HPCSE II Project # 1
Interaction of two Lamb-Oseen Vortices

Franziska Krummenacher and Ramona Hohl
Due: 14.08.16

We have developed a flow solver of Lamb-Oseen vortex dynamics of an incompressible fluid on an unbounded domain. The simulations are on a 2D domain of $[0,1]^2$ with a mesh of N^2 [1] nodes. In order to do so we have implemented the equations (2) to (5) of the remeshed vortex method in five kernels. Our code also consists of the solver for (1), the fast multipole Poisson solver which was developed during the semester.

Implementation

The general structure of our project is as following. The class *Vortexsimulation* contains all used constants/parameters and the most important method called *advance(),* which is called every time step and calls the kernels form *kernel2*. It also has a member of class *Cellist*, which is used for better performance in the PSEM kernel. The number of particles may defer from the number of mesh points, however after the first iteration (convections step to be accurate), they are the same.

The Poisson equation for the stream function (2) is solved with the multipole Poisson solver, which takes the particles and their vorticity. The kernel *Poisson()* builds the specific tree data structure and calls *streamfunction()*. In the end it returns the stream function on the grid points. (I am not going into further detail here, since it was "given".)
The velocity field equals the curl of the stream function (3), we have implemented this by calculating the central finite differences on the mesh points of the stream function. To maintain the boundary conditions, we take the backward and forward differences on the boundary of our mesh/domain accordingly.
For the next step of the algorithm, the computation vorticity field (4), we first implemented the interpolations scheme as described in P2M kernel in Project #5. However, we have realized that it is slow - $O(N^3)$, so we calculate the vorticity field by taking the curl, as we have for the velocity in step (3), which is only $O(N^2)$ and can be better parallelized.
For the viscous step we implemented the particle strength exchange method as described in Project #3. We decided to implement it with the PSEM, because we initially implemented periodic boundary conditions and then the ADI Peaceman-Racheford would result with a matrix system which can't be solved with the Thomas algorithm. With the Thomas algorithm, which can be used for perfect tridiagonal systems the ADI Peaceman-Racheford would have only taken O(N) per time step. Our PSEM implementation uses our *Cellist*, which only has to be computed once ($O(N^2)$), at the initialization of the *Vortexsimulations* class, because we are using a fixed mesh. We have learned in the lecture that a good algorithm is a very important aspect of optimization, and should be preferred over a parallelized slow algorithm, which is why we wrote the class *Cellist*. It allows to compute the interaction for each particle of a cell only with the particles of the surrounding eight cells and the current one. Considering this and our symmetrical implementation, we have an order of O(9*M*p -1). Where p << N is the number of particle per cell and M < N^2 is the number of cells. A normal PEM would be $O(N^4)$. The number of cells M depends on the cutoff[2]

[1] N is the number of mesh points per direction.
[2] The cutoff is defined as 5*eps where eps=2*dx, dx= 1/N -> cutoff= 10/N

which regulates the accuracy and also how much the particles interact. We decided that this kind of accuracy regulation is quiet nice to have, which is another reason why we chose the PSEM over ADI Peaceman-Racheford (second order accurate in time and space).
The convection step (5) we have implemented as given, with the forward Euler scheme. We treat each grid point as a particle which will carry the vorticity. In order to avoid numerical explosions and maintain a stable result, we calculate dt, the Lagrangian time step criterion every time step before calling the convection step. It is computed from the velocity field according to (9). To ensure open boundary condition we also cut off the boundaries at 0.1*M and 0.9*M for both directions.
The particles and the vorticity which they carry, are then put into the multipole Poisson solver where they will be remeshed in the next time step. After the simulation the memory the deconstructor calls *free* in order to free the aligned memory.

Optimization

The Poisson multipole solver has already been vectorized and parallelized with Open MP. The fast multipole method (FMM) Poisson solver scales very well. (see Figure 2)
We have used Open MP for-loop parallelization for the finite differences schemes (vorticity and velocity field). It resulted in a good scaling up to 10 cores for N=100. (see Figure 1). The vorticity scales better than the velocity because we write the calculated vorticities in a temporary array, whereas for the velocities we directly write it into the arrays. (see Figure 4)
We implemented a symmetric scheme for the PSEM which made it easy to parallelized it by using an Open MP-for for the for-loop over the cells without race conditions. The PSEM was also optimized by using a cellist as described above. It scales also very well, see Figure 3. We are very glad that we chose the PSEM because of this good scaling.
The convection step and the rescaling for the Poisson solver we have written in x86 AVX intrinsics, in order to achieve DLP with a theoretical four times faster result, it is very fast and as expected it doesn't scale. For it to scale with the number of cores, we could have added a OpenMp for, since it is already very fast, we chose not to.
Further optimization has been done by reading/writing the data with *fread/fwrite*, by aligning the array correctly using *posix_memalign*, by avoiding reciprocals and by pre-calculating factors.
We have done a strong scaling up to 20 cores:

Figure 1

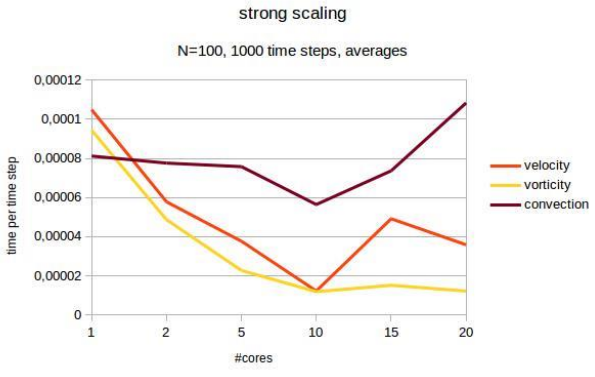


Figure 2

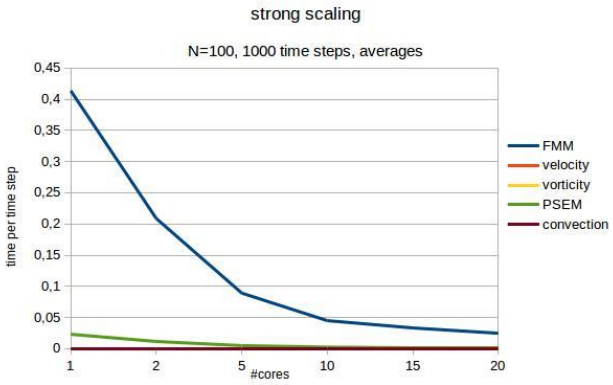


Figure 3

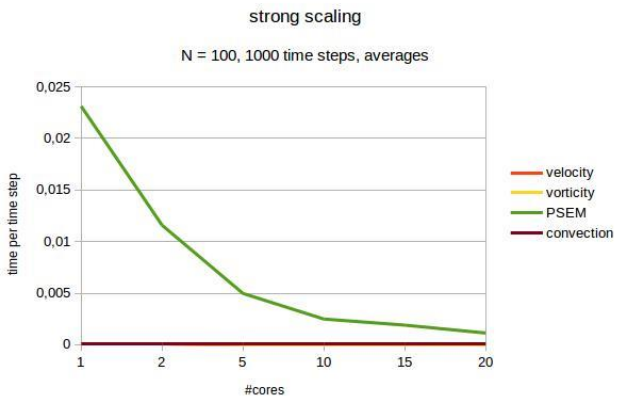
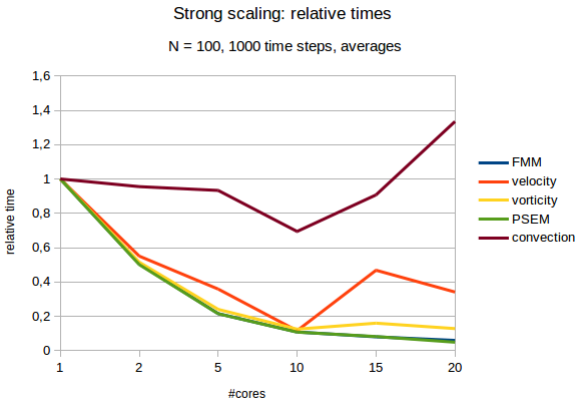


Figure 4



Initial Conditions and Analytical Solution

We calculated the initial conditions and the analytical result in vortex_generator.cpp and doublevoritces_generator.cpp. The initial vorticities are computed using the Gaussian distribution as in equation (10) and (11). They are also scaled by a factor of $1/(2*PI*N^2)$ because the given Poisson solver needs scaling. The values for the coresize were adjusted to our domain, which means that omega0 is 0.1 and the strength was reduced by choice to 5.0.

The comparison from our result to the analytical solution with viscosity of 0.1, coresize of 0.1, strength of 5.0 and at time step 0.220344 is visualized in Figure 4. We do not see a perfect correlation, the results is not as smoothed as the solution, but the order is correct. A reason for the unsmooth result can be, that because we set the boundaries (the small values) of the initial condition and later for the iteration to zero for the Poisson solver.

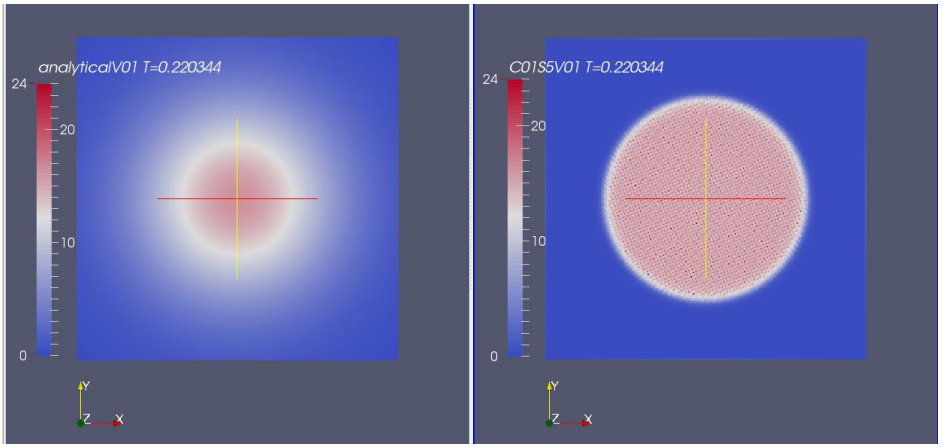


Figure 5

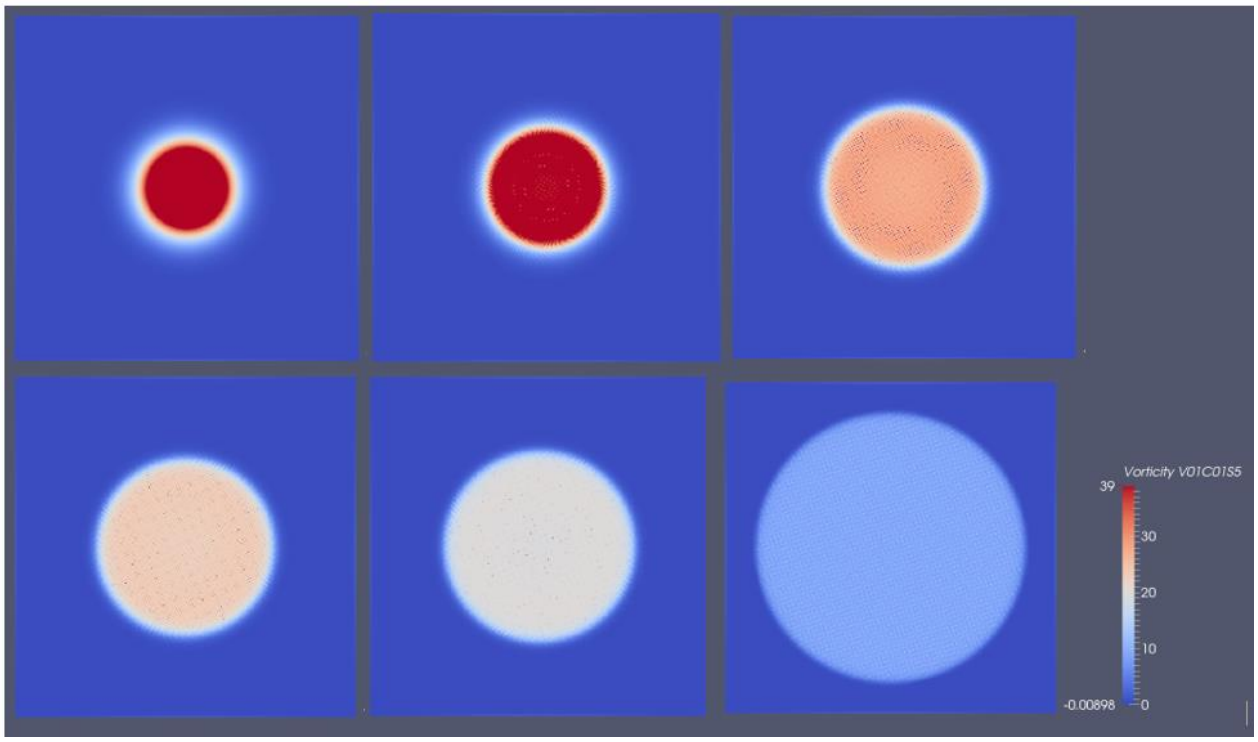
Simulations and Results

We have simulated the double vortices with the same strength (see below) as well as with opposite strength, which results in a slow movement of both vortices in -y direction. See also the attached animations.

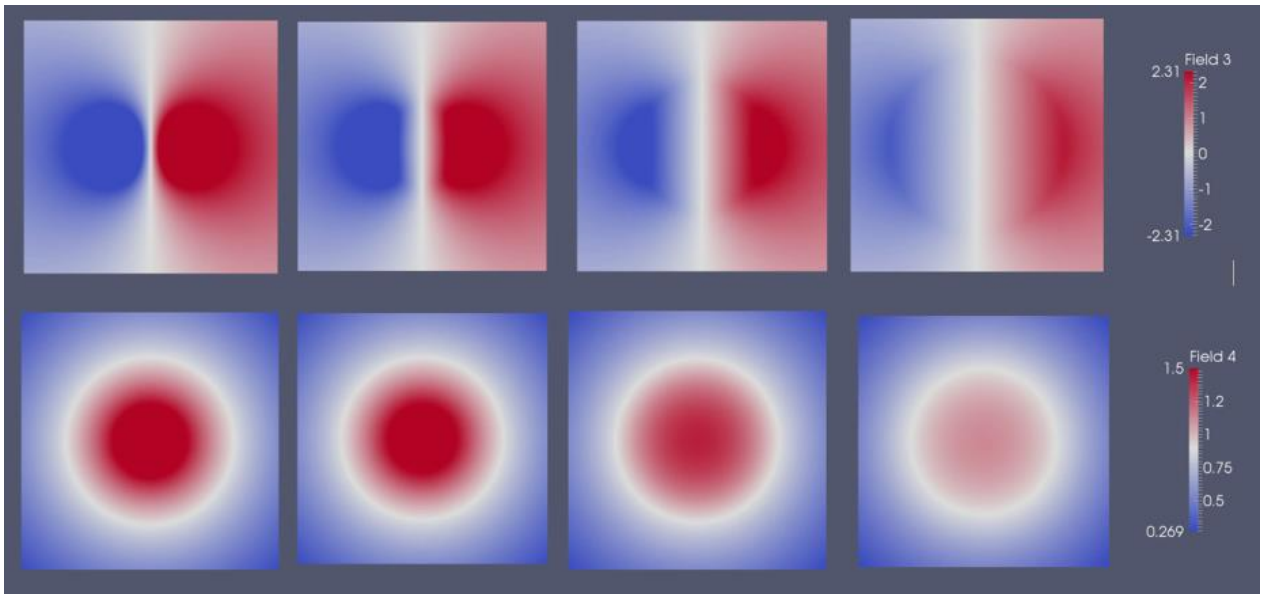
The core size was given as 1.0, however we simulate on a domain of $[0,1]^2$, so we scaled it down to 0.1. There is also some images of the velocity in y direction (x is the same but turned by 90°) and of the potential. The potential smooths down as expected.

Single Vortex: with core size=0.1, strength=5.0, viscosity=0.1

Vorticity Figure 6

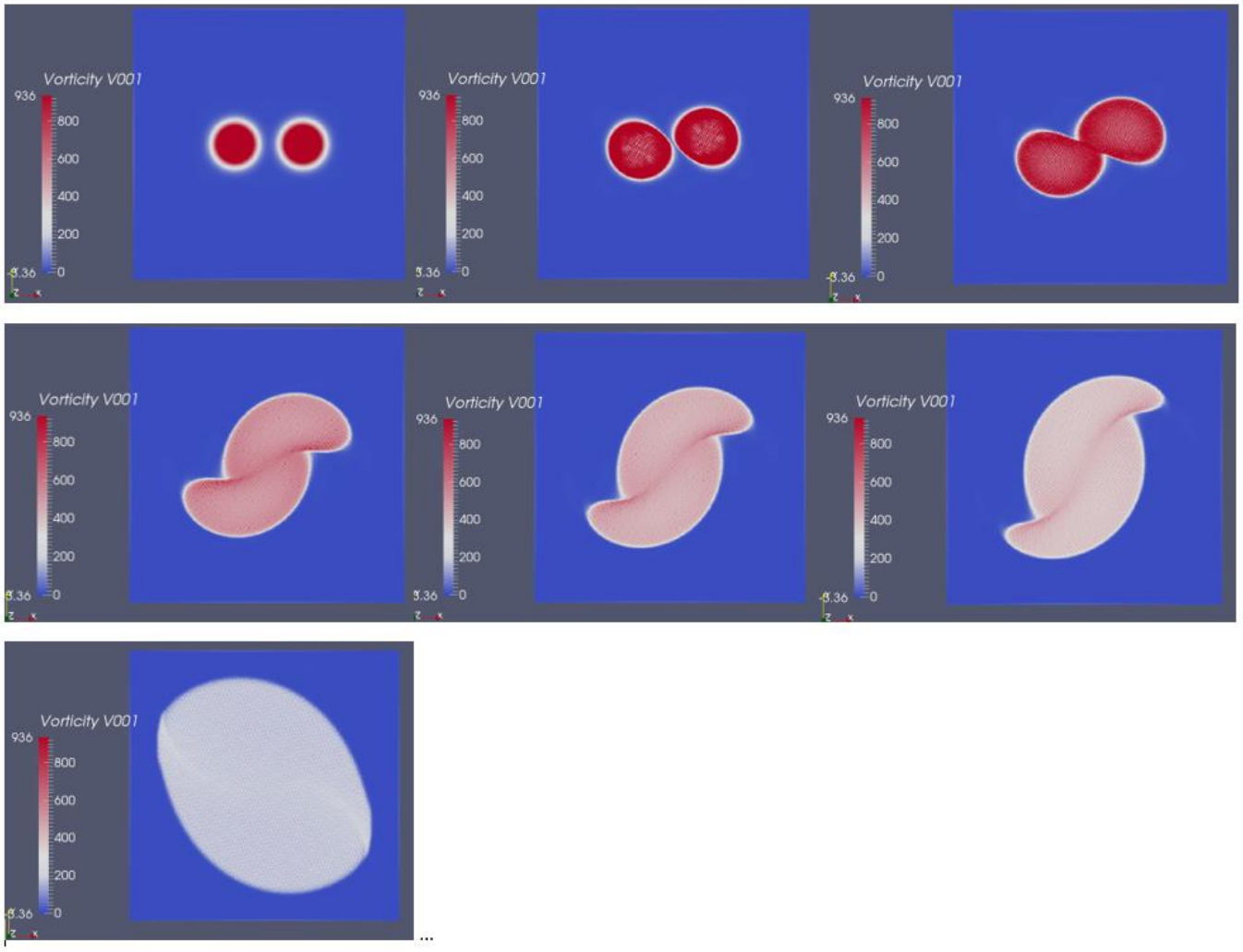


Velocity in y and Potential Figure 7


Field 3
2.31
2
1
0
-1
-2
-2.31
Field 4
1.5
1.2
1
0.75
0.5
0.269

Double Vortices: with core size=0.1, strength=5.0, viscosity=0.01 with a distance of 5*core size

Figure 8



Conclusion

we are mostly satisfied with our results. We had some trouble rendering the animation of the results because Paraview could not always handle the amount of data (N=300) on our laptops nor on my PC at home. It took us quite some time just to visualized them, unfortunately we could not render the animation where we compare the viscosities.

Nevertheless we think that we have achieved a beautiful simulation of the Lamb-Oseen vortices interaction. There is also to say we have definitely learned a lot with this project and we had the chance to use and apply the knowledge we have achieved during these two semesters.