Uppsala University
Department of Information Technology
Scientific Computing

Programming of Parallel Computers, 10hp, 2012-03-13

Time: $8^{00} - 13^{00}$
Help: None

Each of the six problems below can give up to five points. For maximum points, you must give detailed answers and motivate your assumptions. Grade 3: 12p, Grade 4: 18p, Grade 5: 24p

1. A modern desktop computer can contain two (or more) separate processors, each with four (or more) cores, and a general purpose graphical processing unit (GPU). Moreover, it is easy to connect several desktop computers together in a network. We then have four levels of parallelism where the computers, processors, cores, and GPUs can be programmed in parallel with different programming models/languages/libraries. Discuss what programming models/languages/libraries we can use on the different levels and why they are suitable for the respective levels?

2. What sources of parallel overheads do we have when we program with Pthreads?

3. Explain the concepts of *communicator* and *virtual topology* in MPI. Discuss also the benefits of using these concepts.

4. Write a parallel function in Pthreads that computes the matrix 1-norm, $||A||_1 = \max_j (\sum_i |A_{ij}|)$.

5. The Quicksort algorithm can be described as:

```
function quicksort(array, left, right)
   if (right>left) then
      pivotIndex=left+(right-left)/2
      pivotNewIndex=partition(array, left, right, pivotIndex)
      quicksort(array, left, pivotNewIndex-1)
      quicksort(array, pivotNewIndex+1, right)
   end if
end function
```

   Where the function `partition` reorders the array into two parts, with elements smaller respectively larger than the pivot and returns the split point `pivotNewIndex`.

   You have parallelized this algorithm in Pthreads (Assignment 2) where each thread creates new threads for each recursion level until a maximum level is reached. Then the threads continue sorting in serial. One disadvantage with this approach is that we need to create a large number

of threads to get a good load balance but then we get a large overhead in managing these threads if the time for a context switch is high on the cores.

Another idea to parallelize the quicksort algorithm is to create p number of threads at once but let only one thread do recursion in a few levels. When it hits the maximal number of recursion levels it creates two tasks where each task is to sort the remaining fraction of the list serially with quicksort. It puts the tasks to a work queue from where the other threads picks up work. You have implemented this idea in OpenMP using the task directive (Assignment 3). One disadvantage with this approach is that only one thread puts tasks into the queue while the other waits in a barrier until they get tasks to execute.

Now, combine the two parallelization ideas and implement them in OpenMP, i.e., create p number of threads in main but call only the quicksort algorithm on one of these threads (the other wait in a barrier). But in each recursion step use nested parallelism so that each thread creates two new threads which recursively calls quicksort on one half of the array. For this we can use the section directive. Proceed recursively only a few levels (e.g. 3 levels). Then, let each thread continue some additional levels and generate tasks as in the tasks approach above. Assume that you have the function `partition` available.

6. Many physical phenomena can be simulated with particle systems, where each particle interacts with all other particles according to the laws of physics. Examples include the gravitational interaction among the stars in a galaxy or the Coulomb forces exerted by the atoms in a molecule. Mathematically this can be formulated as:

$$\frac{\partial^2}{\partial t^2}\bar{\mathbf{x}}_i(t) = \sum_{j=1, j\neq i}^{N} G m_j \frac{\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j}{|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j|^3} \qquad i = 1, ..., N$$

Were G is the gravitational constant, m_j is the mass of particle j, $\bar{\mathbf{x}}_i$ is the position of particle i (x, y and z coordinate) and $|\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j|$ denotes the distance between particle i and particle j. This equation can then be integrated numerically with the Leap-frog method

$$\bar{\mathbf{x}}_i^{n+1} = 2\bar{\mathbf{x}}_i^n - \bar{\mathbf{x}}_i^{n-1} + \Delta t^2 \sum_{j=1, j\neq i}^{N} G m_j \frac{\bar{\mathbf{x}}_i^n - \bar{\mathbf{x}}_j^n}{|\bar{\mathbf{x}}_i^n - \bar{\mathbf{x}}_j^n|^3} \qquad i = 1, ..., N \qquad n = 1, 2, 3, ...$$

The numerical simulation can now be formulated in an algorithm:

```
1. Set initial positions for the first two time levels n=0,1
2. for each time step n
3.    for each particle i
4.       Compute the interaction (the sum) with all other particles j
5.       Use the Leap-frog scheme to update the position of particle i
6.    end for
7. end for
```

Sketch a parallel implementation of the 3D N-body simulation using MPI. The implementation should be memory and communication efficient.

Good Luck!