HOWTO WRITE FAST NUMERICAL CODE
EXERCISE 3

Pascal Spörri
pascal@spoerri.io

March 21, 2013

1 Cache Mechanics

Running the code yields in the following memory acceses:

Iteration	**Memory Location**	
i	$x[2*i\%5]$	$y[2*i\%5]$
0	$x[0]$	$y[0]$
1	$x[2]$	$y[2]$
2	$x[4]$	$y[4]$
3	$x[1]$	$y[1]$
4	$x[3]$	$y[3]$
5	$x[0]$	$y[0]$
6	$x[2]$	$y[2]$
7	$x[4]$	$y[4]$
8	$x[1]$	$y[1]$
9	$x[3]$	$y[3]$

Figure 1: Memory locations accessed per iteration

The cache for each loop iteratio is represented here:

Cache Line	**Iteration**										
16 Byte Blocks	**Init**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	-	x[0]x[1]	x[0]x[1]	y[4]y[5]	x[0]x[1]	x[0]x[1]	x[0]x[1]	x[0]x[1]	y[4]y[5]	x[0]x[1]	x[0]x[1]
1	-	-	x[2]x[3]	x[2]x[3]	x[2]x[3]	x[2]x[3]	x[2]x[3]	x[2]x[3]	x[2]x[3]	x[2]x[3]	x[2]x[3]
2	-	y[0]y[1]	y[0]y[1]	x[4]x[5]	y[0]y[1]	y[0]y[1]	y[0]y[1]	y[0]y[1]	x[4]x[5]	y[0]y[1]	y[0]y[1]
3	-	-	y[2]y[3]	y[2]y[3]	y[2]y[3]	x[2]x[3]	y[2]x[3]	y[2]y[3]	y[2]y[3]	y[2]y[3]	y[2]y[3]

Figure 2: Cache content for each iteration step

1.1 Hit/Miss Sequences

Based on the cache analysis we are able to determine the miss/hit sequences:

x: MMMMHHHMMH

y: MMMMHHHMMH

1.2 Miss rate for the individual arrays

Both arrays have the same miss rate:

x: $\frac{6}{10}$

y: $\frac{6}{10}$

1.3 Operational Intensity

Flops: For each iteration we observe 2 floating point operations. Which makes 10 floating point operations for the total run of the program.

Bytes Transferred: We observe 6 cache misses per array per execution. This creates a total of $2 \cdot 6 \cdot 16 = 192$ bytes transferred into the cache.

Thus we can conclude that the total operational intensity is:

$$I = \frac{\text{Flops}}{\text{Bytes Transferred}} = \frac{10}{192} \approx 0.05$$

2 Cache Mechanics

For this exercise we evaluate the following code:

```
double x[128], sum;
int i,j;
for (int i=0; i<64; i++) {
    j = i+64;
    sum += x[i]*x[j];
}
```

2.1 Case 1

Cache size: 512 **bytes:** Since the cache has a size of 512 bytes and a set size of 16 bytes we observe that there are 32 sets. Both `x[i]` and `x[j]` will access the set $i\%2$ and evict themselves on each loop iteration. We can therefore conclude that the miss rate is 100%.

Cache size: 1024 **bytes:** `x[i]` will access the set $i\%2$ and `x[j]` will access the set $i\%2 + 32$ on each iteration step. We can therefore conclude that the miss rate is 50%.

2.2 Case 2

Cache size: 512 bytes with 2 way set associativity As we have seen before both `x[i]` and `x[j]` will access the set $i\%2$. Since we now have 2 way set associativity the cache lines won't evict each other. We can therefore conclude that the miss rate is 50%.

Increasing the cache Increasing the cache won't help in this case since the limiting factor is the size of the set: Every two cycles a new set needs to be loaded which will create a miss.

Increasing the cache line Increasing the cache line and keeping the set associativity will definitely help since the memory accesses won't compete for a cache line. Doubling the cache line size to 32 bytes will yield a miss rate of 25%.

3 MMM Analysis

For this exercise we consider the following code:

```
int i, j, k;
for (i = 0; i < n; i++) {
    for (j + 0; j < n; j++) {
        for (k = 0; k < n; k++) {
            C[i][j] = C[i][j] + A[i][j]*B[k][j];
        }
    }
}
```

3.1 Smallest Cache Size needed for MMM

We first proceed to describe the minimal space requirements for each matrix:

A Since we need to access each row of matrix A multiple times, we want to hold the row for the outermost row in the cache. The space requirements for the matrix A are therefore: $n \cdot 8$ bytes.

B We need to access each column of matrix B for each innermost loop iteration. To avoid misses we therefore want to keep the entire matrix B in the cache. The space requirements for matrix B are therefore: $n \cdot n \cdot 8$ bytes.

C For the matrix C it is sufficient to keep one block in the cache: The accesses are coalesced and the values are reused in each inner loop iteration.

For a fixed n we would therefore need a minimal cache size of $(n + n \cdot n) \cdot 8 + B$ bytes.

3.2 Largest n for 8 MB cache

In the 8 MB cache we can store store 1048576 doubles (cache line: 8 doubles). The largest n that we can use to avoid compulsory misses is 1023:

- We need to use at least 1032 doubles to store each row of matrix A (1024 for the values in the current line + padding since we can't selectively load data and for some cases we need an entire additional block).
- We need to use one block of 8 doubles to store the the doubles for matrix C.
- Matrix B needs 1046529 doubles to store the entire matrix. With block padding this results in 1046536 doubles stored for matrix B.

$1032 + 1046529 + 8 = 1047576$ doubles total space needed and thus making a bigger array size impossible (there are only 8000 bytes to spare).

3.3 Operational Intensity

Since every element is only loaded once we can concluded that the total bytes requested by the program are: $3 \cdot n^2 \cdot 8 = 3 \cdot 1024^2 \cdot 8 = 25116696$ bytes. Since only the matrix C is written back to memory the program needs to store $8 \cdot n^2 = 8 \cdot 31^2 = 8372232$ bytes. Thus making the total memory traffic: $4 \cdot 8372232 = 33488928$ bytes or 31 MBytes.

A run of the program yields $2 \cdot n^3$ floating point operations. For the concrete example above this results in a total of 1070599167 floating point operations.

From the total memory traffic and the floating point operations we are therefore able to compute the operational intensity for the concrete example:

$$I_{32} = \frac{1070599167}{8372232} = 127 \left[\frac{\text{Flops}}{\text{Bytes}}\right]$$

4 Proofreading

The computation of the total memory traffic is wrong which results in a wrong computation of the operational intensity. In the exercise we compute the number of misses which corresponds to the number of cache blocks are loaded. For the whole matrix matrix multiplication we load $\frac{1}{8}(n^3 + n^2)$ blocks. Which corresponds to a total memory traffic of $\frac{1}{8}(n^3 + n^2) \cdot 64 = 8(n^3 + n^2)$ bytes. We can therefore see that to total operational intensity is:

$$I_n \leq \frac{2n^3}{8(n^3 + n^2)} \approx \frac{1}{4} \left[\frac{\text{Flops}}{\text{Bytes}}\right]$$