CS 475/575 -- Spring Quarter 2022

Project #0

Simple OpenMP Experiment

30 Points

Due: April 4

This page was last updated: March 22, 2022

Introduction

A great use for parallel programming is identical operations on large arrays of numbers.

Requirements

1. Pick an array size to do the arithmetic on. Something like 16384 (16K) will do. Don't pick something too huge, as your machine may not allow you to use that much memory. Don't pick something too small, as the overhead of using threading might dominate the parallelism gains.

2. Using OpenMP, pairwise multiply two large floating-point arrays, putting the results in another array. Do this in a for-loop.
   `C[ i ] = A[ i ] * B[ i ];`

3. Do this for one thread and do this for four threads:
   #define NUMT 1
   and
   #define NUMT 4

4. Use *omp_set_num_threads(NUMT);* to set the number of threads to use.

5. Time the two runs using two calls to *omp_get_wtime();*. Convert the timing results into "Mega-Multiplies per Second".

6. What speedup, S, are you seeing when you move from 1 thread to 4 threads?

   S = (Execution time with one thread) / (Execution time with four threads) = (Performance with four threads) / (Performance with one thread)

   This number should be greater than 1.0 . If not, be sure you are using the correct numerator and denominator.

7. If your 1-thread-to-4-threads speedup is **S**, compute the parallel fraction:

```
float Fp = (4./3.)*( 1. - (1./S) );
```

Don't worry what this means just yet. This will become more meaningful soon.
You must use only the 1-thread-to-4-threads speedup with this equation. The numbers in this equation depend on that.

8. Your written commentary (turned in as a PDF file) should include:
    1. Tell what machine you ran this on
    2. What performance results did you get?
    3. What was your 4-thread-to-one-thread speedup?
    4. If the 4-thread-to-one-thread speedup is less than 4.0, why do you think it is this way?
    5. What was your Parallel Fraction, Fp?

The main Program

Your main program would then look something like this:

```
#include <omp.h>
#include <stdio.h>
#include <math.h>

#define NUMT             4      // number of threads to use
#define SIZE            ??      // array size -- you get to decide
#define NUMTRIES        ??      // how many times to run the timing -- you get to decide

float A[SIZE];
float B[SIZE];
float C[SIZE];

int
main( )
{
#ifndef _OPENMP
        fprintf( stderr, "OpenMP is not supported here -- sorry.\n" );
        return 1;
#endif

        // inialize the arrays:
        for( int i = 0; i < SIZE; i++ )
        {
                A[i] = 1.;
                B[i] = 2.;
        }

        omp_set_num_threads( NUMT );
        fprintf( stderr, "Using %d threads\n", NUMT );

        double maxMegaMults = 0.;

        for( int t = 0; t < NUMTRIES; t++ )
        {
                double time0 = omp_get_wtime( );

                #pragma omp parallel for
                for( int i = 0; i < SIZE; i++ )
                {
                        C[i] = A[i] * B[i];
                }
```

```
                double time1 = omp_get_wtime( );
                double megaMults = (double)SIZE/(time1-time0)/1000000.;
                if( megaMults > maxMegaMults )
                        maxMegaMults = megaMults;
        }

        printf( "Peak Performance = %8.2lf MegaMults/Sec\n", maxMegaMults );

        // note: %lf stands for "long float", which is how printf prints a "double"
        //        %d stands for "decimal integer", not "double"

        return 0;
}
```

Grading:

Feature	Points
Performance or Execution time results for 1 thread	5
Performance or Execution time results for 4 threads	5
One-thread-to-four-threads Speedup (>1.)	5
Parallel Fraction	10
Commentary	5
Potential Total	**30**