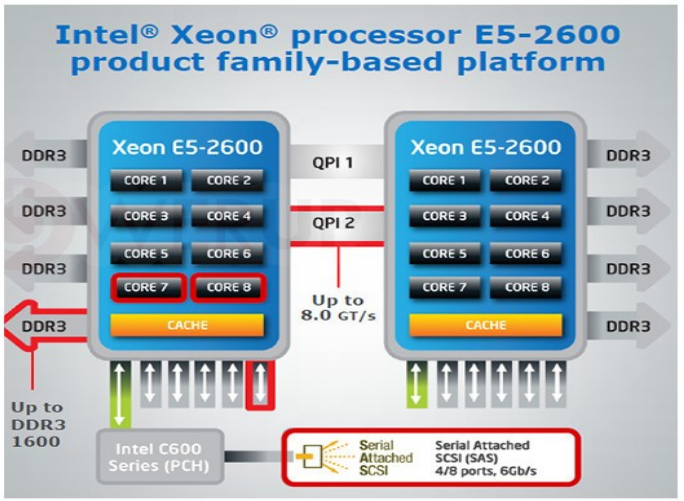
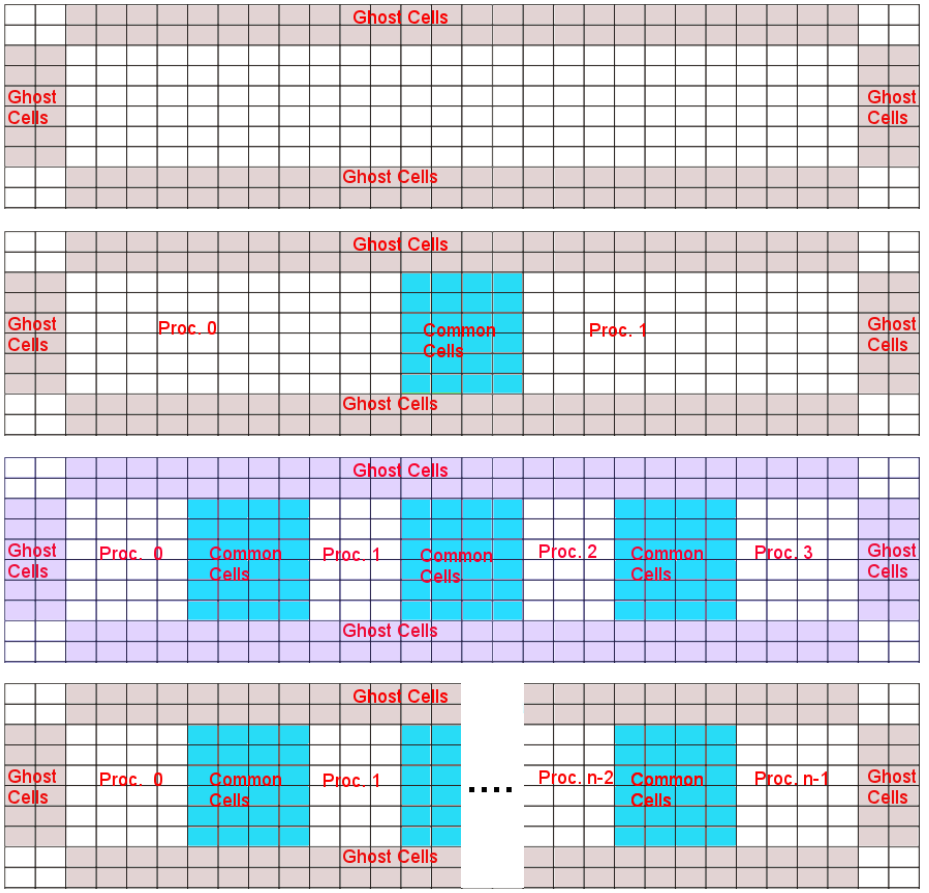
Introduction

Parallelization of a 2D Hydro code using MPI
Course: ESC401 - HPC1b: Parallel Computing
Authors: Rafael Küng, Christian Reinhardt and Mihai Tomozeiu

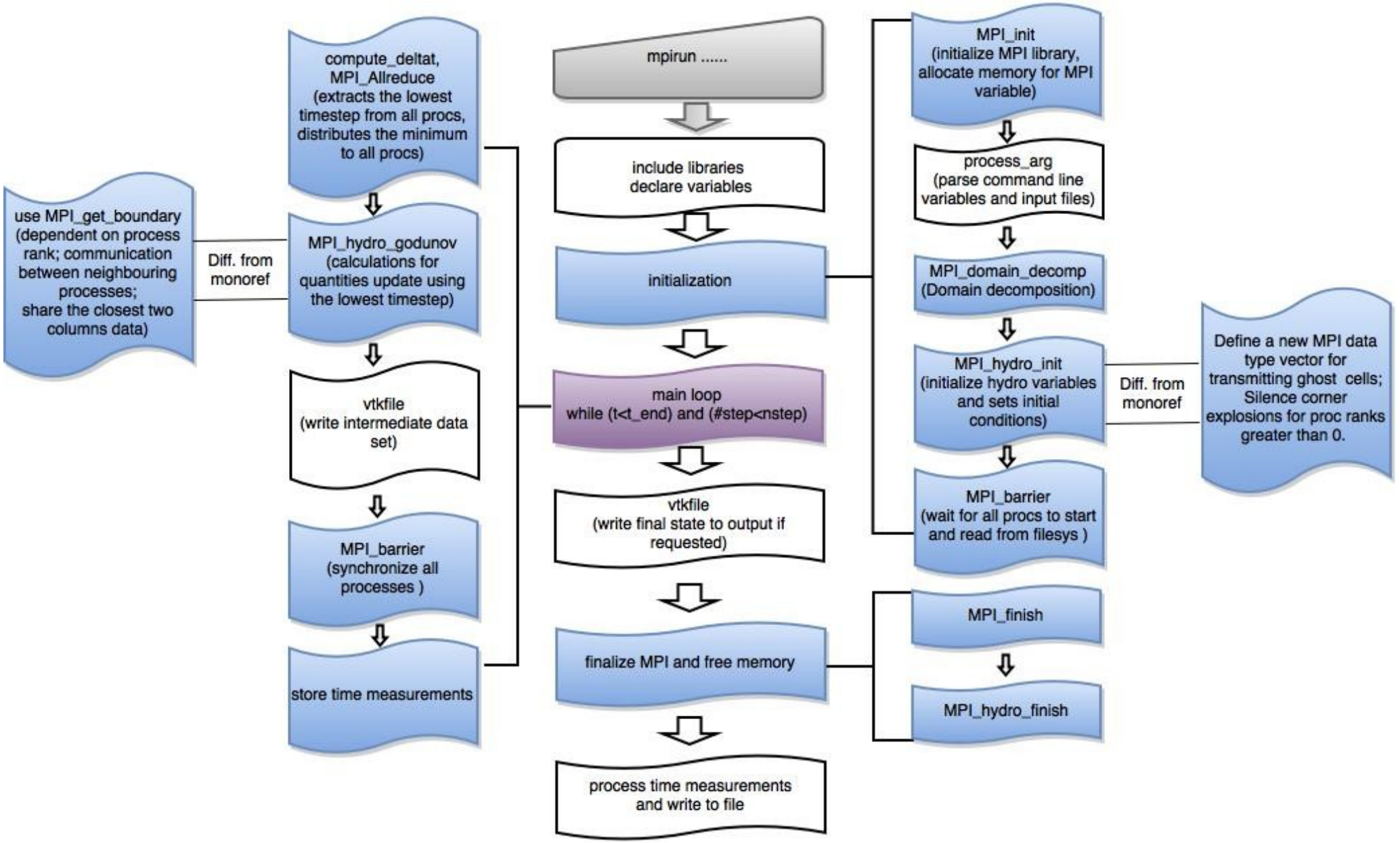
- The original code simulates an explosion in the lower left corner (right images) of a 2D wind tunnel using a single process
- Number of individual proper cells n_x(rows) * n_y(columns) and 2 * (n_x +n_y) ghost cells for reflective boundaries
- Approach:
  - general idea: modify as little code as possible
  - slicing the domain along the x-axis
- Domain distribution in right figures panels:
  - 2nd: 2 cores; 3rd: 4 cores; 4th: n cores
- Each neighboring processes need to communicate the closest two columns of proper cells to each other
- Minimum time step found among processes will become the time step of all processes



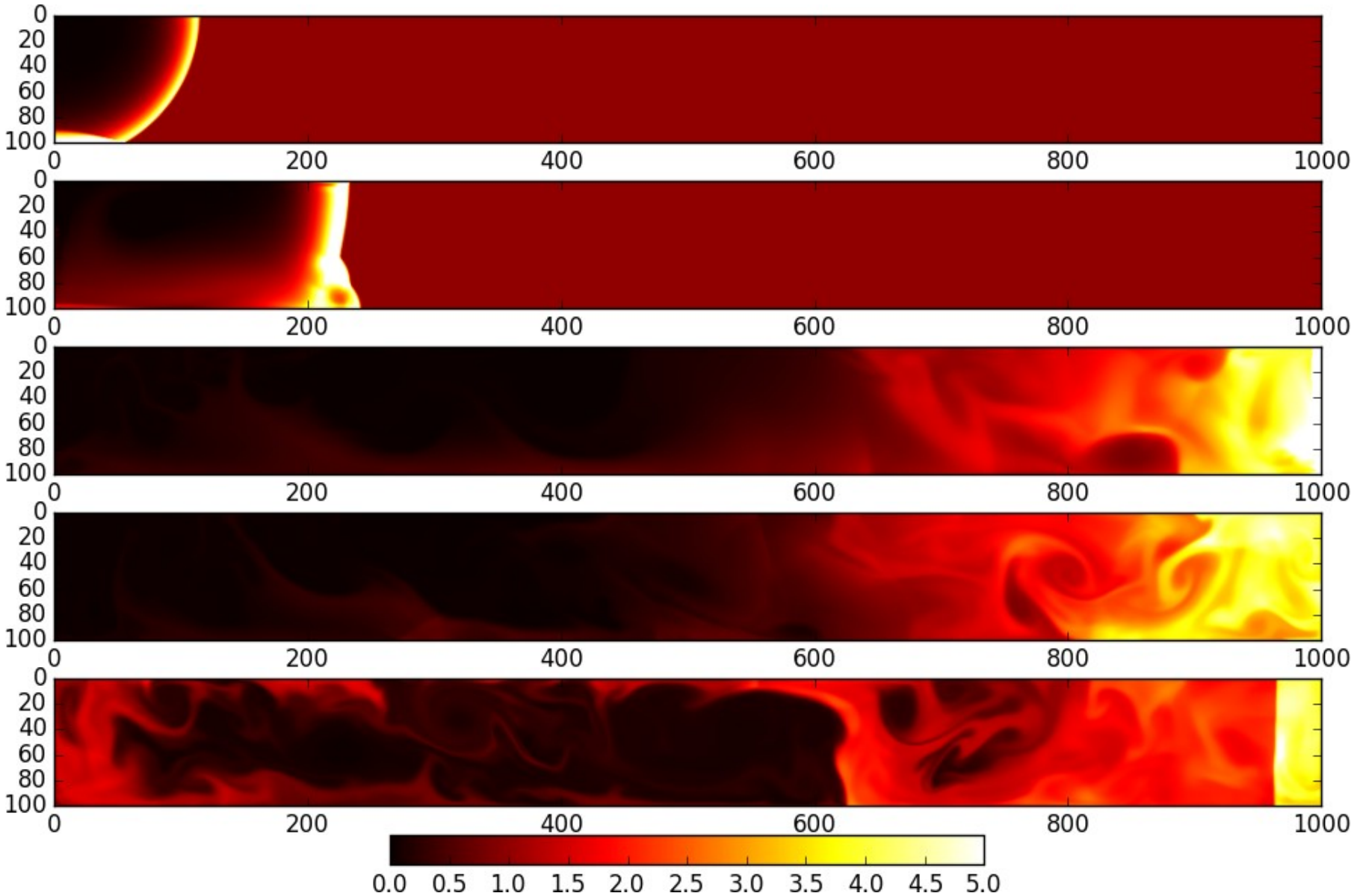


- The simulations were run on the local zbox4
- Intel Xeon E5-2660 (family platform sketch in the left image)
- 8 cores per socket
- 2 sockets per node
- Internode communication with QDR Infiniband

Code Description



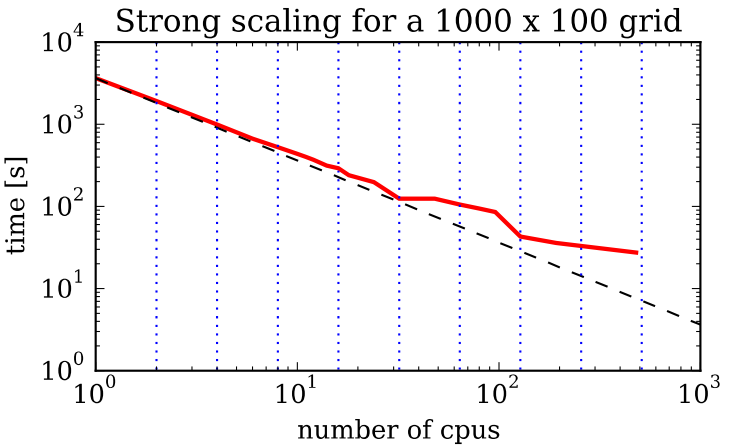
Results / Output

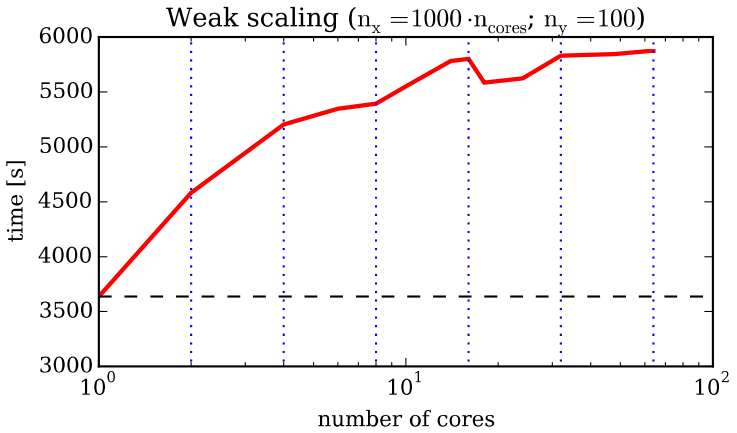


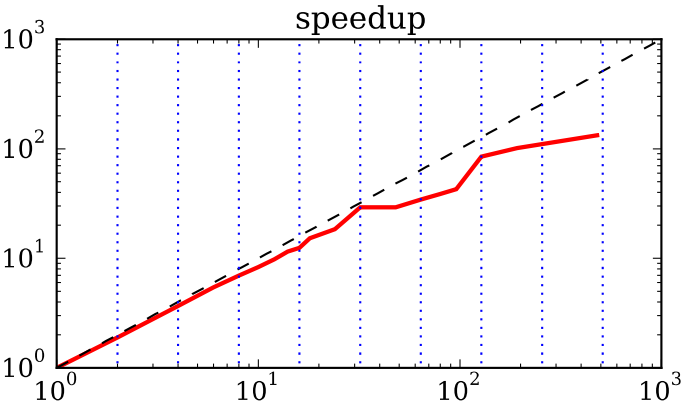
Example of a run (1000x100 grid): An initial overpressure in the top left cell causes a blast wave to propagate trough the computational domain. The walls are reflecting so the shock bounces back at the domain boundaries and interferes with the original wave. This causes an oscillating bulk motion with beautiful and quite complex wave patterns in the inner part of the domain. The colors represent the fluid density where black is zero and white is at least five.

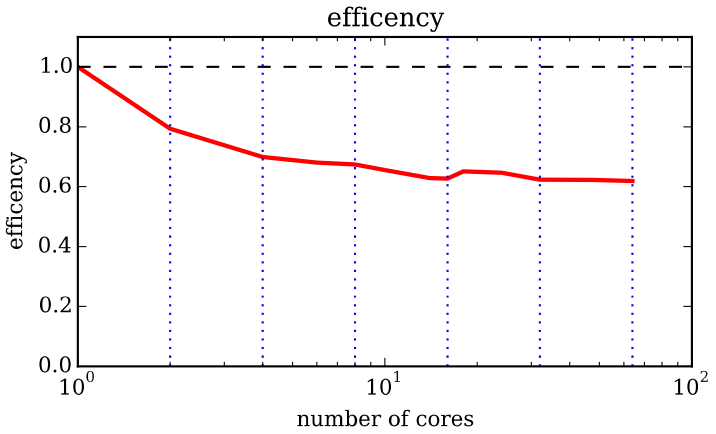
Scaling Behaviour

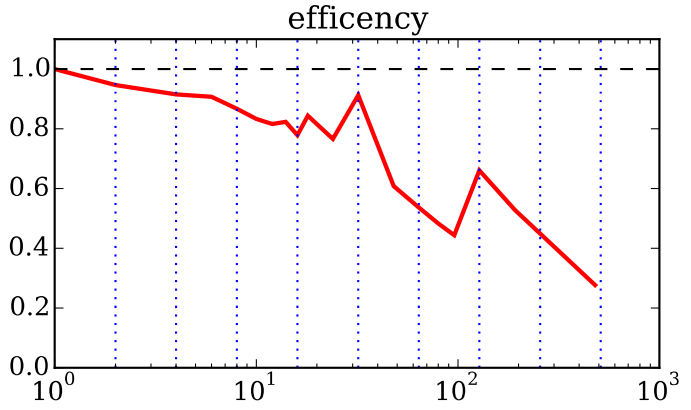
- Strong Scaling
  - performed on a 1000x100 (n_x * n_y) grid
  - right upper figure: results from 1 to 96 cores
  - right lower figure: zoom for 1 to 18 cores range
- Weak Scaling
  - n_y constant
  - n_x scaled with the number of cores
  - the 16 cores point marks the transition to internode communication showing a core performance worsening
  - after 4 cores and no



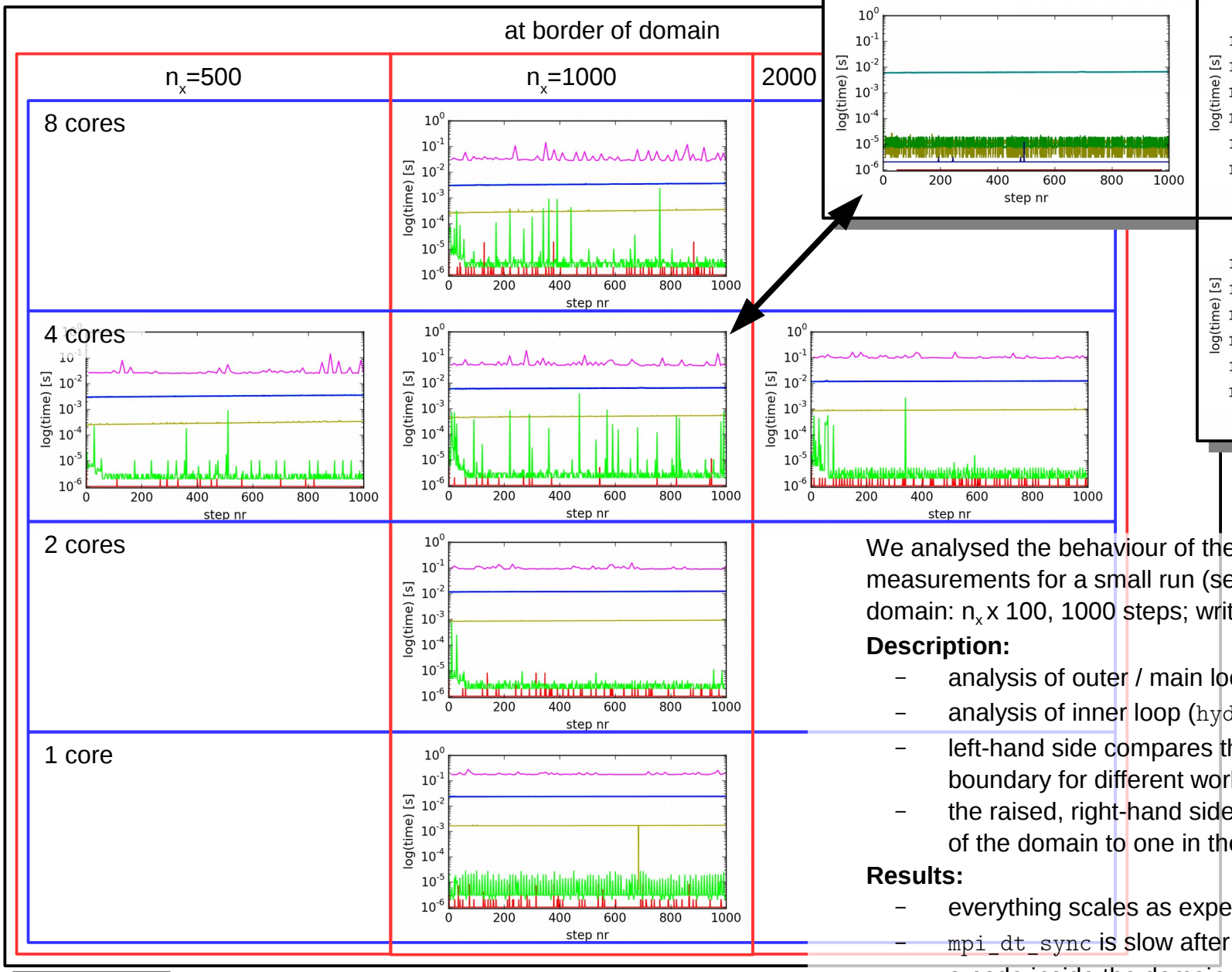


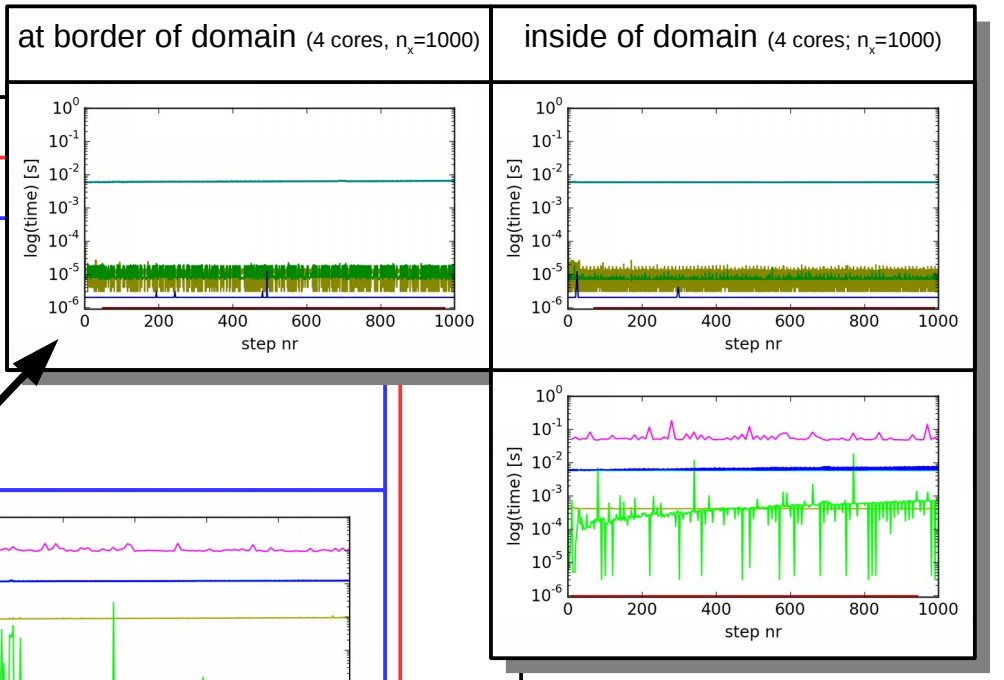






Code Diagnostics – Overview





We analysed the behaviour of the code using several timing measurements for a small run (settings used: run on zbox4; domain: n_x x 100, 1000 steps; write output each 10th step)

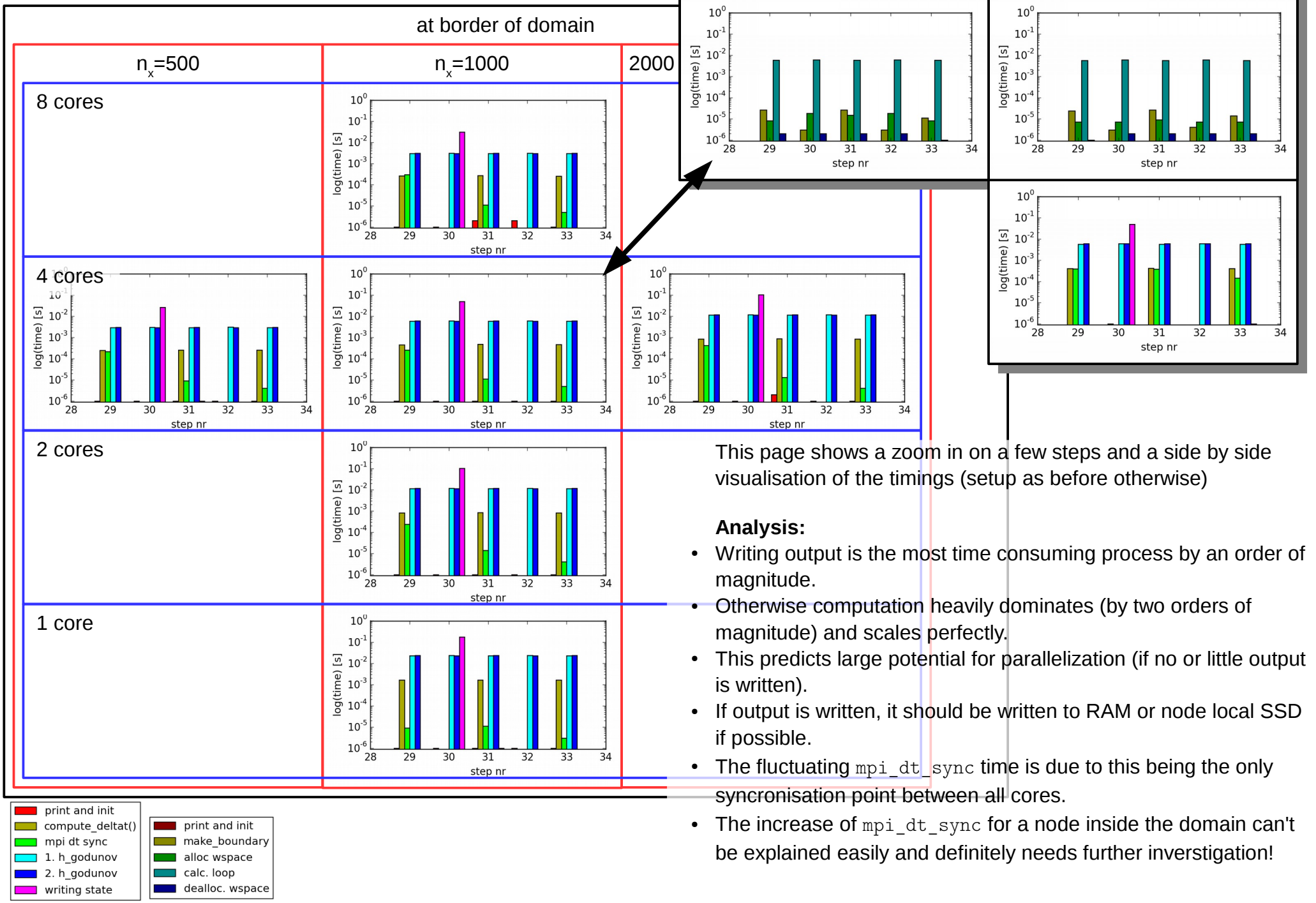
Description:

- analysis of outer / main loop with bright colors
- analysis of inner loop (`hydro_godunov()`) in dark colors
- left-hand side compares the behaviour of a core at the boundary for different world sizes and domain sizes.
- the raised, right-hand side compares a core at the border of the domain to one in the middle.

Results:

- everything scales as expected
- `mpi_dt_sync` is slow after startup, noisy and increases for a node inside the domain
- MPI code inside `make_boundary()` doesn't show this behaviour.

Code Diagnostics – Details



This page shows a zoom in on a few steps and a side by side visualisation of the timings (setup as before otherwise)

Analysis:

- Writing output is the most time consuming process by an order of magnitude.
- Otherwise computation heavily dominates (by two orders of magnitude) and scales perfectly.
- This predicts large potential for parallelization (if no or little output is written).
- If output is written, it should be written to RAM or node local SSD if possible.
- The fluctuating `mpi_dt_sync` time is due to this being the only syncronisation point between all cores.
- The increase of `mpi_dt_sync` for a node inside the domain can't be explained easily and definitely needs further inverstigation!

Conclusions

- We successfully parallelized the existing serial code with MPI.
- Using our code we ran several simulations with different grid sizes on a varying number of CPU cores. Both the hard and the weak scaling plots show a clear performance improvement when run of multiple cores.
- For the hard scaling the simulation time is linearly decreasing with the number of CPUs up to 10 tasks. Our timing analysis shows that after that writing outputs to the hard drive is the main bottleneck and causes a problem if too many outputs are written.