Parallel Programming HomeWork2

Ansh Prakash
2016CS10367

Problem1

(a) For the case when the schedule is static and N=M.The code is parallelization safe. The reason being that threads will be assigned the same iteration space in both the loop, this is guaranteed by OpenMP.

For N!=M and schedule being dynamic. The code is not parallelization safe.The reason is that different thread will be assigned a different iteration space and will lead to race.

(b)

Here we can see the loop dependency(arrow downwards), which implies that I can safely parallelise among columns but not rows.

|/	|/	|/	|/	|/	|/	|/	|/	|/	|/	|/	|/	|/	|/	|/
|/	|/	|/	|/	.	.	.								
|/	|/	|/	|/	.	.	.								
|/	|/	.	|/	.	.	.								
|/	|/	.	.											
|/	|/		.											
|/	|/		.											
|/	|/													
|/	|/													
|/	|/													
|/	|/													


```
#pragma omp parallel for schedule(static)
for(int j=0;j<n;j++){
        for(int i=1;i<n;i++){
                a[i∗n+j]+=a[(i−1)∗n+j];
        }
}
```

(c)

In foo the critical section and in bar the atomic section i.e line 4 and line 13 ,respectively can lead to data races.

The solution is to

Change the foo code:

```
//locatedinsideaparallelregioninfunctionfoo
#pragma omp atomic
{
        a++;//Modifyaexclusively
}
//locatedinsideaparallelregioninfunctionbar
#pragma omp atomic
{
        b++;//Modifybexclusively
}
#pragma omp atomic
{
        a=a+b;//Modifyaexclusively
}
```

Problem2

(a)

Dependencies:

No data dapendency in any loop.

The outer loop is ideally suited for parallelisation than inner loop because if we parallelise the inner loop it will create more overhead due to the creation and joining of threads for all i.

(b)

Code:

Compile:g++ P2.cpp -fopenmp

```
#include <iostream>
#include <vector>
#include <omp.h>
#include <random>
#include <algorithm>
#include <iterator>
#include <functional>

using namespace std;

void seqsort(std::vector<unsigned int> &X){
    unsigned int i,j,count,N=X.size();
    std::vector<unsigned int> tmp(N);
    for(i=0;i<N;i++){
      count=0;
      for(j=0;j<N;j++)
       if(X[j]<X[i]||X[j]==X[i]&&j<i)
        count++;
      tmp[count]=X[i];
    }
    std::copy(tmp.begin(),tmp.end(),X.begin());
}

void parsort(std::vector<unsigned int> &X){
    unsigned int i,j,count,N=X.size();
    std::vector<unsigned int> tmp(N);

    #pragma omp parallel for schedule(static) shared(tmp) private(j,count) num_threads(4)
    for(i=0;i<N;i++){
      count=0;
      for(j=0;j<N;j++)
       if(X[j]<X[i]||X[j]==X[i]&&j<i)
        count++;
      tmp[count]=X[i];
    }
    std::copy(tmp.begin(),tmp.end(),X.begin());
}

int main(int argc,char const*argv[]){

            random_device rnd_device;
    // Specify the engine and distribution.
    mt19937 mersenne_engine {rnd_device()};  // Generates random integers
```

```
    uniform_int_distribution<int> dist {1, 1000};

    auto gen = [&dist, &mersenne_engine](){
                return dist(mersenne_engine);
              };

            std::vector< unsigned int> vec(10000);
            generate(begin(vec), end(vec), gen);
            std::vector<unsigned int> vec2=vec;
            for (auto i = vec.begin(); i != vec.end(); ++i)
            std::cout << *i << ' ';
    std::cout<<"\n";
    double start_time = omp_get_wtime();
            seqsort(vec);
            double time1 = omp_get_wtime() - start_time;
            start_time = omp_get_wtime();
            parsort(vec2);
            double time2 = omp_get_wtime() - start_time;

            for (auto i = vec2.begin(); i != vec2.end(); ++i)
            std::cout << *i << ' ';
    std::cout<<"\n";
    std::cout<<"Time taken by seqsort "<<time1<<endl;
    std::cout<<"Time taken by parsort "<<time2<<endl;
            return(0);
}
```

Threads	2	4
Speed Up	1.62	1.95
Efficiency	0.81	0.48

Problem3

(a)

The code is actually trying to calculate the sum and uses log(n) num of steps as the result half of the threads become ideal at each ‘s’ doubling
The problem with the code is the if condition which made most of the thread ideling, instead try to replace it,with
for (unsigned int s=blockDim.x/2; s>0; s=s/2)
{

```
        sdata[tid] += sdata[tid + s];
        __syncthreads();
}
```

According to me __syncthread cannot be removed as all the thread need to syncronized till this point.

(b)
Ways to avoid Data Race are:
__syncthread: act as a barrier
atomicAdd(), atomicSub(),atomicMin(),atomicMax()
Lock,mutex

__global__ :this means the function will be called from the host and will run of the device,i.e, the GPU
__device__: Function call from the device and run on the device
__host__: function called from the host and executed on the host

Problem4

Future and Promises provide a way to do asynchronous programming by the means of using signals.
Async creates another thread (depends on parameter may work with a single thread) to launch the function provided in the parameter.
The class template std::promise provides a facility to store a value or an exception that is later acquired asynchronously via an std::future object created by the std::promise object.
Basically, we are creating a communication channel between threads to share the values required by a thread when generated.

Example Code:

Compile:g++ P4.cpp -pthread

```
#include <future>
#include <iostream>
#include <unistd.h>

using namespace std;


int Sum(future<int> &f){
	int sum=0;
	int n=f.get();
	for(int i=1;i<n+1;i++){
		sum+=i;
	}
	cout<<"Result of Thread "<<sum<<endl;
	cout<<"I have thread id "<<this_thread::get_id()<<endl;
	return(sum);
}

int main(int argc, char const *argv[])
{

	int x;
	promise<int> p;//Promises are container of future
	future<int> f=p.get_future();
	future<int> fu=async(launch::async,Sum,ref(f));//Sum function launch aynchronously in a
separate thread(launch::async )
	cout<<"Hello main thread here\n";
	cout<<"I have thread id "<<this_thread::get_id()<<"\n\n\n\n";
	//Do some Work to give the value to Sum function
	sleep(2);//doing some work 2s required (lot of work)
	int val=100;//Ok Now I need the SUm of 1st 100 element
	p.set_value(val);//As promised a value is given to p
	//fu.get() will wait till the time the "Sum " exceution is complete,i.e,it is blocking
	x=fu.get();//getting the returned value by thread2

	return 0;
}
```