Appears in Work in Progress Session, HPCA-6, Toulouse, France, January 2000

Energy-Exposed Instruction Set Architectures

Krste Asanović
MIT Laboratory for Computer Science, Cambridge, MA 02139
krste@mit.edu

1 Introduction

Power consumption is emerging as a key factor limiting computational performance in both mobile and tethered systems. Although there has been significant progress in low-power circuit design and low-power CAD and some work in low-power microarchitectures, there has been little work to date at the level of instruction set architecture (ISA) design for low power computing.

Modern ISAs such as RISC or VLIW are based on extensive research into the effects of instruction set design on performance, and provide a purely *performance-oriented* hardware-software interface. These instruction sets avoid features that would impede a high-performance implementation. They also avoid providing alternate ways to perform the same task unless it will increase performance significantly. Implementations of these ISAs perform many energy-consuming microarchitectural operations during execution of each user level instruction and these dominate total power dissipation. For example, when executing an integer add instruction on a simple RISC processor only 1/50th of the total energy consumption is due to the adder circuitry itself. The rest is dissipated by cache tag and data accesses, TLBs, register files, pipeline registers, and pipeline control logic. Modern machine pipelines have been refined to the point where most of the additional microarchitectural work is performed in a pipelined or parallel manner that does not affect the throughput or user-visible latency of a "simple" add instruction. Because their performance effects can be hidden, there is no incentive to expose these constituent micro-operations in a purely performance-oriented hardware-software interface — *their energy consumption is hidden from software.*

This short paper reports on ongoing work in the SCALE (Software-Controlled Architectures for Low-Energy) project at MIT, where we are developing new *energy-exposed* hardware-software interfaces that give software fine-grain control over energy consumption. The key idea is to reward compile-time analysis with run-time energy savings. Our designs provide software with alternative methods of executing an operation, possibly with the same performance, but where greater compile-time knowledge can be used to deactivate unnecessary portions of the machine microarchitecture.

We are initially focusing on integer applications with complex control flow as we believe this type of code will become the energy bottleneck in future embedded systems. Other highly regular and/or parallel computations can be readily mapped to energy-efficient computing structures such as vector/SIMD units [1], reconfigurable units [5], or application-specific coprocessors [3]. The following section gives examples of the techniques we are currently exploring. All simulation numbers we report are for a MIPS R3000-like processor running gcc-compiled SPECint95 binaries. Section 3 discusses the technology we are developing for more sophisticated processor energy accounting, and Section 4 concludes.

2 Example Energy-Exposed ISA Techniques

Tag-Unchecked Loads and Stores: In some cases, static analysis can determine that a memory access will always hit in the cache. For these cases, we provide versions of load and store instructions that do not check cache tags. Our initial design only allows a tag-unchecked load/store to the same cache line as the previous load/store executed. Apart from reducing energy from tag matching (which dominates total energy for associative caches), we do not need to generate a full memory address but only sufficient low-order bits to index within one cache line. A single control bit remembers any interrupts since the last memory access, in which case a tag-unchecked instruction is executed as a regular tag-checked access.

Register File Hierarchy: Fetching register operands consumes a large portion of processor energy. We are investigating a four-level register file hierarchy as a way to reduce this energy. The highest level is the bypass network. Our simulations show that 25–40% of regfile reads can be avoided because values will be supplied by the bypass network. The second level exposes the ALU input latches to software. This creates a hybrid accumulator-RISC architecture that allows software to avoid regfile writes in the frequent case that a value's lifetime only extends from one

instruction to the next. The third level is obtained by splitting the register file into the most popular 8 registers and the remaining 23 real registers (which form the last level in the hierarchy). Our simulations show that these 8 most popular registers account for 80–90% of all regfile accesses in SPECint95. This split is implemented with a transmission gate on the bitlines which adds slightly to the access time and energy for the fourth level but halves access energy for the third level. The bitline split is invisible to existing software, but a smart register allocator can try to keep frequently accessed values in the low-energy registers.

Energy-Reduced Instruction Alternatives: We have found that a high-energy instruction is often used to perform a trivial task because it is fast or convenient. As an example, a fast shifter consumes considerable energy per operation because it drives a large collection of wires stretching across the datapath [4]. Yet, NOPs are encoded as shift instructions in the MIPS instruction set and comprise around 4% of dynamic instructions in our SPECint95 traces. Around 3% of all dynamic instructions are shift left by one, accounting for over half of all left shifts. Also, around a third of right shifts use a shift by 31 to extract the sign bit. In a purely performance-oriented ISA there is no incentive to provide alternative forms because a fast shifter can complete in a single cycle, but in an energy-conscious ISA we might consider alternatives that perform these frequent cases with less energy than a full shift.

Explicit Exception State Management: Current pipelined machines invest significant energy in preserving precise exception semantics. Instruction results are buffered before being committed in order, requiring register rename logic to find the correct value for new instructions. Even a simple five stage MIPS pipeline has a bypass network that effectively performs these functions. In addition, other information such as PC and faulting memory addresses must be preserved in the pipeline until the exception can be serviced. We are exploring the use of relaxed exception semantics to reduce implementation energy. Here a compiler indicates where and when it requires exception information to be preserved by the machine. In other cases, the compiler generates code that can reconstruct machine state in the event of an exception.

3 Processor Energy Accounting

Others have noted that energy consumption in a microprocessor is split across many different circuit blocks, complicating the task of reducing energy [2]. However, we've observed that several of the above changes at the ISA level cause energy savings that ripple through the microarchitecture. For example, tag-unchecked load/stores can be used to reduce energy in address generation and TLB access, as well as the cache itself. The accumulator-RISC hybrid saves energy in the register file and writeback pipeline, but can also save instruction fetch energy by enabling a more compact instruction encoding. To find other example techniques that have similar globally beneficial effects, we believe we must perform processor energy accounting at a level higher than microarchitectural circuit blocks. We are developing a fast energy-performance simulation framework that allows energy consumption to be attributed to various higher level entities such as procedure calling, exception state management, and operand accesses, as well as to individual instructions and circuit blocks.

4 Summary and Future Work

The SCALE project is working towards low-power processors with an integrated approach spanning compiler, ISA, microarchitectural, and circuit-level designs. We aim to demonstrate an initial prototype "Pekoe" within the next year which should provide comparable performance to a MIPS R3000 but at much lower energy by adopting some of the techniques described above.

Acknowledgments: The above reports on work performed by members of the SCALE project including Mark Hampton, Seongmoo Heo, Ronny Krashinsky, Albert Ma, Gong Ke Shen, Jessica Tseng, and Michael Zhang.

References

[1] K. Asanović, J. Beck, B. Irissou, B. Kingsbury, N. Morgan, and J. Wawrzynek. The T0 vector microprocessor. In *Proceedings of Hot Chips VII*, August 1995.

[2] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid State Circuits*, 31(9):1277–1284, September 1996.

[3] J. Goodman, A. Dancy, and A. Chandrakasan. An energy/security scalable encryption processor using an embedded variable voltage DC/DC convertor. *IEEE Journal Solid-State Circuits*, 33(11):1799–1809, November 1998.

[4] J. Scott. Designing the low-power M*CORE architecture. In *Power Driven Microarchitecture Workshop at ISCA98*, Barcelona, Spain, June 1998.

[5] H. Zhang, M. Wan, V. George, and J. Rabaey. Interconnect architecture exploration for low energy reconfigurable single-chip DSPs. In *Proceedings of the WVLSI*, Orlando, FL, USA, April 1999.