Energy-Aware Computing

Lecture 8: Dynamic micro-architecture level techniques in the processor core

Outline

- Energy efficiency of OoO processors
- Adapting issue queue “depth”
- Adapting issue width
- Speculation control
- Instruction throttling

Modern processors

- Modern processors employ a number of techniques to exploit ILP:
  - Multiple functional units
  - Out of order execution / dynamic scheduling
  - Multiple issue
  - Speculation
  - Branch prediction

How energy-efficient?

Zyuban, Kogge 2000, 2001

- Detailed simulation with energy models
- Derive energy per instruction as function of issue width: $E_i = IW^{\gamma}$
- EDP = energy/instr * cycles/instr = (energy/cycle)/$IPC^2 \approx IPC*E_i/IPC^2 = E_i/IPC = IW^{\gamma}/IPC$
- Empirically: $IPC = IW^{\alpha}$
- $EDP = IW^{\gamma-\alpha}$
- Energy efficient processor: energy x delay should **not** grow with increasing IPC
  - why?

Energy efficiency

- Performance improvement (α) outweighs the increase in energy (γ), with wider IW?
- Typically α is much less than 1
- Zyuban's models show γ generally > 1

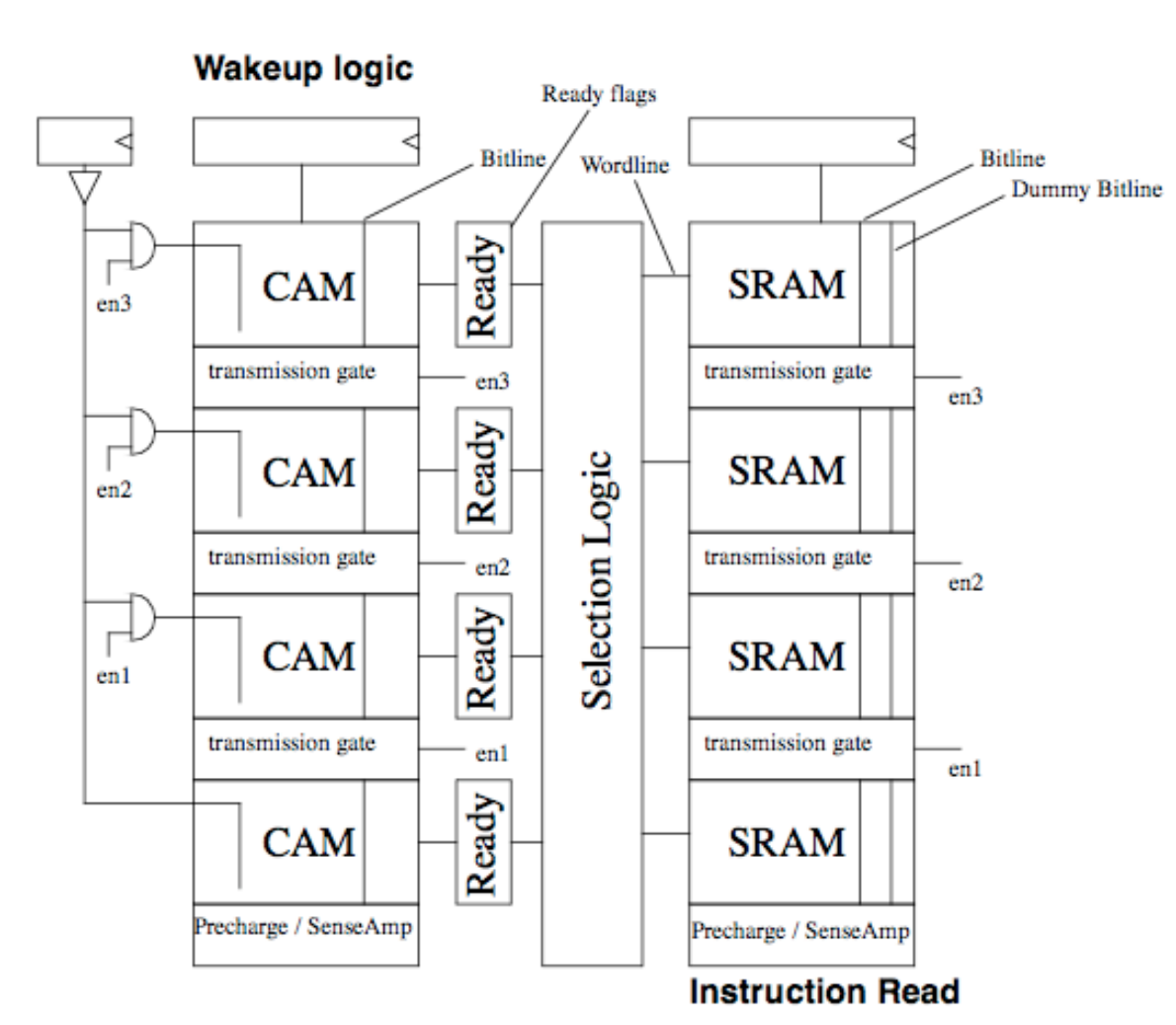
Rename logic	γ=1.1
Issue window/queue	γ=1.9
Multi-ported register file	γ=1.8
Data bypass	γ=1.6
Functional units	γ=0.1

Adaptive issue queue

Buyuktosunoglu et al 2001

- CAM/SRAM design
- SRAM holds instruction info
- CAM search for operand updates
  - *Wakeup* logic
- Break structure into chunks using circuits (transmission gates)
- Dynamically change the working size of the issue queue
- Circuit level simulations

Adaptive IQ


Wakeup logic
Ready flags
Bitline
Wordline
Bitline
Dummy Bitline
CAM
Ready
SRAM
en3
transmission gate
en3
en3
CAM
Ready
SRAM
en2
transmission gate
en2
en2
CAM
Ready
Selection Logic
SRAM
en1
transmission gate
en1
en1
CAM
Ready
SRAM
Precharge / SenseAmp
Precharge / SenseAmp
Instruction Read

When to adapt

- Monitor the activity of entries over a time window using hardware performance counters
- Compare with 2 thresholds to make decisions
- Safety mechanism: reverse last decision if IPC degrades too much

```
if (IPC < Dfactor * lastIPC) increase size
else if (counter < threshold1) decrease size
else if (counter < threshold2) retain current size
else increase size
```

Evaluation

Energy savings	8 entries	16 entries	24 entries
SRAM	45%	25%	10%
CAM	75%	61%	30%

- Most of energy of unit comes from CAMs continuously searching for their expected operand
- Total savings for 8 entries ~70%

Logical-resizing of IQ

Folegnani, Gonzalez 2001

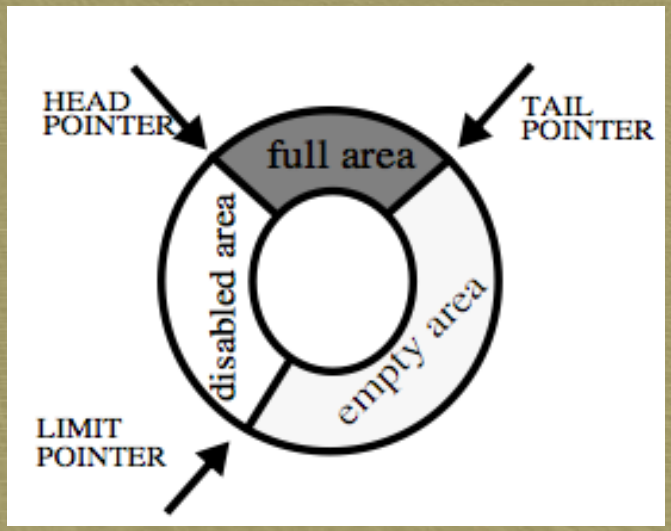
- Key observation: no need to attempt to wake-up empty and ready entries of IQ
- In a 128 entry IQ, 4-wide issue
  - 58 on average are in “full-area”
  - 26 of ***these*** are empty (already issued)
  - Only ¼ of IQ really needs to be searched for waking-up

Wake-up disabling

- Turn off pre-charging of match lines
  - when ready bit is on or when empty entry
  - relatively easy to implement
- Saves 89% of wake-up energy

IQ resizing

- Use disabling mechanism for resizing:
  - force a number of entries to be empty
  - new limit pointer

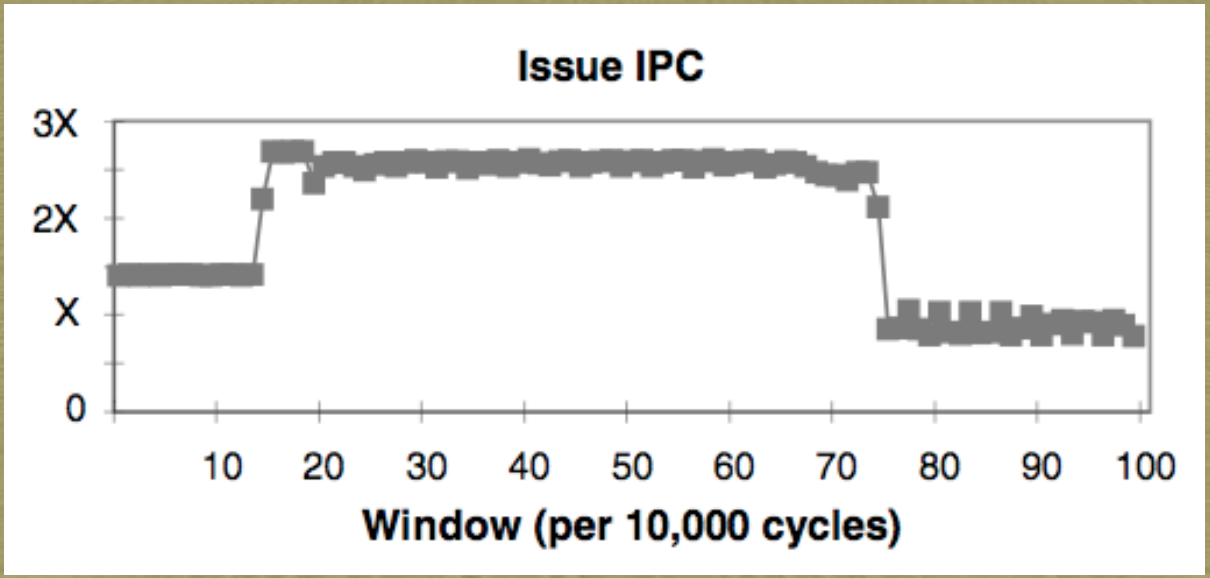


- Adjust size according to program needs
  - Periodically compare contribution to IPC by "youngest instructions" to empirical threshold
- Save 91% energy, 1.7% IPC loss

Issue queue adaptation

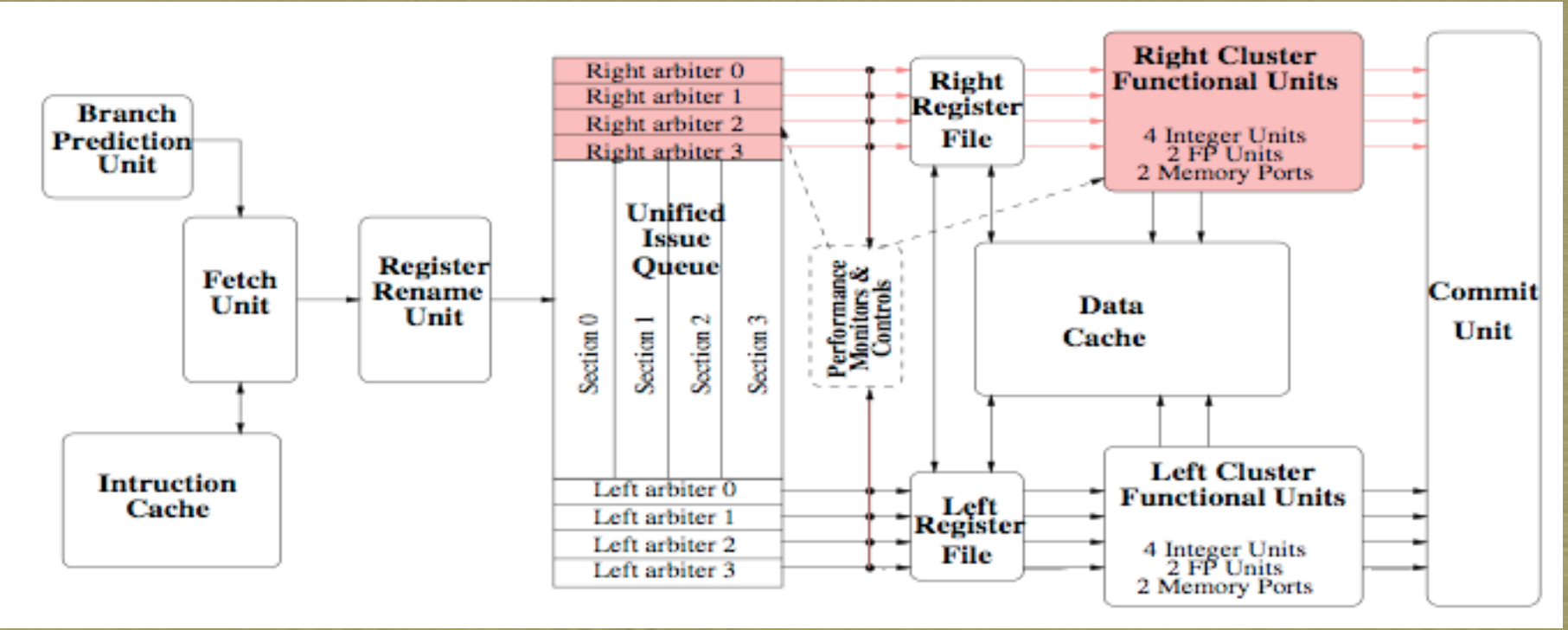
- IQ has worst γ factor
- Many other techniques published:
  - e.g. different feedback control mechanisms for increasing / decreasing Q size
    - E.g. based on the occupancy rather than on its "readyness"

Pipeline balancing



- The issued instructions per cycle can vary by as much as 3x
- This can be exploited to save energy
- Bahar, Manne 2001

Pipeline balancing



- Clustered architecture
- Dynamically change the “width” from 8 issue to 6 or 4 issue
- Instruction window does not change

When to adapt

- Sampling window measures issue IPC
- Decision affects next sampling window
- Rules:

Trigger	Threshold Values
EC_{4w}	$(I_{IPC} < 3.0)$ AND $(FP_{IPC} < 1.4)$ AND (mode history of 2 Consecutive Windows)
DC_{4w}	$(I_{IPC} > 3.2)$ OR $(FP_{IPC} > 1.6)$
EC_{6w}	$(I_{IPC} < 4.5)$ AND $(FP_{IPC} < 1.4)$
DC_{6w}	$(I_{IPC} > 5.0)$ OR $(FP_{IPC} > 1.6)$

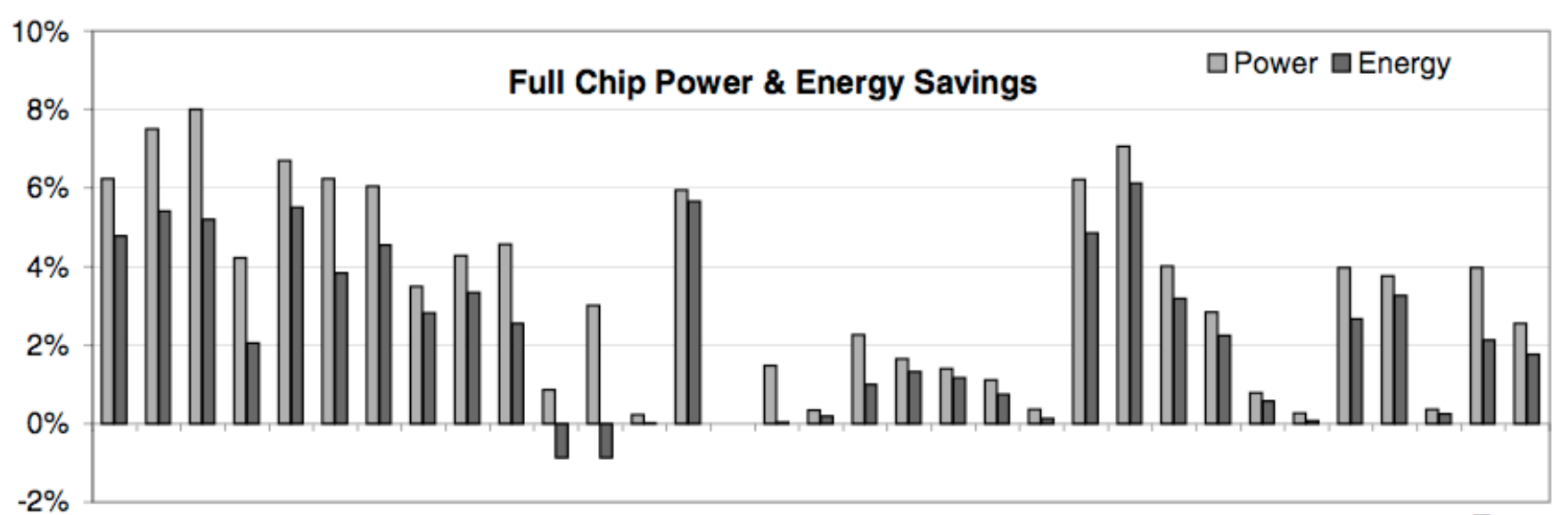
Evaluation

- Excellent, detailed evaluation
- Reading highly recommended to guide how you evaluate your own ideas

- Performance hit on the order of 1-2%

Evaluation: energy

Configuration	Component	%Savings
4-wide	Execution Unit	20%
	Issue Queue	35%
	Total Chip	12%
6-wide	Execution Units	10%
	Issue Queue	17%
	Total Chip	6%



Speculation control

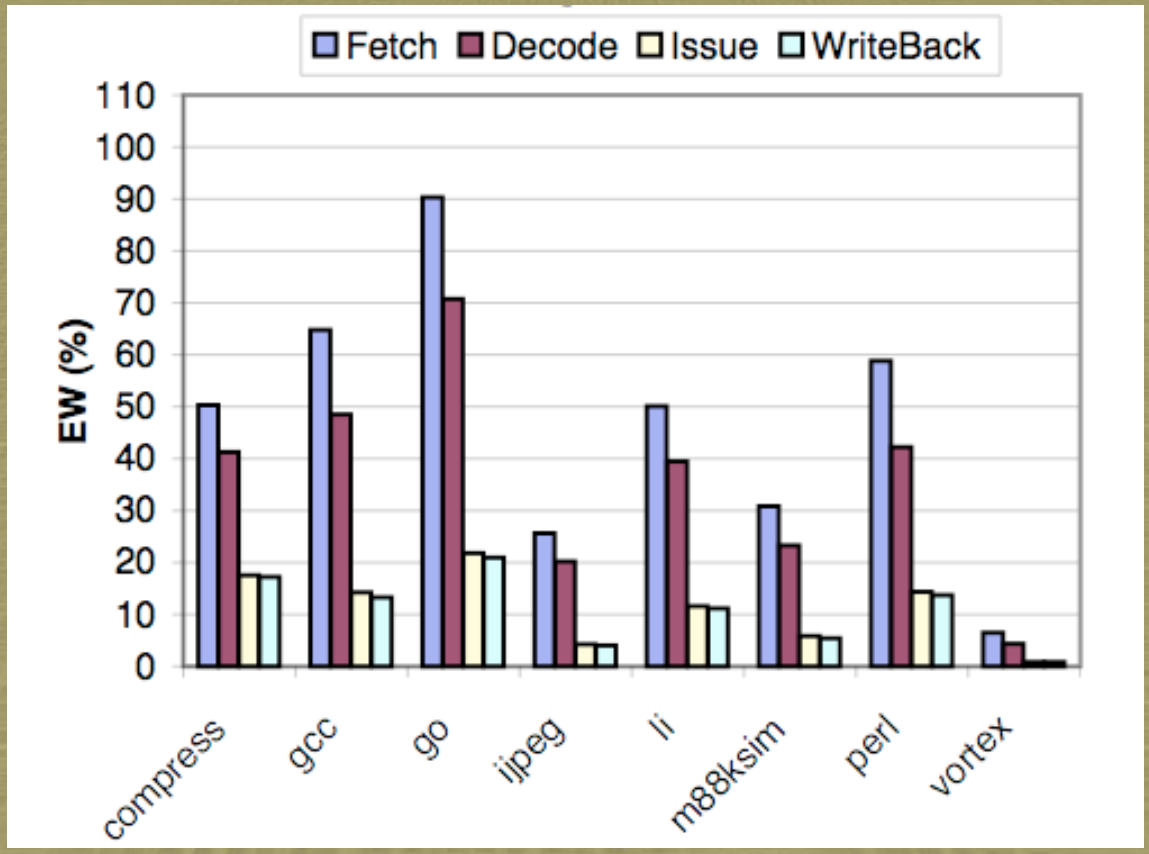
- Multi-issue, out-of-order processors must speculate over control dependencies to extract enough ILP
- Speculation could be bad for energy efficiency:
  - Wrong speculation. Useless work discarded
  - Energy expended in each cycle to support speculation. E.g. info kept to be able to restore the proc state

Pipeline gating

Manne, Klauser, Grunwald, ISCA 98

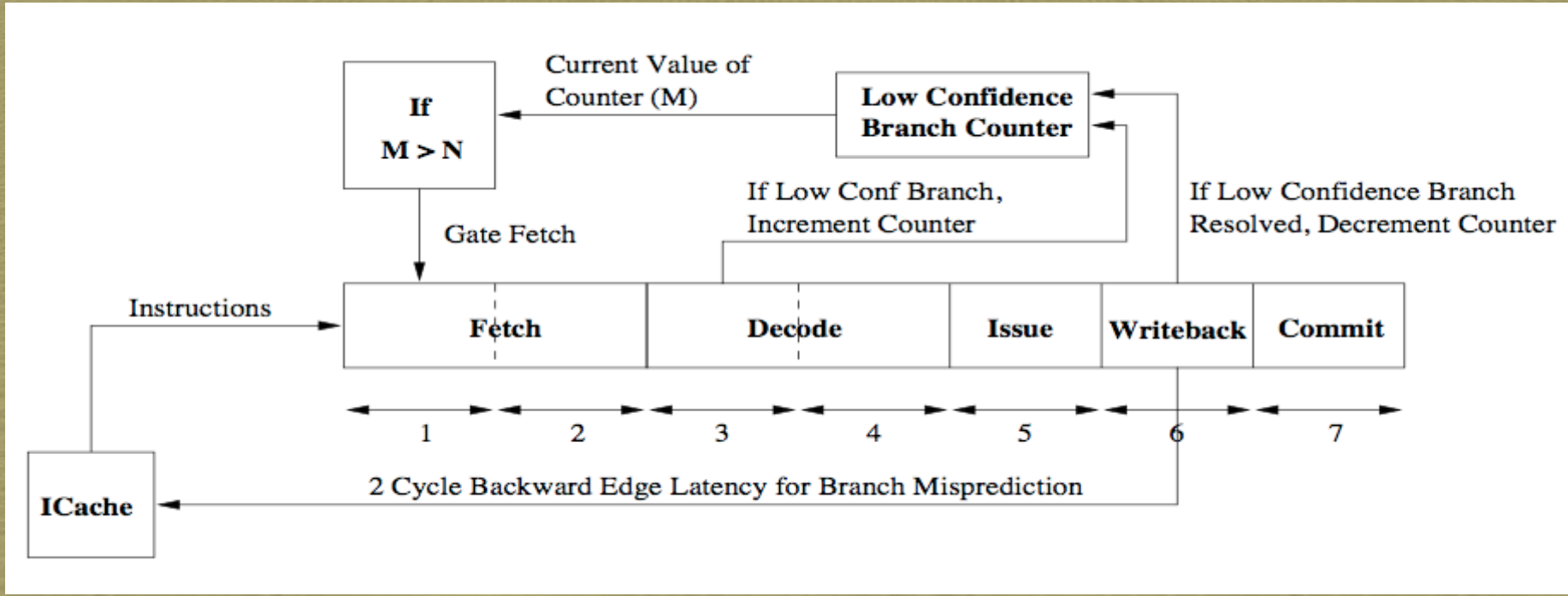
- Assumes a 4 way issue machine
- Add a confidence estimation mechanism to branch brediction
- When more than *x* low confidence branches in the pipeline,
  - Prob. new instructions executed is very low
  - Stop fetching I.e. gate the pipeline

Extra work

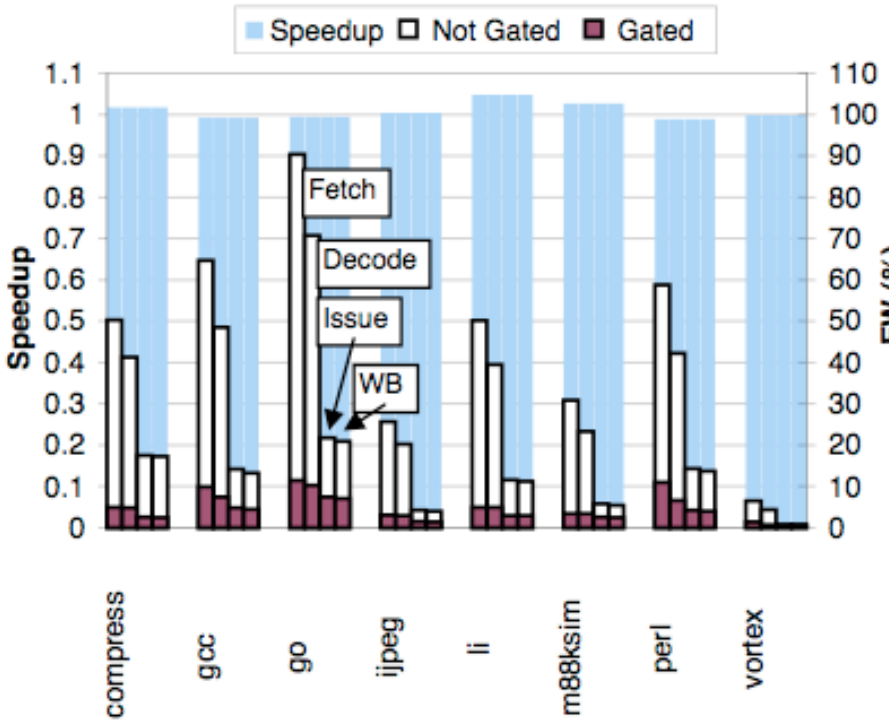


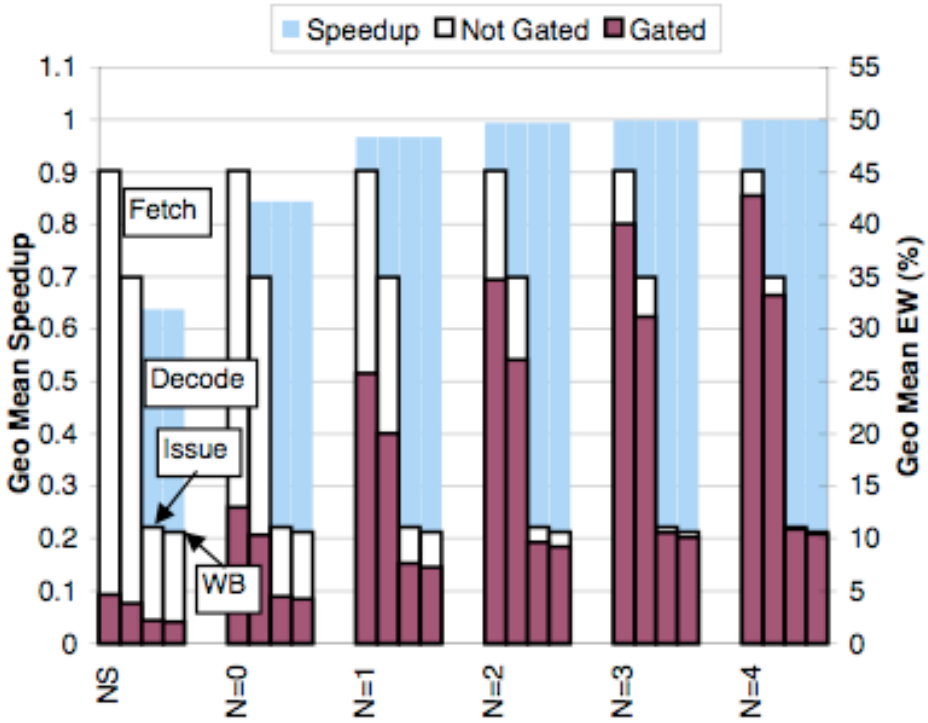
- Extra work (EW) the rate of number of instructions entering a stage to the number of committed instructions

Pipe-gating: implementation



Pipe-gating: results





Perfect confidence estimation

Actual results wrt threshold value

NS:non-speculative processor

Instruction throttling

- Ideally the throughput of a processor's front end (fetch/decode) should match that of the back end
- If front end is running too much ahead of back end we can selectively turn off fetch/decode to save power
- First used in 97 for thermal management of PowerPC processors
  - Halving the fetch rate drops speed by 12% and temperature by 7C

Instruction throttling

Baniasadi and Moshovos 2001

- Machine width: 8 instructions
- Decode/commit rate (DCR)
  - If commit rate too low, lots of unsuccessful speculation
- Dependence based (DEP)
  - If more dependencies than x, stop fetching
- Adaptive DCR/DEP
  - When commit rate is low, use DEP, else use DCR
- Slow down 3.6%, instr throughput drop 15% at fetch, 20% at decode

Summary

- Out-of-order machines inherently energy-inefficient
- A list of dynamic techniques for controlling idle capacity & speculation
  - Adaptive issue queue
  - Issue width adaptation (pipeline balancing)
  - Speculation control (pipe gating)
  - Instruction throttling