CS610 Semester 2024–2025-I
Assignment-3

Sandeep Nitharwal
(210921)

October 23, 2024

Compilation instruction and System information

Provided a make file for compilation.
−− > make
It will give 5 object files problem1.out, problem2.out, problem3.out, problem4-v1.out and problem4-v2.out for corresponding problems and parts

For problem3 use below input format.
./problem3.out -inp=< *input_path* > -thr=*T* -lns=*N* -buf=*M* -out=< *output_path*
where buffersize is > 0.

All the below result are generated on KD csews1 system with ip(172.27.19.1)

Problem.1

Below are the results(in seconds).

N	Seq	SSE4	SSE4_Aligned	AVX2	AVX2_Aligned
1024	1.02	0.19	0.19	0.15	0.13
1536	9.10	0.82	0.81	0.71	0.64
2048	50.6	2.34	2.33	2.01	1.90

Through above data we can observe that AVX2 and SSE4 have similar performance just AVX2 is slightly better and aligning does not have much effect on both and they are far better than sequential code.

Problem.2

Below are the results(in microseconds).

N	Serial	OMP	SSE4	AVX2
1 << 16	147	12	13	11
1 << 20	573	332	290	306
1 << 24	13012	10999	11336	11048
1 << 28	186476	179770	184828	181551
1 << 30	733634	716464	727469	713225

Problem.4

Below are the results(in seconds).

Serial	Part1	Part2
387	147	35

Part(i)

1. **Loop Permutation**: Since all the loops are similar so permutation will not help.

2. **Loop Unrolling**: Since the inner most loop has many instructions already and the loop runs 13 times only so unrolling it will not have much impact.

3. **LICM**: Since the innermost loop has large computational requirements and a major part of it remains constant throughout that loop, we can move these constant calculations outside the loop to save computation time. I applied this optimization for two levels, r9 and r10, which resulted in a speedup of almost 2.5x. Applying it to further levels might not yield significant additional speedups since it would require new variables as well as extra load-store operations. These overhead costs would likely offset the computational savings, as the loop iterations are small. When I applied this optimization for one level, my execution time reduced to 180s, and for two levels, it decreased further to 150s.

Speedup ≈ 2.5x

Part(ii)

For OpenMp version I have applied **#pragma omp parallel for ordered** (parallization) to the outermost loop on top of modified version of part(i).
In order to ensure correctness and ordered printing:
–> Introduced ordered clause in outmost loop as well as for the print statements.
–> Correct description of privates.
–> Use of atomic increment of pnts.

I have also tried of applying vectorization via **#pragma omp simd** but it didn't work.

Speedup as compared to sequential version ≈ 11.3x