Chapter 7. Parallelization

7.1 Introduction

Parallelization of a computer program means making it capable of running on multiple processors, and thereby faster. There are two main technologies for achieving this, depending on the memory configuration of the computer: A shared memory architecture or a distributed memory architecture. The two approaches are implemented using two libraries: OpenMP for shared memory systems and MPI for distributed memory systems.

A shared memory system is in principle more simple, as the programmer does not need to worry about where the information is stored. Parallelization happens on loop-levels, meaning a coding sentence needs to be inserted before each parallelized loop.

Most computer clusters are made up of nodes, with maybe 16 cpus around one shared memory. The cluster then consists of multiple such nodes. An OpenMP system will then only be able to use 16 processors. This is the main disadvantage of the OpenMP approach.

A distributed memory approach can use many nodes. In principle, the geometry or the domain is divided into multiple parts, where each part runs on its own node or cpu. Then communication between the nodes is done with the MPI library. The advantage is that many more cpus can be used. The disadvantage is that the communication between the nodes is often slow, limiting the scaling of the program.

OpenMP is implemented in both *SSIIM 1* and *SSIIM 2*. MPI is only implemented in *SSIIM 2*. When running on clusters as described above, it is possible to use both OpenMP and MPI at the same time.

7.2 OpenMP

The OpenMP parallelization in *SSIIM* is invoked by the *F 206* data set. The program behaves the same as with one processor, it only runs faster.

Problems with the parallelization include inconsistencies in the solver. This means that the results of the program is slightly different depending on how many processors are used. For a water flow computation, this difference is usually very small. However, for a sediment computation with bed elevation changes, small differences tend to amplify and the resulting bed changes may look significantly different. To reduce the problem, a modified solver has been de-

veloped. This is slower than the ordinary solver, but gives more consistent results. It is invoked by *F 164 31*.

The scaling of the different parts of the program can be tested by using the *F 1 P* option. Then, the computed time is written to the boogie file before and after many of the different algorithms. The following numbering is used for *SSIIM 2*:

2-3: Computing eddy-viscosity, water fluxes between cells, coefficients in the Navier-Stoke eq.
4-5: Computing wall laws and pressure source
5-6: Boundary conditions and some source terms
7-8: Computation of multi grid acceleration arrays
8-9: Solver, Navier-Stokes equations
9-10: Water fluxes computations for SIMPLE
10-11: Coefficients and source terms for the SIMPLE correction equation
11-12: Source terms for the nested grid for SIMPLE corrections
13-14: Solver, pressure-correction equation
15-16: SIMPLE corrections
18-19: Coefficients and source terms for turbulent kinetic energy equation
19-20: Boundary conditions for the turbulent kinetic energy equation
22-23: Solver, turbulent kinetic energy
24-25: Coefficient generation, boundary condition and solver for epsilon
35-36: Computation of grain size distributions of the bed
41-42: Regenerating the grid
42-42: Making multi-grid connections
43-48: metric (computation of areas, distances, vectors etc. in the grid)
48-51: Making connections between the 2D and the 3D grid, and interpolations between grids
49-50: Interpolating values from old grid to new grid
45-46: Metric for nested grids
46-47: Making boundary pointer array for nested grid

The information can be used to find out which algorithms take most time in *SSIIM 2* and how they scale.

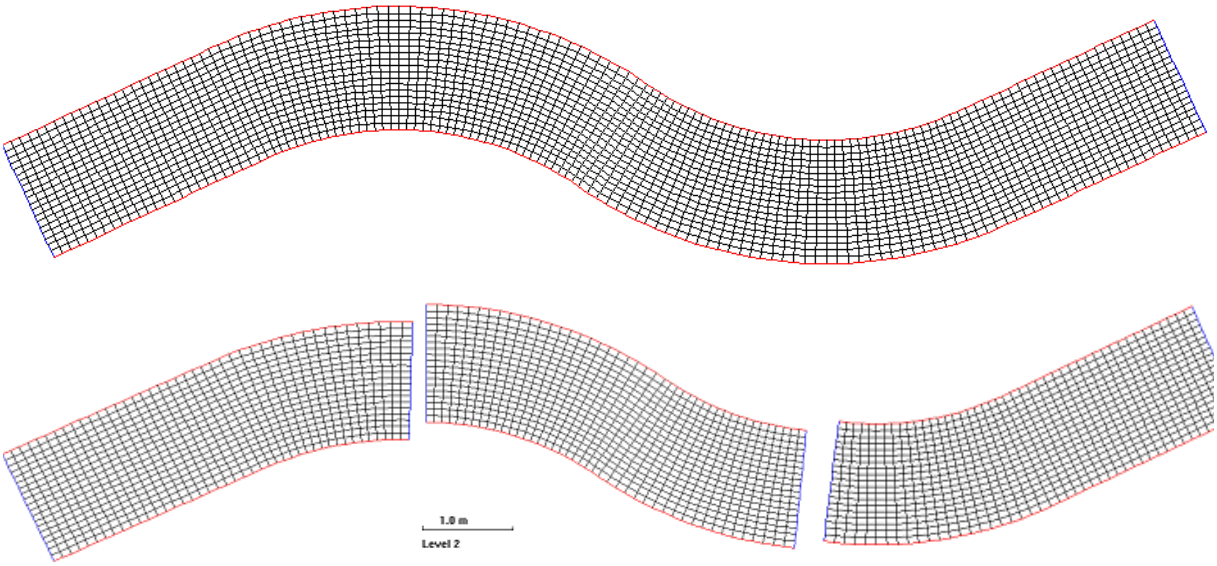
Parallelization can also be tested with the *G 50* data set in the control file. Ten integers are read, each corresponding to a particular part of the program. If the integer is zero, the *F 206* data set will work on that particular algorithm. If the integer is 1 (one), a sequential version of the algorithm is used instead. This can be used to test the parallelization of the different algorithms.

1st: coefficient generation for the Navier-Stokes equation, the Power-law scheme
2nd: computation of eddy-viscosity, setting a_nb coefficients to zero, computing the water flux and making the coefficients for the second-order upwind scheme
3rd: wall laws and source term for the pressure equation
4th: water fluxes computation for the pressure-correction equation
5th: SIMPLE corrections
6th: turbulence computations, wall laws and coefficient generation for k
7th: turbulence computations, wall laws and coefficient generation for epsilon
8th: computation of residuals
9th: solver
10th: time step and residual for k

7.3 MPI

An MPI approach requires a domain decomposition, meaning the geometry needs to be divided into several parts. *SSIIM 2* is then run for each part, and MPI algorithms are used for communication between the parts.

The domain decomposition in *SSIIM 2* assumes that the geometry is much longer than it is wide, so that the long river is cut into blocks by dividing cross-sections. It is not possible to cut the domain in longitudinal slices or in vertical slices. The cuts should be made in areas where the flow direction is in the downstream direction. Recirculation zones should be avoided. All the blocks have to have the same number of cells in the cross-section.



The domain decomposition is not straightforward and needs to be done by the user. One of two methods can be used:

1. The whole grid can be made first, and then divided into blocks (*G 27 / F 2 UA*)
2. The blocks can be made one at the time, starting at the upstream block.

The grid generation, domain decomposition and viewing the results can be done using the normal Windows version of *SSIIM 2*. The execution of the program needs to be done with a special MPI-enabled version of *SSIIM 2*. This version has MPI in its name, for example *ssiim2v_mpi157.exe*.

Essential input to the MPI algorithms in *SSIIM 2* is the *F 308* data set. An integer is read. This integer can be used in two ways: If you are working with one block of the grid, the integer is the number of this block. If you are working with the all the blocks, the integer is the number of blocks you have.

7.3.1 Domain decomposition using method 1

Start by making a grid of the whole geometry. Then give the *F 308* data set with the correct number of blocks and optionally, the *G 27* data sets in the control file. The *G 27* data set specifies the number of the cross-sections where you want to cut the grid. Then start *SSIIM 2* with the *F 2 UA* option in the *control* file. This gives a number of files called *koosurf.mpi1*, *koosurf.mpi2* etc. One *koosurf.mpi* file for each block. After this is done, the other files: *unstruc, koordina, koosurf* etc. needs to be made by the user for each block. This is done by first removing the *F 2 UA* from the control file, and changing the *F 308* data set to 1, and then reading in the *koosurf* file in the grid editor. Then make the grid including other possible modifications. Note that the two most upstream and downstream cross-sections can not be moved. Then generate the 3D grid and write the *unstruc* file. The *unstruc* file is called *unstruc.mpi1*.

Then change the *F 308* data set to 2, and repeat the procedure, making the *unstruc.mpi2* file. Continue until all the *unstruc.mpi* files are made.

The *G 27* data set reads the same number of integers as there are blocks. The last integer is the last cross-section in the total grid. Example with three blocks and 90 cross-sections in the total grid: *G 27 30 60 90*

7.3.2. Domain decomposition using method 2.

The user should specify *F 308 1* in the control file and then make the grid for the first block. Then write the resulting koordina.mpi1, unstruc.mpi1 files etc. Then change the *F 308* data set to 2, and do the same for the second block. The two first cross-sections in block 2 should then not be allowed to move. All the files should be in the same directory, so that the *.mpi1* files are present in the directory where the *.mpi2* files are made. After block 2 is made, then block 3 should be made etc.

7.3.3 General advice

The inflow of water should be specified only in the upstream cross-section of the first block, and outflow of water only specified in the downstream cross-section of the last block. Alternatively, the discharges can be specified on the *F 337* data sets. For the MPI version of SSIIM 2, discharge group 1 is always the upstream inflow and discharge group 2 is always the downstream outflow for all blocks. The *F 314 1 1* data set can be used to define the inflow discharge area in the upstream block, and the outflow discharge area in the downstream block. The *F 314 1 1* data set will automatically use the whole cross-section for inflow/outflow. Using the *F 314* and *F 337* data sets means the discharges need not be specified in the *DischargeEditor*.

The *F 337* data set is similar to the *F 237* data set, except two integers are read before the float (discharge). The first integer is the block number. So for three blocks which has two discharges each (in and outflow), six *F 337* data sets are needed. Example

F 337 1 1 2270.0
F 337 1 2 2270.0
F 337 2 1 2270.0

F 337 2 2 2270.0
F 337 3 1 2270.0
F 337 3 2 2270.0

The *unstruc*, *koordina*, *koomin* files etc. should have the extension *.mpi1* for the first block, *.mpi2* for the second block etc. The same extensions should be made for the *bedrough* file.

When the program is run, the results are written to files called *result.mpi1*, *boogie.mpi1*, *con2res.mpi*, *tecplot.mpi1* etc. The user can choose which result files *SSIIM 2* is reading by specifying the desired block number on the *F 308* data set, and then use the *SSIIM 2* graphics to view the results for this block.

Ensuring similar geometry at borders between blocks

The MPI coupling between the blocks is crucial and possibly problematic. In SSIIM 2, one cross-section of cells is overlapping between the blocks. This cross-section has to have exactly the same number of cells in both blocks to make the connection work. If the number of cells is different, the program will crash.

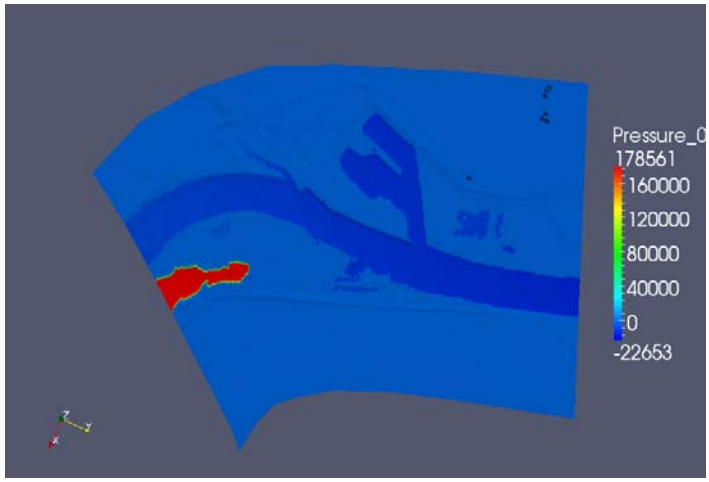
To avoid this problem, the *F 209* data set have to be specified in the control file. This gives the maximum depth of the geometry. Also, some combinations of the *F 159* data set have given problems. The value of the *F 94* data set may need to be changed slightly to avoid the problem.

There are further options to avoid the problem. The *F 313* data set exchanges water and bed level information between the blocks at the overlapping cross-section. An integer is read. If it is 2, both water and bed levels are exchanged. If the integer is 4, additional print-out is sent to the *boogie.mpi* files.

Another option is *F 319 1*. Here, the grid generation procedure exchanges the number of grid cells in the overlapping cross-section. So that this should be the same even if the water level or the bed level differs at the cross-section. If 2 is given instead of 1, additional print-out is written to the *boogie.mpi* files.

Dead ends

If the boundary between the blocks cuts through a pond at the overbank of the river, then the pond will be a dead end channel. The procedure giving the flux into the block will also give an inflow into this dead end. This will cause a very high pressure in the pond. The velocity field will still be ok, but the pressure will cause high water levels if the free surface algorithm is



used. To avoid the problem, several algorithms were tested, on the *F 316* data set. Option *F 316 8* seemed to work best.

Specifying pressure reference cell

Instead of giving the reference cell on the *G 6* data set, it is possible to use the *F 317* data set. Two integers are read. The first is the block number. The second is the *j* index for the down-stream reference cell from the 2D indexing system (*i*,*j*). The value of the *j* index should be given for the grid location with the deepest point in the river, to make sure it will not dry up.

For uniform flow computations, it is only needed to give in the *F 317* data set for the down-stream block.