MICRO35 Paper Submission Site

See all the finished reviews for Paper #104

Go To MICRO35 Site | Go To MICRO35 Submissions

It is currently Wednesday 31st of July 2002 09:26:51 AM MDT

You can see the paper reviews until Friday 02nd of August 2002 01:05:00 PM .

There are 4 finalized reviews and 0 started, but unfinalized, reviews for your paper.
A total of 5 reviews were requested by the program committee.
You will receive an email notifcation when any of the unfinished reviews are finalized.

Respond To The Reviewers Comments

Paper #104	(Download paper of type application/pdf)
Title:	Levo - A Scalable Billion Transistor CPU With High IPC
Abstract:	The Levo high ILP microarchitecture is described and evaluated. Levo employs instruction time-tags and active stations to ensure correct operation in a rampantly speculative and out-of-order resource flow execution model. The Tomasulo-algorithm-like broadcast buses are segmented; their lengths are constant, that is, do not increase with machine size. This helps to make Levo scalable. Known High-ILP techniques such as Disjoint Eager Execution and Minimal Control Dependencies are implemented in novel ways. Examples of basic Levo operation are given. A chip floorplan of Levo is presented, demonstrating feasibility and little cycle-time impact. Levo is simulated, characterizing its basic geometry and its performance.
No author response submitted	

Review #495 For Paper #104

(review last modified Sunday 28th of July 2002 05:58:41 PM)

Attribute	Value
Provide a short summary of the paper and its contributions	This paper describes a machine with an execution core that can execute instructions out-of-order without performing register renaming. Instructions are kept in program order in the execution window. Instructions will execute whenever the architectural register ID of one of their source operands is broadcast by an older instruction (with a few exceptions, e.g. if out-of-order writes occur). When an instruction executes, it broadcasts its architectural register ID (rather than a rename tag as in Tomasulo's algorithm) along with a time tag. The time tags keep track of program order, and are used to detect out-of-order writes. Stores broadcast their memory addresses and values to younger instructions. Since there may be multiple instructions in the window writing to the same architectural register

	or memory address, instructions may execute multiple times. Since the time-tags are used to track out-of-order writes, instructions will eventually end up with the correct source operand values when they reach the head of the instruction window. The CPU is also designed to easily exploit two other techniques for extracting high ILP: hardware-based predication and Disjoint Eager Execution (DEE). Predication eleminates the requirement to flush some off-path instructions, and DEE alleviates the branch misprediction penalty. The main problems with the paper are (1) the cycle time will be very low, compared to how current microarchitectures set their cycle times, and (2) there is no motivation as to why this design is better than a traditional approach.
Provide detailed, construtive comments to the author	[1] The clock frequency may be significantly lower than that of a competing processor with a more traditional microarchitecture. The following three steps have to happen within 1 clock cycle: (1) a shared group must broadcast a destination register to 7 other shared groups within the column (for an 8-4-8 configuration). Since each shared group contains a register file, L0 cache, and store buffer, they are likely to take up a significant amount of area, resulting in long propagation delays. (2) Each shared group must compare all destination register IDs and store addresses received in order to track only the most recent value for each unique destination register ID and memory address in order to prevent write-after-write hazards. The old and new values for each register must then be compared to see if there is a change, since unmodified register values do not require re-execution. (3) Once the modified registers have been identified, instructions within the sharing group must be scheduled for execution. Steps 1 and 2 each take significantly more work than what can be done in one cycle on current microprocessors. You should try to pipeline this path, and re-evaluate the IPC. [2] The PCBs (i.e. store forward buffers) may have to hold more addresses than just those produced by the previous column in the previous cycle. It is possible that a PCB may receive data before a load within the shared group knows its memory address. Is this what the L0 cache is for? The paper never explains what the L0 cache is used for. Although would need to be larger than the 32 entries to handle the worst-case scenario where there are many stores in the window, and there is one load with an unknown address. [3] The definition of the geometry x-y-z is a bit ambiguous. According to

	the definition, x means the number of shared groups per column. But from the rest of the paper, it sounds like the number of shared groups per PAIR of M- and D-columns. To be clear, the paper should say that an x-y-z configuration has --- processing elements and --- active stations. [4] Is there a limitation on the bandwidth for the backwards requests? The bandwidth for just-computed values is limited to 8 if there are 8 processing elements per column. The backwards requests will require additional bandwidth through the RFUs. Without limiting the number that can be satisfied within a cycle, the RFU bandwidth grows by at least the number of architectural registers. [5] The description of the taken branch table used for hardware prediction is confusing. It seems like there should be a MISPREDICTED branch table rather than a TAKEN branch table. [6] The paper applies software concepts to hardware, and it does not make sense. For example, backwards branches are unrolled in the I-Fetch unit and converted to forward branches. This could make sense for a trace cache, but not for a dynamic instruction window. As another example, subroutine calls are inlined in the I-Fetch unit. [7] The implementation of DEE is not explained well. Does each RFU have to keep track of two sets of state (on-path and off-path)? What happens if a D-path spans more than one column? How are dependences on different paths distinguished? half of the window is dedicated to off-path instructions. Why? Is this a performance advantage?

Review #674 For Paper #104

(review last modified Monday 29th of July 2002 07:45:52 PM)

Attribute	Value
Provide a short summary of the paper and its contributions	This paper describes what appears to me to be a novel architecture for achieving potentially high IPC rates by distributing a large number of processing elements among instruction queue entries. This approaches uses a time-tagging system to ensure that the correct result value is used in a computation, instead of a traditional register-renaming scheme. In order to satisfy the demands of a large number of function units, the register file is duplicated for each processing element.
Provide detailed, construtive comments to the author	I believe that this is an interesting work, and would have provided higher marks if the paper had been put together a little better. As it stands, it fails to provide many details that I believe are important (see below for detailed comments). While I am not as well-read as I would like to be, it would surprise me if the time-tagging concept had not been mentioned in print (or in hardware) prior to 1995.

I believe that the work has merit, though I'm a little concerned about the amount of wiring required to implement this type of design.

I like the idea of using replicated register files, and the way that instructions are executed speculatively until their actual input values arrive. While this design is in no way going to be low-power, circuit design should be simplified via localization. Distributing the logic across the die should also allow a great deal of scalability. Also, the actual processing power can be tweaked to just the right "point" by adjusting the knobs as was done in figure 10.

I want to state that I STRONGLY feel that quoting performance numbers in the abstract, introduction, or conclusions that were achieved using anything _other_ than realistic hardware can be misleading. You must either specify your experimental setup for the numbers you have quoted, or quote the real-hardware numbers.

In your methodolgy section, you discuss using a trace-based simulator. Are you able to accurately model wrong-path behavior? If you do, you should mention it. If you don't, you should explain why you believe that your IPC numbers have meaning.

Detailed Suggestions:

I'm very concerned about the wiring density in this type of design, the broadcast busses are WIDE (especially if memory addresses are to be broadcast), and there are quite a few of them. Your floorplan doesn't indicate to me that you have taken this sufficiently into account.

While I'm talking about busses, how many forwarding busses are there in this design? Do the MFU and PFU have separate busses? If not, how do they interact with the RFU busses? For that matter, how do the RFU busses actually interact with the SG's and the AS's within (figure 4 relly doesn't help much in explaining this)?

When are "backwarding busses" actually required? How many of them are there? How do they connect to the SG's, RFU's, etc? You mention these busses in a couple of places, but never really explain this.

I'd be interested to know just how important DEE is to your design... how much performance does it buy you (again, with and without real hardware)? Does it make more sense for all of the AS's to be on the M-Path?

As far as the paper structure goes, I'd suggest a little better top-down approach. When I first read it, I felt as if you had "sprung" ideas on me...

Section 2 could use some help also: ISCA 2002 had quite a number of papers on how to build larger instruction queues, and also on deep pipelines. We also know that register files can be replicated to alleviate most issues related to their speed & size.

	In figure 1, the "instruction window" label is somewhat confusing (I think of an instruction queue when I see that label)... so you may want to change the label or redifine the term.

Review #711 For Paper #104

(review last modified Tuesday 30th of July 2002 09:32:20 AM)

Attribute	Value
Provide a short summary of the paper and its contributions	This paper describes the Levo architecture, an architecture capable of extracting ILP from serial instruction streams. The architecture combines decentralized broadcast, timestamp tagging, value prediction, eager execution, and dynamic predication.
Provide detailed, construtive comments to the author	This paper presents a collection of known techniques combined in the Levo architecture. Given that most of these techniques have been proposed in the past, the paper must have a much more complete related work description. Furthermore, the authors must put the advantages of the Levo design into the context of current designs, by using a baseline that is not a Levo-based design. Specific comments in this regard follow... Regarding your design, you are missing many highly relevant references to prior art. Examples: the timestamping is similar to iteration count "stamps" used in early dataflow systems, the segmented broadcast technique is similar to Raasch's segmented instruction queue (in the last ISCA), the hardware predication technique is similar to Klauser's dynamic predication (PACT'98) or VijayKumar's Skipper architecture (last MICRO), DEE is similar to Klauser's Polypath design, and so on. Regarding your time-tagging technique, how is it that instructions know which broadcasted register or memory value is the most-recent previous value? It is not clear how this is accomplished from the text. It is apparent how timestamps could be used to prevent a later definition from being visible, but it is unclear how the same mechanism can be used to identify the most-recent previous definition! Renaming accomplishes this task in a straightforward manner by renaming instructions in program order, thus the definition tag in the rename table is always the correct defnintion for a given value access. Also, how large are these time tags? What happens if they overflow?

	Many more details are required... What is the purpose of the backward buses, it was not clear from the text. Regarding physical analyses, what are the circuit performance implications of this design? While area is a concern in scheduler design, circuit speed is as great (or greater) concern. How does the design compare (in area and speed) to a more conventional Tomasulo scheduler? Regarding your performance analyses, it is difficult to gauge the benefits of your design because you only compare Levo designs to other Levo designs. How does it compare to a conventional Tomasulo-style scheduler?

Review #753 For Paper #104

(review last modified Wednesday 31st of July 2002 09:13:46 AM)

Attribute	Value
Provide a short summary of the paper and its contributions	This paper presents an out of order microarchitecture design where instruction execution is enabled by time tags and expected branch behavior leading to the instruction. Rather than having a stationary reservation design, instructions are moved along in columns and eventually reach commitment stage. The authors show that a large IPC is possible in this microarchitecture with some simulation studies.
Provide detailed, construtive comments to the author	General comments: The basic ideas of this paper are interesting but not entirely new, mostly related to Uht's previous work. I think that there are some potentially useful ideas in the paper. The paper is, unfortunately very hard to read in its current form. Here are some points that the authors should try to clarify to make the paper more readable. Specific comments: 1. The authors should clarify the amount of information being moved around (time tags, taken branch table contents, RFU contents, etc.) in every clock cycle. This can be illustrated with a good example. 2. Figure 5c is quite confusing. For example, the authors should explain the semantics of the taken branch tables and the canceling predicates carefully before entering the example. Also, what does "I3 pred. And ResTT broadcast mean?" There are quite a few more questions regarding Figure 5c that makes it very hard to evaluate all the activities involved in each prediction and re-execution. 3. Section 4.1, the description of M path and D path are quite confusing. Again, the authors need to carefully explain the design concepts before diving into a specific example. 4. The authors claim that the proposed microarchitecture can tolerate hundreds

	of cycles of memory latency. This should be carefully explained in genral and with a good example. 5. Given the problems with the explanation of the main concepts and operations, Section 5 and 6 become quite meaningless. The paper is way too long as is. The authors need to focus on explaining the key concepts.

Respond To The Reviewers Comments

Goto Main Index Close Window