This assignment has two objectives: first, to become familiar with writing shared-memory programs using POSIX threads (and similar interfaces); and to understand the performance impact of how task synchronization overheads affect the task granularity which can be efficiently supported. You will use the SGI Cray Origin 2000 (modi4.ncsa.uiuc.edu) for this assignment.

As with HW#3, you are allowed to do this assignment in groups (as long as each group has 3 or fewer students). I expect a single write-up per group. All group members will receive the same grade.

Particulars

For this assignment, you will write a shared-memory parallel program which outputs all prime numbers less than an input parameter $N$. The objective of course is to maximize the achieved speedup as much as possible. However, you are not expected to put in an extraordinary amount of effort to parallelize the code: this assignment is worth only 15 points of the overall course grade.

As stated before, you will use the pthreads library on the Origin 2000 (use only the C interface). Unlike HW#2 and HW#3, there is no extra-credit portion to this assignment.

You are free to develop your own prime-finding algorithm. However, to get you started, the following is a simple algorithm which works relatively well:

1. Maintain a central data structure which keeps track of the primes found so far. This could be a boolean array of size $N$: the value true indicates that something is a prime. You can prime this array as part of your initialization: i.e., fill it up with some initial prime numbers.

2. Each thread works on a partition of the interval from 0 to $N$: the size of the partition is one of the key parameters that will affect overall performance. The algorithm that each processor runs for its partition is described below:

   Test each odd number for divisibility by all the prime factors ranging from 3 up to the square root of that number. If the number is not divisible by any of them, then it is a prime and the thread ends up updating the central data structure.

   Note that the above algorithm implies some form of synchronization between updates to the central data structure and testing whether or not a particular number is a prime: you can test for divisibility only after ensuring that all primes up to the square root of the number have been found.

3. Output the primes found in the range 0 to $N$.

The challenging part of this assignment is the synchronization between updates to the central data structure and the divisibility testing of a particular number. An additional challenge arises from making sure that the work is balanced between threads. Note that the work a thread does to test divisibility of a number can vary based upon its factors. To fully appreciate the challenge of load balancing an irregular computation, you will develop two versions of the program:
• **Statically scheduled**: The partition that each processor will work on is decided at the start of the program. You will need to write a function which takes the number of processors, the range in which primes are to be computed, and the specific processor number and produces as output the partition (or set of partitions which the processor will work on). You are free to choose what the function does: the only restriction is that it cannot look up a table that you generate offline. As mentioned above, giving each processor a uniform-sized partition may not be the best thing to do.

• **Dynamically scheduled**: Each processor picks up the partition it will work on at run time. You will need to implement the appropriate synchronization to ensure that two threads do not work on the same partition.

Please verify that your program works with a small number of processors and a small data size before running it on larger input sizes and larger numbers of processors.

**Guidelines**

You will hand in the following:

1. **(3 points)** A code listing and (very) brief description of the sequential algorithm that you end up using.

2. **(5 points)** Modifications to the above listing showing how the code was parallelized using pthread constructs. This portion of the assignment should use static partitioning of work among the threads.

3. **(4 points)** Modifications to the above listing showing the dynamic partitioning of work among threads.

4. **(3 points)** The following information:

   (a) The number of primes found, and the last prime number for \(N = 1000, 5000, 10000,\) and \(20000.\)

   (b) A plot of total speedup (include the cost of initialization) versus the number of processors/threads for \(N = 1000, 5000, 10000, 20000\) and \(P = 1, 2, 4, 8, 12.\) As before, speedup must be computed with respect to the fastest sequential algorithm.

   (c) A plot of total speedup versus \(N\) for \(P=8.\) Use \(N = 1000, 5000, 10000,\) and \(20000.\)

   (d) Analyze your measurements and comment upon any surprises.

As with HW#3, there are several possible ways of writing this program (some of which will give you a slowdown). I suggest that you work on the simplest program that will produce a speedup. Most of the design for such a program (deciding the data structures, the partitioning, the synchronization structure, etc.) can actually be done before you write a single line of code.