Osvaldo Salinas
Eid: ons239
OpenMP HW2

Instructions:

1. Open the Terminal and use the following command to log in to frontera.tacc.utexas.edu: ssh [your_tacc_username]@frontera.tacc.utexas.edu
2. Enter your password and TACC Token when prompted.
3. Once logged in, enter the workspace using "cd $WORK".
4. Use the command " cd pcse/ TheArtOfHPC_vol2_parallelprogramming/exercises/exercises-omp-c/hw1/rb" to enter the folder.
5. Use the following commands in order to compile and run your code successfully:
    a. idev
    b. ./compile prb_a.c
    c. ./dothis a.out_prb_a

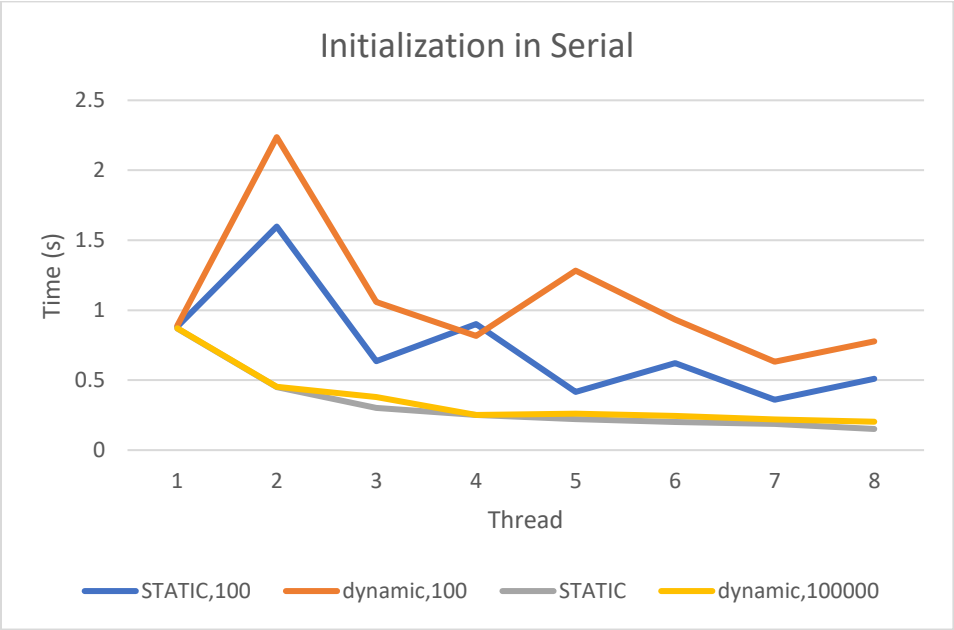
PART A

iv. Execute ./dothis to execute a.out_prb_a (various # of threads and

schedules).Times for 1-8 threads are reported for 4 types of scheduling.

Schedule	STATIC,100	dynamic,100	STATIC	dynamic,100000
1	0.883311	0.889083	0.869121	0.871263
2	1.597862	2.236886	0.450007	0.453488
3	0.635952	1.057798	0.302366	0.379539
4	0.901748	0.816237	0.251577	0.251293
5	0.415440	1.283495	0.221770	0.260383
6	0.621356	0.932881	0.201490	0.244639
7	0.360216	0.632076	0.188158	0.220205
8	0.509403	0.776372	0.151246	0.202966

Osvaldo Salinas
Eid: ons239
OpenMP HW2



The execution time at 1 thread is approximately 0.85 seconds for all four scheduling methods, however, the choice of scheduling method does have an impact. When using static scheduling with 100 chunks and dynamic scheduling with 100 chunks, the execution time peaks at 2 threads, but then decreases as the number of threads increases from 2 to 8. However, when using static scheduling without a specific chunk size and dynamic scheduling with 100000 chunks (columns 4 and 5), the execution time decreases rapidly at the beginning but then plateaus as the number of threads increases. The results of the first two columns indicate that static scheduling is more effective for smaller chunk sizes, whereas dynamic scheduling becomes faster as the chunk size increases. In summary, the choice of scheduling method is important and can affect the execution time of the program.

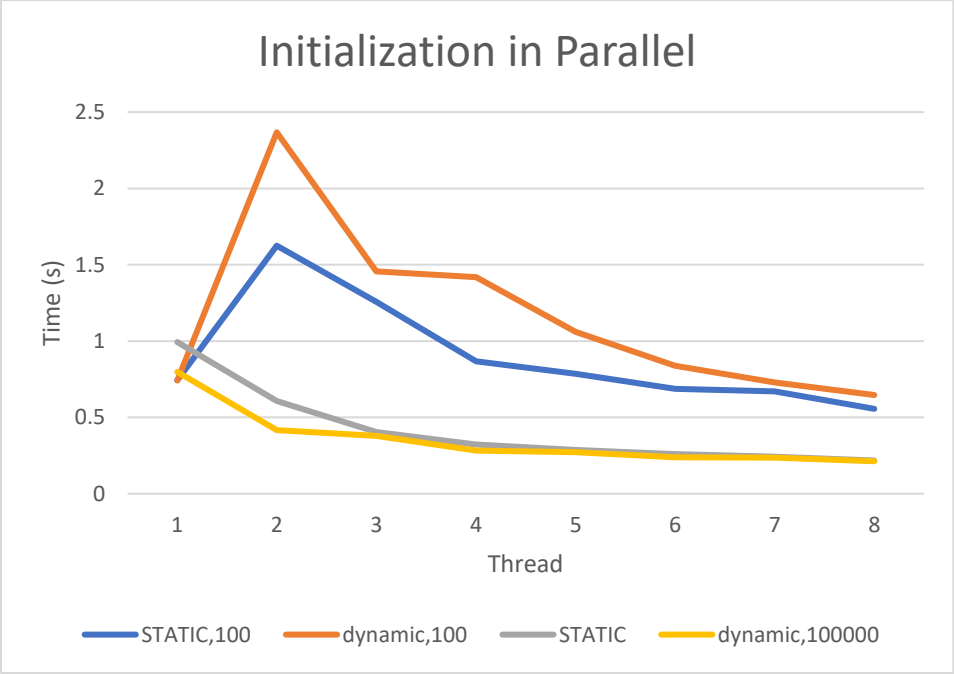
Coding changes shown in red:

```
***
#pragma omp parallel for

for(i = 0; i < N-1; i+=2) {a[i] = 0.0; a[i+1] = 1.0;}

t0 = gtod_timer();
***
```

v. Change code to now parallelize the initialization loop. DO NOT include a schedule clause—let it default to static scheduling. Compile with compile prb_a.c and execute dothis.

Osvaldo Salinas
Eid: ons239
OpenMP HW2

Schedule	STATIC,100	dynamic,100	STATIC	dynamic,100000
1	0.742984	0.741259	0.993338	0.798095
2	1.624746	2.367390	0.607252	0.417499
3	1.257939	1.456800	0.404751	0.377944
4	0.867427	1.418099	0.321093	0.281425
5	0.784599	1.060362	0.287358	0.272753
6	0.687388	0.837088	0.258229	0.237674
7	0.671082	0.728938	0.243401	0.236531
8	0.555913	0.646451	0.217368	0.212656



By parallelizing the program's initialization loop, the scaling has been significantly enhanced. For example, the static scheduling method using 100 chunks experienced a decrease in time from the serial to parallel execution. This same pattern was evident in the static scheduling without a specific chunk size and the dynamic scheduling using 100000 chunks. However, the dynamic scheduling approach using a chunk size of 100 exhibited a small and almost indiscernible increase in execution time.

The reason behind this performance improvement and better scaling is that parallelizing the initialization loop can significantly reduce the time it takes to initialize the program's parameters. However, the different scheduling methods still have a considerable impact on performance and scaling. Scheduling methods with smaller chunks of 100 or below take a slightly longer time from 1-8 threads, while static scheduling without a specific chunk size and dynamic scheduling with a relatively larger chunk size outperforms the other two methods with a significantly faster margin, taking less than 0.2 seconds by the time it reaches 8 threads.

Osvaldo Salinas
Eid: ons239
OpenMP HW2

Watch where the threads are executing with top. (On the node in another window execute "top" and then hit the "1" key to see the loads. Type s and then 1 to have top update every second.)

For the most part, when looking at top when the code is initialized in parallel vs in serial, all columns appear to look the same and so there is similar behavior. Between both versions, the threads are computed on similar CPUs but take different amounts of time.

PART B

b. Using prc_a.c, reduce the number of parallel regions (directives).

i. copy prb_a.c/F90 to prb_b.c or prb_b.F90

ii. Modify directives in the while loop so that there is only one parallel directive in the while loop. (Hint: Think "single" for the error and niter vars.)

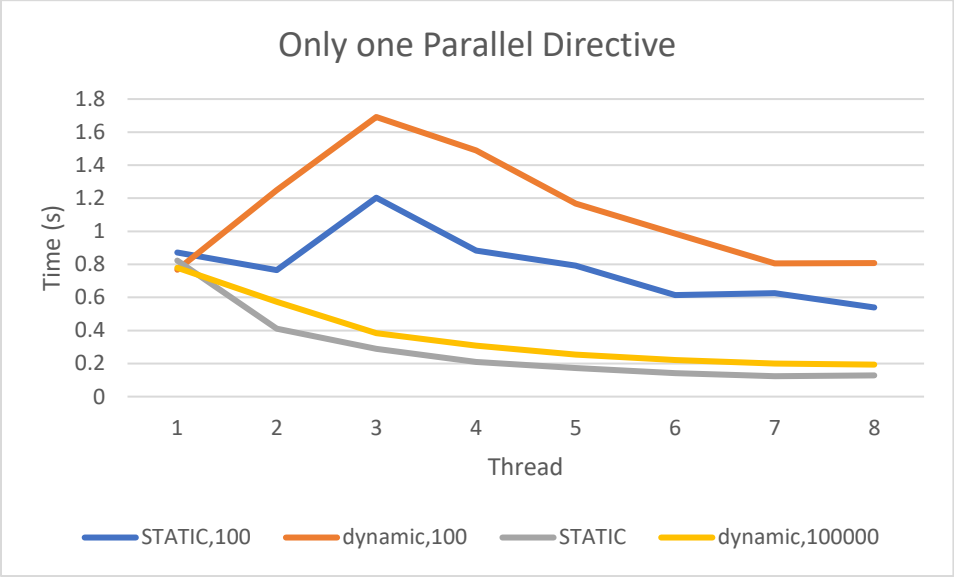
iii. Compile: use compile prb_b.c; and execute: dothis.

Changes to code shown below in red:

```
***
do {
#pragma omp parallel
 {
#pragma omp for schedule(runtime)
 for (i = 1; i < N; i+=2) a[i] = (a[i] + a[i-1]) / 2.0;
#pragma omp for schedule(runtime)
 for (i = 0; i < N-1; i+=2) a[i] = (a[i] + a[i+1]) / 2.0;

 #pragma omp single
 error=0.0; niter++;
#pragma omp for schedule(runtime) reduction(+:error)
***
```

Osvaldo Salinas
Eid: ons239
OpenMP HW2

Schedule	STATIC,100	dynamic,100	STATIC	dynamic,100000
1	0.871147	0.767395	0.822958	0.779481
2	0.764991	1.250020	0.411229	0.573690
3	1.203464	1.691558	0.289161	0.384721
4	0.882800	1.489503	0.209675	0.307929
5	0.791514	1.167068	0.172146	0.255109
6	0.614428	0.985496	0.141663	0.222073
7	0.626010	0.806408	0.123588	0.200085
8	0.539177	0.808522	0.129066	0.193490



Similar to Part A, the execution time behavior of the scheduling methods appears to be relatively comparable in Part B. However, for static and dynamic scheduling using 100 chunks, the time peaks at three threads instead of two, and the scaling has worsened. Despite this, the scheduling methods in Part B exhibit significantly faster run times compared to Part A, except for static scheduling without a specific chunk size.

The reason why the run times in Part A and Part B should be similar, despite significantly reducing the number of forking requests, is that reducing the number of loops increases the time taken by each iteration to run. Consequently, this balances out, as there is still some overhead, which is dependent on the scheduling method.

Osvaldo Salinas
Eid: ons239
OpenMP HW2
PART C

There are 2 optimizations to be performed here: 1.) Use a single parallel region (directive) outside the while loop; 2.) If the 2 red-black loops can be combined, fuse them. We use16 threads here.

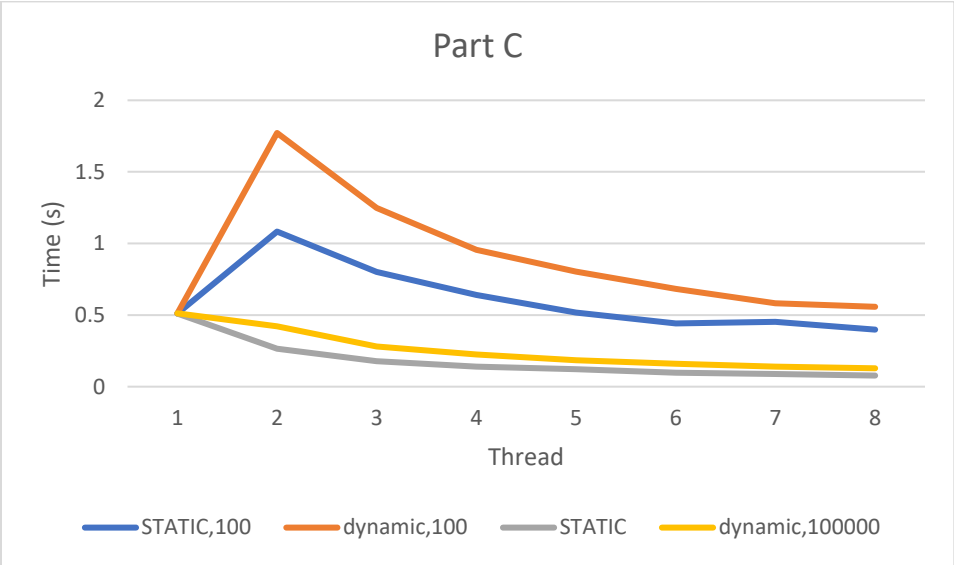
General Instructions: Copy rb.c/F90 to prb_c.c or prb_c.F90 [or copy from prb_b.c/F90], parallelize, compile with compile prb_c.c/F90; use dothis to execute and run on a compute node.

i. Once you have constructed a single parallel directive outside of the while loop, compile and run multiple times with dothis, save output. (Include the initialization of the a array in the parallel region.)

ii. Now combine red-black loops, compile and run multiple times with dothis (save output)

Schedule	STATIC,100	dynamic,100	STATIC	dynamic,100000
1	0.511488	0.511445	0.510510	0.512282
2	1.083384	1.772155	0.265950	0.420899
3	0.800676	1.248286	0.177515	0.280884
4	0.641335	0.955505	0.139607	0.225685
5	0.517870	0.804076	0.123100	0.184194
6	0.442339	0.683562	0.096643	0.160351
7	0.451943	0.582302	0.087829	0.140957
8	0.398536	0.558899	0.077551	0.128407

Osvaldo Salinas
Eid: ons239
OpenMP HW2



Once the two optimizations were implemented, namely utilizing a single parallel region outside the while loop and consolidating the two red-black loops, all four scheduling methods exhibited a considerably faster overall performance, in contrast to Part A and Part B.