Homework 1 (Due 9/14, 12pm)

All computation should use BMI Cluster. Please develop code for the BMI cluster (not Apple M1 or M2).

All write ups should be submitted via github classroom assignment 1. (https://classroom.github.com/a/unOD6dVk). You can use word, markdown, or pdf for the write ups. Please submit a single write up document for all questions to the github repo.

For the sequential matrix multiplication and PI estimation code, you can use your code from assignment 0. If your assignment 0 is python base, you can use ChatGPT to translate to c/c++.

For this assignment, use compilation flag -O1 to avoid compiler auto vectorization for the performance comparison. Do not use compilation flag -O2 or -O3.

1 Memory hierarchy, latency, and bandwidth (10 pts)

1.1 Define spatial locality and temporal locality.

1.2 What is the peak flops of a quad core processor running at 2 Ghz and has ILP of 6 operations per cycle?

1.3 (Grama 2.2) Consider a memory system with a level 1 cache of 32 KB and DRAM of 512 MB with the processor operating at 2 GHz. The latency to L1 cache is one cycle and the latency to DRAM is 100 cycles. In each memory cycle, the processor fetches four words (cache line size is four words). What is the peak achievable performance of a dot product of two vectors of 4K? Note: Where necessary, assume an optimal cache placement policy.

```
/* dot product loop */
for (i = 0; i < dim; i++)
  dot_prod += a[i] * b[i];
```

1.4 (Grama 2.3) Now consider the problem of multiplying a dense matrix with a vector using a two-loop dot-product formulation. The matrix is of dimension 4K x 4K. (Each row of the matrix takes 16 KB of storage.) What is the peak achievable performance of this technique using a two-loop dot-product based matrix-vector product?

```
/* matrix-vector product loop */
for (i = 0; i < dim; i++)
  for (j = 0; i < dim; j++)
    c[i] += a[i][j] * b[j];
```

2 Amdalh's law and Gustafson's law (10 pts)

2.1 Suppose 30% of a program cannot be parallelized, and the rest are perfectly parallelizable. Plot the strong and weak scaling behavior for core counts P 1 to 1,000 (use nearest powers of 2).

2.2 Now suppose the program is 30% sequential globally, 30% that is parallelized by node (sequential within the node, each node containing 4 cores), and the remaining 40% is parallelizable by core. Formulate the performance expressions and plot the strong and weak scaling behavior for core counts P 1 to 1,000 (use nearest powers of 2). Assume data size increases with number of cores P for weak scaling.

3 SIMD Simple matrix multiplication (C[N] = A[N][N] x B[N]) (10points, 5 points for write-up)

3.1 Starting with the simple matrix multiplication code from HW 0, re-implement using SIMD instructions (c/c++). Describe strategies for computing vector dot product using SIMD instructions.

3.2 What is the speedup? Plot the strong scaling (as a function of number of threads) and the weak scaling (as a function of the number of points). Compare to your sequential code.

3.3 Describe the memory access pattern and any spatial or temporal locality.

4 SIMD Compute π (10 points for code, 5 points for write-up)

4.1 In homework 0, you computed PI using a Monte Carlo approach. Parallelize it using SIMD. Compare the error with the value of $\pi = \cos^{-1}(-1.0)$.

4.2 What is the speedup? Plot the strong scaling (as a function of number of threads) and the weak scaling (as a function of the number of points).

4.3 Periodically save the computed error and plot the time to compute vs. computed error.

5 For Fun and Glory

5.1 (Grama 2.4) Extending this further, consider the problem of multiplying two dense matrices of dimension 4K x 4K. What is the peak achievable performance using a three-loop dot-product based formulation? (Assume that matrices are laid out in a row-major fashion.) (

```
/* matrix-matrix product loop */
for (i = 0; i < dim; i++)
  for (j = 0; i < dim; j++)
     for (k = 0; k < dim; k++)
       c[i][j] += a[i][k] * b[k][j];
```

5.2 **For Problem #3**, plot N (size of matrix) vs. Time. On your plot, record the N that you were able to compute in 10s. Competition: Compute the mat-mat product forhte largest matrix N in 10s, but proposing and implementing a different memory access pattern (e.g. process a chunk of a row/column at a time).

5.3 **For Problem #4**, plot the time to compute vs. computed error. Competition: computes the most accurate PI in the least amount of time using sequential and SIMD operations. **You can use other approximation strategies to calculate PI.**