MPI Parallelisation I

Added MPI module with following main submodules:

mpi_init:

- Associates each processor with its rank
- Generates 2D cartesian coordinate space and processor coordinates
- Calculates neighbouring ranks in 2D cartesian coordinate system
- Splits simulation domain according to coordinate space and calculates domain size for each rank => *slabimin/max, slabjmin/max*

init_surround:

- Initializes subarray types which enables to send 2D subarrays between the ranks

get_surround:

- Takes a dimension as an argument
- Communicates with surrounding neighbouring ranks
- Stores surrounding cells of neighbouring ranks in current rank's ghost cells

Other important additions:

- Parallelized *cmpdt* function
- Decreased global resolution (nx,ny,imin,imax,jmin,jmax) to rank specific domain size
- Allreduce time measurements to get time measurements from slowest processor
- Boundary conditions only apply to cells which border no other processors
- Outputs have rank id attached thus can be stitched together afterwards

Possible improvements:

- Call *get_surround* then do operation on available center cells and proceed with surrounding cells which depend on the ghost cells only after get_surround has finished.

MPI Parallelisation II

General disadvantages of MPI:

- Communication has to be done by user which can be complicated
- Cannot make use of shared memory
- Proper debugging pretty much impossible

General advantages of MPI:

- More flexible than OpenMP because more low level
- Can make use of multiple nodes

Implemented feature	Advantages of feature	Disadvantages of feature
2D Domain splitting	Shared data size decreases with increasing number of processors, with 1D splitting this is not the case	Each rank communicates with max 4 processors, for 1D splitting its only max 2, thus waiting time can be bigger.
IRecv / ISend instead of Recv and Send	Rank can start communication with all neighbouring processors before any data is received.	Requires waitall statement to synchronize ranks.
IRecv before ISend	In some cases neighbouring processors calls ISEND before current rank called IRECV, therefore it makes sense to send IRECV first	None

OpenMP

General disadvantages of OpenMP:

- Cannot make use of multiple nodes
- Less flexible because high level library

General advantages of OpenMP:

- Implementation is way easier than with MPI, communication is handled by library
- Debugging is possible
- Can make use of shared memory / Threads

Implementation details:
Added parallel segments to godunov and compdt with parallel do loops.
For *compdt* some variables needed to be set private.

Hybrid (OpenMP + MPI)

General disadvantages of hybrid approach:

- Overheads from both libraries

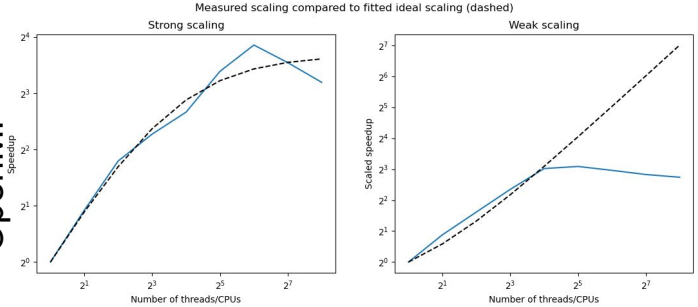
General advantages of hybrid approach:

- Use MPI for internodal communication and thus make use of multiple nodes
- Use OpenMP to take advantage of shared memory in nodes

Implementation details:
Pretty much a union of OpenMP and MPI code. The only difference is we have to use MPI_INIT_THREAD instead of MPI_INIT.

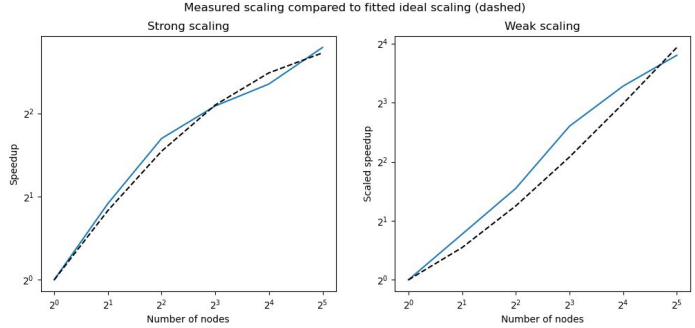
Performance Analysis/Discussion I

OpenMP



p_strong = 0.85
p_weak = 0.6

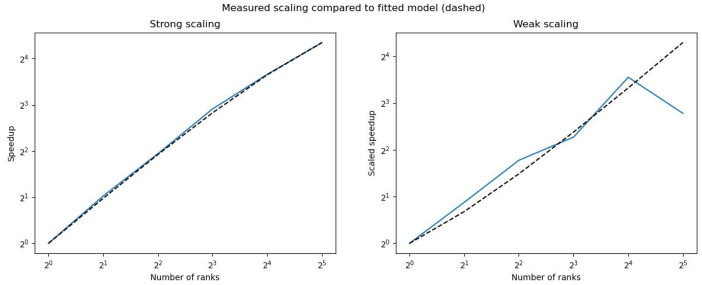
Hybrid



p_strong = 0.87
p_weak = 0.46

nThreads/node = 64

MPI



p_strong = 0.97
p_weak = 0.3

1. *What do the dotted lines represent?*

Weak scaling: Fitted plot of Amdahl's law with parallel fraction p_strong
Strong scaling: Fitted plot of Gustafson's law with parallel fraction p_weak

2. *Why is there a speedup drop in the OpenMP version?*

Usually the ideal number of threads corresponds to the number of CPU's. The peak speed up occurs at 2^6 = 64 and one node has 72 CPU's with hyperthreading turned on.

3. *Why does the MPI version have a steeper speedup curve than the hybrid version?*

The hybrid version starts with a much higher speed for y=1 because it uses OpenMP with 64 threads, thus the speedup is less dramatic than for the MPI version.

4. Why is there a sudden drop in the weak scaling speedup plot for the OpenMP version?

For weak scaling the the size of the simulation domain is increased with each step (ny = log2(nthreads) * 128, nx = 1024), for larger sizes data allocation and other overheads may dominate the execution times, plus there's the additional speedup drop when the thread number exceed the number of CPU's.

5. Why do the coefficient estimates of Gustav's and Amdahl's differ for a single parallelisation strategy?

The overhead may increase when the problem size increases. This is supported by fact that the difference is smaller for the OpenMP version where no data is shared. In the Hybrid and MPI version the amount of data needed to be communicated between ranks grows with the simulation size thus increases overhead.

Performance Analysis/Discussion II

1. *Why is the MPI version faster on a single node than the OpenMP version?*

Even though OpenMP can make use of shared memory, MPI still prevails. OpenMP would probably yield better results when the number of threads would be increased. Also the execution time is probably too short to compensate for the increased overhead caused by OpenMP.

2. *Why is the hybrid version slower than the MPI and OpenMP version?*

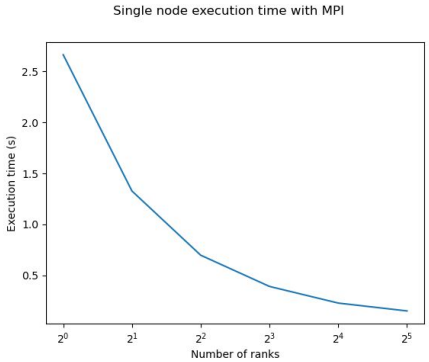
I am not sure about this, either the implementation could be improved, or a different selection of nodes and threads would yield better results or the simulation size is not big enough to see the advantages of the hybrid version. In theory it should be faster. A possible way would be to optimize the split between threads and ranks for a single node by trial and error but this requires a lot of time.

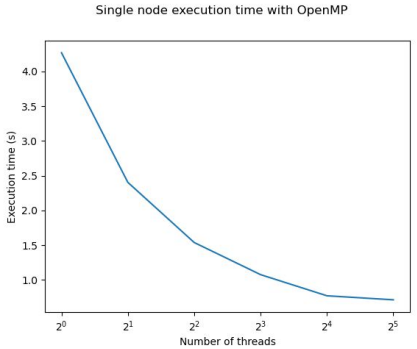
Execution time benchmark comparison with given parameters and 32 nodes:

nx = ny = 1024, nstepmax = 50, dx = 0.08

Strategy	*Time*	*# nprocs*
MPI	0.7s	32
OpenMP	1.3 s	64 (Hyperthreading)
Hybrid	1.6 s	32*64
Non-Parallel	19.5s	-

OpenMP vs. MPI on a single node:



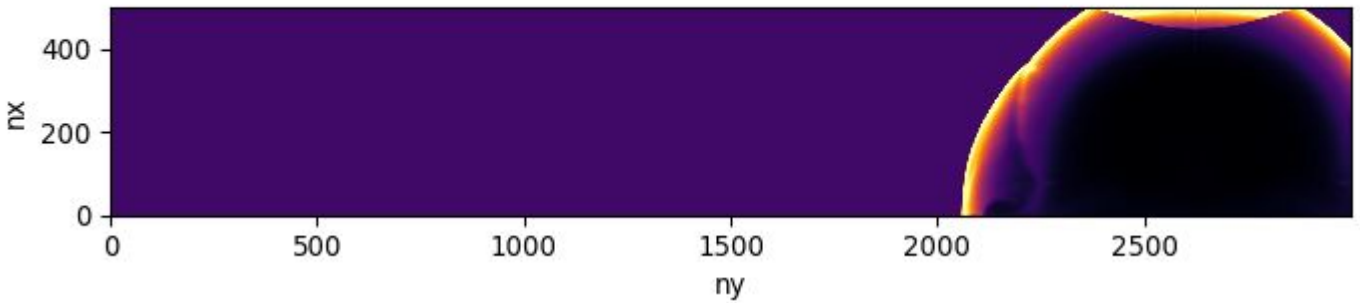


Massive Simulation

Used parameters:
nx=500, ny=3000, dx= 0.008

Ran using the hybrid version and 8 nodes with each 36 threads.

Still frame from the simulation:



Unfortunately I ran into issues with exporting movies from matplotlib animations, no no animated version :(