Tools of high performance computing 2024

Exercise 6

Return by Monday 26.2.2024
Exercise session: Tuesday 27.2.2024

Problem 1. *(6 points)*

Scaling behavior of an algorithm: matrix eigenvalues. Write a Python program that computes eigenvalues of a matrix $N \times N$ using `scipy.linalg.eig`. Do the calculations for matrices of sizes roughly $N = 30..1000$. It's a good idea to use logarithmically evenly distributed values and collect statistics by doing each size many times. Measuring CPU time in Python can be done using the function `process_time` in package `time`.

Plot the CPU time as a function of matrix size N. What is the scaling in the big-oh[1] notation? Search the literature what it is supposed to be and compare with your measurements.

Hint: You might want to skip the data for very small matrices.
Note: In many Python environments scipy function do calculations in parallel by default. You can prevent this (we want the single CPU performance) by setting the number of threads environment variable to one:

```
from os import environ
environ['OMP_NUM_THREADS'] = '1'
```

This setting must be put before you import `scipy`*. Note also that if you are using Spyder for this setting to come into effect you must restart the environment.*

Problem 2. *(6 points)*

Write a parallel program where all processes that have their id-number less than `ntasks-1` send a message to the next process (`0→1`, `1→2`, …, `ntasks-2→ntasks-1`). The message is an array with the first element containing the id-number of the sending process. After sending the message all but process 0 wait for a message. When sending data every process prints out its own and the recipient's id-numbers. When receiving a message every process prints out the sender id-number and the first element of the message array.

Problem 3. *(6 points)*

Change the program in problem 3 so that also the process with id `ntasks-1`, sends the message to process 0 (periodic boundary conditions!). Explain what you see and what you would expect.

1 Informally: notation $O(n^k)$ means that time $T(n)$ used by the calculation behaves as $T(n) \to a\,n^k$, as $n \to \infty$, where a is a constant. Formally it is the set of functions: $O(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0,$ so that $c_1 g(n) \le f(n) \le c_2 g(n) \forall\, n \ge n_0\}$.

Problem 4. *(6 points)*

Using ping-pong communication (sending a message between two processes back and forth many times) measure the latency and bandwidth[1] of the MPI system you are using. This means that you have to measure the wall clock time used by the operation as a function of the message size. In your answer specify on what kind of system you are doing your computations (the hardware, whether it is via shared memory or network).

MPI has a function for obtaining the wall clock time:

C

```
double start,stop,wallclock;
//…
start=MPI_Wtime();
//… do something …
stop=MPI_Wtime();
wallclock=stop-start;
```

Fortran

```
real(8) :: start,stop,wallclock
!...
start=mpi_wtime()
! … do something ...
stop=mpi_wtime()
wallclock=stop-start
```

1 In principle the time l taken by the message passing as a function of the message size N can be expressed as $t = N/b + l$, where b is the bandwidth (in e.g. MB/s) and l is the latency.