Name: Chitwan Goel
Roll No. 210295

CS610 Assignment 1

1 Problem 1

```
    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];
        }
    }
```

Clearly, the size of the cache is more than the size of the array A. Also, one cache block can hold 16 floats. In 8-way associative cache as given in the problem, 1 set will hold 2^3 lines, thus $2^3 \times 64B$ which is $2^9 B$.
So, for $stride \leq 16$, we will have $size/stride$ accesses to array and out of those, every $16/stride$ will be a miss. So the total misses will be equal to $size/16 = 32K/16 = 2K$.
For $stride > 16$, there will be $size/stride$ accesses to the array and each one of them will be a miss because the access doesn't have spatial locality with previously fetched data. So, total misses will be equal to $size/stride$.
It is also clear that there will be no effect of the outer for loop on the number of misses, and also capacity and conflict misses are 0 for the given use-case.
The following table summarises the cache misses for all stride sizes:

stride	*cold misses*	*capacity misses*	*conflict misses*	*total cache misses*
1	$\frac{32K}{16} = 2K$	0	0	$2K$
4	$\frac{32K}{16} = 2K$	0	0	$2K$
16	$\frac{32K}{16} = 2K$	0	0	$2K$
64	$\frac{32K}{64} = 512$	0	0	512
$2K$	$\frac{32K}{2K} = 16$	0	0	16
$8K$	$\frac{32K}{8K} = 4$	0	0	4
$16K$	$\frac{32K}{16K} = 2$	0	0	2
$32K$	$\frac{32K}{32K} = 1$	0	0	1

2 Problem 2

```
    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];
```

Let BL be the cache block size, i.e., the number of words (doubles in this case) one cache block can hold. Here, $BL = 8$. Also let the matrices be of dimensions $N \times N$, here $N = 1024$. Therefore $\frac{N}{BL} = \frac{1024}{8} = 128$. The size of the cache is smaller than the size of the matrices. The line size is of 8 words.

Direct Mapped Cache

⋆ For C, i and j are the row major accesses. So every BL^{th} iteration of the j loop will be a miss and this pattern will repeat for every i. So the contribution of i and j will be N and N/BL. Again, since the matrix is bigger than the cache size, the k loop will also contribute with N.
⋆ For A, j will contribute with 1 because we are accessing the same element in every iteration of j loop. So, when $A[i][k]$ is accessed, this is the column major order access and thus k and i will both contribute N. This is because the cache lines are 2^{13} and for one row of A, we are requiring 2^7 cache lines. So, let's say $A[0][0] - A[0][7]$ is in cache line 0, when we access $A[64][0]$, it evicts the $A[0][0] - A[0][3]$ from the first line and goes there. Thus for every row and every column, there is a cache miss.
⋆ For B, for a fixed k, we are accessing the row $B[k]$ in the j loop. So, every BL^{th} iteration will be a miss and $j's$ contribution is therefore N/BL. Now, for every i, we are accessing the same row, which is already in cache. Thus, $i's$ contribution is 1. Clearly, the pattern will repeat for every k and thus its contribution is N.

Fully Associative Cache

⋆ B and C will not be affected by the fully associative cache because their accesses are row-major.
⋆ For A, there will be effect on the k loop, because due to the full-associativity, when we access $A[64][0]$, $A[0][0] - A[0][7]$ needs not be evicted. Thus, in every BL^{th} iteration of k, there will be N misses due to i loop. Hence $k's$ contribution changes to N/BL.

	A	B	C
i	$N = 1024$	1	$N = 1024$
j	1	$\frac{N}{BL} = 128$	$\frac{N}{BL} = 128$
k	$N = 1024$	$N = 1024$	$N = 1024$
	$N^2 = 2^{20}$	$\frac{N^2}{BL} = 2^{17}$	$\frac{N^3}{BL} = 2^{27}$

(a) Direct Mapped Cache

	A	B	C
i	$N = 1024$	1	$N = 1024$
j	1	$\frac{N}{BL} = 128$	$\frac{N}{BL} = 128$
k	$\frac{N}{BL} = 128$	$N = 1024$	$N = 1024$
	$\frac{N^2}{BL} = 2^{17}$	$\frac{N^2}{BL} = 2^{17}$	$\frac{N^3}{BL} = 2^{27}$

(b) Fully Associative Cache

Figure 1: kij form

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

Direct Mapped Cache

⋆ For A, i and k are the row major accesses. So every BL^{th} iteration of the k loop will be a miss and this pattern will repeat for every i. So the contribution of i and k will be N and N/BL. Again, since the matrix is bigger than the cache size, the j loop will also contribute with N.
⋆ For C, k will contribute with 1 because we are accessing the same element in every iteration of k loop. So, when $C[i][j]$ is accessed, this is the column major order access and thus j and i will both contribute N.

This is because the cache lines are 2^{13} and for one row of A, we are requiring 2^7 cache lines. So, let's say $C[0][0] - C[0][7]$ is in cache line 0, when we access $C[64][0]$, it evicts the $C[0][0] - C[0][7]$ from the first line and goes there. Thus for every row and every column, there is a cache miss.
⋆ For B, the access $B[k][j]$ is like column-major access thus both will contribute with N. Again, suppose for a fixed j, we are accessing the jth column of every row using the k loop. But since the cache is direct mapped the data $B[k][j]$ which was brought into the cache in $(i-1)^{th}$ iteration is now lost. So for every i we will incur the same misses. Thus i will also contribute N.

Fully Associative Cache
⋆ A will not be affected by the fully associative cache because its accesses are row-major.
⋆ For C, there will be effect on the j loop, because due to the full-associativity, when we access $C[64][0]$, $C[0][0] - C[0][7]$ needs not be evicted. Thus, in every BL^{th} iteration of j, there will be N misses due to i loop. Hence $j's$ contribution changes to N/BL.
⋆ For B, $i's$ contribution will change to 1 due to the full-associativity since the block we have brought for every row during the k loop will be present in the block for every i. Also, due to the full-associativity, we will now incur misses for every BL^{th} iteration of the j loop. Thus, $j's$ contribution will also change to N/BL.

	A	B	C
i	$N = 1024$	$N = 1024$	$N = 1024$
j	$N = 1024$	$N = 1024$	$N = 1024$
k	$\frac{N}{BL} = 128$	$N = 1024$	1
	$\frac{N^3}{BL} = 2^{27}$	$N^3 = 2^{30}$	$N^2 = 2^{20}$

(a) Direct Mapped Cache

	A	B	C
i	$N = 1024$	1	$N = 1024$
j	$N = 1024$	$\frac{N}{BL} = 128$	$\frac{N}{BL} = 128$
k	$\frac{N}{BL} = 128$	$N = 1024$	1
	$\frac{N^3}{BL} = 2^{27}$	$\frac{N^2}{BL} = 2^{17}$	$\frac{N^2}{BL} = 2^{17}$

(b) Fully Associative Cache

Figure 2: jik form

3 Problem 3

```
# 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];
```

Let BL be the cache block size, i.e., the number of words (doubles in this case) one cache block can hold. Here, $BL = 32/8 = 4$. Also let the matrices be of dimensions $N \times N$, here $N = 4096$. Therefore $\frac{N}{BL} = \frac{4096}{4} = 1024$.
⋆ The cache size is smaller than the size of the matrices A and X. So, when $A[i][j]$ is accessed, this is the column major order access and thus j and i will both contribute N. This is because the cache lines are 2^{19} and for one row of A, we are requiring 2^{10} cache lines. So, let's say $A[0][0] - A[0][3]$ is in cache line 0, when we access $A[512][0]$, it evicts the $A[0][0] - A[0][3]$ from the first line and goes there. Thus for every row and

every column, there is a cache miss. Again, the k loop also contributes N, because the matrix is bigger than the cache.
$\star$ For X, i contributes 1 because we are accessing the same element in the i loop. For k and j, we are accessing $X[k][j]$ which is row-major access. So, in the j loop, every BL^{th} iteration will be a miss, and the pattern will repeat for every row, i.e., k. So, j and k contribute N/BL and N respectively.
$\star$ For y, the whole vector can be accomodated inside the cache. So the outer 2 loops k and j contribute with a factor of 1 since the vector is in the cache as soon as the i loop is finished once. For the i loop, every BL^{th} iteration will be a miss and thus its contribution is N/BL.
(Since the cache is direct mapped, lines and sets have been used interchangeably in the explanation.)

	A	X	y
i	$N = 4096$	1	$\frac{N}{BL} = 1024$
j	$N = 4096$	$\frac{N}{BL} = 1024$	1
k	$N = 4096$	$N = 4096$	1
	$N^3 = 2^{36}$	$\frac{N^2}{BL} = 2^{22}$	$\frac{N}{BL} = 2^{10}$

4 Problem 4

4.1 Speedups for various combination of Block sizes

BLK_A	BLK_B	BLK_C	Non-blocking Time	Blocking Time	Speedup
2	2	2	85902430	65452500	1.272370
4	4	4	81079720	34373760	2.422773
8	8	8	78905900	26161320	3.183320
16	16	16	82793850	23645920	3.521953
32	32	32	84328060	26468110	3.146422
64	64	64	84804370	32984550	2.524813
128	128	128	85144460	34596830	2.407152
		Average Non-Blocking	**83279826**		

The speedups are calculated by dividing the Average Non-blocking time by the time for that particular block size.

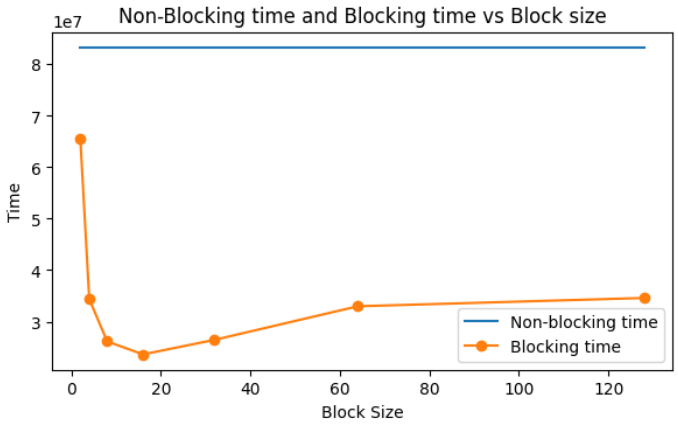

Figure 3: Blocking time for various block sizes

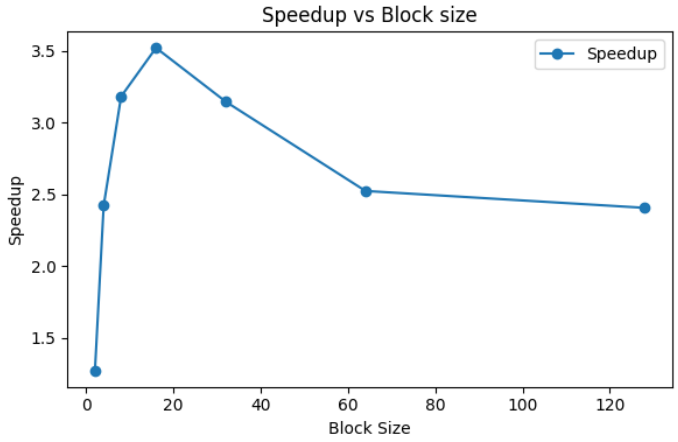

Figure 4: Speedup vs Block Size

4.2 PAPI Counters

For the setup on `csews` server, block size of 16 gives the best speedup. Following is the cache hierarchy for the system:

Parameter	Value
PAPI Version	7.1
L1 Data Cache Total size	32 KB
L1 Data Cache Line size	64 B
L1 Data Cache Number of Lines	512
L1 Data Cache Associativity	8
L2 Unified Cache Total size	256 KB
L2 Unified Cache Line size	64 B
L2 Unified Cache Number of Lines	4096
L2 Unified Cache Associativity	4

`PAPI_L1_DCM` (L1 data cache misses) and `PAPI_L2_DCM` (L2 data cache misses) were the only relevant cache counters available on the systems. Following are the values of performance counters tracked using PAPI:

Block Size	Non-blocking L1 misses	Blocking L1 misses	Non-blocking L2 misses	Blocking L2 misses
2	1.101569×10^{10}	4.245494×10^{9}	2.392295×10^{10}	1.098589×10^{10}
4	1.103240×10^{10}	1.263894×10^{9}	2.434794×10^{10}	1.830289×10^{9}
8	1.098959×10^{10}	3.835413×10^{9}	2.409019×10^{10}	3.703039×10^{9}
16	1.098925×10^{10}	9.335361×10^{9}	2.431837×10^{10}	2.117144×10^{9}
32	1.106300×10^{10}	8.920320×10^{9}	2.421710×10^{10}	3.543645×10^{9}
64	1.107263×10^{10}	8.760813×10^{9}	2.392301×10^{10}	1.051841×10^{10}
128	1.109521×10^{10}	8.680369×10^{9}	2.393708×10^{10}	2.177125×10^{10}

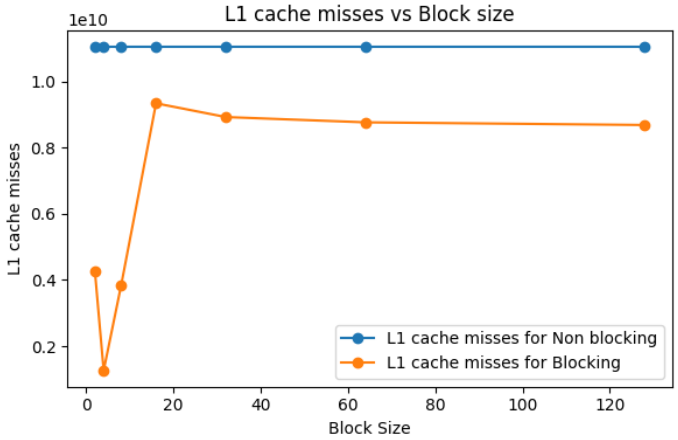

Figure 5: L1 cache misses vs Block Size

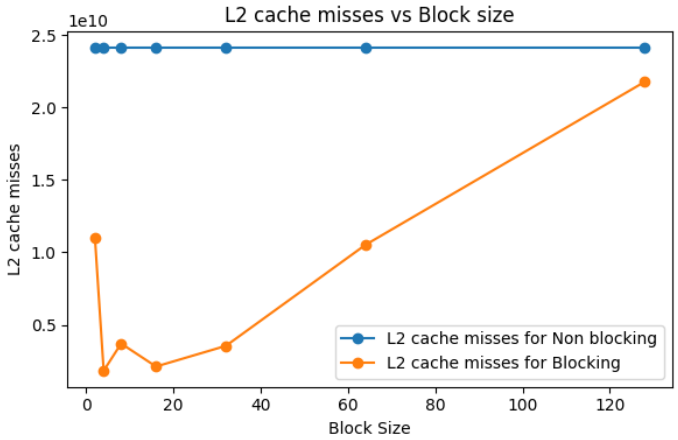

Figure 6: L2 cache misses vs Block Size

The cache misses for L1 as well as L2 cache are reduced for blocking matrix multiplication as compared to the non blocking one, which is in agreement with the reduction in timings for blocking multiplication.