Reducing Hit Time

Techniques:

- Make small, simple, and fast caches
  - Use SRAM instead of DRAM, but,'
- Pipeline cache access Multiple IF stages
  - Multiple IF stages, but,‘
- Cache indexing during address translation

Average Memory Access Time

For a system with a cache:

AMAT –
Cache-friendly Code

- Focus on inner loops of core functions
- Minimize cache misses
  - Temporal locality: repeatedly reference the same variables.
  - \textit{Stride-k} reference patterns: the smaller the \( k \), the better the spatial locality.
  - \textsc{LOOPS} have good spatial and temporal locality (at least wrt instructions): the tighter the loop, the better.

---

Strides and arrays

\textbf{Stride-}k: repeated memory accesses at distance \( k \), defined with respect to some object.

\textbf{Example:}

```c
int sumArray(int A[N])
{
    int i, sum = 0;
    for(i = 0; i < N; ++i)
        sum += A[i];
    return(sum);
}
```

---

Reducing the Miss Rate

- Larger block size reduces... by exploiting spatial locality BUT
  - increases...
  - increases...
- Larger caches reduce...
  - increases...
  - increases...
- Higher associativity — reduces...
  - increases...

“2:1 cache rule of thumb”: a \textit{direct-mapped cache of size } \( N \) \textit{has the same miss rate as a 2-way set-associative cache of size } \( N/2 \).

- Cache-friendly code — to increase both spatial and temporal locality

---

Reducing the Miss Rate — “The Three C’s”

- \textbf{Compulsory} misses
- \textbf{Capacity} misses
- \textbf{Conflict} misses
**Example:** `sumArrayCols()`

```c
int sumArrayCols(int A[][C])
{
    int i, j, sum = 0;
    for(j = 0; j < C; ++j)
        for(i = 0; i < R; ++i)
            sum += A[i][j];
    return(sum);
}
```

Stride: ...

Miss rate: ...

Assume: `sizeof(int) = 4`, 16-byte cache block, coalesced, C is multiple of 4 (for simplicity).

---

**Example:** `sumArrayRows()`

```c
int sumArrayRows(int A[][C])
{
    int i, j, sum = 0;
    for(i = 0; i < R; ++i)
        for(j = 0; j < C; ++j)
            sum += A[i][j];
    return(sum);
}
```

Stride: ...

Miss rate: ...

Assume: `sizeof(int) = 4`, 16-byte cache block, coalesced, C is multiple of 4 (for simplicity).
Example: sumArrayBigBad()

```c
#define N 512
double BIG[N][N];

double sumArrayBigBad(double BIG[N][N])
{
    int i, j;
    double sum = 0.0;
    for(i = 0; i < N; ++i)
        for(j = 0; j < N; ++j)
            sum += BIG[j][i];
    return(sum);
}
```

**BIG[N][N] (and BAD!)**

```plaintext
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>511</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>510</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>511</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Example: sumArrayBIG() (cont.)

Assume: array elements aligned in memory. L1 cache only cold, 16-byte cache block, 100 clock cycle miss penalty, 4 KB page size, 1,000,000 clock cycle page fault, no cached data pages.

- Stride:
- Total # of memory references:
- Data cache miss rate:
- Total # of page faults
  - with 2 MB max available memory:
  - with 1 MB max available memory:
- Total # of clock cycles for data memory stalls:
  - with 2 MB max available memory:
  - with 1 MB max available memory:
**Miss Penalty**

**Critical Word First and Early Restart**

These techniques can only be applied to multi-word caches (i.e., with “large” data block).

**Critical Word First**: fetch from memory the missed word first, send it to the CPU, and let the CPU continue execution while receiving the remaining words in the block.

**Early Restart**: fetch the words in normal order, but as soon as the requested work arrives, send it to the CPU and let the CPU continue executions.

---

**Miss Rate**

**Cache-friendly Code**

*Example: `sumArrayBigBAD()` (cont.)*

Assume: array elements aligned in memory, L1 cache only, cold, 16-byte cache block, 100 clock cycle miss penalty, 4 KB page size, 1,000,000 clock cycle page fault, no cached data pages.

- **Stride:**
  - Total # of memory references:
  - Data cache miss rate:
  - Total # of page faults
    - with 2 MB max available memory:
      - with 1 MB max available memory:
  - Total # of clock cycles for data memory stalls:
    - with 2 MB max available memory:
    - with 1 MB max available memory:

---

**Miss Penalty**

**Multilevel Caches**

**Multilevel Caches**

AMAT = Miss Penalty (L1) = Miss Penalty (L2) = Average Memory Stalls Per Instruction =

---

**Reducing Miss Penalty**

**Techniques:**

- Critical Word First
- Early Restart
- Multilevel Caches
- Victim Caches
**Miss Penalty  Victim Caches**

**Victim Caches**

A small, fully-associative cache between the regular cache and the next level in the hierarchy “recycles” blocks discarded because of a miss “victims.”

Reduces miss penalty and miss rate.

**Miss Penalty  Multi-Level Caches**

**Example 1**

In a program, 50% of the instructions have data memory references. In 1000 memory references (both Instructions and Data) there are 40 L1 misses and 20 L2 misses, Assuming that: Hit Time(L1) = 1 clock, Hit Time(L2) = 10 clocks, and Miss Penalty(L2) = 100 clocks, determine:

- Miss Rate(L1) =
- Miss Rate(L2) =
- AMAT\(_{L1}\), \(_{L2}\) =
- AMSPI\(_{L1}\), \(_{L2}\) =
- Average CPI\(_{L1}\), \(_{L2}\) =

NOTE that Miss Rate(L2) = (# of L2 misses) / (# of L2 references)

**Example 2**

With the same parameters of Example 1 but with no L2 cache, determine:

- AMAT\(_{L1}\) only =
- AMSPI\(_{L1}\) only =
- Average CPI\(_{L1}\) only =
- Performance improvement with L2 cache:
  - AMAT\(_{L1}\) only / AMAT\(_{L1}\), \(_{L2}\) =
  - AMSPI\(_{L1}\) only / AMSPI\(_{L1}\), \(_{L2}\) =
  - CPI\(_{L1}\) only / CPI\(_{L1}\), \(_{L2}\) =