CSCI 624 Distributed Systems - programming problems

9th October 2005

1 Hello world

Write a program in PC or MPI that writes "Hello world from P" from each processor, where P is the number of the process, counting from 0. Modify the code so that it writes these messages in order by processor number (hint - see baton passing).

2 Prime number sieve

For a given integer $k$, let $S(k)$ be the set of all prime numbers less than $k$. We can test if any integer less than $k^2$ is prime by checking if it is exactly divisible by any integer $p \in S(k)$. If it is not, then it must be a prime number. This is because any number $n$ that is not prime must be the product of at least two other integers, $n = i \times j$; since $n < k^2$, then either $i$ or $j$ must be $< k$.

The prime number sieve algorithm finds all primes less than some $k^2$ by starting from some initial value of $S(k)$ and testing all the values of $k \leq n < k^2$ for divisibility by any number $\nu \in S(k)$. Any $n$ that is not divisible by some $\nu$ must be prime, and is added to $S(k^2)$.

One form of the algorithm is as follows:

- initialize $S(2) = \{2\}$, $\kappa = 2$
- loop on $k = \kappa, \kappa^2, \ldots$
  - ....loop on $n = k, k + 1, \ldots, k^2 - 1$
  - .........loop on $\nu \in S(k)$
  - ............if( $n/\nu$ has remainder 0) $n$ is not prime - exit loop
  - ............end $\nu$ loop
  - ........if(completed $\nu$ loop) add $n$ to $S(k^2)$
  - ....end $n$ loop
end \( k \) loop

(Note: there is a more efficient - in time - version of this algorithm that does not need to perform any divisions, but requires storage for every integer less than the largest \( k^2 \) to be tested).

In this algorithm, count every test of \( n/\nu \) as one unit of work; this takes care of the loop iterations also because this test is performed at every loop iteration.

1. Write a sequential program to list all primes less than some number \( M \), using the above algorithm.

2. Estimate the time this program takes as a function of \( M \). Use arbitrary time units, assuming that 1 unit of work takes 1 unit of time. Explain your estimate.

3. Devise a modification to this algorithm to allow the work to be distributed to some fixed number \( N \) of processes.

4. Estimate the time your algorithm would take to complete in parallel, as a function of \( M \) and \( N \), assuming the direct cost of transferring data from one process to another can be neglected. Justify your estimate.

5. Assume you have a function \( M(\text{source}, \text{destination}, X) \) that, when called from your program, transfers the contents of variable \( X \) from process \( \text{source} \) to process \( \text{destination} \). Assume that this function performs a send if called at process \( \text{source} \), performs a receive if called at process \( \text{destination} \), and does nothing otherwise. Express your algorithm from (3) as a program. Write the function \( M(\text{source}, \text{destination}, X) \) using calls to MPI\_SEND, MPI\_RECV and IF statements. Compile it and run it on the machines in jb359. Comment on how the program performs compared with what you expected from assignment 1.

3 Scalar product

Given a vector \( X = [x_1, x_2, \ldots, x_n] \) and a vector \( Y = [y_1, y_2, \ldots, y_n] \), the scalar product of the two vectors is \( s = \sum_i x_i y_i \). Write a program in Pfortran to compute this product in parallel. Your input data is a file of the form:

\[
\begin{align*}
n \\
x_1 \\
x_2 \\
\vdots \\
x_n \\
y_1 \\
y_2 \\
\vdots \\
y_n
\end{align*}
\]
The first number in the file is an integer indicating the length of the vectors, but the elements of $X$ and $Y$ are real numbers.

**NOTE:** stdin is not working correctly in parallel. I have tested the following:

```fortran
open (UNIT=1,FILE='testfile')
read (1,*) jj
```

This reads the file at all processes

You may also enclose this in logic:

```fortran
if (myProc.EQ.0 )
open (UNIT=1,FILE='testfile')
read (1,*) jj
endif
```

This reads only at process 0. You then need to send the data to other processes. You may want to experiment to see which method is faster.

## 4 Distributed sort

Design a distributed sort algorithm based on a leader election algorithm. The algorithm should be suitable for implementation on an SPMD system in which you initially know the number of processes $N$ and process identifiers are 0 to $N - 1$.

Consider two possibilities:

1. You will use $N$ processes to sort $N$ integers in the range $0 \ldots M$, $M \gg N$. Initial data consists of one integer at each process. Final data is a sorted list at some process.

2. You will use $N$ processes to sort $K$ integers in the range $0 \ldots M$, $M \gg K \gg N$. Initial data is a disordered list of $\frac{K}{N}$ integers at each process. Final data is a sorted list of $K$ integers at some process.

Implement the algorithm above. Let $8 \leq N \leq 16$. Use MPI, PC or any other system you can get to work.