Universitetet i Bergen
Det matematisk-naturvitenskaplige fakultet

Exam in INF236 - Parallel programming, Spring 2022

Problem 1

Consider the following code. For the lines marked L1, L2, L3 and L4 what are the possible values that can be printed? The order in which values are printed is not important.

```
void main() {
  int c1 = 0;
  int c2 = 0;

   omp_set_num_threads(4);
   #pragma omp parallel
  {
    #pragma omp atomic
    c1++;
    printf("par: %d\n", omp_get_thread_num( ));             // L1

    #pragma omp master
    {
      c2++;
      printf("master: %d\n", omp_get_thread_num( ));        // L2
    }

    #pragma omp single
    {
      printf("single: %d\n", omp_get_thread_num( ));        // L3
      c2++;
    }
  }
  printf("c1: %d, c2: %d\n", c1, c2);                       // L4
}
```

Problem 2a

Consider the following C-code where ´array´ is of type double and of size N^2:

```
int i,j;

double t1,t2;

for(i=0;i<N;i++) {

   t1 = sin(i);

   t2 = 0.0;

   for(j=0; j<i; j++). {

      t2 += cos(i + j);

   }

   array[i*N+j]= t1 + t2;

}
```

What directives would you insert to parallelise the code with scheduling set to runtime?

Problem 2b
Using your solution from 2a and with p=2 threads, what is the best speedup you would expect to get if you are using static vs dynamic loop-scheduling respectively?

Problem 3
Quicksort has the following three main steps:

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements that are less than the pivot come before the pivot and so that all elements greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

We are now going to look at and analyse different ways to parallelise Quicksort on a shared memory computer (using OpenMP). In your analysis you may assume that the pivot element is always chosen so that it divides the remaining elements into two partitions of equal size. You may also assume that the number of threads *p* is a power of two, ie $p = 2^k$ for some value *k* and that *p* divides the number of elements *n*.

A) Consider an algorithm that assigns each recursive task to exactly one thread and always does the partition operation sequentially. Derive the running time for this algorithm and its speedup.

B) Outline the main parts of an efficient parallel algorithm where multiple threads are used for each partition operation in Quicksort. Hint, if several threads together partition a sequence using a common pivot, how can one assemble these parts into a shared partitioned sequence?

C) Derive an expression for the running time and speedup of your algorithm in 3b.

Problem 4

When does the MPI routine MPI_Irecv() return?

Select one alternative:

- ○ After the complete arrival of the message the routine is waiting for.
- ○ Immediately.
- ○ After a time specified in the routine.
- ○ As soon as the message has been sent from the sender.
- ○ As soon as the message starts arriving at the receiver.

What does the MPI routine MPI_Comm_rank() do?

Select one alternative:

- ○ It returns an integer that is the number of processes in the specified communicator. The number is returned as an argument.
- ○ It converts the Linux process ID to a unique integer larger than or equal to zero.
- ○ It returns an integer that is the rank of the process in the specified communicator. The integer is returned as an argument.
- ○ It returns the priority number of the process .
- ○ All alternatives are correct.

An MPI process is waiting to receive two messages of the same type from a sending process before it can proceed with its calculations. How can you assure that the system does not deadlock?

Select one alternative:

- ○ Use two MPI_Irecv calls.
- ○ Receive the messages in the same order they were sent using two MPI_Recv() calls.
- ○ Receive the messages using two MPI_Recv() calls but use a wildcard for the message ID.
- ○ Have the same message tag for both messages and use two calls to MPI_Recv.
- ○ All alternatives are correct.

Problem 5

Consider a program where we are given a large two dimensional array of doubles A[N][N] and where we want to keep executing the statement

A[i][j] = (A[i][j] + A[i+1][j] + A[i-1][j] + A[i][j+1] + A[i][j-1])/5.0

for every pair (i,j) such that 0<i,j<N-1, until no A-value changes. Note that this will not change the boundary values in A and in each step we set A[i][j] to the average of its own value and those of its four nearest neighbours.

To do this we write an MPI-program using p processes and where we partition A onto a two dimensional process grid of size $\sqrt{p} \times \sqrt{p}$. Ignoring the cost to check if the solution has converged, what is the computation to communication ratio if this is done properly?

Select one alternative:

- ○ $\Theta(\frac{n^2}{p})$
- ○ $\Theta(\frac{n}{p})$
- ○ $\Theta(\frac{n^2}{\sqrt{p}})$
- ○ $\Theta(\frac{n}{\sqrt{p}})$
- ○ $\Theta(\frac{\sqrt{n}}{\sqrt{p}})$
- ○ $\Theta(\frac{\sqrt{n}}{p})$
- ○ $\Theta(n\sqrt{p})$
- ○ $\Theta(p\sqrt{n})$