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.*

General Purpose Processors: Binary Compatibility

Lecture #03: Monday, 29 January 2007
Lecturer: Mattan Erez
Scribe: Paul Zucknick
Reviewer: Min Kyu Jeong

1 The Transmetta Code Morphing Software: Using Speculation, Recovery, and Adaptive Retranslation to Address Real-Life Challenges

1.1 Reference

References

[1] Dehnert, J.C.; Grant, B.K.; Banning, J.P.; Johnson, R.; Kistler, T.; Klaiber, A.; Mattson, J., "The Transmeta Code Morphing™Software: using speculation, recovery, and adaptive retranslation to address real-life challenges," *International Symposium on Code Generation and Optimization, 2003. CGO 2003.*, pp. 15- 24, 23-26 March 2003.

1.2 Introduction

This paper comes from industry. Transmetta's Crusoe microprocessor is a VLIW microprocessor that along with a software layer, the Code Morphing Software (CMS), provides a full implementation of the x86 architecture. The paper discusses the software system and the hardware support for implementing x86, specifically exceptions and interrupts, I/O, and self-modifying code.

The CMS software builds on top of a binary translation system. A binary translation system takes an instruction stream from one machine and executes it into one of more instructions on another machine. Conversion can be either complex-to-simple or simple-to-complex. We use binary translation every day (i.e. Java Virtual Machines). Binary translation involves dynamically taking 1 instruction set and translating it to another instruction set just like a compiler, but not using a high level language. Binary Translation is covered in Section III of this paper.

Transmetta starts with a simple sleek host ISA and they take the target x86 code and translate it to their VLIW architecture. The paper makes the assumption that there is a sequential instruction set with Von Neumann Semantics. Von Neumann was the father of modern computing (**NOTE:** the paper for next time tries to get around the problems of the Von Neumann model). In the Von Neumann model, a program is composed of a set of instructions and state. Instructions manipulate state with precise semantics. Each instruction is executed one at a time and effects everything that follows and everything else in the system.

To Generalize the paper, the goal was to make the common case fast and efficient in HW and handle the exceptional cases (exceptions, etc) in SW with the CMS system. This approach allows the overall design to be sleeker and perhaps better, but the system runs slower than native x86. The overall idea of the solution is that you gain the flexibility to change the underlying ISA and make improvements by using this technique. (This was not explicitly stated in the paper).

Why would we care about this paper? A flaw of the paper is that the motivation is not mentioned. However, the paper is governed by the implicit assumptions within the academic community that improved performance is better. In general as you write a paper you have very little space, and sometimes things end up missing from the final draft. We will discuss the missing elements in class.

2 Discussion Points

2.1 What Problem is being Solved?

- The number one goal of the paper is that the authors are trying to run x86 code on a VLIW machine. (The key is to run the code correctly, but this is assumed to be true.)
- The authors try to address specific issues with regards to translation (precise exceptions, interrupts, memory-mapped I/O, and self-modifying code)
- The authors want the freedom to have the ability to modify the underlying implementation (ISA)
- The authors are trying to build a system that implements an ISA with both the hardware and software and not just hardware alone.

2.2 What is Unique?

- The solution uses both HW and SW to implement the ISA.
- The solution provides optimizations for high performance
- The system "Performs" as well as native x86 HW.
  However, "Perform" is loosely defined? (assumed to be wall clock time)
- The paper claims that their processor uses less power than x86 hardware
  This is an attractive point in terms of power but no proof is provided.
- The authors want to create a "Robust" design (However, the definition of robust could be interpreted in different ways)
  In general a robust design is a design that meets design goals or a specified target without having a large spread in the results.
- The solution provides hardware acceleration for commit and rollback (simple / elegant catch all) This enables "efficient" (NOTE: efficient is not defined in the paper) speculation - no need for compiled bookkeeping code.
- There is no reliance on the OS
- New translation techniques are introduced.

The big idea of the paper is the ability to take a piece of code and get the expected performance regardless of the application without using just HW. The paper presents a HW/SW combination that goes from a legacy binary and allows it to be run on new systems with improved performance capabilities. They change the HW implementation with some benefit to the users and take some more complex things and offload them to SW.

2.3 Who are the Intended users?

- x86 users

2.4 What are the goals of the design?

- Improved wall clock time.
- Reduce power by using less HW (VLIW architecture)
- Reduced "cost"
  - Less cooling
  - Easier to design

  - Easier to verify (is this really true???)
  - The design actually becomes harder to verify because both the software and hardware must be verified.

- The goal of the paper wasn't enable a transition to a VLIW architecture; the goal was to utilize the simpler design and use a HW/SW combination.

- The goal is to have a system that looks like the old system regardless of the level, BIOS, OS, etc.

2.5 General Discussion

What you gain/don't gain depends on the translation, but this isn't talked about in the paper. Also, the paper implies that less hardware means reduced power and a simpler design. This is briefly mentioned but not always necessary and not even slightly addressed in the paper.

- Discussion Points (Questions to consider)

  - Is the purpose that target applications can take advantage of VLIW architecture? -Or- Is this against the main point of the paper?
  - Does running CMS defeat everything?
  - Do they gain or loose by running the SW part of the application?
  - By doing translation, do they end up with more efficient execution?
  - Other papers discussed dynamic voltage scaling, but this paper didn't prove or provide background in regards to reduced power in a larger context.
  - Was the paper claiming more than it set out to discuss in terms of details?

2.6 Chaining Definition

Chaining is the action of pointing from one translated sequence in the cache to the next translated sequence. (similar to a SW trace cache) A translated sequence is joined to another translated sequence without having to go back and look at the x86 code. This is useful when code repeatedly does the same thing (similar to a JIT compiler)

2.7 Commit/Rollback Discussion

- The hardware maintains a working state and shadow register state

- The architectural state is the level that the software sees

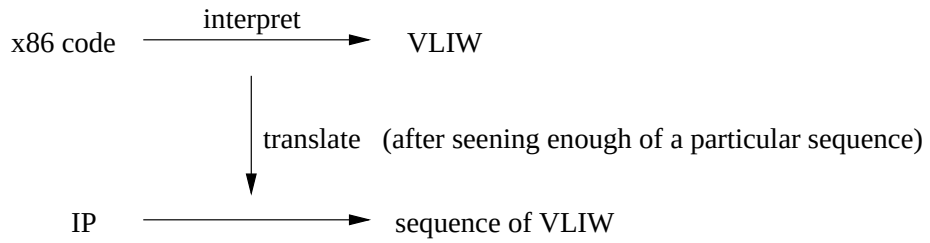

Figure 1: Translation and Chaining Illustration.

Commit/Rollback are used for dealing with precise exceptions. First you re-execute the code to make sure the error didn't occur because of speculation. If the exception occurs again in the second execution it was a true exception and must be handled (1 instruction at a time). The hardware maintains precise state and reports to the SW system. If the exception isn't exceptional (i.e. LD/ST dependence), the SW reduces the scope of the translation and becomes more conservative in the translation. HW support enables a shift from HW to SW by changing partitioning and also allows for flexibility. The goal was that code would see no difference between native x86 and the Crusoe solution.

2.8 How is the idea evaluated?

- The authors don't evaluate the benefit of HW rollback technique well.
- The authors don't really compare to anything. (i.e. a solution like compiling from source)
- Translate not migrate - Doesn't compare 'native" VLIW to CMS

2.9 Is the evaluation in line with the stated user requirements?

- The paper did not provide any comparisons or evaluation in regards to known techniques and the authors do not even mention previously known techniques. Basically they're saying "We're better than not doing anything".
- The paper goes into great detail about support for speculation, recovery, and adaptive retranslation that is needed to support the x86 architecture, but the individual sections are not explained well. This is true even for the section on self-modifying code, which is the largest section of the discussion.
- The authors neglect to mention the hardware costs that would be necessary to build their processor, besides stating that it would be "simple/small".
- The authors' solution has merit, but they do not provide a thorough or convincing evaluation.

- In terms of evaluation techniques it would have been preferred to see how the system performed as a whole compared to other solutions.

- They mentioned that they want their design to be robust and applicable to everyone who runs on the system, but their techniques are unpredictable and time constraints, etc are not mentioned.

2.10 How may other users be affected?

- How does it affect other users of x86?

  - Specific code may see some performance gains; other pieces of code may experience slow downs.
    For example, code that has a lot of interrupts or exceptions might suffer because the processor will constantly be going to software to handle the interrupts/exceptions.
  - Different usages models and paradigms may/may not benefit.
  - Systems with real-time constraints may be at a disadvantage
  - Power numbers are not mentioned.

Comment: The software technique presented in the paper analyzes blocks of up to 200 instructions. This number is actually pretty small and it seems like the SW translation is pretty limited. Overall, they don't take advantage of the partitioning between hardware and software .

In the paper the authors mentioned that when something is identified as "exceptional" the translation system takes a specific action, but the technique/method for identifying something is not mentioned or identified. The authors could have taken the same techniques and developed them more.

2.11 Paper Summary

The main problem, which is a common problem, is that the authors want to maintain the same ISA for legacy code. The authors choose a hybrid approach using both hardware and software translation techniques. The hardware is efficient for commit and rollback and the software is used to handle exceptions and save the cost of having exceptions implemented in hardware. This is an interesting and important paper; however a better evaluation is necessary. It would be interesting to see the techniques presented in the paper compared and validated against known techniques. Overall, the paper presents different ideas about HW/SW partitioning and uses this partitioning to gain benefits for what the users want.

3 Next Time Prep

The paper for Wednesday (Wavescalar) will discuss a similar topic, but the paper takes the existing source code and uses a different model than the Von Neumann model. The paper builds hardware that follows the dataflow execution model and they move to a new architecture with power, performance, and ease of design all factors that have to be considered.

3.1 DataFlow Example

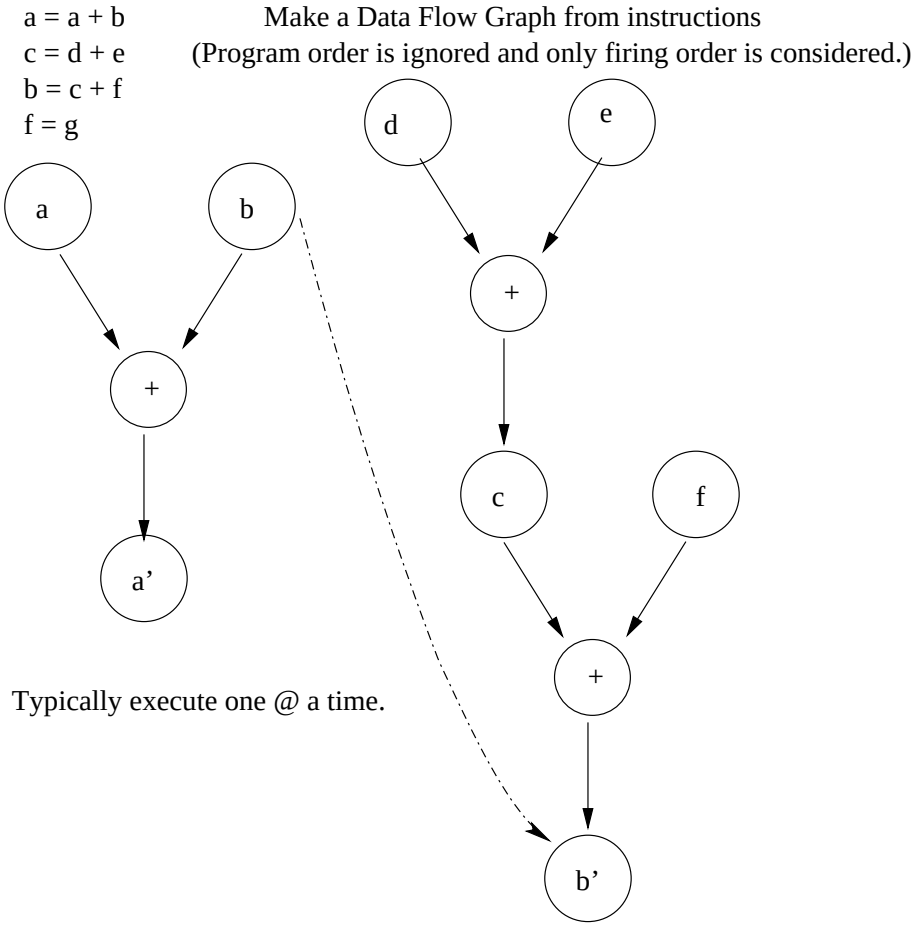

Figure 2: Dataflow Example.