Cache Performance Improvement

Dr A Sahu

Eleven Advanced Optimization for Cache Performance

- Reducing hit time
- Reducing miss penalty
- Reducing miss rate
- Reducing miss penalty * miss rate

Ref: 5.2, **Computer Architecture: A Quantitative Approach**, *Hennessy Patterson* Book, 4th Edition, PDF Version Available on Course website (Intranet)

Reducing Cache Hit Time

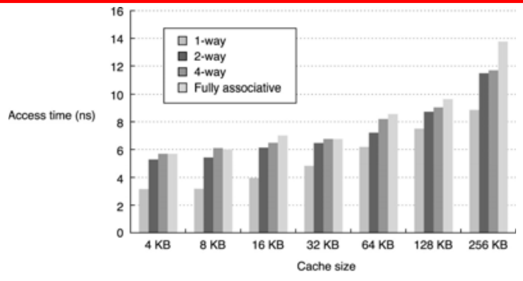
Reducing Hit Time

- Small and simple caches
- Pipelined cache access
- Trace caches
- Avoid time loss in address translation
  - Virtually indexed, physically tagged cache
    - simple and effective approach
    - possible only if cache is not too large
  - Virtually addressed cache
    - protection?, multiple processes?, aliasing?, I/O?

Small and Simple Caches

- Small size => faster access
- Small size => fit on the chip, lower delay
- Simple (direct mapped) => lower delay
- Second level – tags may be kept on chip

Cache access time estimates using CACTI



.8 micron technology, 1 R/W port, 32 b address, 64 b o/p, 32 B block

Pipelined Cache Access

- Multi-cycle cache access but pipelined
- reduces cycle time but hit time is more than one cycle
- Pentium 4 takes 4 cycles
- Greater penalty on branch misprediction
- More clock cycles between issue of load and use of data
  - **IF IF IF in pipeline**

A Sahu 7

Trace Caches: Pre-decoded

- What maps to a cache block?
  - not statically determined
  - decided by the dynamic sequence of instructions, including predicted branches
- Used in Pentium 4 (NetBurst architecture)
- starting addresses not *word size * powers of 2*
- Better utilization of cache space
- downside – same instruction may be stored multiple times

A Sahu 8

Reducing Cache Miss Penalty

A Sahu 9

Reducing Miss Penalty

- Multi level caches
- Critical word first and early restart
- Giving priority to read misses over write
- Merging write buffer
- Victim caches

A Sahu 10

Multi Level Caches

Average memory access time =

$\text{Hit time}_{L1} + \text{Miss rate}_{L1} * \text{Miss penalty}_{L1}$

Miss penalty_{L1} =

$\text{Hit time}_{L2} + \text{Miss rate}_{L2} * \text{Miss penalty}_{L2}$

A Sahu 11

Critical Word First and Early Restart

- Read policy: Concurrent Read/Forward
- Load policy: Wrap around load

More effective when block size is large

A Sahu 12

Read Miss Priority Over Write

- Provide write buffers
- Processor writes into buffer and proceeds (for write through as well as write back)

On read miss
- wait for buffer to be empty, or
- check addresses in buffer for conflict

A Sahu 13

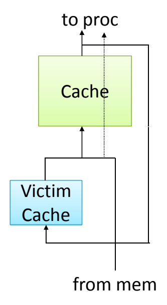
Merging Write Buffer

Merge writes belonging to same block in case of write through

A Sahu 14

Victim Cache: Recycle bin/Dust bin

- Evicted blocks are recycled
- Much faster than getting a block from the next level
- Size = 1 to 5 blocks
- A significant fraction of misses may be found in victim cache



A Sahu 15

Reducing Cache Miss Rate

A Sahu 16

Reducing Miss Rate

- Large Block Size
- Larger Cache
- Higher Associativity
- Way prediction and pseudo-associative cache
- Compiler optimizations

A Sahu 17

Large Block Size

- Take benefit of spatial locality
- Reduces compulsory misses
- Too large block size - misses increase
- Miss Penalty increases

A Sahu 18

Large Cache

- Reduces capacity misses
- Hit time increases
- Keep small L1 cache and large L2 cache

Higher Associativity

- Reduces conflict misses
- **8-way is almost like fully associative**
- **Hit time increases: What to do ?**
  - **Pseudo Associativity**

Way Prediction and Pseudo-associative Cache

Way prediction: low miss rate of SA cache with hit time of DM cache

- Only one tag is compared initially
- Extra bits are kept for prediction
- Hit time in case of mis-prediction is high

Pseudo-assoc. or column assoc. cache: get advantage of SA cache in a DM cache

- Check sequentially in a pseudo-set
- Fast hit and slow hit

Compiler optimizations

- Loop interchange
  - Improve spatial locality by scanning arrays row-wise
- Blocking
  - Improve temporal and spatial locality

Improving Locality

Matrix Multiplication example

$$[C] = [A] \times [B]$$

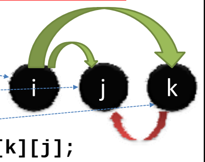
$L \times M$ $L \times N$ $N \times M$

Cache Organization for the example

- Cache line (or block) = 4 matrix elements.
- Matrices are stored row wise.
- **Cache can't accommodate a full row/column.**
  - **L, M and N are so large w.r.t. the cache size**
  - After an iteration along any of the three indices, when an element is accessed again, it results in a miss.
- Ignore misses due to conflict between matrices.
  - As if there was a **separate cache for each matrix**.

Matrix Multiplication : Code I

```
for (i = 0; i < L; i++)
  for (j = 0; j < M; j++)
    for (k = 0; k < N; k++)
       C[i][j] += A[i][k] * B[k][j];
```

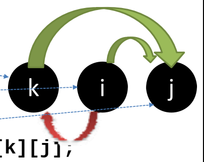


	C	A	B
accesses	LM	LMN	LMN
misses	LM/4	LMN/4	LMN

Total misses = LM(5N+1)/4

Matrix Multiplication : Code II

```
for (k = 0; k < N; k++)
  for (i = 0; i < L; i++)
    for (j = 0; j < M; j++)
       C[i][j] += A[i][k] * B[k][j];
```

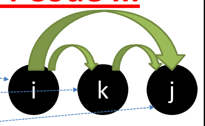


	C	A	B
accesses	LMN	LN	LMN
misses	LMN/4	LN	LMN/4

Total misses = LN(2M+4)/4

Matrix Multiplication : Code III

```
for (i = 0; i < L; i++)
  for (k = 0; k < N; k++)
    for (j = 0; j < M; j++)
       C[i][j] += A[i][k] * B[k][j];
```



	C	A	B
accesses	LMN	LN	LMN
misses	LMN/4	LN/4	LMN/4

Total misses = LN(2M+1)/4

Reducing MissRate*MissPenality

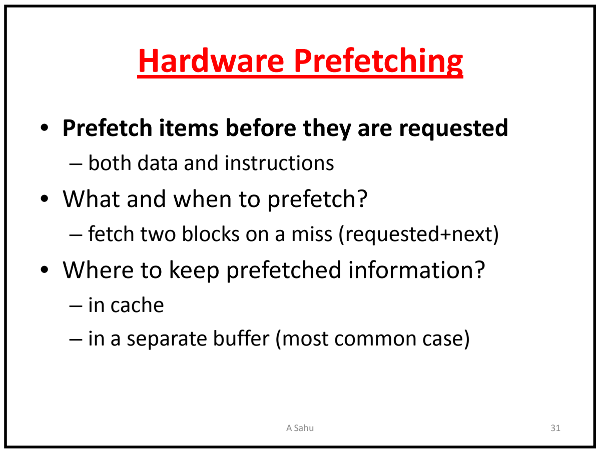
Reducing Miss Penalty * Miss Rate

- Non-blocking cache
- Hardware prefetching
- Compiler controlled prefetching

Non-blocking Cache

In OOO processor

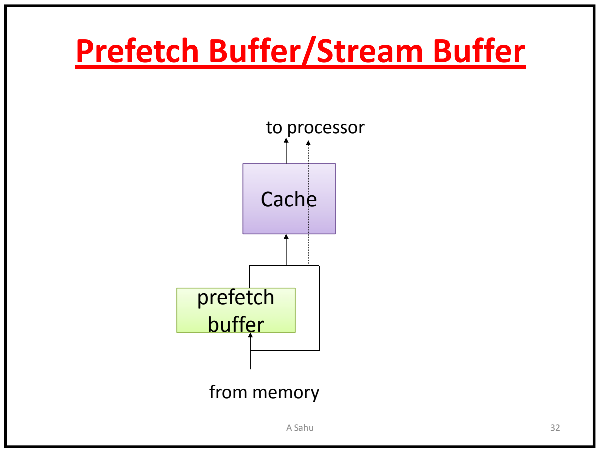
- Hit under a miss
  - complexity of cache controller increases
- Hit under multiple misses or miss under a miss
  - memory should be able to handle multiple misses



Hardware Prefetching

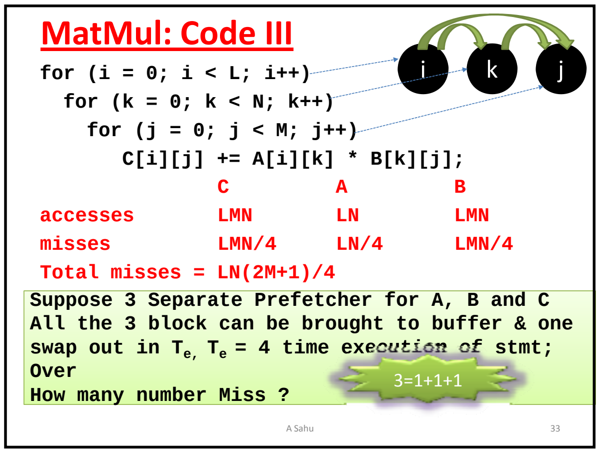
- **Prefetch items before they are requested**
  - both data and instructions
- What and when to prefetch?
  - fetch two blocks on a miss (requested+next)
- Where to keep prefetched information?
  - in cache
  - in a separate buffer (most common case)

A Sahu 31



Prefetch Buffer/Stream Buffer

A Sahu 32



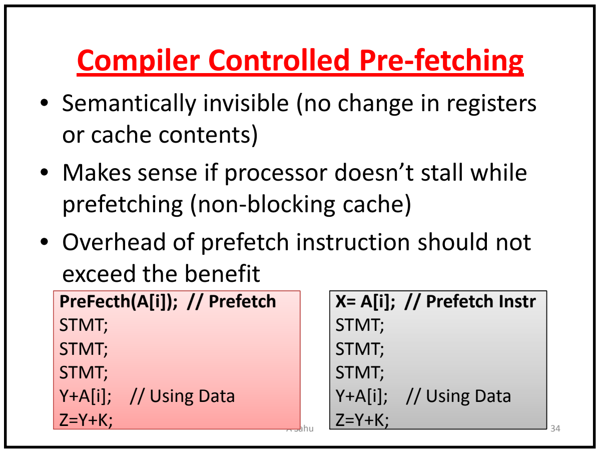
MatMul: Code III

```
for (i = 0; i < L; i++)
  for (k = 0; k < N; k++)
    for (j = 0; j < M; j++)
       C[i][j] += A[i][k] * B[k][j];
               C          A          B
accesses       LMN        LN         LMN
misses         LMN/4      LN/4       LMN/4
Total misses = LN(2M+1)/4
```

Suppose 3 Separate Prefetcher for A, B and C
All the 3 block can be brought to buffer & one swap out in T_e, T_e = 4 time execution of stmt; Over
How many number Miss ?

3=1+1+1

A Sahu 33

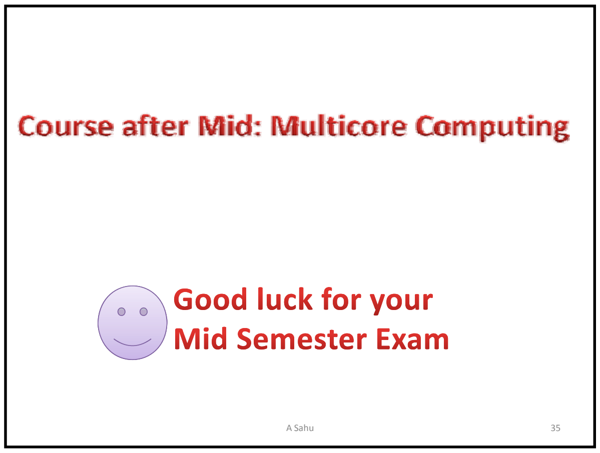


Compiler Controlled Pre-fetching

- Semantically invisible (no change in registers or cache contents)
- Makes sense if processor doesn't stall while prefetching (non-blocking cache)
- Overhead of prefetch instruction should not exceed the benefit

```
PreFecth(A[i]); // Prefetch      X= A[i]; // Prefetch Instr
STMT;                            STMT;
STMT;                            STMT;
STMT;                            STMT;
Y+A[i];   // Using Data          Y+A[i];   // Using Data
Z=Y+K;                           Z=Y+K;
```

A Sahu 34



Course after Mid: Multicore Computing

Good luck for your Mid Semester Exam

A Sahu 35