Lab 5 : Performance Report For Matrix Multiplication using perf tool

Vedant Saboo
ROLL NUMBER CS19B074
March 18, 2021

1 PROBLEM STATEMENT

Consider three square matrices A, B, and C of size 1024x1024. Fill A and B with random integers ranging from 0 to 10. Write a C program to perform matrix multiplication of A and B and store the result in C. Assume that matrix C is not initialized and any initialization to elements in C should also be counted in runtime calculations. Compile the code with gcc compiler (-O3 optimization) and run the program 10 times and report the average runtime (in milliseconds), cpu-cycles, number of instructions, branch-instructions, cache-references, cache-misses, L1-dcache-loads, L1-dcache-loadmisses, dTLB-loads, dTLB-load-misses, LLC-loads, and LLC-load-misses across the 10 runs for each configuration. Use gettimeofday( ) for calculating the runtime and perf tool to measure the other events.
Consider two scenarios :

- Scenario #1: Both A and B matrices are in row major order.
- Scenario #2: Matrix A is in row major order and matrix B is in column major order.

2 SCENARIO 1

Two matrices, each 1024×1024 are assigned random elements from 0 to 10. These matrices are stores in row major order. These matrices are multiplied and the resultant matrix is stored. We analyse the performance of this matrix multiplication.
NOTE: Although we have calculated average run times for the matrix multiplication function

only, the other stats include the randomisation cost as well.
Matrix Multiplication implements the following :

$$C[i][j] = \sum_{k=0}^{1023} A[i][k] \times B[k][j] \tag{2.1}$$

What we observed :

System Configuration	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz; Memory 12914MB (154MB used); OS Ubuntu 18.04.3 LTS |1 physical processor; 6 cores; 12 threads|12x 2592.01 MHz|12914856|1|6|12| L1 cache : 384KB L2 cache : 1MB L3 cache : 12MB
Scenario #	1
instructions	8,73,75,07,163 :: 1.21 instructions per cycle
branch-instructions	1,12,36,59,203
cache-references	3,11,67,16,967
cache-misses	1,88,525 :: miss per hit ratio = 0.006 %
L1-dcache-loads	2,20,79,50,240
L1-dcache-load-misses	1,08,29,35,760 :: miss per hit ratio = 49.05 %
dTLB-loads	2,17,54,39,194
dTLB-load-misses	1,016 :: miss per hit ratio nearly zero percent
LLC-loads	1,24,53,66,485
LLC-load-misses	2,242 ::miss per hit ratio nearly zero percent
Average Runtime	1.9203102 seconds
CPU Cycles	7,19,25,62,858 :: 3.712 GHz

Table 2.1: Observation Table

3 SCENARIO 2

Two matrices, each 1024 × 1024 are assigned random elements from 0 to 10. First Matrix is stored in row major order while second one is stored in column major order. These matrices are multiplied and the resultant matrix is stored. We analyse the performance of this matrix multiplication.
NOTE: Although we have calculated average run times for the matrix multiplication function only, the other stats include the randomisation cost as well.

$$C[i][j] = \sum_{k=0}^{1023} A[i][k] \times B[j][k] \tag{3.1}$$

What we observed :

System Configuration	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz; Memory 12914MB (154MB used); OS Ubuntu 18.04.3 LTS |1 physical processor; 6 cores; 12 threads|12x 2592.01 MHz|12914856|1|6|12| L1 cache : 384KB L2 cache : 1MB L3 cache : 12MB
Scenario #	2
instructions	2,32,49,22,789 :: 3.30 instructions per cycle
branch-instructions	1,11,64,21,346
cache-references	7,40,24,194
cache-misses	1,70,047 :: miss per hit ratio = 0.230 %
L1-dcache-loads	2,17,53,63,311
L1-dcache-load-misses	3,38,43,564 :: miss per hit ratio = 1.56 %
dTLB-loads	2,19,79,86,754
dTLB-load-misses	958 :: miss per hit ratio nearly zero percent
LLC-loads	9,26,956
LLC-load-misses	1,754 :: miss per hit ratio = 0.19 %
Average Runtime	0.5920697 seconds
CPU Cycles	2,32,49,22,789 :: 3.819 GHz

Table 3.1: Observation Table

4 Inferences and Conclusion

Following prime centres of differences between both the scenarios is evident :

1. Cache references is scenario 1 is huge in comparison to scenario 2
2. Cache misses being same, the miss rate is higher in scenario 2
3. L1 dcache loads are nearly same in both scenarios
4. L1 dcache load misses are very high in scenario 1 (nearly 50 %) while it is significantly lesser in scenario 2 (1.5 %)
5. dTLB loads are similar in both scenarios and misses are minimal
6. LLC loads are high in the scenario 1
7. LLC load miss rate is higher in scenario 2
8. Summing up, number of instructions and cpu cycles both are higher in scenario 1
9. Overall, scenario 2 way is better in terms of latency (1.9 seconds versus 0.6 seconds)

4.1 To Reason the Observations

Scenario 2 exploits the locality provided when a block is loaded into L1 data cache, because during one inner iteration, adjacent memory blocks are being used.
Scenario 1 fails to do so for the matrix B. So it loads a new block very instruction for the second matrix. It is but no surprise that the miss per hit ratio is nearly fifty percent.

It is this L1 cache that brings upon this vast difference of run times in the two scenarios.

———— Reference files ————
code files: CS19B074_01.c, CS19B074_02.c, CS19B074_03.c
list of commands: commands.sh
executable files: test1, test2
output files: outputf1.txt, outputf2.txt, output-perf.txt