Date

PROJECT 1

I Serial Solver with constant u, v and periodic boundaries

i Brief description of algorithm, implementation

First, we prepare for the delta_x and delta_t.

Then, check if parameters meet the Courant stability condition, and we need to initialize Cn with a 2D Gaussian pulse initial condition. I used two nested for loops to implement it.

For periodic boundary conditions, I expand the C_n array from N*N to (N+2)*(N+2). I let the column 0 be equal to column N and column N+1 be equal to column 1, which is same for row 0 and row N+1.

Then implement the Lax method in the three nested for loops.

Use `std::chrono::steady_clock::now()` to calculate the running time.

ii Demonstration of correctness

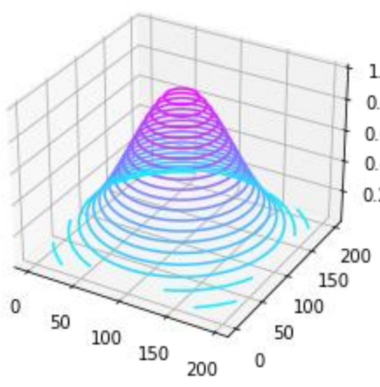


Figure 1: initialized Gaussian distribution

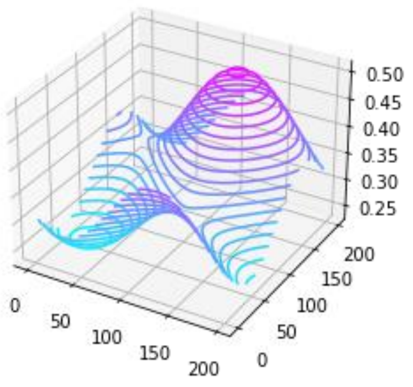


Figure 2: half way

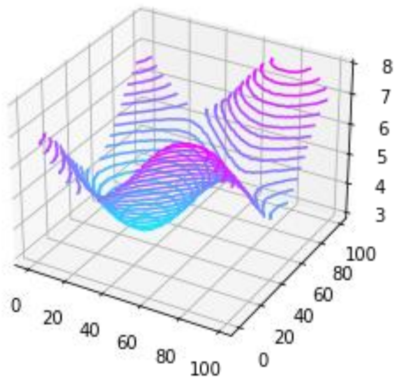


Figure3 Final result

According to the three figures above, we can see that it clearly show the development of the advection process over time.

iii Performance analysis

With the following parameter:

- N = 400 (Matrix Dimension)
- NT = 20000 (Number of timesteps)
- L = 1.0 (Physical Cartesian Domain Length)
- T = 1.0e6 (Total Physical Timespan)
- u = 5.0e-7 (X velocity Scalar)
- v = 2.85e-7 (Y velocity Scalar)

The program runs in 3s with '-O3' optimization.

The estimation of amount of memory required: 8 byte for double, there are 400*400 *2 * 8 bytes = 2560KB.

II OpenMP parallel solver

i Brief description of implementation

Used omp_get_wtime() to calculate running time.

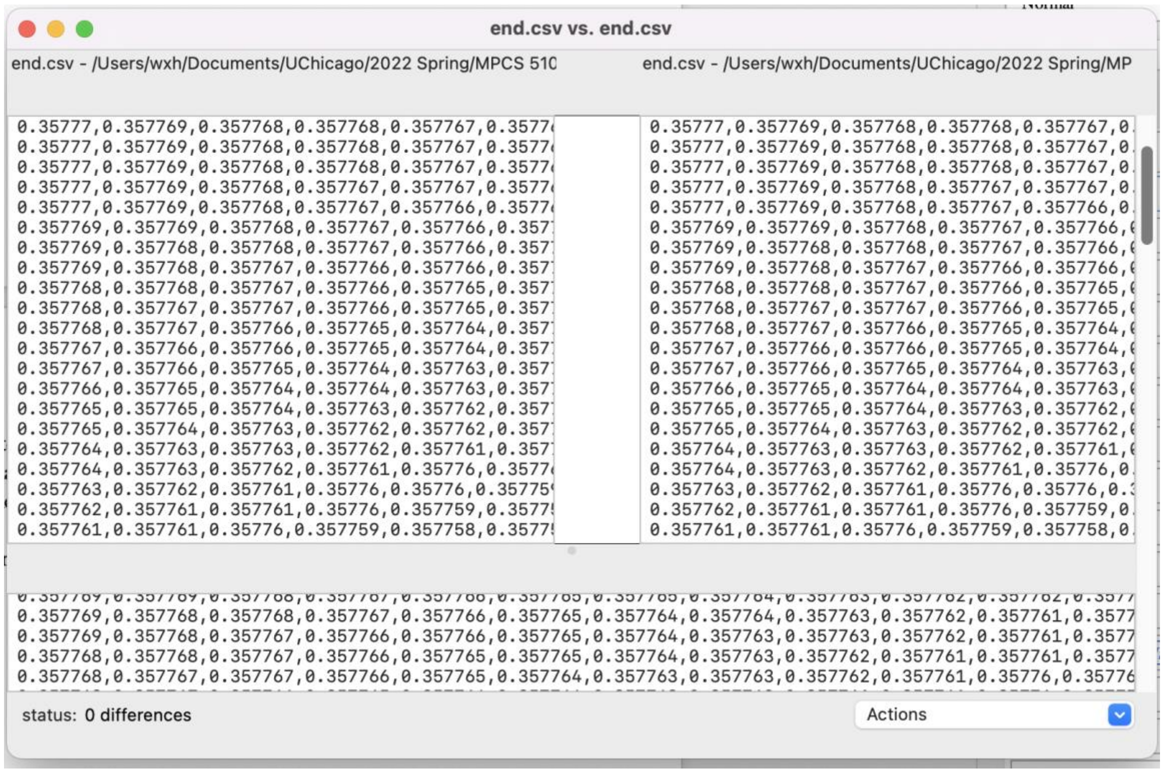
Used omp_set_num_threads(2) to set threads num.

For the for loop of initialization, used:
#pragma omp parallel for default(none) shared(N, L, delta_x, C_n)

For the main for loop, used:
#pragma omp parallel for default(none) shared(N, C_n, C_n1, delta_t, delta_x, u, v, delta) schedule(static)

ii Demonstration of correctness

I used the FileMerge to compare the results of parallel and serial versions, which shows that 0 differences.


end.csv vs. end.csv

end.csv - /Users/wxh/Documents/UChicago/2022 Spring/MPCS 510

```
0.35777,0.357769,0.357768,0.357768,0.357767,0.3577
0.35777,0.357769,0.357768,0.357768,0.357767,0.3577
0.35777,0.357769,0.357768,0.357768,0.357767,0.3577
0.35777,0.357769,0.357768,0.357767,0.357767,0.3577
0.35777,0.357769,0.357768,0.357767,0.357766,0.3577
0.357769,0.357769,0.357768,0.357767,0.357766,0.357
0.357769,0.357768,0.357768,0.357767,0.357766,0.357
0.357769,0.357768,0.357767,0.357766,0.357766,0.357
0.357768,0.357768,0.357767,0.357766,0.357765,0.357
0.357768,0.357767,0.357767,0.357766,0.357765,0.357
0.357768,0.357767,0.357766,0.357765,0.357764,0.357
0.357767,0.357766,0.357766,0.357765,0.357764,0.357
0.357767,0.357766,0.357765,0.357764,0.357763,0.357
0.357766,0.357765,0.357764,0.357764,0.357763,0.357
0.357765,0.357765,0.357764,0.357763,0.357762,0.357
0.357765,0.357764,0.357763,0.357762,0.357762,0.357
0.357764,0.357763,0.357763,0.357762,0.357761,0.357
0.357764,0.357763,0.357762,0.357761,0.35776,0.3577
0.357763,0.357762,0.357761,0.35776,0.35776,0.35775
0.357762,0.357761,0.357761,0.35776,0.357759,0.3577
0.357761,0.357761,0.35776,0.357759,0.357758,0.3577
```

end.csv - /Users/wxh/Documents/UChicago/2022 Spring/MP

```
0.35777,0.357769,0.357768,0.357768,0.357767,0.
0.35777,0.357769,0.357768,0.357768,0.357767,0.
0.35777,0.357769,0.357768,0.357768,0.357767,0.
0.35777,0.357769,0.357768,0.357767,0.357767,0.
0.35777,0.357769,0.357768,0.357767,0.357766,0.
0.357769,0.357769,0.357768,0.357767,0.357766,
0.357769,0.357768,0.357768,0.357767,0.357766,
0.357769,0.357768,0.357767,0.357766,0.357766,
0.357768,0.357768,0.357767,0.357766,0.357765,
0.357768,0.357767,0.357767,0.357766,0.357765,
0.357768,0.357767,0.357766,0.357765,0.357764,
0.357767,0.357766,0.357766,0.357765,0.357764,
0.357767,0.357766,0.357765,0.357764,0.357763,
0.357766,0.357765,0.357764,0.357764,0.357763,
0.357765,0.357765,0.357764,0.357763,0.357762,
0.357765,0.357764,0.357763,0.357762,0.357762,
0.357764,0.357763,0.357763,0.357762,0.357761,
0.357764,0.357763,0.357762,0.357761,0.35776,0.
0.357763,0.357762,0.357761,0.35776,0.35776,0.
0.357762,0.357761,0.357761,0.35776,0.357759,0.
0.357761,0.357761,0.35776,0.357759,0.357758,0.
```

```
0.357769,0.357768,0.357768,0.357767,0.357766,0.357765,0.357764,0.357764,0.357763,0.357762,0.357761,0.3577
0.357769,0.357768,0.357767,0.357766,0.357766,0.357765,0.357764,0.357763,0.357763,0.357762,0.357761,0.3577
0.357768,0.357768,0.357767,0.357766,0.357765,0.357765,0.357764,0.357763,0.357762,0.357761,0.357761,0.3577
0.357768,0.357767,0.357767,0.357766,0.357765,0.357764,0.357763,0.357763,0.357762,0.357761,0.35776,0.35776
```

status: 0 differences

Figure 3 comparation with serial solver

iii Performance analysis

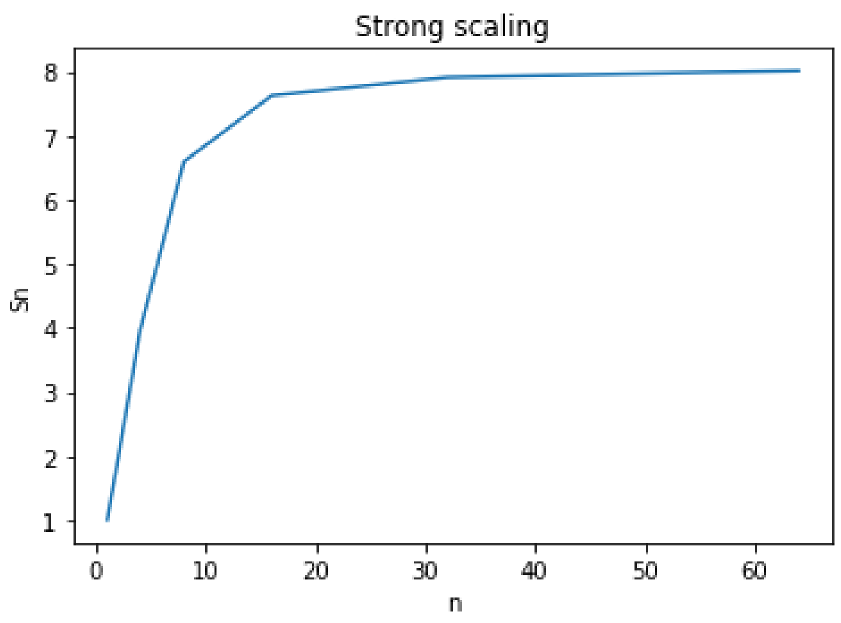
(a) Strong scaling

Runs on midway3

Compiler: g++ with "-O3" optimization

Parameter: N=3200   NT=400   L = 1.0   T = 1.0e3   u = 5.0e-7   v = 2.85e-7

Cores	Time(s)	Sn
1	6.33	1
2	3.24	1.95
4	1.60	3.96
8	0.96	6.59
16	0.83	7.63
32	0.80	7.91
64	0.79	8.01



From the plot above, we can see at this scale, the improvement of the performance decreased as the cores increased. This is because as the number of cores increased, it will spend more time on communication and, etc. between threads.

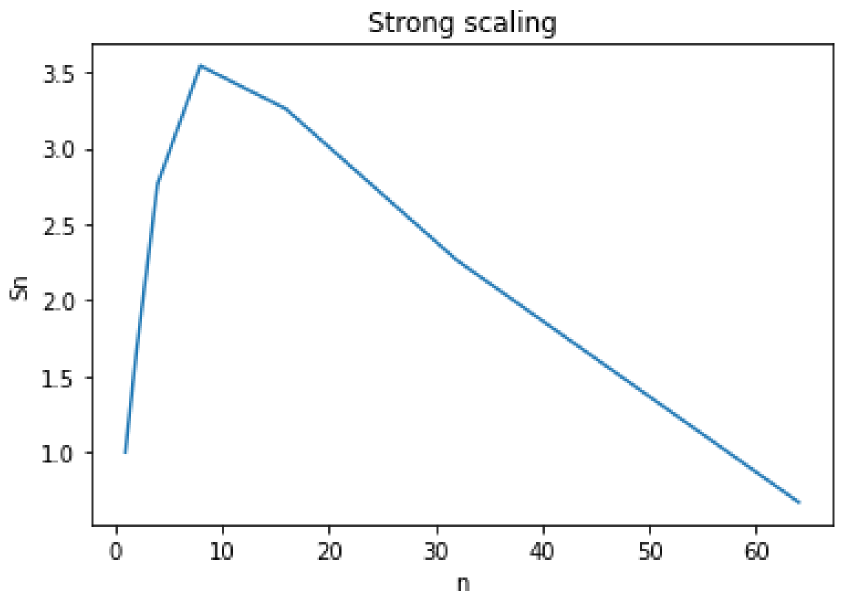
(b) Second strong scaling

Runs on midway3

Compiler: g++ with "-O3" optimization

Parameter: N=2200 NT=400 L = 1.0 T = 1.0e3 u = 5.0e-7 v = 2.85e-7

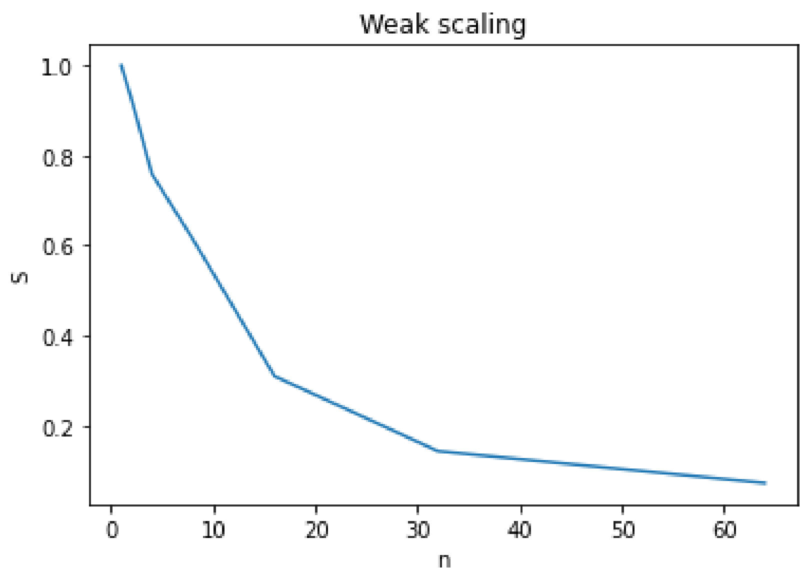
Cores	Time(s)	Sn
1	0.0163	1
2	0.0099	1.65
4	0.0059	2.76
8	0.0046	3.54
16	0.0050	3.26
32	0.0072	2.26
64	0.0243	0.67



From the plot above, we can see at this scale, the performance increased at the beginning then decreased as the number of cores increased. This is mainly because this is a small calculation, and as the cores increased the proportion of the time that spent on the communication between cores, and etc. increased. And it will eventually exceed the time that save by parallel computing.

(c) Weak scaling

Cores	Gridpoints	N	Time(s)
1	640,00	800	0.258
2	1,280,000	1131	0.279
4	2,560,000	1600	0.340
8	5,120,000	2262	0.420
16	10,240,000	3200	0.833
32	20,480,000	4525	1.802
64	40,960,000	6400	3.563



The above plot shows that the even if the ratio of total computation to the number of cores remains constant, running time doesn't stay the same.

III Hybrid MPI/OpenMP solver

i Description of parallel strategy. Note that following requirements

Used MPI's topology function *MPI_Cart_create* to create a 2d virtual topology.

Then used *MPI_Cart_shift* to determine 2D neighbor ranks for this MPI rank.

Write a *alloc_2d_double* that can allocate a 2d array in continuous memory. Let *C* and *C_new* be the decomposed local variable for each rank.

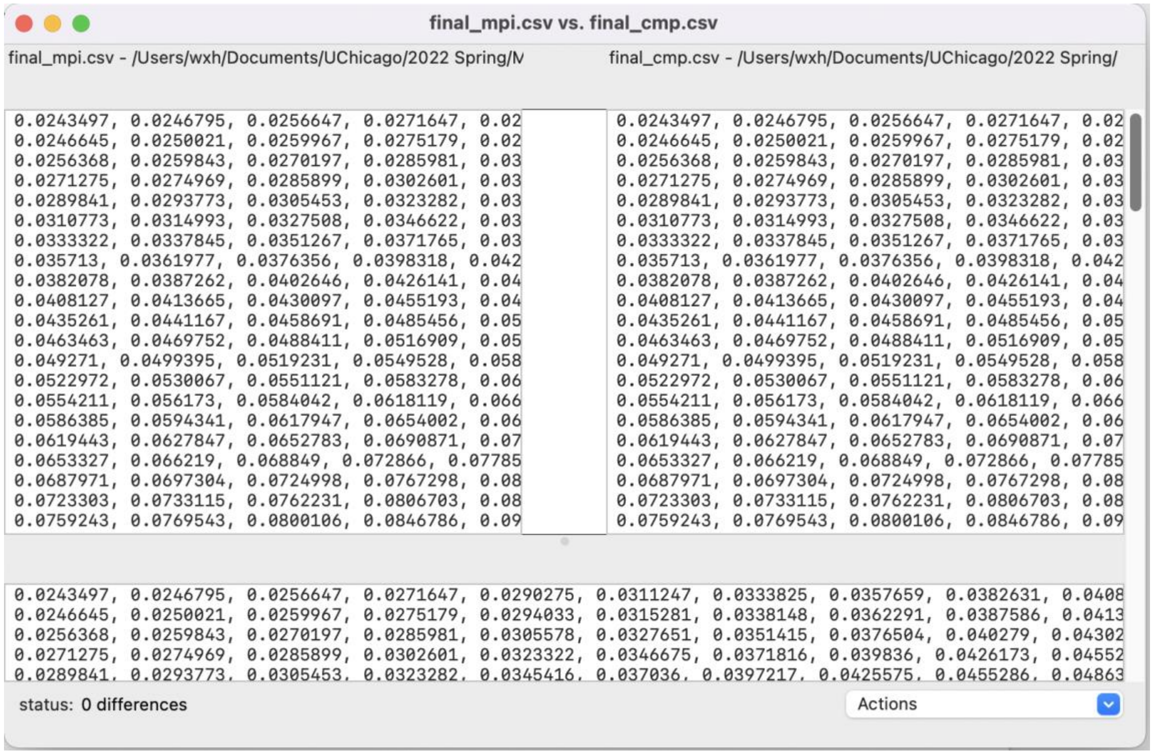
Initialize *C* in each rank, for each *x* and *y*, use *i + N_l * coords[0]* and *j + N_l * coords[1]* to calculate.

In the main loop, first allocate memory for 4 arrays, *right_ghost_cells*, *left_ghost_cells*, *up_ghost_cells*, *down_ghost_ cells* to receive the data from neighbor. For the left and right, I made two buffers called *left_sending_buffer* and *right_sending_buffer,* because the memory is not continuous for these data. For up and down, I just use *C[0]* and *C[N_l-1]*. I used *MPI_Sendrecv* to implement the sending and receiving function.

In the end, I let rank 0 to collect the data from other ranks and then add rank 0's own data.

ii Demonstration of correctness

I used the FileMerge to compare the results of it and OPENMP versions, which shows that 0 differences.



final_mpi.csv vs. final_cmp.csv

final_mpi.csv - /Users/wxh/Documents/UChicago/2022 Spring/M

```
0.0243497, 0.0246795, 0.0256647, 0.0271647, 0.02
0.0246645, 0.0250021, 0.0259967, 0.0275179, 0.02
0.0256368, 0.0259843, 0.0270197, 0.0285981, 0.03
0.0271275, 0.0274969, 0.0285899, 0.0302601, 0.03
0.0289841, 0.0293773, 0.0305453, 0.0323282, 0.03
0.0310773, 0.0314993, 0.0327508, 0.0346622, 0.03
0.0333322, 0.0337845, 0.0351267, 0.0371765, 0.03
0.035713, 0.0361977, 0.0376356, 0.0398318, 0.042
0.0382078, 0.0387262, 0.0402646, 0.0426141, 0.04
0.0408127, 0.0413665, 0.0430097, 0.0455193, 0.04
0.0435261, 0.0441167, 0.0458691, 0.0485456, 0.05
0.0463463, 0.0469752, 0.0488411, 0.0516909, 0.05
0.049271, 0.0499395, 0.0519231, 0.0549528, 0.058
0.0522972, 0.0530067, 0.0551121, 0.0583278, 0.06
0.0554211, 0.056173, 0.0584042, 0.0618119, 0.066
0.0586385, 0.0594341, 0.0617947, 0.0654002, 0.06
0.0619443, 0.0627847, 0.0652783, 0.0690871, 0.07
0.0653327, 0.066219, 0.068849, 0.072866, 0.07785
0.0687971, 0.0697304, 0.0724998, 0.0767298, 0.08
0.0723303, 0.0733115, 0.0762231, 0.0806703, 0.08
0.0759243, 0.0769543, 0.0800106, 0.0846786, 0.09
```

final_cmp.csv - /Users/wxh/Documents/UChicago/2022 Spring/

```
0.0243497, 0.0246795, 0.0256647, 0.0271647, 0.02
0.0246645, 0.0250021, 0.0259967, 0.0275179, 0.02
0.0256368, 0.0259843, 0.0270197, 0.0285981, 0.03
0.0271275, 0.0274969, 0.0285899, 0.0302601, 0.03
0.0289841, 0.0293773, 0.0305453, 0.0323282, 0.03
0.0310773, 0.0314993, 0.0327508, 0.0346622, 0.03
0.0333322, 0.0337845, 0.0351267, 0.0371765, 0.03
0.035713, 0.0361977, 0.0376356, 0.0398318, 0.042
0.0382078, 0.0387262, 0.0402646, 0.0426141, 0.04
0.0408127, 0.0413665, 0.0430097, 0.0455193, 0.04
0.0435261, 0.0441167, 0.0458691, 0.0485456, 0.05
0.0463463, 0.0469752, 0.0488411, 0.0516909, 0.05
0.049271, 0.0499395, 0.0519231, 0.0549528, 0.058
0.0522972, 0.0530067, 0.0551121, 0.0583278, 0.06
0.0554211, 0.056173, 0.0584042, 0.0618119, 0.066
0.0586385, 0.0594341, 0.0617947, 0.0654002, 0.06
0.0619443, 0.0627847, 0.0652783, 0.0690871, 0.07
0.0653327, 0.066219, 0.068849, 0.072866, 0.07785
0.0687971, 0.0697304, 0.0724998, 0.0767298, 0.08
0.0723303, 0.0733115, 0.0762231, 0.0806703, 0.08
0.0759243, 0.0769543, 0.0800106, 0.0846786, 0.09
```

```
0.0243497, 0.0246795, 0.0256647, 0.0271647, 0.0290275, 0.0311247, 0.0333825, 0.0357659, 0.0382631, 0.0408
0.0246645, 0.0250021, 0.0259967, 0.0275179, 0.0294033, 0.0315281, 0.0338148, 0.0362291, 0.0387586, 0.0413
0.0256368, 0.0259843, 0.0270197, 0.0285981, 0.0305578, 0.0327651, 0.0351415, 0.0376504, 0.040279, 0.04302
0.0271275, 0.0274969, 0.0285899, 0.0302601, 0.0323322, 0.0346675, 0.0371816, 0.039836, 0.0426173, 0.04552
0.0289841, 0.0293773, 0.0305453, 0.0323282, 0.0345416, 0.037036, 0.0397217, 0.0425575, 0.0455286, 0.04863
```

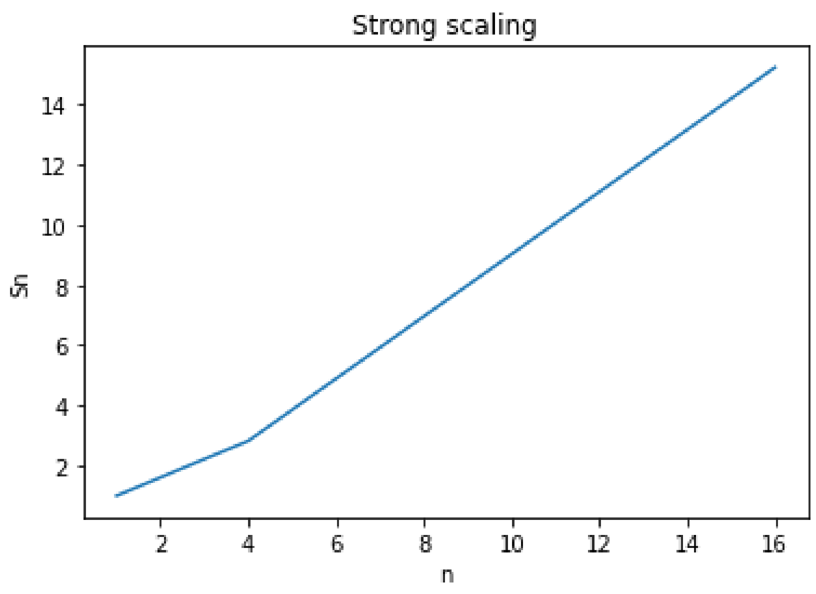
status: 0 differences

iii Performance analysis

(a) Strong scaling for

N = 10000, NT = 400, number of OpenMP threads per rank = 48

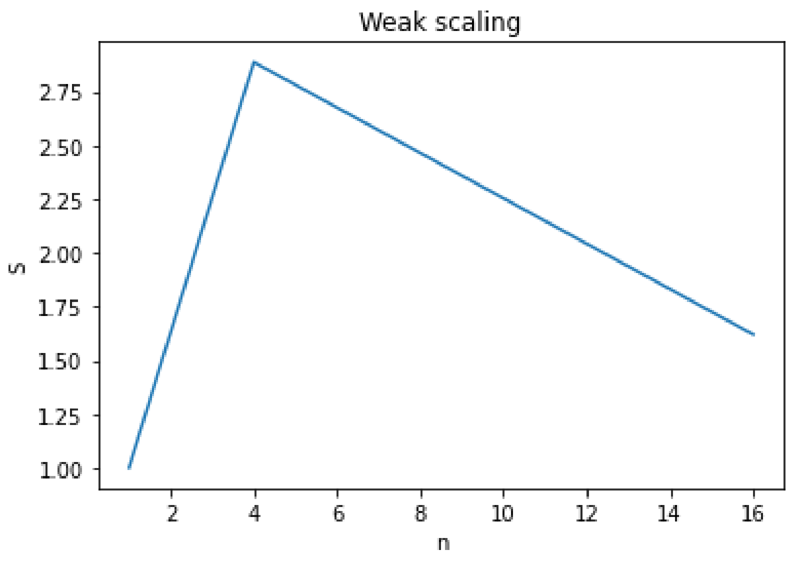
Nodes	Time(s)	Sn
1	27.88	1
4	9.87	2.82
16	1.84	15.15



It shows that under the current computing scale, the computing efficiency has been improved as the number of nodes increased.

(b) Weak scaling:

Nodes	N	Time(s)
1	10000	27.88
4	20000	9.66
16	40000	17.21



This is strange but when I tried many times, it still looks like this.

(c) Without openmp

Nodes	Time(s)	Sn
1	80.36	1

4	23.366	
16	6.46	