Solutions to IIa and IIc

Problem IIa

A multiprocessor consists of 100 processors, each capable of a peak execution rate of 2 Gflops (i.e., 2 billion floating point operations per second). What is the peak performance of the system as measured in Gflops for an application where 10% of the code is sequential and 90% is parallelizable?

Let s = the sequential execution time. Then, the parallelizable portion is .9s. If we observe linear speedup in the parallel portion, then peak performance will be .1s + .9s /100 = .109s. If we were fully utilizing the 100 processors, execution time would be .01s. So parallel efficiency = .01/.109 = 9.17%. Peak performance = 200Gflops, and at 9.17% efficiency = 18.34 Gflops.

Problem IIc

Construct a task-parallel (similar to producer-consumer) pipelined code to identify the set of prime numbers in the sequence of integers from 1 to n. A common sequential solution to this problem is the sieve of Erasthones. In this method, a series of all integers is generated starting from 2. The first number, 2, is prime and kept. All multiples of 2 are deleted because they cannot be prime. This process is repeated with each remaining number, up until but not beyond sqrt(n). A possible sequential implementation of this solution is as follows:

```
for (i=2; i<=n; i++) {
  prime[i] = true;
for (i=2; i<= sqrt(n); i++) {
  if (prime[i]) {
     for (j=i+i; j<=n; j = j+i) { // multiples of i are set to non-prime
          prime[j] = false;
     }
}
```

The parallel code can operate on different values of i. First, a series of consecutive numbers is generated that feeds into the first pipeline stage. This stage eliminates all multiples of 2 and passes the remaining numbers onto the second stage, which eliminates all multiples of 3, etc. Although this is not the most efficient solution, assume that each pipeline stage sends a single number to the next stage. The parallel code terminates when the “terminator” element arrives at each pipeline stage.

Problem IIc, cont.

Instead of marking out individual elements multiple times, we only mark out ones that have not been marked previously. The following is an example of an acceptable solution.

```
Master:
create new task for 2-evaluator
for j = 2, N
    send j to 2-evaluator

Thread i-evaluator, private(first=1)
Receive num
if (num == "terminator") exit;
if (num%i != 0) {
  if (first) {
    next = task_create(num);
    first=0;
  }
  else send num to next-evaluator
}
Else prime(num) = false;
```

Problem IIc, cont.

Using “pseudo-OpenMP”

```
FirstTime= 1;
#pragma omp parallel{
#pragma omp single nowait private(flag){
#pragma omp task_create private(2)
for (j = 2; i<=N; i++)
      a[2] = I;
    #pragma omp flush
      flag = 1;
     #pragma omp flush(flag)
}

#pragma task firstprivate(FirstTime), private(flag)

//Thread i-evaluator, private(first=1)
 while (!flag)
      #pragma omp flush(flag)
  #pragma omp flush
 num=a[myid];
if (num == “terminator”) exit;
if (num%i != 0) {
  if (FirstTime) {
      #pragma omp task_create private(num)
     FirstTime=0;
  }
  else {
       a[nextid] = num;
       #pragma omp flush
      flag = 1;
     #pragma omp flush(flag)
  }
}
Else prime(num) = false;
}
```