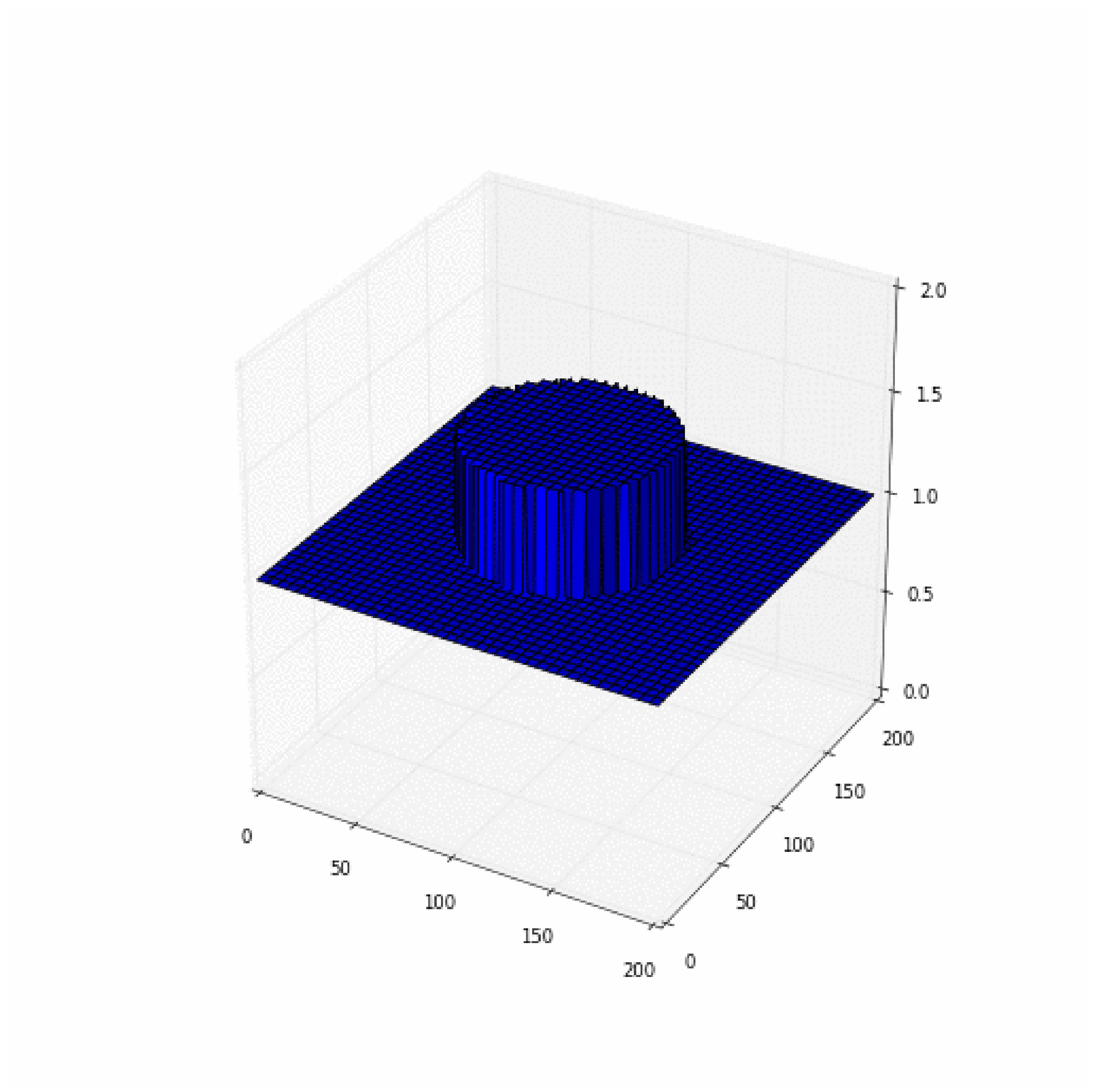
CS 5220 – Project 2

Junteng Jia – jj585
Sania Nagpal – sn579
Bryce Evans – bae43



We did the domain decomposition for speed up and it works!

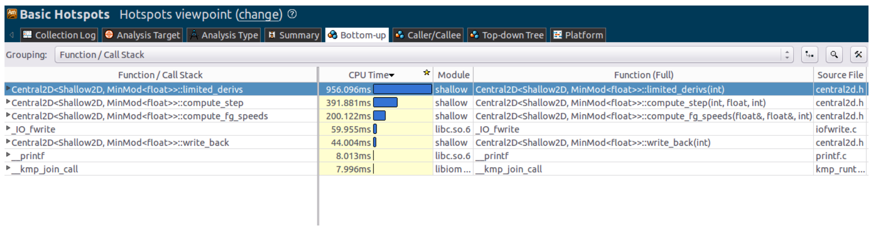
Our work

There are three major optimization base on the original code.

1. Profiling
2. Parallelization
3. Tuning

Profiling

We did some hotspot analysis of the serial code, and this is what we find out:



Basic Hotspots Hotspots viewpoint (change)

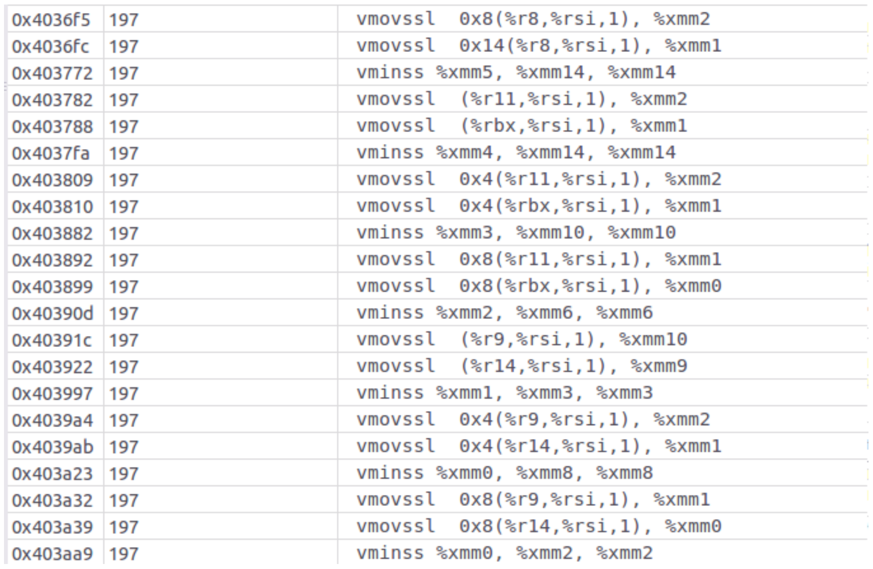
Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform

Grouping: Function / Call Stack

Function / Call Stack	CPU Time	Module	Function (Full)	Source File
Central2D<Shallow2D, MinMod<float>>::limited_derivs	956.096ms	shallow	Central2D<Shallow2D, MinMod<float>>::limited_derivs(int)	central2d.h
Central2D<Shallow2D, MinMod<float>>::compute_step	391.881ms	shallow	Central2D<Shallow2D, MinMod<float>>::compute_step(int, float, int)	central2d.h
Central2D<Shallow2D, MinMod<float>>::compute_fg_speeds	200.122ms	shallow	Central2D<Shallow2D, MinMod<float>>::compute_fg_speeds(float&, float&, int)	central2d.h
_IO_fwrite	59.955ms	libc.so.6	_IO_fwrite	iofwrite.c
Central2D<Shallow2D, MinMod<float>>::write_back	44.004ms	shallow	Central2D<Shallow2D, MinMod<float>>::write_back(int)	central2d.h
__printf	8.013ms	libc.so.6	__printf	printf.c
__kmp_join_call	7.996ms	libiom…	__kmp_join_call	kmp_runt…

Figure 1: Hotspot analysis by amplxe-cl

We find that more 58% of time is spent on the function call "limited_derivs". However, to our surprise, the function has been vectorized on assembly language level. We think since the data is not continuous in memory, there is a delay for acquiring data from memory to register. Probably that's why this function call is so slow. To further optimize this function call, we have to work on data dependence.



```
0x4036f5  197    vmovssl  0x8(%r8,%rsi,1), %xmm2
0x4036fc  197    vmovssl  0x14(%r8,%rsi,1), %xmm1
0x403772  197    vminss %xmm5, %xmm14, %xmm14
0x403782  197    vmovssl  (%r11,%rsi,1), %xmm2
0x403788  197    vmovssl  (%rbx,%rsi,1), %xmm1
0x4037fa  197    vminss %xmm4, %xmm14, %xmm14
0x403809  197    vmovssl  0x4(%r11,%rsi,1), %xmm2
0x403810  197    vmovssl  0x4(%rbx,%rsi,1), %xmm1
0x403882  197    vminss %xmm3, %xmm10, %xmm10
0x403892  197    vmovssl  0x8(%r11,%rsi,1), %xmm1
0x403899  197    vmovssl  0x8(%rbx,%rsi,1), %xmm0
0x40390d  197    vminss %xmm2, %xmm6, %xmm6
0x40391c  197    vmovssl  (%r9,%rsi,1), %xmm10
0x403922  197    vmovssl  (%r14,%rsi,1), %xmm9
0x403997  197    vminss %xmm1, %xmm3, %xmm3
0x4039a4  197    vmovssl  0x4(%r9,%rsi,1), %xmm2
0x4039ab  197    vmovssl  0x4(%r14,%rsi,1), %xmm1
0x403a23  197    vminss %xmm0, %xmm8, %xmm8
0x403a32  197    vmovssl  0x8(%r9,%rsi,1), %xmm1
0x403a39  197    vmovssl  0x8(%r14,%rsi,1), %xmm0
0x403aa9  197    vminss %xmm0, %xmm2, %xmm2
```

Figure 2: Assmbly for hotspot

Junteng Jia – jj585
Sania Nagpal – sn579
Bryce Evans – bae43

Parallelization

We parallelized our code based on domain decomposition idea. The `-d` option specify how many time we split the domain along one direction.

In our code, each pocessor has its own domain of simulation with `nghost` layer of ghost cells. We maintain an extra data structure `sg_` for the whole simulation board. And there are three stages in each iteration of our parallel code:

- `Apply periodic`: In this stage, each thread copy the periodic boundary to their own ghost cells. After this stage, each one of those subdomains is ready for simulation.
- `Run step`: In this stage, each thread do simulation on their own subdomain, and the water height and water speed of each cell is updated.
- `Write back`: In this stage, every thread copy their updated subdomain back to the `sg_` data structure. There is a barrier after this stage to make sure before next iteration, the data needed by ghost cells is up-to-date.

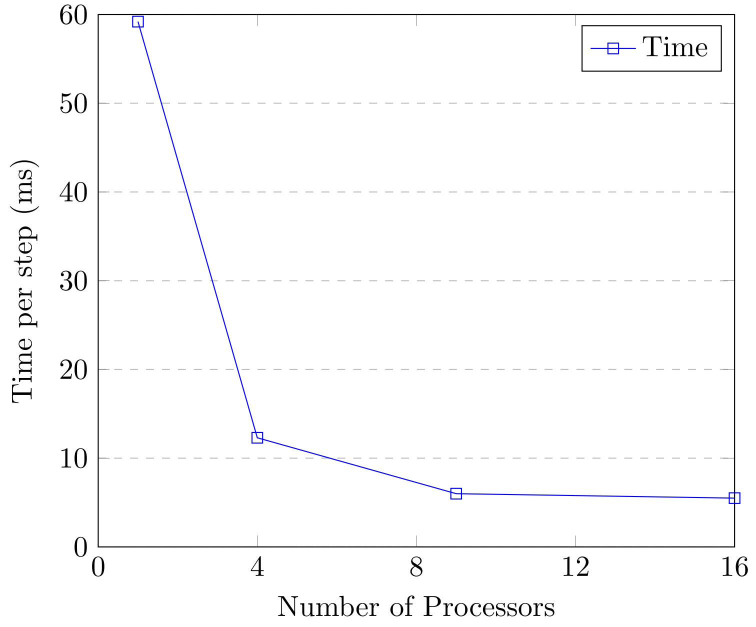


Figure 3: Strong Scaling

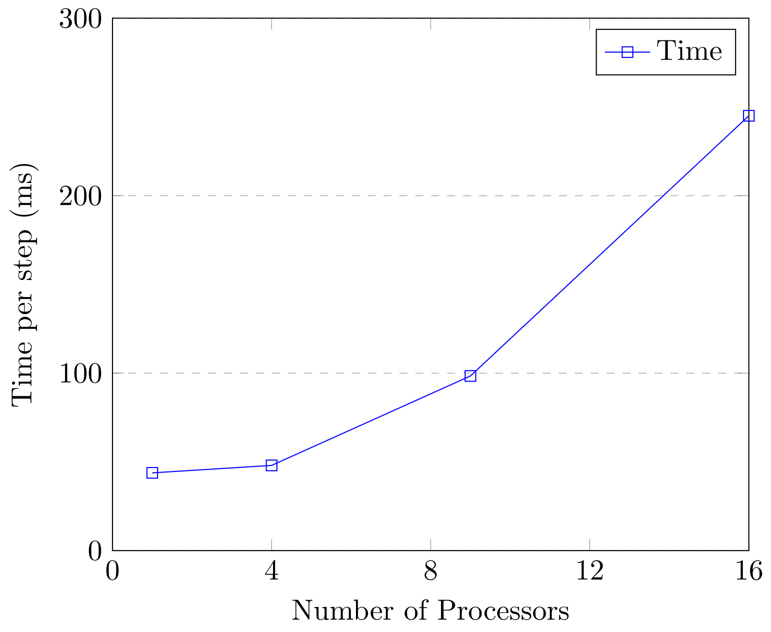


Figure 4: Weak Scaling

Summary

Through this project:

- We analysed the hotspot of our code, yet we haven't done any tuning.
- Strong scaling for the parallel code is okey for the first three points, but the fourth point is bad.
- Weak scaling for the parallel code is okey for the first three points, but the fourth point is terrible.

In the furture:

- We will try to cut down the time for calling `limited_diffvs` by tuning.
- We will use more layer of ghost cells and batch of steps simulations.
- We will further optimize the code by hotspot analysis.