Exploring Instruction Level Parallelism using Resource Flow execution

student **David Morano**

committee **Professor David Kaeli**
Professor Xinping Zhu
Professor Waleed Meleis

Northeastern University
Computer Architecture Research

07/04/26

introduction

- **goal: we want to make programs execute faster!**
  - SMP, multicore, multi-thread do not make single programs faster !
  - Explicit Parallel Instruction Computing (EPIC) has not delivered on its promise
- **what is needed for speeding up single programs?**
  - more CPU resources needed to execute a single program thread
  - larger CPU ICs needed for more execution resources
  - future CPU resources will necessarily be more dispersed on the IC and distributed
- **requires a microarchitecture for managing large numbers of parallel and distributed resources**
  - handling OoO parallelism
  - a machine that can physically scale

our approach

- **constraints for our approach**
  - binary compatibility to existing ISAs (like x86 !)
  - means dynamic instruction scheduling
- **we want to explore OoO execution with**
  - breaking control dependencies
  - breaking data depedencies
  - using time-tags as the dependency enforcement mechanism
  - allowing for easy re-executions without pipeline flushes
- **philosophy**
  - just execute everything we can in parallel
  - clean up afterwards through re-executions for commitment

potential ILP performance

Lam and Wilson, "Limits of Control Flow on Parallelism" ISCA-19, 1992 (Stanford)

using the Spec-92 benachmarks they evaluated the limits on the amount of possible parallelism in the programs

some upper-bound results:

mean speed-up	execution constraints
1.0	instructions execute one after the other
2.14	an instruction cannot execute until its immediately preceding branch is resolved
6.80	instructions execute when their immediately preceding mispredicted branch is resolved
39.62	instructions execute when their mispredicted control dependent branches (all of them) resolve
158.26	all branches are perfectly predicted and all instructions are allowed to speculatively execute ahead (Oracle)

striving towards the potential

- **in order to get the big gains, we must have a machine that allows (according to Lam and Wilson) :**
  - the elimination of all but true data dependencies
  - speculative execution beyond just a speculatively executed branch
  - independent regions of code must be allowed to execute speculatively and simultaneously (minimal control dependencies)
  - use of some sort of predicated instruction execution (guarding)
  - execution of multiple paths simultaneously
- **we also break data dependencies**
  - form of last value prediction

machine concepts we will use

- **resource flow computing (new)**
  - execution proceeds primarily on resource availability rather than data or control dependencies !
- **hardware branch predication (new)**
- **wide instruction window and issue (old)**
  - "A 21st century microprocessor may well [issue] up to dozens of instructions [per cycle, peak] ..." -- Dave Patterson, 1995
- **issue station concept (new -- extension of a reservation station or RUU)**
  - instructions re-execute as necessary until retired
- **execution (resource) sharing groups (new and old)**
- **result forwarding buses -- extended Common Data Buses (Tomasulo)**
- **high-bandwidth interleaved memory interface (old)**
- **memory and register filter units (scalability and bandwidth conservation)**
- **physical machine scalability** (need for numbers of resources)
- **disjoint eager execution (old)**

outline

- **background**
- Resource Flow execution
- the OpTiFlow microarchitecture
- Levo -- a physically scalable microarchitecture
- conclusions

other ILP proposals (1)

- **MultiScalar -- Franklin, et al**
  - compiler-directed attempt to partition a program into multiple successive "tasks"
  - not binary compatible with anything -- requires a new architecture
  - tasks actually somewhat resemble "threads" in current processors (separate stacks) -- but only a single user program is being executed
  - IPCs average about 2.9 over integer sequential benchmarks (8 x OoO 2-way issue processors)
- **Parallel Execution Windows (PEWs) -- Kemp, Franklin**
  - employs a set of "execution windows" (EW), each of which has an independent instruction window to hold several instructions
  - data dependencies must be determined before dispatch to an EW
  - instructions are assigned to an EW based on their data dependencies, data dependent groups are assigned to the same EW
  - IPCs between 2 and 4 achieved for the largest machines investigated

other ILP proposals (2)

- **Superspeculative Microarchitecture -- Lipasti, Shen**
  - builds on existing speculative machine techniques
    - aggressive branch prediction
    - trace cache (new at the time)
  - various values are predicted:
    - data operand dependencies
    - source operand value prediction is employed (registers and memory)
    - memory alias prediction (to attempt to parallelize memory disambiguation)
  - an IPC of about 7.3 is achieved on the Spec-95 benchmark suite
- **UltraScalar -- Henry, et al**
  - "How I learned to stop worrying and love out-of-order execution"
  - consists of a reusable circular ring of execution stations
  - execution stations somewhat resemble an RUU (Sohi) entry combined with a processing element
  - primary feature is a large operand switching network to forward operands; O(log W), W is number of execution stations -- "asymptotic complexity"

outline

- background
- **Resource Flow execution**
- the OpTiFlow microarchitecture
- Levo -- a physically scalable microarchitecture
- conclusions

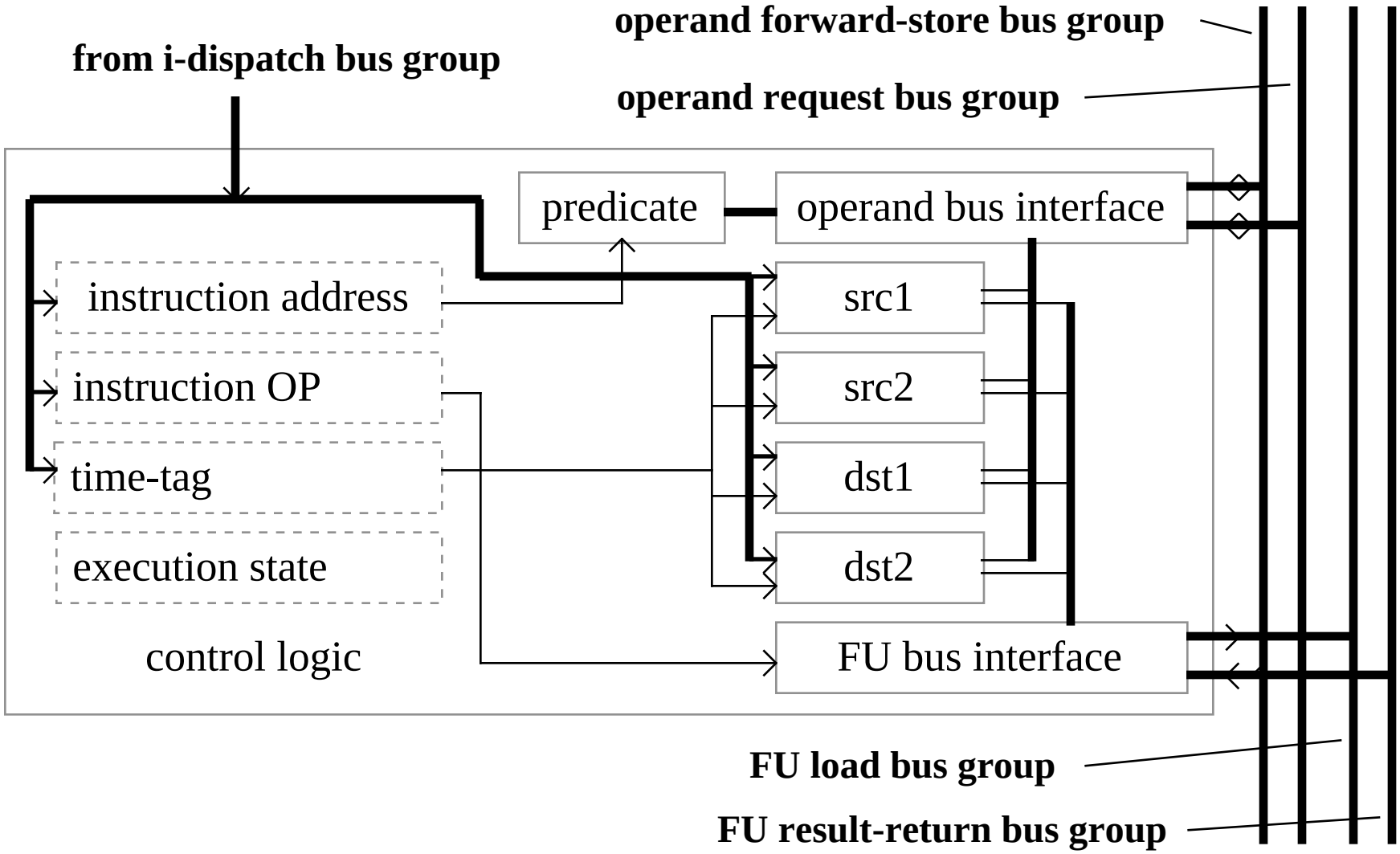
resource flow execution model

- **instruction dispatch and issue will not stall due to either control or data dependencies**
  - employs conventional branch prediction
  - employs somewhat less conventional value prediction
- **only the structural hazards of the machine (due to available machine "resources") will limit instruction and operand flow**
  - issue slots
  - execution units
  - buses
- **operands are elevated to first class entities**
- **program dependencies are determined dynamically during execution**
  - not during fetch or even at dispatch time!
  - eliminates cycle time slowdowns due to complicated dependency determinations
- **a means is introduced to manage operand flow and program dependencies**
  -- we will introduce the idea of the "issue station"

the issue station idea

- **an extension to the reservation station (Tomasulo) or RUU (Sohi)**
- **serves as a place to wait for operands and control flow changes**
- **holds an instruction until it is ready to retire**
- **snoops all result forwarding buses coming to it for new operands**
- **makes requests for operands on backwarding buses (register & memory)**
- **monitors control flow (prediction) changes by snooping for control-flow predicate operands (in Levo)**
- **re-executes an instruction when one or more of its operands change**
- **can squash its own execution with a "relay" operation (**restoring its output operand as if it had not executed**)**

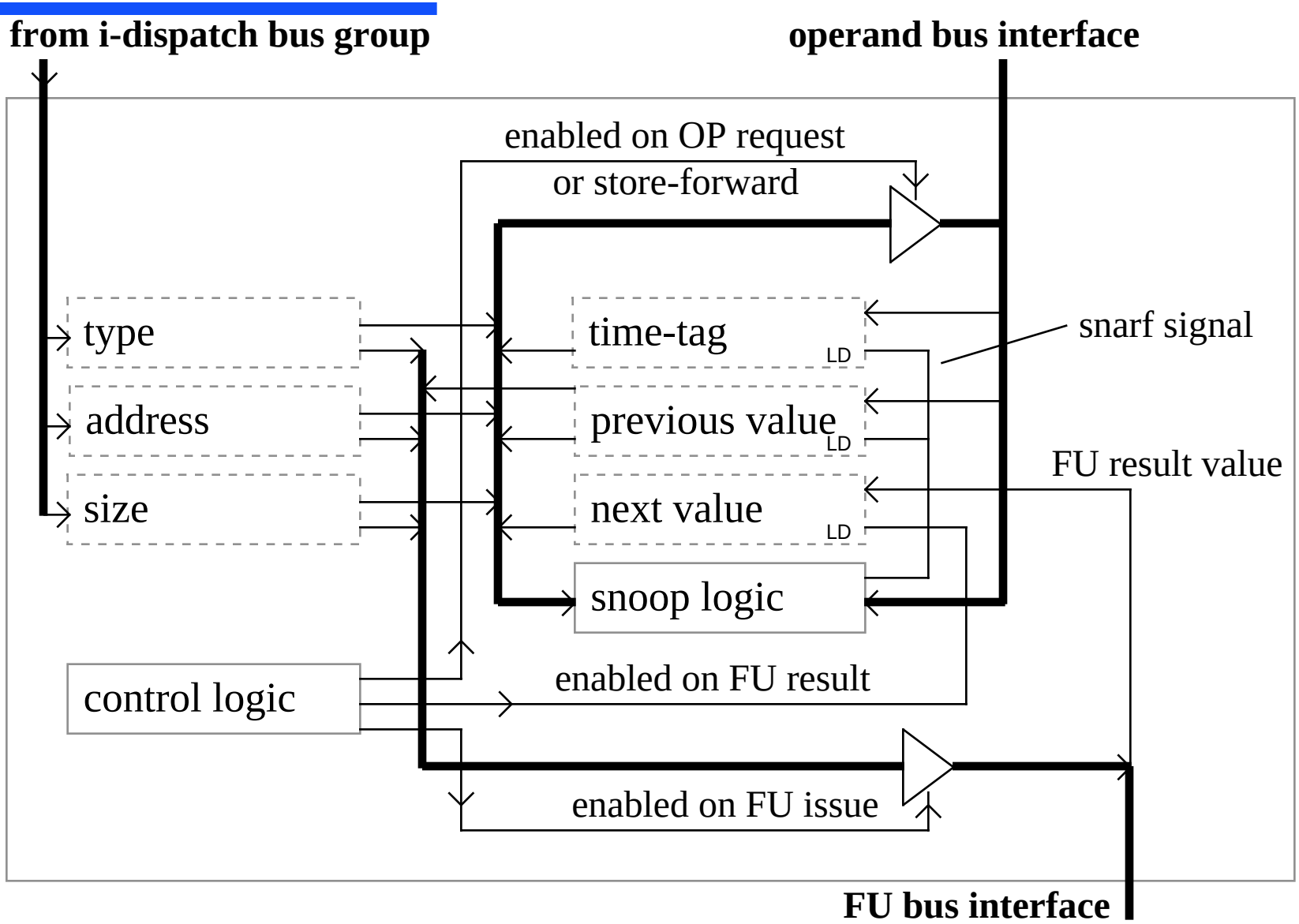
issue station



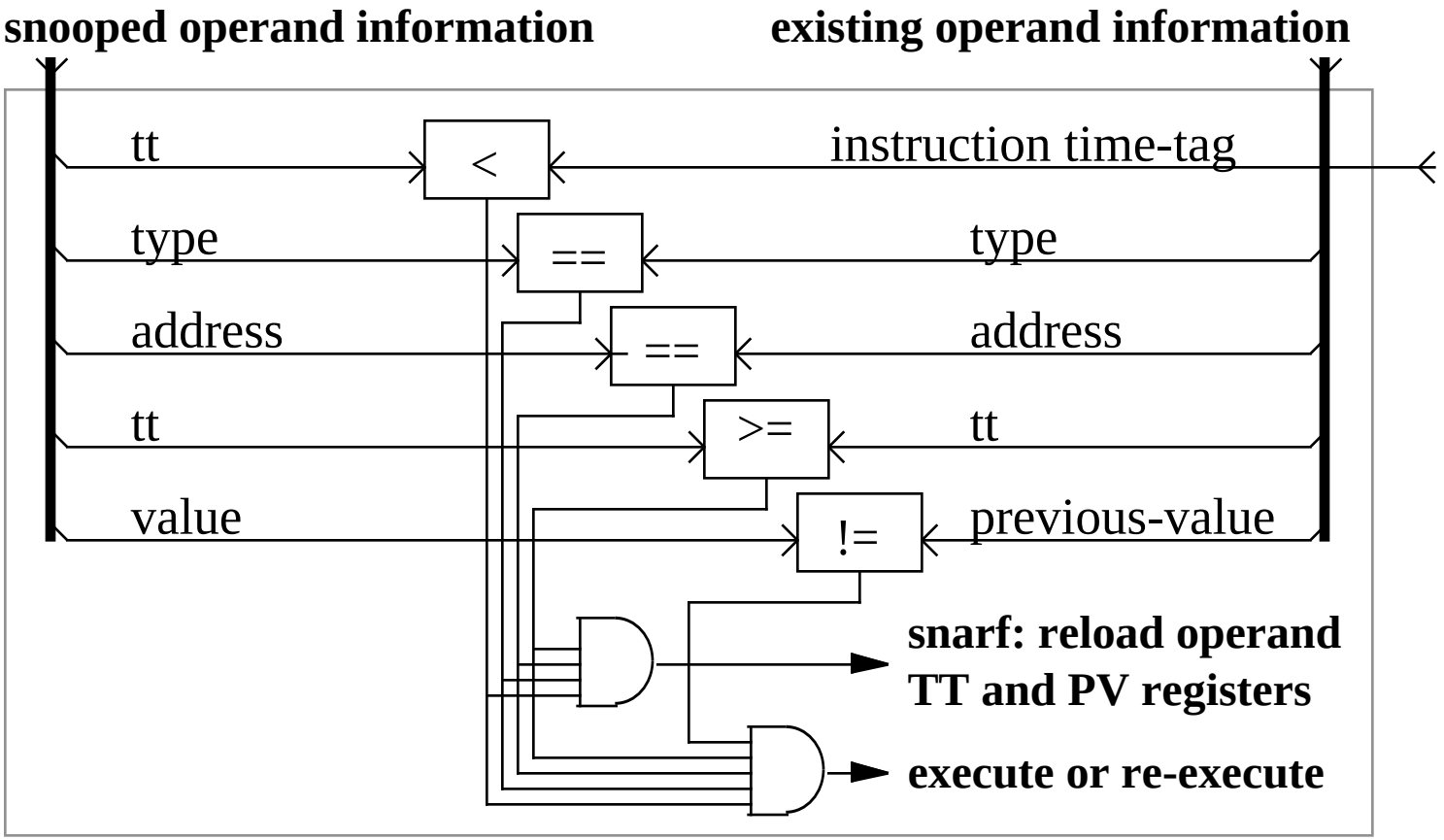
operand block (1)

- **holds all information about one operand**
  - either a register or memory operand
  - predicates are handled specially
- **includes necessary logic to snoop for operand updates**
- ∞ **operand "names" take the form** -- type : path : time-tag : seq : addr
  - type -- type of operand
  - path -- used as ID for multipath execution
  - time-tag -- time-tag of originating IS
  - seq -- operand sequence number used when operands can pass each other during forwarding operations
  - addr -- architected address of operand
- ∞ **example for a register** -- register : 1 : 27 : 3 : r6
- ∞ **predicates are operands also but have additional state**

operand block (2)



snooping logic for operands



example execution

code fragment

label	*TT*	*instruction*
i1	0	r3 <- 1
i2	1	r4 <- r3 + 1
i3	2	r3 <- 2
i4	3	r5 <- r3 + 2

example execution schedule

cycle	*execute*	*forward*	*snarf*
0	i1		
1		i1{r3=1}	i2{r3=1}, i4{r3=1}
2	i2, i4		
3	i3	i2{r4=2}, i4{r5=3}	
4		i3{r3=2}	i4{r3=2}
5	i4		
6		i4{r5=4|}	

- we want *r3* to have value =2 after execution of i3
- we want *r5* to have value =4 after execution of i4

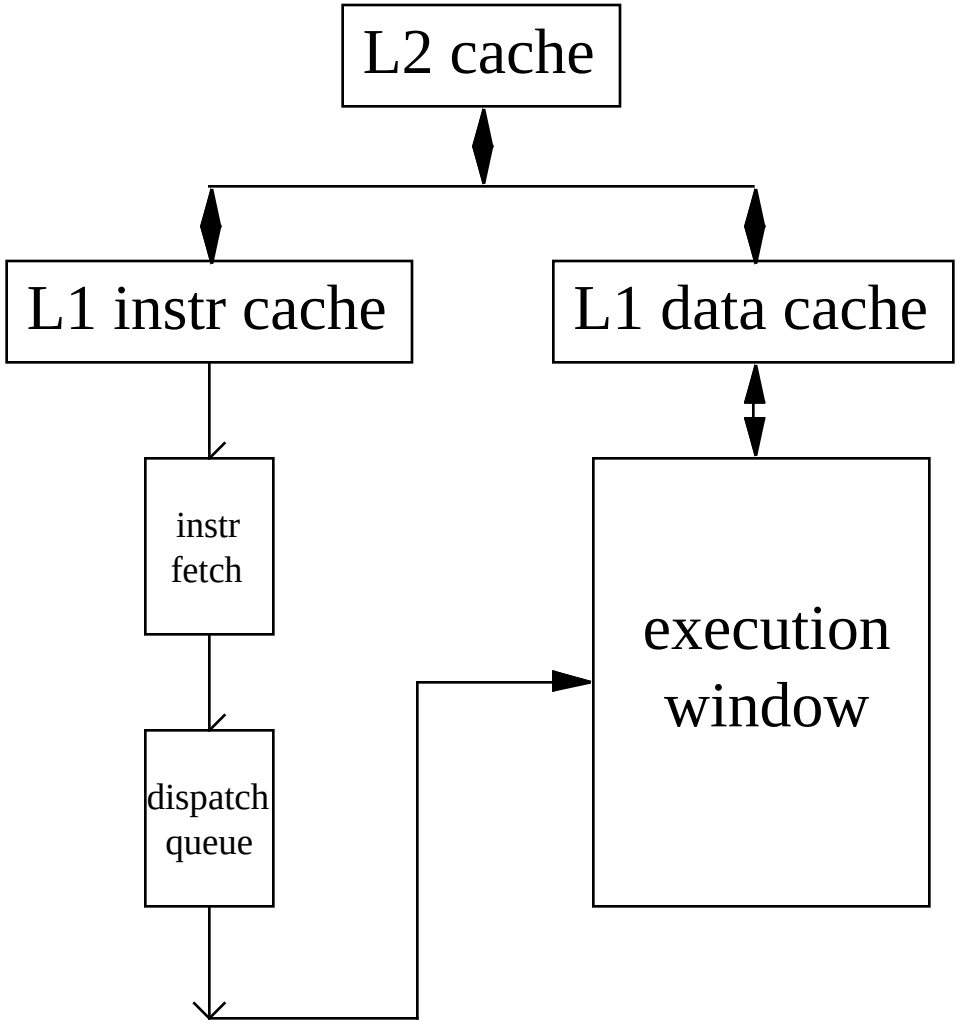
committed results

- i4 executes in clock 2 after snarfing *r3* from i1, resulting in *wrong* result
- i4 executes again after snarfing *r3* from i3, giving correct result

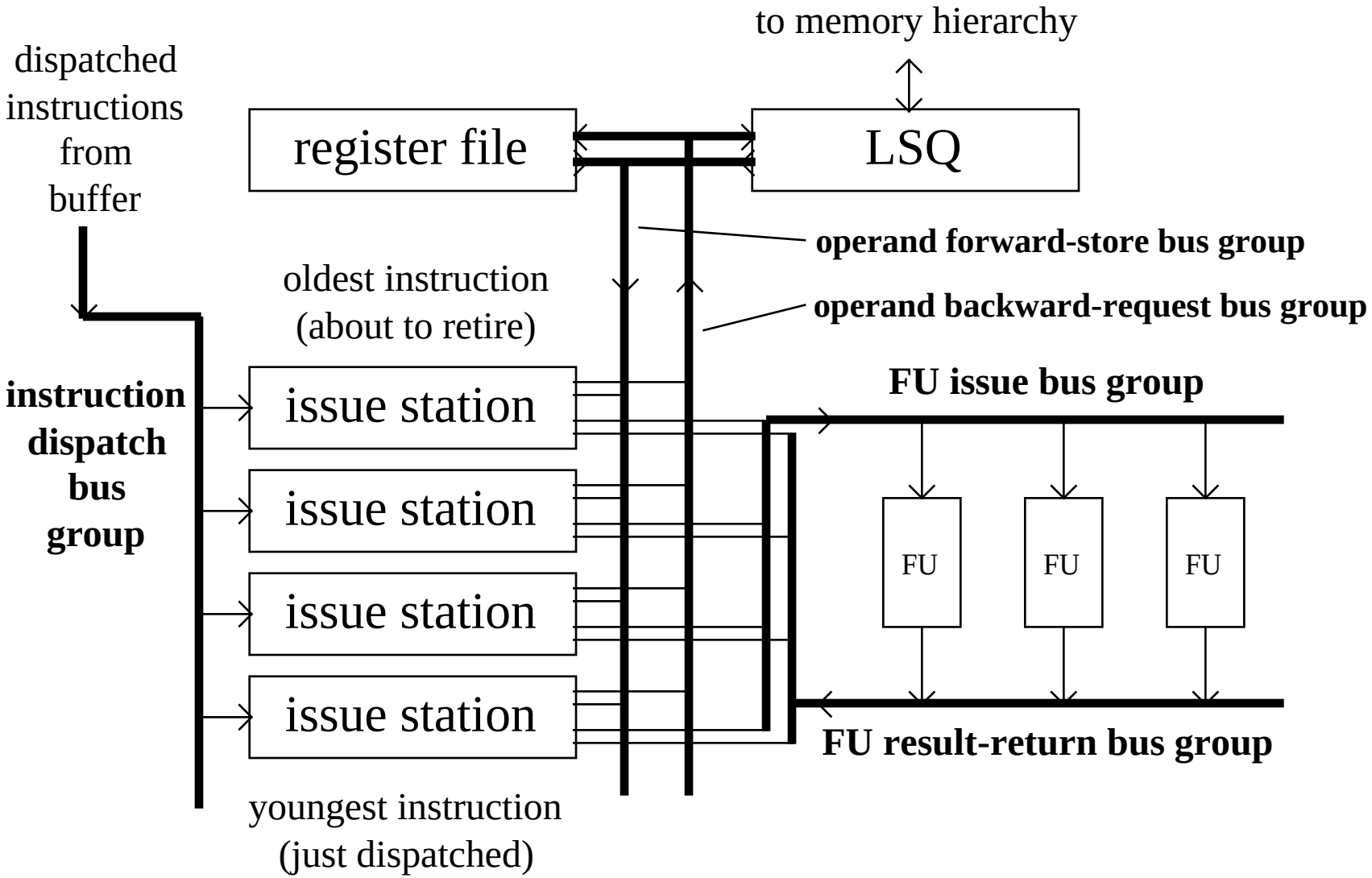
outline

- background
- Resource Flow execution
- **the OpTiFlow microarchitecture**
- Levo -- a physically scalable microarchitecture
- conclusions

microarchitecture overview



execution window



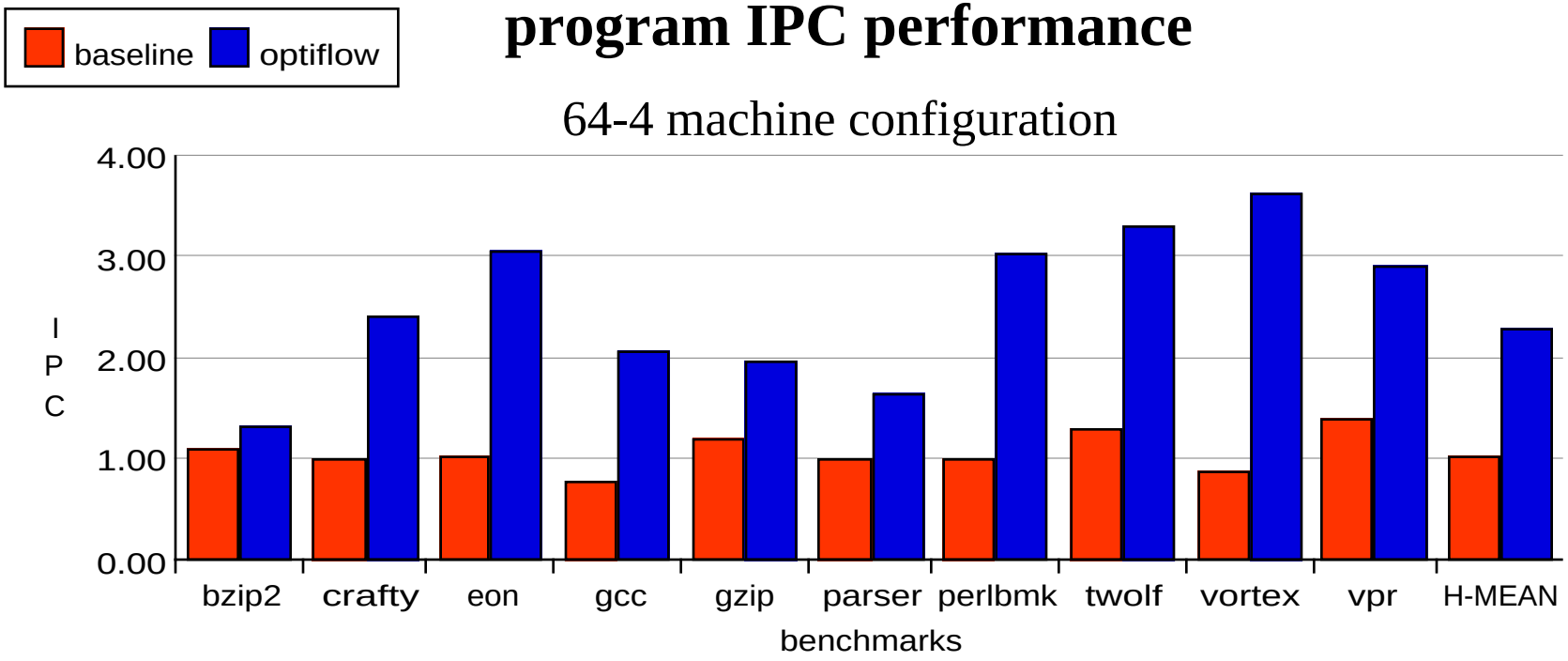
basic operation

- **instructions are decoded at fetch time**
- **decoded instructions are stored in a dispatch buffer**
- **decoded instructions are dispatched to issue stations (IS)**
- **ISes contend with each other for an execution unit (FU) as needed**
- **ISes send an instruction-operation along with its input operands to a FU when an an execution slot is granted**
- **the originating IS retrieves its FU execution result when it becomes available**
  - can stall for result or issue additional requests
- **IS forwards the result operand to other ISes in its program-ordered future**
- **ISes who snarf new (different) input operands proceed to re-execute their own instructions**
- **commitment occurs in-order from oldest IS towards younger ones**

experimental results

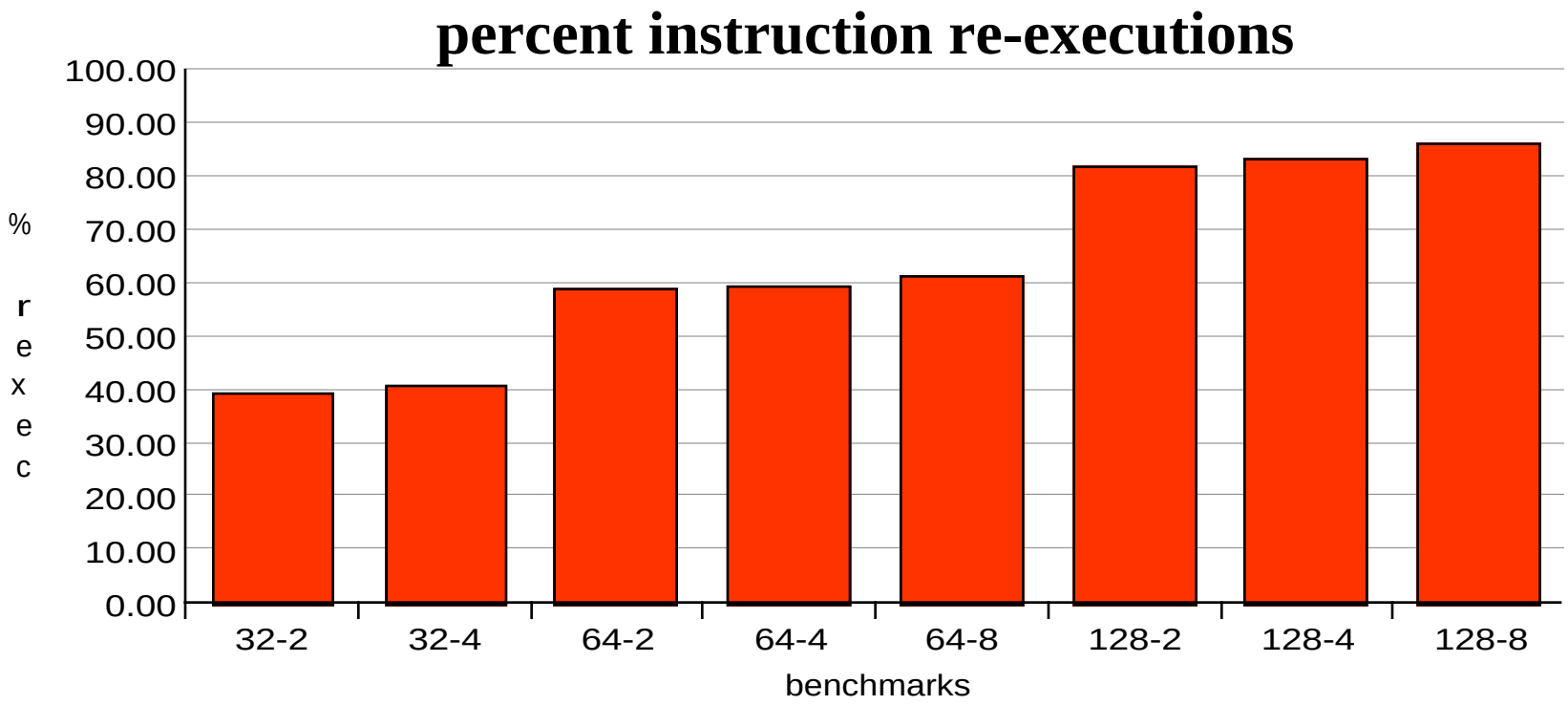
- **simulated the baseline machine using SimpleScalar (MASE)**
  - similar to typical superscalar processors, ie: Pentium-4 Northwood (P7)
- **simulated OpTiFlow microarchitecture using our own simulator**
- **both machines are configured with identical parameters for similar machine components and latencies:**
  - memory hierarchy organization, sizes, and latencies
  - branch predictor type and size
  - fetch width
  - dispatch width
  - issue width
  - numbers of function units
  - function unit latencies
- **ten (10) benchmarks from the SpecINT-2000 suite are executed**
- **the simulators skip 300 million instructions (maintaining proper program state), then functionally simulate 50 million instructions**

comparison to baseline conventional machine



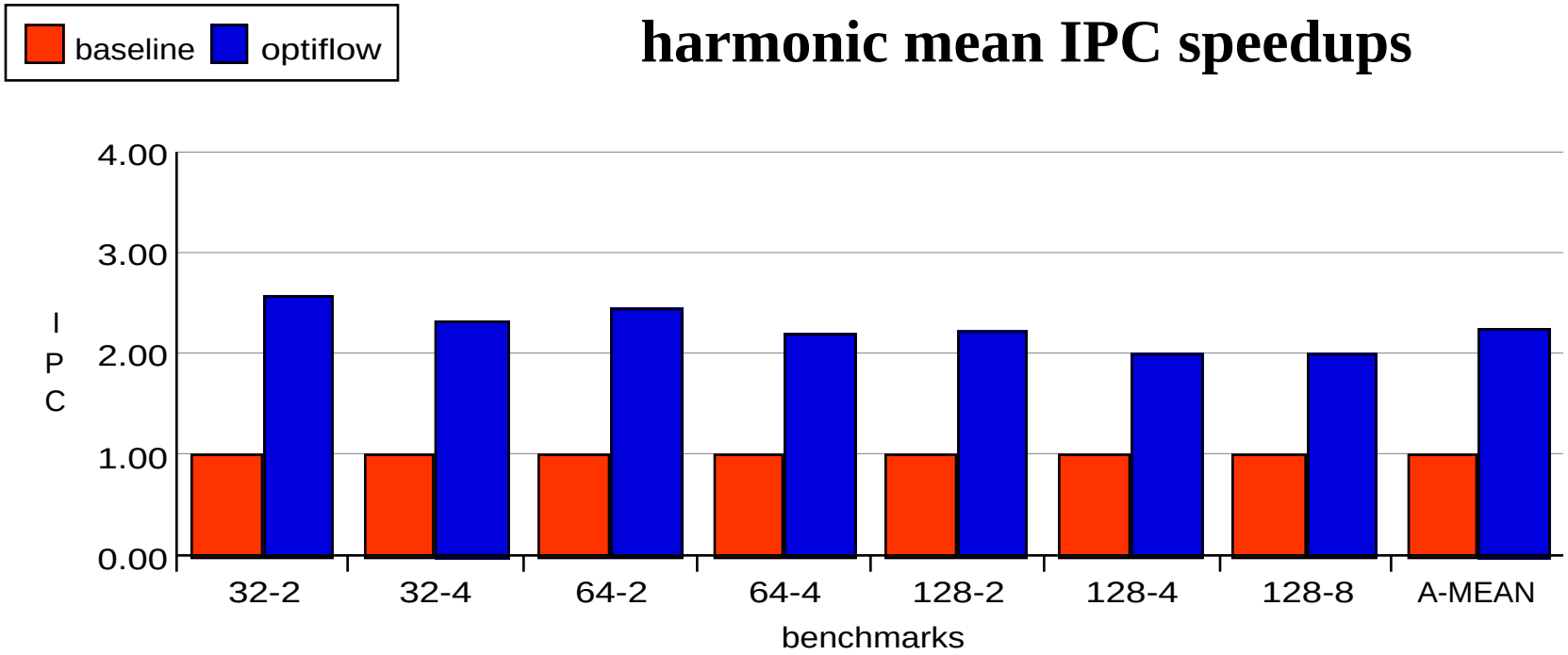
- 64 issue stations, issue width of 4 (64-4)
- OpTiFlow outperforms the baseline conventional superscalar on all benchmark programs -- harmonic mean IPC speedup of about 2.25

instruction re-execution behavior



- number of re-executions is most closely related to the number of issue stations (not the ratio of issue stations to issue width)
- large numbers of re-executions reduce performance -- limit performance scalability (in addition to not being physically scalable)
  - both problems are solved with operand filter units

summary harmonic mean IPC comparison speedups



- comparison of the baseline conventional superscalar machine with OpTiFlow
- machine configurations given in 2-tuples (issue stations, issue width)
- OpTiFlow performs about two times better (minimum IPC speedup 1.99) than the baseline conventional superscalar machine

outline

- background
- Resource Flow execution
- the OpTiFlow microarchitecture
- **Levo -- a physically scalable microarchitecture**
- conclusions

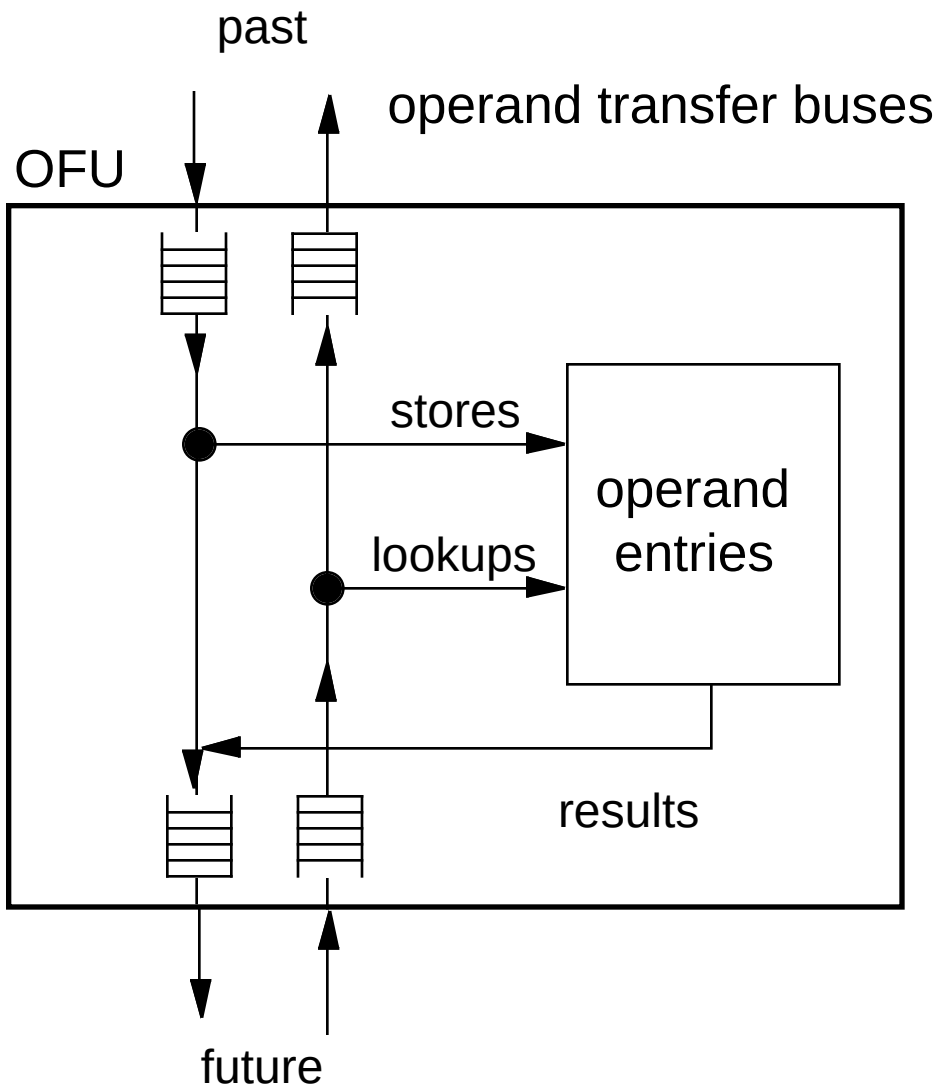
making a scalable machine

- **adopt the basic philosophy of OpTiFlow**
  - fetch, decode, dispath
  - memory hierarchy (although interleaved for bandwidth)
- **use the existing IS component as a basic building block**
- **break long buses up into segments with an Operand Forwarding-Filter Unit (OFU); three types**
  - register forwarding unit (RFU)
  - memory forwarding unit (MFU)
  - predicate forwarding unit (PFU)
- **group local ISs together into Sharing Groups (SG)**
  - each sharing group of ISs will share a Processing Element (PE)
- **abandon a single architected register file**
  - architected persistent state rotates within the RFUs
- **part of LSQ function now served by MFUs**

other additions to machine

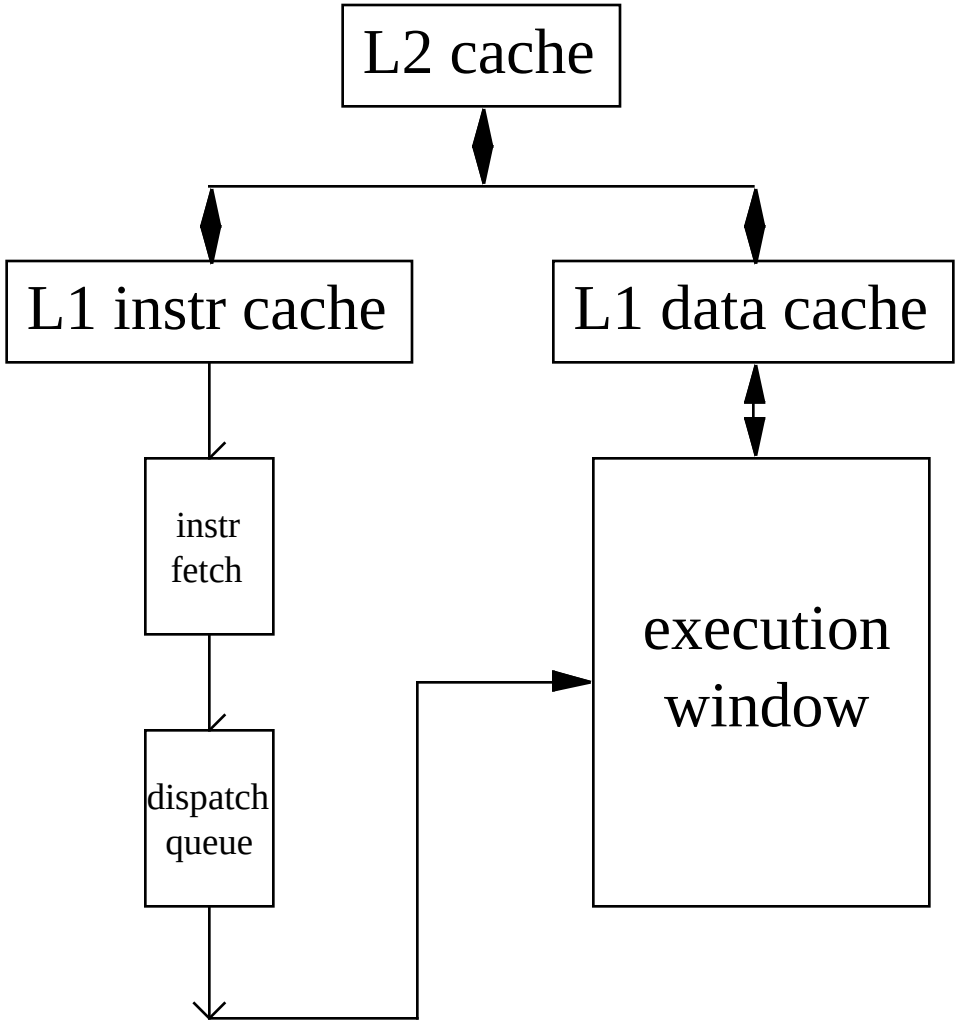
- **microarchitectural predication for all instructions**
  - facilitates recovery from mispredictions without flushing instructions
  - allows for execution of control-flow independent instructions beyond conditional branches
- **disjoint spawning of alternative (not predicted) execution paths from conditional branches**
  - will have a “mainline” speculative path
  - and disjoint eager execution (DEE) paths
- **operand forwarding units now also perform a “filtering” function**
  - removes unnecessary operand fowarding transfers
  - small memory operand caches (L0) are also distributed throughout the machine within the MFUs
  - can satisfy a request without passing it further back to the source

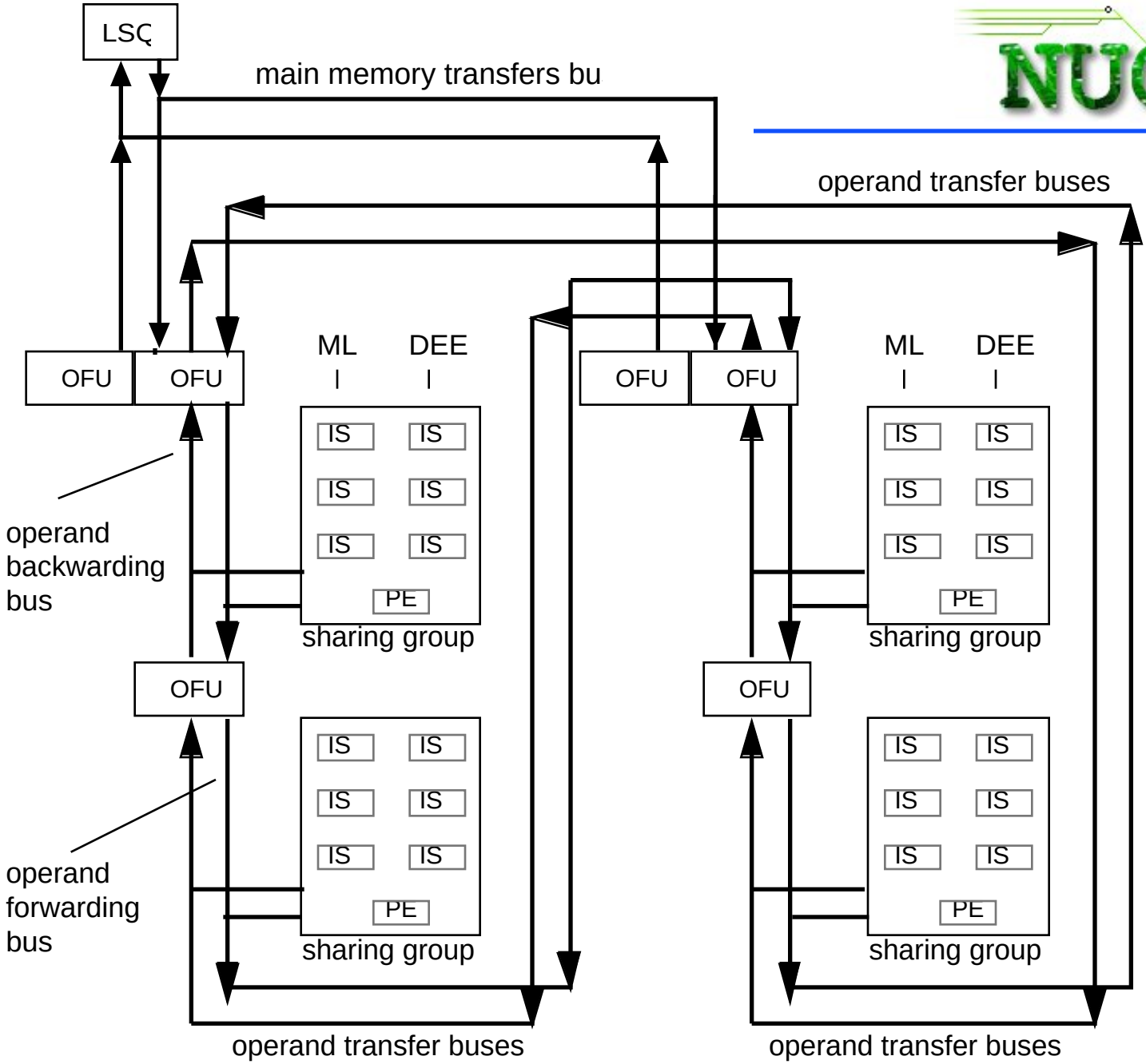
operand filter unit (OFU)



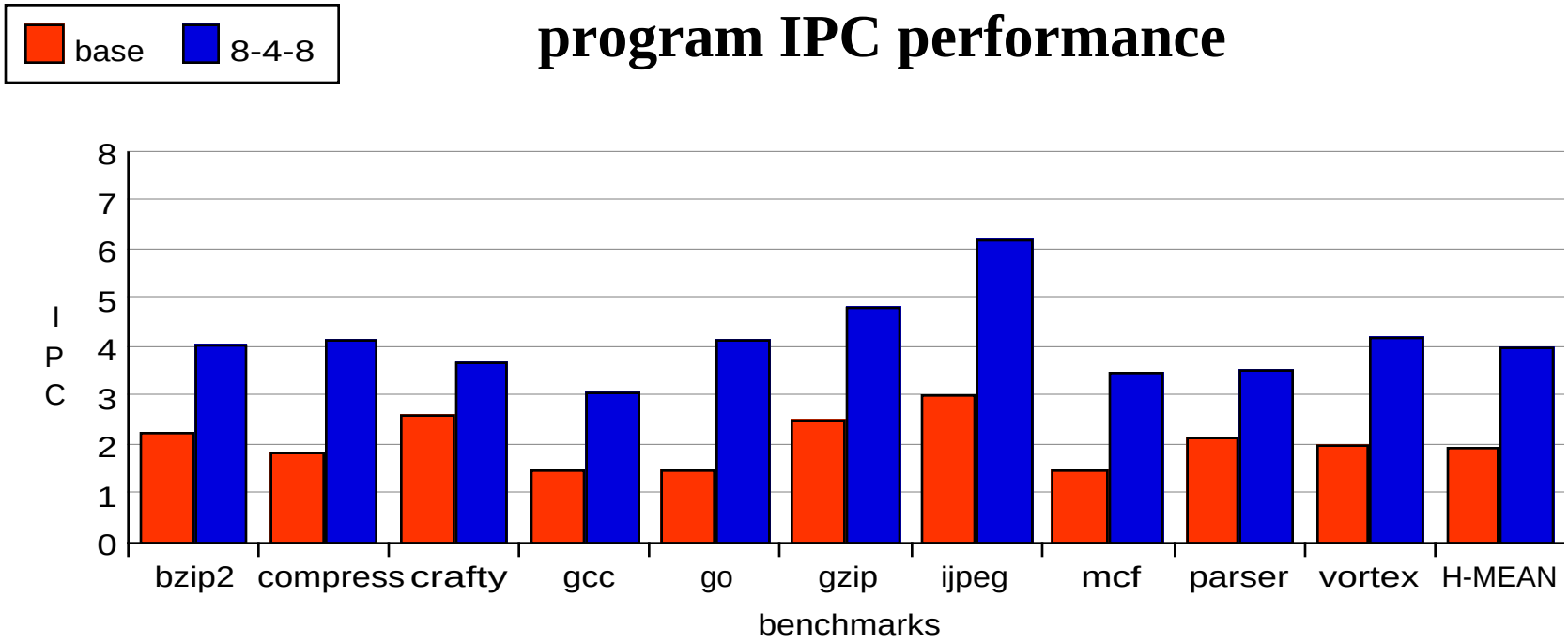
- **OFU provides**
  - forwarding function
  - backward propagation for requests
  - operand filtering function
  - satisfy backwarding requests
- **operand entries**
  - register: all regs.
  - memory: L0 cache
  - predicate: all possible

microarchitecture overview



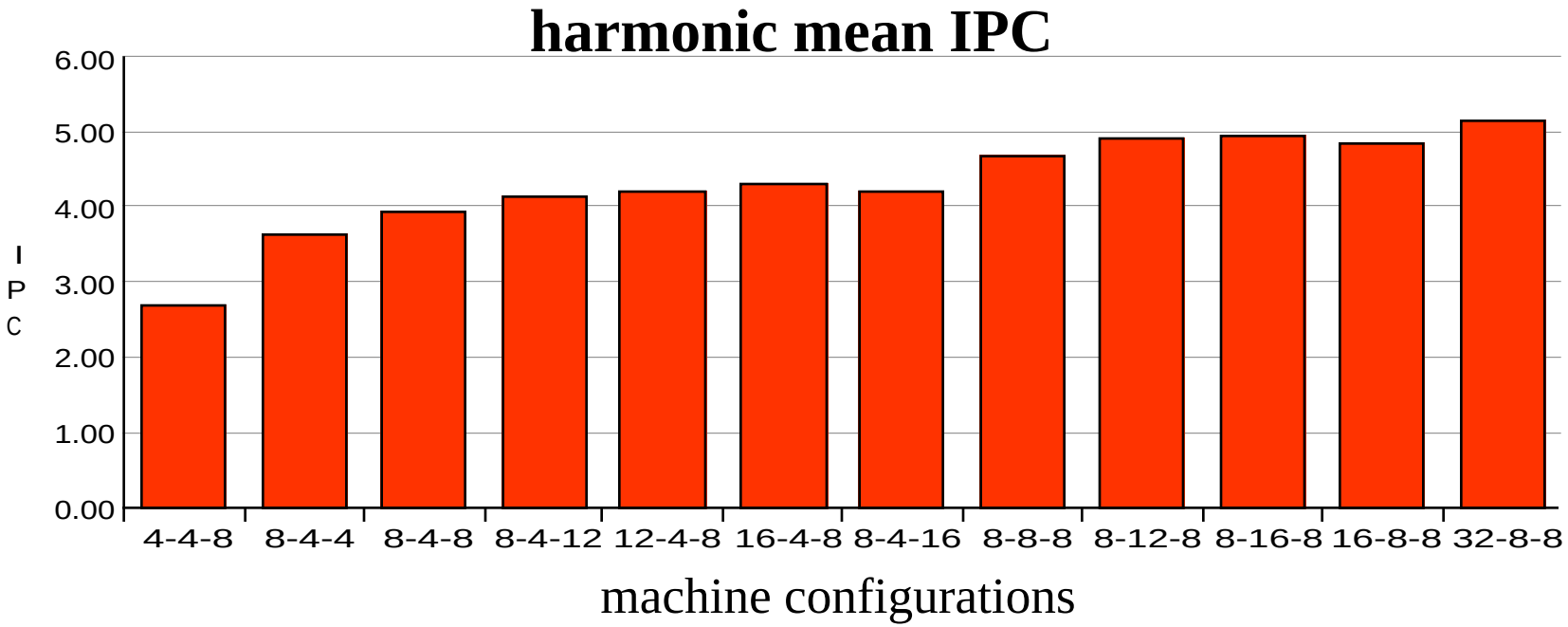

LSQ
main memory transfers bu
operand transfer buses
OFU
OFU
ML
DEE
IS
PE
sharing group
operand
backwarding
bus
operand
forwarding
bus
operand transfer buses

comparison to baseline



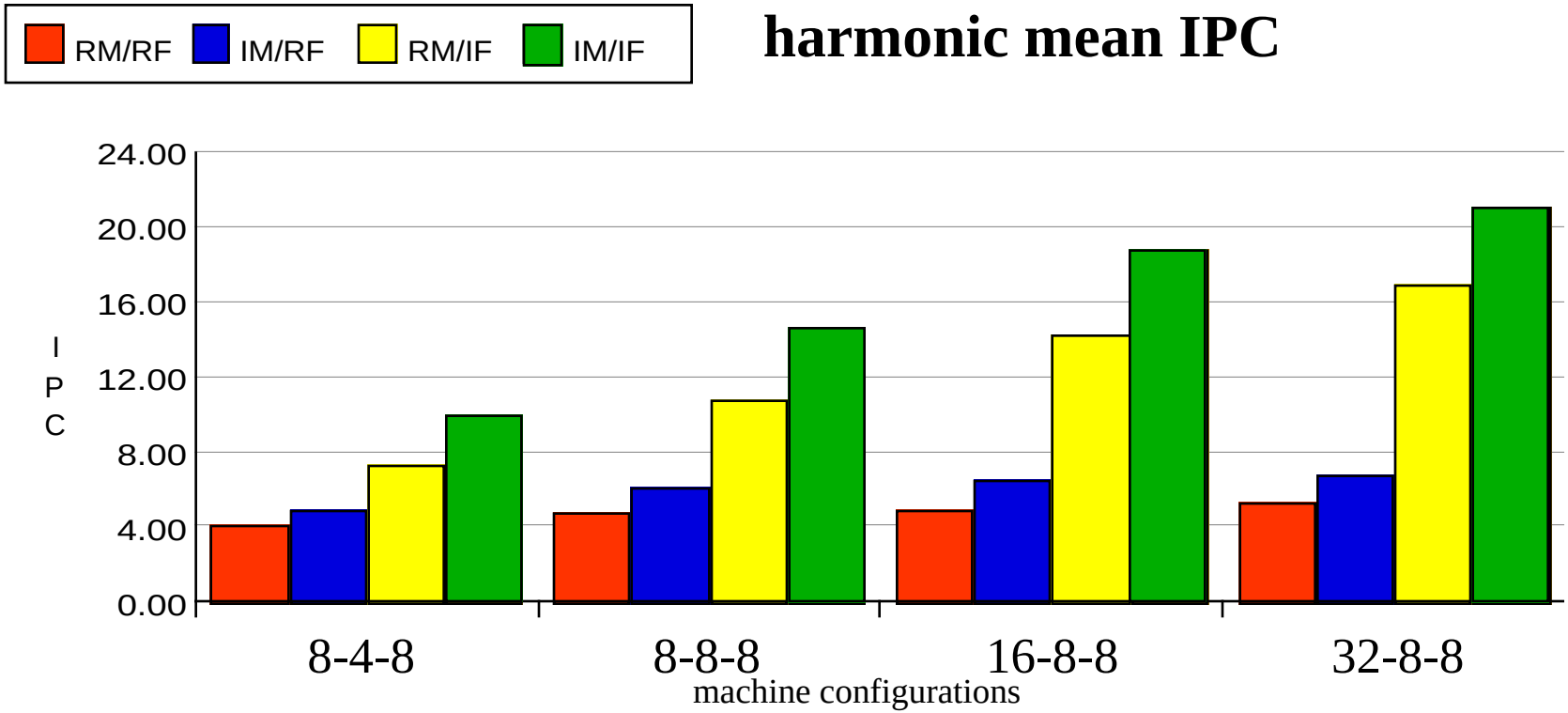
- Levo achieves approximately twice the IPC compared with the baseline conventional superscalar machine
- Levo achieves an harmonic mean IPC of about 2.0 over the baseline

performance overview



- performance returns diminish with increasing machine resources
- organization matters
  - 16-4-8 performs better than 8-4-16 even though they have the same number of execution units (same number of issue stations)
  - 8-16-8 performs better than 16-8-8 which has more execution units (same number of issue stations) but increased operand transfer delay

relaxing machine parameters



- Real and Idealized Memory -- Real and Idealized Fetch prediction
- performance increases with relaxed memory latency (no L1 misses)
- but performance is mostly affected (positively) with relaxed branch misprediction (to zero)
- this result shows where the most performance potential for Levo is

outline

- background
- Resource Flow execution
- the OpTiFlow microarchitecture
- Levo -- a physically scalable microarchitecture
- **conclusions**

conclusions (1)

- **introduced and developed an execution model we call “Resource Flow”**
- **introduced and developed a key machine component (the Issue Station) that manages instruction execution and re-execution and the flow of operands amongst instructions**
- **introduced and developed a Resource Flow microarchitecture-- OpTiFlow**
- **introduced and developed a new and suitable simulator for modeling the OpTiFlow microarchitecture**
  - OpTiFlow microarchitecture has been implemented with it
- **simulation results of OpTiFlow show that it outperforms a comparable baseline superscalar with a speedup of at least 2 on all benchmarks -- harmonic mean IPC speedup about 2.25**

conclusions (2)

- developed an improved version of dynamic instruction predication
- developed the idea of operand forwarding/filter units, for registers, memory, and microarchitectural predicates
- developed a new microarchitecture (Levo) that uses the preceding microarchitectural innovations to achieve physical scalability
- Levo outperforms the baseline conventional machine with an harmonic mean IPC speedup of about 2.0
- Levo generally achieves increased performance with increasing resources but with very substantial diminishing marginal returns
- there is a large amount of additional performance in the Levo microarchitecture if branch mispredictions can be mitigated

major contributions

- **contributed to the Resource Flow execution idea**
- **contributed to the development of the Issue Station idea and its operation**
- **developed the OpTiFlow microarchitecture**
- **developed a detailed simulator for OpTiFlow**
  - implemented (coded) the OpTiFlow microarchitecture with it
- **enhanced microarchitectural predication (previously introduced by Uht) to a fully distributed design (no centralized components)**
- **developed the idea of Operand Forwarding (Filter) Units**
  - register, memory, and predicate
- **contributed to the Levo microarchitecture (a physically scalable distributed Resource Flow microarchitecture)**
- **contributed to the Levo simulator (FastLevo)**
- **contributed to a patent for the Rsource Flow execution idea**
- **contributed to a patent for microarchitectural predication**

papers, reports, patents (1)

- **A. K. Uht, D. Morano, A. Khalafi, M. de 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, Technical 092001-001, September 18, 2001.**
- **A. K. Uht, A. Khalafi, D. Morano, T. Wenisch, M. de Alba, and D. Kaeli, "Levo: IPC in the 10's via Resource Flow Computing," IEEE TCCA Newsletter, Special Issue, December 2001. Presented at PACT 2001 Work-In-Progress (WIP) Session, September 2001.**
- **D. Morano, D. R. Kaeli, and A. K. Uht, "Preserving Dependencies in a Large-Scale Distributed Microarchitecture," Dept. of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 02864, Technical 022002-001, December 21, 2001.**
- **A. K. Uht, S. Langford, and D. Morano, "Interactive High-Performance Processor Understanding via the Web," in Proceedings of the SSGRR 2002w International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet. L'Aquila, Italy, January 21-27, 2002.**
- **D. Morano, "Execution-Time Instruction Predication," Dept. of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 02881, Technical Report 032002-0100, March 2002.**

papers, reports, patents (2)

- **A. Khalafi, D. A. Morano, D. R. Kaeli, and A. K. Uht, "Realizing High IPC Through a Scalable Memory-Latency Tolerant Multipath Microarchitecture," Department of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 02881-0805, Technical Report 032002-0101, April 2, 2002.**
- **A. Khalafi, D. A. Morano, D. R. Kaeli, and A. K. Uht, "Multipath Execution on a Large-Scale Distributed Microarchitecture," Department of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 02881-0805, Technical Report 032002-0103, February 15, 2002.**
- **A. Uht, A. Khalafi, D. Morano, M. d. Alba, and D. Kaeli, "Realizing High IPC Using Time-Tagged Resource Flow Computing," in Proceedings of the Euro-Par 2002 Conference, Springer-Verlag Lecture Notes in Computer Science. Paderborn, Germany: ACM, IFIP, August 28, 2002, pp. 490-499.**
- **A. K. Uht, D. Morano, A. Khalafi, and D. Kaeli, "Levo - A Scalable Billion Transistor CPU With High IPC," Dept. of Electrical and Computer Engineering, University of Rhode Island, Kingston, RI 02881-0805, Technical Report 082002-1000, August 2002.**

papers, reports, patents (3)

- **D. Morano, A. Khalafi, D. R. Kaeli, and A. K. Uht, "Realizing High IPC Through a Scalable Memory-Latency Tolerant Multipath Microarchitecture," in Proceedings of the Workshop On Chip Multiprocessors: Processor Architecture and Memory Hierarchy Related Issues (MEDEA2002), at PACT 2002. Charlottesville, Virginia, USA, September 22, 2002. To appear in ACM SIGARCH Computer Architecture Newsletter, March 2003.**
- **David A. Morano , "Preserving Program Dependencies in a Distributed Microarchitecture," Dept. of Electrical and Computer Engineering, Northeastrn University, ECE-CEG-02-002, July 8, 2002.**
- **Alireza Khalafi , David A. Morano , David R. Kaeli , Augustus K. Uht, "Using Timetags for Program Dependency Enforcement,"Dept. of Electrical and Computer Engineering, Northeastern University, ECE-CEG-02-003, July 19, 2002.**
- **D. Morano, A. Khalafi, D. R. Kaeli, and A. K. Uht, "Implications of Register and Memory Temporal Locality for Distributed Microarchitectures," Dept. of Electrical and Computer Engineering, Northeastrn University, Boston, MA, USA, Technical Report, October 2002.**
- **David A. Morano , "Supplemental Data for Characterization of Register Temporal Locality," Dept. of Electrical and Computer Engineering, Northeastrn University, ECE-CEG-02-005, October 25, 2002.**

papers, reports, patents (4)

- **A. K. Uht, D. Morano, A. Khalafi, and D. R. Kaeli, "Levo - A Scalable Processor With High IPC," The Journal of Instruction-Level Parallelism, vol. 5, August 2003 (http://www.jilp.org/vol5).**
- **Alireza Khalafi , David A. Morano , David R. Kaeli , Augustus K. Uht , "Dynamic Predication and Fetch Heuristics," Dept. of Electrical and Computer Engineering, Northeastrn University, ECE-CEG-03-001 March 15, 2003.**
- **David A. Morano, David R. Kaeli, Augustus K. Uht, "Resource Flow Microarchitectures," in Speculative Execution in High Performance Computer Architectures, edited by D.R. Kaeli and P. Yew, Chapman & Hall/CRC, ISBN-10 1584884479, May 2005.**
- **A.K. Uht, D.A. Morano, D. Kaeli, "Resource Flow computing device," U.S. Patent 6976150, December 2005.**
- **A.K. Uht, D.A. Morano, D. Kaeli, "Automatic and transparent hardware conversion of traditional control flow to predicates" U.S. Patent 7210025, April 2007.**

all are avalable online, including the US patents,
except for the chapter from the book "**Speculative Execution in High Performance Computer Architectures**" (but available from me electronically)