ACM40640 High Performance Comp Assignment 1

Ian Towey

04128591

June 29, 2018

Abstract

OpenMP code analysis

Contents

0 Matrix-Matrix Multiplication **2**

1 Area under a curve **7**

2 Computing π using Monte Carlo methods **8**

0 Matrix-Matrix Multiplication

This section analyses matrix mutliplcation performance using OpenMP on the fionn HPC at ICHEC.

The OpenMP pragma to parallelise this program is

```
#pragma omp parallel for schedule(dynamic,50) collapse(2)
private(i,j,k) shared(a,b,c)
for(i = 0;i < n; i++){
    for(j = 0;j < n; j++){
        for(k = 0;k < n; k++){
            c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
    }
}
end = omp_get_wtime();
```

After logging onto To initialise environment run

```
$ cd /ichec/home/users/ph5xx10/PH504/assignment1/code/q1
$ . setenv
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
```

Script compile_run.sh takes two parameters in the following order

- Matrix Dimension (integer)
- Number of runs (integer) - this is used to run the matrix multiplcation operation N times and then average run time is reported (total_time_for_N_runs/ N)

The script compiles the program main.c and runs the matrix multiplcation program for 1, 5, 10, 15, 20, 25, 30, 35, 40 threads, Below is a sample run and output from local laptop with 8 cores

```
 $ sh compile_run.sh 500 1
**********************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
**********************************************************************
1,500,1,0.158027
**********************************************************************
**********************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
**********************************************************************
5,500,1,0.046936
**********************************************************************
**********************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
**********************************************************************
10,500,1,0.043478
**********************************************************************
**********************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
**********************************************************************
```

```
15,500,1,0.042990
****************************************************************
****************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
****************************************************************
20,500,1,0.049902
****************************************************************
****************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
****************************************************************
25,500,1,0.042828
****************************************************************
****************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
****************************************************************
30,500,1,0.044276
****************************************************************
****************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
****************************************************************
35,500,1,0.043903
****************************************************************
****************************************************************
*thread count, Matrix Dimension, Number of runs, Avg run time
****************************************************************
40,500,1,0.042143
****************************************************************
```

Executing the script on fionn for various dim sizes

```
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ sh compile_run.sh 500 1
```

Matrix Dim : 500		
No. Threads	**Time Taken (s)**	**Relative speedup**
1	0.136818	1
5	0.032769	4.17522658610272
10	0.03897	3.5108545034642
15	0.061737	2.21614266971184
20	0.109617	1.24814581679849
25	0.094764	1.44377611751298
30	0.085122	1.60731655741172
35	0.074294	1.841575362748
40	0.061368	2.22946812671099

```
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ sh compile_run.sh 1000 1
```

Matrix Dim : 1000		
No. Threads	**Time Taken (s)**	**Relative speedup**
1	8.32384	1
5	1.789278	4.6520663641983
10	0.98218	8.47486204158097
15	0.670694	12.4107864391213
20	0.627197	13.2714920511418
25	0.687821	12.1017532177703
30	0.638659	13.0333088549602
35	0.546503	15.2310966270999
40	0.499063	16.6789363266762

```
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ sh compile_run.sh 2000 1
```

Matrix Dim : 2000		
No. Threads	**Time Taken (s)**	**Relative speedup**
1	81.919144	1
5	17.612401	4.65121955831008
10	9.42893	8.68806365091267
15	6.309168	12.9841437095985
20	4.834022	16.9463738476987
25	5.673005	14.4401677770423
30	4.915481	16.6655397508403
35	4.304687	19.0302207802797
40	4.444649	18.4309591151067

```
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ sh compile_run.sh 3000 1
```

Matrix Dim : 3000		
No. Threads	**Time Taken (s)**	**Relative speedup**
1	379.905421	1
5	72.595633	5.23317182178162
10	38.912887	9.76297186584999
15	26.133908	14.5368775691718
20	19.820675	19.1671283142476
25	21.94133	17.3146031256993
30	19.670081	19.3138717120687
35	17.172593	22.1227755761754
40	17.271112	21.9965814013597

```
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ sh compile_run.sh 4000 1
```

Matrix Dim : 4000		
No. Threads	**Time Taken (s)**	**Relative speedup**
1	732.608332	1
5	183.152083	4
10	102.377162	7.15597422010976
15	67.713204	10.8192832228113
20	51.827723	14.1354527961801
25	55.588827	13.1790572231359
30	50.389406	14.5389356643736
35	43.283283	16.9258956627666
40	42.753439	17.1356585373167

```
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
$ sh compile_run.sh 5000 1
```

Matrix Dim : 5000		
No. Threads	**Time Taken (s)**	**Relative speedup**
1	1548.988168	1
5	387.247042	4
10	209.308222	7.40051276151015
15	141.228929	10.9679240575421
20	107.211277	14.4479966225941
25	111.409326	13.9035772283552
30	105.991707	14.6142392819468
35	97.041795	15.9620725070059
40	93.206691	16.618851623002

Below chart shows almost linear speed up of the matrix maltuiplication using OpenMP pragmas up to about 20 threads, then the benefit of mutlithread processing deminishes significately

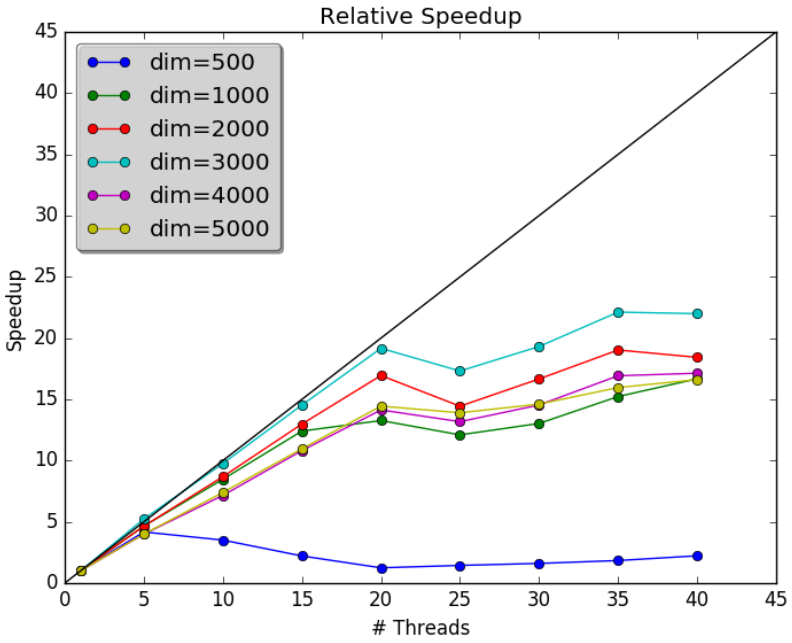


Figure 1: Relative Speed

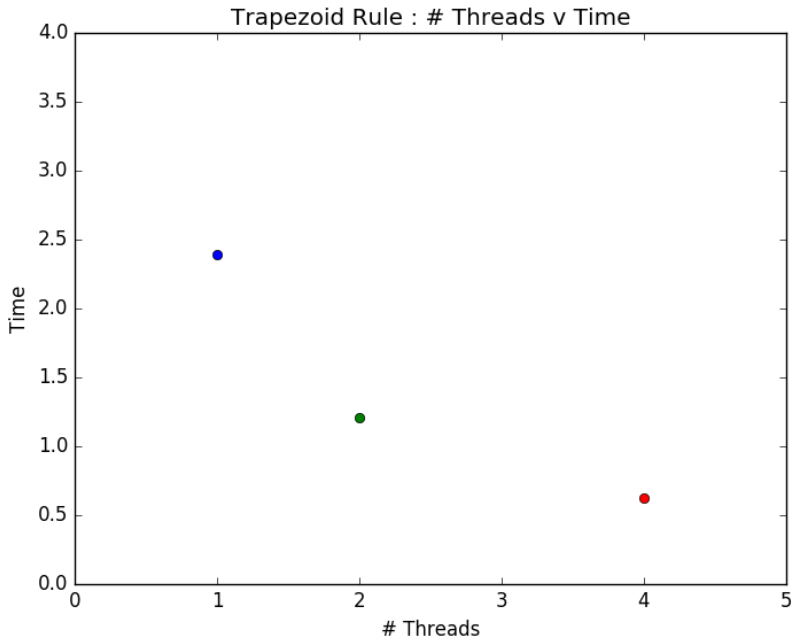
1 Area under a curve

Calculating Area under a curve using the trapezoid rules and OpenMP

```
$ cd /ichec/home/users/ph5xx10/PH504/assignment1/code/q2
$ . setenv
$ qsub -I -l walltime=00:20:00 -l nodes=3:ppn=24 -N mma -j oe -r n -A ph5xx
```

Example output of running

```
$ sh compile_run.sh
************************************************************
*Threads count ,Runs, Avg_loop_time/Runs, Computed area
************************************************************
1, 1, 2.39603304862976074219, 6.41113026187355483643
************************************************************
************************************************************
*Threads count ,Runs, Avg_loop_time/Runs, Computed area
************************************************************
2, 1, 1.21004486083984375000, 6.41113026188736423450
************************************************************
************************************************************
*Threads count ,Runs, Avg_loop_time/Runs, Computed area
************************************************************
4, 1, 0.62412595748901367188, 6.41113026188496260005
************************************************************
```



2 Computing π using Monte Carlo methods

1. A Crtical section is identified to limit the impact of the threads on shared variable

```
  count=0;
  #pragma omp parallel for ordered private(i) shared(count,x,y,z,
    seed)
  for(i=1;i<=niter;i++){
    #pragma omp critical
    {
        seed=i;
        rand_tuple *ran_val1 = ran3(seed);
        x = ran_val1->val;
        rand_tuple *ran_val2=ran3(ran_val1->seed);
        y = ran_val2->val;
        free(ran_val1);free(ran_val2);
        z=x*x+y*y;
        if(z<1){
          count+=1;
        }
    }
  }
  pi=count*4.0/niter;
```

2. The function ran3 is the thread safe implementation of ran2, it takes a seed as parameter , but returns a struct of the mutated seed and random value.

```
void main_q1(){

  int niter, i, j;
  long seed;
  double count;
  double x,y,z,pi;
  extern float ran2();

  niter=10000;
  count=0;
  #pragma omp parallel for ordered private(i) shared(count,x,y,z,
    seed)
  for(i=1;i<=niter;i++){
    #pragma omp critical
    {
        seed=i;
        x=ran2(&seed);
        y=ran2(&seed);
        z=x*x+y*y;
        if(z<1){
          count+=1;
        }
    }
  }
  pi=count*4.0/niter;
  printf("The value of pi is %8.14f\n",pi);
}
```

3.

4.