CS 5220 HW 2

Laura Herrle lmh95, Sijia Ma sm2462, Kaiyan Xiao kx58

October 19, 2015

1 Profiling

1.1 VTune Amplifier

We attempted to set up and use VTune Amplifier. Since we all have Macs, and VTune can only be used on Linux or Windows, we attempted to run it in a virtual machine (that had plenty of resources allocated). However, this ended up being incredibly difficult to use. We first attempted to use the profiler remotely with the cluster, but that took too much of our time to set up. We then tried to run it locally on the VM, and while we could set it up, attempting to run it locked up the VM. In the end, we decided to time the code manually (this ended up going much more quickly).

1.2 Manual timing

We timed how long each of the four method calls in `run` took, and furthermore how long each section of `compute_step` took. The results for one run of `make big` are in Table 1.

Based on these results, there are two main bottlenecks where we should concentrate our efforts: `limited_drivers` and the corrector section of `compute_step`.

`apply_periodic`	`compute_fg_speeds`	`limited_derivs`	`compute_step`	predictor	corrector	copy
0	19	72	61	13	44	3
0	21	63	60	13	44	2
0	18	62	52	10	38	3
0	29	71	51	10	38	2
0	16	71	52	10	38	3
0	21	70	56	12	39	3
0	21	69	54	10	41	2
0	18	69	54	13	36	4
0	20	70	40	9	28	2
0	15	70	41	10	28	2
0	17	65	52	13	35	3

0	21	71	47	10	32	3
0	20	73	60	13	42	2
0	18	71	55	13	38	3
0	15	70	51	12	35	3
0	21	67	52	11	37	2
0	17	70	61	12	45	2
0	19	62	49	9	37	2
0	19	71	50	13	34	2
0	14	69	63	12	46	3
0	20	63	53	13	36	3
0	22	71	57	10	43	3
0	17	70	62	14	44	4
0	16	78	48	12	31	3
0	16	68	72	13	54	4
0	24	64	49	11	34	3
0	22	71	58	14	41	2
0	20	69	55	11	42	2
0	16	73	62	11	47	3
0	22	71	55	13	38	3
0	22	69	69	14	42	2
0	15	73	53	13	37	2
0	16	62	50	13	34	2
0	15	75	55	11	41	2
0	19	73	66	13	49	3
0	21	74	58	13	42	2
0	21	83	57	11	41	4
0	17	68	47	13	31	2
0	20	68	63	13	47	2
0	16	70	54	14	34	5
0	18	80	58	14	40	3
0	21	61	54	13	38	2
0	21	75	44	12	29	1
0	18	63	55	9	41	3
0	22	66	55	11	40	3
0	21	73	46	11	32	2
0	19	75	64	9	51	3
0	21	67	48	13	32	2
0	16	69	52	10	38	3

0	21	65	47	13	31	2

Table 1: Time per Section

2 Parallelization

2.1 Domain Decomposition

The basic idea for parallizing this shallow water problem is to divide the domain into small subdomains and compute each subdomain seperately in different threads. And we also need ghost cells around each subdomain to update several steps before communicate with other threads. The number of ghost cell decide the communication frequency and therefore affect the effiency of parallization.

One of the problem we have in domain decomposition is that we have to estimate the wave speed, or the maximum value of the wave speed, which determines the updates. It is important to know it for parallelizing the problem. And we will do some experiment to find out.

2.2 Parallelize Using OpenMP

In `central_2D`, we can just simpy use `pragama omp parallel` outside the while loop to parallelize the code. But we have not tested it yet.

3 Future Work

We plan to add parallelization and tune based on the results of our profiling. We will definitely add parallelization in `limit_derivs`, at least (naively) in four parts with the four calls to `limdiff` per inner for loop. We will also tune the corrector stage of `compute_step`, and possibly add parallelization since the computations are independent and not memoized.