Levo - A Scalable Billion Transistor CPU With IPC in the 10's

Augustus K. Uht, David Morano, Alireza Khalafi and David Kaeli

1 Introduction

Why we still care about GP microprocessors: GP Microprocessor technologies drive much of the advances in embedded systems. As was case with mainframes of 30 years ago and microprocessors of 10 years ago, is case with micro's of today and embedded systems, resp. (Reconfiguration and other approaches also used.) –Keystroke time: Performance will be an issue until everything computed on a processor can happen in "Keystroke time"- time to press a key on the keyboard.

Want high IPC with realizable hardware, preferably scalable as more ILP extraction techniques become available. Have opportunity: in few years, billion transistor chips available. In 1997 Computer: what to do with a billion transistors? No one really knew; situation hasn't changed, until now, until Levo.

Caveat about hardware efficiency. Efficiency is in the eye of the beholder, e.g., 32-bit bit-serial adder vs. 32-bit parallel ripple carry or carry-lookahead, etc., adder: former is much more efficient, but the costlier latter is always used; has always been architecture trend. Less power consumption when less used; other methods will deal with power issues, not our focus.

Levo presented: realizes high IPC with distributed non-centralized scalable hardware, makes good??? use of billion transistor chip. Resource flow model.

2 High IPC Problems

Cost, delay (cycle time) and I-performance (IPC) issues (with big I-window (= our E-window) ???

2.1 High Cost

- of large Reorder Buffers and other techniques is prohibitive and not scalable; typical cost is $O(n^2)$ for dependency checking/enforcement, etc. (n is size of reorder buffer &/or I-window). Cite (Who?, CONDEL.)

2.2 Unscalable Microarchitecture

- long buses, etc. lead to longer cycle times, reduced overall performance. Centralized resources awkward to use: long bus delays, prohibitively high number of ports (cite EV8).

2.3 Low IPC

- overall performance low. Prior work: need DEE AND MCD. Also use data value speculation for perhaps bigger gains (cite Gonzalez).

3 Levo High IPC Solutions and Description

(Includes some prior work; that directly related.)

Big picture: Dave K.'s I,E,M window picture, reversed. Details (follows order of problems listed in Section 2.) Standard processor, std. ISA. No compiler assistance necessary; for legacy code; with compiler assistance: future work.

3.1 Time Tags with Active Stations

Both **new** – Time tag idea, why is not O(n-squared) hardware, why only RAW dependencies enforced. State: same for memory operands, predicate operands. Low cost: O(n), n is number of instructions in E-window.

1) Related approaches: Ultrascalar, Metaflow (today's typical micro), Tomasulo
2) Example: time line showing must get closest previous value for minimal RAW dependency, correct operands – without Levo-isms.
3) Example: "Details" of Active Station, ala Dave M.'s figure, with example – with Levo-isms.

3.2 Constant-Length Result Buses

Short Distance Same-Cycle Value Propagation – Observation from multiscalar: most instructions' results used soon after creation (near, in program time)

1) *Constant length spanning buses:* segmented version of CDB – **new –** column at time, use (R,M,P)FU's – big picture of Levo microarchitecture (mine, without interleaving or DEE or many RFU's or any MFU's or any PFU's shown).*Register-less datapaths:* no centralized resources. – **new**

3.3 IPC Enhancement Methods

Levo realizes DEE static tree heuristic with less than 10% cost overhead. MCD realized with hardware predication, also cheap. Data value speculation also cheap. (Is it? Estimates? TBD.)

1) *DEE – Disjoint Eager Execution* – **implementation new, -** describe, show revised sharing group with DEE AS's. Include release heuristic, if necessary.*Hardware Predication* – **new –** describe: composite technique: mine, Ali's, Dave's; describe canceling predicate. Example: modification of section 3.1-3) example above, with closest result predicate-disabled (branched-around); give corresponding code.*Data Value Speculation* – **implementation new –** describe, whatever it is. TBD.

4 Other Issues and Levo Solutions

This section needs to be kept small; it is not the main thrust of Levo. It supports the E-window, wherein lies the major novelty. This is here to keep the reviewers happy, for completeness, and to show other nice aspects of Levo.

1) *I-fetch* – Is an issue: getting correct instructions into E-window at high speed – Describe whatever we're doing here. Some TBD.

   a. Nominally static order fetch – keeps bandwidth high, holds complete (small) hammocks for good DEE operation.
   b. Heuristic – load in static order unless branch is predicted taken and target is far away (~greater than E-window instruction size)
   c. Loops hardware unrolled; backwards branches of up to last iteration loaded are converted to forwards branches

2) Memory latencies getting very large – This is basically addressed in the PACT paper in depth, as I recall; we can mention the high points here. Can also cite SSGRR2002s paper (which is to be on the Levo memory system; more on this later).
   a. Memory system – basically standard system; Levo deep window and ??? Gives high latency tolerance. – cite Smith? Was that paper published?
   b. Unusual components - L0 caches, TTWB, MFU. Briefly describe each; picture?

5 Levo Microarchitectural Diagram – Logical View

Memory and predicates treated about the same as register accesses; describe differences. My style picture, one bus shown for M-buses, some buses shown for R-buses, one bus for P-buses (with extension dots or dashes), show limited spanning bus and FU's, show column-to-column flow. Logically: shifting the columns left. Commit at left column; no separate r-file. Operation Example: big picture version incorporating prior mini-examples.

6 Levo – Physical Operation and Possible Floorplan

1) Columns renamed – no actual shifting – reduces power consumption, wiring complexity
2) Columns physically oriented end-to-end, to keep critical path length low. Easily scalable, even across chips. Show Floorplan. Mention high points, including support of high clock frequencies. What is shown is existence proof, not necessarily optimal layout.