

Power-Aware Microprocessors

Emily Chan

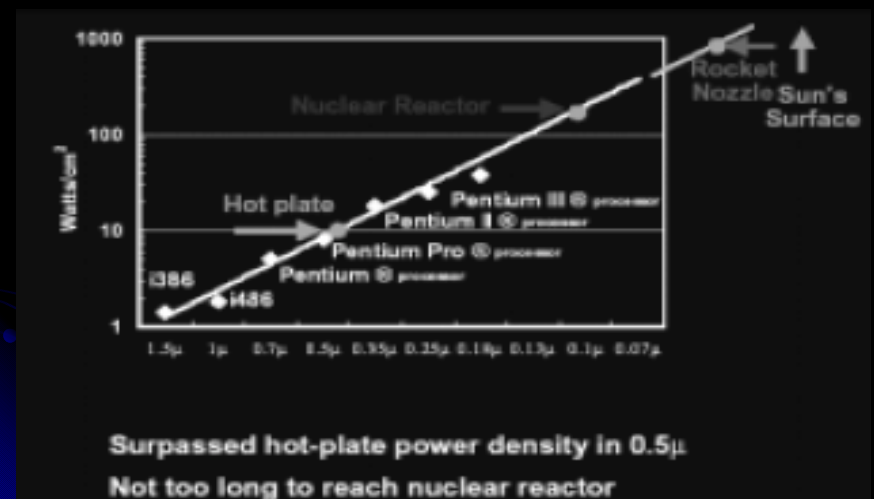
- Introduction
- Focus of the paper
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

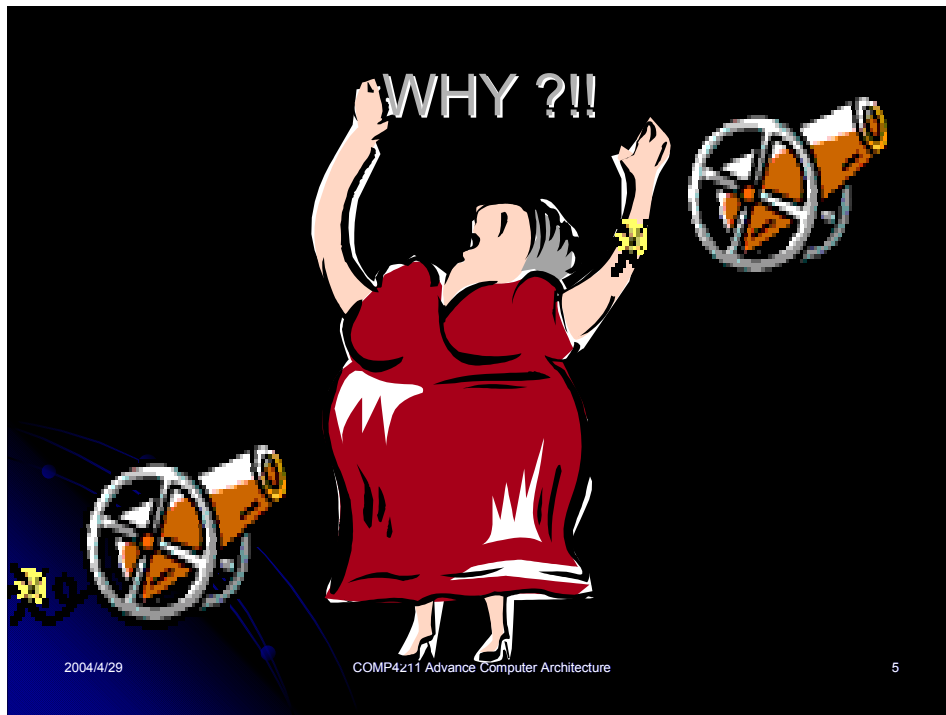
Paper

Yu Bai and R. Iris Bahar.
A Dynamically Reconfigurable Mixed In-Order/Out-of-Order Issue Queue for Power-Aware Microprocessors.

Outline

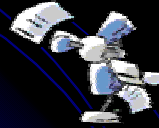
WHY?





Two Major Issues

- Battery Life ñ Mobile phones, Laptops and any other portable equipments.
- Cooling Package ñ When Pentium ìNî comes out, you may have to keep it in a freezer.



2004/4/29

COMP4211 Advance Computer Architecture

6

What is the problem?

- Different applications may vary widely in:
 - Degree of instruction-level parallelism (ILP)
 - Branch behavior
 - Memory access behavior

→ Datapath resources not optimally utilized by all applications

HOWEVER, Still consuming power!!!!

2004/4/29

COMP4211 Advance Computer Architecture

7

How can we solve the problem?

Golden Rule:

A good design strategy should be flexible enough to dynamically reconfigure available resources according to the program's needs.




2004/4/29

COMP4211 Advance Computer Architecture


8

Outline



- Introduction 
- Focus of the paper
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

Focus of the paper

- ìReconfigurabilityî of the issue queue in out-of-order superscalar processors
 - ➔ a large source of the total power dissipation

- Believe it or Not: 
For Alpha 21264, **46%** of the total power goes to the issue logic!

Outline

- Introduction 
- Focus of the paper 
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

Overview of Approaches Taken

- Partition issue queue into several sets (FIFOs) -- Why?
- Only instructions at the head of each FIFO are visible to the request and selection / arbitration logic -- Why?
- Each FIFO issues in-order though the overall issue logic is still out-of-order -- What are the benefits?

Outline

- Introduction
- Focus of the paper
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

Related Work Done

- Hardware dynamically monitors performance
 - ➔ disabling part of integer and/or floating point pipelines
- Varying the instruction issue width to allow disabling of a cluster of function units
- Dynamically reducing the number of active entries in the instruction window

Drawbacks

- No way to tell whether an instruction is ready to be issued or not and all instructions are visible to the selection and wake up logic
 - ➔ power inefficient
- Dynamically adjusting the issue queue size
 - ➔ narrows the scope of instructions available for exposing ILP

Palacharla's approach

- Uses FIFOs as well
- Simplifies wake up and selection logic which puts chains of **dependent instructions** into FIFO buffers
- Issues instructions from multiple buffers in parallel

Palacharla's Drawbacks

- Uses a single fixed-sized data structure
→ not always beneficial for different applications

Why is data structure such an important issue?

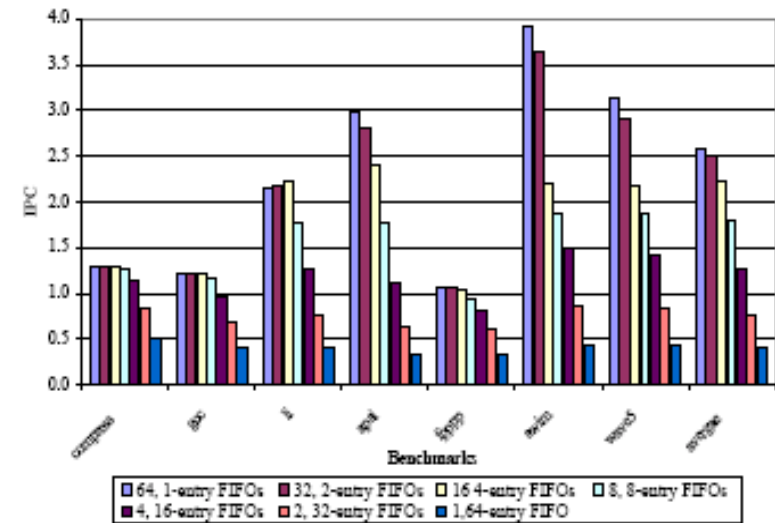


Figure 1. IPC comparison using fixed-sized FIFOs.

Performance Analysis

- Use a 1-entry FIFO configuration as a base case, on average:
 - 2-entry FIFO → 3% drop
 - 4-entry FIFO → 14% drop
 - 8-entry FIFO → 30% drop
 - 64-entry (a single FIFO) → 84% drop
- For *li*, performance improves up to 4-entry FIFO → avoids executing wrong path instructions effectively

Outline

- Introduction
- Focus of the paper
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

Implementations

- Scheme # 1

Completely disable some under-utilized FIFOs in the issue queue according to feedback from performance monitor (hardware)

Pro: By completely disabling a FIFO → any signals associated disabled → more power savings

Con: Shrinking the overall size of the issue queue → Limit exposure to potential ILP → not suitable for Floating Point execution

Implementations

- Scheme # 2

- vary the number and size of the FIFOs simultaneously according to feedback from performance monitor
- size of FIFOs increases while the number of FIFOs decreases
- retain same number of issue queue entries at all times but the queue appears to be smaller

Pro: more flexibility in exposing potential ILP

Con: entries are only made invisible → associated signals still enabled → less power savings

Implementations

- When performance is suffering

→ a large fraction of the issue queue is turned back on (Scheme # 1) or made visible (Scheme # 2) to the request and selection logic

Pipeline Organization

- Up to 6 instructions each cycle

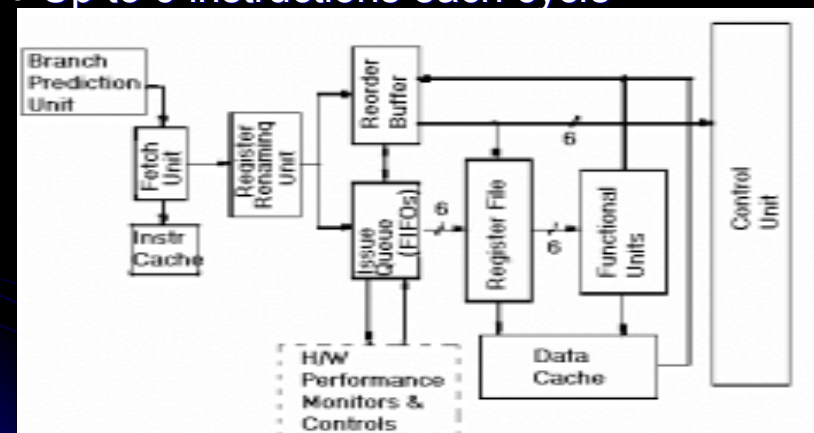


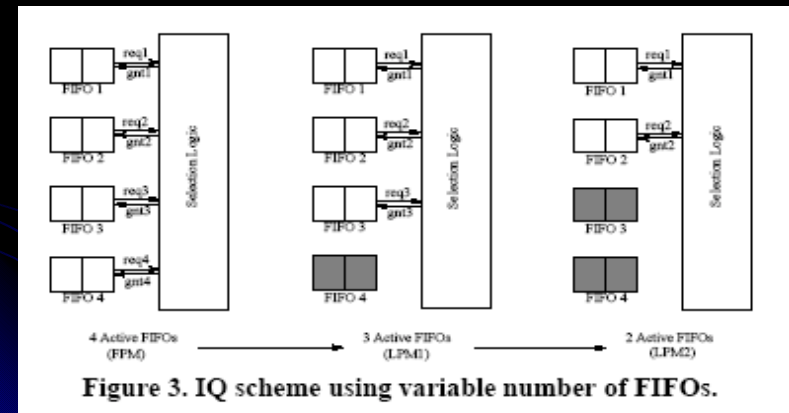
Figure 2. Pipeline organization.

Two Major Components

- Issue queue
 - a set of reconfigurable FIFOs
 - insert at the tail; issue from head of a FIFO
 - only heads of FIFOs are visible
- Hardware performance monitors
 - determine optimal issue queue configuration
 - statistics gathered over a fixed interval of cycles called a **cycle window** (1024 cycles)

Issue Queue Design

- Scheme # 1

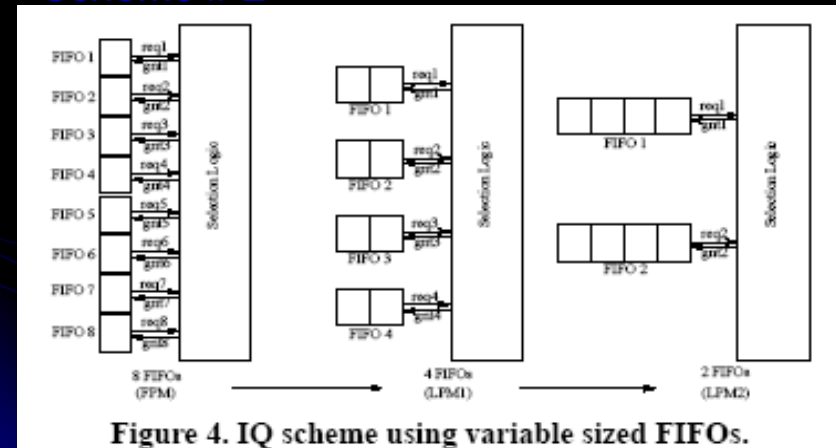


Scheme # 1 Design

- When under-utilized, disable a FIFO
- FIFO must be drained of all valid entries before being disabled
- Reduces number of instructions bidding for an issue slot → power saving in the wake-up and selection logic!
- Not having to update the ready status of the disabled instruction entries → power saving!

Issue Queue Design

- Scheme # 2





Scheme # 2 Design

- Vary size and number of FIFOs simultaneously
- Assumed no cycle overhead in changing from one configuration to another since each instruction has a set of arbiter enable signals indicating its arbiter assignment
- Arbiter signals are disabled except for heads of FIFO → power saving!
- Power savings only when reduced activities in the request and selection logic


Allocations of instructions into FIFOs

- Important that most of the ready instructions are at the heads of FIFOs
→ use a **dependency-based** strategy
- Attempt to place an instruction in the same FIFO as one or both of its source dependencies

Dependency-based Strategy

- If ready → new empty FIFO
→ if no empty FIFO then  !!!
- If one pending operand
→ steer to the same FIFO as the producer if possible
→ if fail, try a new empty FIFO
→ if no empty FIFO then  !!!!

Dependency-based Strategy

- If two pending operands
→ implement a **Last Operand Predictor** (LOP) to predict which of two operands will become available later
→ try the late arrived producer first
→ if fail, try the other producer
→ if fail again, try a new empty FIFO
→ if no empty FIFO then  !!!!

Hardware Performance Monitors

- At the end of each cycle window, determine which operating mode next
- A combination of different monitoring techniques used → better control



Monitoring Techniques

- Monitoring IPC
 - low IPC → disable / hide part of the issue queue and enter low-power mode (LPM)
- Detecting variations in IPC
 - if issue and commit rates vary significantly → a high branch misprediction → decrease the number of FIFOs

Monitoring Techniques

- Performance degradation
 - drop in IPC between two cycle windows exceeds a threshold value → back to higher power mode
- Monitoring ready instructions
 - too many stalls → increase the number of FIFOs
 - very little stalls → decrease the number of FIFOs

Monitoring Techniques

- Issue queue usage
 - low occupancy → reduce the number of FIFOs
- Non-Critical Instructions
 - if no instruction is placed behind a ready instruction by the time it is removed from the queue → **non-critical instruction**
 - delaying such ready instruction won't hurt
 - too many non-critical instructions → reduce the number of FIFOs

Power Estimations

- Extrapolated from available Alpha 21264 power estimates
- Different issue queue designs but both use an out-of-order issuing scheme
- Assume *issue logic* = *register file* + *register mapping* + *issue queue*
- *Issue queue* = *register scoreboard* + *request logic* + *arbiters*

2004/4/29

COMP4211 Advance Computer Architecture

37

Power Estimations

- Estimates:
 - arbitration logic → 60% of issue queue power
 - request logic → 15% of issue queue power
 - register scoreboard and rests → remaining 25%
- **Reminder:** Reduce numbers of FIFO → reduce activity on the arbiter enable signals, and the request logic and signals → power savings!

2004/4/29

COMP4211 Advance Computer Architecture

38

Request Logic

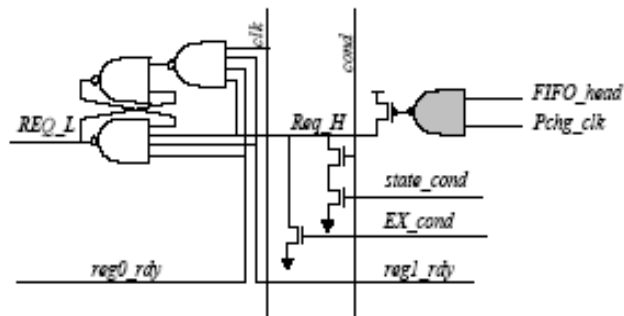


Figure 5. Request logic for one row of the scoreboard with modifications shown in gray. Taken from [4].

2004/4/29

COMP4211 Advance Computer Architecture

39

Request Logic

- Only request lines of heads of FIFOs are enabled → be **precharged!**
- Use the **FIFO_head** signal to achieve this
- **REQ_L** asserted iff **FIFO_head** asserted
- Conventional out-of-order issue queue: precharges every request lines each cycle!
- Execution assignment info (**state_cond** and **Ex_cond**) updated no matter what → save power only by completely disabling the FIFO (Scheme # 1)

2004/4/29

COMP4211 Advance Computer Architecture

40

Arbitration Logic

- Precharge only the grant lines of heads of FIFO
- Assume power used in arbitration logic is directly proportional to the number of active FIFOs
 - ➔ save more power by disabling all the grant lines associated with the unused issue slots

Register Scoreboard Logic

- Track data dependencies among instructions in the issue queue
- Necessary to update information for each issue queue entries unless a FIFO is completely disabled ➔ only Scheme # 1 can achieve power saving

Experimental Methodology

- Uses **SIMPLESCALAR**
- Original *Register Update Unit (RUU)* = *instruction window + array of reservation stations + reorder buffer (ROB)*
- RUU split into ROB and issue queue (IQ)
 - ➔ more accurate modeling of current and next generation processors
- ROB ➔ order instructions according to their input dependencies before entering the queue

Complete Configuration

Table 1. Processor resources

Parameter	Configuration
Inst. Window	256-entry LSQ, 512-entry ROB
	64-entry IQ
Machine Width	6-wide fetch, issue, commit
Fetch Queue	8
FUs & Latency	8 Int add (1), 2 Int mult/div (3/20) 4 FP add (2), 2 FP mult/div/sqrt (4/12/24) 4 Load/Store (1)
L1 Icache	32KB 2-way; 32B line; 1 cycle
L1 Dcache	32KB 2-way; 32B line; 1 cycle
L2 Cache	256KB 4-way; 64B line; 6 cycle
Memory	128 bit-wide; 20 cycles on hit, 50 cycles on page miss
Branch Pred.	4k 2lev + 4k bimodal + 4k meta 6 cycle mispred. penalty
BTB	1K entry 4-way set assoc.
RAS	32 entry queue
ITLB	64 entry fully assoc.
DTLB	64 entry fully assoc.

Outline

- Introduction
- Focus of the paper
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

Specific Monitor Technique for Scheme # 1

- Disable one FIFO when either (ordered according to relative importance):
 - less than θ of ready instructions are stalled;
 - less than $2/3$ of the FIFOs are actually used on average;
 - more than 15% of dispatched instructions are non-critical;
 - current IQ occupancy rate is less than θ of the average occupancy rate

Specific Monitor Technique for Scheme # 1

- Enable one FIFO when either (ordered according to relative importance):
 - current issue rate (IPC_{issue}) drops by more than 10% compared to the last cycle window executed in FPM;
 - current IPC_{issue} drops by more than 15% compared to the previous cycle window;
 - more than $1/3$ of ready instructions are stalled

Results for Scheme # 1

Table 2. Results for Scheme #1.

Benchmarks	Avg. # of FIFOs	64-entry IQ			
		16, 4-entry FIFOs		64, 1-entry FIFOs	
		Power Saving	Δ IPC	Power Saving	Δ IPC
compress	7.5	51.4%	3.6%	75.9%	3.6%
gcc	11.1	29.8%	3.5%	65.2%	3.9%
go	11.7	25.8%	3.7%	63.2%	4.5%
jpeg	13.4	15.8%	2.4%	58.3%	5.1%
li	12.2	22.9%	5.8%	61.8%	2.7%
perl	12.8	19.6%	3.3%	60.1%	8.3%
average	11.5	27.6%	3.7%	64.1%	4.7%

Comments on Scheme # 1

- Only applied to integer benchmarks
- Reasonable job dynamically changing the 16 4-entry FIFOs
- But not as good for the non-FIFO (64 1-entry) scheme; but still for *compress* → 75% power saving with only 3.6% drop in performance
- Average best cases:
 - 16 4-entry FIFOs → 27.6% power saving with 3.7% drop in performance
 - 64 1-entry FIFOs → 64.1% power saving but 4.7% drop in performance (not as impressive)

Specific Monitor Techniques for Scheme # 2

- Halves the number of FIFOs & doubles the size of each FIFO when either (ordered according to relative importance) :
 - $(IPC_{issue} \hat{n} IPC_{commit}) > 1.0$;
 - less than 3% of ready instructions are stalled;
 - $IPC_{issue} < 2.7$ (threshold lowered by 0.2 for each successive reduction in number of FIFOs);
 - current IQ occupancy rate < 20% of average;
 - $(AVG_IPC_{issue} \hat{n} IPC_{issue}) > 0.15$ (threshold increased by 0.15 for each successive reduction in number of FIFOs)

Specific Monitor Techniques for Scheme # 2

- Double number of FIFOs and halves size of each FIFO when either (ordered according to relative importance):
 - current IPC_{issue} drops by > 8% compared to the last cycle window
 - current IPC_{issue} drops by > 6% compared to the last cycle window in FPM
 - more than 15% of ready instructions are stalled

FIFO usage for Scheme # 2

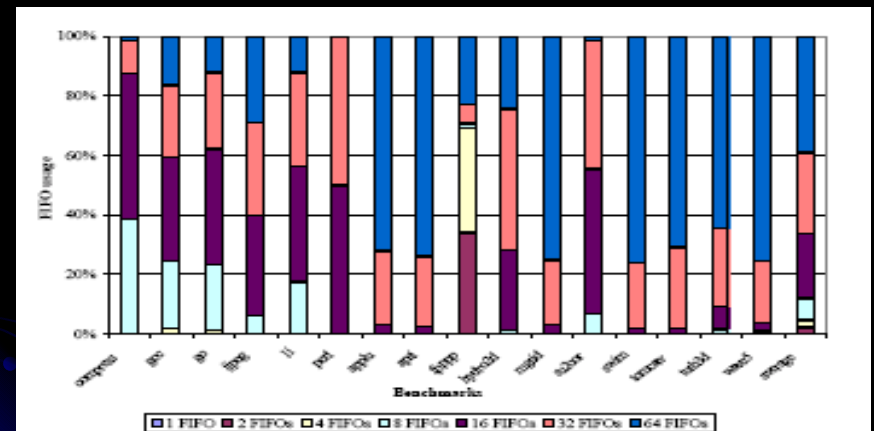


Figure 6. FIFO usage for Scheme #2. Note we always retain the number of IQ entries to 64.

Comments on FIFO usage

- For several FP benchmarks (*applu*, *apsi*, *mgrid* and *swim*), can't reduce number of FIFOs → need more flexibility in reordering instructions
- For most Integer benchmarks → cut the FIFOs at least in half for a significant portion of the running time

Results for Scheme # 2

Table 3. Results for Scheme #2.

Benchmarks	64-entry IQ			
	Power Saving			Δ IPC
	Request	Arbitration	Total	
compress	38.6%	75.9%	51.4%	0.7%
gcc	29.5%	59.9%	40.4%	2.4%
go	30.5%	62.4%	42.0%	3.5%
jpeg	21.3%	46.0%	30.8%	2.2%
li	28.8%	60.1%	40.4%	2.6%
perl	27.9%	62.4%	41.6%	2.5%
applu	6.0%	14.6%	9.7%	1.3%
apsi	5.6%	13.6%	9.0%	2.2%
fpppp	45.6%	73.2%	50.8%	3.0%
hydro2d	19.7%	44.9%	29.9%	4.2%
mgrid	5.4%	13.1%	8.7%	1.4%
su2cor	29.4%	64.0%	42.8%	4.7%
swim	5.0%	12.3%	8.1%	4.0%
tomcatv	6.1%	14.9%	9.9%	1.9%
turb3d	8.7%	20.1%	13.4%	3.2%
wave5	5.6%	13.2%	8.8%	2.9%
average	19.6%	40.7%	27.3%	2.7%

Comments on Scheme # 2

- Easier to cut number of FIFOs for integer benchmarks → save at least 30% of the issue queue power
- Most FP benchmarks need 64 FIFOs for a large % of running time but Scheme # 2 works reasonably well (*fppp*, *hydro2* and *su2cor*)
- Average: 27.3% power saving with only 2.7% drop in performance

Outline

- Introduction
- Focus of the paper
- Overview of Approaches Taken
- Related Work Done
- Implementations
- Experimental Results
- Conclusion

FINALLY!!!!!!!!!!



- Programs vary in ILP
- Dynamically reconfigure issue queue to save power
- Two approaches taken; Scheme # 2 works more efficiently
- **THANK YOU & BYE-BYE !!!!!**
- Oops .. ONE LAST THINGÖ ..

References



- Yu Bai and R. Iris Bahar. A Dynamically Reconfigurable Mixed In-Order/Out-of-Order Issue Queue for Power-Aware Microprocessors.
- James A. Farrell and Timothy C. Fischer. Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor.
- J.E. Smith. Advanced Computer Architecture 1 ìPower Efficient Architectureî Lecture Notes.
- K. Wilcox and S. Manne. Alpha processors: A history of power issues and a look to the future.