Fetch Engines and Databases

Carlos Navarro

Alex Ramírez

Josep-L. Larriba-Pey

Mateo Valero

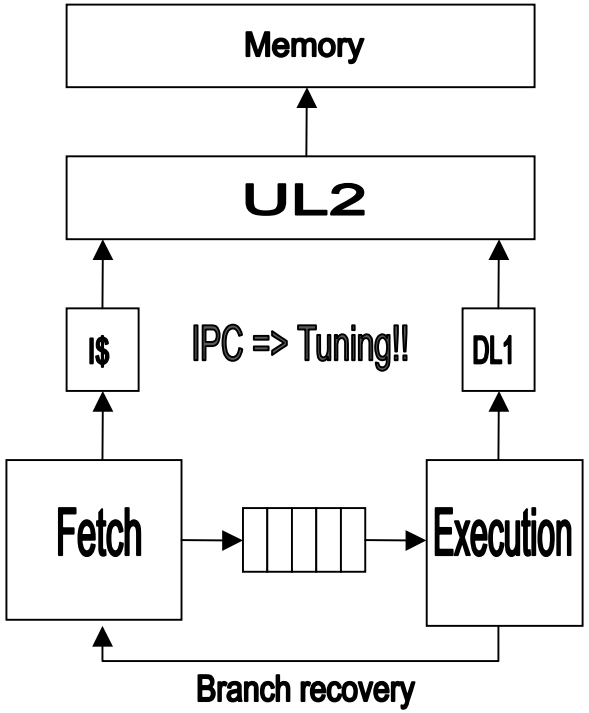
UPC-Barcelona

CEPBA

UPC

Fetch Engine

- Current microprocessors have a decoupled structure
- We can not execute faster than we fetch.
- Fetch characterised by:
  - Speed
    - Effective **latency :**
      - **Instruction cache (I-cache) hit rate**
    - Effective fetch **bandwidth :**
      - **Hardware limit**
      - **Execution of non-contiguous instructions**
  - ´Quality´ of instructions supplied
    - **Branch prediction accuracy.**
- All the three parameters should be considered



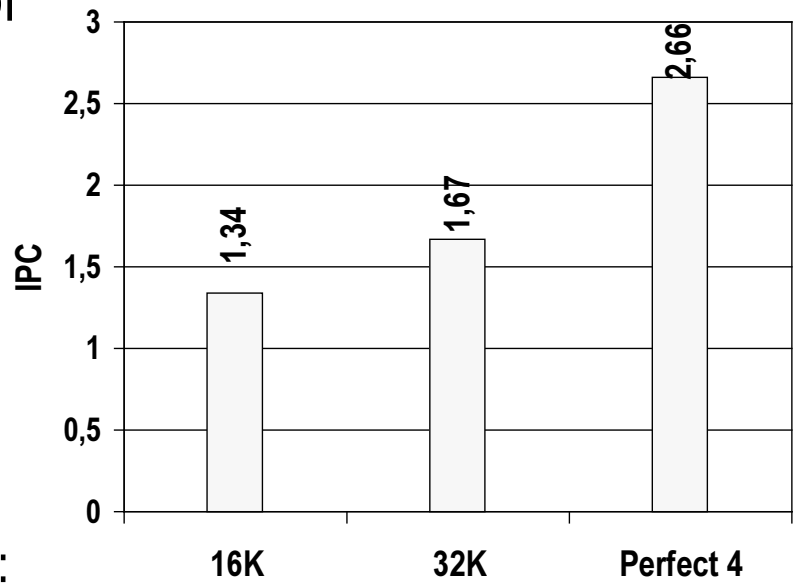
Databases

- DBMSs :
  - Highly structured codes
    - **Many procedure calls**
    - **Control of all error situations**
  - Big instruction working sets

- Extra pressure on conventional fetch mechanisms
  - Evidence in [Ailamaki, VLDB99]
    - **For several RDBMSs running on a Pentium II Xeon:**
      - **´I-cache stall time is a major memory stall component´, 20% of the execution time**
      - **´I-cache stall time is impossible to hide´**

Fetch Engine Impact on IPC

- High benefits can be expected of a better fetch
  - Even with low misprediction penalties!
- Perfect 4 speed-up:
  - Over 16K, Bimodal 12, Recover 1:
    - **98 %**
  - Over 32K, Bimodal 12, Recover 1:
    - **59 %**



Outline

- Workload and methodology
- I-cache effect on performance
- Branch prediction effect on performance
- Misprediction penalties
- Fetch bandwidth effect
- Results summary
- Using a code reordering technique
- Conclusions

Workload

- Relational DBMS
  - PostgreSQL 6.4
  - Academic, but well tuned
- DSS Workload:
  - TPC-D Queries 3 and 17
    - **Exhibit the higher I-cache miss rates**
  - 100 MB, B-Tree indexed database on primary and foreign keys

Methodology

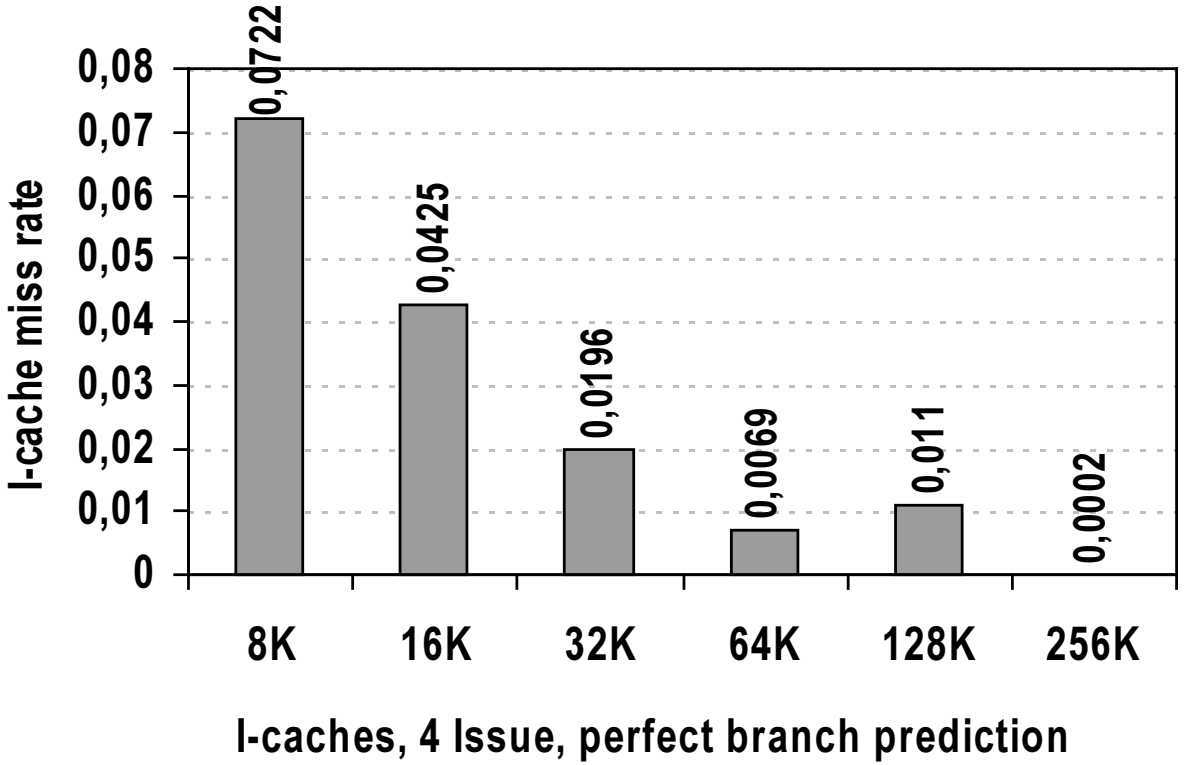
- Simplescalar´s Alpha v3.0 sim-outorder derived simulator
  - Full Runs of TPC-D Q3 and Q17 using a sampling technique
    - **50 Million instructions detailed / 100Million instructions fast**
    - **6 samples (300 Milion instructions)**
- Fetch engine setup:
  - Dynamic branch prediction and Icache based
  - 4 Issue processor:
    - **Limit of 4 instructions fetched per cycle**
    - **1 branch prediction per cycle**
  - 8 Issue processor:
    - **Limit of 8 instructions fetched per cycle**
    - **2 branch predictions per cycle up to first taken branch**

Methodology

Parameter	Value
Issue Width	4, 8
Window Size	64, 128
L/S queue size	64, 128
FU´s	Infinite, fully pipelined.
Memory	
Data L1	64KB / 32 bytes / 2 way.
Combined L2	2MB / 64 bytes / 4 way.
Latencies	1 / 7 / 80 cycles
TLB	
Instructions	16 entries / 4KB / 4 way
Data	32 entries / 4KB / 4 way
Branch Prediction	
Address	BTB 4K entries / 4 way
RAS	128 entries

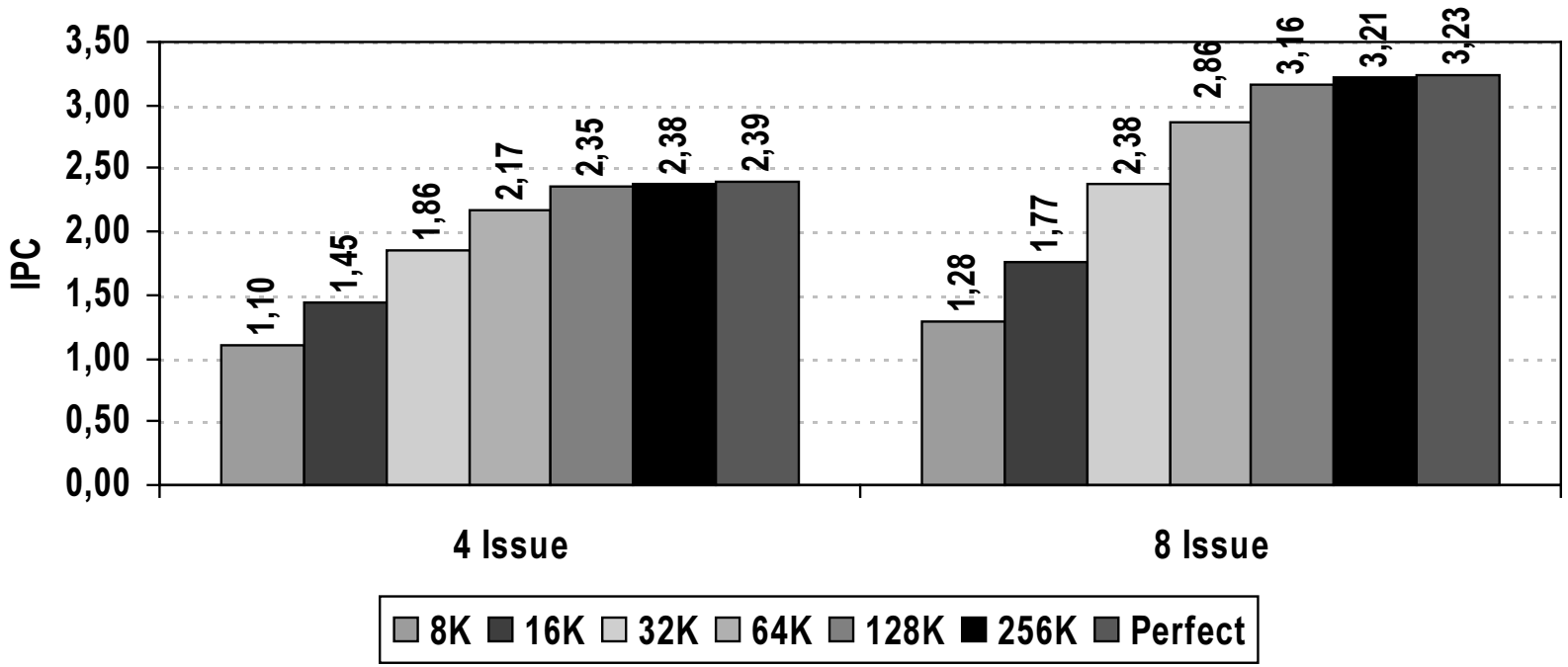
- Branch predictors
  - Bimodal
  - G-share
  - Hybrid
  - Perfect
- L1 I-caches:
  - 32 bytes, 2 way
  - 8, 16, 32, 64, 128, 256KB
  - Perfect
- Recovery cycles
  - 1, 3, 6, 12

I-cache performance



- Other branch predictors change the rates slightly, but the behaviour is the same

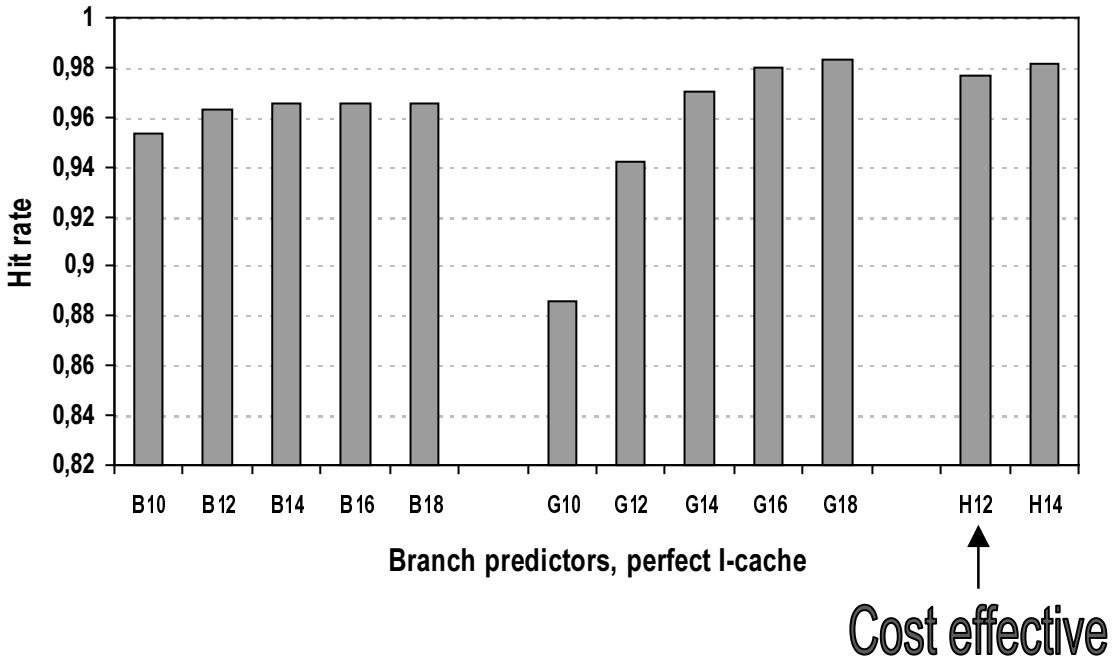
I-cache size effect on performance



Perfect Branch prediction, 1 cycle recover

- Untolerated instruction latencies have a big impact in overall performance
- The main working set must fit in the I-cache

Branch prediction performance

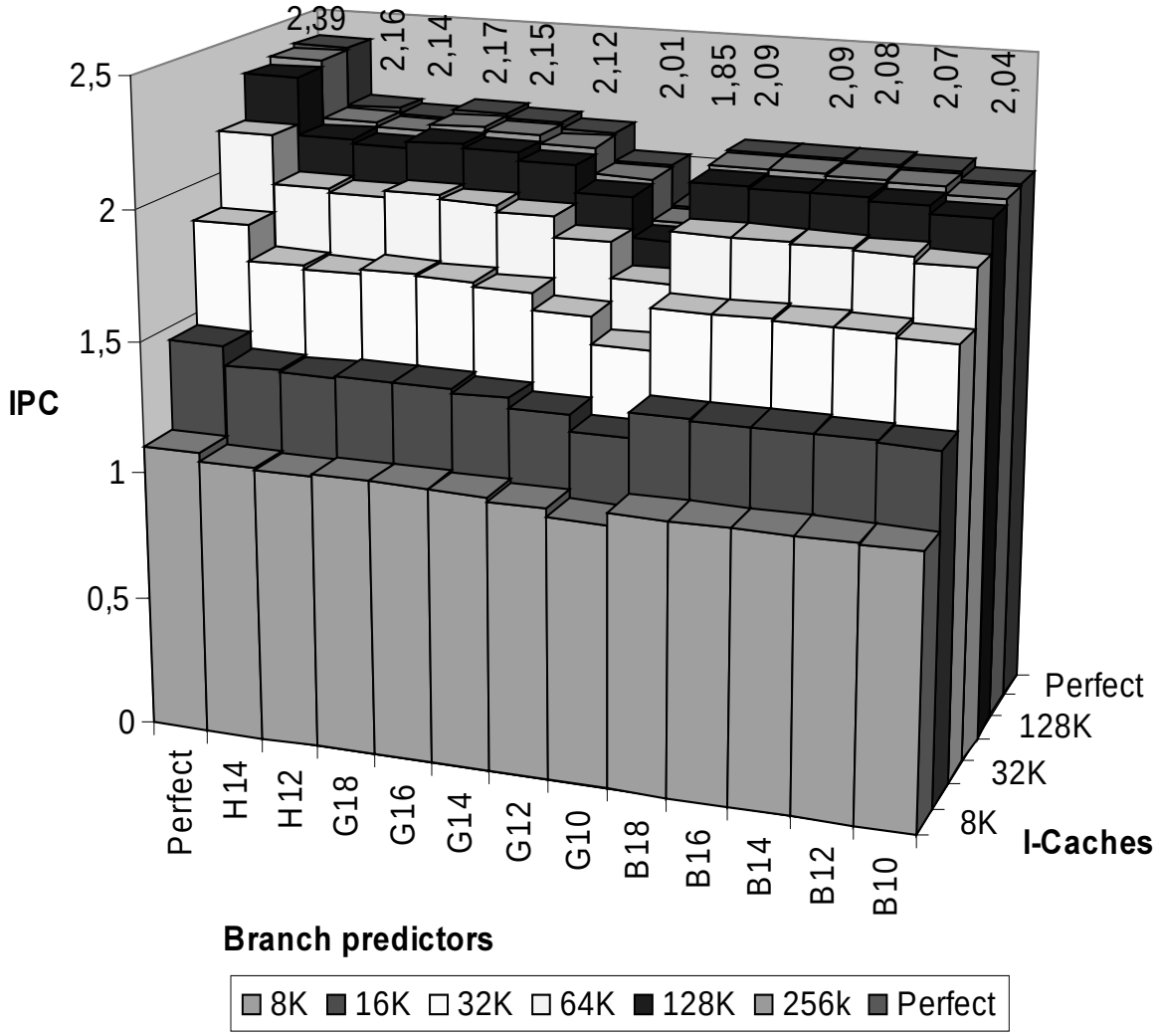


Predictor	Entries	Cost
B10 , G10	1K	256B
B12, G12	4K	1KB
B14, G14	16K	4KB
B16, G16	64K	16KB
B18, G18	256K	64KB
H12	12K	3KB
H14	48K	12KB

- Large difference on the performance of branch predictor schemes.

	Binary	Dyn. Q3, 17	St. Q3, 17
Insts.	376.288	1.087.240.704	59.068
Branches	24.801	106.249.585	4.583
Other control	38.015	74.238.735	5.179

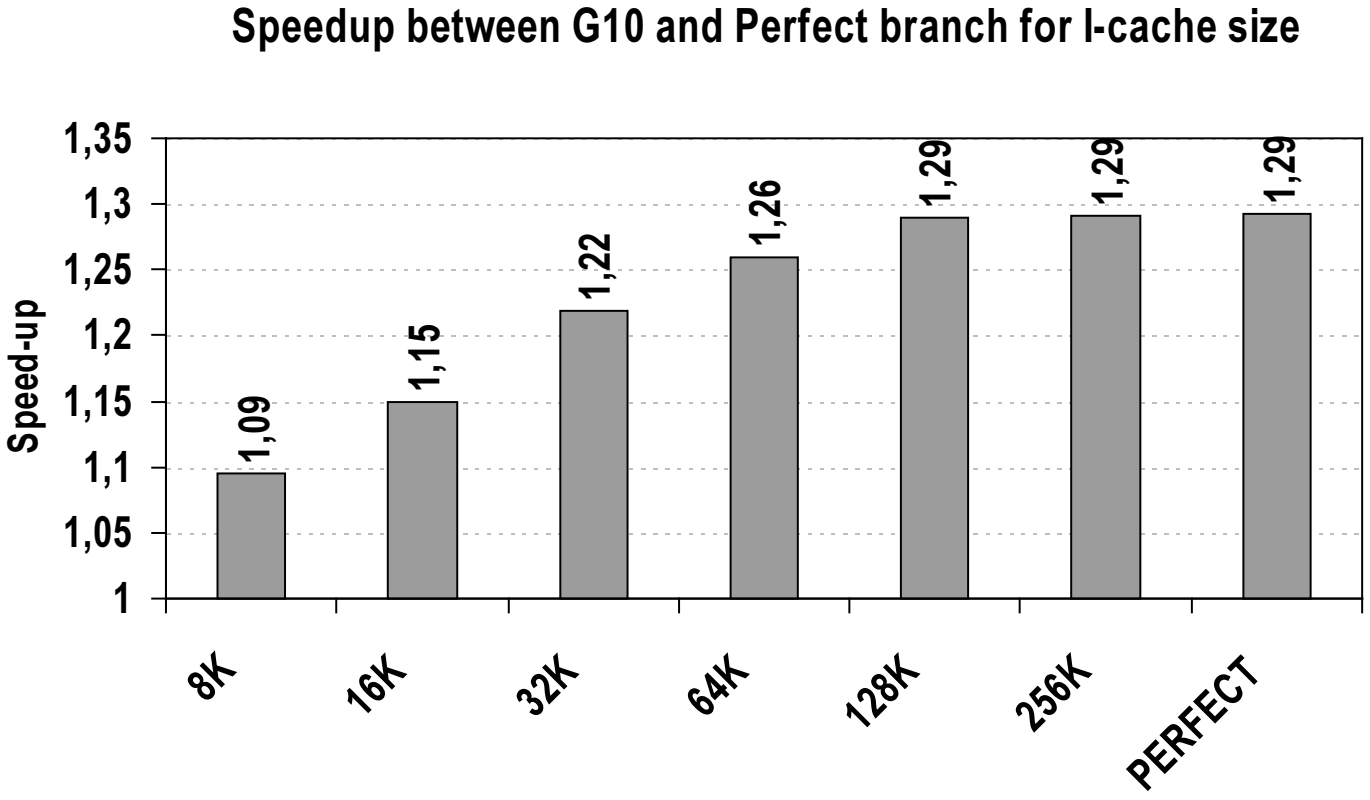
Branch prediction effect on performance



4 way Issue

1 cycle recovery

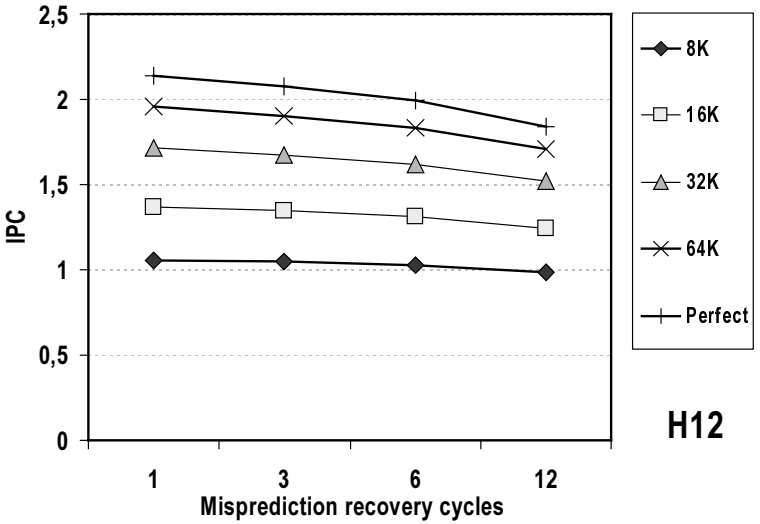
Branch prediction effect on performance



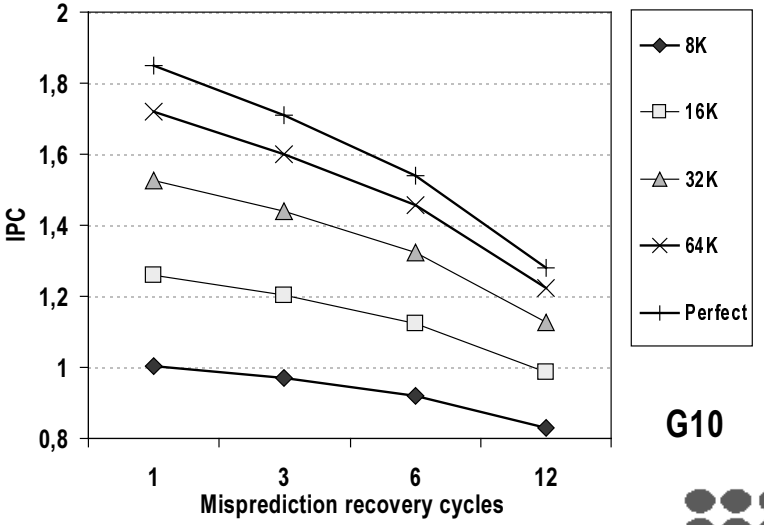
- High instruction latencies can easily hide the benefits of a better branch prediction

Playing with misprediction penalties

- For H12 the Icache still dominates
- For G10 the Icache dominates for sizes up to 16KB
- There is a trade off between:
  - Branch misprediction rate
  - Penalty
  - Icache ability in hiding latency

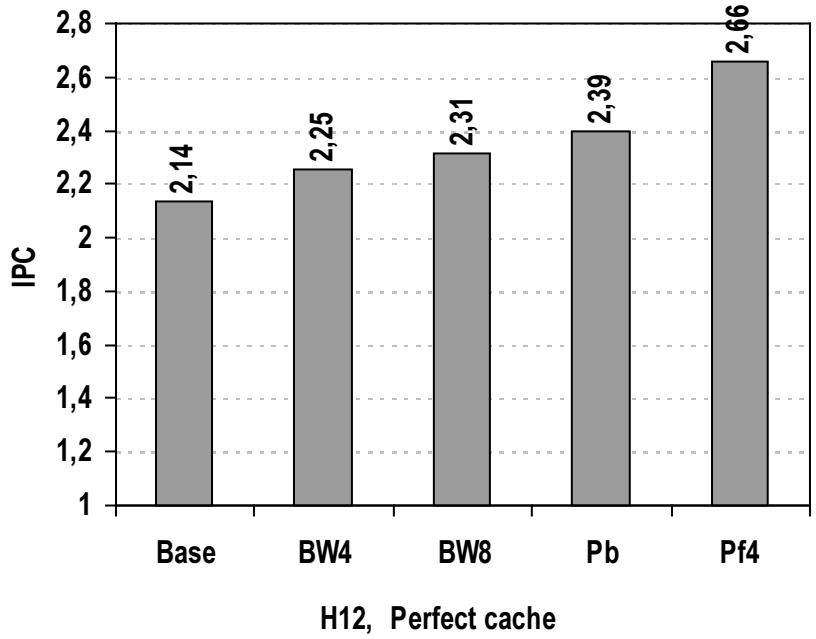


H12



G10

Fetch Bandwidth Effect



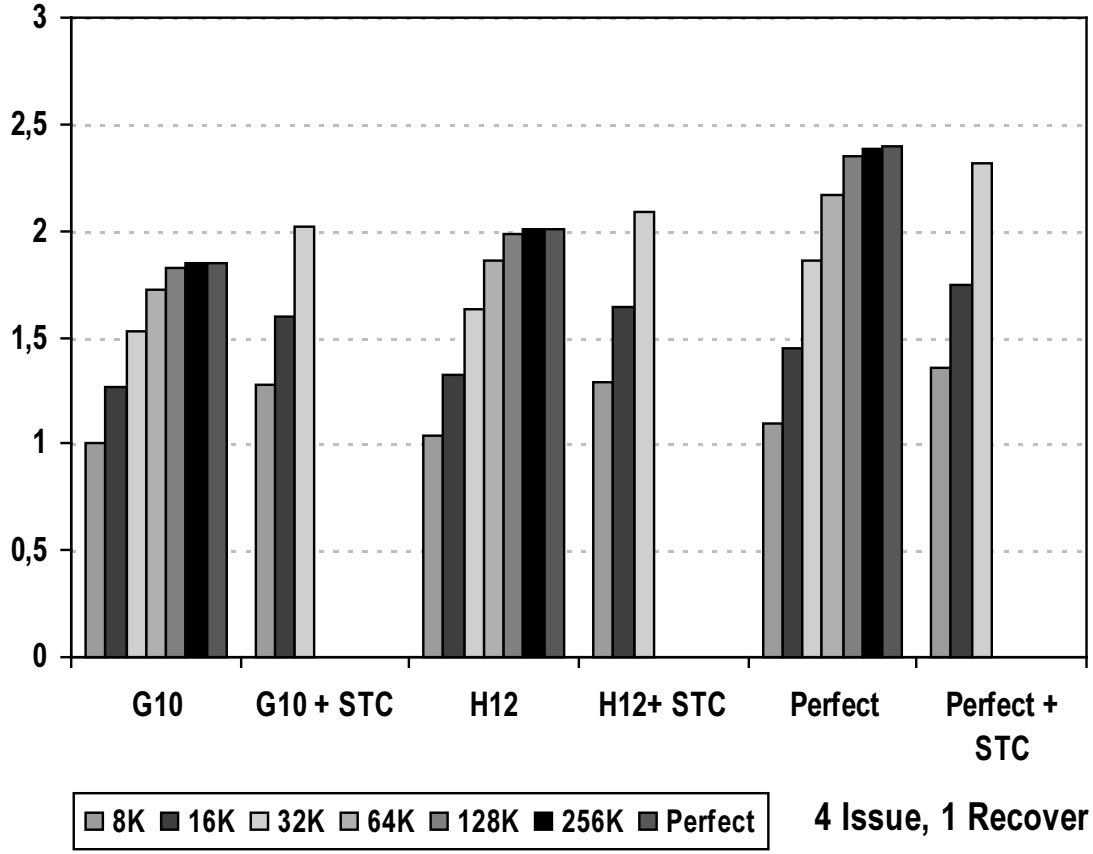
- Increasing fetch bandwidth:
  - Up to 4 insts.: 5.5% speed-up
  - Up to 8 insts.: 8.3% speed-up
- With BW8 we achieve 69% of the difference between Base and Pb.

Results summary

- Low Instruction latency must be the main target in a fetch engine design
- Is the I-cache the complete solution, to the latency hiding problem?
  - Codes sizes are growing
    - **Software developing techniques**
  - It does not scale up!
    - **Hardware fixed**
  - It is in the processor´s critical path
    - **Pipelining increases misprediction penalty!**
      - **Prefetching techniques**
      - **Code reordering**

Using a code reordering technique

- Software Trace cache reordering [Ramirez, ICS 99]
  - Increase BW
  - Reduce Icache misses.



Conclusions

- Bad Instruction fetch design has large impact on the IPC
- Due to their large instruction working sets, Instruction latency is the main fetch problem of DBMS
- Untolerated latencies can hide the benefits of a better branch prediction
- I-caches are fixed!
  - Code reordering
  - Prefetch techniques