Final Take-Home Exam

Due: 11:59 PM, Friday, May 14, 2021
Submission: Compressed file into Canvas
Total score: 100

Question 1 (35 points): MergeSort using OpenMP

- Implement a serial mergesort program.
  - Verify the code with randomly generated numbers (n=10).

- Implement a parallel mergesort program using OpenMP.
  - This kind of recursive problems could be processed by OpenMP Sections or Task. E.g.,

```
#pragma omp parallel sections
{

   #pragma omp section
   {
      TODO // left recursion
   }

   #pragma omp section
   {
      TODO // right recursion
   }
}

or

#pragma omp parallel
{

   #pragma omp task
   TODO // left recursion

   #pragma omp task
   TODO // right recursion
}
```

  - Check the difference of OMP section and task from online tutorials (LLNL OpenMP Tuorial and others), and describe shortly.

  - Pick one of them that has a fit for this type of divide-and-conquer recursion problem. Which one did you pick and why? Describe shortly.

  - Make a first parallel program with your pick. Show the trace result with time for n=64 and p=2 to validate your program. Report the results.

- Test the parallel program with n=1E8 random numbers and 1, 2, 4, 8 threads. Compare wall-clock times using omp_get_wtime(). Do you see a good scalability? Report the times with the different threads.

- If you don't get a fast or scalable result using multi-threads, we need to optimize the parallel code. One solution is pruning (just work for specific condition using "if" clause". Create a new task only if the amount of work is sufficient. Then, test it the same problem (n=1E8) using 1, 2, 4, 8 threads.

- Optimize the parallel program as best to get a smallest wall-clock time.

Question 2 (35 points): Matrix multiplication using 1-D partitioning

Consider the problem of multiplying two square n-by-n matrices:

$$C \leftarrow A \times B, \quad A, B, C \in \Re^{n \times n}$$

Write the parallel matrix multiplication using 1-D partitioning.

- Consider a homogeneous parallel machine with the number of processors p a perfect (i.e., $\sqrt{p}$ is an integer). $P_{i,j}$ is the processor row i and column j where $0 \le i, j \le \sqrt{p} - 1$.
- We consider only the simple case where the matrix dimension n is a multiple of $\sqrt{p}$ (i.e., $n/\sqrt{p}$ is an integer).
- Implement the program using collective communications such as MPI_Scatter, MPI_Bcast, and MPI_Gather. For example, scatter A matrix and broadcast B matrix to the processes. Compute partial multiplication and gather partial C into C.
- Validate the program with $A_{i,j} = B_{i,j} = i \times n + j$, $n = 4$, and $p = 16$. Initialize the element of matrix value only at root (rank=0) process. Print the results and put it to the document file.
- Analyze the results with $p = 1, 4, 9, 16$ with n=1440 and 2880 after initializing $A_{i,j} = B_{i,j} = 1$ at root process. Measure the time, draw the speedup versus number of processors curve, and the CPU time versus the number of processors curve. Comment on what these curves mean from the point of view of scalability.

Question 3 (30 points)

- Implement a parallel vector dot product program using CUDA.
  - To declare shared memory use __shared__ keyword.
  - Device function should perform element multiplication.
  - Initialize the values of a and b with 1 or i (index number) like the lecture code.
  - Sum up all the multiplications could be done either in the device function or in the host.
  - __syncthreads() should be tested if you sum up in the device function.
  - You should implement the program using **both block and thread**. Then launch the dot() kernel with 10 block and 10 threads.
  - Test with N=100 and verify the results.

- Submit the ipython notebook that you tested on Google colab.

Important:

- The compressed file should have all source codes, shell files, and final.docx or final.pdf to contain all analyses and graphs.
- Your codes should be compliable and runnable. I will compile and run your programs. So, do not provide binary files.