Presented at PACT2001 Work-In-Progress (WIP) Session.
To appear in IEEE TCCA Newsletter.

Levo: IPC in the 10's via Resource Flow Computing

Augustus K. Uht*, David Morano°, Alireza Khalafi°, Marcos de Alba°,
Thomas Wenisch*, Maryam Ashouei° and David Kaeli°

Departments of Electrical and Computer Engineering

*University of Rhode Island
{uht,iota}@ele.uri.edu

°Northeastern University
{morano,akhalafi,mdealba,mashouei,
kaeli}@ece.neu.edu

I. Introduction[1]

Many studies have concluded that typical programs (e.g., SPECint) contain a significant amount of Instruction Level Parallelism (ILP) for non-oracle assumptions. For example, Lam and Wilson[2] reported an ILP of about 40 for SP-CD-MF (single path speculative execution with minimal control dependencies[3]).

So why are academia and industry still working with single digit IPC's? In short, nobody has been aggressive enough: issue widths are typically set to four instructions and advanced high-ILP techniques have not been used. One of the major problems faced by current designers is that none of the existing methods scales well and the required hardware is both complex and costly.

The Levo machine model attempts to extract available ILP and still considers both scalability in hardware and simplicity in design; see Figure 1.

We have initially evaluated Levo with a limit study having real hardware constraints.

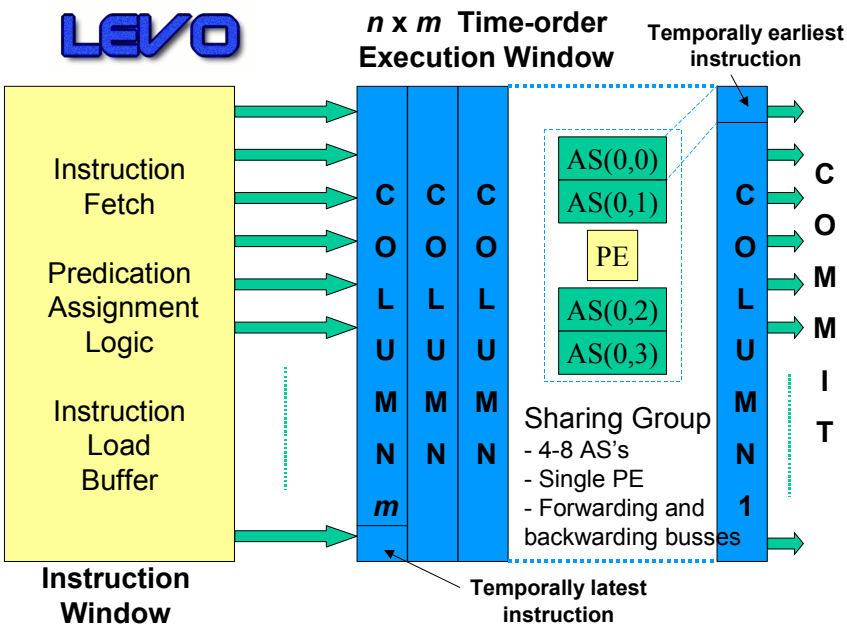

Figure 1. Levo Block Diagram. Memory Window not shown.

II. Key Ideas in Levo

First, instruction issue is in-order and speculative. Although currently single-path (SP), we are also considering multi-path issue.

Instruction time tags – used to enforce data and control dependencies; each tag is typically a small number of bits per instruction.

Hardware runtime predication – used for all forward branches with targets within the execution window. Backward branches are handled via unrolling and conversion to forward branches.

Registerless datapath – there is no central ISA register file. This reduces contention for a key shared resource.

Active stations (AS) – a more intelligent version of Tomasulo's reservation stations.

Resource flow execution – aggressive speculation. Instructions are executed independently of any data flow or control flow dependencies, as long as execution resources, such as Processing Elements (PE), are available.

III. Levo Time Tags

The time tags enforce the nominal sequential order only of dependent instructions. This results in minimal data dependencies being enforced (only flow), and in conjunction with the hardware predication mechanism results in minimal control dependencies.

Time tags accompany all in-flight (in Execution Window) register and storage values. Time tags also accompany predicate outputs of branch instructions, which are broadcast similarly to register and storage values.

Each time tag has two parts:

- *Column tag* – this is decremented by 1 whenever the left-most column is loaded, corresponding to a column right-shift in the Execution Window.
- *Row tag* – this never changes.

[1] This work was partially supported by the National Science Foundation through grants: MIP-9708183, DUE-9751215, EIA-9729839; by the URI Office of the Provost; by an equipment grant from the Champlin Foundations; by donations from Mentor Graphics Corporation and Xilinx Corporation; and by the Spanish Ministry of Education. Patents applied for.

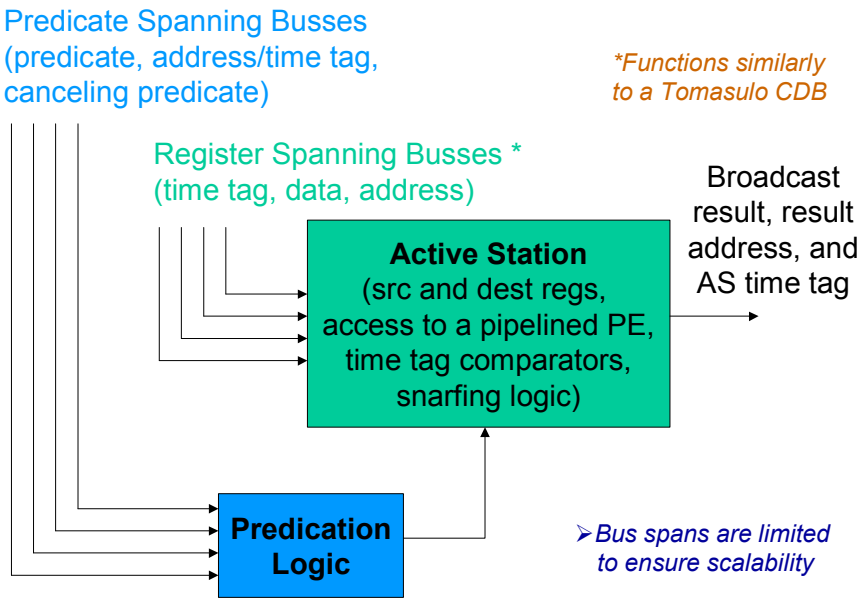

Figure 2. Active Station. Details and bus connections. Memory Spanning Busses not shown; are similar to Register Spanning Busses.

The concatenation of the column tag with the row tag gives a unique indicator of the position of the associated instruction in the execution window. With the column decrementation indicated above, the time tags are always locally accessed small integers, unlike other approaches[1].

Time tags are also used in Levo's memory commit stage. Depending on the results of our current studies, we may also use them in the Memory Window.

IV. Levo Description

Referring to Figure 1, we see that Levo is composed of three windows: Instruction, Execution and Memory (not shown). The Instruction Window performs branch and loop predictions, fetches instructions, and puts them in the Instruction Load Buffer for future loading to the leftmost column of the Execution Window.

The Execution Window is an *n*-by-*m* array of Active Stations, each Active Station corresponding to one instruction and one time tag. Multiple Active Stations in a column are combined into *Sharing Groups (SG)*. The AS's within a Sharing Group vie for the resources of the group, including one or more pipelined Processing Elements (like complete ALU's) (PE) and the broadcast bus output(s) to the appropriate *Spanning Buss(es)*. The spanning busses interconnect the Sharing Groups. Each SG sources one or more spanning busses. Each spanning bus is connected to temporally adjacent Sharing Groups among one or more columns. The spanning bus length is constant and does not change with the size of the Execution Window; this ensures scalability. A spanning bus is typically one column long.

Spanning busses are comprised of both *forwarding* and *backwarding* busses. Forwarding busses are used to broadcast register, storage and predicate values. If an AS needs an input value, it sends the request to earlier AS's via a backwarding bus. The requested data is returned via the standard forwarding bus protocol.

Referring to Figure 2, an Active Station connects to the spanning busses corresponding to the AS's position in its column. Each AS performs simple comparison operations on the time tags and addresses broadcast on the spanning busses to determine whether or not to snarf data or predicates.

A *Predicate* is assigned to AS's after a branch but before the branch's target (the branch *domain*[3]). A *Canceling Predicate* is assigned to instructions after the domain, to remove the enabling/disabling function of the predicate.

V. Modeling and Results

Three levels of modeling are used: –Trace-driven model (FastLevo), –Cycle-accurate model (LevoSim), and –Synthesizable VHDL hardware model (HDLevo).

With a 256 (32x8) AS Levo, FastLevo gives IPC's of 8-12 on a subset of SPECint2000; there were various conservative and optimistic assumptions made, which will be removed in later studies. The initial results were validated by both LevoSim and HDLevo.

More information on the Levo microarchitecture and these initial results may be found in [4], available via: http://www.ele.uri.edu/~uht .

References

[1] J. G. Cleary, M. W. Pearson, and H. Kinawi, "The Architecture of an Optimistic CPU: The Warp Engine," in Proceedings of the Hawaii International Conference on Systems Science (HICSS), vol. 1: University of Hawaii, January 1995, pp. 163-172.

[2] M. S. Lam and R. P. Wilson, "Limits of Control Flow on Parallelism," in Proceedings of the 19th Annual International Symposium on Computer Architecture. Gold Coast, Australia: IEEE and ACM, May 1992, pp. 46-57.

[3] A. K. Uht, "A Theory of Reduced and Minimal Procedural Dependencies," IEEE Transactions on Computers, vol. 40, no. 6, pp. 681-692, June 1991.

[4] A. K. Uht, D. Morano, A. Khalafi, M. d. Alba, T. Wenisch, M. Ashouei, and D. Kaeli, "IPC in the 10's via Resource Flow Computing with Levo," Department of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 092001-001, September 18, 2001.