Daniel Garon
Dr Sanjay Rajopadhye
CS 475
2023-04-21

PA5: MPI vs OpenMP Jacobi

Introduction

The provided program performs an iterative three-point stencil over the elements of a one-dimensional array of length *n*. The computation is the average of each point *j,* its predecessor *j-1* and its successor *j+1*. The values are read from and input array and the result is stored at the corresponding location in an output array. A while loop controls the repeated iterations over the array using a counter variable *t* and the specified number of iterations *m*. At the end of each iteration, the array pointers are swapped so that the previously computed values become the input for the next iteration, overwriting the previous iteration's inputs which are no longer necessary.

I have prepared two parallelized versions of the program, one using OpenMP and one using MPI.

The OpenMP implementation parallelizes the while loop using an *omp parallel* pragma. The nested for loop is parallelized using an *omp for* pragma. A *single* pragma ensures that only one thread will update the shared pointers upon completing the for loop.

The MPI program divides the main computations into segments of size *n/p* where *p* is the number of processes. The first iteration can be computed by each process because of the copied fully populated input array. After each iteration the only values available to each process are those that it computed in the previous iteration. Each successive iteration will shrink the segment which can be computed. One solution to this problem is for the processes to exchange messages with their neighboring processes containing the values needed by each. This message passing is relatively expensive in terms of time. By having the processes iterate over segments that overlap, we can reduce the frequency with which messages need be passed at the cost of redundant computation. A third argument to the program *k* represents the number of redundant computations performed by each process on each of its shared borders. This value also represents the size of the messages exchanged between processes.

One Processor / Multicore Comparison

I compared jacOMP and jacMPI on one CSU lab machine for 1 to 6 threads / processes, with the following parameter values: $n = 120{,}000$ $m = 12{,}000$ $k = 1$ (jacMPI only)

The sequential version of the program executed in 1.58 seconds.

Threads / Processes	jacOMP time (s)	jacMPI time (s)
1	1.593	1.583
2	0.823	0.830
3	0.581	0.565
4	0.431	0.440
5	0.364	0.361
6	0.315	0.345

Table 1: jacOMP and jacMPI execution times

Threads / Processes	jacOMP speedup	jacMPI speedup
1	0.99	1.00
2	1.92	1.90
3	2.72	2.80
4	3.67	3.59
5	4.34	4.37
6	5.01	4.58

Table 2: jacOMP and jacMPI speedups

The times and speedups recorded for the OMP and MPI versions were very consistent with each other on one machine with the same respective number of threads / processes.

I believe based on the similarity in execution times that on a single machine, there is likely to be a similar amount of overhead and optimization built into each of the parallelization libraries.

Multi-Processor / Multicore Comparison

I compared jacOMP and jacMPI on one CSU lab machine with 6 processes and again on two CSU lab machines with a total of 12 processes. I conducted repeated trials, each time doubling the size of the buffer, testing a range of sizes for *k* of [1-1024]. For both trials I used the following parameter values: $n = 480{,}000$ $m = 12{,}000$

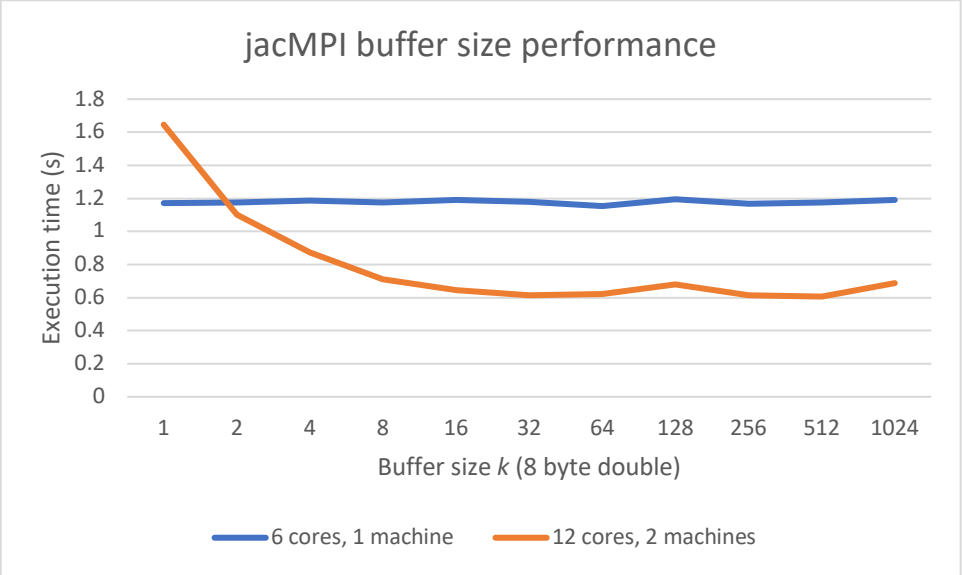

Figure 1: effect of buffer size on performance

I believe that this chart confirms my suspicion that on a single machine, the cost of communication between processes is relatively low and / or has been highly optimized.

This chart also shows that as the buffer size increases, performance improves, but only until such a point where the cost of the redundant computations and of passing larger and larger messages adds up to negate the benefit of reducing the frequency of communications. A buffer size of approximately 32 seemed to yield the best performance. The performance is relatively flat as the buffer size grows beyond that and seems to begin to degrade as the buffer size exceeds about 500.