Examination Parallel Computing

2013-06-18

- The use of electronic or printed material is not allowed (GEEN OPEN BOEK).
- Give sufficient explanations to you answers.
- In some of the problems you may need to use a specific directive or function from the corresponding application program interface. If you do not remember the exact name and syntax of the directive or function, you may give it a descriptive 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 and cannot be parallelized. For a given program, a speed-up $S(p)$ of 112 is achieved with $p = 128$ processors, $S(128) = 112$.

(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 ratio x for the considered program?
(d) What is the maximum value of the speed-up which can be achieved for that program (and why)?
(e) On how many processors p should one execute that program in order to achieve half of the maximum possible speed-up?
(f) A certain program takes 100 s to execute on one processor and 14 s to execute on 8 processors. After compiler optimization, the same program takes 17 s on one processor and 3 s on 8 processors. What are the values of the speed-up before and after optimization? What is your explanation of the difference between the two speed-up values?
(g) State Amdahl's law and Gustafson's law. Put the two laws in perspective to one another.

2. OpenMP parallelisation

Consider the following part of a program for Gaussian elimination:

```
/* swap rows k and j */
for (i=0; i<n; ++i)
{
   tmp = A[k][i];
   A[k][i] = A[j][i];
   A[j][i] = tmp;
}
```

```
for (i=k+1; i<n; ++i)
{
   A[i][k] /= A[k][k];
   for (j=k+1; j<n; ++j)
   {
      A[i][j] -= A[i][k]*A[k][j];
   }
}
```

(a) Modify this program segment, adding to it OpenMP statements and constructs, such that (a default number of) multiple threads will be created and will share the workload.
(b) How can you change or modify a data structure or a variable (and which one) to improve the degree of parallelism in the first loop (for a row swap)?

3. OpenMP races
Consider the following part of a program:

```
x = 5;
#pragma omp parallel sections shared(x)
   {
     #pragma omp section
     {
       int w = x;
       w--;
       x = w;
     } x-1                    4
     #pragma omp section
     {
       int y = x;
       y = 2*y+1;
       x = y;
     } 2x+1                   9
     #pragma omp section
     {
       int z = x;
       z = 3*z;
       x = z;
     } 3x
   }
```

(a) What will be the output of this program when it is run by one thread only.
(b) Suppose that we run this program on three threads. Assume that a single assignment is atomic. What are all possible outcomes for x?

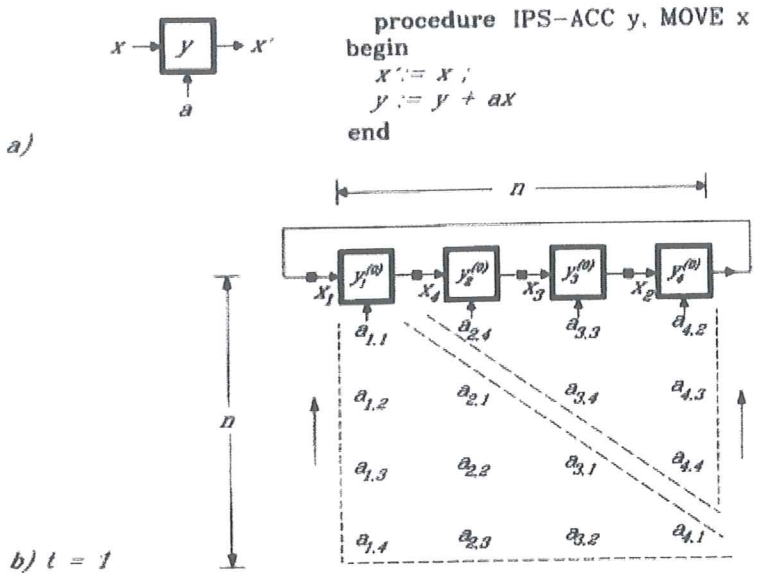

Figure 1: Systolic algorithm for parallel matrix-vector multiplication y = A.x.

4. Parallel matrix-vector multiplication in MPI

Consider the parallel algorithm for the multiplication of a matrix A with a vector x shown in Figure 1. Each process keeps one row of A. The elements of the vecotr x are shifted circularly from process to process and the elements of the result vector y=A.x are computed in the processes, one element per process.

Implement this algorithm in MPI.

5. Parallel bubble sort in Pthreads

Figure 2 inllustrates a stage in the bubble-sort algortihm. The upper data flow in Fig. 2(b-e) is the array that needs to be ordered (sorted). The lower (instruction) flow specifies the sequence of instructions carried out by the stage. In the first time step, the stage loads the first element it receives from the right in an internal variable a_{in}. In each of the following time steps, the stage compares the element it holds internally with the next element in the sequence coming from the right. It keeps the larger of these two elements and outputs to the left the smaller one. After the last element in the sequence is processed, the element which is held by the internal variable is output to the left to follow the other elements. After the whole sequence is processed, the largest element is in the last position. Using a pipleine of such stages the input array can be sorted completely.

Implement this processing stage in a Pthread and create a pipeline of such Pthreads to implement the complete bubble-sort algorithm.

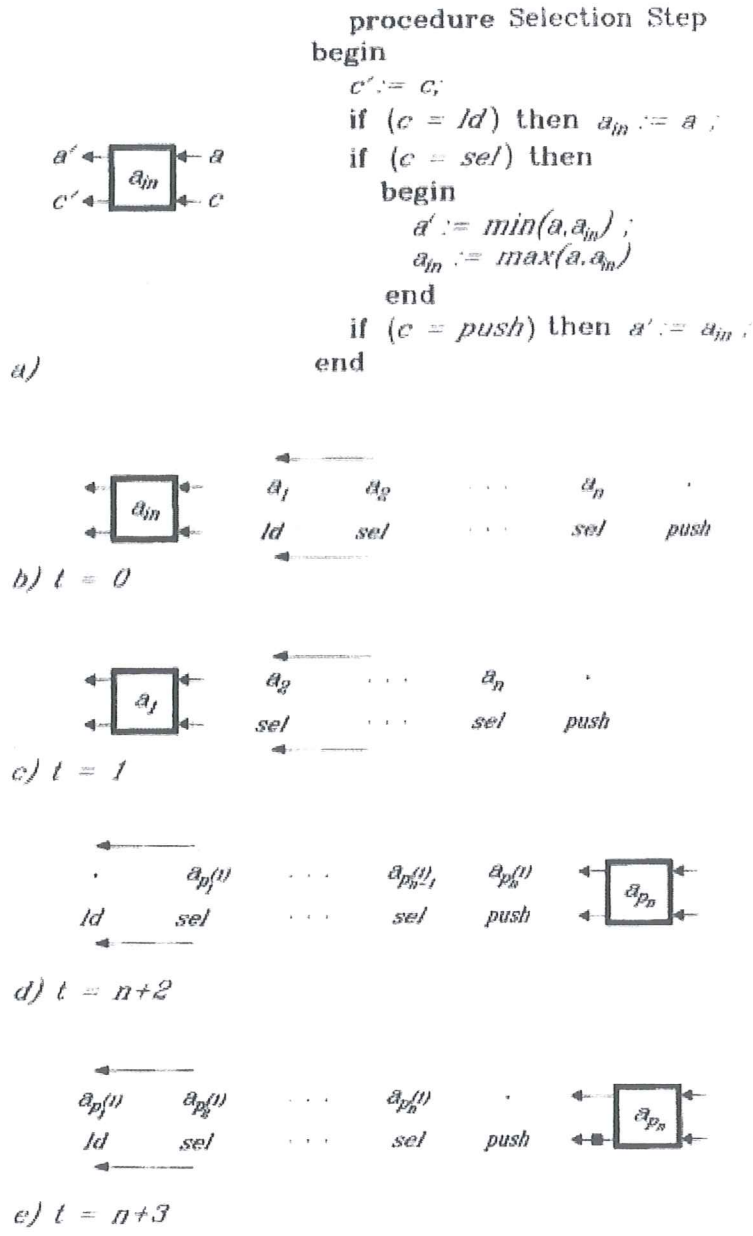

Figure 2: A stage in the bubble sort algorithm.