ACM40640 High Performance Comp Assignment 1

Ian Towey

04128591

June 24, 2018

Abstract

OpenMP code analysis

Contents

0 Matrix-Matrix Multiplication

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

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

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

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

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

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

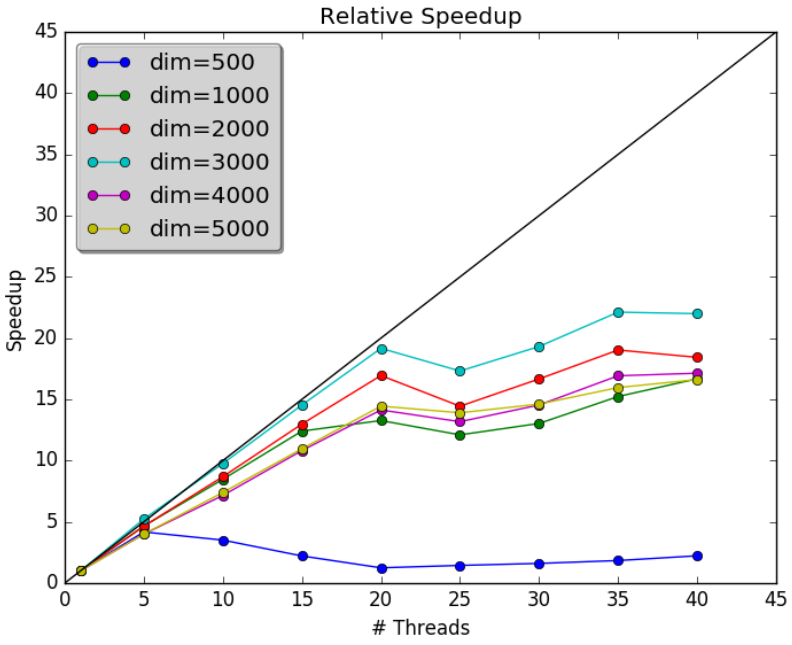

Figure 1: Relative Speed

1 Q2

2 Q3

2.1 Parallelize code using synchronous directives

```
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);
}
```

2.2 Ran2 thread safe

```
#ifndef MAIN_H_INCLUDED
#define MAIN_H_INCLUDED

#include <stdlib.h>
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

typedef struct {
    long seed;
    float val;
} rand_tuple;


#endif // MAIN_H_INCLUDED
```

```
#include <main.h>

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
```

```
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

rand_tuple *ran3(long seed) {
  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  float temp;
  rand_tuple *resp = (rand_tuple*)malloc(sizeof(rand_tuple));
  resp->seed = seed;
  resp->val = 0;
  if (resp->seed <= 0) {
    if (-(resp->seed) < 1) resp->seed=1;
    else resp->seed = -(resp->seed); idum2=(resp->seed);
    for (j=NTAB+7;j>=0;j--) {
       k=(resp->seed)/IQ1;
     resp->seed=IA1*(resp->seed-k*IQ1)-k*IR1;
      if (resp->seed < 0) resp->seed += IM1;
      if (j < NTAB) iv[j] = resp->seed;
    }
    iy=iv[0];

  }
  k=(resp->seed)/IQ1;
  resp->seed=IA1*(resp->seed-k*IQ1)-k*IR1;
  if (resp->seed < 0) resp->seed += IM1;
  k=idum2/IQ2;
  idum2=IA2*(idum2-k*IQ2)-k*IR2;
  if (idum2 < 0) idum2 += IM2;
  j=iy/NDIV;
  iy=iv[j]-idum2;
  iv[j] = resp->seed;
  if (iy < 1) iy += IMM1;

  if ((temp=AM*iy) > RNMX) {
    resp->val = RNMX;
  }
  else {
    resp->val = temp;
  }

  return resp ;

}
```

```

void  main_q2(){
  int niter, i, j;
  long seed;
  double count;
  double x,y,z,pi;
  extern rand_tuple *ran3();

  niter=10000;
  count=0;
  #pragma omp parallel for ordered private(i) shared(count,x,y,z,
    seed)
  for(i=1;i<=niter;i++){
```

```
13      #pragma omp critical
14      {
15          seed=i;
16          rand_tuple *ran_val1 = ran3(seed);
17          x = ran_val1->val;
18          rand_tuple *ran_val2=ran3(ran_val1->seed);
19          y = ran_val2->val;
20          free(ran_val1);free(ran_val2);
21          z=x*x+y*y;
22          if(z<1){
23            count+=1;
24          }
25      }
26    }
27    pi=count*4.0/niter;
28    printf("The value of pi is %8.14f\n",pi);
29  }
```

Appendices