and Signal Processing*, Apr. 1994, pp. 409-412.
- [10] J. L. Hennessy and D. A. Patterson, *Computer Architecture A Quantitative Approach*, 1st ed. San Mateo, CA: Morgan Kaufmann, 1990.
- [11] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *Symp. Low Power Electr.*, Oct. 1994, pp. 8-11.
- [12] Integrated Device Technology, Inc., *Orion Product Overview*, Mar. 1994.
- [13] N. P. Jouppi and D. Wall, "Available instruction-level parallelism for superscalar and superpipelined machines," in *Int. Conf. Architect. Sup. Programming Language and Operating Systems*, Apr. 1989, pp. 272-282.
- [14] G. Kane, *MIPS RISC Architecture*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
- [15] D. Pham *et al.*, "A 3.0 W 75SPECint92 85SPECfp92 superscalar RISC microprocessor," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1994, pp. 212-213.
- [16] M. D. Smith, "Support for speculative execution in high-performance processors," Ph.D. thesis, Stanford University, Stanford, CA, Nov. 1992.
- [17] M. D. Smith, M. Johnson, and M. Horowitz, "Limits on multiple instruction issue," in *Int. Symp. Computer Architecture*, Boston, MA, Apr. 1989, pp. 290-302.
- [18] D. Wall, "Limits of instruction-level parallelism," in *Int. Conf. Architect. Sup. Programming Language and Operating Systems*, Santa Clara, CA, Apr. 1991, pp. 176-188.
- [19] N. K. Yeung *et al.*, "The design of a 55SPECint92 RISC processor under 2 W," in *IEEE Int. Solid-State Circuits Conf.*, Feb. 1994, pp. 206-207.



**Ricardo Gonzalez** received the B.S. and M.S. in electrical engineering from Stanford University, Stanford, CA, in 1990 and 1992, respectively. He is currently working towards the Ph.D. in the same field at Stanford University. His research interests are in the area of processor architecture and low-power circuits.



**Mark Horowitz** received the B.S. and M.S. in electrical engineering from MIT, Cambridge, MA in 1978 and the Ph.D. in the same field from Stanford University, Stanford, CA in 1984.

Since September 1984, he has been working in the Computer Systems Laboratory at Stanford where he is currently an Associate Professor in electrical engineering. His research area is in digital system design. He has led a number of processor design projects at Stanford including MIPS-X, one of the first processors to include an on-chip instruction cache, and TORCH, a statistically-scheduled, superscalar processor. He has also worked in a number of other chip design areas including high-speed memory design, high-bandwidth interfaces, and fast floating point. In 1990 he took leave from Stanford to help start Rambus, Inc., a company designing high-bandwidth memory interface technology. His current research includes multiprocessor design, low-power circuits, memory design, and processor architecture.

Dr. Horowitz is the recipient of a 1985 Presidential Young Investigator Award, an IBM Faculty development award, as well as the 1993 Best Paper Award at the 1994 International Solid-State Circuits Conference.