HIGH PERFORMANCE COMPUTER ARCHITECTURE

ASSIGNMENT 3

REPORT

Arpit Mishra (SR no.24131): arpitmishra@iisc.acin

Ayush Kumar Singh (SR no.24851): ayushs@iisc.ac.in

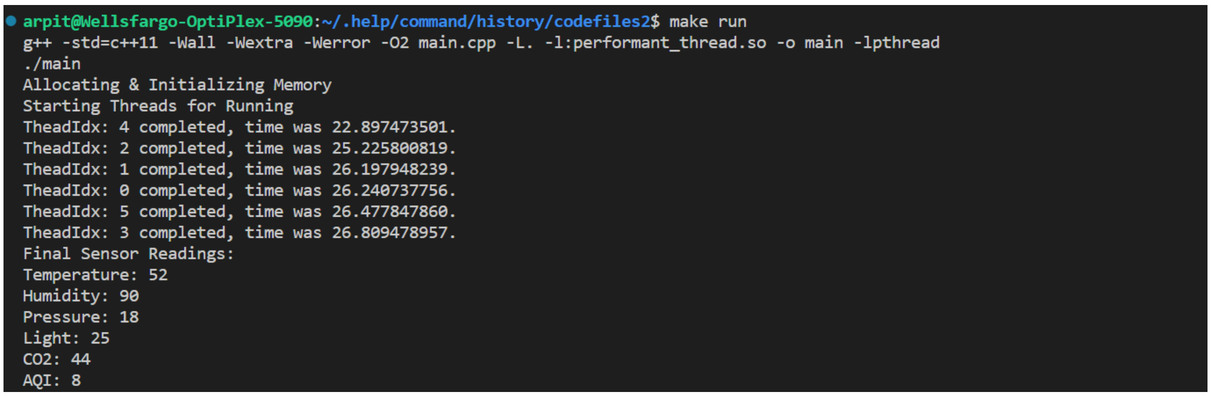
--- Ques 1---

DIFFERENT STEPS FOLLOWED :

Step 1 : Running the main.cpp

>>Make run

This command will execute the main.cpp and will generate the executable main file.



```
arpit@Wellsfargo-OptiPlex-5090:~/.help/command/history/codefiles2$ make run
 g++ -std=c++11 -Wall -Wextra -Werror -O2 main.cpp -L. -l:performant_thread.so -o main -lpthread
 ./main
 Allocating & Initializing Memory
 Starting Threads for Running
 TheadIdx: 4 completed, time was 22.897473501.
 TheadIdx: 2 completed, time was 25.225800819.
 TheadIdx: 1 completed, time was 26.197948239.
 TheadIdx: 0 completed, time was 26.240737756.
 TheadIdx: 5 completed, time was 26.477847860.
 TheadIdx: 3 completed, time was 26.809478957.
 Final Sensor Readings:
 Temperature: 52
 Humidity: 90
 Pressure: 18
 Light: 25
 CO2: 44
 AQI: 8
```

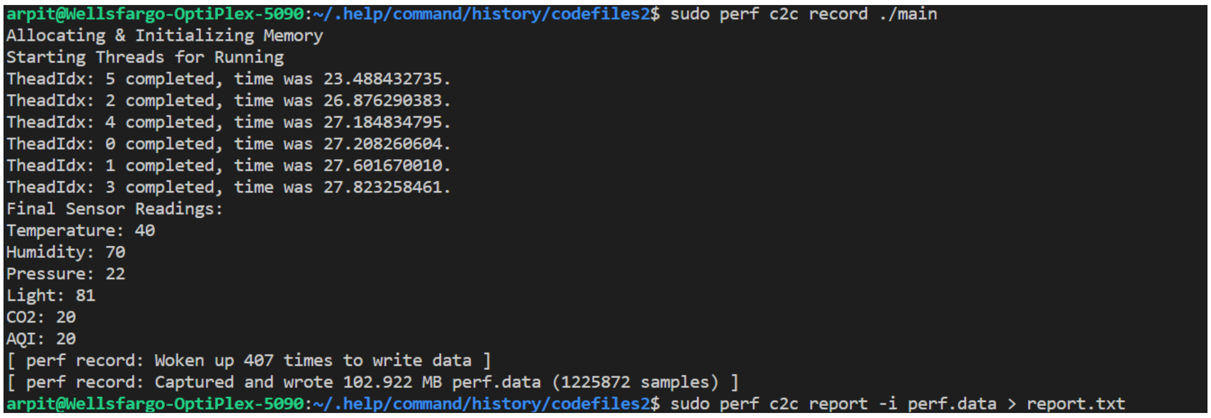
Step 2 : Record the performance using the perf tools

>> sudo perf c2c record ./main

This will record the performance and will generate perf.data file.

>> sudo perf c2c report -i perf.data > report.txt

This will convert the perf.data file to readable report.txt (text file).



```
arpit@Wellsfargo-OptiPlex-5090:~/.help/command/history/codefiles2$ sudo perf c2c record ./main
Allocating & Initializing Memory
Starting Threads for Running
TheadIdx: 5 completed, time was 23.488432735.
TheadIdx: 2 completed, time was 26.876290383.
TheadIdx: 4 completed, time was 27.184834795.
TheadIdx: 0 completed, time was 27.208260604.
TheadIdx: 1 completed, time was 27.601670010.
TheadIdx: 3 completed, time was 27.823258461.
Final Sensor Readings:
Temperature: 40
Humidity: 70
Pressure: 22
Light: 81
CO2: 20
AQI: 20
[ perf record: Woken up 407 times to write data ]
[ perf record: Captured and wrote 102.922 MB perf.data (1225872 samples) ]
arpit@Wellsfargo-OptiPlex-5090:~/.help/command/history/codefiles2$ sudo perf c2c report -i perf.data > report.txt
```

Step 3 : Make changes in the main.cpp file

Since we found that the program is performing worse because of the false sharing present in the code we made changes in the main.cpp file to remove false sharing.

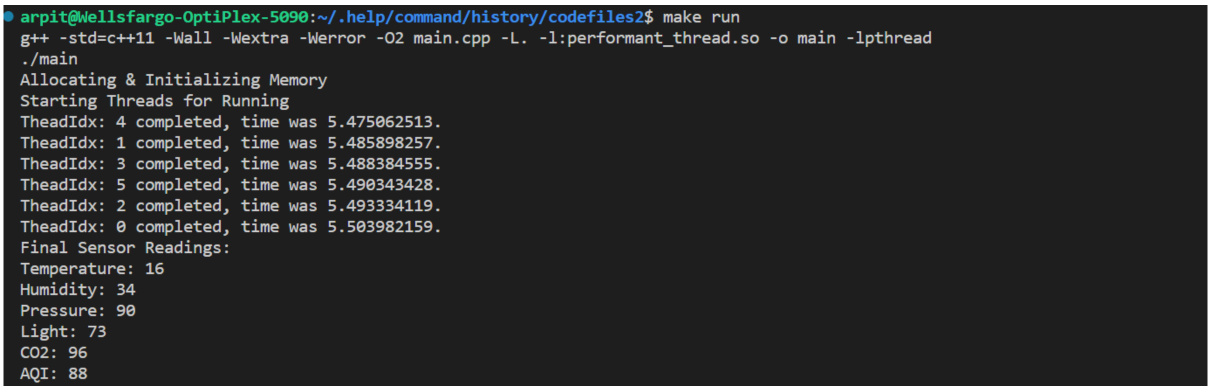
Changes in the code :

```
struct SensorReadings {
    alignas(64) int8_t temperature;
    alignas(64) int8_t humidity;
    alignas(64) int8_t pressure;
    alignas(64) int8_t light;
    alignas(64) int8_t co2;
    alignas(64) int8_t aqi;
};
```

Step 4: Again run the main.cpp file

```
>>Make run
```

This command will execute the main.cpp and will generate the executable main file.



```
arpit@Wellsfargo-OptiPlex-5090:~/.help/command/history/codefiles2$ make run
g++ -std=c++11 -Wall -Wextra -Werror -O2 main.cpp -L. -l:performant_thread.so -o main -lpthread
./main
Allocating & Initializing Memory
Starting Threads for Running
TheadIdx: 4 completed, time was 5.475062513.
TheadIdx: 1 completed, time was 5.485898257.
TheadIdx: 3 completed, time was 5.488384555.
TheadIdx: 5 completed, time was 5.490343428.
TheadIdx: 2 completed, time was 5.493334119.
TheadIdx: 0 completed, time was 5.503982159.
Final Sensor Readings:
Temperature: 16
Humidity: 34
Pressure: 90
Light: 73
CO2: 96
AQI: 88
```

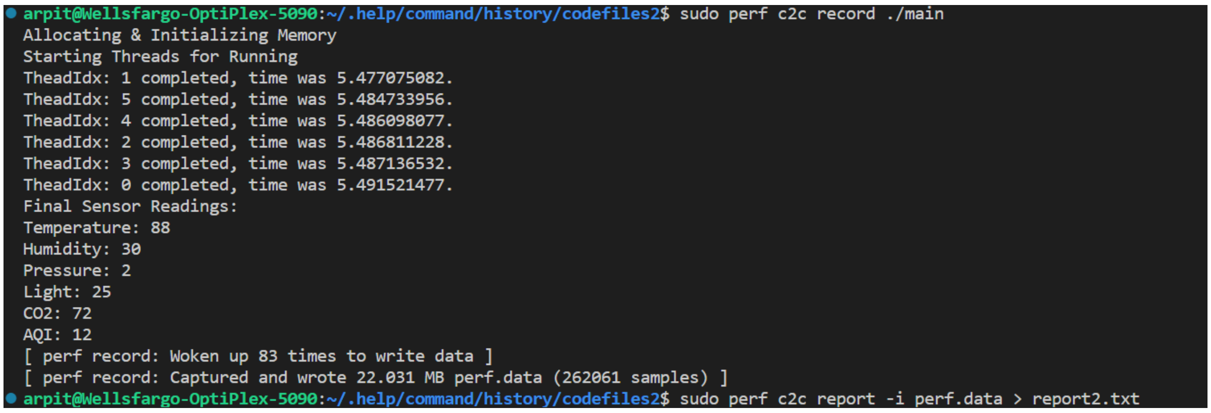
Step 5: Again Record the performance using perf tool

```
>>sudo perf c2c record ./main
```

This will record the performance and will generate a perf.data file.

```
>> sudo perf c2c report -i perf.data > report.txt
```

This will convert the perf.data file to readable report.txt (text file).



```
arpit@Wellsfargo-OptiPlex-5090:~/.help/command/history/codefiles2$ sudo perf c2c record ./main
Allocating & Initializing Memory
Starting Threads for Running
TheadIdx: 1 completed, time was 5.477075082.
TheadIdx: 5 completed, time was 5.484733956.
TheadIdx: 4 completed, time was 5.486098077.
TheadIdx: 2 completed, time was 5.486811228.
TheadIdx: 3 completed, time was 5.487136532.
TheadIdx: 0 completed, time was 5.491521477.
Final Sensor Readings:
Temperature: 88
Humidity: 30
Pressure: 2
Light: 25
CO2: 72
AQI: 12
[ perf record: Woken up 83 times to write data ]
[ perf record: Captured and wrote 22.031 MB perf.data (262061 samples) ]
arpit@Wellsfargo-OptiPlex-5090:~/.help/command/history/codefiles2$ sudo perf c2c report -i perf.data > report2.txt
```

ANALYSIS:

When we ran the "make run" for the first time, the execution time of all the threads were –

Threads	Execution time (in seconds)
0	26.24
1	26.19
2	25.22
3	26.80
4	22.89
5	26.47

Each thread was taking longer time due to the problems in the main.cpp file. After running the program, we also generated a report.txt file to record the performance using the perf tool.

When we observed the report.txt file, we found that

Local HITM Events	6494
Load HITs on shared lines	36084
Total records processed	1,225,872
Local L1d hit	562,481
Local LLC hit	10,163

Local HITM events : A Local HITM event occurs when one core modifies a cache line that another core has, causing it to invalidate and reload that line.

Shared Cache Access : It is measured using the Load HITs on shared lines. The original report showed significant access to shared cache lines, which contributed to contention and false sharing.

Cache efficiency : It is measured through the local L2d hit and Local LLC hit. This all gives proof that there is false sharing between the threads.

This all helped us to identify that our program may suffer from the problem of false sharing. False sharing is a performance issue that can occur in multithreaded applications when threads on different processors modify variables that reside on the same cache line. Despite these variables being logically independent and not actually sharing data, they can still lead to performance degradation due to how modern CPUs manage caching.

After updating main.cpp, the values observed were –

	Before optimization	After optimization
Local HITM Events	6494	0
Load HITs on shared lines	36084	0
Total records processed	1,225,872	262061
Local L1d hit	562,481	130114
Local LLC hit	10,163	42

The report2.txt (after optimization) shows a significant improvement, demonstrating that optimizations effectively reduced false sharing.

Key Differences Between report.txt (Before Optimization) and report2.txt (After Optimization)

1. **Local HITM Events (Hit Modified):**
   - **Before Optimization:** Local HITM events = 6,494
   - **After Optimization:** Local HITM events = 0
   - **Conclusion:** The reduction to zero Local HITM events post-optimization indicates that no false sharing is present. Each thread is now likely working on independent cache lines without cross-core interference.
2. **Shared Cache Line Access:**
   - **Before Optimization:** Total Load HITs on shared lines = 36,084
   - **After Optimization:** Total Load HITs on shared lines = 0
   - **Conclusion:** The original report showed significant access to shared cache lines, which contributed to contention and false sharing. With zero shared cache line hits after optimization, each thread operates on its own distinct cache lines, minimizing interference.
3. **Total Records Processed:**
   - **Before Optimization:** 1,225,872 total records
   - **After Optimization:** 262,061 total records
   - **Conclusion:** The reduction in total records suggests improved performance, as the optimized code accesses fewer records overall. This drop likely reflects the elimination of redundant or stalled accesses due to cache contention.
4. **Cache Efficiency:**
   - **Load L1D hit:**
     - **Before Optimization:** 562,481 L1D hits
     - **After Optimization:** 130,114 L1D hits
   - **Load LLC hit:**
     - **Before Optimization:** 10,163 LLC hits
     - **After Optimization:** 42 LLC hits
   - **Conclusion:** The optimized code shows a greater reliance on the L1 cache with very minimal LLC usage. This improvement indicates better cache locality, as threads can access data directly in their private cache (L1) without needing shared cache levels (LLC).

After optimization the execution time also reduced for each thread :

Threads	Execution time(in seconds)
0	5.50
1	5.48
2	5.49
3	5.48
4	5.47
5	5.49

SUMMARY:

Before Optimization: All threads executed in 22 to 27 seconds. This range suggests potential contention or inefficiencies. The slower threads (around 30 seconds) indicate that some threads might be experiencing delays or competition for shared resources like caches, memory, or even CPU cycles.

After Optimization: All Threads Executed in 5 Seconds. This significant reduction in execution time shows that the changes have effectively resolved the issue. By aligning the SensorReadings structure, we eliminated false sharing between threads. False sharing occurs when multiple threads modify different variables that happen to be on the same cache line, causing unnecessary cache coherence traffic and stalling.

SPEEDUP CALCULATION:

$$\textbf{speedup} = \frac{\boldsymbol{Execution\ time\ before\ optimization}}{\boldsymbol{Execution\ time\ after\ optimization}}$$

for thread0 = 26.24/5.50 = 4.77

for thread1 = 26.19/5.48 = 4.77

for thread2 = 25.22/5.49 = 4.59

for thread3 = 26.80/5.48 = 4.89

for thread4 = 22.89/5.47 = 4.18

for thread5 = 26.47/5.49 = 4.82

taking maximum execution time before optimization = 26.80

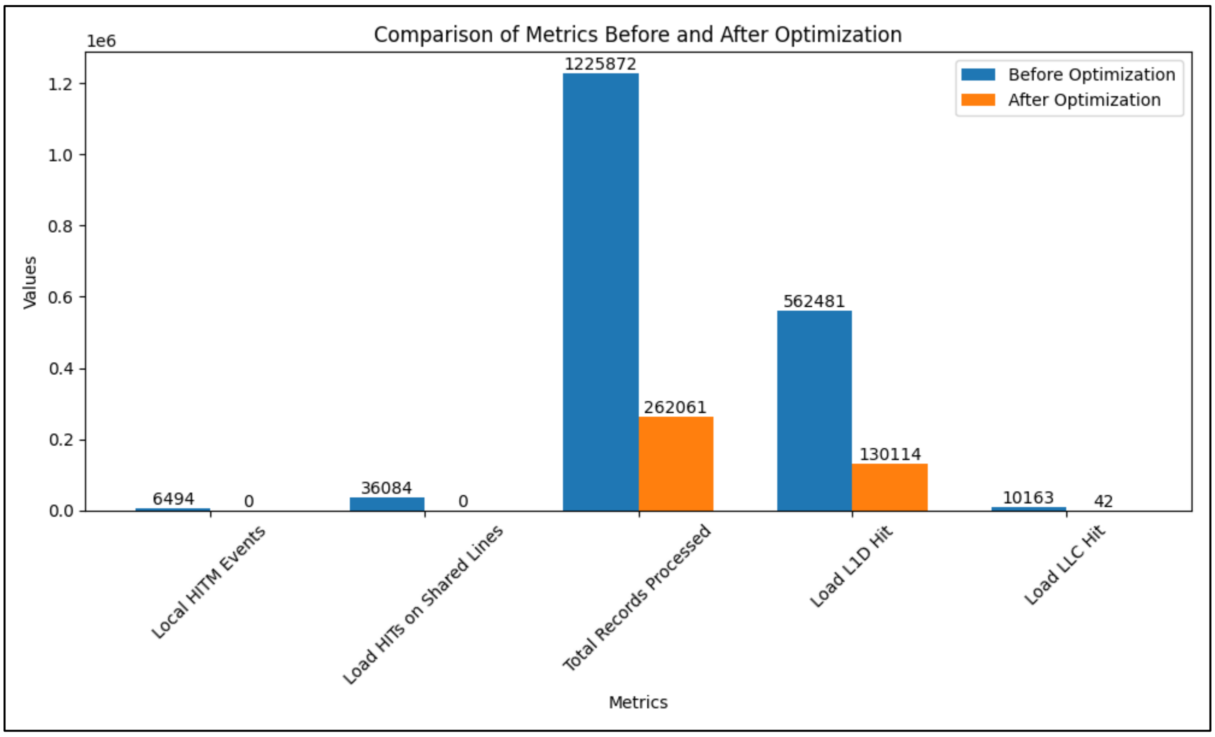
taking maximum execution time after optimization = 5.50

speedup = 26.80 / 5.50 = 4.87

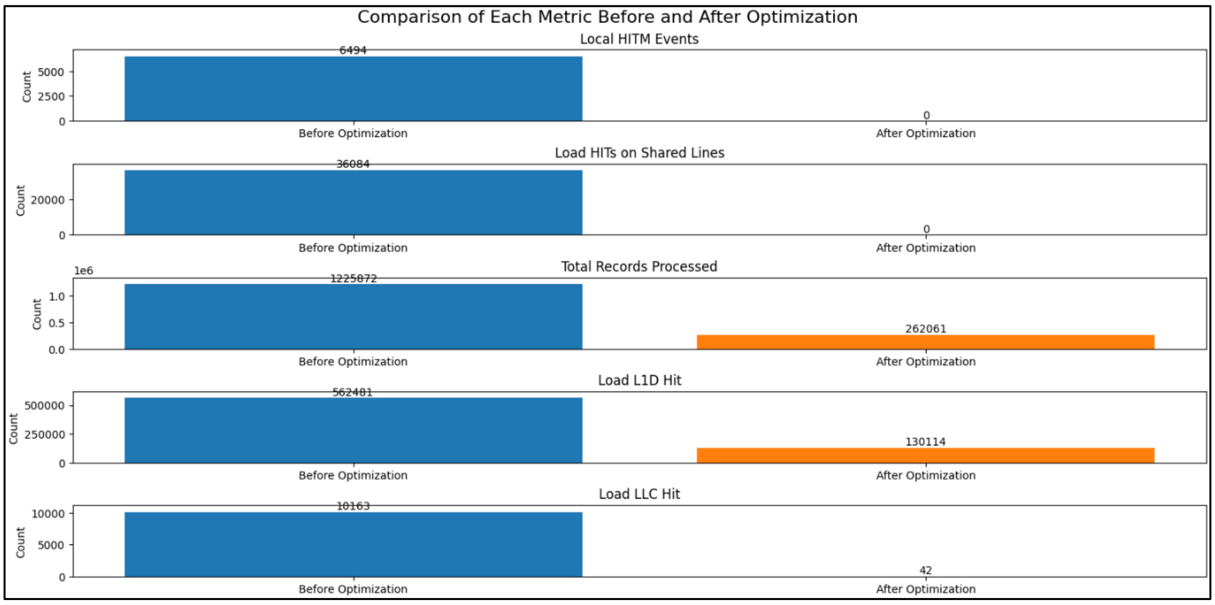
GRAPHS :

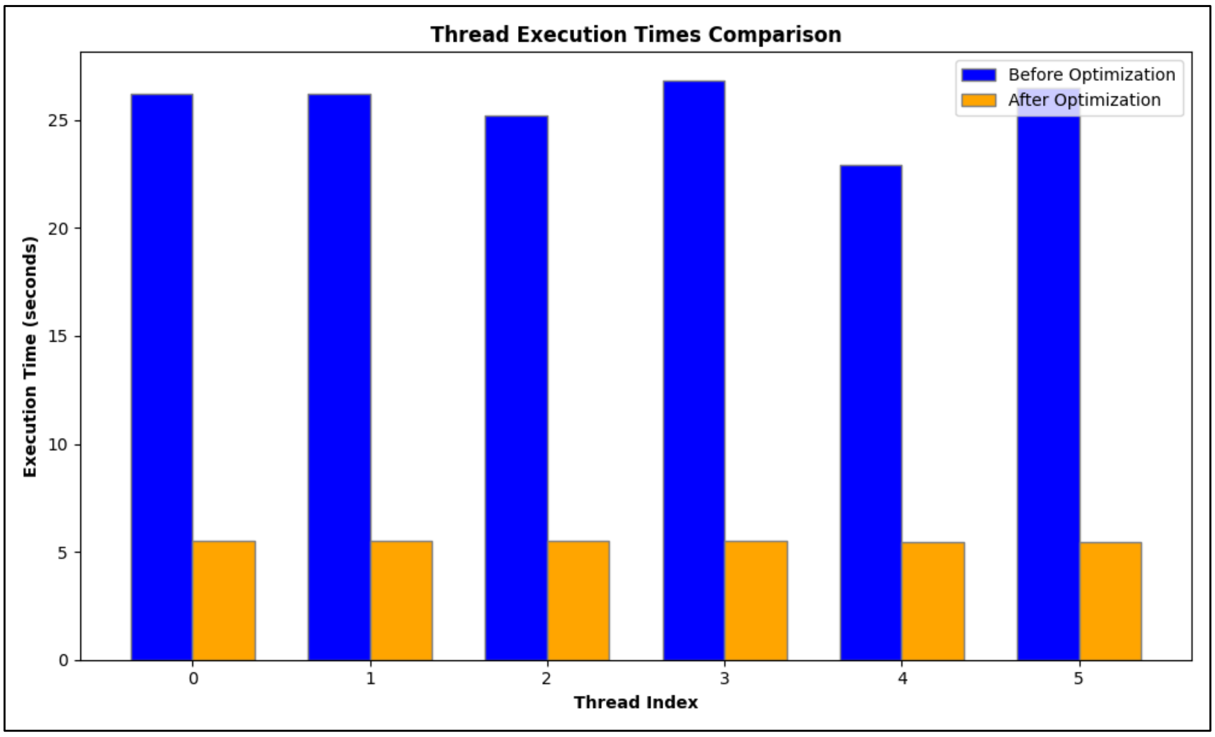
Given below are graphs showing the performance before and after optimization.

Bar Graph showing data before and after optimization



Bar Graph for Individual parameters





Through graphs, it is clearly visible that the performance before optimization was far lower than that of after optimization. But after modifying the main.cpp file i.e. by removing the false sharing between the threads, the performance gained much with a speedup of 4.87.

As given in the question, a team from the Systems Lab wrote a test program in main.cpp to harness the power of the performant_thread.so library. The task seemed straightforward: each thread would work independently on a different part of the dataset, with zero overlap or contention between them. However, as they ran their implementation, they found something troubling. The program performed worse than the original serial implementation.

So we dived into the main.cpp program and found that the test code is suffering from the problem of false sharing as the structure of the sensorReadings was not defined properly. So by making a small change in 'struct sensorReadings' structure by using alignas(64) we removed the problem and the code worked properly giving speedup of approximately 4.87.