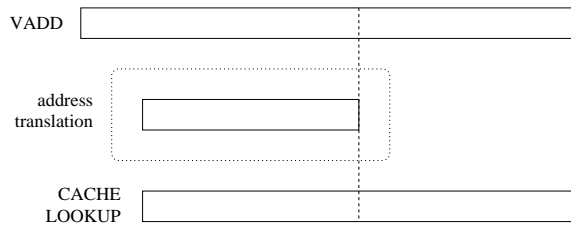


Reducing Hit Time

Techniques:

- Make small and simple caches
- Pipeline cache access — multiple IF stages
 - Any problems?
- Cache indexing during address translation



CMPE110 Fall 2003 More On Memory

- Average Memory Access Time
- Reducing Hit Time
- Reducing Miss Rate
- Reducing Miss Penalty



Virtual versus Physical Caches

- Easier page-level protection with physical(ly-addressed) caches
- Same virtual addresses in different processes refer to different physical addresses:
 - flush the cache at every context switch
 - add a PID extension
- *Synonyms* (or *aliases*): the OS and user programs may have different virtual addresses point to the same shared physical address.
 - PB with virtual(ly-addressed) caches?



Average Memory Access Time

AMAT =



Cache-friendly Code

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



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

Three kinds of misses:

Compulsory misses

Capacity misses

Conflict misses



Strides and arrays

Stride-k: repeated memory accesses at distance k , defined with respect to some object.

Example:

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

VMEM



Reducing the Miss Rate

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

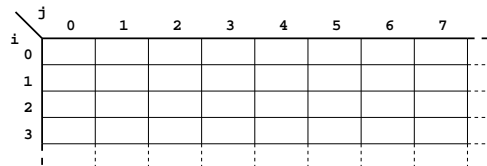
“2:1 cache rule of thumb:” *a direct-mapped cache of size N has the same miss rate as a 2-way set-associative cache of size $N/2$.*
- Cache-friendly code — to increase both spatial and temporal locality



Example: sumArrayCols()

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

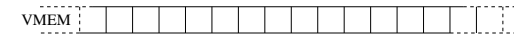
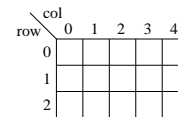
Stride =

Assume: `sizeof(int) = 4`, single cache, 16-byte cache block, cold cache.

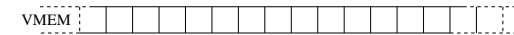
references to A, H or M

**Array Access and Allocation**

Row-Major and Column-Major:



ROW-MAJOR memory allocation



COLUMN-MAJOR memory allocation

**Example: sumArrayBIG()**

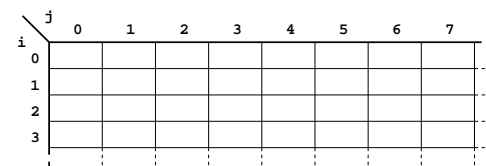
```
#define N 64
double A[N][N][N];

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

**Example: sumArrayRows()**

```
int sumArrayRows(int A[R][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 =

Assume: `sizeof(int) = 4`, single cache, 16-byte cache block, cold cache.

references to A, H or M



Example: sumArrayBigBAD() (cont.)

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

- Stride =
- Total # of memory references =
- Data cache miss rate =
- Total # of page faults =
- Total # of clock cycles for data memory stalls =



Example: sumArrayBIG() (cont.)

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

- Stride =
- Total # of memory references =
- Data cache miss rate =
- Total # of page faults =
- Total # of clock cycles for data memory stalls =



Reducing Miss Penalty

Techniques:

- Critical Word First
- Early Restart
- Multilevel Caches
- Victim Caches



Example: sumArrayBigBAD()

```
#define N 64
double A[N][N][N];

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



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 cc, Hit Time(L2) = 10 cc, and Miss Penalty(L2) = 100 cc, determine:

- Miss Rate(L1) =
- Miss Rate(L2) =
- $AMAT_{L1, L2} =$
- $AMSPI_{L1, L2} =$
- Average $CPI_{L1, L2} =$

NOTE that $Miss\ Rate(L2) = (\# \text{ of L2 misses}) / (\# \text{ of L2 references})$



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



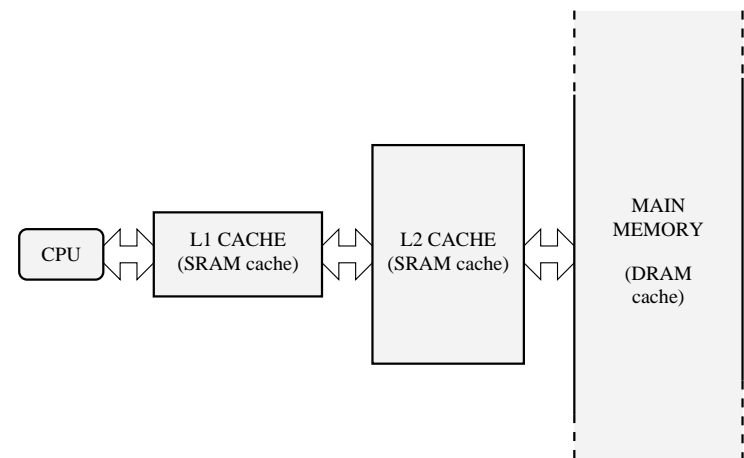
Example 2

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

- $AMAT_{L1 \text{ only}} =$
- $AMSPI_{L1 \text{ only}} =$
- Average $CPI_{L1 \text{ only}} =$
- Performance improvement with L2 cache:
 - $AMAT_{L1 \text{ only}} / AMAT_{L1, L2} =$
 - $AMSPI_{L1 \text{ only}} / AMSPI_{L1, L2} =$
 - $CPI_{L1 \text{ only}} / CPI_{L1, L2} =$



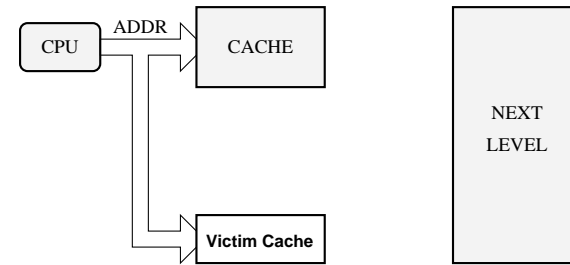
Multilevel Caches



AMAT =
 Miss Penalty (L1) =
 Miss Penalty (L2) =

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.”



Reduce miss penalty and miss rate.



Recommended exercises

-

