# Examination Parallel Computing

2011-06-28

- The use of electronic or printed material is not allowed (GEEN OPEN BOEK).
- Read carefully the problems before solving them.
- Give sufficient explanations to you answers.
- In some of the problems you may need to use a specific function (routine) from the corresponding application program interface. You are not expected to remember the exact name and syntax of that function. If you do not remember these details, you may give the function an appropriate name and syntax and explain what it is supposed to do.

## 1. Speed-up

The execution time $t(1)$ of a program on one processor ($p = 1$) consists of a part $t_{par}(1)$ spent on computations that can be executed in parallel and another part $t_{seq}(1)$ spent on computations that are inherently sequential. For a given program, a speed-up $S(p)$ of 75 is achieved with $p = 100$ processors, $S(100) = 75$.

**(a)** Give a definition of the speed-up $S(p)$ with which a program can be executed on a parallel computer with $p$ processors.

**(b)** Derive a formula for the speed-up $S(p)$ as a function of $p$ and the ratio $x = t_{par}(1)/t_{seq}(1)$.

**(c)** What is the value of the the ratio $x$ for the considered program?

**(d)** What is the maximum value of the speed-up which can be achieved for that program?

**(e)** On how many processors $p$ should one execute that program in order to achieve half of the maximum possible speed-up?

## 2. Cannon's algorithm

Describe and explain Cannon's algorithm for matrix multiplication $C = A.B$ where $A$, $B$ and $C$ are $n \times n$ matrices and the algorithm executes on a $n \times n$ process array. Illustrate the algorithm for the case $n = 3$ by a paper simulation that gives the position of data for the appropriate number of elementary time steps needed to complete the algorithm.

## 3. OpenMP

a) Consider the following program:

```
void main()
{
   int ID = 0;
   printf("Hello world from thread %d !\n", ID);
}
```

Modify this program, adding to it OpenMP statements and constructs, such that (a default number of) multiple threads will be created and each thread will print "Hello world from thread i !" where i is the number of a thread.

b) Consider the following section of code:

```
for(i=0;i<N;i++) {c[i]=a[i]+b[i];}
```

Add to this code an appropriate OpenMP work sharing directive or construct, such that the work within the loop is shared by multiple threads.

c) Consider the following section of code:

```
for(i=0;i<N;i++) {c[i]=a[i]+b[i];}
for(i=0;i<N;i++) {z[i]=x[i]*y[i];}
```

Add to this code appropriate OpenMP work sharing directives or constructs, such that the work is shared by two threads whereby one loop is executed by one thread and the other loop is executed in parallel by another thread.

d) Consider the following section of code which is executed by multiple threads whereby id is the (identification) number of a thread:

```
x(id)=function1(id);
for(i=0;i<N;i++) {b[i]=function2(i,x);}
c=c+function3(b);
```

Add to this code an appropriate OpenMP work sharing directive, such that the work within the loop is shared by the available threads. Note that function2 makes use of an array x whose elements will be computed by different threads (provided that you have added the above mentioned directive). Include another appropriate OpenMP directive to ensure that all elements of the array x have been computed (using function1) before any thread starts executing its part of the loop. Include a third appropriate OpenMP directive that will ensure that the value of the variable c will be incremented with the value of function3(b) by only one thread.

## 4. Pthreads

a) Consider the following program:

```
void main()
{
   int ID = 0;
   printf("Hello world from thread %d !\n", ID);
}
```

Modify this program, adding to it Pthreads statements and constructs, such that 4 threads will be created and each thread will print "Hello world from thread i !" where i is the number of a thread.

b) Consider the following section of code:

```
for(i=0;i<N;i++)  {c[i]=a[i]+b[i];}
```

Assume that $N = 10^6$. Add to this code appropriate Pthreads constructs, such that the work within the loop is shared by 10 threads.

c) Consider the following section of code:

```
for(i=0;i<N;i++)  {c[i]=a[i]+b[i];}
for(i=0;i<N;i++)  {z[i]=x[i]*y[i];}
```

Add to this code appropriate Pthreads constructs, such that the work is done by two threads whereby one loop is executed by one thread and the other loop is executed by another thread.

d) Consider the following section of code:

```
for(i=0;i<N;i++)  {s=s+a[i];}
```

Add to this code appropriate Pthreads constructs, such that the work is shared by multiple threads. Each thread computes a partial sum of the elements of a section of the array a. The partial sums computed by the individual threads are then added together by incrementing the global variable s by the values of the partial sums computed by the individual threads. Use a mutex variable to ensure that the mentioned incrementation process is done properly.

**5. MPI**

Assume that the function testprime(i) returns a 1, if an integer number i is a prime number, and a 0 otherwise. Write an MPI program which determines all prime numbers in a given (very large) range $a < i < b$. When the program is run by $p$ processes, each process should do approximately the same amount of work (load balancing). In this respect, be aware that the time needed to execute the function testprime(i) increases with the value of i. When a given process finds a prime number, it needs to send that number to the process with rank 0. The process with rank 0 collects all the prime numbers it receives from the other processes and the prime numbers that it finds itself. After it ensures that all processes (including itself) have completed their prime number searching tasks, it broadcasts the collected set of prime numbers to all processes.