Lab Course: Efficient Programming of Multicore Processors and Supercomputers

Group 1: Jonas Mayer, Paul Preißner, Konrad Pröll

June 20, 2017

5. Parallelization of heat with MPI

Input Parsing

When parsing the input, we quickly found two different concepts: Parsing in one process and broadcasting the data or parse on all different processes. First, we thought that parsing in one process and broadcasting the data would be the smarter choice, as we would not have trouble managing the IO, however, we noticed that broadcasting the heatsources, an array of structs, was not as trivial as we thought, so we changed to all processes reading the data.

Local Grinds

We came up with two concepts for handling the processes local grids.

The first concept was to simply let every single process initialize the entire grid but just relax one specific tile of the grid. This would be realized by passing on a specific tile offset and tile size in x and y to the relaxation function before exchanging the border cells with its neighbors. This first concept promises a first good speedup without having to do any major modifications to the initialization and handling of the grid.

The second concept was to initialize only the tiles relevant for each process. This way we wouldn't waste memory on parts of the grid that we don't touch.

Process communication

The next step was to synchronize the data between the neighboring processes. To simplify this, we made use of the MPI topologies (MPI_Cart_shift). Therefore, we had each process sent the outermost calculated gridpoints to the neighboring processes according to the cartesian grid, and receive the data from the neighboring processes into the outermost gridpoints, which where not recalculated due to missing neighbors. To do this communication, we used selfdefined datatypes to describe the different datalayouts for sending the values up/down

and left/right: Up and Down are continous, while left and right are vectors, in which each entry has a distance of linesize to the last one.

In the beginning, we did this routine right in heat.c after the call of the relaxation, but optimized it later so that we swapped the data nonblockingly at the beginning of the calculation, calculated the gridpoints which do not rely on data from other processes, which are located at coordinates 2 to np-2. Afterward, we synchronize the communication and calculate the last layer.

Gather the Grid/Output

Next up is the task to actually print the picture. So far there are two ideas on how to implement this: simple sends or gather operations. This hasn't been implemented yet though.

Current State

Right now, our program does parse the data as it should, initialize the local array (incl heatsrc), transfer the ghost cells between neighbors and calculate the residua. Thus far, we have not implemented the graphical output (the picture).

After that, all processes initialize their data, i. e. in particular their arrays. We added some more or less useful parameters to the param struct, namely the size of the local array (x and y, as we no longer are guaranteed to have squares), the length of useful data in the array, because we decided to pad the array, if we could not distribute them evenly, and the processes coordinates in the cartesian grid. We do at no point initialize the global array, but only the local portion of it (plus one row/column in each direction, "ghost cells" for communication purposes) and afterwards use the position of the process in the grid to initialize the heatsources. Afterwards, we compared the arrays produced with the array the vanilla version for OpenMP/MPI produced, and after we fixed some minor bugs, had the same data.

Speedup

To be added when implementation fully works.