CS5220 Project 2: Shallow Water Equations

Nick Cebry, Jiahao Li, Max Ruth

October 5, 2020

1 Introduction

In this project, we worked with the a finite volume solver for the shallow water equations. These PDEs model the movement of waves in scenarios where the waves are long compared to the depth of the water. We were provided with a moderately performant single threaded version of the code, and made an effort to improve performance by parallelizing the algorithm. For our implementation, we chose to use MPI as our parallelization paradigm. Our algorithm divides the grid up into a series of subdomains, and assigns one subdomain to each processor. Processors are responsible for calculating the behavior of the water in their portion of the domain. They communicate information about the cells on the edge of their domain with the processors of neighboring domains so that information propagates properly across domain boundaries.

The aims for this project are: 1) using MPI as parallelization implementation for the single-core code; 2) conducting weak and strong scaling studies on the shallow water problem to test the robustness of our implementation; 3) profiling and tuning the codes for higher-level optimization based on computational resources.

2 The Algorithm

For the purposes of this algorithm, we are considering two consecutive calls to `central2d_step` to be "one time step," in order to alleviate any confusion with the shifting grids in Jiang and Tadmor.

To address the problem of parallelizing the method of Jiang and Tadmor, we use a domain decomposition method like the one introduced for Conway's Game of Life. For the problem, we assume that the domain $\Omega = (0, L)^2$ is divided into square cells, with a resolution of M cells in each direction. Because the problem is periodic, this amounts to M "points" at which the height h, x-velocity u, and y-velocity v are known. To parallelize the code, we divide the domain into an $N_x \times N_y$ grid of subdomains, where each subdomain is owned by a process. By this construction, we are using a total of $N_x N_y$ processors.

As a part of this construction, each subdomain is responsible for knowing the value of $m_x \times m_y$ points *at all times* where $m_x = M/N_x$ and $m_y = M/N_y$. However, due to the way information flows through a hyperbolic system of equations, we also require each processor to store a "halo" of ghost cells with width m_g. Each processor is not tasked with knowing the value of the ghost cells, but rather retrieves the values of the ghost cells from its neighbors. The width of the halo is related to how many time steps each processor can perform independently before information must be exchanged – called m_t – by the relation $m_g = 4m_t$. Independent of the number of processors (assuming M is fixed), the we will call the total number of time step blocks N_t, so that the total number of time steps is $m_t N_t$.

We must also keep track of the maximum speed of information across all of the processors, so that each processor takes the same time step. In the code, we are currently sharing this information *at every time step*, but this turns out not to be a large burden for the sizes of problems we are considering.

In total, the algorithm can be summarized by Alg. 1. Inside of the main loop, we first share the ghost cells. This goes in the order of pass-left and receive-right; pass-right and receive-left; pass-up and receive-down; pass-down and receive-up. In the left/right passing of ghost cells, we pass a block of size $m_g m_y$, whereas for up/down we pass a block of size $m_g(m_x + 2m_g)$, as we need to communicate the corners of the halo along with the sides in this step.

Algorithm 1 Parallel Jiang-Tadmor Main Loop

```
for $i \leftarrow 1$ to $N_t$ do
    Share ghost cells via 4 calls of MPI_Sendrecv
    for $j \leftarrow 1$ to $m_t$ do
        Get local time step using speed
        Get global time step via MPI_Allreduce
        Perform time step
    end for
end for
```

2.1 Model Speed Up

The general process:

1. Used MPI
2. Transfer Boundary information to left and right neighbors, the top and bottom (transferring the corners to top and bottom)
3. Transfer boundary information every τ steps
4. Transfer time step information every τ steps as well (This is what is being used in the model anyway)

The model

1. Four different contributions to batches of time steps
2. Give predictions for how strong scaling, weak scaling, and optimal time batching look like.

3 Scaling and Profiling Results

3.1 Scaling Studies

We performed both strong and weak scaling studies to analyze the performance of our algorithm. For the strong scaling study, we used the dam break scenario, and fixed the resolution of the grid cells at 1000 cells in both the X and Y dimensions. We then varied the number of processors used to compute the simulation up until a fixed simulation time. The size of the sub-domains varied with the number of processors in order to keep the problem size constant. We compare the wall clock time required to solve a fixed problem as the amount of compute resources available varies. For the weak scaling study, we again used the dam break scenario, but varied the resolution of the grid cells with the number of processors used. The resolution of the grid was varied such that each subdomain was always 300 cells on each side. The simulation was run until a fixed amount of time had been simulated. In order to account for the different CFL conditions with different grid resolutions, we recorded the total number of time steps processed, and divided the wall clock time by this number to determine the average amount of time required to compute a single tick of the simulation. We compare the wall clock time required to compute a single simulation tick as both the size of the problem and the computational resources available scale.

3.2 Profiling and Tuning

We used Tau package on Comet as the profiler tool to study the performance of our parallelization and higher-level optimization of the original codes. We used compiler-based instrumentation method, which records timing for both the MPI routines and user-defined functions in the codes. By changing the compiler from `mpicc` to `tau_cc.sh` and adding `-tau_options=-optCompInst` to the argument for compiling, running the program returns profiling results for each processor and we can look them up by `pprof` command. The profiling result shows the proportion of time spent, exclusive time, the total inclusive time, number of calls, number of subroutines as well as the average inclusive time per call for each of the functions in the program.

As the parallel computing on each individual processor, except for the rank 0 core where we expect to see different behavior due to all the subdomain data gathering to output solution after certain time steps for a rationale check, mostly the mean times spent on all the cores are compared straightforward.

We first did a direct profiling comparison between a 200×200 grid and a 1000×1000 grid, where in both cases 9 processors were called based on Comet terminal environment. Since the two are computed using the same number of cores, the main difference is the size of subdomains divided into each core, which also indicates blocking size affects the performance of codes. For a small grid problem, the `MPI_Init()` used 35.5% of the total profiling time; the major of the code `central2d_xrun()` consists of three parts, `central2d_step()`, `central2d_periodic()` and a MPI routine `MPI_Allreduce()`, taking 53.0%, 7.0% and 0.0%, respectively. `central2d_periodic()` is the modified code for inter-core communication of ghost cell; `MPI_Allreduce()` is used for synchronizing each subdomain at the same physical time. Numerical PDE computing as well as core communication does not take up full. When using same amount of computational resources, the situation of the large scale becomes different, where `central2d_step()`, `central2d_periodic()` and `MPI_Allreduce()` take up 95.1%, 3.3% and 0.0% of the total time; `MPI_Init()` is only 0.1%. This indicates the bottleneck in a large scale problem running on 9 processors is positioned on how to make the PDE computing itself faster. The most time is spent in the actual PDE numerical computing, the cost of core communication is pretty small. The profiling tables are attached in the end of the report.

This paragraph talking on the blocking strategies: using more cores, reduce size, cache; use 81 cores profiling

This paragraph talking on advancing the timesteps: less communication, but more ghost cells, trade-off; use 81 core profiling with 2/3 timesteps

4 Conclusion

What we would add for next time:

1. More careful cache performance
2. Tuning number of ghost cells to block size
3. Different sized domains/initial conditions
4. Deal with the fact that not all processors are the same
5. Think about the tradeoff between a conservative time step and communicating every step (how uneven are the time steps really?)
6. Communication between processors at different nodes is different than that of the same nodes?

A Profiling table: small scale on 9 cores

```
FUNCTION SUMMARY (mean):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            6       16,162           1           1   16162524 .TAU application
100.0        0.954       16,156           1           3   16156274 main
 63.8        0.596       10,313           1     119.444   10313260 run_sim
 60.0       0.0372        9,703          50          50     194062 central2d_run
 60.0        0.538        9,703          50        1250     194061 central2d_xrun
 53.0            7        8,559         500     37833.3      17118 central2d_step
 35.5        5,737        5,737           1           0    5737190 MPI_Init()
 26.7           49        4,322         500      218000       8644 central2d_predict
 26.0        4,175        4,205      213500     50014.3         20 limited_deriv1
 26.0           68        4,200         500      309001       8401 central2d_correct
 25.9        4,150        4,179      213500     49986.7         20 limited_derivk
  7.0            1        1,135         250        7000       4542 central2d_periodic
  7.0        1,131        1,131        1000           0       1131 MPI_Sendrecv()
  3.0       0.0558          482          51     96.3333       9452 gather_sol
  2.5        0.037          398     45.3333     45.3333       8787 send_full_u
  2.5          398          398     45.3333           0       8787 MPI_Send()
  0.6          104          104           1           0     104869 MPI_Finalize()
  0.5           80           81          51     9614.89       1597 copy_u
  0.5       0.0621           73     45.3333     90.6667       1631 recv_full_u
  0.4           60           60     5.66667           0      10762 solution_check
  0.4           43           58      100001      100001          1 limdiff [THROTTLED]
  0.2           12           28     36833.3     36833.3          1 shallow2d_flux
  0.2           25           25     5.66667           0       4555 viz_frame
  0.2           25           25           1           0      25248 MPI_Barrier()
  0.1           20           20      100001           0          0 central2d_correct_sd [THROTTLED]
  0.1           16           16     36833.3           0          0 shallow2dv_flux
  0.1           15           15      100001           0          0 xmin2s [THROTTLED]
  0.0            7            7    0.111111           0      66771 viz_close
  0.0            5            5         250           0         22 MPI_Allreduce()
  0.0            3            5           1     13333.3       5336 lua_init_sim
  0.0            2            2        6000           0          0 copy_subgrid
  0.0            2            2           1           0       2708 viz_open
  0.0        0.133            2         250         250         10 shallow2d_speed
  0.0            2            2         250           0          9 shallow2dv_speed
  0.0            2            2     45.3333           0         47 MPI_Recv()
  0.0            1            1       11837           0          0 central2d_offset
  0.0            1            1     11111.2           0          0 central2d_offset [THROTTLED]
  0.0       0.0189       0.0201           1           2         20 central2d_init
  0.0      0.00733      0.00733           1           0          7 copy_basic_info
  0.0      0.00122      0.00122           2           0          1 central2d_free
  0.0     0.000778     0.000778           1           0          1 MPI_Comm_size()
  0.0     0.000444     0.000444           1           0          0 MPI_Comm_rank()
```

B Profiling table: large scale on 9 cores

```
FUNCTION SUMMARY (mean):
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            5    15:34.549           1           1  934549844 .TAU application
100.0        0.461    15:34.543           1           3  934543980 main
 99.8        0.763    15:32.687           1     119.444  932687449 run_sim
 98.4       0.0854    15:19.270          50          50   18385400 central2d_run
 98.4            4    15:19.269          50        6060   18385398 central2d_xrun
 95.1        2,382    14:48.441        2424      104849     366519 central2d_step
 47.7        1,531     7:25.437        2424 4.93526E+06     183761 central2d_predict
 47.3     7:21.737     7:21.766 4.91345E+06       49979         90 limited_derivk
 47.1        2,236     7:20.259        2424 4.99163E+06     181625 central2d_correct
 47.1     7:20.090     7:20.119 4.91345E+06       50022         90 limited_deriv1
  3.3            9       30,591        1212       33936      25240 central2d_periodic
  3.3       30,494       30,494        4848           0       6290 MPI_Sendrecv()
  1.1        0.131       10,712          51     96.3333     210043 gather_sol
  0.9        0.102        8,610     45.3333     45.3333     189947 send_full_u
  0.9        8,610        8,610     45.3333           0     189945 MPI_Send()
  0.2        1,977        1,977          51           0      38767 copy_u
  0.2           14        1,877     45.3333     90.6667      41415 recv_full_u
  0.2        1,508        1,508     5.66667           0     266283 solution_check
  0.1        1,302        1,302           1           0    1302127 MPI_Init()
  0.1          616          616     5.66667           0     108842 viz_frame
  0.1          553          553           1           0     553943 MPI_Finalize()
  0.1          503          503           1           0     503194 MPI_Barrier()
  0.0           34          361      100001      100001          4 shallow2d_flux [THROTTLED]
  0.0          327          327      100001           0          3 shallow2dv_flux [THROTTLED]
  0.0            1          205        1212        1212        169 shallow2d_speed
  0.0          204          204        1212           0        168 shallow2dv_speed
  0.0          109          109     45.3333           0       2419 MPI_Recv()
  0.0           87           87       29088           0          3 copy_subgrid
```

```
0.0          55          70           1      100001       70665 lua_init_sim
0.0          43          58      100001      100001           1 limdiff [THROTTLED]
0.0          42          42      100001           0           0 central2d_correct_sd [THROTTLED]
0.0          27          27        1212           0          22 MPI_Allreduce()
0.0          15          15      100001           0           0 xmin2s [THROTTLED]
0.0          14          14      100001           0           0 central2d_offset [THROTTLED]
0.0           4           4    0.111111           0       36135 viz_close
0.0       0.639       0.639           2           0         320 central2d_free
0.0       0.227       0.227           1           0         227 viz_open
0.0      0.0163      0.0171           1           2          17 central2d_init
0.0     0.00878     0.00878           1           0           9 copy_basic_info
0.0    0.000556    0.000556           1           0           1 MPI_Comm_size()
0.0    0.000222    0.000222           1           0           0 MPI_Comm_rank()
```