High Performance Computing 1b: Hydro Code

Mladen Ivkovic, Mischa Knabenhans, Rafael Souzalima

June 2016

The Hydro Code

The hydro code solves the Euler equations $\frac{\partial \vec{U}}{\partial t} + \vec{\nabla}\vec{F} = 0$, where $\vec{U} = (\ldots)$ is the vector containing the conserved quantities ... and $\vec{F} = (\ldots)$ blablabla for a two dimensional hydrodynamic simulation.

The simulation domain is rectangular, divided in square cells. The boundary conditions for the outermost cells are set to simulate a wall by mirroring the last two rows of cells before the wall.

It uses a Godunov's scheme with an exact Riemann solver, which yields the following numerical equation to be solved for each cell of the domain and for both dimensions separately:

$$U_{i,j}^{n+1} = U_{i,j}^{n} + \frac{\Delta t}{\Delta x}\left(F_{x,i+\frac{1}{2}}^{n+\frac{1}{2}} - F_{x,i-\frac{1}{2}}^{n+\frac{1}{2}}\right)$$

Parallelisation Strategy

The strategy used for the parallelisation of the code was to split the simulation domain between the processors, so each processor would only work on its own part of the domain, without having knowledge of what is happening in the domains of other processors. Before each step, the borders in the direction that is being calculated are communicated to neighbouring processors, creating a connection between the split domain parts.

In the initialisation phase of the code, first the best processor distribution for a given total number of processors and simulation domain size is calculated by choosing a combination of processors in x and y direction which minimises the communication time, considering the lattency and the message sizes.
Then, the domain is split amongst those processors as evenly as possible. If the grid size in x (resp. y) direction is not a multiple of the number of processors assigned to the x (resp. y) direction, the remaining amount of cells of this integer division is distributed one per processor, starting with the last in x (resp. y) direction, so that the processor ranked 0 would have less work, since it's the one responsible for runtime output.
After this step, every processor is "taught" who its neighbours are, or whether there is a wall next to it.

The advantage of this parallelisation is that any domain size can be calculated for any numbers of processors and the most expensive part of the work is done in parallel, but the processor distribution along a direction is not freely choosable, even though it may be enforced by setting the simulation domain sizes appropriately.

Bottlenecks

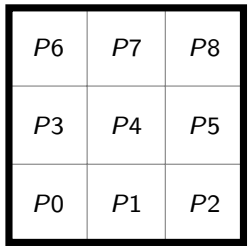
We identified the bottlenecks of the code to be in the communication between the processors.

On one hand, on each calculation step, the maximal allowed time step, which ensures conservation of the conservative variables, must be recalculated by each processor. Then the minimal value of this maximal timestep must be communicated to all processors for the simulation to evolve at the same pace for each part of the domain. For this communication, we used the MPI_ALLREDUCE routine. This is a blocking call and might waste processing time if processors have to wait for eachother often. To minimise waiting time, we tried to distribute the work as evenly as possible amongst the processors.

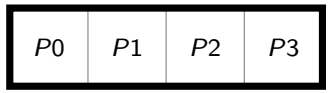
On the other hand, before each calculation step, the borders of the domain have to be communicated by each processor to each of its neighbours. For this, we used a MPI_SENDRECV routine, which is a blocking call as well. Since we're sending a part of an array for every communication, the MPI routine will create an array temporary for each communication. This bottleneck might be optimised by using nonblocking calls and defining a new MPI structure type, but we couldn't realise those ideas due to time limitations.

Performance and Scaling: Weak Scaling

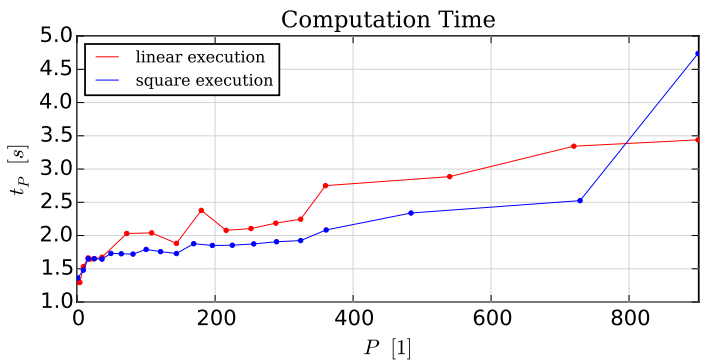
Since our parallelisation allows domain splitting in both directions, we wanted to create a comparison between a strictly "linear execution", where the domain is split only in one direction, and a "square execution", where the number of processors assigned to each direction is equal.

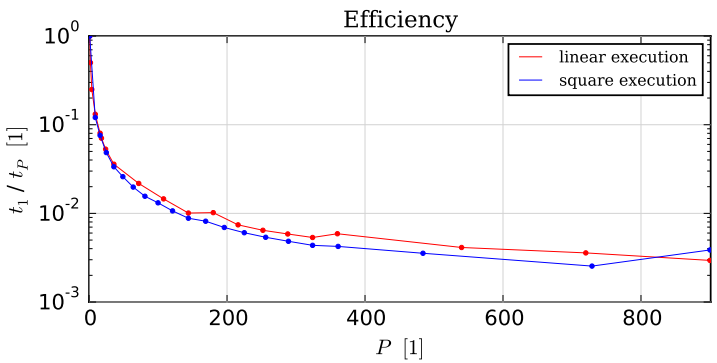


Processor distribution for a "square execution"



Processor distribution for a "linear execution"

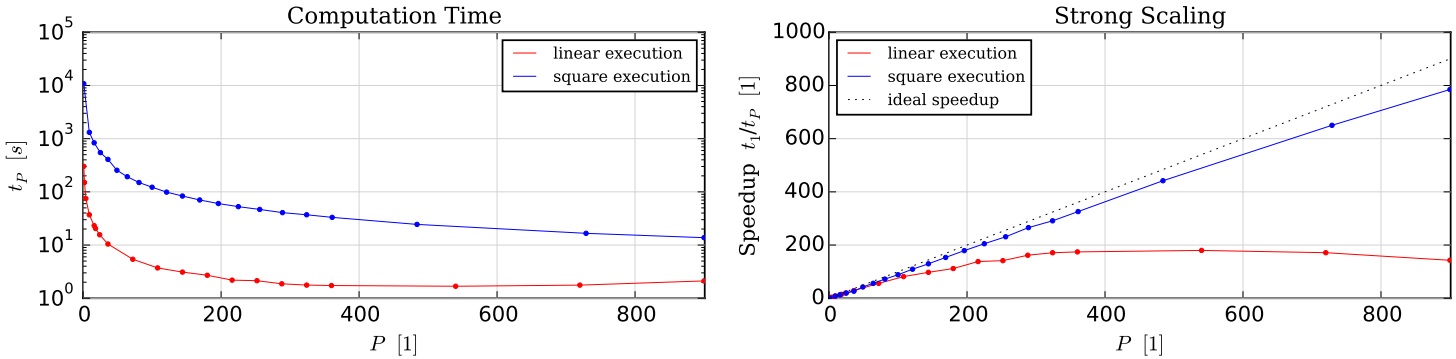




P: Number of processors used. t_P : Measured execution time for P processors.

For the weak scaling measurements, all processors were assigned the same workload of 200 $\times$ 200 cells for 100 calculation steps, so the total number of cells increases with the number of processors. Surprisingly the square execution requires less computation time, even though a square processor distribution requires more communications between processors.

Performance and Scaling: Strong Scaling

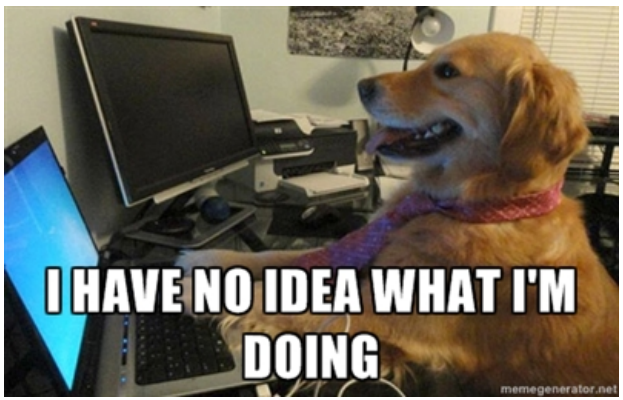


P: Number of processors used. t_P : Measured execution time for P processors.

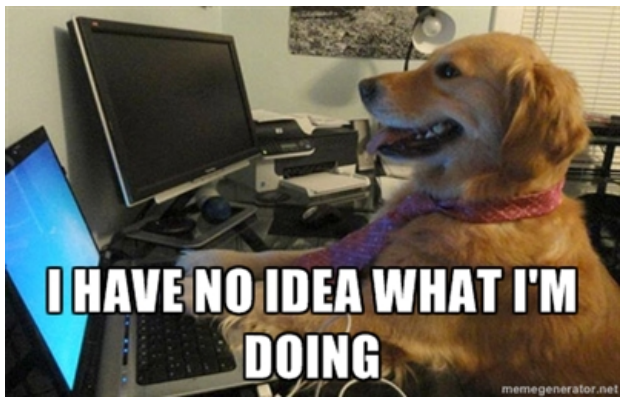
For the strong scaling performance test, a fixed grid size was chosen and calculated for 100 calculation steps and computed with a different amount of processors.
The grid size for the linear execution was 45360×200 and for the square execution it was 16200×16200. Please note that therefore the computation times in the plot above cannot be compared directly.
The square execution shows much better scaling than the linear execution. This can be explained by looking at how the communication time changes with the number of processors used: while we may assume the total lattency effects to be constant for all numbers of processors used, the transfer time ("talking time") reduces with the number of processors for a fixed total size grid because less and less cells need to be communicated between two processors. It can be shown (see appendix) that the rate of change of the communication time for a whole calculation step with changing processor number $\frac{dt_C}{dP}$ is $\propto -P^{-2}$ for the linear execution and $\propto -P^{-3/2}$ for the square execution. So the communication time for the square execution will decrease faster with increasing number of used processors, leading to the better scaling property for the square execution.

High Quality Output

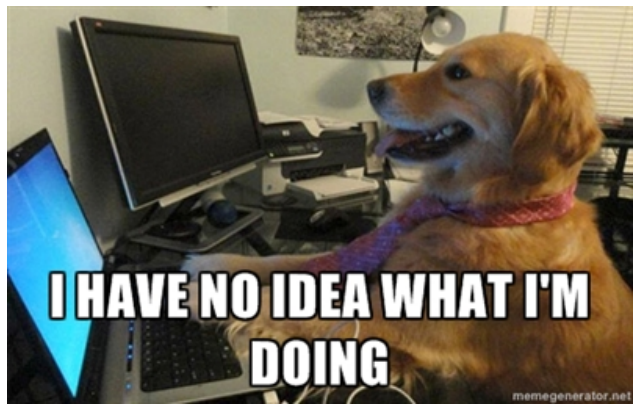
Below are three high quality outputs at calculation steps X, Y and Z for the gridsize $A \times B$, calculated on n processors.



Output X



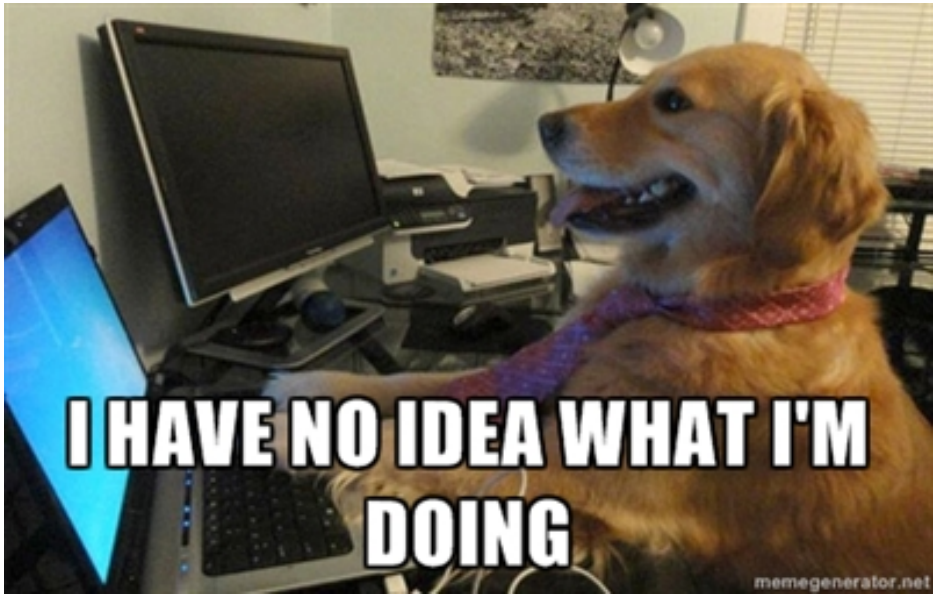
Output X



Output X

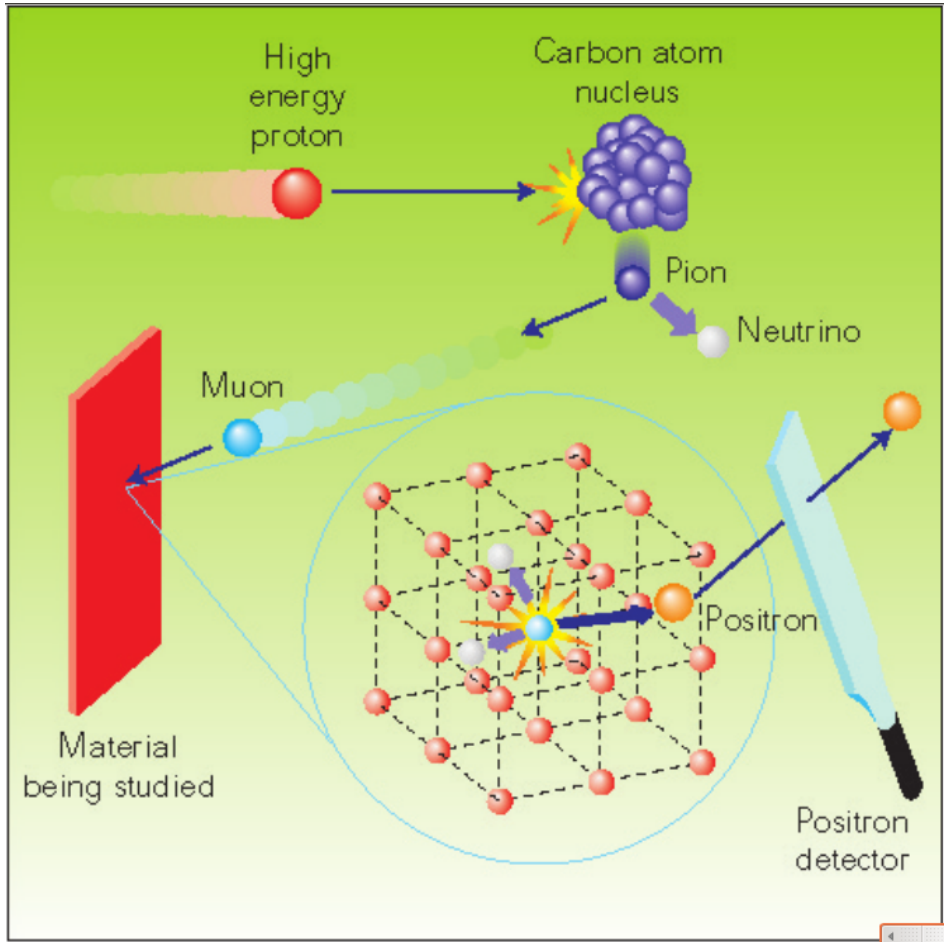
Appendix

Zweispaltige Sachen



1. Start
2. Stopp

General principle of μSR

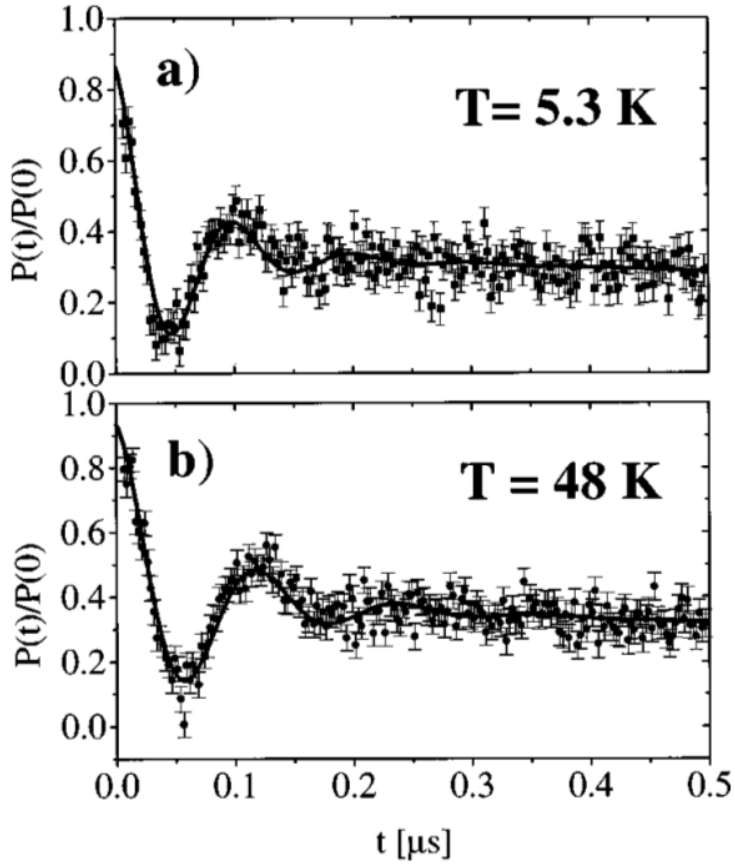


Dalmas de Réotier, Pierre (2010): *Introduction to muon spin rotation and relaxation (μSR)* [Online]. Availible: `http://inac.cea.fr/Pisp/pierre.dalmas-de-reotier/introduction_muSR.pdf`

Coexistence of ferromagnetism and superconductivity in $RuSr_2GdCu_2O_8$

- ferromagnetic phase is homogenous on a microscopic scale
- it accounts for most of the sample volume
- magnetic order is not significantly modified at the onset of superconductivity

C. Bernhard, J. L. Tallon, Ch. Niedermayer, Th. Blasius, A. Golnik, E. Brücher, R. K. Kremer, D. R. Noakes, C. E. Stronach, and E. J. Ansaldo, Phys. Rev. **B** 59, 14099 (1999)



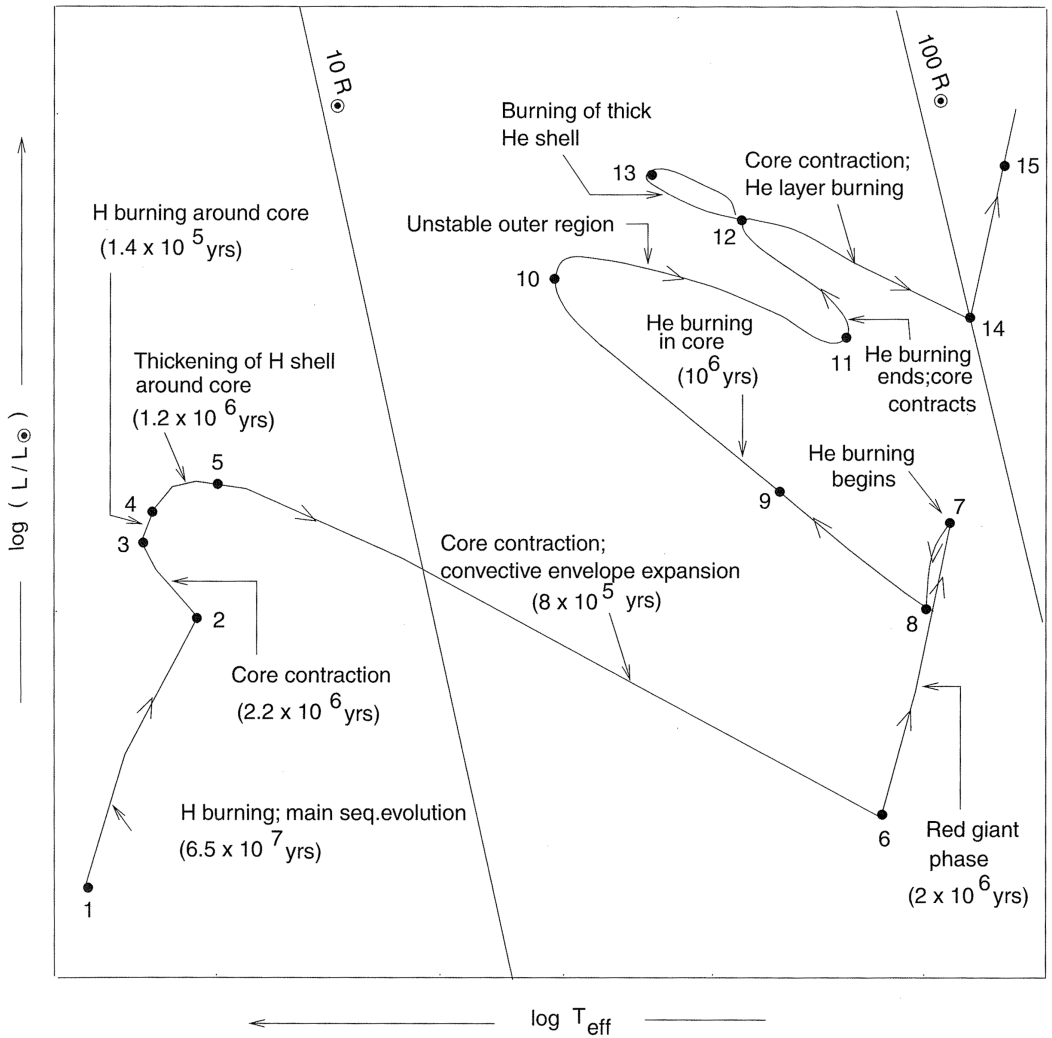
Time-resolved normalised muon-spin polarisation $P(t)/P(t=0)$ at temperatures $T = 5.3K < T_{c,sc}$ and at $T_{c,sc} < T = 28K < T_{c,m}$. The large oscillatory component gives clear evidence for the presence of a magnetically ordered state.

5 - 6
He core is homogenous (convective mixing).
It will be nearly isothermal.

More and more *He* is produced by shell burning, the core becomes more massive

At some point, core cannot support envelope mass anymore:

$\Rightarrow$ core contracts, envelope expands



T. Padmanabhan, "Theoretical Astrophysics Volume II: Stars and Stellar Systems". New York: Cambridge University Press, 2001.