Alexander Schnapp
Max Menges

Intro HPC: Blatt 4

17.11.1014

4.1 Reading

4.2 Matrix Multiply – parallel version using MPI

For the Matrix Multiply in parallel version we use the source code 'matrix_ multiply.cc'. To run and compile use our file 'make+run.sh'.

In this code we first initialise the matrizes A and B by the master process (rank==0). This process is then distributing the diffrent lines of A to the different slave processes. The Matrix B is distributet to all processes completely. After the processes have calculated the result resulting of a line of A they are sending the result back to the master process. After this has got all the results it prints the solution. (This is not contribution to the calculation time).

The outcome of this calculation is the following Matrix C:

0	30	60	90	120
0	40	80	120	160
0	50	100	150	200
0	60	120	180	240
0	70	140	210	280

4.3 Matrix Multiply – scaling process count

We now excecute our programm with a diffrent number of processes. For that we encount the nodes creek[01-04]. We are calculation a Matrix of 1000x1000 entries. This wwe compare with our sequential implementation.

Plotting the Time that was needed for the calculations we see that its decreasing very fast but is reaching limit very soon.

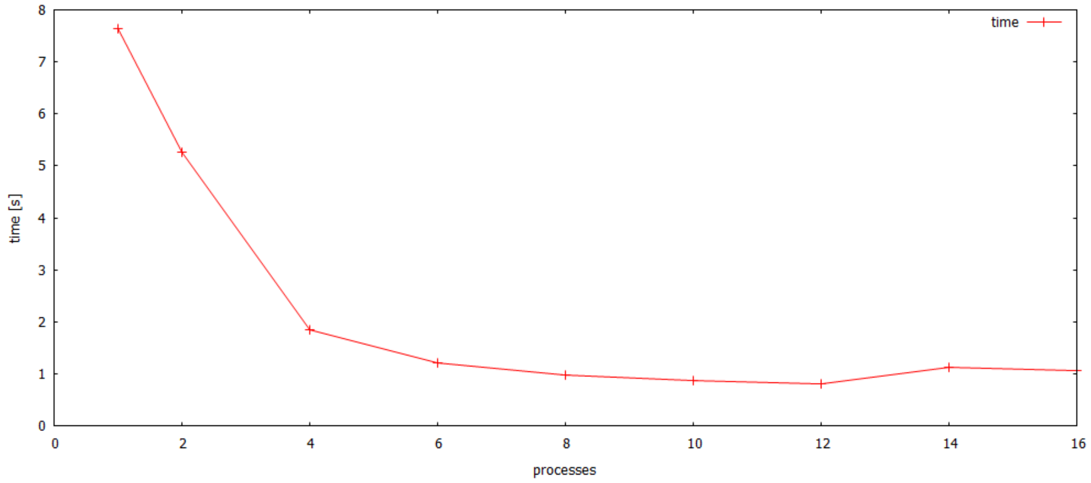
Plotting the speed-up we see a increase very similar to a linear speed up in the beginnen but then it starts to deviate more and more.

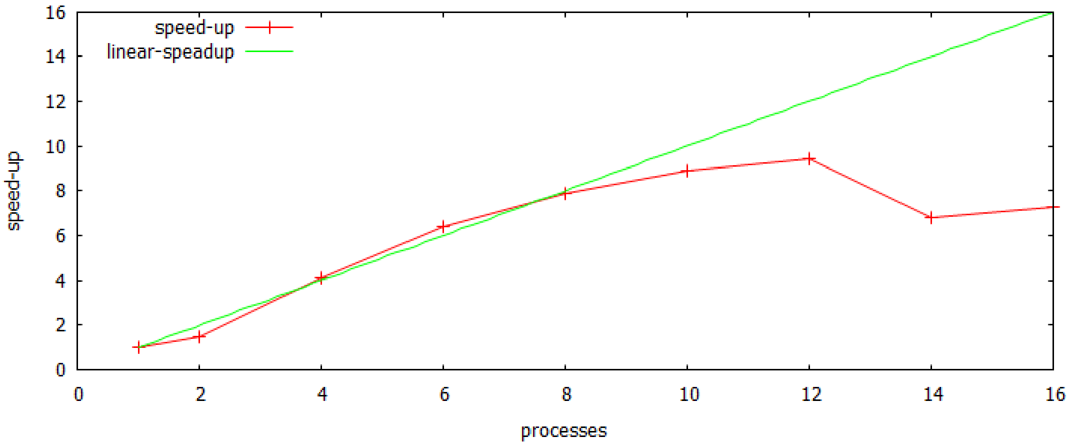
The Effeciency is nearly 1 for about 4-8 processes but then drastically decreasing for higher numbers.

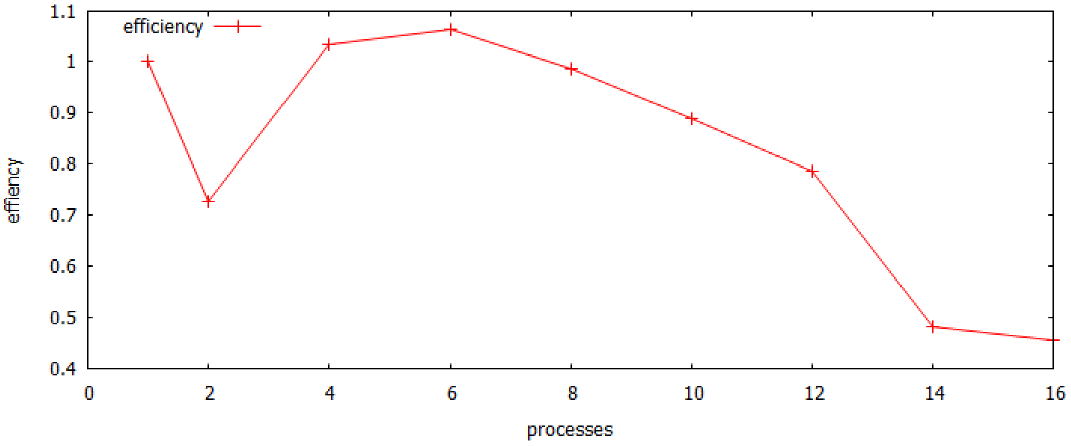
All this is a consequence of the process and the used data being at is best at about 4-8 processes. With higher values the message traffic gets too high, so that the parallalization is not very effective any more.

4.4 Matrix Multiply – scaling problem size

This can be changed if you have more data, so that the single process is having more work to do with nearly the same ammount of messege traffic.


time
time [s]
processes
0
1
2
3
4
5
6
7
8
0
2
4
6
8
10
12
14
16


speed-up
linear-speadup
speed-up
processes
0
2
4
6
8
10
12
14
16
0
2
4
6
8
10
12
14
16


efficiency
effiency
processes
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
0
2
4
6
8
10
12
14
16