Disclaimer: "The contents of this document are scribe notes for The University of Texas at Austin EE382V Spring 2007, Computer Architecture: User System Interplay". The notes capture the class discussion and may contain erroneous and unverified information and comments.

WaveScalar

Lecture #4: Wednesday, 31 January 2007
Lecturer: Mattan Erez
Scribe: Sabrina Smith
Reviewer: Min Kyu Jeong and Mattan Erez

1 Basic Concepts

In between discussing the WaveScalar paper, we discussed a couple of key concepts which appeared in the paper. Those are discussed first in these notes, to make the rest of the notes a little more cohesive.

1.1 Hyperblock

A hyperblock is a collection of basic blocks with a single entry point, multiple exits, and no loops. It is an extension of a superblock. The difference is that a superblock does not allow multiple control flows within the block; whereas, a hyperblock does by predicating on the control flows (such as the If-conversion) for better utilization of VLIW.

1.2 Memory Ordering

Total memory ordering (also known as strict memory consistency) is the requirement that all stores and loads are viewed as in the sequential ordering given be the compiled control flow. WaveScalar uses waves and tags to implement total memory ordering. The compiler gives every memory operation a tag which consists of three components: the ID of the previous memory operation, the ID of the current memory operation, and the ID of the successive memory operation (in control flow order as specified in the code). The tag might look something like the following: <0,1,?>. The '?' signifies that the ID of the successive memory operation is not known. ?'s can be resolved by introducing one or more additional states into the control flow diagram.

2 Problems Being Solved

Modern day superscalar processors are becoming more and more difficult to scale due to many factors. Some of these include wire problems (wires of the same logical distance scale, but wires of increased logical distance do not), complexity of the design, complexity of the verification process, and reliability of the technology. Because of this, the authors of *WaveScalar* attempt to create an alternative architecture for low- complexity/high-performance processors. This architecture should support von Neumann semantics and imperative languages (languages which may have side effects) in dataflow.

3 Uniqueness of Solution

There are many aspects of the solution, some of which are unique and some of which are not. Some of the less unique aspects include simple building blocks, a hierarchical on-chip network, computations that are done in memory (WaveScalar uses a model of computation occurring where the data is, which is unique and different from other compute-in-memory ideas), and non- uniform communication that is exposed to the compiler, but not to the languages. A list of the unique aspects of the solution appears below.

- Attempts to minimize communication cost instead of maximizing ALU processing utilization.
- Supports total memory ordering in dataflow.
- Completely distributed dataflow, allowing multiple concurrent use of data.
- Support for dynamic dataflow instruction placement and caching?
- Concept of "waves": A wave is an acyclic directed graph of part of the program. It is supposedly an extension of a hyperblock, but there is some confusion about the actual differences between a wave and a hyperblock. In the paper, a wave is described to have a single entry point and no loops, but it may include branches or joins.
- Concept of a "wave cache", in which each datum is associated with a computation (previous compute-in-memory were more of ALUs in DRAM).
- Supports binaries (post-translation) from languages that require memory ordering but don't give hints about how to enforce it.

4 Intended Users of System

The intended users of WaveScalar are general purpose processor (GPP) users. WaveScalar is intended to be a GPP replacement.

5 Evaluation of Solution

The following bullets describe the evaluation (or lack thereof) of the WaveScalar processor.

- Showed they could successfully run applications on the simulator.
- Compared WaveScalar with a superscalar processor and a TRIPS processor.
- Used "Alpha equivalent IPC" (AIPC) to count the IPC of their own system. This was done because Alpha is a very clean instruction set.
- Performed sensitivity study of queue sizes, wave cache size, speculation, and clustering degree.
- Assumed perfect L1 caches, i.e. assumed that memory bandwidth and latency don't affect performance. They made this assumption for both WaveScalar and comparison machines, but this could lead to skewed results. In fact, the clustering sensitivity study performed for this paper hinted at what later research proved – the assumption of perfect L1 caches benefited the WaveScalar machine for than the comparison machines.
- Didn't consider interrupts, exceptions, OS, etc.
- Assumed perfect speculation, so didn't talk about how to recover from or the potential costs of missed speculation. (As a side point, important to realize that memory speculation affects load/store order and control flow.) This assumption was made for part of the evaluation only.
- Utilization was not evaluated – implied that peak/utilization numbers are meaningless because you can build as many ALUs as you may want on modern systems. This is a unique aspect of this paper.
- Peak performance not evaluated.
- Complexity and reliability talked about some, but not really evaluated.

6 Technology as a Factor

Technology was definitely a factor in the solution. It was the driving motivation for the need to rethink superscalar processor architectures and come up with a radically different architecture based on dataflow.

7 New Tools, Software

The designers of the WaveScalar processor also wrote a binary translator (the WaveScalar compiler) that divides a program into waves and adds some wave management instructions. This compiler breaks the control flow graph of the program into waves, reasons about the memory ordering within those waves, and translates the code into dataflow.

8 How Solution May Affect Other Users

WaveScalar could affect other users in a variety of ways:

- The fact that interrupts, exceptions, etc. were not evaluated could be problematic.
- The code density of the programs may cause the code not to fit within embedded systems (the WaveScalar compiler had to grow the code to achieve the proper dataflow).
- Thread level parallelism was not discussed.
- Real-time was ignored.

9 How Paper Relates to General Concepts of Class

The discussion of how this paper relates to the general concepts of the class was left to the next lecture.