Monique Nguyen
03/26/25
CSC656-01

PART 1

```
[100%] Built target sum_direct
moniquee@perlmutter:login13:~/sum_harness_instructional/build> ./sum_direct
Working on problem size N=8388608
 inside direct_sum problem_setup, N=8388608
 inside direct_sum perform_sum, N=8388608
 Elapsed time is : 0.01
 Sum result = 16777216.000000
Working on problem size N=16777216
 inside direct_sum problem_setup, N=16777216
 inside direct_sum perform_sum, N=16777216
 Elapsed time is : 0.02
 Sum result = 33554432.000000
Working on problem size N=33554432
 inside direct_sum problem_setup, N=33554432
 inside direct_sum perform_sum, N=33554432
 Elapsed time is : 0.03
 Sum result = 67108864.000000
Working on problem size N=67108864
 inside direct_sum problem_setup, N=67108864
 inside direct_sum perform_sum, N=67108864
 Elapsed time is : 0.07
 Sum result = 134217728.000000
Working on problem size N=134217728
 inside direct_sum problem_setup, N=134217728
 inside direct_sum perform_sum, N=134217728
 Elapsed time is : 0.13
 Sum result = 268435456.000000
Working on problem size N=268435456
 inside direct_sum problem_setup, N=268435456
 inside direct_sum perform_sum, N=268435456
 Elapsed time is : 0.27
 Sum result = 536870912.000000
moniquee@perlmutter:login13:~/sum_harness_instructional/build>
```

PART 2

```
moniquee@perlmutter:login13:~/sum_harness_instructional/build> ./sum_vector
Working on problem size N=8388608
 inside sum_vector problem_setup, N=8388608
 inside sum_vector perform_sum, N=8388608
 Elapsed time is : 0.01
 Sum result = 105553103683584.000000
Working on problem size N=16777216
 inside sum_vector problem_setup, N=16777216
 inside sum_vector perform_sum, N=16777216
 Elapsed time is : 0.02
 Sum result = 422212439900160.000000
Working on problem size N=33554432
 inside sum_vector problem_setup, N=33554432
 inside sum_vector perform_sum, N=33554432
 Elapsed time is : 0.03
 Sum result = 1688849809932288.000000
Working on problem size N=67108864
 inside sum_vector problem_setup, N=67108864
 inside sum_vector perform_sum, N=67108864
 Elapsed time is : 0.07
 Sum result = 6755399340392448.000000
Working on problem size N=134217728
 inside sum_vector problem_setup, N=134217728
 inside sum_vector perform_sum, N=134217728
 Elapsed time is : 0.13
```

PART 3

```
moniquee@perlmutter:login13:~/sum_harness_instructional/build> ./sum_indirect
Working on problem size N=8388608
 inside sum_indirect problem_setup, N=8388608
 inside sum_indirect perform_sum, N=8388608
 Elapsed time is : 0.01
 Sum result = 35185329287552.000000
Working on problem size N=16777216
 inside sum_indirect problem_setup, N=16777216
 inside sum_indirect perform_sum, N=16777216
 Elapsed time is : 0.02
 Sum result = 140747573256960.000000
Working on problem size N=33554432
 inside sum_indirect problem_setup, N=33554432
 inside sum_indirect perform_sum, N=33554432
 Elapsed time is : 0.03
 Sum result = 563039580898816.000000
Working on problem size N=67108864
 inside sum_indirect problem_setup, N=67108864
 inside sum_indirect perform_sum, N=67108864
 Elapsed time is : 0.07
 Sum result = 2251825355787264.000000
Working on problem size N=134217728
 inside sum_indirect problem_setup, N=134217728
 inside sum_indirect perform_sum, N=134217728
 Elapsed time is : 0.13
 Sum result = 9007091559372800.000000
Working on problem size N=268435456
 inside sum_indirect problem_setup, N=268435456
 inside sum_indirect perform_sum, N=268435456
 Elapsed time is : 0.29
 Sum result = 36029977089926896.000000
moniquee@perlmutter:login13:~/sum_harness_instructional/build>
```

Analysis Questions

1)What types of operations are more expensive and why, and which of the codes is performing a larger number of more expensive operations?

The most expensive operation would be the indirect sum operator. This type of operator uses random memory access which often has cache misses and high latency. While compared to the vector sum and the direct sum they both access memory by sequential access. Cache misses can be calculated using the miss rate and miss penalty, which are critical metrics for evaluating memory system efficiency. (Page 400)

2) Computational rate. Which of the 3 methods has the best computational rate (MFLOP/s)? Why?

Out of the three methods the one with the best computational rate would be the vector method. The reason why the vector method would be the best is because it uses SIMD which allows for performing multiple additions in parallel within a single instruction. Meaning that vectors can execute adding commands much faster than scalar operations. (Page 510-513)

3) Memory bandwidth usage. Of the 2 methods vector sum and indirect sum, which has higher levels of memory bandwidth utilization? Why?

Vector sum has a higher level of memory bandwidth utilization. In the above solutions we stated that indirect sum uses random memory access while vector sum uses sequential memory access. Therefore why vector sum has a higher memory bandwidth utilization.(Page 386-404)

4) Memory latency. Of the 2 methods vector sum and indirect sum, which shows lower levels of memory latency? Why?

Vector sum would have a lower memory latency, because of the use of SIMD and the sequential access pattern. Since the indirect sum accesses memory randomly there are higher chances of cache misses and inefficient memory utilization causing higher memory latency. (Page 400)