CS 475/575 -- Spring Quarter 2025

Project #0

Simple OpenMP Experiment

30 Points

Due: April 8

This page was last updated: March 12, 2025

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. Be sure to use the **`#pragma omp parallel for`** as the line before the for-loop.

3. Pairwise multiplication means this: `C[ i ] = A[ i ] * B[ i ];`

4. Do this twice, once for one thread and once for four threads:
   #define NUMT 1
   and
   #define NUMT 4

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

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

7. 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.

8. If your 1-thread-to-4-threads speedup is **S**, then use this equation to 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.**

9. 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 1-thread-to-4-thread speedup?
    4. Your 1-thread-to-4-thread speedup should be less than 4.0. Why do you think it is this way?
    5. What was your Parallel Fraction, Fp? (Hint: it should be less than 1.0, but not much less.)

The main Program

This is your complete main program. All you have to do is compile and run it.

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

#ifndef NUMT
#define NUMT            1      // number of threads to use -- do once for 1 and once for 4
#endif

#ifndef SIZE
#define SIZE            ??      // array size -- you get to decide
#endif

#define NUMTRIES        20      // how many times to run the timing to get reliable timing data

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

int
main( )
{
#ifdef  _OPENMP
        fprintf( stderr, "OpenMP version %d is supported here\n", _OPENMP );
#else
        fprintf( stderr, "OpenMP is not supported here - sorry!\n" );
        exit( 0 );
#endif

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

        omp_set_num_threads( 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;
```

```
	}

	fprintf( stderr, "For %d threads, Peak Performance = %8.2lf MegaMults/Sec\n", NUMT, maxMegaMults );

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

	// Speedup = (Peak performance for 4 threads) / (Peak performance for 1 thread)

	return 0;
}
```

Turn-In

Turn in your PDF file and your cpp file on Canvas. Go to the Canvas Week #1 or #2 modules, scroll down to the Projects, go to the Project #0 row and click on **Submit**. When you get the Project #0 Assignment page, click on the **Start Assignment** black button in the upper-right corner. Upload your files.

Grading:

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