CS610: Programming for Performance
Assignment 3

Yash Gupta (190997)

1 Problem 1

The provided reference code could have poor performance due to cache misses while accessing `A[j][i]` (as it is a column wise access) and other cache misses due to the arrays not fitting in the cache.

Outer loop unrolling + inner loop jamming would reduce control overhead of loops and would also exploit spatial locality while accessing `A[j][i]`. Loop tiling/blocking could also improve data reuse. Further, we could use `#pragma GCC ivdep` along with `__restrict__` and `__builtin_assume_aligned()` to help the compiler vectorize the code and optimize it further.

1.1 Performance: Optimization level -O2

Reference Version: Matrix Size = 8192, 0.35889 GFLOPS; Time = 0.74795 sec

Optimized Version 1: Loop interchange: Matrix Size = 8192, Time = 0.70209 sec, Speedup = 1.06533
No differences found between base and test versions

Optimized Version 2: Loop fission: Matrix Size = 8192, Time = 0.72491 sec, Speedup = 1.03179
No differences found between base and test versions

Optimized Version 3: 2 times inner loop unrolling: Matrix Size = 8192, Time = 0.70254 sec, Speedup = 1.06464
No differences found between base and test versions

Optimized Version 4: 2 times outer loop unrolling + inner loop jamming: Matrix Size = 8192, Time = 0.51300 sec, Speedup = 1.45800
No differences found between base and test versions

Optimized Version 5: 2x2 blocking: Matrix Size = 8192, Time = 0.53715 sec, Speedup = 1.39244
No differences found between base and test versions

Optimized Version 6: 4 times outer loop unrolling + inner loop jamming: Matrix Size = 8192, Time = 0.35708 sec, Speedup = 2.09464
No differences found between base and test versions

Optimized Version 7: 4x4 blocking: Matrix Size = 8192, Time = 0.41542 sec, Speedup = 1.80049
No differences found between base and test versions

Optimized Version 8: 8 times outer loop unrolling + inner loop jamming: Matrix Size = 8192, Time = 0.47429 sec, Speedup = 1.57700
No differences found between base and test versions

Optimized Version 9: 8x8 blocking: Matrix Size = 8192, Time = 0.51589 sec, Speedup = 1.44984
No differences found between base and test versions

Optimized Version 10: 4 times outer loop unrolling + inner loop jamming + ivdep: Matrix Size = 8192, Time = 0.35772 sec, Speedup = 2.09088
No differences found between base and test versions

Optimized Version 11: 4x4 blocking + ivdep: Matrix Size = 8192, Time = 0.41641 sec, Speedup = 1.79620
No differences found between base and test versions

Optimized Version 12: 4 times outer loop unrolling + inner loop jamming + ivdep + restrict: Matrix Size = 8192, Time = 0.27281 sec, Speedup = 2.74171
No differences found between base and test versions

Optimized Version 13: 4x4 blocking + ivdep + restrict: Matrix Size = 8192, Time = 0.37413 sec, Speedup = 1.99920
No differences found between base and test versions

Optimized Version 14: 4 times outer loop unrolling + inner loop jamming + ivdep + restrict + aligned: Matrix Size = 8192, Time = 0.26578 sec, Speedup = 2.81418
No differences found between base and test versions

Optimized Version 15: 4x4 blocking + ivdep + restrict + aligned: Matrix Size = 8192, Time = 0.37343 sec, Speedup = 2.00295
No differences found between base and test versions

Intrinsics Version: Optimized Version 14 (4 times outer loop unrolling + inner loop jamming + ivdep + restrict + aligned): Matrix Size = 8192, Time = 0.28887 sec, Speedup = 2.58921
No differences found between base and test versions

1.2 Performance: Optimization level -O3

Reference Version: Matrix Size = 8192, 0.39204 GFLOPS; Time = 0.68471 sec

Optimized Version 1: Loop interchange: Matrix Size = 8192, Time = 0.62893 sec, Speedup = 1.08869
No differences found between base and test versions

Optimized Version 2: Loop fission: Matrix Size = 8192, Time = 0.71324 sec, Speedup = 0.96000
No differences found between base and test versions

Optimized Version 3: 2 times inner loop unrolling: Matrix Size = 8192, Time = 0.72218 sec, Speedup = 0.94812

No differences found between base and test versions

Optimized Version 4: 2 times outer loop unrolling + inner loop jamming: Matrix Size = 8192, Time = 0.52214 sec, Speedup = 1.31135
No differences found between base and test versions

Optimized Version 5: 2x2 blocking: Matrix Size = 8192, Time = 0.49312 sec, Speedup = 1.38854
No differences found between base and test versions

Optimized Version 6: 4 times outer loop unrolling + inner loop jamming: Matrix Size = 8192, Time = 0.36615 sec, Speedup = 1.87006
No differences found between base and test versions

Optimized Version 7: 4x4 blocking: Matrix Size = 8192, Time = 0.38081 sec, Speedup = 1.79806
No differences found between base and test versions

Optimized Version 8: 8 times outer loop unrolling + inner loop jamming: Matrix Size = 8192, Time = 0.47556 sec, Speedup = 1.43979
No differences found between base and test versions

Optimized Version 9: 8x8 blocking: Matrix Size = 8192, Time = 0.51804 sec, Speedup = 1.32174
No differences found between base and test versions

Optimized Version 10: 4 times outer loop unrolling + inner loop jamming + ivdep: Matrix Size = 8192, Time = 0.27424 sec, Speedup = 2.49675
No differences found between base and test versions

Optimized Version 11: 4x4 blocking + ivdep: Matrix Size = 8192, Time = 0.38155 sec, Speedup = 1.79458
No differences found between base and test versions

Optimized Version 12: 4 times outer loop unrolling + inner loop jamming + ivdep + restrict: Matrix Size = 8192, Time = 0.23573 sec, Speedup = 2.90461
No differences found between base and test versions

Optimized Version 13: 4x4 blocking + ivdep + restrict: Matrix Size = 8192, Time = 0.33540 sec, Speedup = 2.04151
No differences found between base and test versions

Optimized Version 14: 4 times outer loop unrolling + inner loop jamming + ivdep + restrict + aligned: Matrix Size = 8192, Time = 0.25099 sec, Speedup = 2.72803
No differences found between base and test versions

Optimized Version 15: 4x4 blocking + ivdep + restrict + aligned: Matrix Size = 8192, Time = 0.33698 sec, Speedup = 2.03191
No differences found between base and test versions

Intrinsics Version: Optimized Version 14 (4 times outer loop unrolling + inner loop jamming + ivdep + restrict + aligned): Matrix Size = 8192, Time = 0.29261 sec, Speedup = 2.34002
No differences found between base and test versions

1.3 Summary

For both optimization levels -O2 and -O3, "4 times outer loop unrolling + inner loop jamming + ivdep + restrict + aligned" gave the best performance with a speedup of 2.81 and 2.73 respectively.

The AVX2 intrinsic versions gave a speedup of 2.59 and 2.34 for optimization levels -O2 and -O3 respectively.

2 Problem 2

Sequential sum: 140737479966720 in 0.012142 seconds
Parallel sum (thread-local, atomic): 140737479966720 in 0.00296654 seconds, Speedup = 4.093
Parallel sum (worksharing construct): 140737479966720 in 0.00136787 seconds, Speedup = 8.8766
Parallel sum (OpenMP tasks): 140737479966720 in 0.116067 seconds, Speedup = 0.104612

3 Problem 3

3.1 Compiler

Compiler version:
g++ (Ubuntu 9.4.0-1ubuntu1 20.04.2) 9.4.0

Compilation command:
`g++ -msse4 -mavx2 -mavx512f -march=native -O3 -o 190997-prob3 190997-prob3.cpp`

3.2 Speedups

Serial version: 2048 time: 1290
OMP version: 2048 time: 1477
SSE version: 2048, time: 768, speedup wrt serial version: 1.67969, speedup wrt OMP version: 1.92318
AVX2 version: 2048, time: 1542, speedup wrt serial version: 0.836576, speedup wrt OMP version: 0.957847
AVX512 version: 2048, time: 2246, speedup wrt serial version: 0.574354, speedup wrt OMP version: 0.657614

4 Problem 4

Workstation used: csews27

Compilation command:
```
gcc -O3 -std=c17 -D_POSIX_C_SOURCE=199309L -fopenmp 190997-problem4-v3.c -o 190997-problem4-v3.out
```

Sequential version:
result pnts: 11608
Total time = 417.346649 seconds

OpenMP version:
result pnts: 11608
Total time = 12.105336 seconds

Speedup = 34.48