Demand-Only Broadcast: Reducing Register File and Bypass Power in Clustered Execution Cores

Mary D. Brown   Yale N. Patt

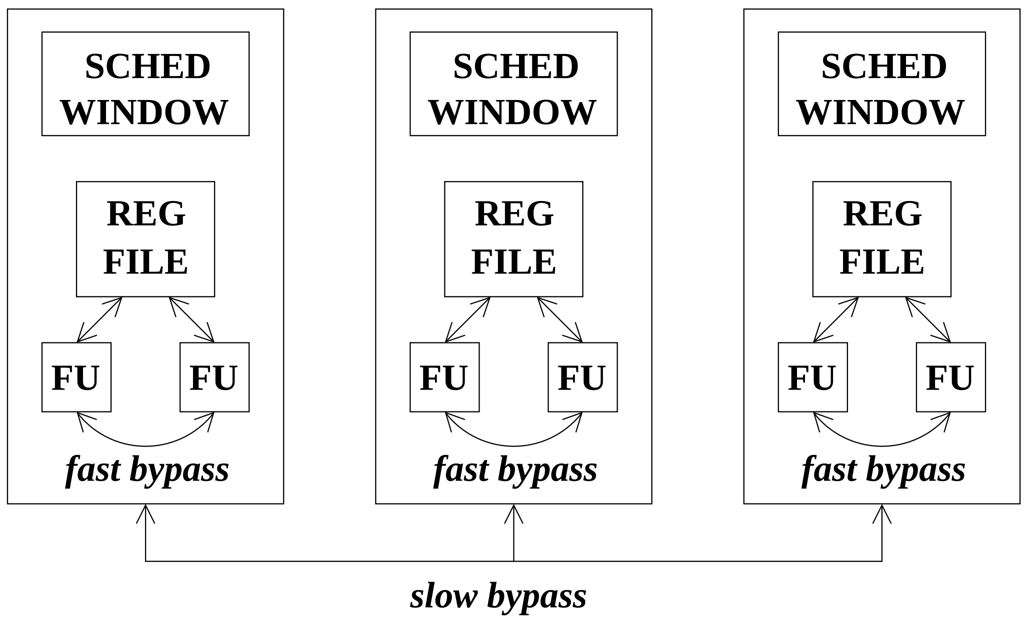
Dept. of Electrical and Computer Engineering
The University of Texas at Austin
{mbrown,patt}@ece.utexas.edu

Outline

- Introduction
- Baseline Processor Overview
- Demand-Only Broadcast Implementation
- Related Work
- Experiments and Results
- Conclusion

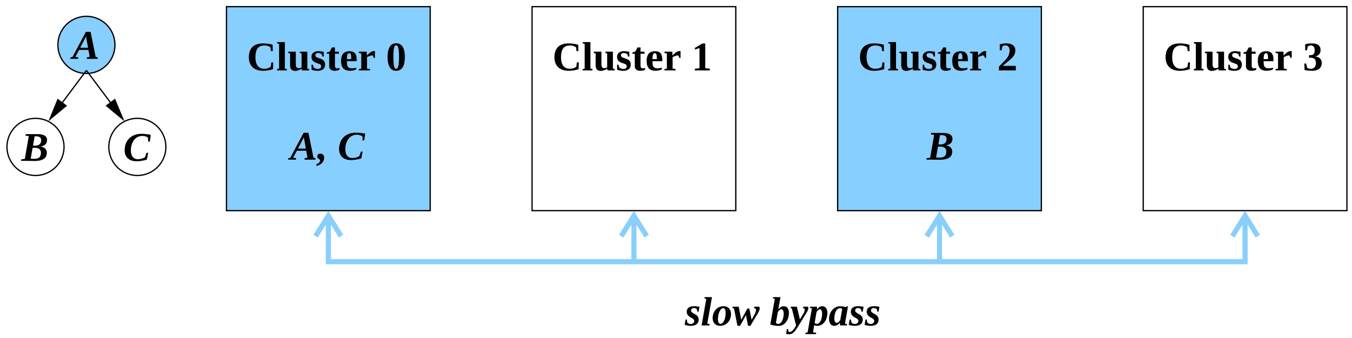
Clustering

- As execution width increases, bypass latency increases.
- Clustering reduces common-case data forwarding delays.
- Register file partitioned or replicated.



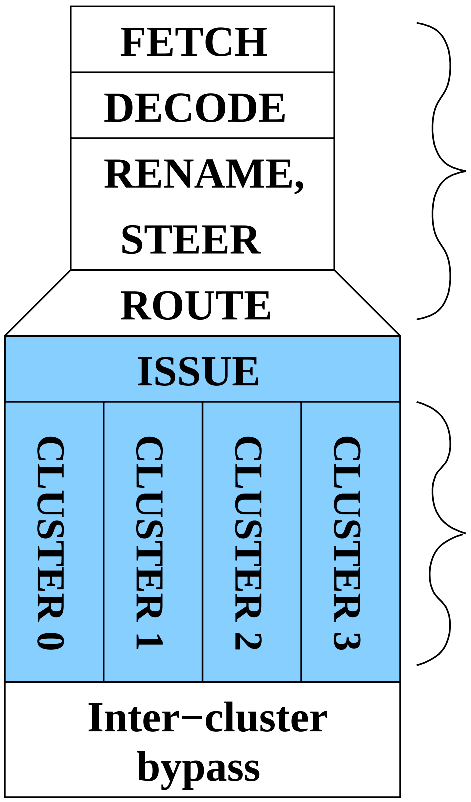
Demand-Only Broadcast Concept

- Don't broadcast result within clusters where it is not needed.
- Eliminates 59% register file writes and intra-cluster broadcasts.
- Reduces switching power of bypass network and register file.



Baseline Processor Overview

15−Stage Pipeline



9 cycles

**schedule,
register file read,
execute**

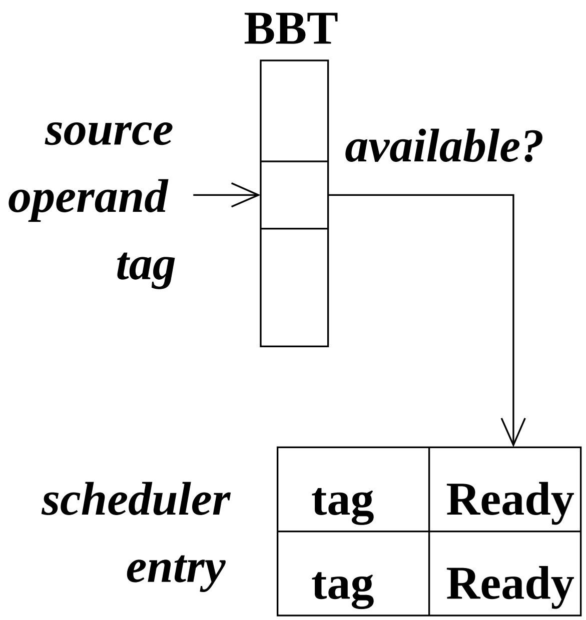
**Blue: local to
each cluster**

Cluster Contents

- 4 functional units
- Local scheduling window
- *Copy* of entire physical register file
  - Register Write Specialization (Seznec et al.):
    Instructions in a cluster can write to a subset of physical register file.
    Instructions can read from all subsets.
  - 4 write ports per entry (4 functional units per cluster)
  - 8 read ports (assumption: 2 sources per instruction)
- Busy-Bit Table (BBT)
- Bypasses for register file and BBT

Busy-Bit Table

- Used to initialize Ready bit of scheduler entry.
- One entry per physical register.
- Read at issue time.
- Set during scheduling tag broadcast.
- Cleared when physical register re-allocated
- One copy per cluster, like register file.



Demand-Only Broadcast Implementation

- Result only gets broadcast within:
  - clusters that contained consumers when its tag was broadcast.
  - inter-cluster bypass.
- BBTs indicate which clusters contain consumer instructions.
- BBT entry in each cluster read at time of tag broadcast to check for consumers.
- Eliminates intra-cluster result broadcasts.

BBT information

- BBT entry has two bits: **Broadcast** and **Use**.

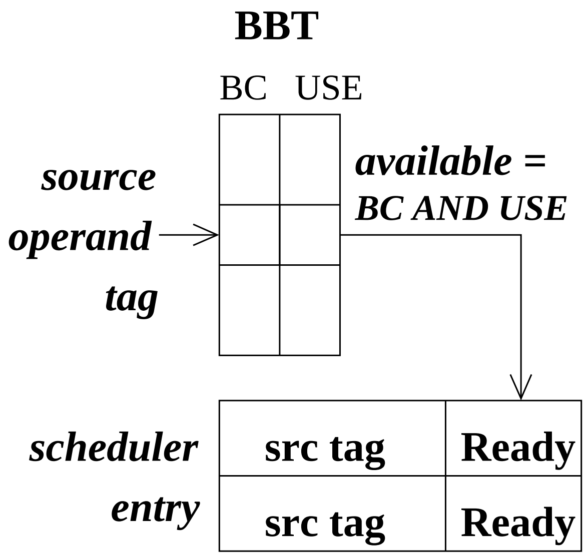
- {Broadcast, Use} =

  - {0,0}: Tag not yet broadcast.
    Data will not be broadcast.

  - {0,1}: Tag not yet broadcast.
    Tag and data will be broadcast.

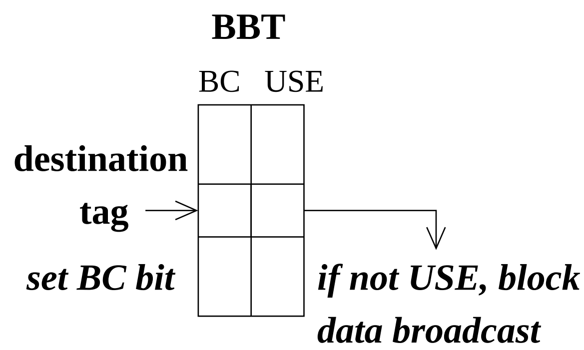
  - {1,0}: Tag was broadcast, but
    data was not and will not be broadcast.

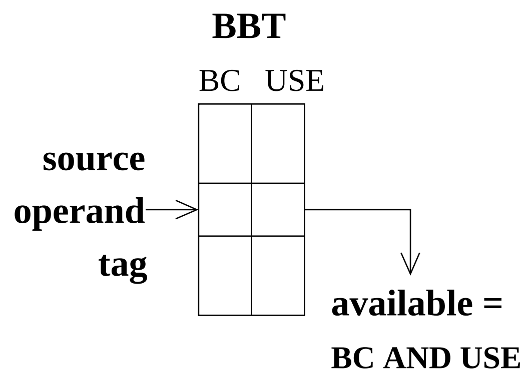
  - {1,1}: Result available.



Reading and Writing the BBT

- During tag broadcast:
  - Set BC bit.
  - Read Use bit to check for consumers.
- During issue:
  - Read entries to see if source operands are available.
  - If ($\{$BC, Use$\}$ == $\{1,0\}$), reset BC bit and insert *copy instruction*.
  - Set Use bit of destination register.
- Both bits reset when corresponding physical register is re-allocated.





Copy Instructions

- Needed when a consumer is issued to a cluster after producer's tag broadcast to that cluster and Use bit is 0.
- Re-broadcast a register value, like MOVE instructions to the same physical register.
- They take issue, scheduling, and execution bandwidth from regular instructions.
- It takes 5 cycles to detect and issue the copy instruction to the scheduling window in the producer cluster.
- Little impact on IPC.

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between Clusters 0 and 3.
2-cycle delay between tag and data broadcast.

	Action
Initial State	*A* is in Cluster 0.

	BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
A		**X**		
B				

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between clusters.
2-cycle delay between tag and data broadcast.

Cycle	Action		BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
Init	*A* is in Cluster 0.	A		**X**		
		B				
0	*A* is selected, broadcasts tag to cluster 0.	A	**X**	**X**		
		B				

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between clusters.
2-cycle delay between tag and data broadcast.

Cycle	Action		BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
Init	*A* is in Cluster 0.	A		**X**		
		B				
0	*A* is selected, broadcasts tag to cluster 0.	A	**X**	**X**		
		B				
3	*A*'s tag broadcast to Cluster 3. Read BBT-3[A].use. Block data broadcast (2 cycles later).	A	**X**	**X**	**X**	
		B				

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between clusters.
2-cycle delay between tag and data broadcast.

Cycle	Action		BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
Init	*A* is in Cluster 0.	A		**X**		
		B				
0	*A* is selected, broadcasts tag to cluster 0.	A	**X**	**X**		
		B				
3	*A*'s tag broadcast to Cluster 3. Read BBT-3[A].use. Block data broadcast (2 cycles later).	A	**X**	**X**	**X**	
		B				
4	*B* is issued to Cluster 3. Read BBT-3[A]. Request Copy.	A	**X**	**X**		**X**
		B				

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between clusters.
2-cycle delay between tag and data broadcast.

Cycle	Action		BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
Init	*A* is in Cluster 0.	A		**X**		
		B				
0	*A* is selected, broadcasts tag to cluster 0.	A	**X**	**X**		
		B				
3	*A*'s tag broadcast to Cluster 3. Read BBT-3[A].use. Block data broadcast (2 cycles later).	A	**X**	**X**	**X**	
		B				
4	*B* is issued to Cluster 3. Read BBT-3[A]. Request Copy.	A	**X**	**X**		**X**
		B				
9	*Copy-A* is issued, already awake.					

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between clusters.
2-cycle delay between tag and data broadcast.

Cycle	Action		BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
Init	*A* is in Cluster 0.	A		**X**		
		B				
0	*A* is selected, broadcasts tag to cluster 0.	A	**X**	**X**		
		B				
3	*A*'s tag broadcast to Cluster 3. Read BBT-3[A].use. Block data broadcast (2 cycles later).	A	**X**	**X**	**X**	
		B				
4	*B* is issued to Cluster 3. Read BBT-3[A]. Request Copy.	A	**X**	**X**		**X**
		B				
9	*Copy-A* is issued, already awake.					
12	*Copy-A* broadcasts tag in Cluster 3. *B* wakes up.	A	**X**	**X**	**X**	**X**
		B				

Example of Inter-Cluster Broadcast

Assumptions: 3-cycle delay between clusters.
2-cycle delay between tag and data broadcast.

Cycle	Action		BBT-0 BC	BBT-0 USE	BBT-3 BC	BBT-3 USE
Init	*A* is in Cluster 0.	A		**X**		
		B				
0	*A* is selected, broadcasts tag to cluster 0.	A	**X**	**X**		
		B				
3	*A*'s tag broadcast to Cluster 3. Read BBT-3[A].use. Block data broadcast (2 cycles later).	A	**X**	**X**	**X**	
		B				
4	*B* is issued to Cluster 3. Read BBT-3[A]. Request Copy.	A	**X**	**X**		**X**
		B				
9	*Copy-A* is issued, already awake.					
12	*Copy-A* broadcasts tag in Cluster 3. *B* wakes up.	A	**X**	**X**	**X**	**X**
		B				
13	*B* is selected and broadcasts its tag to Cluster 3.	A	**X**	**X**	**X**	**X**
		B			**X**	

Related Work

- Multiscalar Processors (Sohi, Breach, Vijaykumar).
  - Compiler breaks program into *tasks*.
  - Only live-outs of tasks forwarded between processing elements.
- Multicluster Architecture (Farkas, Jouppi, Chow).
  - Architected registers divided among clusters.
  - Difficulty with load balancing.
- Clustered microarchitectures with dynamic steering (Hrishikesh ; Kemp and Franklin ; Zyuban and Kogge ; Canal, Parcerisa, and González)
  - Limited Intra-cluster bandwidth requires arbitration: inter-cluster buffers or copy instructions

Partitioned Register File Model (PART)

- Physical register file is partitioned rather than replicated.
- Results only broadcast to local cluster.
- *Copy instructions* used to send register values to other clusters.
- Copy instructions use rename, issue, scheduling, and execution bandwidth.
- Need for copy instructions detected during rename stage after steering.
- Rename tables must be larger to contain multiple mappings.
- More scheduling window and register file entries needed.

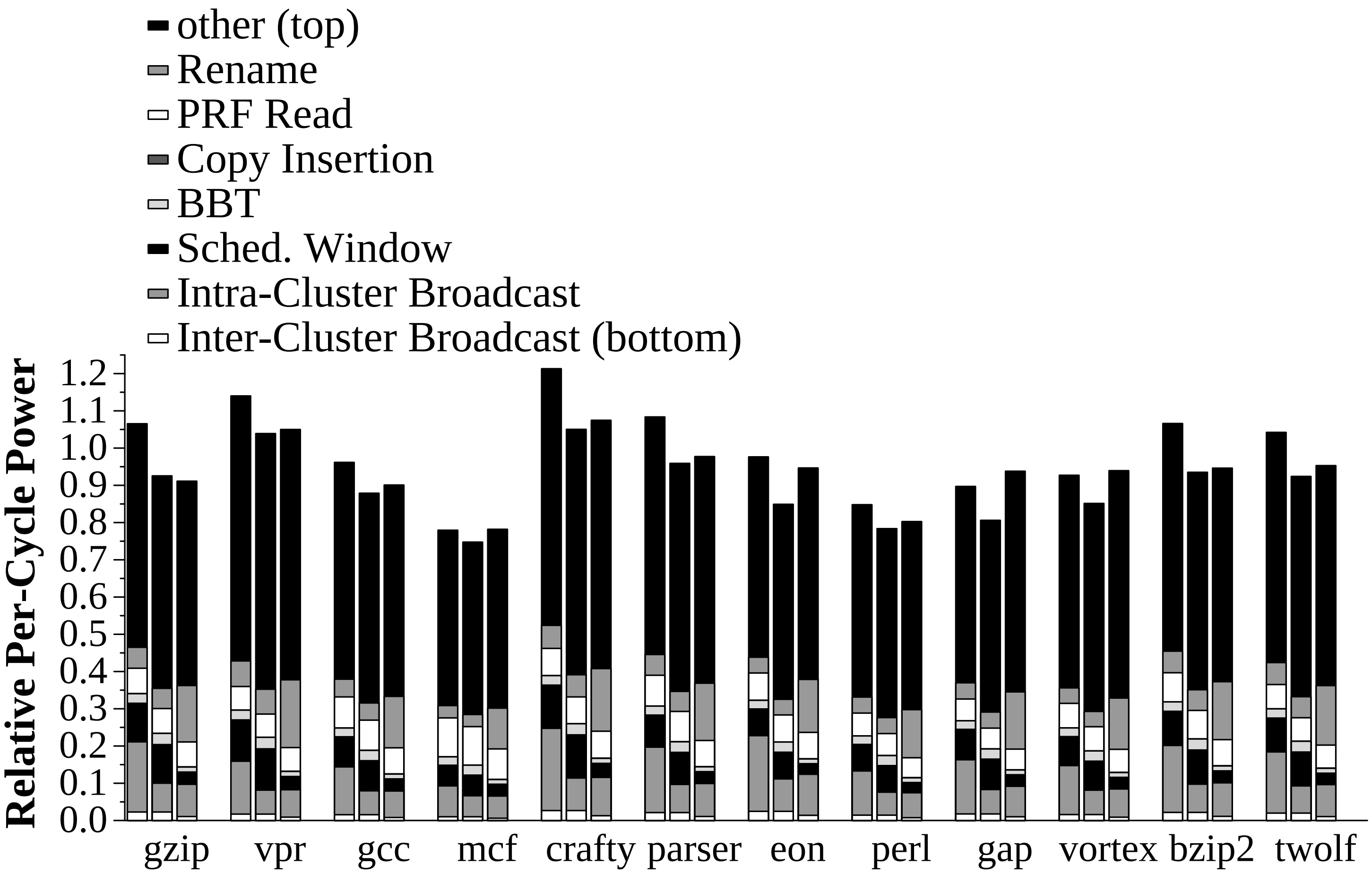
Experimental Framework

- Execution-driven simulator executing the Alpha ISA.
- Microarchitecture:
  - 15-stage pipeline
  - 512-entry window
  - 16-wide execution across 4-clusters
- Power model is derived from Wattch
  (Brooks, Tiwari, Martonosi, ISCA-27)
- IPC and per-cycle relative power estimates
- SPEC2000 integer benchmarks

Three Processor Models

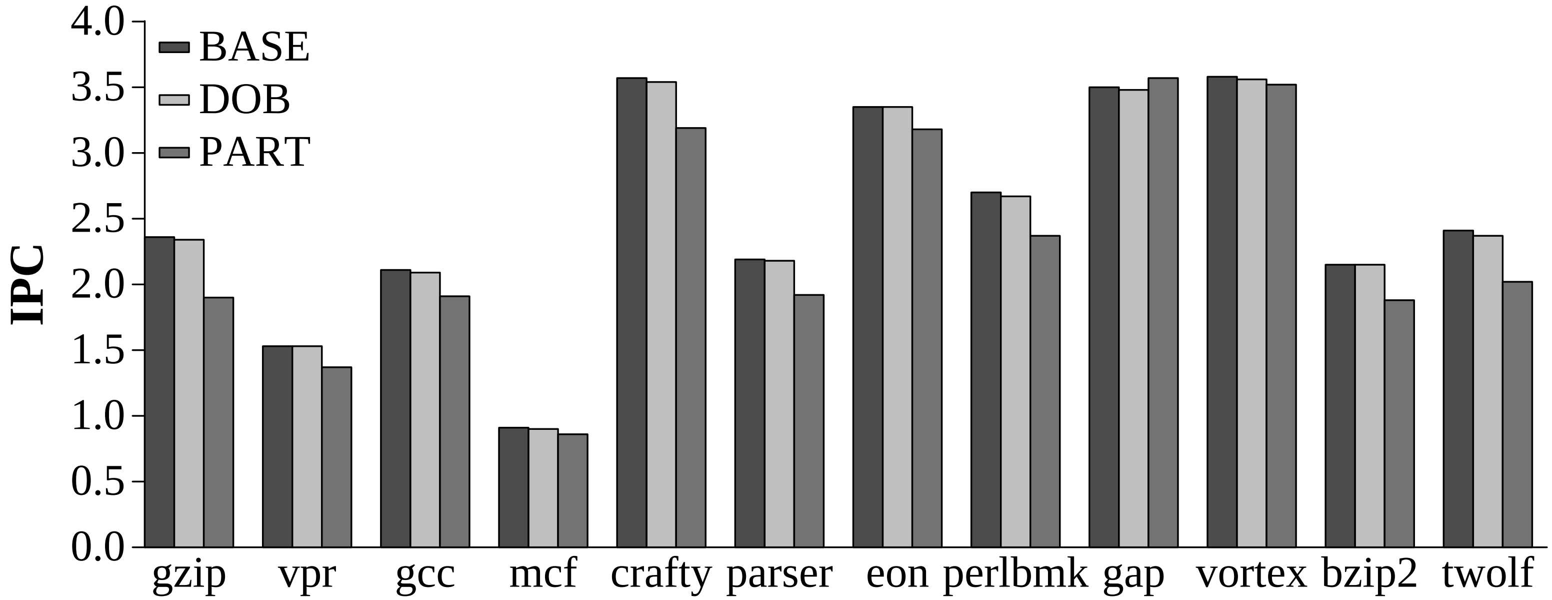
- **Baseline** (BASE)
  - Replicated register file
  - Results get broadcast to all clusters
- **Demand-Only Broadcast** (D.O.B.)
  - Replicated register file
  - Bigger BBT (2 bits per entry vs. 1)
  - Logic for inserting copy instructions
- **Partitioned Register File** (PART)
  - Bigger windows (**96** entries vs. 64 per cluster)
  - Fewer register file entries (**224** vs. 512 per cluster)
  - Bigger Rename Table (4 entries per architected register vs. 1)
  - Fewer ports for the register file, scheduling window, and BBT
  - Support for copy instructions

Breakdown of Relative Power Consumption



Models left to right: BASE, DOB, PART

IPC on SPECint2000 Benchmarks



Models left to right: BASE, DOB, PART

Summary of Experimental Results

- IPC of D.O.B. model is within 1% of Base, 10% higher than PART.
  - PART IPC is lower due to copy instructions.
  - PART has 16 times as many copy instructions as D.O.B. model
- Processor power reduced by 10% in Demand-Only Broadcast, 7% in PART.
  - Power of directly-affected components reduced by 26% in D.O.B., 16% in PART.
  - Demand-Only Broadcast gets rid of 59% of register file writes and local data broadcasts.
  - PART has less register file and scheduling power, but more rename power

Conclusions

- Demand-Only Broadcast prevents unnecessary data broadcasts.
- Eliminates 59% of register file writes in a 4-cluster core.
- Saves switching power.
- Could also be used to reduce the number of register file write ports.
- Higher performance than model with partitioned register file and limited inter-cluster bandwidth.

Example of Inter-Cluster Forwarding

- A issued to Cluster 0. B issued to Cluster 3.
- 3 cycle delay between clusters 0 and 3
- 2 cycle delay between tag and data broadcasts

Cycle	
0	A's tag broadcast in cluster 0. Set BBT-0[A].
2	A's data broadcast in cluster 0.
3	A's tag broadcast in cluster 3. Set BBT-3[A].
4	B's tag broadcast in cluster 3. Set BBT-3[B].
5	A's data broadcast in cluster 3.
6	B's data broadcast in cluster 3.