

CS610 PROGRAMMING FOR PERFORMANCE

Assignment 3

October 23, 2024

SAHIL BASIA

241110061

Note

- For problems 1 -3, I have tested my code both on my machine and KD lab machines. I used the CSEWS4 machine in the KD lab.
- All the results of problem 4 are produced using the machine in the KD lab. A CSEWS4 machine was used to produce the results.

Ans: **Problem - 1**

Matrix - Matrix Multiplication Implementation

Less Optimized Approach

In this approach I used i j k loop pattern. So in this approach I am picking elements from row of A matrix and column elements of B matrix.

But because the elements of B matrix were stored in row major order, so this resulted in *less_optimized approach*, because now elements are picked in column major order, thus giving poor spatial locality.

SSE4 and AVX2 intrinsics used

1. SSE

- `_mm_load_ps()`
- `_mm_set_ps()`
- `_mm_mul_ps()`
- `_mm_hadd_ps()`

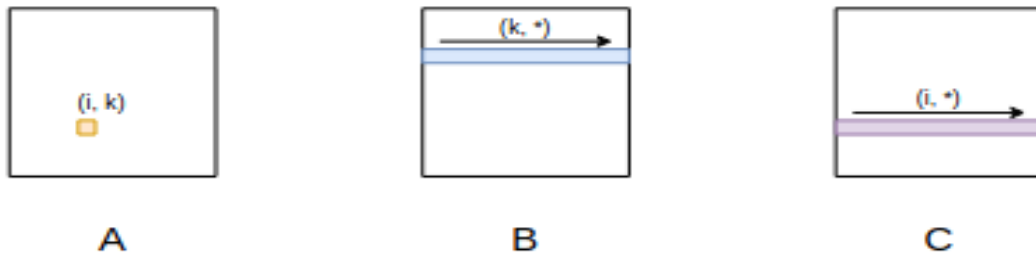
2. AVX2

- `_mm256_load_ps()`
- `_mm256_set_ps()`
- `_mm256_mul_ps()`
- `_mm256_permute2f128_ps()`
- `_mm256_hadd_ps()`

I have written working of these intrinsics as comments in my source code.

More Optimized Approach

In this optimized approach I am using i k j loop pattern. So in this, I am picking 1 element of matrix A and then picking row elements of matrix B and multiplying each element of B with that single element of matrix A and summing these elements in row elements of matrix C. Something like this in below image.



In this approach, we have good spatial locality as access to B and C is row-oriented.

SSE4 and AVX2 intrinsics used

1. SSE

- `_mm_load_ps()`
- `_mm_set1_ps()`
- `_mm_mul_ps()`

- `_mm_add_ps()`
- `_mm_store_ps()`
- `_mm_loadu_ps()`

2. AVX2

- `_mm256_load_ps()`
- `_mm256_set1_ps()`
- `_mm256_mul_ps()`
- `_mm256_store_ps()`
- `_mm256_add_ps()`
- `_mm256_loadu_ps()`

Commands Used

1. Command for optimized approach

```
>>> g++ -g -std=c++17 -masm=att -march=native -O2 -fopenmp -fverbose-asm -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti problem1.cpp -o problem1.out
```

```
>>> ./problem1.out
```

2. Command for less-optimized approach

```
>>> g++ -g -std=c++17 -masm=att -march=native -O2 -fopenmp -fverbose-asm -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti problem1_less_opt.cpp -o problem1_less_opt.out
```

```
>>> ./problem1_less_opt.out
```

Results and Screenshots

```
sahilbasia24@cssews4:~/assignment_3$ ./problem1_less_opt.out
Below is aligned version implementation
Aligned Matmul seq time: 988 ms
1048576 Diffs found over THRESHOLD 1.19209e-07; Max Diff = 0.000293732
Aligned Matmul SSE4 time: 734 ms
1048576 Diffs found over THRESHOLD 1.19209e-07; Max Diff = 0.000291824
Aligned Matmul AVX2 time: 737 ms

Everything below is unaligned version implementation
UnAligned Matmul seq time: 1001 ms
1048576 Diffs found over THRESHOLD 1.19209e-07; Max Diff = 0.000293732
UnAligned Matmul SSE4 time: 779 ms
1048576 Diffs found over THRESHOLD 1.19209e-07; Max Diff = 0.000291824
UnAligned Matmul AVX2 time: 803 ms
sahilbasia24@cssews4:~/assignment_3$
```

The Above image is less-optimized version of matmul. We can see diffs and max diff values. This is because of precision error. In this less_optimized version, I was picking value from a vector using a float pointer. And that pointer is giving values in precision to 2 places only. Therefore there is some precision error in this approach.

```
sahilbasia24@cssews4:~/assignment_3$ ./problem1.out
Below is aligned version implementation
Aligned Matmul seq time: 978 ms
No differences found between base and test versions
Aligned Matmul SSE4 time: 137 ms
No differences found between base and test versions
Aligned Matmul AVX2 time: 86 ms

Everything below is unaligned version implementation
UnAligned Matmul seq time: 973 ms
No differences found between base and test versions
UnAligned Matmul SSE4 time: 182 ms
No differences found between base and test versions
UnAligned Matmul AVX2 time: 139 ms
sahilbasia24@cssews4:~/assignment_3$
```

The above image is optimized approach of matmul.

Approximate speedup of SSE4 = 6.48

Approximate speedup of AVX2 = 8.83

These speedups are related to optimized version

Note: I have attached problem1.cpp(best answer) and problem1_less_opt.cpp(less optimized answer) in zip file.

Ans: Problem - 2

Inclusive Sum Prefix question

For Problem 2, I tested my code on my own machine and on KD lab machine CSEW4.

The results were varying too much. Below are the screenshot of the result.

AVX2 Intrinsics used

- `_mm256_setzero_si256`
- `_mm256_load_si256`
- `_mm256_slli_si256`
- `_mm256_add_epi32`
- `_mm256_add_epi32`
- `_mm256_permutevar8x32_epi32`
- `_mm256_permute2x128_si256`
- `_mm256_store_si256`

This below image is from my personal machine

```
problem2.out  
rubbish@rubbish:~/final_assign3/finals$ ./problem2.out
```

```
Serial version: 65536 time: 97
```

```
OMP version: 65536 time: 55
```

```
SSE version: 65536 time: 41
```

```
AVX2 version: 65536 time: 61
```

Below are the speedups of avx2 over other approaches

speed up of avx2 over sequential approach = 1.59016

speed up of avx2 over openmp approach = 0.901639

speed up of avx2 over sse4 approach = 0.672131

Below are the speedups of openmp over other approaches

speed up of openmp over sequential approach = 1.76364

speed up of openmp over sse4 approach = 0.745455

speed up of openmp over avx2 approach = 1.10909

Below are the speedups of sse4 over other approaches

speed up of sse4 over sequential approach = 2.36585

speed up of sse4 over openmp approach = 1.34146

speed up of sse4 over avx2 approach = 1.4878

This below image is from KD - lab machine CSEWS4


```

sahilbasia24@csews4:~/assignment_3$ ./problem2.out
Serial version: 65536 time: 28
OMP version: 65536 time: 12
SSE version: 65536 time: 15
AVX2 version: 65536 time: 11

Below are the speedups of avx2 over other approaches
speed up of avx2 over sequential approach = 2.54545
speed up of avx2 over openmp approach = 1.09091
speed up of avx2 over sse4 approach = 1.36364

Below are the speedups of openmp over other approaches
speed up of openmp over sequential approach = 2.33333
speed up of openmp over sse4 approach = 1.25
speed up of openmp over avx2 approach = 0.916667

Below are the speedups of sse4 over other approaches
speed up of sse4 over sequential approach = 1.86667
speed up of sse4 over openmp approach = 0.8
speed up of sse4 over avx2 approach = 0.733333
sahilbasia24@csews4:~/assignment_3$ ./problem2.out
Serial version: 65536 time: 98
OMP version: 65536 time: 10
SSE version: 65536 time: 15
AVX2 version: 65536 time: 10

Below are the speedups of avx2 over other approaches
speed up of avx2 over sequential approach = 9.8
speed up of avx2 over openmp approach = 1
speed up of avx2 over sse4 approach = 1.5

Below are the speedups of openmp over other approaches
speed up of openmp over sequential approach = 9.8
speed up of openmp over sse4 approach = 1.5
speed up of openmp over avx2 approach = 1

Below are the speedups of sse4 over other approaches
speed up of sse4 over sequential approach = 6.53333
speed up of sse4 over openmp approach = 0.666667
speed up of sse4 over avx2 approach = 0.666667
sahilbasia24@csews4:~/assignment_3$ □

```

From the above images, we can conclude that the performance of code is system-dependent.

System Specifications

Personal system

```

rubbish@rubbish:~/final_assign3/finals$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:          48 bits physical, 48 bits virtual
Byte Order:             Little Endian
CPU(s):                 12
On-line CPU(s) list:    0-11
Vendor ID:              AuthenticAMD
Model name:             AMD Ryzen 5 4600H with Radeon Graphics
CPU family:             23
Model:                 96
Thread(s) per core:     2
Core(s) per socket:     6
Socket(s):              1
Stepping:               1
Frequency boost:        enabled
CPU max MHz:            3000.0000
CPU min MHz:            1400.0000
BogoMIPS:               5989.05
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext f
xsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf rapl pni pclmulqdq monitor
ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 c
dp_l3 hwpstate ssbd mba ibrs ibpb stibp vmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb
sha_ni xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr rdpru wbnoinvd
cpbpc arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold avic v_vms
ave_vmload vgif v_spec_ctrl umip rdpid overflow_recov succor smca

Virtualization features:
Virtualization:         AMD-V
Caches (sum of all):
L1d:                    192 KiB (6 instances)
L1i:                    192 KiB (6 instances)
L2:                     3 MiB (6 instances)
L3:                     8 MiB (2 instances)
NUMA:
NUMA node(s):           1
NUMA node0 CPU(s):      0-11

```

KD-lab CSEWS4


```
terminal
sahilbasia24@csews4:~/assignment_3$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:          39 bits physical, 48 bits virtual
Byte Order:             Little Endian
CPU(s):                 12
On-line CPU(s) list:    0-11
Vendor ID:              GenuineIntel
Model name:             Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
CPU family:             6
Model:                 158
Thread(s) per core:     2
Core(s) per socket:     6
Socket(s):              1
Stepping:               10
CPU max MHz:            4600.0000
CPU min MHz:            800.0000
BogoMIPS:               6399.96
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge nca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp ln constant_tsc art arch_perfmon
pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4.1 sse4.2 x2apic movbe p
opcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb pti ssbd ibrs ibpb stibp tpr_shadow flexpriority ept vpid ept_ad fsgsbase tsc_adjust
bmi1 avx2 snep bmi2 erms invpcid npx rdseed adx snap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp vnni md_clea
r flush_l1d arch_capabilities

Virtualization features:
Virtualization:         VT-x
Caches (sum of all):
L1d:                    192 KiB (6 instances)
L1i:                    192 KiB (6 instances)
L2:                     1.5 MiB (6 instances)
L3:                     12 MiB (1 instance)
NUMA:
NUMA node(s):           1
NUMA node0 CPU(s):      0-11
```

Commands Used

```
>>> g++ -g -std=c++17 -masm=att -march=native -O2 -fopenmp -
fverbose-asm -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti
problem2.cpp -o problem2.out
>>> ./problem2.out
```

Note: To check the correctness of the code, I have commented out the print function of the array. Uncomment to print the array

Ans: Problem - 3

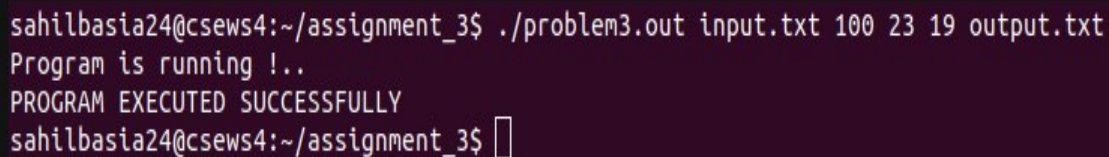
In this question, I have done two implementations. I have commented on the other implementation within the code only.

Command Used

```
>>> g++ -g -std=c++17 -masm=att -march=native -O2 -fopenmp -
fverbose-asm -fno-asynchronous-unwind-tables -fno-exceptions -fno-rtti
problem3.cpp -o problem3.out
>>> ./problem3.out input.txt 100 23 20 output.txt
```

Syntax of running:

```
./problem3.out [path_to_input_to_read]
[number_of_producer_threads] [number_of_line_each_thread_reads]
[size_of_shared_memory_buffer] [path_to_output_file]
```



```
sahilbasia24@cs4:~/assignment_3$ ./problem3.out input.txt 100 23 19 output.txt
Program is running !..
PROGRAM EXECUTED SUCCESSFULLY
sahilbasia24@cs4:~/assignment_3$
```

Ans: Problem - 4

The original file given took this much time to execute: **420.2261 seconds**

Part -1

Commands Used

- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0.c -o problem4-v0.out
- ./problem4-v0.out

- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_1.c -o problem4-v0_1.out
- ./problem4-v0_1.out

- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_2.c -o problem4-v0_2.out
- ./problem4-v0_2.out

```
- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_3.c  
-o problem4-v0_3.out  
- ./problem4-v0_3.out
```

```
- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_4.c  
-o problem4-v0_4.out  
- ./problem4-v0_4.out
```

```
- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_5.c  
-o problem4-v0_5.out  
- ./problem4-v0_5.out
```

```
- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_6.c  
-o problem4-v0_6.out  
- ./problem4-v0_6.out
```

The same commands will be used for the rest of the files for part 1

```
- gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L problem4-v0_7.c  
-o problem4-v0_7.out  
- ./problem4-v0_7.out
```

Below is the approach and a brief explanation

1. In the provided source code, we were given a 10-level loop nest. There was no dependency between the for loops. For optimization, I first tried to remove extra variables or extra ALU computations that were performed in the code. And therefore changed the code in for loop section to remove those extra computations that were performed.

The code for this change is present in **problem4-v0_1.c** and the result file is created as **results-v0_1.txt**

Time of execution : **362.5401 seconds**

Speedup achieved is : **1.116**

2. Next I tried loop permutations, but it was no use because the results came different in the result file because we can see that in original code the for loop executes first, also ordering the loops didn't resulted in any performance gain. I swapped for loop 1 with for loop 10 and for loop 2 with for loop 9. The code is present in **problem4-v0_2.c**, and the result file is created with **results-v0_2.txt**.

The result was very slow in comparison to the original loop permutation. Also, the results in original **results-v0.txt** and **results-v0_2.txt** were not the same.

Time of execution : **549.5401 seconds**

Speedup achieved is : **0.765**

Not only this, the loop permutation also results in generating wrong result file and it didn't matched with original result file of sequential version.

3. After this I worked with original loop permutation only and started optimizing the code. I introduced if conditions prior only after each assignment of the q^* variable. In the original code, each time the $q_1, q_2, q_3 \dots$ were computed, the condition on them is checked in the end only. In case any of the conditions is false, the results were not written in the file. This results in a lot of ALU computation.

So my changed code is present in **problem4-v0_3.c** and result file is created as **results-v0_3.txt**. This time i got some improvement on my speedup and the time of execution reduced drastically.

Time of execution : **181.7175 seconds**

Speedup achieved is : **2.320**

4. Next, I tried to optimize the **problem4-v0_3 further.c** code by unrolling the inner loop. I initially started with 2 times unrolling the inner loop, and the code is present in **problem4-v0_4.c**, and

the result is generated in **results-v0_4.txt**. I further tried doing 3 times unrolling loop, but it degraded the performance as compared to 2 times unrolling. 3 time unrolling code is present in **problem4-v0_5.c** and results in generated in **results-v0_5.txt**
Time of execution (**problem4-v0_4.c**): **187.5401 seconds**
Speedup achieved is (**problem4-v0_4.c**): **2.249**

Time of execution (**problem4-v0_5.c**): **198.2401 seconds**
Speedup achieved is (**problem4-v0_5.c**): **2.1212**

5. Next, I took reference from the original source code that was given. In that code, we can see that some variables are precomputed within each loop. So, I have tried to improve the spatial locality by introducing the array concept. The code is present in **problem4-v0_6.c**, and the result is stored in **problem4-v0_6.txt**. To further optimize, I introduced a 2-time inner loop unrolling in this code only and created a new file as **problem4-v0_7.c**, and the result of this is stored in **results-v0_7.txt**.

Time of execution (**problem4-v0_6.c**): **188.3401 seconds**
Speedup achieved is (**problem4-v0_6.c**): **2.2340**

Time of execution (**problem4-v0_7.c**): **193.5971 seconds**
Speedup achieved is (**problem4-v0_7.c**): **2.1761**

From all the approaches I tried, code problem4-v0_3.c, problem4-v0_4.c gave me nearly the same speedup. That is, using code optimization + 2x inner most loop unrolling.

Best speed up achieved is = 2.320 and percentage speed up increase is $\approx 56\%$.

The final source code is saved as 241110061-prob4-v1.c

And command to this source code:

```
gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L 241110061-prob4-v1.c -o 241110061-prob4-v1.out
```

Part 2

For this part, I used problem4-v0_4.c source code and implemented openmp constructs.

Construct used is :

1. `#pragma omp parallel for reduction(+ : pnts) num_threads(10) private(x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, q1, q2, q3, q4, q5, q6, q7, q8, q9, q10)`
2. `#pragma omp critical`

Commands Used

- `gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L 241110061-prob4-v2.c -o 241110061-prob4-v2.out -fopenmp`
- `./241110061-prob4-v2.out`

Results

Sequential version time: 420.2261 seconds

OpenMP version time : 49.0235 seconds

Speed Up achieved: $8.572 \approx 8.5X$ *speedup*.

System Information

KD system - CSEWS4

```

sahilbasia24@csews4:~/assignment_3$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Address sizes:          39 bits physical, 48 bits virtual
Byte Order:             Little Endian
CPU(s):                 12
On-line CPU(s) list:    0-11
Vendor ID:              GenuineIntel
Model name:             Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
CPU family:             6
Model:                  158
Thread(s) per core:     2
Core(s) per socket:     6
Socket(s):              1
Stepping:               10
CPU max MHz:            4600.0000
CPU min MHz:            800.0000
BogoMIPS:               6399.96
Flags:                  fpu vme de pse tsc msr pae nce cx8 apic sep mtrr pge nca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpeigb rdtscp ln constant_tsc art arch_perfmon
pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4.1 sse4.2 x2apic movbe p
opcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb pti ssbd ibrs ibpb stibp tpr_shadow flexpriority ept vpid ept_ad fsgsbase tsc_adjust
bmi1 avx2 snep bmi2 erms invpcid mpx rdseed adx snap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp vnni md_clea
r flush_lld arch_capabilities

Virtualization features:
Virtualization:         VT-x
Caches (sum of all):
L1d:                    192 KiB (6 instances)
L1i:                    192 KiB (6 instances)
L2:                     1.5 MiB (6 instances)
L3:                     12 MiB (1 instance)
NUMA:
NUMA node(s):           1
NUMA node0 CPU(s):      0-11

```

