LEVO

A Scalable Processor With High IPC

Augustus K. Uht
Dept. of Electrical and
Computer Engineering

&:

Alireza Khalafi
David Morano
David Kaeli
Dept. of Electrical and
Computer Engineering

μRI Road Show, Spring 2003

Acknowledgements

Work supported in present or past by:

- U.S. National Science Foundation
- URI Office of the Provost
- Intel
- Mentor Graphics
- Xilinx

Outline

1. Closely Related Work
2. Needs and LEVO Solutions
3. High-Level Architecture and Microarchitecture
4. Floorplan
5. Time-Tags and Active Stations, w/ Example
6. Resource-Flow Execution
7. Experiments
8. Summary
9. Some Future Work

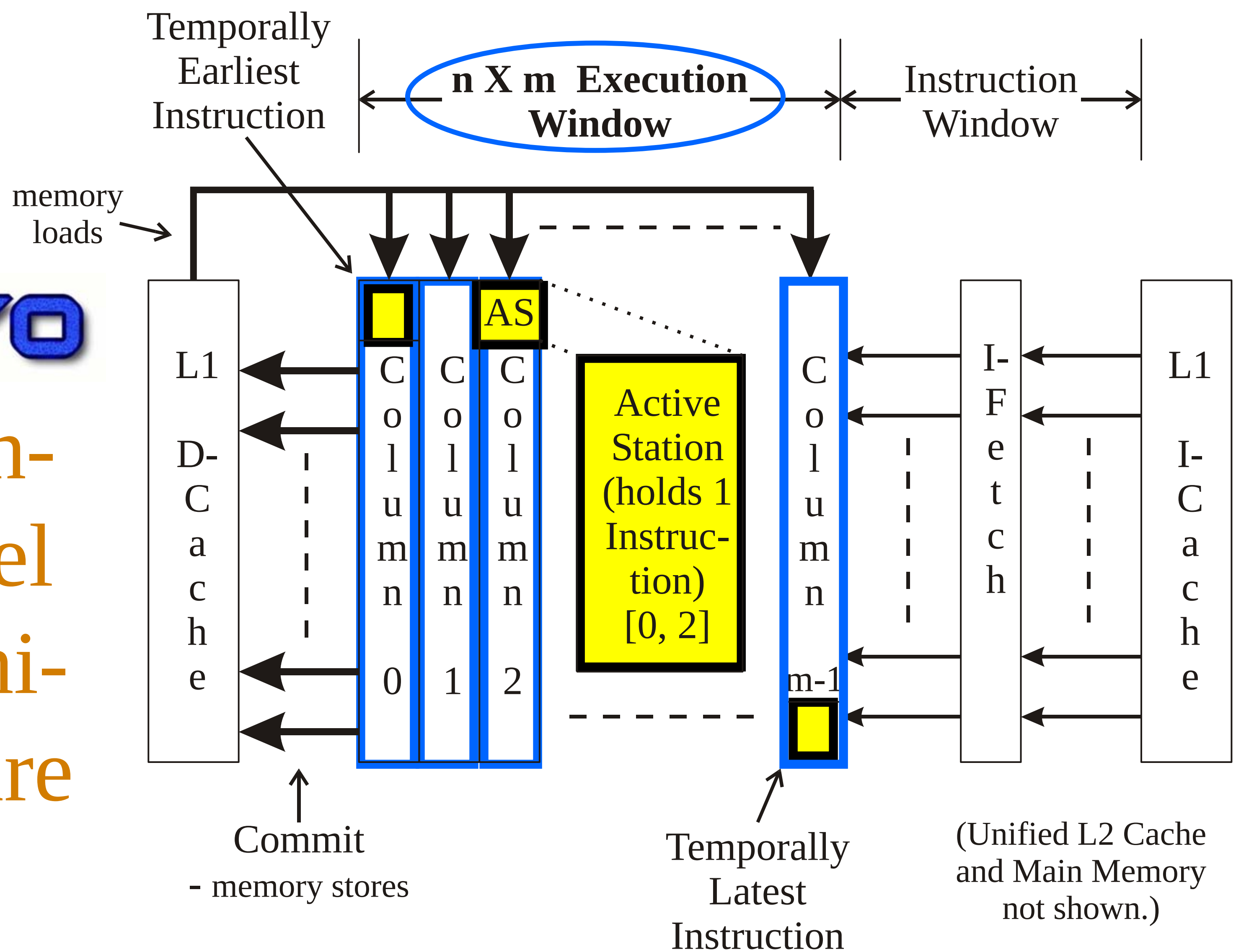
Closely Related Work

- Riseman & Foster (1972), Lam & Wilson (1992) and others (unconstrained resources):
  much ILP in General Purpose code: > x100
- But: *little IPC realized in real machines:* ~ 1-2
- Segmented IQ's – ISCA2002, etc.: don't scale, etc.
- Tomasulo '67 – elegant, but doesn't scale
- Limited register lifetime – Sohi et al '92
  - One key to Levo scalability
- Warp machine – Cleary et al '95 – *time-tags*
  - Memory accesses only, only for control-flow, large tags
- TRIPS – Burger et al, UT Austin – 'grid': software needed

Needs and LEVO Solutions

1. Cheap and scalable dependency detection & operand linking
   → ***time-tags*** (small): link & order operand usage.
2. Little cycle-time impact & scalability
   → constant length ***segmented*** or ***spanning buses***
3. Simple execution algorithm
   → ***resource-flow*** execution: Instructions flow to PE's, executed regardless of dependencies.
4. High IPC → *hardware predication* &
   Disjoint Eager Execution (DEE) - smart multipath
5. Legacy code → ISA independent, no compiler assist

High-Level Architecture



Processing Elements (PEs) are distributed among AS's.

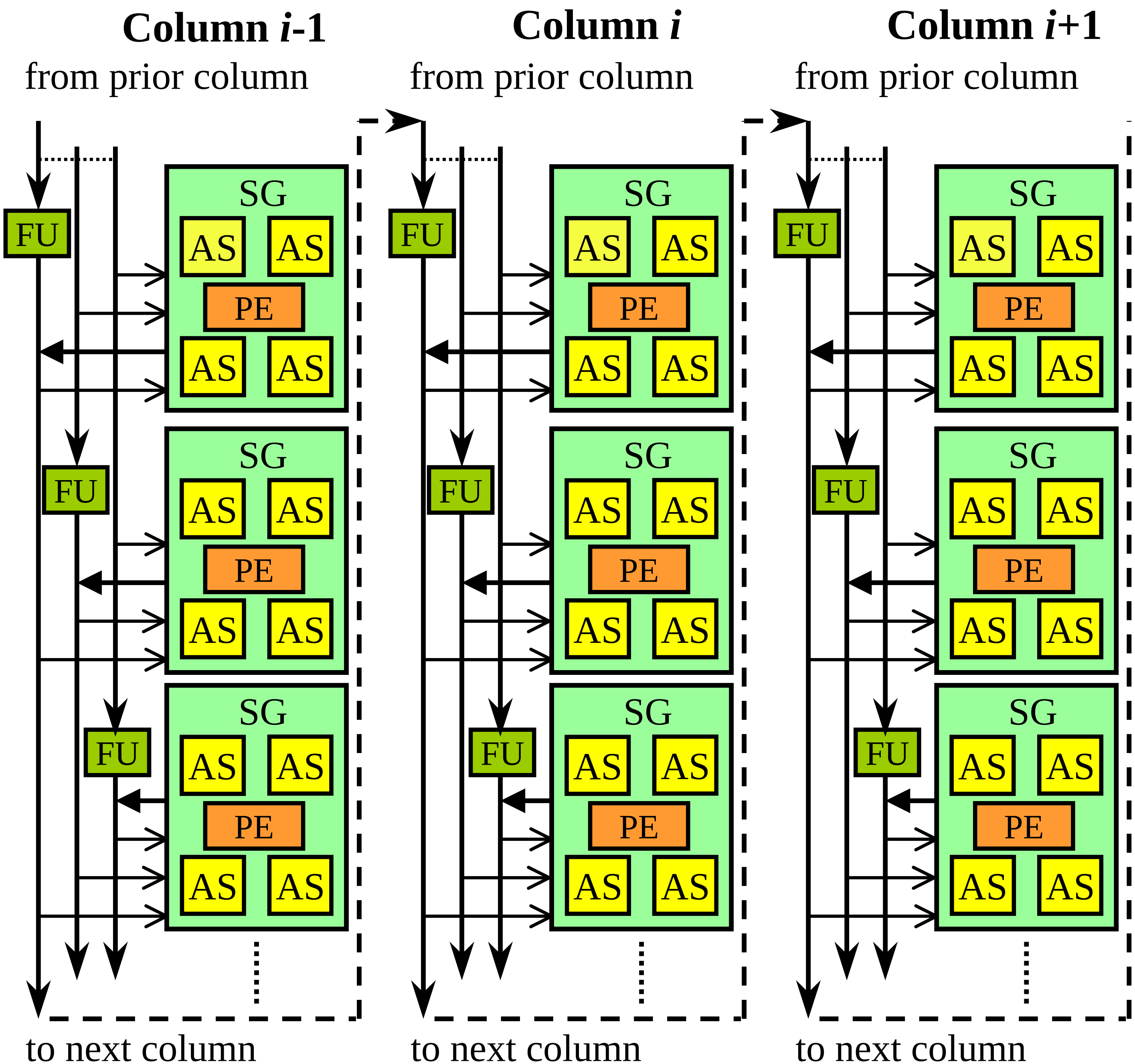
Micro-architecture

(Execution Window)

•No central register file: Reg. Fwd. Units used

•No reorder buffer

•SG: Sharing Group

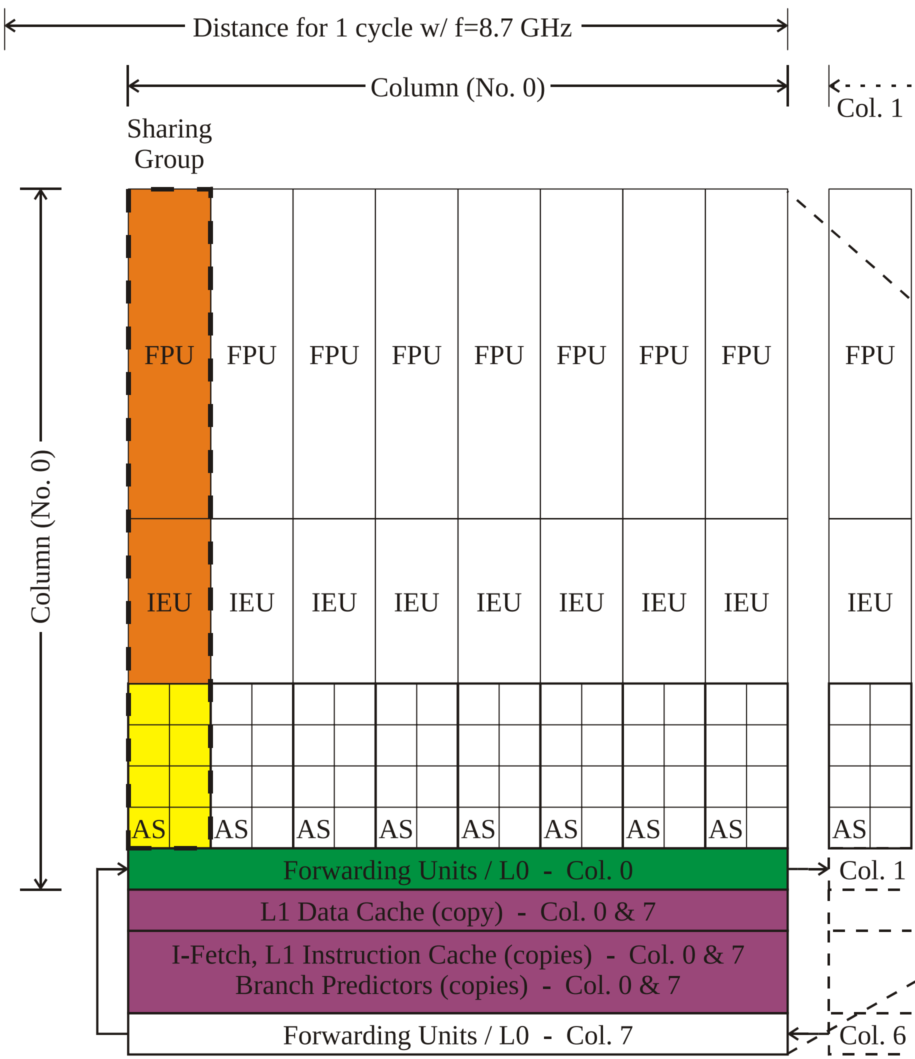


LEVO Floorplan: 8-4-8

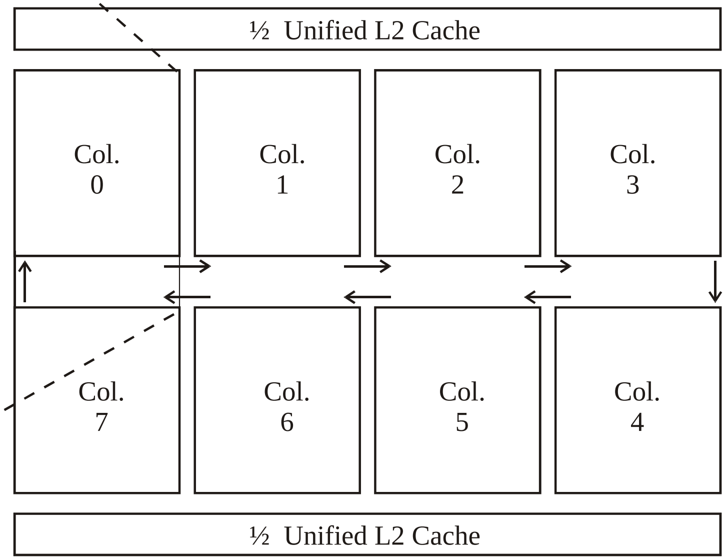
(SG/Column - AS/SG - # Columns)

- PE for Sharing Group
- 8 Active Stations of Sharing Group
- Forwarding Units, Col. 0
- L1 Caches (copies), Col. 0

– Partial expanded view

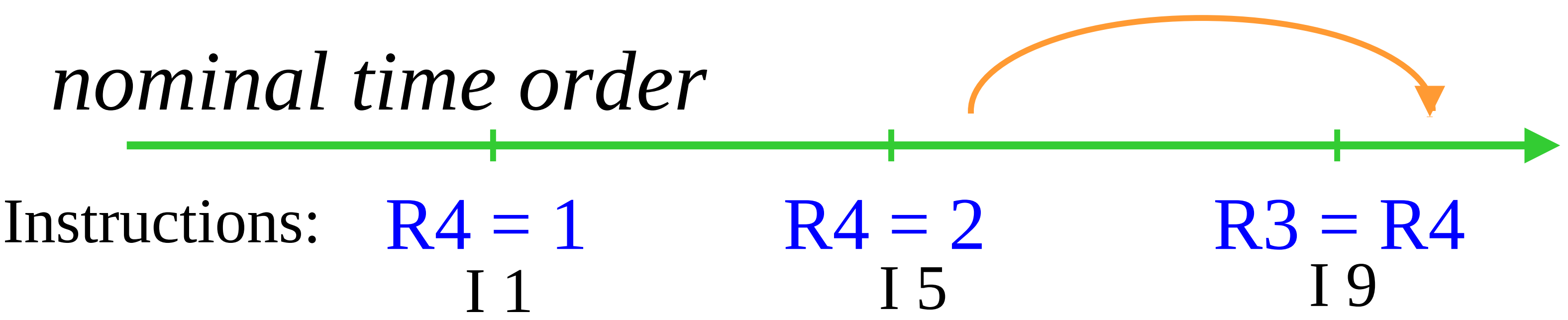


Overall chip:



Time-Tags

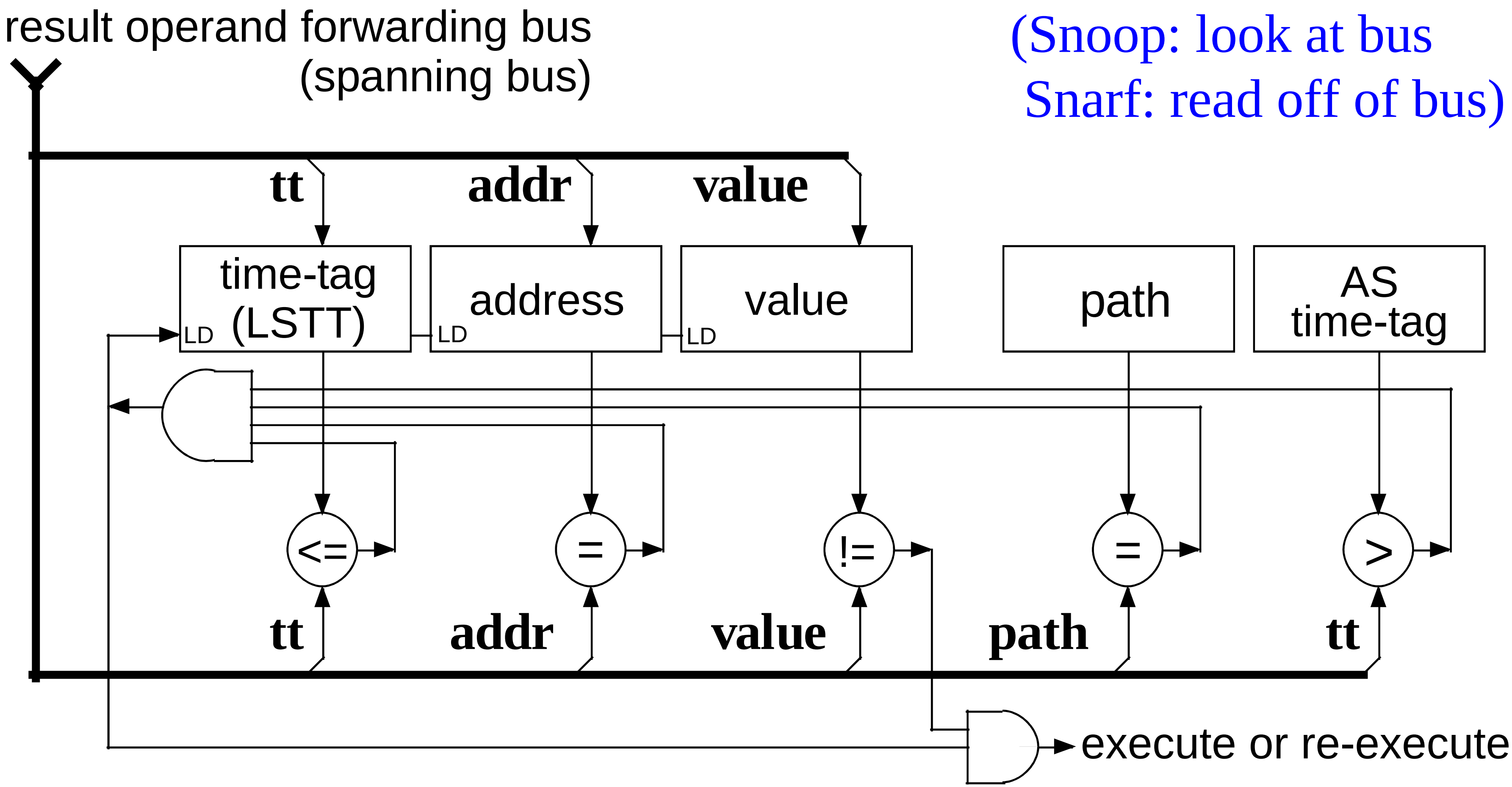
- Small integers = position in E-window
- Enforce and minimize dependencies
- Provide operand linking (sink-to-source)
- Basic problem:
  - R3 must wind up with *closest previous value* of R4 (2)
  - Must be independent of execution order of instructions



Active Station (AS)

- LSTT (Last-Snarfed Time Tag) key for opnd. linking

(Snoop: look at bus
Snarf: read off of bus)



Time-Tag Example: Case 1

- Recall: R3 ← *closest previous value* of R4 (2)

nominal time order

Instructions: R4 = 1 (I 1) R4 = 2 (I 5) R3 = R4 (I 9)

Time – 1: I 1 brdcsts.
R4 address matches,
TT(I 1) >= LSTT(I 9),
I 1 info snarfed: R3=1

Time – 2: I 5 brdcsts.
R4 address matches,
TT(I 5) >= LSTT(I 9),
I 5 info snarfed: R3=**2**

Time-Tag Example: Case 2

- Recall: R3 ← <u>*closest previous value*</u> of R4 (2)

nominal time order

Instructions:	R4 = 1	R4 = 2	R3 = R4
	I 1	I 5	I 9

<u>*Time – 1*</u>: I 5 brdcsts. R4 address matches,
TT(I 5) >= LSTT(I 9),
I 5 info snarfed: R3=<u>**2**</u>

<u>*Time – 2*</u>: I 1 brdcsts. R4 address matches,
TT(I 1) < LSTT(I 9),
I 1 info **not** snarfed.

Resource-Flow Execution

What it is:

Execute everything, then clean up.

(Example of this: in prior TT example, if: I1, I5, I9 all execute in first cycle, then either Case 1 or 2.)

Or, more precisely:

Execute any instruction regardless of the presence of its operands or predicates, resources permitting, then apply programmatic constraints to obtain correct execution.

~33% performance gain.

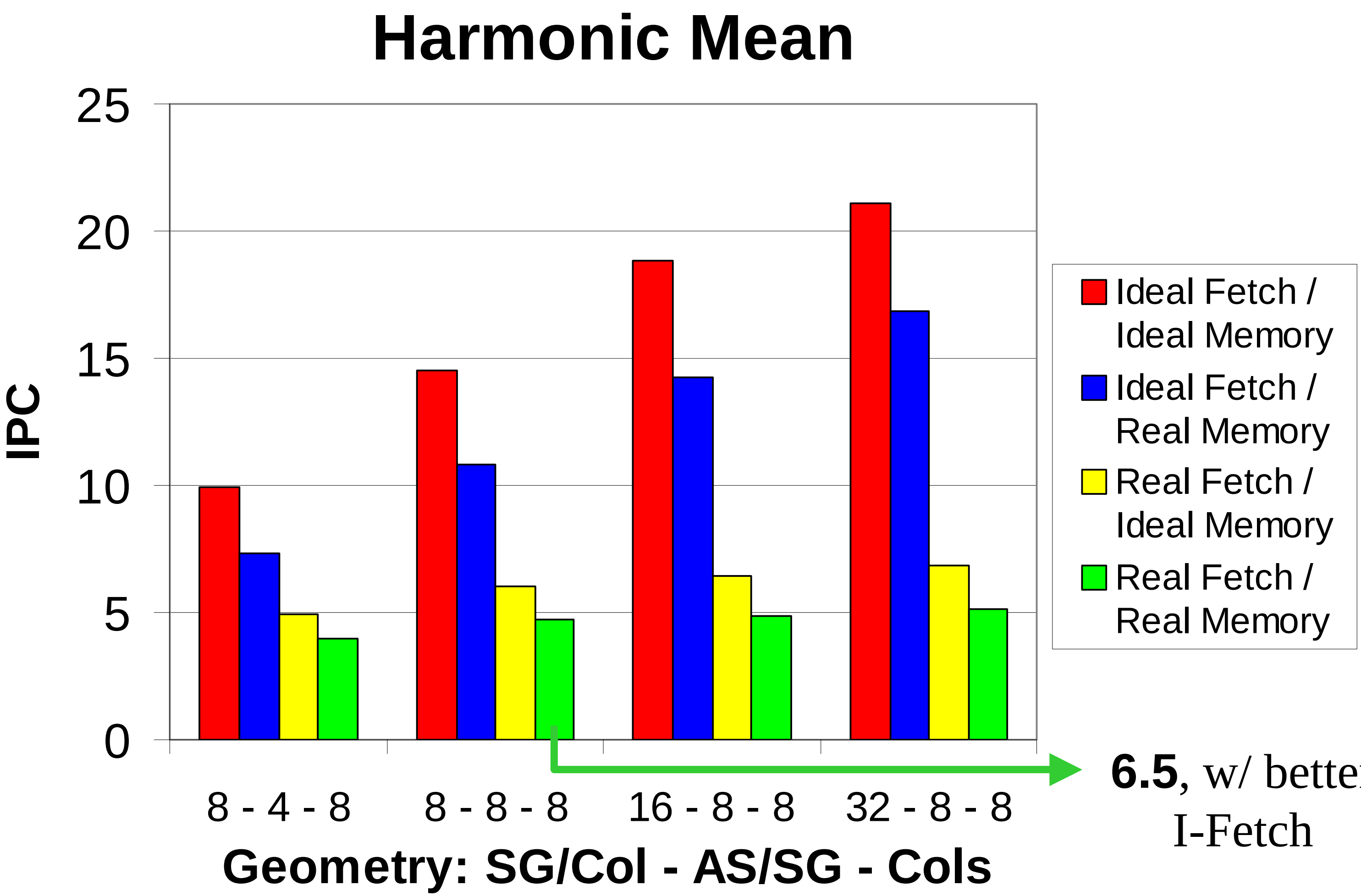
Experimental Methodology

- Trace-driven simulator used:
  - 100 million instruction warm-up, then:
  - 500 million instruction execution and data gathering
- MIPS-1 ISA binaries executed
- SPECint95: compress, go, ijpeg
  SPECInt2000: bzip2, crafty, **gcc**, gzip, mcf,
  parser, vortex – <u>10 benchmarks, total</u>
- *Point of comparison:* SimpleScalar –
  - w/ 32-way instruction issue (physically unrealizable)
  - Harmonic-mean IPC = <u>1.96</u>

Experimental Method. (cont.)

Parameter	Default Value
L1 D,I-caches	separate, 64 KB, 1 cycle latency
L2 cache:	unified, 2 MB, 10 cycle latency
Main memory:	100 cycle latency, no misses
Spanning bus delay	1 cycle (if no contention)
Spanning bus length	8 SG's (constant)
Forwarding Unit delay	1 cycle (if no contention)
Buses per RFU and per M(emory)FU	2 input and 2 output
Buses per P(redicate)FU	1 input and 1 output
M-path to D-path switch time	1 cycle (+ re-broadcast of data)
Columns per D-path	1 column

Real & Ideal Performance



6.5, w/ better I-Fetch

Summary

- LEVO:
  - New execution core
  - Novel techniques for scalability with low cycle time
  - **Time-Tags**, **Active Stations**, **Segmented Buses** & **Resource Flow Execution** are wins
  - High-IPC, & more there:
    - With better I-Fetch: much more IPC
    - With better data value prediction: much more IPC?

Some Future Work

- Try different form of value prediction:
  - *Distributed* throughout execution window
  - Put into Forwarding Units
  - Utilize local characteristics
  - Possibly utilize some global characteristics
- → Better value prediction
- Possibly 2-3x better ILP (Gonzalez *et al*)
  - 2-3x better IPC ?

LEVO

A Scalable Processor With High IPC

Augustus K. Uht
Dept. of Electrical and
Computer Engineering

University of Rhode Island
μRI @ URI Microarchitecture Research Institute University of Rhode Island

&:

Alireza Khalafi
David Morano
David Kaeli
Dept. of Electrical and
Computer Engineering

Northeastern University
NUCAR

μRI Road Show, Spring 2003