241110031-assign1.pdf

Jatin Jangir

August 2024

1 Problem 1

[20 marks]

Consider the following loop. Assume an 8-way set-associative cache with a capacity of 256 KB, line size of 64 B, and word size of 4 B (for float). The cache is empty before execution and uses an LRU replacement policy. Given size=32K, determine the total number of cache misses on A for the following access strides: 1, 4, 16, 64, 2K, 8K, 16K, and 32K. Consider all the three kinds of misses: cold, capacity, and conflict.

```
float s = 0.0, A[size];
int i, it, stride;
for (it = 0; it < 1000 * stride; it++) {
    for (i = 0; i < size; i += stride) {
        s += A[i];
    }
}
```

Answer

Cache Configuration:

- Cache Capacity: **256 KB** = 262,144 bytes
- Line Size: **64 bytes**
- Word Size: **4 bytes (for float)**
- Associativity: **8**-way set-associative

Derived Parameters:

- Number of cache lines:

$$\frac{2^{18}\ bytes}{2^{6} bytes}$$

$$2^{12}\ cache\ lines.$$

- Number of cache sets: Since the cache is 8-way set-associative, the number of sets is

$$\frac{[2^{12}\ \textit{cache lines.}]}{[2^{3}\ \textit{line per set}]}$$

$$= 2^{9}\ \textit{cache lines.}$$

Array Configuration:

- Array Size (size): 32K*4B =

$$2^{17}\ B$$

The array size (2^{17} bytes) is smaller than the cache capacity (2^{18} bytes), so **capacity misses will not occur**. We need to focus on cold misses and conflict misses.

Stride Analysis:

Stride = 1:

- Every consecutive element is accessed, leading to spatial locality. Each cache line holds 64bytes and 4bytes/element

$$2^{4}$$

- Cold Misses: The first access to each cache line will result in a miss.
- Since the array size is 2^{17} bytes, the number of cache blocks needed is = 2^{11} blocks
- Conflict Misses: Since the total number of cache blocks 2^{11} lines is less than the cache capacity (2^{12} blocks), **no conflict misses occur.**
- Total Misses: **2,048 misses.**

Stride = 4:

- Every 4th element is accessed, resulting in skipping 3 elements in between.
- Each cache line still holds 16 elements, so the first access will load 4 elements (1st, 2nd, 3rd, 4th elements of that cache blocks).
- Cold Misses: Number of cache blocks needed remains 2^{11}, so **2,048 cold misses**.
- **Conflict Misses: No conflict misses.**
- Total Misses: **2,048 misses.**

Stride = 16:

- Every 16th element is accessed, so each access fetches the first element of a new cache line.
- Cold Misses: Number of cache blocks needed = 2,048.
- Conflict Misses: No conflict misses.
- Total Misses: **2,048 misses.**

Stride = 64:

- Every 64th element is accessed, so each access fetches an entirely new cache blocks.
- Cold Misses: Number of cache blocks needed = $2^{15}/2^{6}$.
- Conflict Misses: No conflict misses.
- Total Misses: 2^9 **misses.**

Stride = 2K:

- Every 2,048th element is accessed, which corresponds to accessing every 128th cache line.
- Cold Misses: Number of cache blocks needed = $2^{15}/2^{11}$.
- Conflict Misses: No conflict misses.
- Total Misses: 2^4 **misses.**

Stride = 8K:

- Every 8,192nd element is accessed, which corresponds to accessing every 512th cache line.
- Cold Misses: Number of cache blocks needed = $2^{15}/2^{13}$.
- Conflict Misses: No conflict misses.
- Total Misses: 2^2 **misses.**

Stride = 16K:

- Every 16,384th element is accessed, which corresponds to accessing every 1024th cache line.
- Cold Misses: Number of cache blocks needed = $2^{15}/2^{14}$.
- Conflict Misses: No conflict misses.
- Total Misses: 2^1 **misses.**

Stride = 32K:

- Only one element is accessed every time. so 1 cold misses.
- Conflict Misses: No conflict misses.
- Total Misses: **1 misses.**

2 Problem 2

[40 marks]

Consider a cache of size 64K words and lines of size 8 words. The matrix dimensions are 1024×1024. Perform cache miss analysis for the kij and the jik forms of matrix multiplication (shown below) considering direct-mapped and fully associative caches. The arrays are stored in row-major order. To simplify the analysis, ignore misses from cross-interference between elements of different arrays (i.e., perform the analysis for each array, ignoring accesses to the other arrays).

Answer

Cache Configuration:

- Cache Capacity: **16K word** = 2^{14} word
- Number of Words per Cache Line: = 8 words per cache line.
- Number of Cache Line: = 2^{11} words per cache line.

Array Configuration:

Array A[N][N]:

- Size: N=2^{10} x 2^{10}
- Memory required: 2^{20} word.

Array B[N][N]:

- Size: N=2^{10} x 2^{10}
- Memory required: 2^{20} word.

Array C[N][N]:

- Size: N=2^{10} x 2^{10}
- Memory required: 2^{20} word.

Fully Associative:

Listing 1: kij form:

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

Memory Access Pattern and Cache Miss Analysis:

Array A[N][N]:

- Each access is sequential at column level, accessing A[i][k] one element at a time in 'i' loop.
- in j loop no item is changed thus it will have no effect.
- **Misses:** There will be **1(j) x N(i) x N(k)** misses. Here N=2^{10}. Thus **Misses:** 2^{20}

Array B[N][N]:

- Each access is sequential at row level, accessing B[k][j] one element at a time in 'j' loop, Thus it can use blocks.
- for i loop there will be all elements present in cache from first loop.
- **Misses:** There will be **N/Bl(j) x 1(i) x N(k)** misses. Here N=2^{10} and Bl=8. Thus **Misses:** 2^{17}

Array C[N][N]:

- Each access is sequential at row level, accessing C[i][j] one element at a time in 'j' loop.
- for i loop the elements are in next row thus need to access them diffrently.
- **Misses:** There will be **N/Bl(j) x N(i) x N(k)** misses. Here N=2^{10} and Bl=8. Thus **Misses:** 2^{27}

Table 1: Question 2: Cache Miss Estimation for (fully associative) A,B,C in kij

	A	**B**	**C**
k	N	N	N
i	N	1	N
j	1	N/Bl	N/Bl
Total	N^2	N^2/Bl	N^3/Bl

Table 2: Question 2: Cache Miss for (fully associative) A,B,C in kij

	A	**B**	**C**
k	2^{10}	2^{10}	2^{10}
i	2^{10}	1	2^{10}
j	1	$2^{10}/2^3$	$2^{10}/2^3$
Total	2^{20}	$2^{20}/2^3$	$2^{30}/2^3$

Listing 2: jik form:

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

Array A[N][N]:

- Each access is sequential at row level, accessing A[i][k] one element at a time in 'k' loop.
- **Cold misses:** There will be **N/Bl(k) x N(i) x N(j)** cold misses. Here N=2^{10} and Bl=8. Thus **Cold misses:** 2^{27} **misses**

Array B[N][N]:

- Each access is sequential at column level, accessing B[k][j] one element at a time in 'k' loop.
- for i loop there will be all elements present in cache from first loop.
- for j loop we can use blocks from cache.
- **Misses:** There will be **N(k) x 1(i) x N/Bl(j)** misses. Here N=2^{10} and Bl=8. Thus **Misses:** 2^{17} **misses**

Array C[N][N]:

- Each access is sequential at column level, accessing C[i][j] one element at a time in 'i' loop.
- **Misses:** There will be **1(k) x N(i) x N/Bl(j)** misses. Here N=2^{10} and Bl=8. Thus **Misses:** 2^{17}

Table 3: Question 2: Cache Miss Estimation(fully associative) for A,B,C in jik

	A	**B**	**C**
j	N	N/Bl	N/Bl
i	N	1	N
k	N/Bl	N	1
Total	N^3/Bl	N^2/Bl	N^2/Bl

Table 4: Question 2: Cache Miss (fully associative) for A,B,C in jik

	A	**B**	**C**
j	2^{10}	$2^{10}/2^3$	$2^{10}/2^3$
i	2^{10}	1	2^{10}
k	$2^{10}/2^3$	2^{10}	1
Total	$2^{30}/2^3$	$2^{20}/2^3$	$2^{20}/2^3$

Direct:

Listing 1: kij form:

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

Memory Access Pattern and Cache Miss Analysis:

In case of direct mapping the cache block will get change in 2^6th iteration of i.

Array A[N][N]:

There will be **1(j) x N(i) x N(k)** misses. Here N=2^{10}. Thus **Misses:** 2^{20}

Array B[N][N]:

There will be **N/Bl(j) x N(i) x N(k)** misses.
Here N=2^{10}. Thus **Misses:** 2^{27}

Array C[N][N]:

There will be **N/Bl(j) x N(i) x N(k)** misses.
Here N=2^{10}. Thus **Misses:** 2^{27}

Table 5: Question 2: Cache Miss Estimation for (direct) A,B,C in kij

	A	**B**	**C**
k	N	N	N
i	N	N	N
j	1	N/Bl	N/Bl
Total	N^2	N^3/Bl	N^3/Bl

Table 6: Question 2: Cache Miss for (direct) A,B,C in kij

	A	**B**	**C**
k	2^{10}	2^{10}	2^{10}
i	2^{10}	2^{10}	2^{10}
j	1	$2^{10}/2^3$	$2^{10}/2^3$
Total	2^{20}	$2^{30}/2^3$	$2^{30}/2^3$

Listing 2: jik form:

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

Memory Access Pattern and Cache Miss Analysis:

In case of direct mapping the cache block will get change in 2^6th iteration of loop in column major order.

Array A[N][N]:

There will be **N(k) x N(i) x N/Bl(j)** misses. Here N=2^{10}. Thus **Misses:** 2^{27}

Array B[N][N]:

There will be **N(j) x N(i) x N(k)** misses.
Here N=2^{10}. Thus **Misses:** 2^{30}

Array C[N][N]:

There will be **1(j) x N(i) x N(k)** misses.
Here N=2^{10}. Thus **Misses:** 2^{27}

Table 7: Question 2: Cache Miss Estimation for (direct) A,B,C in jik

	A	**B**	**C**
j	N	N	N
i	N	N	N
k	N/Bl	N	1
Total	N^3/Bl	N^3	N^2

Table 8: Question 2: Cache Miss for (direct) A,B,C in jik

	A	**B**	**C**
j	2^{10}	2^{10}	2^{10}
i	2^{10}	2^{10}	2^{10}
k	$2^{10}/2^3$	2^{10}	1
Total	$2^{30}/2^3$	2^{30}	2^{20}

3 Problem 3

[30 marks]

Consider the following code

```
#define N (4096)
double y[N], X[N][N], A[N][N];
for (k = 0; k < N; k++)
    for (j = 0; j < N; j++)
        for (i = 0; i < N; i++)
            y[i] = y[i] + A[i][j] * X[k][j];
```

Assume a direct-mapped cache of capacity 16 MB, with 32 B cache lines and a word of 8 B. Assume that there is negligible interference between the arrays A, X, and y (i.e., each array has its 16 MB cache for this question), and arrays are laid out in the row-major form. Estimate the total number of cache misses for A, X, and y.

Answer

Cache Configuration:

- Cache Capacity: **16MB** = 2^{24} **bytes**
- Cache Line Size: 32 bytes
- Word Size: 8 bytes

- Number of Words per Cache Line: =4 words per cache line.

Array Configuration:

Array y[N]:

- Size: N=2^{12}
- Memory required: 2^{12} x 8byte = 32 KB.

Array A[N][N]:

- Size: N=2^{12} x 2^{12}
- Memory required: 2^{12} x 2^{12} x 8byte = 128MB.

Array X[N][N]:

- Size: N=2^{12} x 2^{12}
- Memory required: 2^{12} x 2^{12} x 8byte = 128MB.

Memory Access Pattern and Cache Miss Analysis:

Array y[N]:

- Each access is sequential, accessing y[i] one element at a time.
- Memory required: 32 KB is less than cache size 16MB for **No capacity and conflict misses**.
- **Cold misses:** There will be **N/Bl** cold misses. Here N=4096=2^{12} and Bl=4. Thus **Cold misses:** 2^{10} **= 1024 misses**

Array A[N][N]:

- Each access is sequential at column level, accessing A[i][j] one element at a time in 'i' loop.
- Memory required: 128MB is more than cache size 16MB.
- **Cold misses:** There will be **N(i) x N/Bl(j) x N(k)** cold misses. Here N=4096=2^{12} and Bl=4. Thus **Cold misses:** 2^{34} **misses**

Array X[N][N]:

- Each access is sequential at column level, accessing X[k][j] one element at a time in 'j' loop.
- Memory required: 128MB is more than cache size 16MB.
- **Cold misses:** There will be **1(i) x N/Bl(j) x N(k)** cold misses. Here N=4096=2^{12} and Bl=4. Thus **Cold misses:** 2^{22} **misses**

Table 9: Question 3: Cache Miss Estimation for Y,A,X

	Y	**A**	**X**
k	1	N	N
j	1	N	N
i	N/Bl	N	1
Total	N/Bl	N^3	N^2

Table 10: Question 3: Cache Miss for Y,A,X

	Y	**A**	**X**
k	1	4096	4096
j	1	4096	4096
i	1024	4096	1
Total	1024	2^{36}	2^{24}

4 Problem 4

[40 marks]

Assume a naïve O(n3) sequential matrix multiplication kernel (ijk form) for matrices of dimensions 4096×4096 (given).

1. Implement an optimized version of the sequential matrix multiplication implementation using loop blocking. Experiment with different block dimensions (for e.g., 4×4 to 32×32). Use a table to report the speedup of your optimized implementation over the sequential implementation for the different block sizes that you have tried. The first three columns will be the block sizes for matrices A, B, and C, and the last column will report the speedup compared to the sequential version.

2. Note the block size that works best for your setup. Remember that the cache hierarchy will influence the optimal block dimensions. Justify the improved performance by tracking performance counters with PAPI. List the PAPI version and the performance counters you have used.

- Do not change the ijk loop order.
- You may experiment with different block sizes for the three matrices.
- Create separate functions for your optimized variants for ease of debugging and evaluation.
- You should time your code on a noiseless system (i.e., no concurrent applications are running). Take times for 3–5 runs, and use the arithmetic mean.

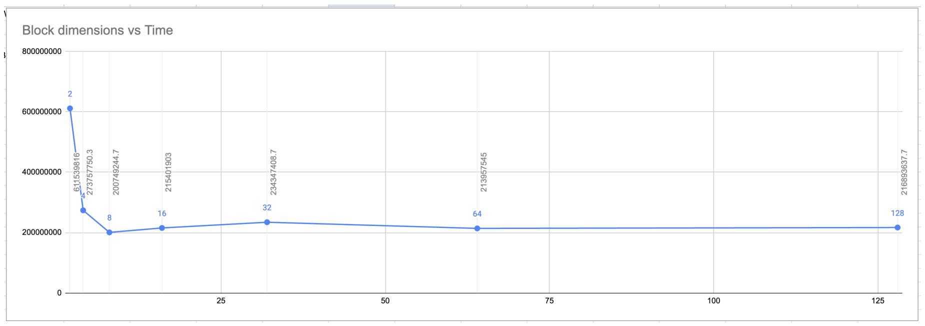

Figure 1: Question 4: Block Dimension vs Time required

- Report the cache hierarchy for the system you will use.
- Ensure that you are using relatively recent versions of PAPI (v5.7+).
- We will try to reproduce your results.

Answer

Table 11: Question 4: Block Dimension vs Time required

Block Size for A,B,C	**Time1**	**Time2**	**Time3**	**Average**	**speedup**
without blocking	502722273	517685376	508513600	**509640416**	1
1	1413284127	1442176949	1427888235	1427783104	0.35
2	621028286	619965983	593625179	611539816	0.83
4	276370321	278384272	266518658	273757750	1.87
8	203462796	199385828	199399110	**200749244**	2.53
16	215234145	211759121	219212443	215401903	2.36
32	232441908	237221667	233378651	234347408	2.16
64	214156181	213490879	214225575	213957545	2.37
128	208634630	232516093	209530190	216893637	2.34
256	207048324	207359840	207156271	207188145	2.45
512	243620114	250194967	247892735	247235938.7	2.06

Cache Configuration:

Apple M2

1. **Level 1 (L1) Cache:**
   - High-performance cores (Avalanche) have larger L1 caches (192 KB instruction, 128 KB data), while high-efficiency cores (Blizzard) have smaller L1 caches (128 KB instruction, 64 KB data). This allows for efficient handling of diverse workloads, as larger caches cater to compute-intensive tasks and smaller caches optimize for power efficiency.

2. **Level 2 (L2) Cache:**
   - Shared L2 cache: Both high-performance and high-efficiency cores share a 16 MB L2 cache, enabling data sharing and reducing memory access latency. This shared cache helps to improve performance for workloads that exhibit data locality.

Most Optimal block size in M2 chip (apple Air) is 8 or above.

4.1 PAPI

Without Blocking

Naive execution time : 11290217 us
Naive execution time : 11349253 us
Naive execution time : 11245094 us
TOT$_INS$: 103122374103
L1$_TCM$: 11034456063
L2$_TCM$: 223265543659
L3$_TCM$: 5528263302

With Blocking

TOT$_INS$: 179209872604
L1$_TCM$: 18824236440
L2$_TCM$: 5773274008
L3$_TCM$: 889930758
The speedup is 1.968 for matrix size: 2048 and block size 8

Justification Number of cache miss is more in L1 for block size 8. Number of miss for L1 and L2 is less for blocking technique so this leds to massive improvement in blocking. This leds to more speedup.