CS 4230/6230
Programming Assignment 2 (Vectorization)
Due Monday, 11/18/2024, 11:59pm

For code development and testing for this assignment, use any computer with OpenMP, e.g., CADE Lab or your laptop. After code development, performance is to be reported on a 20-core node on the Lonepeak CHPC cluster (using batch submission). Note that you should not perform direct execution of programs on CHPC login nodes (account will get suspended if some threshold of total interactive CPU time is exceeded.
Submission: 1) A PDF report providing explanations as specified for each question, 2) Submission of the _opt.c files for questions 2, 3, 4, and 5 (optional, extra credit question).

For this assignment, use the clang C compiler with the following options to get feedback on which loops were vectorized and which could not be vectorized:
clang -O3 -fopenmp -Rpass-missed=loop-vectorize -Rpass=loop-vectorize

1. (25 points) Compile and execute the program **vec1_main.c**. It invokes functions **vec1a**, **vec1b**, and **vec1c**, which execute loops enclosing the following 3 statements: **vec1a**: w[i] = w[i]+1; **vec1b**: w[i] = w[i+1]+1; **vec1c**: w[i] = w[i-1]+1;

   Which of these would you expect to vectorize and why? Do your expectations match the vectorization report from clang? Is there a difference in achieved performance for the statements that are vectorized?

2. (25 points) The following loop code in **vec2_ref.c** is not vectorized by clang. Can it be transformed into functionally equivalent code (i.e., satisfies all data dependences) that can be vectorized by clang? Explain why or why not? If modification to enable vectorization is feasible, modify the code in **vec2_opt.c** (initially identical to the code in **vec2_ref.c)** to achieve higher (single thread) performance via vectorization.
```
for (j=0; j<n; j++)
 for(i=1; i<n; i++)
 //  A[i][j] = A[i-1][j]+1;
     A[i*n+j] = A[(i-1)*n+j]+1;
```
3. (25 points) The following loop code in **vec3_ref.c** is not vectorized by clang. Can it be transformed into functionally equivalent code (i.e., satisfies all data dependences) that can be vectorized by clang? Explain why or why not? If feasible, modify the code in **vec3_opt.c** (initially identical to the code in **vec3_ref.c)** to achieve higher (single thread) performance via vectorization.
```
for(i=1; i<n; i++)
  { w[i] = y[i]+1;
    y[i+1] = 2*x[i];}
```
4. (25 points) The following loop code in **vec4_ref.c** is not vectorized by clang. Can it be transformed into functionally equivalent code (i.e., satisfies all data dependences) that can be vectorized by clang? Explain why or why not? If feasible, modify the code in **vec4_opt.c** (initially identical to the code in **vec4_ref.c)** to achieve higher (single thread) performance via vectorization.
```
for(i=1; i<n; i++)
  { w[i+1] = y[i]+1;
    y[i+1] = x[i]+w[i];}
```
5. (25 points, **Extra credit**) The following code in **vec5_ref.c** is not vectorized by clang. Can it be transformed into functionally equivalent code (i.e., satisfies all data dependences) that can be vectorized by clang? If so, create equivalent code in **vec5_opt.c** to achieve higher (single thread) performance. You may use additional data declarations if desired; the only requirement is that sequential performance increases significantly (at least 2x), and the correctness check passes].
```
for (i=0; i<N; i++)
 {sum = 0.0;
  for(j=0; j<N; j++)
//  sum += A[j][i]*A[j][i];
    sum += A[j*N+i]*A[j*N+i];
  x[i] = sum;}
```