Group 18 Project 2

Eric Gao – emg222
Yu Su – ys576
Vikram Thapar – vt87

1 Profiling

We began the project with some basic profiling with basic hotspot analysis. We first looked at the vectorization report to see that nothing was vectorized and that could be something we could start with. We then used VTune Amplifier to see where the bottlenecks in the code are.

The initial report created by VTune Amplifier was:

```
Function               Module          CPU Time   Spin Time   Overhead Time
---------------------  --------------  ---------  ----------  --------------
limited_derivs         shallow            1.174s          0s              0s
compute_step           shallow            0.562s          0s              0s
compute_fg_speeds      shallow            0.199s          0s              0s
write_frame            shallow            0.005s          0s              0s
run                    shallow            0.003s          0s              0s
solution_check         shallow            0.001s          0s              0s
```

We see that the slowest part of the code are the two functions: `limited_derivs` and `compute_step`. Looking at the vectorization report, we see that there is a lot of unvectorized code that could be optimized. Furthermore, there are several parts that are inside of loops that could be parallelized `omp parallel for` or some other scheme of parallelization.

We then could use VTune Amplifier to look at specific lines of the code for the given functions and see how long they take. It seems pretty obvious what we need to do for the `limited_derivs` function: we need to implement some blocking scheme so that we can do some parallelization using `OpenMP`. We then saw that the slowest part of the `compute_step` function was the corrector step of the function, which took about 3 times as long as the predictor step. The equation that was being implemented involves operations on vectors that could first be vectorized and then parallelized. To do this, we thought we would have to modify how the data was laid out in memory so that it was contiguous in memory.

2 Parallelization

We have used "pragma loop for" in two functions which are the bottleneck in the calculations: `limited_derivs` and `compute_step`. The report created by VTune amplifier after modifying the code was:

Function	Module	CPU Time	Spin Time	Overhead Time
limited_derivs	shallow	1.173s	0s	0s
compute_step	shallow	1.422s	0s	0s
compute_fg_speeds	shallow	0.368s	0s	0s
write_frame	shallow	0.004s	0s	0s
run	shallow	0.015s	0s	0s
solution_check	shallow	0.009s	0s	0s

The above time report shows that the parallelized version of the code for the two bottleneck functions is slower than the basic code. Contradictorily, we found that if we compare the times reported in .o files, the parallel code takes a total time which is approximately 2 times faster than the basic code. We are not able to understand this discrepancy as one would expect similar trends in both the cases. Also, we are planning to use MPI rather than omp and combine it with domain decomposition to obtain the significant speed up.

3 Tuning

We tried writing a kernel for the corrector step of the `compute_step` function. However, it looked like the most effective thing that we could do was also one of the simplest. We saw that we were always using the `vec` of length 3 for the innermost loop, so we just unrolled that loop. The compiler then took care of the rest and we went from taking approximately **4.5e-2** time to about **3.6e-2**, which is about a 25% speed up due to this simple optimization.

We tried another thing which was to try to cache all the `vecs` that were being used to attempt to increase speed, but it resulted in times similar to the basic approach. We also attempted to implement our own kernel, but it looked like the compiler beat us, since were were getting times comparable to the basic approach. Our attempted kernel is commented out in our `central2d.h` in the `compute_step` function.

We realize we could do some sort of blocking scheme to maximize memory reuse in this kernel. We will try to implement this for the final report.

Final notes:

Please see the relevant `.txt` files for timings produced by running under different configurations.