19.5: Breaking the GOP/Watt Barrier with EDGE Architectures

Doug Burger and Stephen W. Keckler
Department of Computer Sciences
The University of Texas at Austin
1 University Station, C0500, Austin, TX 78712-0233
{dburger, skeckler}@cs.utexas.edu

Abstract: *Achieving excellent power/performance ratios is easy for processor designs that have sufficiently low performance needs. The techniques traditionally used to extract higher levels of performance from RISC or CISC architectures, however, have either exacerbated power limitations or placed an undue burden on the programmer. In this paper, we describe how Explicit Data Graph Execution (EDGE) architectures have the potential to offer both high performance and high programmer productivity while achieving a high performance/power ratio.*

Keywords: Embedded systems; microprocessors; computer architecture; low-power systems.

Introduction

Balancing power and performance has become one of the dominant issues in system design, from embedded systems to supercomputers. Designers are tasked with maximizing performance given a power budget, or minimizing power given a required level of performance. In either case, designers strive to maximize performance per watt.

Power/performance tradeoffs may be made at both the microarchitectural level and the circuit level. With the former, the architect's goal is to maximize the ratio of fundamental, useful work (ALU operations) to active overhead circuitry, such as control logic. Single-issue, in-order RISC cores, such as low-power ARM or PowerPC implementations, have excellent compute/overhead logic ratios, so are often used in ultra-low power embedded systems. That design point, however, is a low-performance solution; scaling to higher performance while maintaining power efficiency is difficult with current approaches.

Current alternatives for achieving higher performance have power drawbacks. Deeper, power-hungry pipelines increase latch counts, control overhead, and suffer CPI drops. Wide-issue RISC or CISC superscalar cores suffer from increased bookkeeping logic and power-hungry structures that increase quadratically with issue width [3]. Wide-issue VLIW cores, such as Intel's Itanium, push the wrong responsibilities, particularly execution scheduling – into the compiler, resulting in poor performance. Finally, CMPs place the responsibility of obtaining performance squarely on the programmer, significantly reducing productivity. The overheads of running parallel program,s particularly load imbalances, can also result in poor performance/power ratios.

An alternative to these approaches is the design of new instruction sets that provide both high performance and high power efficiency. In this paper, we describe such an approach: Explicit Data Graph Execution (EDGE) ISAs. We also describe the TRIPS architecture, which is a specific example of an EDGE ISA (just as PowerPC is an example of a RISC ISA), and compare it to an Alpha 21264 on a number of microbenchmarks. We show that the TRIPS prototype either eliminates the Alpha's power-hungry hardware structures outright, or greatly reduces their activity factors. The sole exception are the load/store queues, which are considerably larger (and thus power inefficient) in TRIPS.

EDGE Architectures

EDGE architectures [2], conceived and developed as a part of DARPA's Polymorphous Computing Architectures program, are designed to offer high instruction-level concurrency along with significant reductions in overhead and control logic. EDGE ISAs encode dependences in the instructions themselves, permitting limited dataflow-like execution that supports efficient mapping of many different application classes to a common compute substrate. This direct instruction-to-instruction communication permits most operands to bypass the register file. Instructions can be issued whenever their operands arrive from their parent instructions, permitting aggressive out-of-order execution with little overhead or extra control. The hardware thus supports flexible, energy-streamlined dataflow execution, while appearing as a morphable, highly concurrent substrate to the software *without* requiring explicit programmer support or new languages.

The TRIPS hardware prototype implements one type of EDGE ISA on a chip that contains two 16-wide issue cores. The TRIPS ISA breaks code into blocks of instructions and amortizes the control overhead across those instruction blocks. This solution completely eliminates many of the power-hungry structures in traditional superscalar architectures, and reduces the access frequency of many others.

As shown in Figure 1, each block in the TRIPS prototype architecture can contain up to 128 instructions (including 32 loads or stores and up to eight branches). The TRIPS compiler partitions the program into a control-flow graph

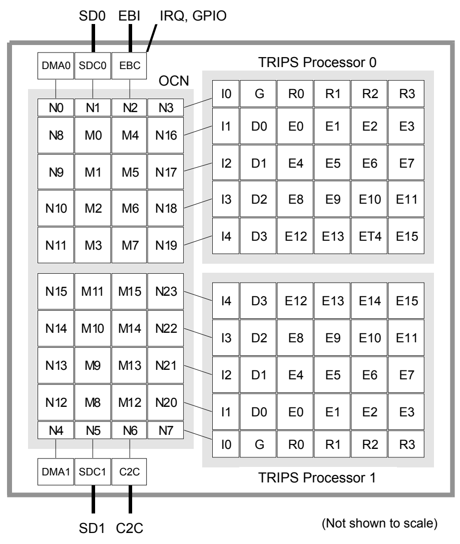

Figure 1. Diagram of the TRIPS ASIC Prototype.

of these large blocks by merging smaller basic blocks into predicated hyperblocks and performing transformations such as inlining and loop unrolling to enlarge the blocks further. The compiler assigns each instruction of a block to one of the sixteen ALUs in the execution array. Up to eight instructions per block can be assigned to any given ALU, hence the maximum block size of 128 instructions (8 instructions per ALU across 16 ALUs).

In the TRIPS execution model, the compiler chooses the physical placement of the instructions, but instructions within a block issue in run-time (dynamic) order, based on when each instruction's operands arrive at its ALU. Unlike a VLIW architecture, the TRIPS compiler does not statically determine the issue order of the instructions. When an instruction fires, it determines the location of its dependent children by examining the target fields encoded in the instruction itself. Each field specifies the ALU and reservation station where a dependent instruction is located. The resultant operand is then routed to the consuming ALU across a lightweight switched network. If the child instruction is mapped to the same ALU as its parent, no routing occurs; if the child instruction has received all of its other operands, it can fire in the cycle immediately following the parent instruction.

When the compiler assigns a block's instructions to ALUs, it attempts to balance concurrency, in which independent instructions are mapped to different ALUs, with reduced latency, in which dependent instructions are mapped to the same or nearby ALUs. In this manner, the wire-delay problems associated with future CMOS technologies are mitigated, as most communication is local or near-neighbor if the compiler is successful

When a block is to be executed, the control tile (marked "G" in Figure 1) performs a block branch prediction, speculating which block is the next to execute. It then accesses its instruction cache tags, which reside only in the G-tile, to see if the block is contained in its instruction cache. If so, the G-tile streams the block's I-cache index to a number of slave banks ("I-tiles" in Figure 1), which each access their portion of the block. The I-tiles stream those instructions across the row of ALUs to their reservation stations at the execution tiles ("E-tiles") where the ALUs are located. Concurrently, each register file bank (or "R-tile"), reads the registers needed by the block and injects them into the routed operand network to initiate block execution. Instructions within the block communicate directly as previously described, without accessing the register files. Only live-out values from the block are routed to the register file and stored, prior to block completion. Within a block, all communication occurs from one instruction directly to another, but all inter-block communication occurs through the register file.

Loads and stores are routed to a column of data cache banks on the side of the processor, and are kept ordered by sequencing queues in those cache banks ("D-tiles"). Before completion, a block sends all stores to the store queues in the D-tiles, all register writes to the R-tiles, and the result of one branch, determining the next block to execute, to the G-tile. At that point, a new block is fetched and mapped into the resources vacated by the old block.

The TRIPS microarchitecture allows up to eight such blocks to be in flight at once. Each ALU must thus contain 64 reservation stations, eight instructions per block at that ALU times eight blocks. Since each TRIPS processing core contains 16 ALUs, each core thus supports up to 1024 instructions in flight. More detail on the TRIPS architecture and overall EDGE concepts can be found elsewhere [2, 4].

This architecture has two capabilities that make it *polymorphous* or able to adapt to many types of workloads such as single speculative thread, loop, vector, streaming, and multithreaded. First, the compiler's ability to place instructions on individual ALUs--combined with the flexibility of the routed mesh inter-ALU network--means that many shapes of dataflow graphs may be mapped to the substrate by the compiler. Whether the graph is a single long dependence chain, or many short parallel slices, the compiler is able to schedule them to the substrate. The second polymorphous feature involves the memory system, also shown in Figure 1, which is a two-dimensional array of polymorphous banks connected by a second lightweight routed network. These banks are individually addressable, and can be configured as a large shared L2 cache, partitioned per-processor L2 caches, or a mix of cache and scratchpad memories managed by the compiler.

Power Advantages of EDGE Architectures

EDGE architectures have the potential for improved power profiles because of the intrinsic instruction-to-instruction communication, which ideally would allow most energy to be spent performing useful computation.

The TRIPS ISA augments this potential advantage through its use of blocks of instructions. By amortizing control and bookkeeping logic across each large block of instructions, the per-instruction energy consumed by control and non-compute circuitry can be potentially much lower than in RISC or CISC architectures. Below we enumerate the potential power advantages of the TRIPS ISA and microarchitecture:

- **Branch prediction**: Only one prediction is done for each TRIPS block, as opposed to one per branch in conventional pipelines. Since each block contains up to eight branches, and many branches are converted to intra-block predicates, a potential order-of-magnitude reduction in branch predictor accesses is possible.
- **Register accesses**: A superscalar architecture requires a large physical register file to support the many in-flight instructions, along with many ports to support the issue of multiple instructions per cycle. In TRIPS blocks, any value which is created and consumed within a block—and is not live past the end of the block—will never access the register file. Since the number of register file accesses is reduced, fewer ports are required; TRIPS supports four register banks, each of which has only one read and one write port. The Alpha 21264, conversely, contains two register banks, each of which has 4 read and 4 write ports, with every register result being written to both banks.
- **Instruction fetching**: In the TRIPS microarchitecture, one instruction cache access provides an entire block of instructions. Only one tag comparison is performed for each block, as opposed to a tag compare for every 4-instruction packet in a superscalar architecture. In TRIPS, when the I-cache tags are accessed and a hit occurs, the microarchitecture sends the correct index to 5 slave banks; each bank accesses part of the block and sends it to the reservation stations in its row.
- **Issue window design**: In a superscalar processor, the issue window contains a CAM in which every entry compares its input operand register tags with the tags broadcast by each completing instruction. In a 4-wide machine with an 80-instruction issue window, 640 tag comparisons per cycle must be made in the worst case. Since EDGE architectures support direct instruction communication, each operand obtains the location of its consumer(s) from its instruction tag bits, is routed to the reservation station holding the consuming instruction, and accesses a RAM entry, without ever performing an associative lookup. The disadvantage to this model is that a distribution tree is required to fan an operand out if it has many consumers.
- **Register renaming**: TRIPS does not need typical register renaming, and requires none of the associative matching used to detect inter-fetch packet dependences in conventional architectures. Since TRIPS does support forwarding of register outputs of one block to register inputs of another in-flight block, it needs some logic to track inter-block register dependences. However, since the fraction of instructions accessing registers is small (10-30%), the energy required to implement this forwarding is likely much less than in conventional register renaming.
- **Loop reuse**: One feature not supported in the TRIPS prototype, but which we will explore in future designs, is the ability to refresh loops. Since instructions are statically mapped, a committing block (in a loop) that finds a predicted block with the same address, need not trigger an instruction fetch or decode, it simply sends a signal that "refreshes" the block by clearing all valid bits and re-injecting the new register values. Thus, if loops are aligned properly with the number of blocks supported by the hardware, they can execute in steady state without instruction fetching or decoding.

Another major source of power consumption is operand routing/bypassing from ALU to ALU. In TRIPS, operands are transmitted along a lightweight 2-D mesh routing network. Reducing router power in TRIPS depends on the compiler scheduling dependent instructions near one another to minimize the number of hops required. In superscalar processors, operands are sent along an all-to-all bypass network of wires, comparators, and muxes. The one structure where TRIPS is currently at a clear power disadvantage is the load/store queue; each memory operation requires an associative search against all other memory instructions in flight. The load and store queues in the Alpha 21264 have 32 entries each; in TRIPS, the load/store queue has 256 entries and there are four copies of it.

Experimental Evaluation

Since power consumption can vary greatly based on design tools, engineering effort, and choice of circuit families, we focused on comparing TRIPS microarchitectural activity counts with those of a conventional, high-performance superscalar architecture to highlight the power pros and cons of the TRIPS architecture and implementation.

The reference model against which we compare is an Alpha 21264 [1], chosen as an example of a high-ILP core that supports branch prediction, dependence prediction, and 4-wide superscalar issue, but which was also designed to run at a high frequency. We compared hand-assembled TRIPS microbenchmarks to the same microbenchmarks compiled for an Alpha with gcc –O3. The alpha binaries were then simulated on our 21264 Alpha simulator, which has been validated against an actual Alpha workstation. The 13 microbenchmarks consist of loops extracted from the SPEC2000 benchmarks.

Table 1. Ratio of TRIPS to Alpha operations.

Benchmark	# Insts	Speedup	Predictions	I-cache	Registers	LSQ	Op. net
ammp_1	75.4%	345.5%	4.2%	3.0%	2.5%	129.7%	225.6%
ammp_2	134.3%	304.6%	13.4%	5.4%	7.8%	54.1%	311.1%
art_1	122.3%	176.3%	15.2%	4.0%	6.1%	49.9%	343.9%
art_2	98.7%	542.9%	24.6%	7.0%	5.4%	73.7%	328.4%
art_3	82.2%	781.2%	15.5%	3.8%	2.4%	69.5%	221.8%
bzip2_3	111.5%	459.0%	11.0%	3.2%	1.6%	52.8%	265.0%
equake_1	71.6%	174.0%	24.7%	3.8%	7.2%	72.8%	264.5%
gzip_2	144.2%	193.7%	17.3%	7.5%	6.1%	2.3%	223.4%
matrix_1	47.0%	315.9%	10.8%	4.0%	5.8%	62.8%	149.8%
parser_1	80.1%	78.7%	8.4%	4.1%	4.2%	2.5%	148.2%
sieve	107.9%	73.0%	53.2%	28.6%	11.9%	39.8%	218.4%
twolf_3	58.4%	140.7%	14.1%	2.8%	1.1%	0.2%	72.6%
vadd	73.2%	186.0%	15.9%	5.5%	2.1%	57.6%	204.3%
Mean	92.8%	290.1%	17.6%	6.4%	4.9%	51.4%	229.0%

Table 1 shows that TRIPS executes only 93% of the instructions, on average, that the same programs compiled to an Alpha ISA would. The speedup (cycle count reduction assuming normalized clocks) factor of three shows the performance potential of the architecture.

The activity factor reductions are also significant. The TRIPS microarchitecture reduces the branch predictor accesses by over 80%, the I-cache tag matches by well over 90%, and the register file accesses (despite requiring fewer ports) by 95%. TRIPS requires fewer load/store queue accesses because it is able to register allocate more loads and stores for the loops. However, each load/store queue access is more expensive for TRIPS than the 21264, as discussed earlier. We do not show issue window lookups, since each TRIPS lookup is an inexpensive RAM access as opposed to an expensive CAM search for the superscalar model. Finally, in the last column of Table 1 we show the ratio of the total TRIPS operand routing hops to the number of Alpha instructions bypassed. This number shows break-even factor by which one TRIPS router hop must be more energy efficient than bypassing one operand along the entire superscalar network. On average, one superscalar instruction bypass must consume slightly more than twice one TRIPS router hop for TRIPS to break even.

Conclusions

Power is a fundamental limiting factor for future high-performance systems. Embedded systems that require more performance than a single-issue RISC core can provide currently face a number of unappealing options.

New instruction set paradigms may enable much more efficient performance/power ratios on systems that can easily achieve a sustained giga-op per watt. In this paper, we have shown that EDGE architectures have the potential to accomplish these goals. Using the TRIPS EDGE architecture as our experimental platform, we have shown that as much of a 20x reduction in activity factors are possible for many of the power-hungry structures in today's high-performance processors. The sole remaining challenge is building power and area-scalable load/store queues, which is the one structure that is significantly worse in the TRIPS design than conventional designs. We have proposed some preliminary solutions to this challenge [5], and are working to address this last remaining issue comprehensively.

Acknowledgments

We are grateful to our sponsors for their support. This research is supported by the Defense Advanced Research Projects Agency under contract F33615-03-C-4106, with additional support from NSF CAREER awards 9985109 and 9984336. We also thank Karu Sankaralingam, Ramdas Nagarajan, Raj Desikan, and Robert McDonald for their assistance with data collection and figures.

References

1. Alpha 21264 Microprocessor Hardware Reference Manual, July 1999. Compaq Computer Corporation.
2. Doug Burger, Stephen W. Keckler, et al. Scaling to the End of Silicon with EDGE Architectures. IEEE Computer, 37(7):44–55, July 2004.
3. Subbarao Palacharla et al. Complexity-effective Superscalar Processors. In Proc. of the 24th International Symposium on Computer Architecture, pages 206–218, June 1997.
4. Karthikeyan Sankaralingam et al.. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In Proc. of the 30th International Symposium on Computer Architecture, pages 422–433, June 2003.
5. Simha Sethumadhavan et al. Scalable Hardware Memory Disambiguation for High ILP Processors. In Proc. of the 36th Ann. Symp. on Microarchitecture, pages 399–410, Dec 2003.