CCC Wkshp. on Advancing Computer Architecture Research (ACAR): What Now in ILP?, Seattle, WA, Sep. 2010

In Search of Big Instructions

Arrvindh Shriraman, Sandhya Dwarkadas, and Michael L. Scott

Department of Computer Science, University of Rochester

Instruction-level parallelism (ILP) typically refers to the concurrent execution of instructions (as defined in the instruction set architecture (ISA)) in an application's sequential thread of execution. There are many design choices in both software (e.g., choice of algorithm) and hardware (e.g., pipelining, superscalar execution) that control the degree of independence and thus the available ILP in today's machines. Given a specific algorithm, however, the ability to expose the theoretical limits of ILP inherent in the algorithm clearly depends on the types of instructions in the ISA. Our premise is that future ISAs need to return to the traditional CISC philosophy with a modern twist—specifically, using encodings that convey more program-level intent to hardware, including data flow and inherent concurrency—information that hardware would otherwise require extra work to extract. Such "big instructions" can encode the relationships among large numbers of low-level operations. They can also capture higher-level operations (e.g., FFT, mpeg encoding) amenable to execution on specialized hardware. Both styles of "big instruction" reduce the energy per arithmetic operation and enable more concurrent operations within the power budget.

Why now/End-of-the-road for RISC ILP ?

RISC advocates the use of a limited set of instructions and addressing modes, most of them operating on a limited set of registers, to keep the microarchitecture simple and the instructions of roughly equal hardware cost whenever possible. This has enabled hardware developers to emphasize pipelining and to ride the technology scaling wave. Over time, architects have augmented the RISC philosophy with techniques such as superscalar instruction-issue, out-of-order execution, and speculation to try to mine and execute many RISC instructions in parallel.

Modern RISC-style architectures (whether at the ISA level or using the x86 approach of breaking up instructions into μops) represent one design point that constrains ILP extraction to the concurrent execution of independent instructions, each the size of a single arithmetic operation. The philosophy of simple instructions implies a small amount of useful work per instruction, resulting in higher relative overheads with deeper pipelines and increasing superscalar widths. Processors spend a large fraction of their energy in the structures required to fetch and decode, detect independence among instructions, and then issue them, resulting in poor energy and area scalability [2]. While technology advances have enabled designers to assume a virtually unlimited number of transistors, power consumption per transistor is no longer scaling with feature size, resulting in energy and power being the primary design constraints. Increasing transistor budgets will only lead to larger fractions of the chip being relegated to dark (inactive) silicon.

CISC Revisited

Modern processors already employ complex multi-cycle instructions that encode data parallelism and are becoming increasingly popular for their performance and energy efficiency. Examples include vector and matrix operations, and common mathematical functions. To reduce energy consumption per operation and to continue to scale performance, we believe this approach needs to be taken to the extreme. Where traditional CISC developed big instructions to reduce the end-to-end latency of complex operations, modern CISC would use them to save energy—to amortize the cost of instruction fetch and decode; eliminate much dynamic dependence tracking; and reduce the amount of architected intermediate state, which complicates exception handling and other aspects of instruction semantics.

Several inroads have been made in this direction. For example, the TRIPS project [1] demonstrated the scaling advantage of explicitly encoding the dependences among large numbers of low-level operations. Tensilica [3] allows customization through an extensible ISA, but requires that the high-level extensions be defined at design time. The challenge remains to create a more dynamic and application-specific environment for instruction fusion. Several key areas of investigation are required:

- *Large and variable granularity instructions:* Unlike previous CISC approaches, which fused a few instructions (e.g., multiply-and-accumulate), we will need instructions that fuse many more operations in order to achieve commensurate energy efficiency and enable concurrent execution. In the general-purpose market with many varied classes of applications, an instruction (as in VLIW) that fuses a fixed number of low-level operations is unlikely to be broadly efficient. Making the choice of number and type of instructions dynamically is still an unsolved problem.
- *Language/compiler support:* Hand-in-hand with the above, identifying high-level operations suitable for new instructions requires an extensive study of common application needs. Modern applications make extensive use of libraries and templates, and spend a significant fraction of their time in these libraries. Preliminary analysis suggests that library APIs may be a valuable source of complex instructions' definitions.
- *Instruction state:* As the number of operations in flight increases, architects will need to move away from the staging of data in programmer-visible registers. Allowing high-level, data-parallel instructions to operate directly on memory may allow hardware to optimize the use of bandwidth more effectively than compiler-generated loads and stores. At the same time, if instructions try to access state that could potentially raise an exception (e.g., page fault), the cost of a restart could be large and/or require significant operating system support. For pipelined execution of low-level operations, intermediate state may be managed significantly more efficiently if it is not architecturally visible. Again, this introduces challenges for the precision and efficiency of exception handling.
- Semantics: Traditionally, techniques that exploit ILP have preserved the notion of program order and obeyed data and control dependence. As the number of in-flight operations increases, more distributed forms of completion detection may be required. Current hardware also seeks to maintain the appearance of instruction atomicity, an increasingly difficult goal with complex instructions. Future machines may need to expose internal state to the operating system (but not the application).

References

[1] D. Burger et al. Scaling to the End of Silicon with EDGE Architectures. *IEEE Computer*, 37, 2004.

[2] R. Hameed et al. Understanding sources of inefficiency in general-purpose chips. In *Proc. of the 37th ISCA*, 2010.

[3] A. Wang et al. Hardware/Software Instruction Set Configurability for System-on-Chip Processors. In *Proc. of the 38th DAC*, 2001.