CSE260 Assignment3 - Aliev-Panfilov Electrocardiac Simulation

Ho-Lun Wu
A53271935
hlwu@eng.ucsd.edu

Abstract

In this assignment, we are going to implement parallel Aliev-Panfilov Electrocardiac Simulation on the computer cluster. After dividing the whole matrix into sub-matrices evenly, applying SSE instruction and several optimizations, we can achieve 2016 Gflops when using 480 processors.

Section 1 - Performance Study

Table 1: Performance

	w/ communication					w/o communication				
proc	fused w/o SSE	fused w/ SSE	unfused w/o SSE	unfused w/ SSE	ref	fused w/o SSE	fused w/ SSE	unfused w/o SSE	unfused w/ SSE	ref
no MPI	N/A	N/A	N/A	N/A	N/A	0.4595	2.451	1.131	1.845	0.46
1	N/A	N/A	N/A	N/A	N/A	0.461	2.388	1.141	1.7250	0.46
2	0.909	4.822	2.245	3.612	0.907	0.911	4.868	2.258	3.681	0.911
4	1.809	9.277	4.449	7.117	1.811	1.821	9.726	4.543	7.338	1.820
8	3.586	17.88	8.743	13.43	3.602	3.658	19.41	9.082	14.720	3.639
24	115	200	97	165	115	117	206	99	171	118
48	226	387	192	323	226	235	415	198	343	235
96	427	712	366	605	427	461	798	391	668	464
192	527	1106	594	952	531	530	1167	604	987	540
384	1432	1800	1245	1686	1479	1493	1945	1264	1781	1556
480	1799	2016	1509	1896	1752	1965	2287	1650	2158	1895

Table 2: Scaling

proc	fused w/ SSE w/ communication	fused w/ SSE w/o communication
24-48	0.968	1.00
48-96	0.919	0.961
192-384	0.814	0.83
384-480	0.896	0.94

Table 3: Communication Overhead

proc	fused w/o SSE	fused w/ SSE	unfused w/o SSE	unfused w/ SSE	ref
2	0.24%	0.95%	0.58%	1.91%	0.36%
4	0.66%	4.84%	2.11%	3.11%	0.50%
8	2.01%	8.56%	3.88%	9.61%	1.03%
24	2.26%	3.25%	2.03%	3.57%	2.35%
48	3.94%	7.26%	3.45%	6.12%	3.94%
96	7.95%	11.99%	6.88%	10.41%	8.54%
192	0.57%	5.52%	1.70%	3.66%	1.79%
384	4.26%	8.06%	1.53%	5.63%	5.21%
480	9.23%	13.44%	9.34%	13.82%	8.16%

Figure 1: Performance

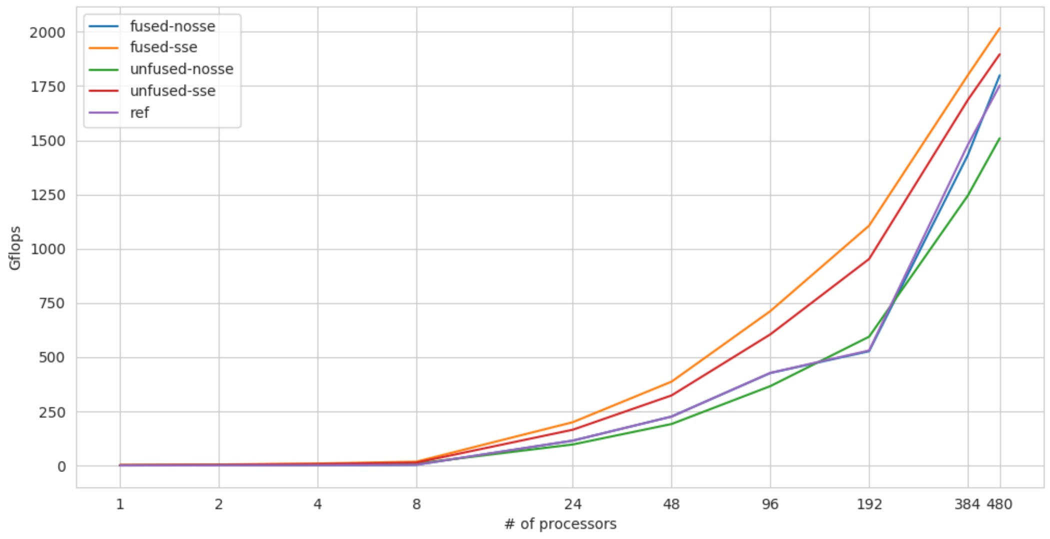


Figure 2: Communication Overhead

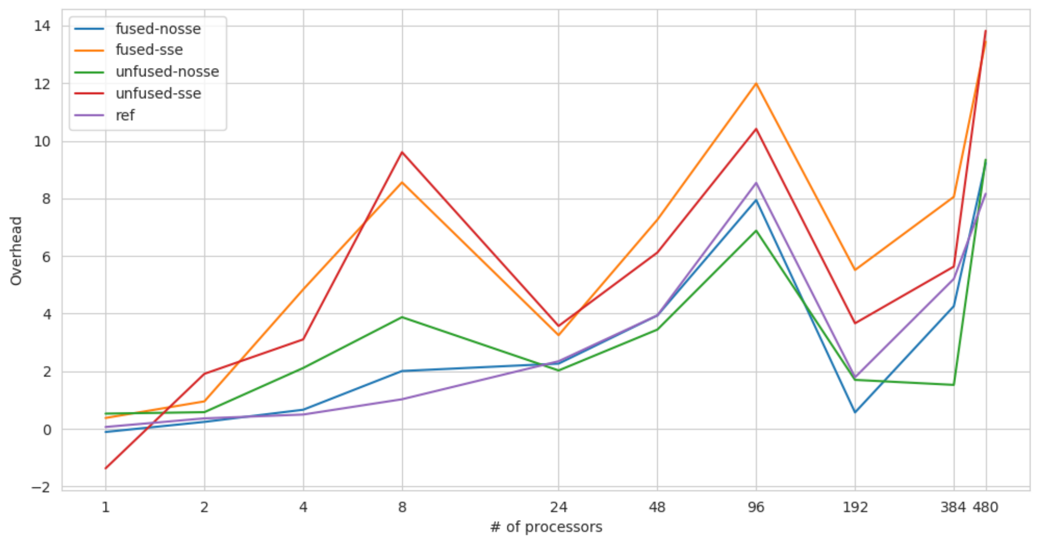


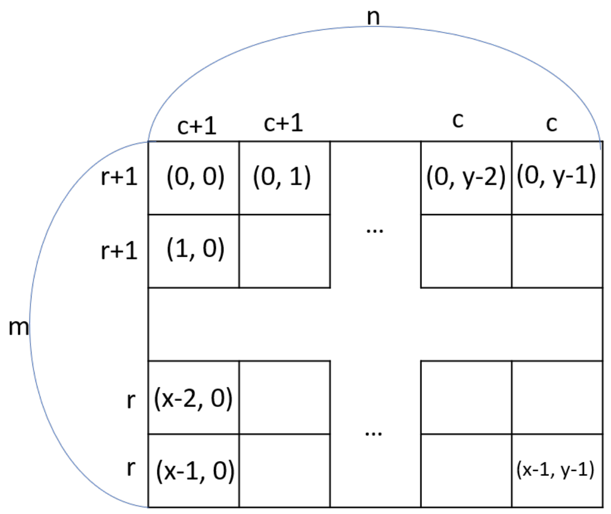
Table 1 is the performance of different numbers of processors and Table 3 is the overhead of the communication in different numbers of processors. For the small scale of evaluation (number of processors ≤ 8), we obtain the performance on the UCSD Band computing cluster. For the big scale of evaluation (number of processors ≥ 24), our program runs on the Comet supercomputer. Those GFlops value with communication is the best value among geometries $(1, N)$ and $(N, 1)$ and some geometries based on our assumption discussed in the later section. Moreover, we set $N = 400$ and $iter = 400$ when there is one processor, $N = 1800$ and $iter = 2000$ when the number of processors is between 2 and 8, and $N = 2000$ and $iter = 2000$ when number of processors is greater or equal than 24. Then, we choose the means value of the GFlops values among all possible geometries because the GFlops values of different geometries should be the same without communication overhead.

According to Figure 1 and Table 1, we can notice that kernels with SSE perform better than those without SSE and fused kernel with SSE has the best performance which achieves 6484 GFlops. According to Figure 2 and Table 3, although the trending of overhead to the number of processors is not stable due to the unstable network environment, we can still notice that the overhead is increasing along with the number of processors.

According to Table 2, we can find out that the values of strong scaling of those with communication are smaller than those without communication because the overhead of communication also increases along with the number of processors. However, we are not able to achieve strong scaling ≥ 1 in our best implementation because the more optimizations we apply in our program the more the communication overhead is. Also, the communication overhead is determined by the network environment so we might not be able to get the perfect value in busy environment such as Comet. Although we also calculate the scaling efficiency for the program without communication, the scaling values are not able to achieve 1.0 as well because we only disable the communication of ghost cells in our program instead of stopping all the communication between processors implemented by MPI.

Program Behavior

Figure 3: Blocking Strategy



Algorithm 1 Psuedo Code

```
1: procedure Parallel Aliev-Panfilov
2:     $rankid \leftarrow$ MPI Rank
3:     $(rank_x, rank_y) \leftarrow (rankid/y, rankid\%y)$
4:     $r, c \leftarrow m/x, n/y$
5:     if $rank_x < m - r * x$ then
6:         $r \leftarrow r + 1$
7:     end if
8:     if $rank_y < n - c * y$ then
9:         $c \leftarrow c + 1$
10:    end if
11:    for $i = 0 \rightarrow iter$ do
12:        Commute
13:        Kernel
14:    end for
15: end procedure
16: procedure Commute
17:    for $n \in \{Top, Bottom, Right, Left\}$ do
18:        if $n$ exists then
19:            $MPI\_Irecv$ ghost cells to $n$
20:            $MPI\_Isend$ ghost cells to $n$
21:        end if
22:    end for
23:    $MPI\_Waitall$
24:    update the matrix
25: end procedure
```

In the beginning of the program, we need to divide the entire matrix into several sub-matrices according to Figure 3. Assume the matrix has m rows and n columns, the sub-matrices has r or $r+1$ rows and c or $c+1$ columns where $r = \lfloor \frac{m}{x} \rfloor$ and $c = \lfloor \frac{n}{y} \rfloor$ depending on the indexes of the sub-matrices $(rank_x, rank_y)$. For those $rank_x < m - r * x$ and $rank_y < n - c * y$, the sub-matrix has $r+1$ rows and $c+1$ columns respectively;

otherwise, the sub-matrix has r rows and c columns respectively. By doing so, we divide the whole matrix into several sub-matrices evenly to make each core do almost the same computing effort. After dividing the matrix, we fill the all sub-matrices by the original approach. We calculate the global index of each entry in the sub-matrix to the whole matrix and fill 0 or 1 into the entry based on the original conditions.

Secondly, we need to handle the ghost cells exchanges between the sub-matrices. To exchange the cells, we use asynchronized functions MPI_Isend and MPI_Irecv to send and receive the cells. Because these two functions are asynchronized, we use the function $MPI\_Waitall$ to wait until all communications finish in each sub-matrix. Instead of waiting for all sub-matrices to finish their communication, each sub-matrix only needs to wait for its own communication. By doing so, we can reduce the overhead of synchronization and each core can do its future work in advance.

After finishing the communication between sub-matrices, we start to optimize the computation kernel of this program. There are two versions of the kernel, including fused and unfused. Because there are two matrices used in the computation kernel, the unfused kernel calculates the two matrices separately, that is, calculate in two for loops, to achieve better cache locality. On the other hand, the fused kernel calculates the two matrices in a single for loop. To further improve the performance of the computation kernel, we apply SSE instructions to vectorize the for loop so that we are able to calculate two entries in one instruction.

Section 2 - Analysis

According to Figure 1, we can find out that MPI has little overhead because the performance between no MPI and using MPI with one processor is almost the same.

When using 8 processors, the overhead is larger than the overhead when using only 2 or 4 processors. because there are more bytes need to communicate in each iteration.

When using 24 processors on Comet, the overhead is smaller than using 8 processors on Bang. We think the reason is that Comet runs faster than Bang and the network environment is also better than Bang so the data can be communicated faster.

Section 3 - Development Process

Dividing Sub-matrices

In the beginning, we didn't divide the matrix correctly so the sub-matrices in the last row and the last column have much large size than the others. After the investigation, we figure out these bigger sub-matrices finish their computation much later than the smaller sub-matrices. To average the computation effort, we re-design the approach to divide the matrix. We make sure that no sub-matrix has more than 1 row or column more work than any other sub-matrix.

Sub-matrices Initialization

In the beginning, we initialize the whole matrix in the 0_{th} processor and distribute the sub-matrices to the other processors by MPI communication. However, we find out that it takes lots of time to initialize. To improve the performance, we re-implement the initialization to make each processor initialize its sub-matrix according to the original rules.

SSE Instruction

To improve the performance, we apply SSE instruction to speed up the kernel. Because there are some unaligned entries, we can only use unaligned load to load values from memory into SSE registers. In order to use aligned load instruction to obtain better performance, we pad the sub-matrix to make sure that each entry are aligned. However, we find out that the plotting function and the statistic function provided by starter code don't deal with the padding.

Section 4 - Determine geometries

Figure 4: N = 96

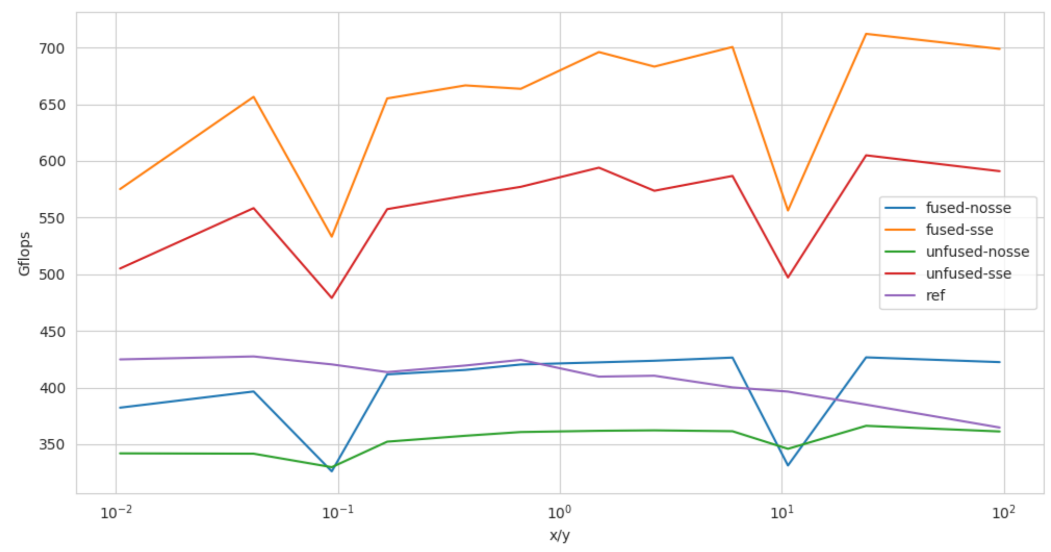
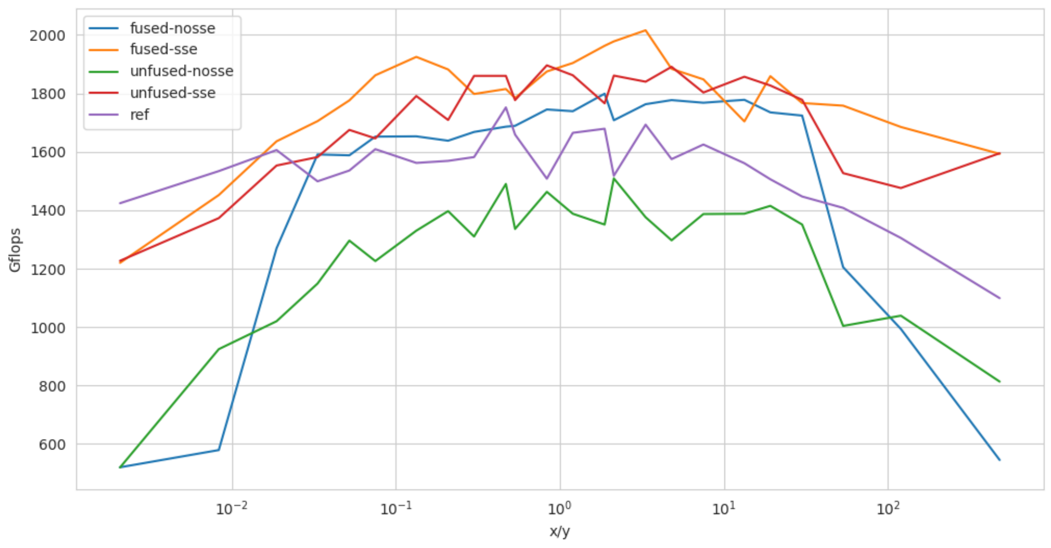


Figure 5: N = 480



When evaluating the performance, we notice that the performance of different geometries are varying because the numbers of bytes of communication are also varying among different geometries. To choose the best geometries, we can calculate the number of bytes in each communicate by the following formula.

$$bytes = n * (x - 1) + n * (y - 1) = n * (x + y) - 2 * n + 1$$

We can figure out that n is constant so the number of bytes is determined by the value of $x+y$. Due to that, our goal is to find $min_{x*y=n}(x+y)$. After calculating, we can derive that $min_{x*y=n}(x+y) = 2*\sqrt{n}$, that is, $x = y = \sqrt{n}$. So, we will choose the geometry whose ratio of x to y is closest to 1. Also, communication with top and bottom sub-matrices is faster than communication with left and right sub-matrices because the top and bottom ghost cells are located in contiguous memory address which the address can be passed into MPI_Irecv directly; however, left and right ghost cells are not located in contiguous memory so we need to receive the data into an extra memory space and then copy into the ghost cells.

According to Figure 4 and Figure 5, our assumption fit the evaluation. When $x/y \sim 0$ and $x > y$, the performance is the best.

Section 6 - Future Work

Because there are lost of people using Comet, the communication between processors is unstable. Although the queue system allows us to allocate processors, we are not able to allocate a dedicated network for our computation. If we are able to obtain a stable environment, we can get better and stabler performance.