E0 243:HIGH PERFORMANCE COMPUTER ARCHITECTURE

Programming Assignment part 1

Submitted By:

Name: Chetan Pant

SR no : 18071

PART A-1:Optimize Single Threaded DMM

System configuration:

Processor	: Intel Core i5 7th Gen 7200U
Clock Speed	: 2.5 GHz
RAM	: 4GB

Analysis and Bottlenecks in the code :

We analyse the program using the “Perf” tool . We run perf command on all three input sizes (4K,8K,16K).

Perf command used : perf stat -e L1-dcache-load-misses:u ,L1-icache-load-misses:u, dTLB-load-misses:u,branch-misses:u ./diag_mult data /input_4096.in

This is the perf command used for input size 4096.Similiar command is used for remaining two input size.

Input Size	Number of L1 D cache misses
4096	15,04,04,369
8192	80,86,94,809
16384	3,24,90,73,341

By analysing the value of different hardware events specifies in the perf command we found that the number of misses in the L1 D cache is very high and also number of dTLB misses is very high .If we reduce the number of cache and TLB misses ,the runtime of the program will be lower. But in our optimization we only focused on reducing the number of L1 cache misses.

The main Bottleneck in this given program is that the program is not taking full advantage of spatial locality of references. We know that array are store in row major order and the number of cache lines we can store in L1 cache is very limited . In the program , after some amount of accesses the cache becomes full and for new request it will replace the old cache blocks . So,when the program again access this replaced block it will be a miss. To optimize the code we need to utilize the cache block fully when it is accessed first time.

Optimization:

To optimize the program ,We used three techniques

1. **Blocking** : we divide the matrix into blocks of 16*16 .We tried this in different block size but 16*16 is showing good results.We first multiply a block of A to the corresponding block of B .For this ,We are accessing the block of A in row major and column of B in column major .Instead of multiplying the whole matrix at once we are first multiplying the block of both matrix . Cache blocks in the matrix block will remain in cache until our calculation is completed because its size is less than the L1 cache capacity.Therefore this will reduce the number of misses in L1 cache .

2. **Prefetching** :To cover some misses due to accessing the blocks of memory first time (Cold Misses) ,we used prefetching. Prefetching the data early enough will reduce/eliminate the miss latency . For prefetching, we use ***__builtin_prefetch(address,0,1)*** function available in C .

3. **Loop Unrolling** :To further reduce the execution time of the program, we used loop unrolling technique . for unrolling we assumes that the blocks consists of sub blocks of size 2*2 . Therefore we take the corresponding sub-block of A and B , and multiply this .This will take 4 iterations but we unrolls this and do all the computation in one big iteration. Loop unrolling will give more chances to the processor for exploiting instruction level parallelism.

Results:

After using all these optimizing techniques ,our optimized program shows a significant reduction in execution time and also reduction in the number of L1 D cache load misses for all three input sizes.

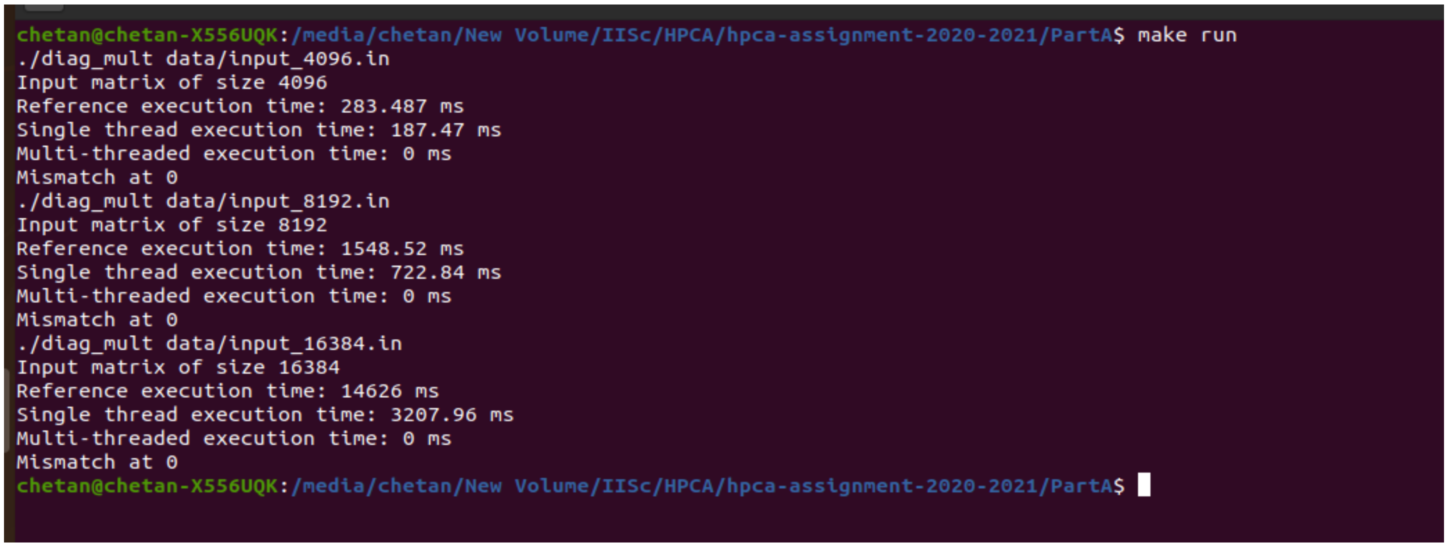
L1 D cache misses after optimization:

Input Size	Number of L1 D cache misses
4196	11,39,18,977
8192	59,73,80,114
16384	2,39,77,08,679

Improvement in Execution time/Speed up:

Input Size	Execution time before Optimization	Execution time after Optimization	Speed Up
4096	283.487	187.47	1.513
8192	1548.52	722.84	2.142
16384	14626	3207.96	4.560

Screenshot of the results:



```
chetan@chetan-X556UQK:/media/chetan/New Volume/IISc/HPCA/hpca-assignment-2020-2021/PartA$ make run
./diag_mult data/input_4096.in
Input matrix of size 4096
Reference execution time: 283.487 ms
Single thread execution time: 187.47 ms
Multi-threaded execution time: 0 ms
Mismatch at 0
./diag_mult data/input_8192.in
Input matrix of size 8192
Reference execution time: 1548.52 ms
Single thread execution time: 722.84 ms
Multi-threaded execution time: 0 ms
Mismatch at 0
./diag_mult data/input_16384.in
Input matrix of size 16384
Reference execution time: 14626 ms
Single thread execution time: 3207.96 ms
Multi-threaded execution time: 0 ms
Mismatch at 0
chetan@chetan-X556UQK:/media/chetan/New Volume/IISc/HPCA/hpca-assignment-2020-2021/PartA$
```

Part A-2 : Optimize Multi Threaded DMM:

Implementation Details:

We applied multithreading on the given unoptimized code. For multithreading , we provide the calculation of some set of output elements on thread basis.Each thread will get work on only a portion of output vector. Every thread portion is disjoint ,it means thread are working on different elements of output vector.

For eg; if we have 15 elements on output vector and 4 thread ,then first thread will work on the elements 0,1,2,3 and 2nd thread will work on 4,5,6,7 and so on..

Since Threads are working on different output elements ,then there will be no synchronization problem among them . Note that all the thread are using same matrix pointers. We will wait for all the thread to complete execution before returning to the function .So for this we used pthread_join() function available in C .

Note:For multithreaded execution we used the Server provided by the instructor.

Results and Scalability of implementation:

We implemented the program on different thread count .The results are shown in the table given below.

Execution time on different thread count.

Input size	Referenced Execution time	M E T (number of thread =4)	M E T (number of thread =8)	M E T (number of thread =16)	M E T (number of thread =32)
4096	191.85	130.67	59.25	48.14	29.08
8192	812.76	463.32	255.99	157.39	102.40
16384	3667.72	2067.24	1167.62	632.25	372.83

* MET = Multithreaded execution time.

Speedup obtained on different thread count

Input size	Speedup (number of thread =4)	Speedup (number of thread =8)	Speedup (number of thread =16)	Speedup (number of thread =32)
4096	1.47	3.23	4.00	6.62
8192	1.75	3.18	5.171	7.96
16384	1.77	3.14	5.80	9.85

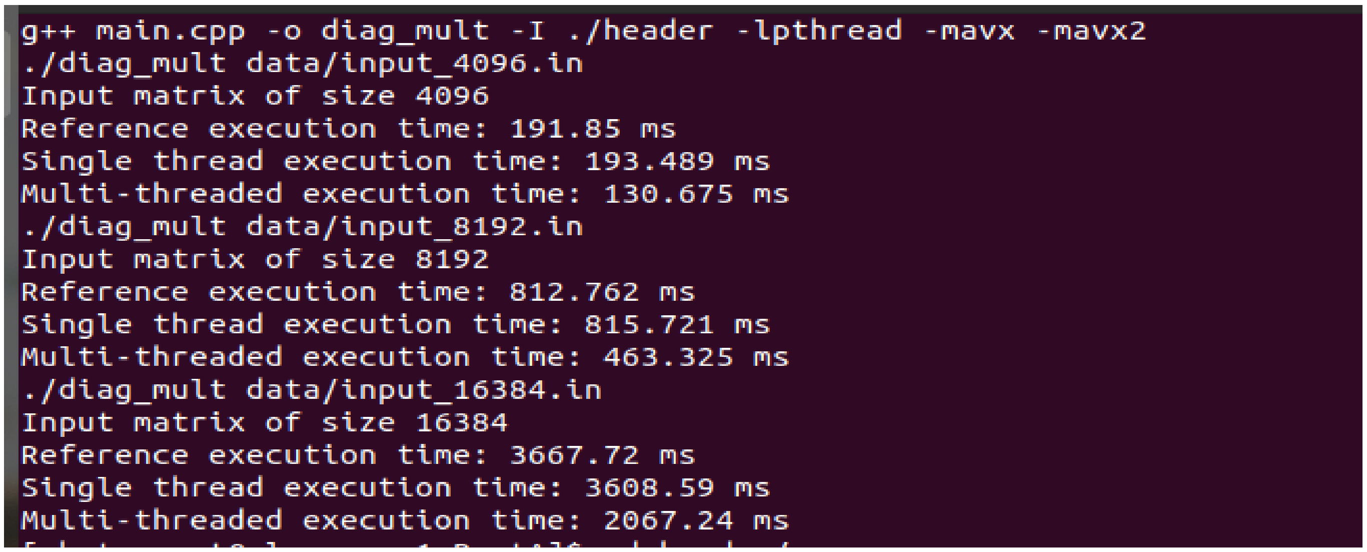
From the above data of execution time ,we can clearly see that on increasing the number of thread by the factor of 2 ,the execution time of the program reduces approximately by a factor of 2.This is due to the more parallelism provide by more number of thread.

Note that this is not always true that on increasing the number of thread program time always decrease.After a certain threshold on number of thread ,program execution time may increase,this is due to more overhead on processor because of large number of threads (eg. context switching between large number of thread will be expensive).

Screenshots of the results:

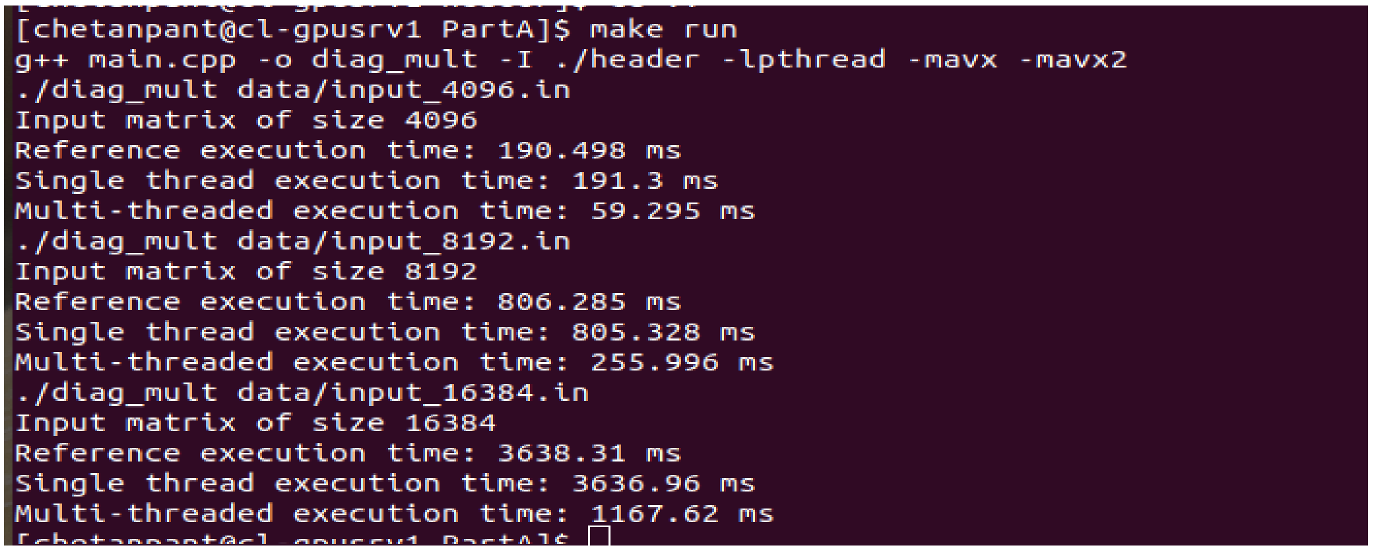
Note that here we are only showing optimization due to multithreaded,so single threaded execution time in the screenshots below is not the optimized single threaded execution time.

- Number of thread=4



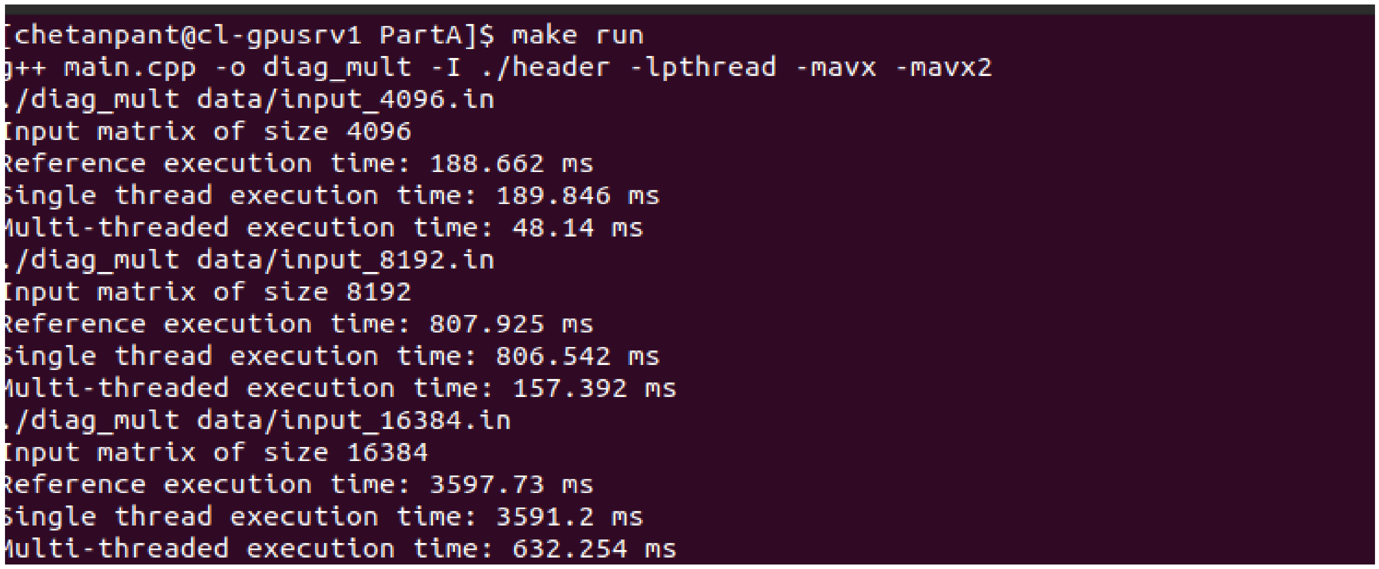
```
g++ main.cpp -o diag_mult -I ./header -lpthread -mavx -mavx2
./diag_mult data/input_4096.in
Input matrix of size 4096
Reference execution time: 191.85 ms
Single thread execution time: 193.489 ms
Multi-threaded execution time: 130.675 ms
./diag_mult data/input_8192.in
Input matrix of size 8192
Reference execution time: 812.762 ms
Single thread execution time: 815.721 ms
Multi-threaded execution time: 463.325 ms
./diag_mult data/input_16384.in
Input matrix of size 16384
Reference execution time: 3667.72 ms
Single thread execution time: 3608.59 ms
Multi-threaded execution time: 2067.24 ms
```

- Number of thread=8



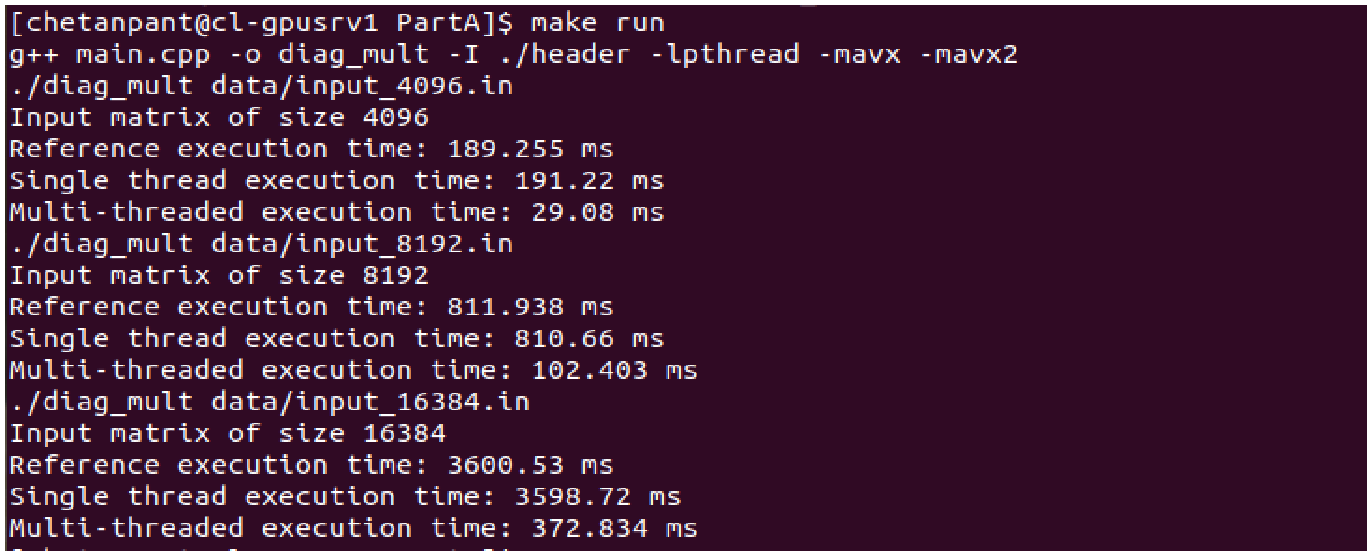
```
[chetanpant@cl-gpusrv1 PartA]$ make run
g++ main.cpp -o diag_mult -I ./header -lpthread -mavx -mavx2
./diag_mult data/input_4096.in
Input matrix of size 4096
Reference execution time: 190.498 ms
Single thread execution time: 191.3 ms
Multi-threaded execution time: 59.295 ms
./diag_mult data/input_8192.in
Input matrix of size 8192
Reference execution time: 806.285 ms
Single thread execution time: 805.328 ms
Multi-threaded execution time: 255.996 ms
./diag_mult data/input_16384.in
Input matrix of size 16384
Reference execution time: 3638.31 ms
Single thread execution time: 3636.96 ms
Multi-threaded execution time: 1167.62 ms
[chetanpant@cl-gpusrv1 PartA]$
```

- Number of thread=16



```
[chetanpant@cl-gpusrv1 PartA]$ make run
g++ main.cpp -o diag_mult -I ./header -lpthread -mavx -mavx2
./diag_mult data/input_4096.in
Input matrix of size 4096
Reference execution time: 188.662 ms
Single thread execution time: 189.846 ms
Multi-threaded execution time: 48.14 ms
./diag_mult data/input_8192.in
Input matrix of size 8192
Reference execution time: 807.925 ms
Single thread execution time: 806.542 ms
Multi-threaded execution time: 157.392 ms
./diag_mult data/input_16384.in
Input matrix of size 16384
Reference execution time: 3597.73 ms
Single thread execution time: 3591.2 ms
Multi-threaded execution time: 632.254 ms
```

- Number of thread=32



```
[chetanpant@cl-gpusrv1 PartA]$ make run
g++ main.cpp -o diag_mult -I ./header -lpthread -mavx -mavx2
./diag_mult data/input_4096.in
Input matrix of size 4096
Reference execution time: 189.255 ms
Single thread execution time: 191.22 ms
Multi-threaded execution time: 29.08 ms
./diag_mult data/input_8192.in
Input matrix of size 8192
Reference execution time: 811.938 ms
Single thread execution time: 810.66 ms
Multi-threaded execution time: 102.403 ms
./diag_mult data/input_16384.in
Input matrix of size 16384
Reference execution time: 3600.53 ms
Single thread execution time: 3598.72 ms
Multi-threaded execution time: 372.834 ms
```

References:

[1]. ”Other Built-in Functions Provided by GCC” ,referenced from https://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html#:~:text=Built%2Din%20Function%3A%20void%20__,likely%20to%20be%20accessed%20soon.

[2]. “Perf Tutorials” ,referenced from https://perf.wiki.kernel.org/index.php/Tutorial

[3]. “MultiThreading in C”,referenced from https://www.geeksforgeeks.org/multithreading-c-2/