Programming Assignment #1

This assignment is intended to familiarize you with our parallel programming environment, give you some notion of performance measurement, and provide an introductory look at the promise and difficulties of parallel programming.

Prime Number Generator

You are to take an existing uniprocessor algorithm (the source in `~bilas/class/pa1`) that determines prime numbers, and parallelize it in a number of different ways to run on the SGI multiprocessor. The given program works by testing each odd number (up to a specified limit) for divisibility by all the factors ranging from 3 up to the square root of that number. The algorithm is not very efficient, but it is easy to parallelize. Your goal is to make the speedup as large as possible using the existing algorithm. Do not change the underlying algorithm.

The program takes two parameters as input. The first argument is the number of processors to execute the program on, and the second argument specifies the range of numbers to check for primeness. The program uses an array to hold boolean values indicating whether the corresponding numbers are prime. The array only holds odd numbers since no even numbers are prime (except 2). The core code is executed 100 times to provide a reasonable execution time. **You should parallelize only the actual core code and not this outer loop!**

To compare performance, we would like you to implement two different versions of the parallel code. You should use `#defines` and `#ifdef`'s so that there is only one source file to maintain.

- Method 1: Dynamically scheduled loop - Each processor picks the loop indices that it will operate on at run time. This can be implemented using the ANL macros' distributed loop constructs (GETSUB).
- Method 2: Statically scheduled loop - Loop indices are determined by the programmer by writing the loop code such that the loop indices are divided "evenly" between the processors. Hint: You will need to consider what subdivision of the indices ensures good load balancing.

To evaluate the performance of the parallel program, measure the following times using the ANL `CLOCK()` macro.

- *Initialization Time*: This is the time required to do all of the data structure initialization, read the command line arguments, and create the separate processes. Start timing when the program starts. Have the first instruction in the created routine (which the main process also calls when it has finished creating other processes) be a BARRIER, so all processes start the parallel part of the program at the same time. End timing (for initialization time) just after this barrier.
- *Computation Time*: This is simply the time to compute the answer. Start timing when you ended the initialization timing (you don't need a new CLOCK call) and finish when all the answers have been calculated; i.e. after a BARRIER that you should insert at the end of the routine at which created processes start at. (Think about how you would measure and interpret time and speedup if you didn't use the barriers at the beginning and the end, and what complications this would cause.

Note that: *Total Time* = *Computation Time* + *Initialization Time*. Speedup is calculated as follows: T_1 / T_P where T_1 is the time for one processor, and T_P is the time for *P* processors. *Computation Speedup* uses only *Computation Time* in its calculations and *Total Speedup* uses the *Total Time*.

What to turn in:

The writeup for this assignment should contain the following information for both the static and dynamic case:

1. The number of primes found and the last prime number for N = 1000, 5000, 10000, and 20000.
2. A plot of total speedup and computation speedup versus the number of processors (P) for N = 1000, 5000, 10000, and 20000. Use P = 1, 2, 4, 8 and 12.
3. A plot of total speedup and computation speedup versus N for P = 8. Use N = 1000, 5000, 10000, and 20000.
4. A brief discussion of the results you expected and an explanation of factors leading to any non-ideal behavior you observed. Anything that surprised you or on which you'd like to comment.

Additionally, you should put your source code in a directory readable by the instructor. Include in your report a pointer to this directory.

BE SURE TO READ E-MAIL FOR LATE-BREAKING PROGRAM INFORMATION