2017年春季 – 并行与分布式计算导论

HOMEWORK 1

Assigned: 04/16/2017, Due: 05/07/2017
Instructor: 罗国杰 (gluo@pku.edu.cn)

Name: ________________
UID: ________________
Email: ________________

Problem 1 - Matrix Multiplication

Task: Parallelize MatrixMultiply.c by OpenMP, and submit the parallelized source code and the report, together with a short report about the execution time versus different number of threads and different sizes.

Instructions:

1) Use the matrix generator, gen.c, create two input matrices (matrix_a and matrix_b).

2) Compile the MatrixMultiply.c with OpenMP pragmas to parallelize the *matrix_multiply()* function.

3) Verify your results and make sure they are correct.

4) Execute the program with 1, 2, 3, 4 or more threads, and pay attention to the difference in the execution time.

5) Report the speedup and efficiency you achieved after comment out the printing statements.

Note: You can write your own matrix multiplication codes, but you should submit this code and the code parallelized by OpenMP.

Problem 2 – Find Prime

Task: Parallelize the source code below by OpenMP, and submit the parallelized source code, together with a short report about the execution time versus different number of threads and different problem sizes.

Note: You can write your own find prime codes, but you should submit this code and the code parallelized by OpenMP.

```c
/*
 *   This program uses the Sieve of Eratosthenes to determine the
 *   number of prime numbers less than or equal to 'n'.
 *
 *   Adapted from code appearing in "Parallel Programming in C with
 *   MPI and OpenMP," by Michael J. Quinn, McGraw-Hill (2004).
 */

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

int main (int argc, char *argv[])
{
   int    count;        /* Prime count */
   int    first;        /* Index of first multiple */
   int    i;
   int    index;        /* Index of current prime */
   char  *marked;       /* Marks for 2,...,'n' */
   long long int    n;  /* Sieving from 2, ..., 'n' */
   long long int    N;  /* Size of sieve and loop bounds */
   int    prime;        /* Current prime */

   if (argc != 2) {
      printf ("Command line: %s <m>\n", argv[0]);
      exit (1);
   }
   n = atoi(argv[1]);
   N = n+1;

   marked = (char *) malloc (N);  //alocate slots for numbers in range [0,n]
   if (marked == NULL) {
      printf ("Cannot allocate enough memory\n");
      exit (1);
   }

   for (i = 0; i < N; i++) marked[i] = 1;
   marked[0] = 0;
   marked[1] = 0; // not primes
   index = 2;
   prime = 2;
   do {
      first = 2 * prime;
      for (i = first; i < N; i += prime) marked[i] = 0;
      while (!marked[++index]) ;
      prime = index;
   } while (prime * prime <= n);

   count = 0;
   for (i = 0; i < N; i++)
      count += marked[i];
   printf ("\nThere are %d primes less than or equal to %d\n\n", count, n);
   return 0;
}
```

Attention: You can run your program on the server 222.29.98.17, which has two 10-core CPUs. Username: letter “s” + your university id, password: 123.