Realizing High IPC Using Time-Tagged Resource-Flow Computing

→ LEVO

Augustus K. Uht
Dept. of Electrical and Computer Engineering
University of Rhode Island

&

Alireza Khalafi
David Morano
Marcos de Alba
David Kaeli
Dept. of Electrical and Computer Engineering
Northeastern University

Euro-Par August 28, 2002

Acknowledgements

Work supported by:

- U.S. National Science Foundation
- URI Office of the Provost
- Intel
- Mentor Graphics
- Xilinx
- Ministry of Education, Culture and Sports of Spain (D. Kaeli)

Outline

1. Closely Related Work
2. Needs and LEVO Solutions
3. High-Level Architecture and Microarchitecture
4. Time-Tag Example
5. Resource-Flow Execution
6. High-IPC Multipath Method Example
7. Experiments
8. Summary

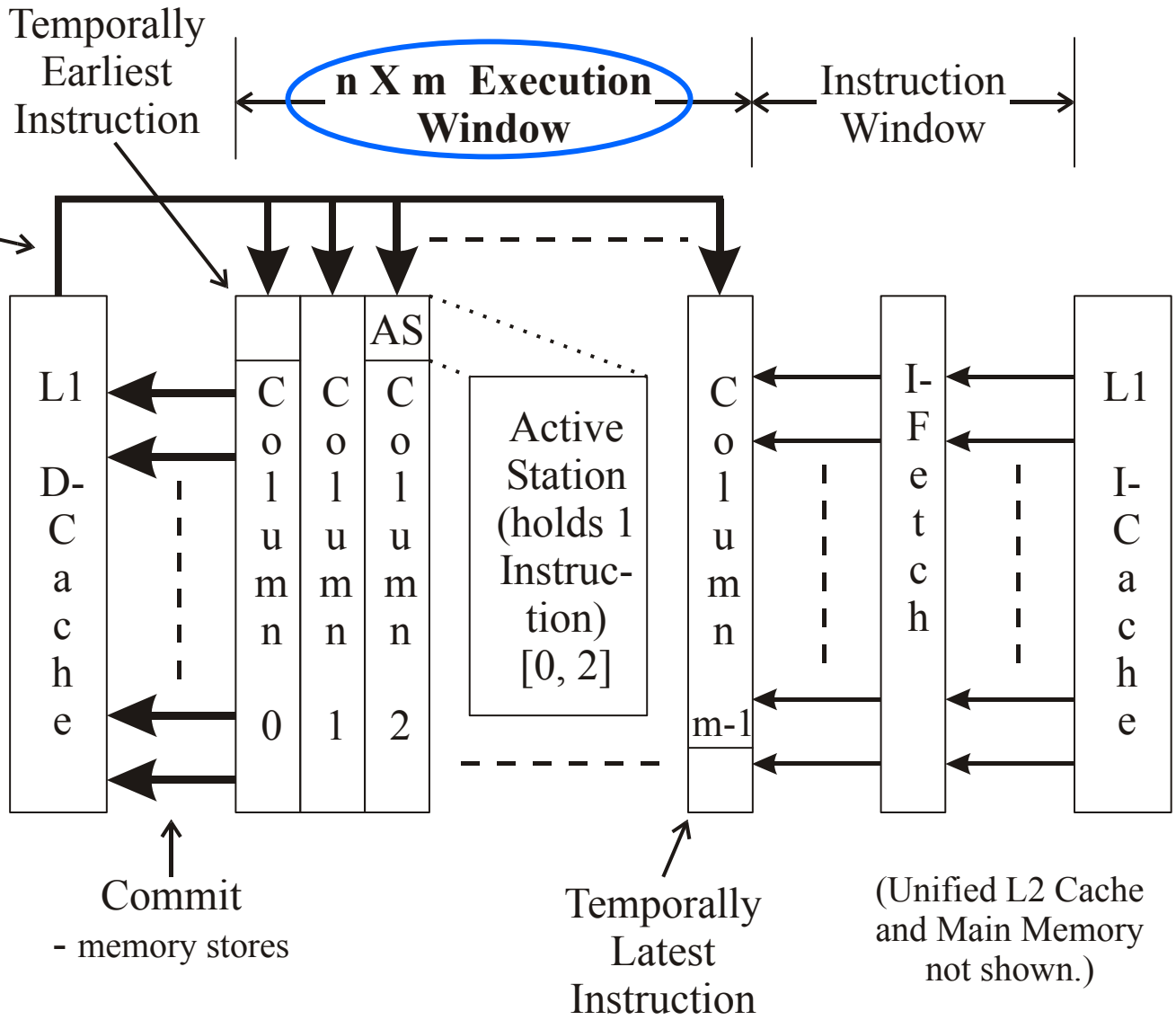
Closely Related Work

- 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,
  in dispatch stage, PE's not distributed; we predate.
- 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*
  - Basic idea good, but used floating-point tags

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

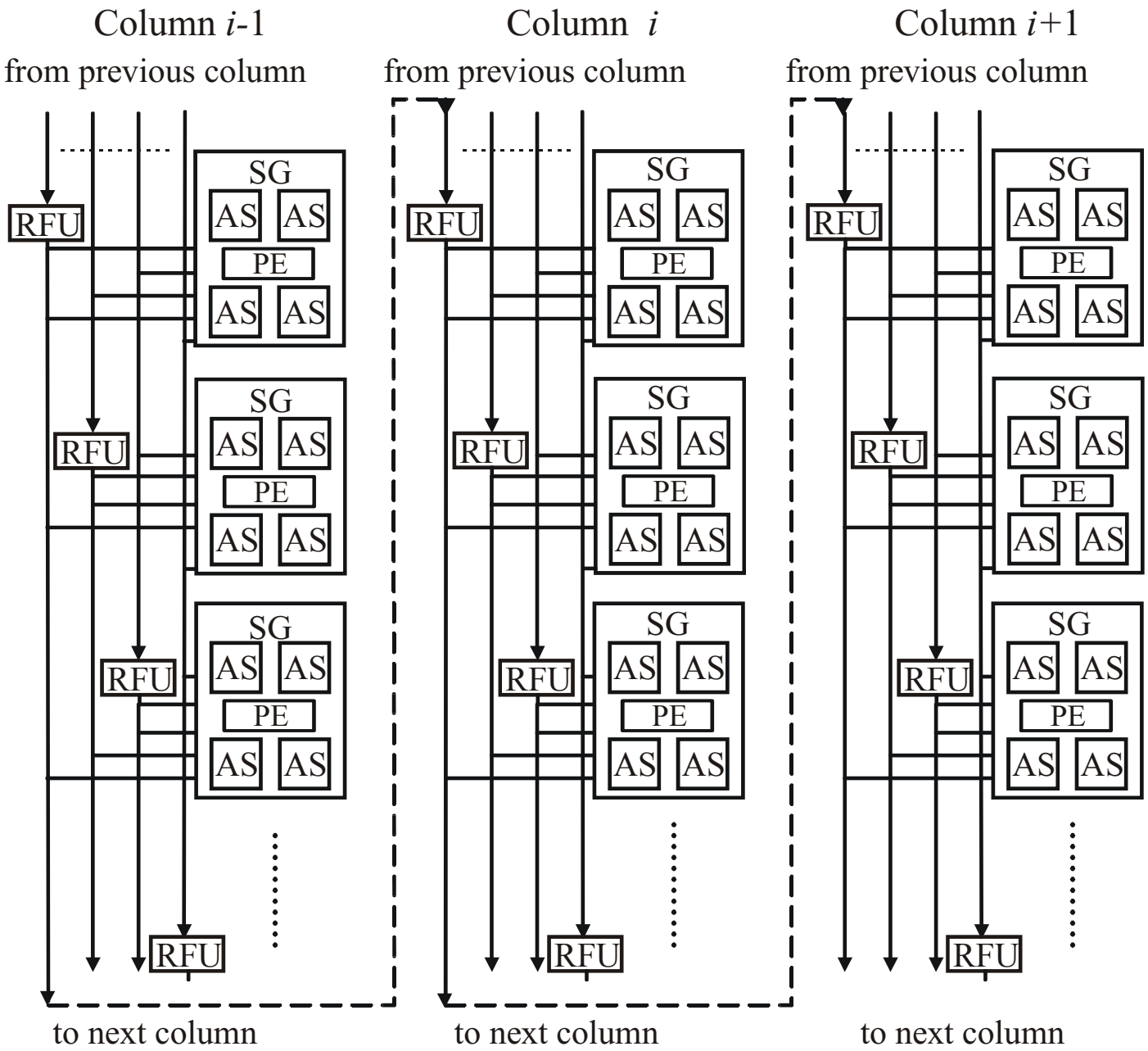
High-Level Architecture



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

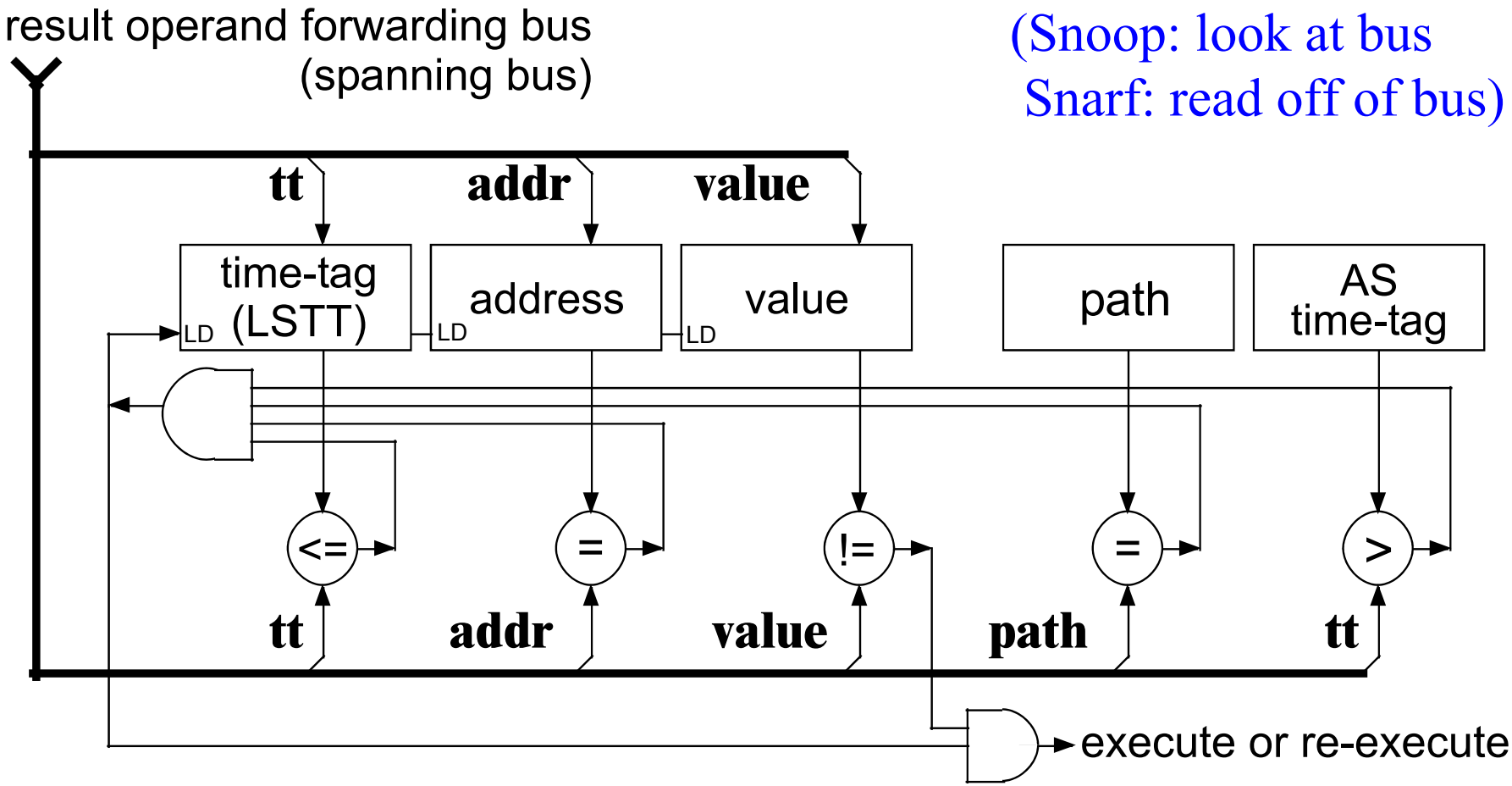
Micro-architecture (Execution Window)

- Note: no central register file: Reg. Fwd. Units used
- SG: Sharing Group



Active Station (AS)

- LSTT (Last-Snarfed Time Tag) is key to operand linking



Time-Tag Example

Instruction Number		Instruction, Result Time Tag (ResTT)		Last Snarfed Time Tag (LSTT). In Active Station.
1.	R4 = 1	1	R4 = 1	–
5.	R4 = 2	5	R4 = 2	–
9.	R3 = R4	9	R3 = R4	1, then 5

Sequential Execution (at end, R3 holds '2')

Out-of-Order (OOO) Execution.

Case 1
- I1 result and ResTT broadcast,
  – R3 = 1, LSTT = 1
- I5 result and ResTT broadcast,
  – R3 = 2, LSTT = 5

(at end, R3 holds '2')

Case 2
(Same result if I5 broadcasts first;
LSTT is set to and stays at '5';
I1 result not snarfed by I9.)

(i) Program Code

(ii) With Time Tags

Time-Tag Example

I9 needs closest previous value of R4. Case 1:

Broadcast (I1):

TT: 1 R: 4 V: 1

↓ bus

AS (I9): Snoop and Snarf:

TT >= LSTT, R=ADDRESS:

LSTT -1 → 1, VALUE → 1

Instruction, Result Time Tag (ResTT)		Last Snarfed Time Tag (LSTT). In Active Station.
1	R4 = 1	–
5	R4 = 2	–
9	R3 = R4	1, then 5

Out-of-Order (OOO) Execution.

- I1 result and ResTT broadcast,
  - R3 = 1, LSTT = 1
- I5 result and ResTT broadcast,
  - R3 = 2, LSTT = 5

(at end, R3 holds '2')

(Same result if I5 broadcasts first; LSTT is set to and stays at '5'; I1 result not snarfed by I9.)

(ii) With Time Tags

Time-Tag Example

Instruction, Result Time Tag (ResTT)		Last Snarfed Time Tag (LSTT). In Active Station.
1	R4 = 1	–
5	R4 = 2	–
9	R3 = R4	1, then 5

Broadcast (I5):

TT: 5 R: 4 V: 2

↓ bus

AS (I9): Snoop and Snarf:

TT >= LSTT, R=ADDRESS:

LSTT 1 → 5, VALUE → 2

DONE (Case 1):
R3 =2

Out-of-Order (OOO) Execution.
- I1 result and ResTT broadcast,
  – R3 = 1, LSTT = 1
- I5 result and ResTT broadcast,
  – R3 = 2, LSTT = 5

(at end, R3 holds '2')
(Same result if I5 broadcasts first;
LSTT is set to and stays at '5';
I1 result not snarfed by I9.)

(ii) With Time Tags

Time-Tag Example

I9 needs closest previous value of R4. Case 2:

Instruction, Result Time Tag (ResTT)		Last Snarfed Time Tag (LSTT). In Active Station.
1	R4 = 1	–
5	R4 = 2	–
9	R3 = R4	~~-1, then~~ 5

Broadcast (I5):

TT: 5 R: 4 V: 2

↓ bus

AS (I9): Snoop and Snarf:

TT >= LSTT, R=ADDRESS:

LSTT -1 → 5, VALUE → 2

DONE:
R3 =2

Out-of-Order (OOO) Execution.
- I1 result and ResTT broadcast,
  – R3 = 1, LSTT = 1
- I5 result and ResTT broadcast,
  – R3 = 2, LSTT = 5

(at end, R3 holds '2')

(Same result if I5 broadcasts first; LSTT is set to and stays at '5'; I1 result not snarfed by I9.)

(ii) With Time Tags

Time-Tag Example

Broadcast (I1):

TT: 1 R: 4 V: 1

bus

*AS (I9): Snoop and **NO** Snarf:*

TT < LSTT, R=ADDRESS:

LSTT stays at 5,
VALUE stays at 2.

I9 already has closest previous value (Case 2):
Already DONE: R3 =2

Instruction, Result Time Tag (ResTT)		Last Snarfed Time Tag (LSTT). In Active Station.
1	R4 = 1	–
5	R4 = 2	–
9	R3 = R4	~~1, then~~ 5

Out-of-Order (OOO) Execution.
- I1 result and ResTT broadcast,
  – R3 = 1, LSTT = 1
- I5 result and ResTT broadcast,
  – R3 = 2, LSTT = 5

(at end, R3 holds '2')
(Same result if I5 broadcasts first;
LSTT is set to and stays at '5';
I1 result not snarfed by I9.)

(ii) With Time Tags

Resource-Flow Execution

What it is:

Execute everything, then clean up.

(Example of this: in last set of slides, 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.

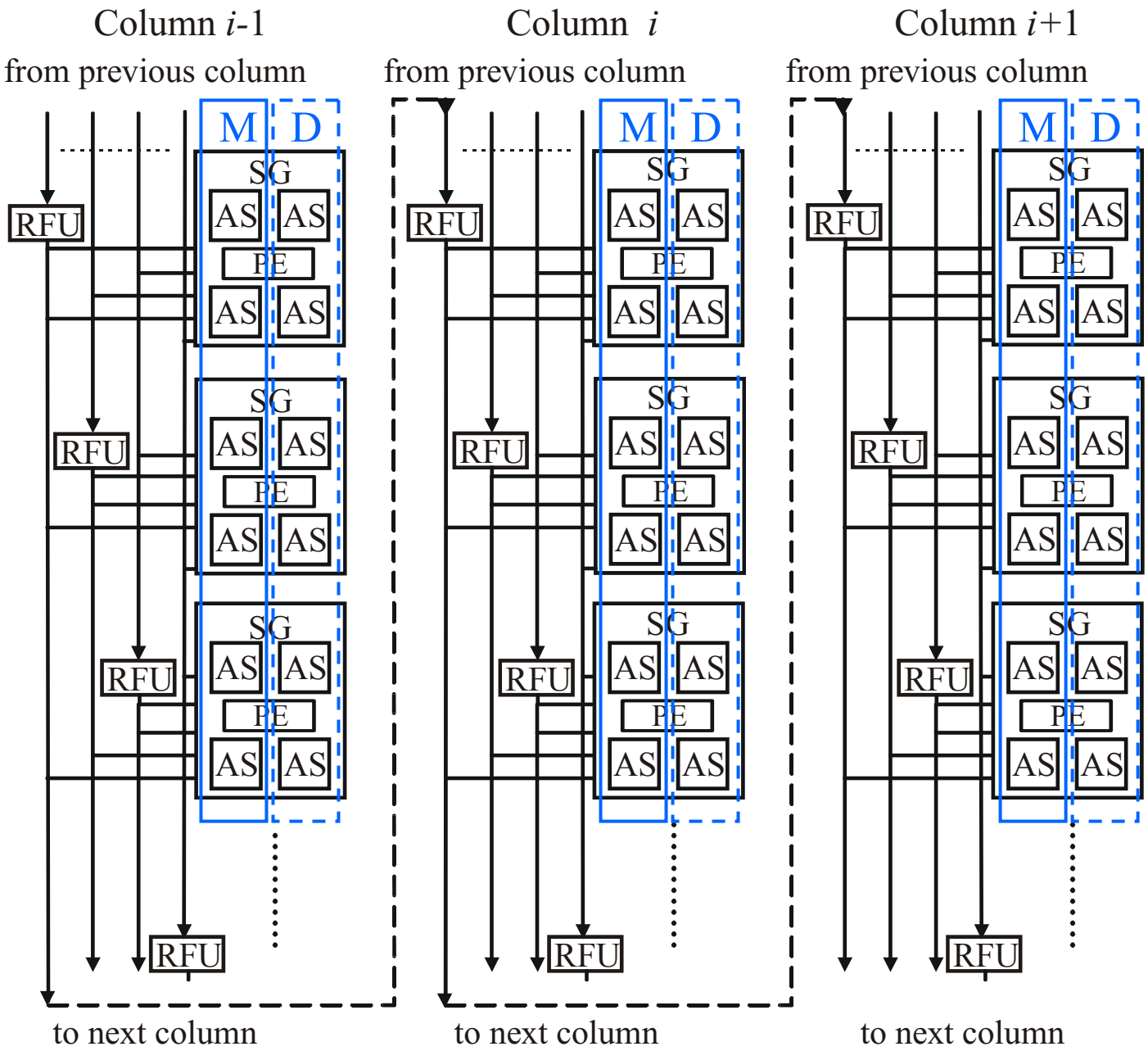
High-IPC Methods

- *Hardware predication*:
  - Predicates generated with hardware
  - Branch domains determined with hardware

- *D-paths*: multipath execution based on DEE
  - *Not-predicted path* of some branches executed just-in-case; has lower priority for resources

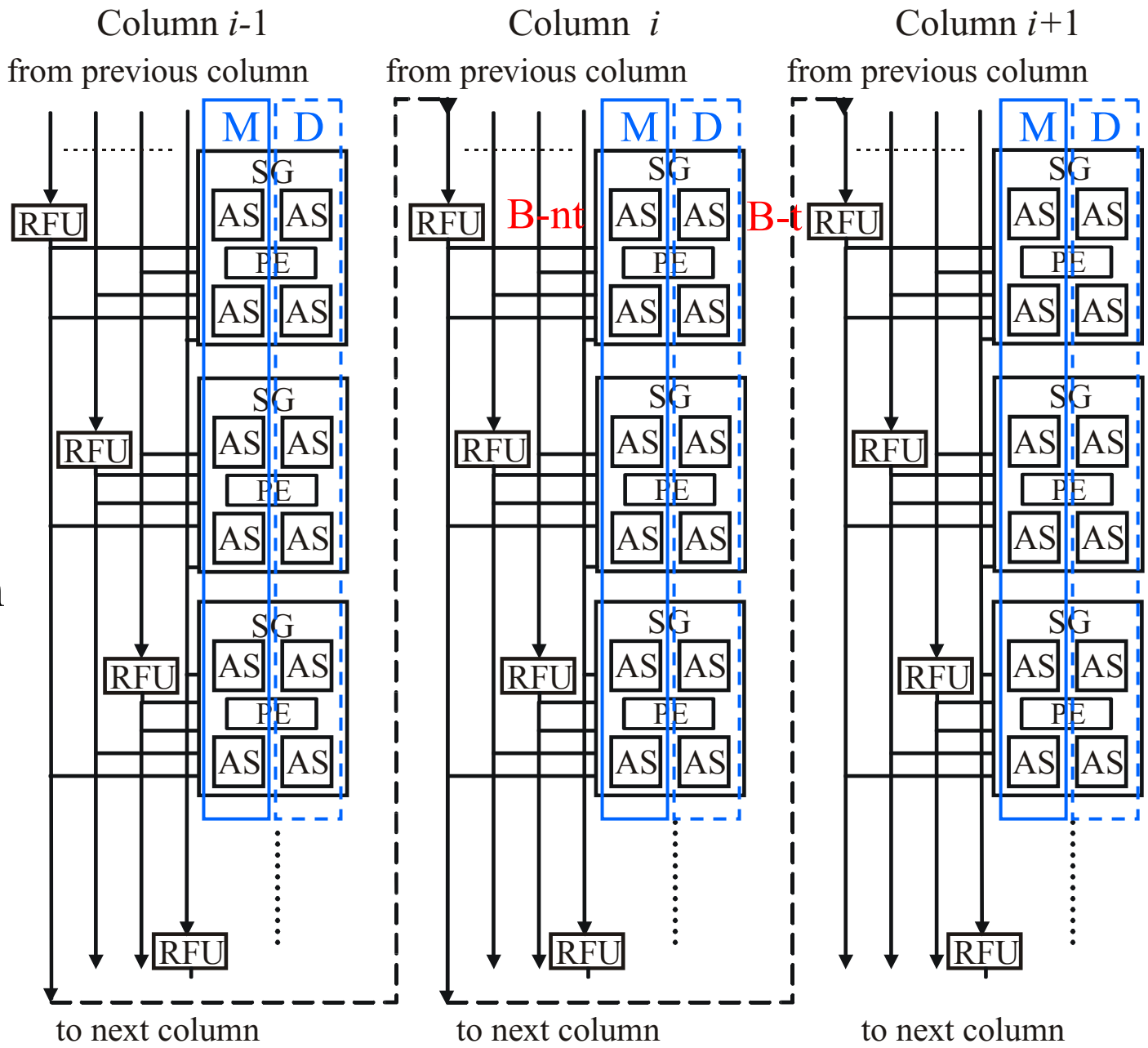
Micro-architecture (Execution Window)

- "M" – Mainline Path
- "D" – DEE Path



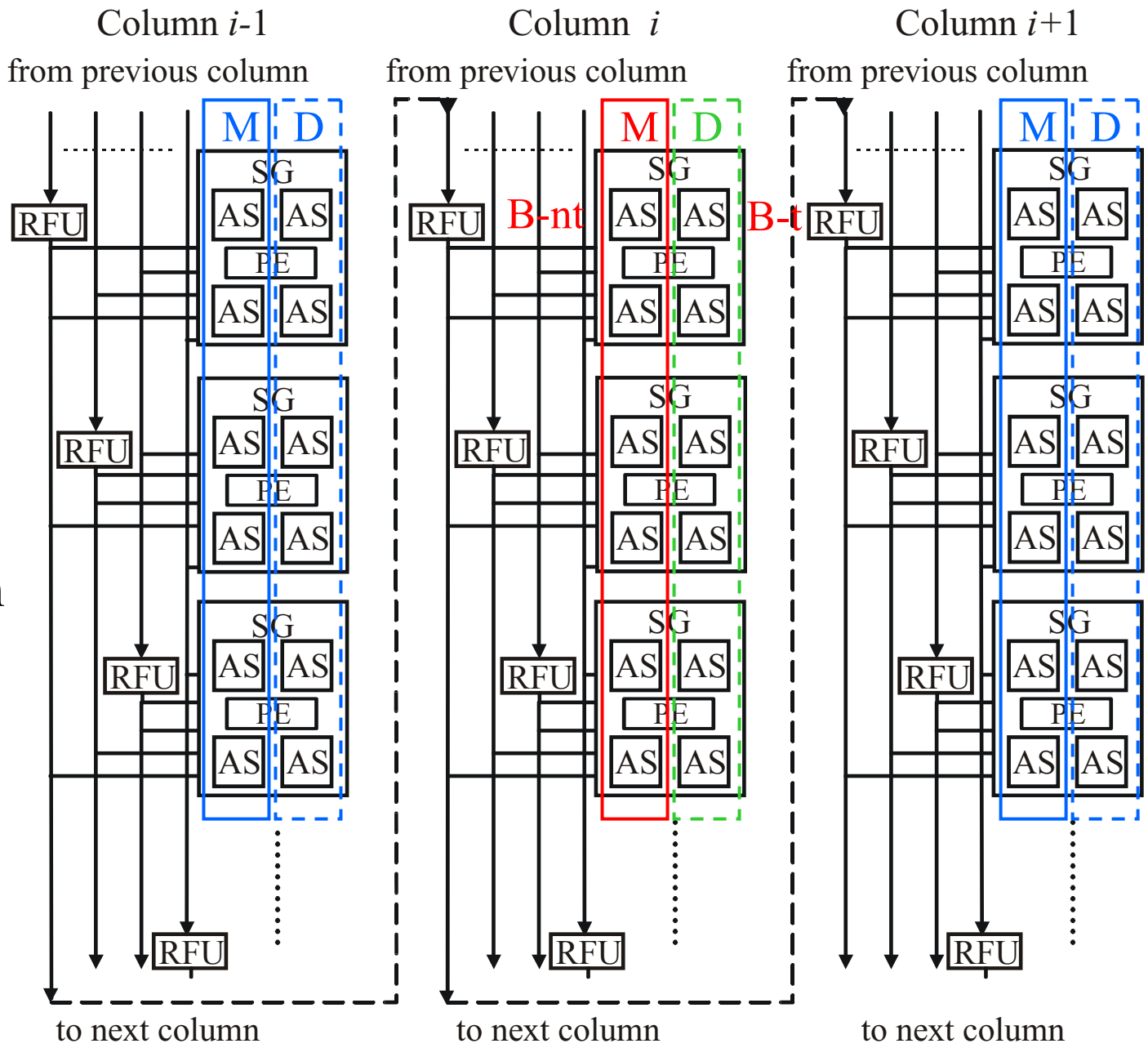
Micro-architecture

- "M" – Mainline Path
- "D" – DEE Path
- "B-nt" – Branch pred. not taken



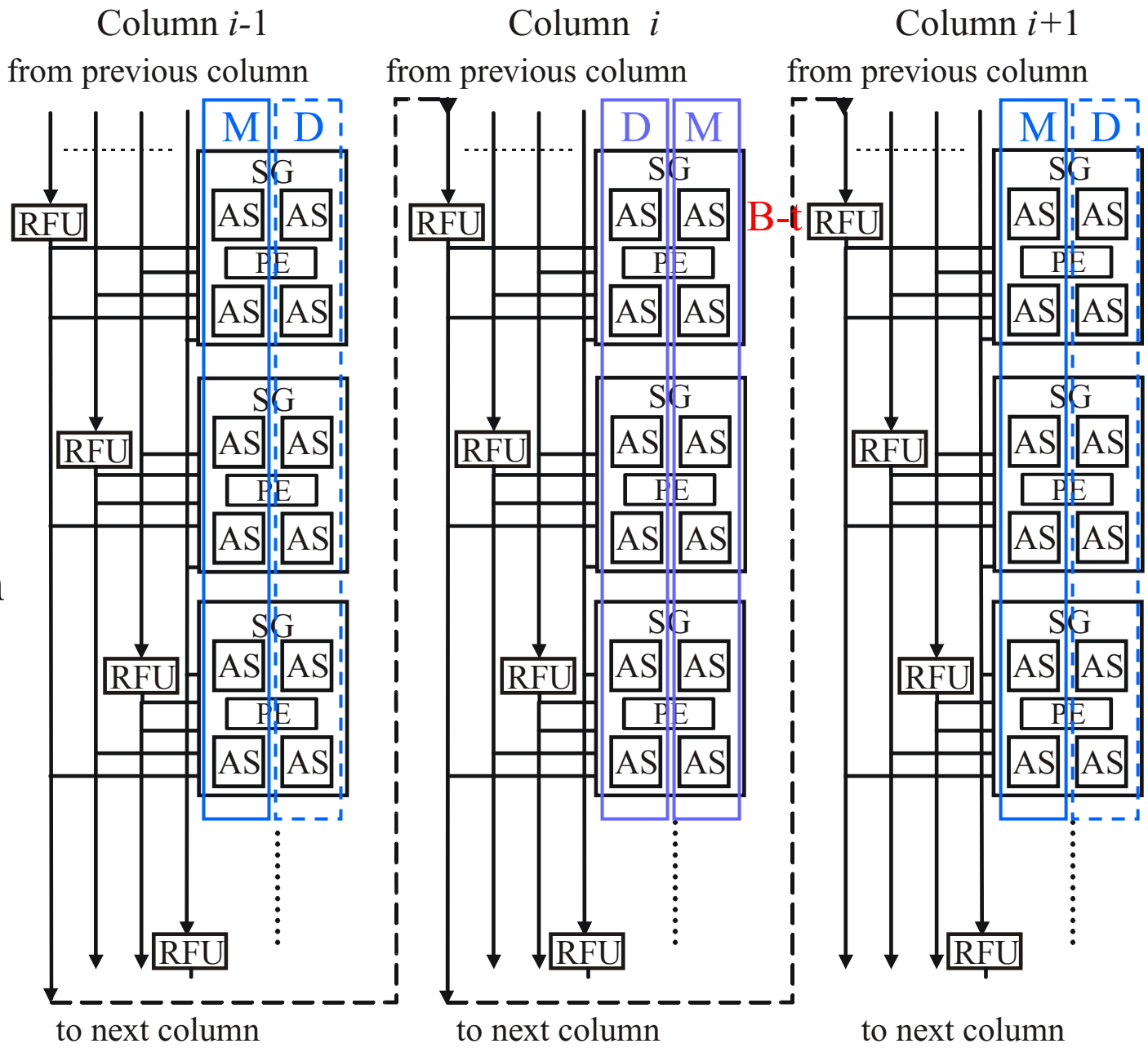
Micro-architecture

- "M" – Mainline Path
- "D" – DEE Path
- "B" – Branch mispredicted



Micro-architecture

- "D" → "M" Mainline Path
- "M" → "D" DEE Path
- "B-t" – Branch now pred. taken



Experimental Methodology

- Trace-driven simulator used
- MIPS-1 ISA binaries simulated
- Five SPECint95 and SPECint2000 benchmarks simulated
- L1 D-cache: 1 cycle hit, 10 cycles miss
- L1 I-cache, L2, memory: perfect (100% hit)
- *Baseline Machine (BM)*: bound by true dependencies, no time-tagging, no resource flow, no D-paths.
- *BM-CM*: baseline with Conventional Memory

Experiments

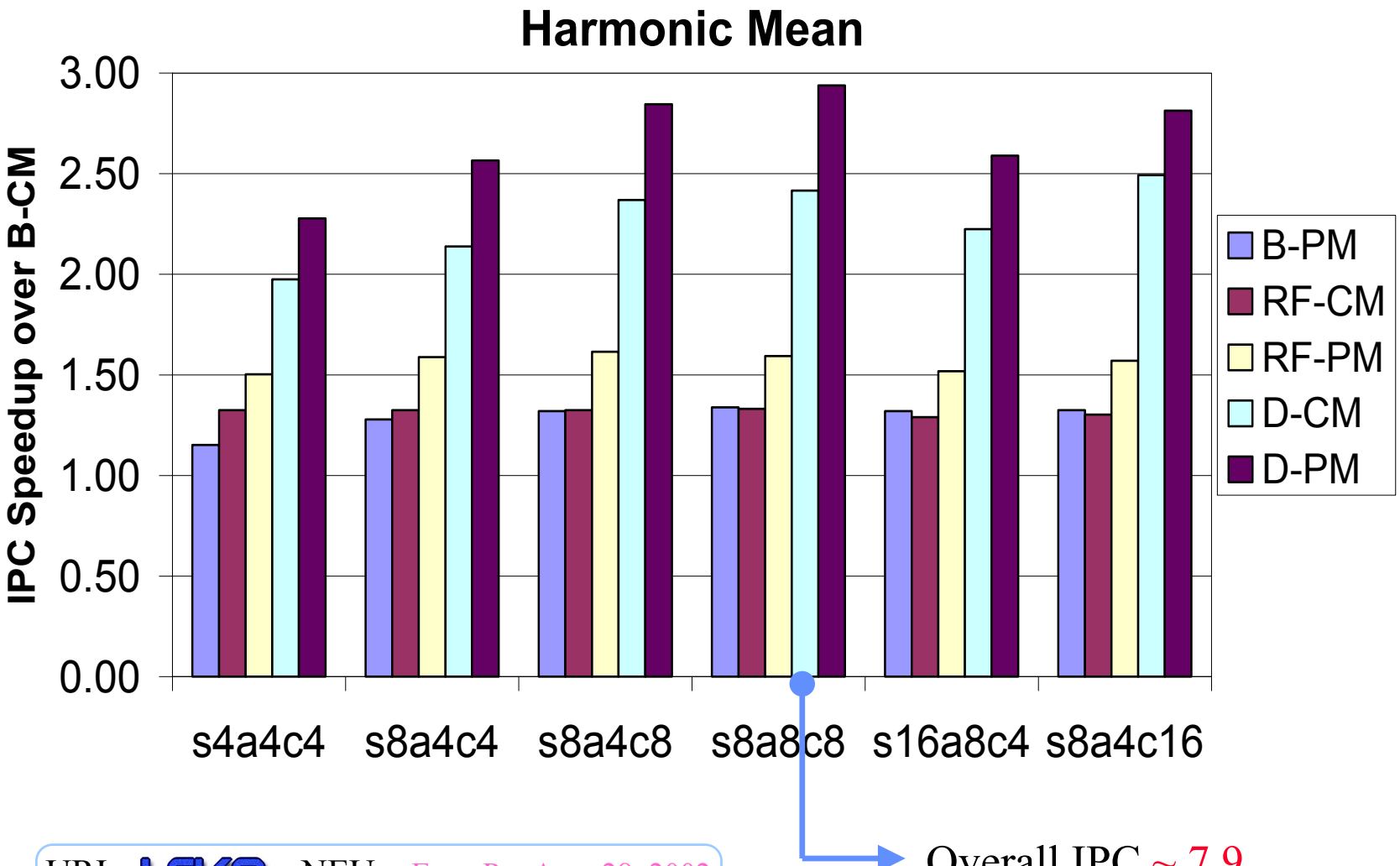
- Varying machine configuration:
  s(SG's/column) a(M-path AS's /SG) c(columns)
  [*c* is # M-path columns,
  is also # D-path columns when present]
- *CM vs. PM* (Perfect Memory: 100% L1 hit)
- *BL*: baseline – no resource flow, no D-paths
  vs. RF: w/resource flow but no D-paths
  vs. D: w/resource flow and D-paths

Raw IPC vs. Configuration

Machine Config	SGs per Column	ASs per SG	Columns	gzip BL-CM IPC	gap BL-CM IPC	parser BL-CM IPC	bzip BL-CM IPC	go BL-CM IPC
s4a4c4	4	4	4	2.3	2.4	1.8	1.9	1.7
s8a4c4	8	4	4	2.8	3.5	2.4	2.5	2.4
s8a4c8	8	4	8	2.9	3.9	2.5	2.7	2.5
s8a8c8	8	8	8	4.1	4.4	2.7	2.9	2.7
s16a8c4	16	8	4	3.1	3.9	2.5	2.6	2.5
s8a4c16	8	4	16	3.1	4.2	2.5	2.5	2.5

Levo machine configurations and BL-CM IPC values for the 5 benchmarks.
s = SGs per column, a = ASs per SG and c = Columns.

Speedups vs. Config. & Machine Type



Summary

- LEVO:
  - New execution core
  - Novel techniques for scalability with low cycle time
  - **Time-Tags** & **Resource Flow Execution** are wins
  - High-IPC, & more there:
    - D-CM with branch oracle: about 50% more IPC
  - Conventional memory IPC close to perfect memory
  - D-paths quite effective at improving performance

Relevant Web Sites

Levo links:

www.ele.uri.edu/~uht

Or: www.levo.org

Levo visualization (direct):

ovel.ele.uri.edu:8080