Laboration 3 - TDDC78

Jonathan Karlsson - jonka293 - 890201-1991
Niclas Olofsson - nicol271 - 900904-5338

21 maj 2013

1 Program description

The program we got was using the Jacobi method to calculate temperatures on a quadratic plate. Basically what the method does is, for each point on the plate, calculate the temperature by adding the fore neighbors together.

Since fortran stores arrays column wise we want to iterate the plate column wise to save operation overhead. By identifying the dependencies in the program and saw that the last column can be written to at the same time as it is read by another node. We solved this by starting the parallel section, calculated the interval each node should work on and with this we got what the first and last column was going to be.

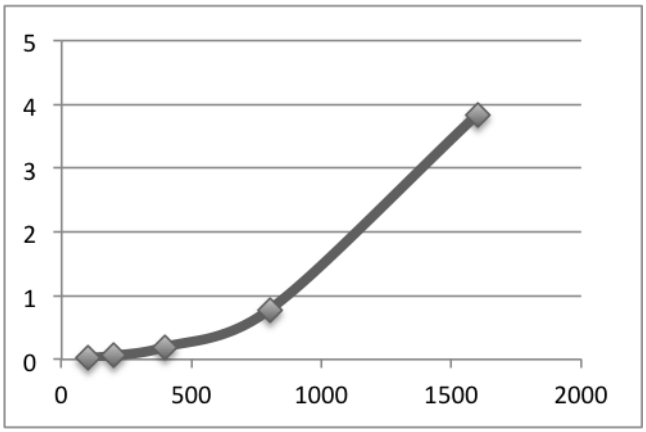
Now we created a loop with "maxiter" iterations, in this loop we get the first and last column, make a barrier so each node has read the columns and now we create a new loop that iterates every column of the plate. Here openmp's "omp do" is used to divide the columns into even work for each node. Inside this loop the regular Jacobi method is used with our previously read last column in the special case of then node being at the last iteration.

The reduction method is used to calculate the maximum error of each node, if this error is below our tolerance level then all nodes exit the loops and the program is done. The higher tolerance the better resolution but it will take significantly longer time.

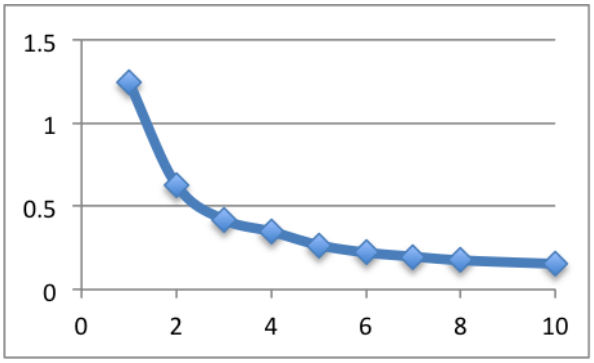
2 Execution times

Below are figures showing the execution times. Figure 1 shows the execution times with two nodes, $maxiter = 1000$ and increasing $n = 100, 200, ..., 1600$. What we would like to see that if the problem size is doubled as well as the number of nodes we get a linear curve, this isn't really achieved since we can't simply double n and double the problem size since the plate have the dimensions $n * n$. But the figure shows that the result is somewhat linear so it tell us that the solution is probably correct.

Figure 2 shows the execution times with increasing number of nodes on the same problem size, $n = 1000$ and $maxiter = 1000$. This result is good with double amount of nodes gives half execution time, as expected.

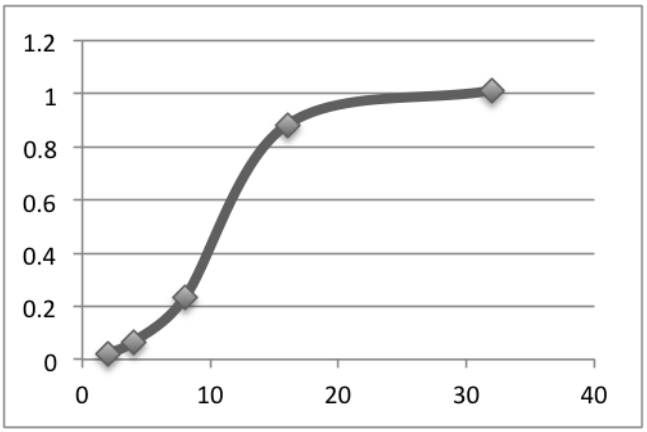


Figur 1: Execution times with two nodes, increasing problem size.



Figur 2: Execution times on different number of nodes and same problem size.

In figure 3 both the number of nodes and the problem size is increased. The point of this is to see a somewhat constant line, but again we have the problem with the problem size not really being doubled as with the number of nodes, but the result is still pretty close with the exception of 16 and 32 but that could be explained with bigger communication overhead over several processors.



Figur 3: Execution times with increasing number of nodes and an increasing problem size.