COMP4300
PARALLEL SYSTEMS
Assignment 2
Advection Solver For Shared Address Space Programming Models

student name : Yang Wang

University ID: u5833090

Course Lecturer : Peter Strazdins

Question 1

The following cases is disused with the combinations of ways to parallel via 1D Decomposition, the test inputs for the parallelisation is M=2000,N=2000,r=100, furthermore to have a more clear view of the cache miss rate, i use `perf:` Linux profiling with performance counters to calculate it.

a. **Maximize performance**

To maximise the performance, here i choose to parallel the outer loop of advection update stage 1 and stage 2 with the the dynamic scheduling and the for loop variable i,j are set to be private, with dynamic scheduling the calculation is more load balanced and is more suitable for the loop when the computation of inner loop iterations is different from that of outer loop iterations, and further more, with dynamic scheduling it can reduce the cache miss in compiler optimisations because code can be rearranged to avoid conflicts in a direct-mapped cache, and accesses to arrays can be reordered to operate on blocks of data rather than processing rows of the array.

```
#pragma omp parallel  default(shared) private (i,j,tid)
   #pragma omp for schedule(dynamic)
   for (j=0; j < N+1; j++){ // advection update stage 1
      for (i=0; i < M+1; i++){
      V(ut,i,j) = 0.25*(V(u,i,j) + V(u,i-1,j) + V(u,i,j-1) + V(u,i-1,j-1))
       -0.5*dt*(sy*(V(u,i,j) + V(u,i,j-1) - V(u,i-1,j) - V(u,i-1,j-1)) +
             sx*(V(u,i,j) + V(u,i-1,j) - V(u,i,j-1) - V(u,i-1,j-1)));
    }
   }
#pragma omp parallel  default(shared) private (i,j,tid)
   #pragma omp for schedule(dynamic)
   for (j=0; j < N; j++){ // advection update stage 2
   for (i=0; i < M; i++){
      V(u, i, j) +=
       - dtdy * (V(ut,i+1,j+1) + V(ut,i+1,j) - V(ut,i,j) - V(ut,i,j+1))
       - dtdx * (V(ut,i+1,j+1) + V(ut,i,j+1) - V(ut,i,j) - V(ut,i+1,j));
    }
   }
```

Test Result:

1. when p = 8

[yw8443@raijin3 openmp]$ perf stat -e cache-references,cache-misses numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100

Advection of a 2000x2000 global field on 8 threads for 100 steps.

dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04

Advection time 6.697e-01, GFLOPs rate=1.37e+01 (per core 1.72e+00)

Average error of final field: 6.806e-07

Performance counter stats for 'numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100':

148514857 cache-references

23778876 cache-misses # 16.011 % of all cache refs

0.966137026 seconds time elapsed

2. when p = 16

[yw8443@raijin3 openmp]$ cat batch2.o5183544

Advection of a 2000x2000 global field on 16 threads for 100 steps.

dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 3.786e-01, GFLOPs rate=2.43e+01 (per core 1.52e+00)
Average error of final field: 6.806e-07

b. **Maximize the number of OpenMP parallel region entry**
To maximise the number of parallel region entry, here i set update boundary, update advection stage to be parallel

```
#pragma omp parallel for schedule(dynamic) default(shared) private (i)
  for (i=1; i < M+1; i++) { //update left & right boundaries
    V(u, i, 0)   = V(u, i, N);
    V(u, i, N+1) = V(u, i, 1);
  }
#pragma omp parallel for schedule(dynamic) default(shared) private (j)
  for (j=0; j < N+2; j++) { //update top & bottom boundaries
    V(u, 0, j)   = V(u, M, j);
    V(u, M+1, j) = V(u, 1, j);
  }

  u = &V(u, 1, 1); // make u relative to the interior points for the updates
#pragma omp parallel  default(shared) private (i,j,tid)
  #pragma omp for schedule(dynamic)
  for (j=0; j < N+1; j++){ // advection update stage 1
#pragma omp parallel default(shared) private (i,j,tid)
#pragma omp for schedule(dynamic)
     for (i=0; i < M+1; i++){
        V(ut,i,j) = 0.25*(V(u,i,j) + V(u,i-1,j) + V(u,i,j-1) + V(u,i-1,j-1))
         -0.5*dt*(sy*(V(u,i,j) + V(u,i,j-1) - V(u,i-1,j) - V(u,i-1,j-1)) +
                  sx*(V(u,i,j) + V(u,i-1,j) - V(u,i,j-1) - V(u,i-1,j-1)));
   }
  }
#pragma omp parallel  default(shared) private (i,j,tid)
  #pragma omp for schedule(dynamic)
  for (j=0; j < N; j++){ // advection update stage 2
#pragma omp parallel default(shared) private (i,j,tid)
 #pragma omp for schedule(dynamic)
    for (i=0; i < M; i++){
        V(u, i, j) +=
         - dtdy * (V(ut,i+1,j+1) + V(ut,i+1,j) - V(ut,i,j) - V(ut,i,j+1))
         - dtdx * (V(ut,i+1,j+1) + V(ut,i,j+1) - V(ut,i,j) - V(ut,i+1,j));
   }
  }
```

Test Result:
1. when p = 8
[yw8443@raijin3 openmp]$ export OMP_NUM_THREADS=8
[yw8443@raijin3 openmp]$ perf stat -e cache-references,cache-misses numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100
Advection of a 2000x2000 global field on 8 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 3.121e+00, GFLOPs rate=2.95e+00 (per core 3.69e-01)
Average error of final field: 6.417e+221

Performance counter stats for 'numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100':

180042172 cache-references
1479238 cache-misses # 0.822 % of all cache refs

3.427658349 seconds time elapsed

2. when p = 16
[yw8443@raijin3 openmp]$ cat batch2.o5181360
Advection of a 2000x2000 global field on 16 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 2.241e+00, GFLOPs rate=4.11e+00 (per core 2.57e-01)
Average error of final field: 3.207e+221

From the result of perf and output, the cache miss rate has been reduced a lot after maximising the number of openmp parallel regions, however, the advection time of the program has been prolonged a lot, this may due to too many parallel regions are running and there is no synchronised mechanism implemented here, the OMP parallel for no wait directives can be used to improve the performance here.

c. **Maximize shared cache misses involving coherent reads**

To maximise the shared cache miss, here i use the static scheduling so that the iterations blocks are mapped statically to the execution threads in a round-robin fashion, in this circumstance it would increase the possibility that several threads are computing the same era of advection computation in the cache line because they are not scheduled as a first serve first out sequence , which has a false sharing, and when the thread redo the computation after other threads finished their round robin time, it have a greater possibility that the data need for the thread is updated or changed so that the cache miss will happen more frequently.

```
#pragma omp parallel  default(shared) private (tid)
   #pragma omp for schedule(static)
   for (j=0; j < N+1; j++){ // advection update stage 1
      for (i=0; i < M+1; i++){
           //  tid = omp_get_thread_num();
           //        printf("this is second one  %d\n", tid);
          V(ut,i,j) = 0.25*(V(u,i,j) + V(u,i-1,j) + V(u,i,j-1) + V(u,i-1,j-1))
           -0.5*dt*(sy*(V(u,i,j) + V(u,i,j-1) - V(u,i-1,j) - V(u,i-1,j-1)) +
                    sx*(V(u,i,j) + V(u,i-1,j) - V(u,i,j-1) - V(u,i-1,j-1)));
    }
   }
#pragma omp parallel  default(shared) private (tid)
   #pragma omp for schedule(static)
   for (j=0; j < N; j++){ // advection update stage 2
    for (i=0; i < M; i++){

          V(u, i, j) +=
           - dtdy * (V(ut,i+1,j+1) + V(ut,i+1,j) - V(ut,i,j) - V(ut,i,j+1))
           - dtdx * (V(ut,i+1,j+1) + V(ut,i,j+1) - V(ut,i,j) - V(ut,i+1,j));
    }
   }
```

Test Result :
When p = 8

[yw8443@raijin3 openmp]$ perf stat -e cache-references,cache-misses numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100
Advection of a 2000x2000 global field on 8 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 7.051e-01, GFLOPs rate=1.30e+01 (per core 1.63e+00)
Average error of final field: 6.806e-07

 Performance counter stats for 'numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100':

      75132458   cache-references
      20181687   cache-misses   #   26.861 % of all cache refs

    1.005042508 seconds time elapsed

when p = 16

[yw8443@raijin3 openmp]$ cat batch2.o5182476
Advection of a 2000x2000 global field on 16 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 3.805e-01, GFLOPs rate=2.42e+01 (per core 1.51e+00)
Average error of final field: 6.806e-07

From the test result it is clear to see that after implementing the round-robin scheduling , the cache miss rate has been increased greatly which is appropriate for my approximation that the static scheduling will make a great influence on the cache miss rate.

d. Maximize the shared cache misses involving coherent writes
In this case, i keep the scheduling methods as static scheduling as it is tested to be quite effective, to test the coherent write, i parallel the write operation in advection update stage 1 and stage 2 and also i parallel the write operation in update boundary step

```
 #pragma omp parallel default(shared) private (i)
   #pragma omp for schedule(static)
  for (i=1; i < M+1; i++) { //update left & right boundaries
  V(u, i, 0)   = V(u, i, N);
  V(u, i, N+1) = V(u, i, 1);
 }
  #pragma omp parallel default(shared) private (j)
 #pragma omp for schedule(static)
 for (j=0; j < N+2; j++) { //update top & bottom boundaries
  V(u, 0, j)   = V(u, M, j);
  V(u, M+1, j) = V(u, 1, j);
 }

 u = &V(u, 1, 1); // make u relative to the interior points for the updates
 for (j=0; j < N+1; j++){ // advection update stage 1
  #pragma omp parallel default(shared) private (i,j,tid)
  #pragma omp for schedule(static)
    for (i=0; i < M+1; i++){
        V(ut,i,j) = 0.25*(V(u,i,j) + V(u,i-1,j) + V(u,i,j-1) + V(u,i-1,j-1))
          -0.5*dt*(sy*(V(u,i,j) + V(u,i,j-1) - V(u,i-1,j) - V(u,i-1,j-1)) +
                   sx*(V(u,i,j) + V(u,i-1,j) - V(u,i,j-1) - V(u,i-1,j-1)));
  }
 }
```

```
for (j=0; j < N; j++){ // advection update stage 2
  #pragma omp parallel default(shared) private (i,j,tid)
  #pragma omp for schedule(dynamic)
  for (i=0; i < M; i++){

        V(u, i, j) +=
          - dtdy * (V(ut,i+1,j+1) + V(ut,i+1,j) - V(ut,i,j) - V(ut,i,j+1))
          - dtdx * (V(ut,i+1,j+1) + V(ut,i,j+1) - V(ut,i,j) - V(ut,i+1,j));
 }
}
```

Test Result:
when p = 8
[yw8443@raijin3 openmp]$ perf stat -e cache-references,cache-misses numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100
Advection of a 2000x2000 global field on 8 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 2.441e+01, GFLOPs rate=3.77e-01 (per core 4.71e-02)
Average error of final field: 1.132e+222

when p = 16
[yw8443@raijin3 openmp]$ cat batch2.o5183225
Advection of a 2000x2000 global field on 16 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 6.389e+01, GFLOPs rate=1.44e-01 (per core 9.00e-03)
Average error of final field: 1.132e+222

From the test result it is clear to see that the after implanting the methods of coherent writes, the advection time has been increased greatly,this may due to the effect of cache coherency protocol, the time cost of threads waiting for the status changing from invalidate to valid cost a lot to compute.

case	P=8	P=16
A	6.697E-01	3.786E-01
B	3.121e+00,	2.241E+00
C	7.051E-01	3.805E-01
D	2.441E+01	6.389E+01

From the result given form the above table we can conclude that the maximise performance has the best effect of the advection calculation , thus i will keep this model in the following file.

Question 2

The performance model is partially referenced from the sample solution given from lecturer Peter Strazdins, let m = M/P, n = N/Q, ,during each iteration, the process need to update mn elements, the advection calculation in each iteration need 23 floating point operations , since the performance model is adapted to the 1D process, thus we can assume that the value of P should be equal to one all the time, the application's execute time is thus given by :

$$t = r(t_c(m,Q)+23mn\ t_f)$$
$$t_c(m,Q) = (t_s+n(t_{w,R}+t_{w,R})) \text{ if } Q>1$$
$$=0, \text{ otherwise}$$

the value of tf here i will keep using the calculated result if assignment 1 which is tf= 0.34e-09 second, to decide the relationship of the word time for cache read miss and cache write miss, on a read miss, lets assume that the L1 cache being modified is dirty, first write back the old data to L2 which takes 10 cycles, the read the new data from L2 in 10 cycles to write to L1, the L1 cache line is now marked clean, the total time for a read-miss can be estimated as 10 cycles, as for the write-miss, the L1 cache line is still assumed to dirty, then writing the old data back to L2 in 10 cycles, red the new data from L2 in 10 cycles to write in the L1 cache line, then write the new data to the L1 cache line, the L1 cache line remains dirty, total time for write miss can be estimated as 20cycles, thus here i have relationship that the per word time for cache write mess is twice than that of cache read miss, which we have $t_{w,w} = 2\ t_{w,R}$

To calculate the ts time, i set wall_time function in the parallel loop for two advection update stage to measure the cost of a parallel region entry/exits, to determine t_s we chose M = 1,N = p, and r = 1000 , from the test result we have t_s = 1.907e-06

To calculate the the word time, we choose N=Q=1(n = 1) to minimise the t_f term, and M = 1000 and r = 100 , i set $t_w = t_{w,w} + 2\ t_{w,R}$, thus from the formula i given above, we have
$100(t_s + M/P * t_w + 23\ mn\ t_f) = 3.28e\text{-}03$, from that we can calculate the t_w is 3.01e-08 , and $t_{w,R}$ = 2.01e-08, $t_{w,w}$ = 1.0e-08
The performance model is t = r(1.907e-06+ n(1.0e-08+2.01e-08)+23mn*0.34e-09)

Type of results	p=3 (M=N=2000, r=100)	p=8 (M=N=2000, r=100)	p=12 (M=N=2000, r=100)	p=16 (M=N=2000, r=100)
Estimated Results	0.784	0.392	0.261	1.96
Actual Results	8.847E-01	5.028E-01	4.538E-01	1.554

From the table above it is clear to see that the calculated result is quite approached to the actual test results, the performance model can be regarded as an effective one.

Question 3

To solve the 2d decomposition problem, here i set the advection update stage 1 and update stage 2 into the same parallel entry, the difference between 1d and 2d decomposing is the assign of the rows and columns of the advection field, the way of representing the location of each node in the 2 dimension is different from representing in one line, thus i set the start point and end point for each thread to do the computation, first_i = tid*(M/P); last_i = first_i +M/P+(tid<P-1? 0 : M % P); this is the same way of practical 4 sum section of parallel system , then i assign each advection update stage a static scheduling for loop and each for loop have one thread operate on it , the test result if good for small number of M and N ,when it comes to large number, it will become segmentation fault. Here i use M = 100 , N = 100 , r = 10 to test:

[yw8443@raijin3 openmp]$ cat batch2.o5183225
Advection of a 100x100 global field on 16 threads for 10 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 2.639e-04, GFLOPs rate=3.14e+00 (per core 1.96e-01)
Average error of final field: 5.437e-05

The advection of same input for 1d decomposition and 2d decomposition illustrates that 2d solution is not as effective as 1d result, this may due to he thread need to concerns the assignment of starting point and end point in a 2 dimension space and computation in one parallel entry is more than that of 1d decomposition.

Question 5

In the 2d decomposition, i map the index of the each node in the advection field into a 2d matrix , thus the thread and block can be represented as ix = threadIdx.x + blockIdx.x * blockDim.x,iy = threadIdx.y + blockIdx.y * blockDim.y, within this kind of index the adjacent thread has continuous threadx.x, which means the thread will be continuous as (0,0)(1,0)(2,0)… , the same way as my implementation of 2d openmp, i assign each thread with the starting point and ending point of the node location they need to calculate and i use dim3 grid, dim block to let 2 advection update filed step can be implemented in 2 dimension.

The test input remains the same with above test which is M=N=2000, r = 100
Test Result:

Advection time	Gx,Gy,Bx,By 1,10,1,10	Gx,Gy,Bx,By 10,1,10,1	Gx,Gy,Bx,By 1,20,1,20	Gx,Gy,Bx,By 20,1,20,1
1D parallel	2.972E+00	3.259E+00	1.032E+00	1.134E+00

Advection time	Gx,Gy,Bx,By 10,10,10,10	Gx,Gy,Bx,By 20,20,20,20	Gx,Gy,Bx,By 100,100,100,100	Gx,Gy,Bx,By 200,200,200,200	Gx,Gy,Bx,By 500,500,500,500	Gx,Gy,Bx,By 512,512,512,512
2D parallel	1.472E+00	1.546E+00	1.486E-01	3.325E-01	1.124E+00	1.144E+00

1D parallelisation : in 1d decomposition , from the test result given above , it is clear to see that the performance of 1*B is better than that of B*1blocks, the advection time also get smaller with the increment of the number of grid and blocks, however when the grid size larger than 50, in 1D parallelisation methods, it cause program to run a great long time,this may due to the deficiency that i directly map the whole location of nodes to a 2d matrix .

2D parallelisation: The performance of 2d decomposition is far greater than the 1d version, with the increment of the block and grid numbers ,the advection time to computation get smaller, however when the grid and block size greater than 500, it seems cost program more time to calculate, since each grid has 512 blocks and each block has 512 threads, here i set block and thread to be 512 for the test, the performance didn't seem good

Speedup against GPU and host :
To test the result here i choose the optimal gx,gy,bx,by for the case of GPU

Host time ./testAdvect -h 2000 2000 100
Advection of a 2000x2000 global field on host for 100 steps.
Advection time 2.622e+00, GFLOPs rate=3.51e+00
Average error of final field: 6.806e-07

single GPU time /testAdvect -g 200,200 -b 200,200 2000 2000 100
Advection of a 2000x2000 global field on GPU for 100 steps.
        using 200x200 blocks of 200x200 threads (1D decomposition)
Advection time 3.325e-01, GFLOPs rate=2.77e+01
Average error of final field: 7.693e-02

]

The speed up of multiple GPU vs host:
Speed up = 2.622e+00/3.325e-01 = 7.885
The ARM core i used is core-i7 6700 ,which is a supersonic speed CPU, the effect may not satisfying enough, but the speed up still 7 times than that of GPU.

Question 7

The MPI, openMP, CUDA are 3 quite different platform to perform the parallel job, from my experience during this 2 assignments and all labs we've attended, the MPI is easier to understand but hard to implement, the shared memory platform(cuda,openmp,pthreads) are hard to understand but easy to implement, in shared memory platform, we need to consider the synchronisation, cache coherency and a number of problem, but MPI seems less stressed to do that , only focus on message passing won't cause deadlock issue should be safe enough. However when we implement it, the mpi model, especially for multi dimension simulation or stencil computations will be harder than i think due to the latency or rendezvous ,

The following is the comparison of the advection server of my implementations in MPI, OPENMP, and CUDA. The test case is 3 parallel programming model under the circumstance of 2d decomposition

MPI: M=N=2000 r = 100 , Process number is 8
[yw8443@raijin6 r]$ mpirun -np 8 testAdvect -P 2 2000 2000 100
mpirun: ERROR: testAdvect does not exist in $PATH
Advection of a 2000x2000 global field over 2x4 processes for 100 steps.
Advection time 1.66e+00 sec, GFLOPs rate=5.44e+00 (per core 1.361e+00)
Average error of final field: 7.731e-07

OpenMP M=N=2000 r = 100 , Process number is 8
[yw8443@raijin3 openmp]$ perf stat -e cache-references,cache-misses numactl --cpunodebind=0 --membind=0 ./testAdvect 2000 2000 100
Advection of a 2000x2000 global field on 8 threads for 100 steps.
dx=5.000000e-04 dy=5.000000e-04 dt=1.250000e-04
Advection time 2.441e+01, GFLOPs rate=3.77e-01 (per core 4.71e-02)
Average error of final field: 1.132e+222

CUDA M=N=2000 r = 100 , Process number is 200^4
 /testAdvect -g 200,200 -b 200,200 2000 2000 100
Advection of a 2000x2000 global field on GPU for 100 steps.
        using 200x200 blocks of 200x200 threads (1D decomposition)
Advection time 3.325e-01, GFLOPs rate=2.77e+01
Average error of final field: 7.693e-02

Speed up comparison
In the term of MPI
MPI vs OPENMP = 14 ; MPI vs Cuda = 0.203

In the term of CUDA
CUDA VS MPI = 0.049, CUDA vs OpenMP = 0.73

In the term of OPENMP
OPENMP vs CUDA = 3.03 , OpenMP vs MPI = 0.068

From the comparison we can see that the CUDA model has the best performance considering the other model's speedup for it is less than 1.

SUMMARY

This two assignments are fantastically combined with the common 3 parallel programming model, with the normal problem of advection simulation to let us have a practical experience that how the actual problems can be solved under the multiple processor, the lab is also closely combined with the requirements of the assignment, the questions in assignment is also distributed fluently so that students can solve the problem by steps.