CSC 453 introduction to parallel processing

Term Project

Due date: 6-12-2020 (before 11:00 PM)

You may work as groups of 5 students please make sure to work as group with full understanding. You will have one project report for the group. Also, Each student will submit a one page short report concluding the findings, the limitations, and how to enhance the code in the future.

In this project you need to write a parallel program using OpenMP that calculates the multiplication of 2 matrices A and B of size MxM. You may use the code in appendix A as a guideline. Run the code using p= 1,2,3,4 processors (threads) and with M=100, 200, 500, 1000, 2000 print the results in a table as below. Test your program on small matrix (M=3) to make sure the multiplication is correct before running for large size matrix to measure the time.

RUN#	p	M	Time1	Time2	Time3	Time4	Time5	Avg Time
1	1	100						
2	2	100						
3	3	100						
4	4	100						
5	1	200						
6	2	200						
7	3	200						
8	4	200						
9	1	500						
10	2	500						
11	3	500						
12	4	500						
13	1	1000						
14	2	1000						
15	3	1000						
16	4	1000						
17	1	2000						
18	2	2000						
19	3	2000						
20	4	2000						

You need to run each case multiple times to take a representative average. When measuring the time make sure that machine has only the parallel program running to reduce variations in run time measurement.

Also find the speedup and efficiency for each run and list it down in a table

RUN#	p	M	T1	Tp	Speedup	Efficiency
1	1	100				
2	2	100				
3	3	100				
4	4	100				
5	1	200				
6	2	200				
7	3	200				
8	4	200				
9	1	500				
10	2	500				
11	3	500				
12	4	500				
13	1	1000				
14	2	1000				
15	3	1000				
16	4	1000				
17	1	2000				
18	2	2000				
19	3	2000				
20	4	2000				

Submit your report that include the following:

1. **Cover page:** Align the text that contains your names & IDs, Course name and code, "Term Project" in the center.
2. **Introduction:** What is the problem to be solved? And how you will solve it?
3. **Body:** The body of your report should contain the main points from your work. Provide information about the **hardware used** including processor type, number of cores, cache size, etc. Provide information about the code and how it was run in parallel, what OpenMP statements used and why, so that the reader can further understand your parallel code. What are **the limitations** of the program? How you measured the time. How many runs you made to take the average? Include the sample output of your program for different cases with important findings like **serial and parallel run time, the speedup, and efficiency.**

Use **graphs and table** to explain how the performance changes as the number of processors and the size of the problem change. You may include graphs that show time vs number of processors (threads), speedup vs number of processors (threads).

4. **Conclusion:** End with a summary of the results and findings. Was the parallel code useful to enhance the execution time? If you have chance in the future what changes you may do for your code? Why?
5. **Appendices:** Add appendices that include your coding and results
    a. Appendix A: Your parallel code.
    b. Appendix B: The sample output (please print the time not the matrix).
    c. Appendix C: Tables of the results of all runs.

The group will have one full report and <u>each student must write a one page conclusion report that explains his own findings and suggested future work</u>.

Appendix A.

```
# include <stdlib.h>

# include <stdio.h>

# include <math.h>

# include <omp.h>


int main ( int argc, char *argv[] );

void r8_mxm ( int l, int m, int n );

double r8_uniform_01 ( int *seed );


/******************************************************************
***********/


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


/******************************************************************
***********/

/*

  Purpose:


    MAIN is the main program for MXM.


  Licensing:
    This code is distributed under the GNU LGPL license.

  Modified:

    19 April 2009

  Author:

    John Burkardt
*/
{
  int id;
  int l;
  int m;
  int n;

  printf ( "\n" );
  printf ( "MXM\n" );
  printf ( "  C/OpenMP version.\n" );
  printf ( "\n" );
  printf ( "  Matrix multiplication tests.\n" );

  printf ( "\n" );
  printf ( "  Number of processors available = %d\n",
omp_get_num_procs ( ) );

  printf ( "  Number of threads =              %d\n",
omp_get_max_threads ( ) );



  l = 500;

  m = 500;

  n = 500;



  r8_mxm ( l, m, n );

/*

  Terminate.

*/

  printf ( "\n" );

  printf ( "MXM:\n" );

  printf ( "  Normal end of execution.\n" );



  return 0;

}

/******************************************************************
***********/



void r8_mxm ( int l, int m, int n )
```

```
/******************************************************************
***********/

/*

  Purpose:


    R8_MXM carries out a matrix-matrix multiplication in R8
arithmetic.


  Discussion:


    A(LxN) = B(LxM) * C(MxN).


  Licensing:


    This code is distributed under the GNU LGPL license.


  Modified:


    13 February 2008


  Author:


    John Burkardt
  Parameters:

    Input, int L, M, N, the dimensions that specify the sizes of the
    A, B, and C matrices.
*/
{
  double *a;
  double *b;
  double *c;
  int i;
  int j;
  int k;
  int ops;
  double rate;
  int seed;
  double time_begin;
  double time_elapsed;
  double time_stop;
/*
  Allocate the matrices.
*/
  a = ( double * ) malloc ( l * n * sizeof ( double ) );
  b = ( double * ) malloc ( l * m * sizeof ( double ) );
  c = ( double * ) malloc ( m * n * sizeof ( double ) );
/*
  Assign values to the B and C matrices.
*/
  seed = 123456789;

  for ( k = 0; k < l * m; k++ )
  {
    b[k] = r8_uniform_01 ( &seed );
  }

  for ( k = 0; k < m * n; k++ )
  {
    c[k] = r8_uniform_01 ( &seed );
  }
/*
  Compute A = B * C.
*/
  time_begin = omp_get_wtime ( );

# pragma omp parallel \
  shared ( a, b, c, l, m, n ) \
  private ( i, j, k )
```

```
# pragma omp for
  for ( j = 0; j < n; j++)
  {
    for ( i = 0; i < l; i++ )
    {
      a[i+j*l] = 0.0;
      for ( k = 0; k < m; k++ )
      {
        a[i+j*l] = a[i+j*l] + b[i+k*l] * c[k+j*m];
      }
    }
  }
  time_stop = omp_get_wtime ( );
/*
  Report.
*/
  ops = l * n * ( 2 * m );
  time_elapsed = time_stop - time_begin;
  rate = ( double ) ( ops ) / time_elapsed / 1000000.0;

  printf ( "\n" );
  printf ( "R8_MXM matrix multiplication timing.\n" );
  printf ( "  A(LxN) = B(LxM) * C(MxN).\n" );
  printf ( "  L = %d\n", l );
  printf ( "  M = %d\n", m );
  printf ( "  N = %d\n", n );
  printf ( "  Floating point OPS roughly %d\n", ops );
  printf ( "  Elapsed time dT = %f\n", time_elapsed );
  printf ( "  Rate = MegaOPS/dT = %f\n", rate );

  free ( a );
  free ( b );
  free ( c );

  return;
}
/******************************************************************
***********/

double r8_uniform_01 ( int *seed )

/******************************************************************
***********/
/*
  Purpose:
    R8_UNIFORM_01 is a unit pseudorandom R8.

  Discussion:

    This routine implements the recursion

      seed = 16807 * seed mod ( 2**31 - 1 )
      unif = seed / ( 2**31 - 1 )

    The integer arithmetic never requires more than 32 bits,
    including a sign bit.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    11 August 2004

  Author:

    John Burkardt
  Reference:

    Paul Bratley, Bennett Fox, Linus Schrage,
    A Guide to Simulation,
    Springer Verlag, pages 201-202, 1983.

    Bennett Fox,
    Algorithm 647:
    Implementation and Relative Efficiency of Quasirandom
    Sequence Generators,
    ACM Transactions on Mathematical Software,
    Volume 12, Number 4, pages 362-376, 1986.

  Parameters:

    Input/output, int *SEED, a seed for the random number generator.

    Output, double R8_UNIFORM_01, a new pseudorandom variate,
strictly between
    0 and 1.
*/
{
  int k;

  double r;


  k = *seed / 127773;


  *seed = 16807 * ( *seed - k * 127773 ) - k * 2836;


  if ( *seed < 0 )

  {

    *seed = *seed + 2147483647;

  }


  r = ( double ) ( *seed ) * 4.656612875E-10;


  return r;

}
```

Appendix B.

The following links may help you more to solve the problem and write a good report:

https://medium.com/tech-vision/parallel-matrix-multiplication-c-parallel-processing-5e3aadb36f27

https://www.appentra.com/parallel-matrix-matrix-multiplication/

https://people.sc.fsu.edu/~jburkardt/c_src/mxm_openmp/mxm_openmp.html