CSE 351 – Section 7: Caching & Processes

Aaron Miller
David Cohen
Spring 2011
The Memory Mountain

Pentium III Xeon
550 MHz
16 KB on-chip L1 d-cache
16 KB on-chip L1 i-cache
512 KB off-chip unified L2 cache

Read throughput (MB/s)

Stride (words)

Working set size (bytes)
Example: Array Copy (HW0)

```c
int src[2048][2048];
int dst[2048][2048];

/* Row-major */
int i, j;
for(i = 0; i < 2048; i++) {
    for(j = 0; j < 2048; j++) {
        dst[i][j] = src[i][j];
    }
}

/* Column-major */
for(j = 0; j < 2048; j++) {
    for(i = 0; i < 2048; i++) {
        dst[i][j] = src[i][j];
    }
}
```

L1 Cache:
32 KB
2-way set associative
16 B blocks

1. What are the hit and miss rates for the two different loops?
2. Assuming a miss penalty of 4 cycles, what is the Avg. Memory Access Time (AMAT) for the different loops?
Optimizations for the Memory Hierarchy

- Write code that has locality
  - Spatial: access data contiguously
  - Temporal: make sure access to the same data is not too far apart in time

- How to achieve?
  - Proper choice of algorithm
  - Loop transformations

- Cache versus register-level optimization:
  - In both cases locality desirable
  - Register space much smaller
    + requires scalar replacement to exploit temporal locality
  - Register level optimizations include exhibiting instruction level parallelism (conflicts with locality)
Example: Matrix Multiplication

c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            for (k = 0; k < n; k++)
                c[i*n + j] += a[i*n + k]*b[k*n + j];
}"
Cache Miss Analysis

Assume:
- Matrix elements are doubles
- Cache block = 8 doubles
- Cache size C << n (much smaller than n)

First iteration:
- \( \frac{n}{8} + n = \frac{9n}{8} \) misses
  (omitting matrix c)

Afterwards in cache:
  (schematic)
Cache Miss Analysis

- **Assume:**
  - Matrix elements are doubles
  - Cache block = 8 doubles
  - Cache size \( C \ll n \) (much smaller than \( n \))

- **Other iterations:**
  - Again:
    \[
    \frac{n}{8} + n = \frac{9n}{8} \text{ misses}
    \]
    (omitting matrix \( c \))

- **Total misses:**
  - \[
  9n/8 \times n^2 = (9/8) \times n^3
  \]
Blocked Matrix Multiplication

```c
double *c = calloc(sizeof(double), n*n);

void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
                /* B x B mini matrix multiplications */
                for (i1 = i; i1 < i+B; i++)
                    for (j1 = j; j1 < j+B; j++)
                        for (k1 = k; k1 < k+B; k++)
                            c[i1*n + j1] += a[i1*n + k1]*b[k1*n + j1];
}
```

Block size B x B

![Diagram](image)
Cache Miss Analysis

■ Assume:
  ▪ Cache block = 8 doubles
  ▪ Cache size $C \ll n$ (much smaller than $n$)
  ▪ Four blocks fit into cache: $4B^2 < C$

■ First (block) iteration:
  ▪ $B^2/8$ misses for each block
  ▪ $2n/B \times B^2/8 = nB/4$
    (omitting matrix $c$)
  ▪ Afterwards in cache (schematic)
Cache Miss Analysis

- **Assume:**
  - Cache block = 8 doubles
  - Cache size $C \ll n$ (much smaller than $n$)
  - Three blocks fit into cache: $3B^2 < C$

- **Other (block) iterations:**
  - Same as first iteration
  - $2n/B \times B^2/8 = nB/4$

- **Total misses:**
  - $nB/4 \times (n/B)^2 = n^3/(4B)$
Summary

- **No blocking:** \((9/8) \times n^3\)
- **Blocking:** \(1/(4B) \times n^3\)

- If \(B = 8\) difference is \(4 \times 8 \times 9 / 8 = 36x\)
- If \(B = 16\) difference is \(4 \times 16 \times 9 / 8 = 72x\)

- **Suggests largest possible block size \(B\), but limit \(4B^2 < C\)!**
  
  (can possibly be relaxed a bit, but there is a limit for \(B\))

- **Reason for dramatic difference:**
  - Matrix multiplication has inherent temporal locality:
    - Input data: \(3n^2\), computation \(2n^3\)
    - Every array elements used \(O(n)\) times!
  - But program has to be written properly