Published in the proceedings of the Parallel Computing 2003 Conference (ParCo2003), Sep. 2 – 5, Dresden, German.
www.parco.org and www.tu-dresden.de/zhr/ParCo2003/

Parallel Simulation of Cavitated Flows in High Pressure Systems

Panagiotis A. Adamidis[a], Frank Wrona[b], Uwe Iben[b], Rolf Rabenseifner[a] and Claus-Dieter Munz[c]

[a]High-Performance Computing-Center Stuttgart, Allmandring 30, D-70550 Stuttgart, Germany

[b]Robert Bosch GmbH, Dept. FV/FLM, P.O. Box 106050, D-70059 Stuttgart

[c]Institute for Aero- and Gasdynamics (IAG), Pfaffenwaldring 21, D-70550 Stuttgart, Germany

This paper deals with the parallel numerical simulation of cavitating flows. The governing equations are the compressible, time dependent Euler equations for a homogeneous two-phase mixture. The equations of state for the density and internal energy are more complicated than for the ideal gas. These equations are solved by an explicit finite volume approach. After each time step fluid properties, namely pressure and temperature, must be obtained iteratively for each cell. The iteration process takes much time, particularly if in the cell cavitation occurs. For this reason the algorithm has been parallelized by domain decomposition. In case where different sizes of cavitated regions occur on the different processes a huge load imbalance problem arises. In this paper a new dynamic load balancing algorithm is presented, which solves this problem efficiently.

1. Introduction

Cavitation is the physical phenomenon of phase transition from liquid to vapor and occurs in a vast field of hydrodynamic applications. The reason for fluid evaporation is that the pressure drops beneath a certain threshold, the so called steam pressure. Once generated, the vapor fragments can be transported through the whole fluid domain, as been depicted in Fig. 1. Finally they are often destroyed at rigid walls which leads to damages.

Cavitation can also occur in various forms. Sometimes, they appear as large gas bubbles like in cooling systems of power plants. The work in this paper is extended for high pressure injection systems, where the fluid is expanded from a pressure of 1800 bar to nearly 10 bar. In such systems cavitation occurs as small vapor pockets and clouds. Due to the structure of cavities, the assumption that the flow field is homogenous, i.e. pressure, temperature and velocity of both phases are the same, is justified.

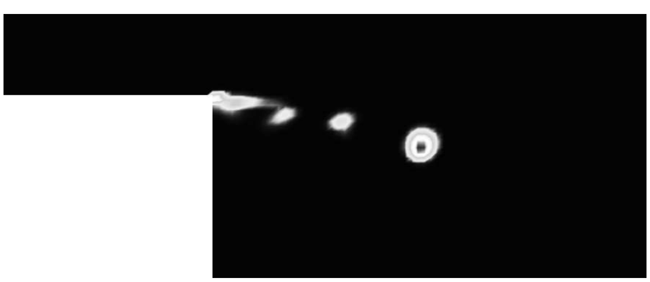


Figure 1: Cavity formation behind a backward-facing step

The more complicated challenge is to model the cavitation process, in a fashion that it is valid for all pressure levels, which can occur in such injection systems. Therefore the fluid properties are described by ordinary equations of state [1,2]. Among other things, this guarantees that the most important fluid property for modeling cavitation, e.g. the pressure, is always positive.

As mentioned before, the complete flow field is treated as compressible, even if the fluid does not evaporate. Additionally, enormous changes in the magnitude of all flow properties occur, if the fluid is cavitating. Hence, the governing equations can only be treated explicitly, but this limits the time step size strongly. Moreover, some fluid properties can just be obtained iteratively. Wherever cavitation occurs, this iterative process takes much more time than for the non cavitating case. This leads to very large computation times if one wish to solve larger problems.

Due to these facts parallization of the algorithm is unavoidable. But for an efficient parallel run the load imbalance problem, introduced by the cavitating cells, has to be solved. In the next section, the governing equations are presented. In section 3, the numerical algorithm and in section 4, the parallelization and the solution of the load imbalance problem are described. Finally in section 5, results of the parallel tests are presented.

2. Governing equations

For showing the main difficulty in the parallel running case, viscous terms can be neglected. Further, the regarded fluid is water and all necessary functions can be obtained from [3]. Therefore the governing equations are the two dimensional Euler equations, symbolically written as

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x + \mathbf{g}(\mathbf{u})_y = 0, \tag{1}$$

with

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho v \\ \rho w \\ E \end{pmatrix} \quad \mathbf{f}(\mathbf{u}) = \begin{pmatrix} \rho v \\ \rho v^2 + p \\ \rho v w \\ v(E+p) \end{pmatrix} \quad \mathbf{g}(\mathbf{u}) = \begin{pmatrix} \rho w \\ \rho v w \\ \rho w^2 + p \\ w(E+p) \end{pmatrix}. \tag{2}$$

Here derivatives are denoted by an index. In Eq. (2) ρ is the density, of the homogeneous mixture, v and w the velocity in x-, respectively in y-direction. Further the property E is introduced, which describes the total energy $\rho(e + 1/2(v^2 + w^2))$ per unit volume. The density and the internal energy e are functions of the pressure p and the temperature T.

The density and the internal energy are expressed as mixture properties

$$\frac{1}{\rho} = \frac{\mu}{\rho_{\mathrm{G}}} + \frac{1-\mu}{\rho_{\mathrm{L}}} \quad \text{and} \quad e = \mu e_{\mathrm{G}} + (1-\mu)e_{\mathrm{L}}. \tag{3}$$

Here the properties with the subscript G are the one of the gaseous phase and with the subscript L of the liquid phase. The gaseous phase is treated as ideal gas and, the functions of the liquid phase are obtained from the IAPWS97 [3]. The mass fraction is defined by

$$\mu = \frac{m_{\mathrm{G}}}{m_{\mathrm{G}} + m_{\mathrm{L}}}, \tag{4}$$

and describes the fractional mass portion of the gaseous phase to the total mass in a cell. Finally to close the system the mass fraction must also be expressed as a function of pressure and temperature. With the assumption that the mixture of liquid and gas, also called steam, is always in thermodynamical equilibrium and that the fluid evaporates at constant entropy, the mass fraction can be expressed as

$$\mu(p,T) = \frac{h(p,T) - h'(p)}{h''(p) - h'(p)}. \tag{5}$$

The enthalpies h, h'' and h' can also be obtained from the IAPWS97 [3]. If the actual enthalpy h is less than the enthalpy h' at the boiling line, μ is set to zero, because the fluid is outside of the two-phase regime and consists of pure liquid. This happens, when the pressure is greater than the steam pressure p_{steam}, which depends only on the temperature. Similar to the mass fraction a void fraction can be defined, and can be calculated from Eq. (3); it can be used to detect cavitating cells:

$$\varepsilon = \frac{V_{\mathrm{G}}}{V_{\mathrm{G}} + V_{\mathrm{L}}} \quad \text{and} \quad \varepsilon = \mu \frac{\rho}{\rho_{\mathrm{G}}}. \tag{6}$$

3. Numerical solution

For solving the conservation of mass, momentum and energy numerically, a transient finite volume approach is used, which calculates the system of conservative variables at each time step explicitly.

The underlying mesh is unstructured and consists of triangles and rectangles. Therefore Eq. (1) is discretized as

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + \frac{\Delta t}{\Omega_i} \sum_{j \in \mathcal{N}(i)} \mathcal{L}(\mathbf{u}_i^n, \mathbf{u}_j^n) \tilde{f}(\mathbf{u}_i^n, \mathbf{u}_j^n) l_{ij}, \tag{7}$$

where $\mathcal{N}(i)$ are the set of the neighbor cells of the ith cell and Ω_i its volume. $\mathcal{L}(\mathbf{u}_i^n, \mathbf{u}_j^n)$ denotes an operator for different time integration methods. The numerical flux $\tilde{f}$ depends on the conservative variables in the ith cell and its neighbors. This fluxes are calculated by approximate Riemann solvers, namely the HLLC-Solver [4]. Note that Eq. (7) is a strongly simplified presentation of the whole method. For more information refer [5].

After the flux calculation and the update of the conservative variables from $\mathbf{u}_i^n$ to $\mathbf{u}_i^{n+1}$, the primitive variables must be computed for each cell. For some properties this is very easy, because they can be obtained analytically (the subscripts i and superscripts $n+1$ are dropped for convenience)

$$\rho = u_1, \quad v = \frac{u_2}{\rho}, \quad w = \frac{u_3}{\rho} \quad \text{and} \quad e = \frac{u_4}{\rho} - \frac{1}{2}(v^2 + w^2). \tag{8}$$

Now the pressure and the temperature for every cell is needed. But as mentioned before they can only be computed iteratively. This can be done by an iteration for the formulas in Eq. (3), because the internal energy and the density are already known

$$h_1(p,T) = \frac{1}{\rho} - \frac{\mu}{\rho_{\mathrm{G}}} - \frac{1-\mu}{\rho_{\mathrm{L}}} = 0 \quad \text{and} \quad h_2(p,T) = e - \mu e_{\mathrm{G}} - (1-\mu) e_{\mathrm{L}} = 0. \tag{9}$$

At the beginning h_1 is iterated with fixed pressure for the new temperature by a bisection method. Afterward, with the new value for the temperature, h_2 is iterated for the new pressure also by a bisection. These two steps are repeated until both values converge. Unfortunately these two values converge slower if cavitation arises in the cell.

4. Parallel Algorithm

The iterative method described above has been proven to be the most CPU time consuming part of the whole simulation. Especially, the time needed for cells in which cavitation occurs is much more than the one needed by the others, because there are more iterations executed in order to determine pressure and temperature. This leads to enormous computation times for large realistic problems, which are in the range of several days or weeks.

To solve such problems, parallelization of the algorithm is unavoidable, thus taking advantage of more processors and memory space of a parallel computing environment. As mentioned before, the numerical method is an explicit scheme. The natural approach to parallelize such an algorithm is domain decomposition. The whole computational domain is decomposed into several subdomains equal to the number of available processors. This initial partitioning is done using the tool METIS [9].

The whole algorithm consists of several parts, as can be seen in Fig. 2. The calculation of the fluxes is carried out by each processor on the subdomain which has been assigned to it. For calculations on cells lying on the artificial boundary of each subdomain, data from their neighboring cells are needed. Because these neighbors belong to adjacent subdomains, the subdomains are expanded. The new additional layer, the so called halo layer, contains the aforementioned neighboring cells. The data for the halo cells is being communicated using MPI [8].

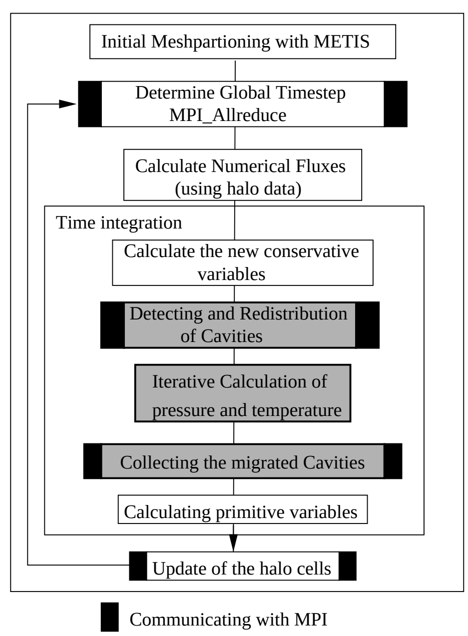


Figure 2: Flow chart of parallel algorithm

However, the appearance of cavitation affects the iterative calculation of pressure and temperature in the time integration part in two ways. On one hand, the cavitating cells are not distributed homogeneously over the whole computational domain, which means that subdomains having more such cells need much more CPU time to finish their calculations, so that subdomains with less cells have to wait for them. This causes a very significant load imbalance. On the other hand, the locations of the cavities move across subdomains.

The approach followed in this work keeps the same initial partitioning, and reassigns only the work done in the iterative process of determining pressure and temperature from partitions with more load to partitions with less load, see shaded boxes in Fig. 2. Thereby, only the information needed to determine pressure and temperature of a cell is moved to another process and the result is recollected to the original location of the cell, because the cell itself is not moved to the other process. In particular, first we store the iteration time t_i needed by each cell i. Next, the iteration time t_{p_j}, spend for each original partition j is determined by adding up the iteration time of its cells

$$t_{\mathrm{p}_j} = \sum_{i=1}^{ncell_{\mathrm{j}}} t_i, \qquad (10)$$

where $ncellj$ is the number of cells in partition j. The optimal CPU time distribution would now be

$$t_{opt} = \frac{1}{nprocs} \sum_{j=1}^{nprocs} t_{\mathrm{p}_j} \qquad (11)$$

where $nprocs$ stands for the number of processes. With the assumption that in the next time step the cell times are slightly different we determine the time difference

$$t_{diff,j} = t_{\mathrm{p}_j} - t_{opt}, \qquad (12)$$

for each process.

Depending on whether $t_{diff,j}$ is greater than, less than or equal zero, the ith process is going to migrate part of the calculations done on its cells, or will be a receiver of such a workload, or will not reassign any work. In this way processes are classified as "senders" and "receivers". Sending processes, are sending workload to the receiving processes until all processes have approximately reached the optimum CPU time t_{opt}. In each time step, the above described decision-making is based upon the CPU times measured in the previous time step. With this migration, only a small part of the cell-information must be sent. All other cell-information remains at the original owner of the cell. Halo data is not needed, because the time-consuming iterative calculation of pressure and temperature is done locally on each cell.

Each process calculates its own t_{p_j} value. This values are communicated by an all-gather operation to the other processes. Thus, each process determines the number, the indices of the cells and the receiving processes to which it will send the necessary information in order to have pressure and temperature calculated. After this calculation the results are sent back to the owners of the cells, and the remaining of the calculations is executed on the initial partitioning of the mesh. The computation time needed for a cavitating cell is about 1 ms and only about the fourth part is needed for a non cavitating cell, on an Intel Xeon 2GHz processor. Each sending process must send 32 Bytes (4 doubles) of data per cell to the receiving processes, and must receive 16 Bytes (2 doubles) of results. This means that 48 Bytes per cell must be communicated. Due to the small number of bytes for each cell, the approach of transferring workload implies only a very small communication overhead. On a 100 Mbit/s Ethernet a communication time of about 4.3 μs is needed.

Whereas, treating the introduced load imbalance by repartitioning the whole mesh, or by diffusion schemes, as it is done by state of the art strategies [6,7] means that cells would have to be migrated to other partitions, and the amount of data which would have to be communicated, is 1536 Bytes (32 integers and 176 doubles) per cell. This means a communication time of about 140 μs on a 100 Mbit/s Ethernet, which is 32 times more than with our approach. Considering the fact that this redestribution would have to be carried out in each time step, due to the steadily chainging number of cavitating cells, it is obvious that the overhead introduced by the entire redistribution of cells is much more than the overhead of our strategy, which temporarily redistributes partial cell information. In this way, we also avoid expanding the halo layer.

5. Results

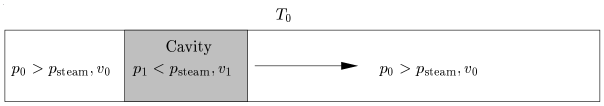


Figure 3: Schematic illustration of the benchmark

As benchmark, a shock tube problem is defined. The tube has a length of one meter. The mesh is a Cartesian grid with ten cells in y-direction and 1000 in x-direction. The computational domain is initialized as illustrated in Fig. 3. It consists of a cavity, which is embedded in pure liquid. This can be managed by setting the whole flow field with the same temperature and the same pressure, except in the cavitated region the pressure is chosen that it is below the steam pressure. The cavity has a thickness of 0.1m and its beginning is positioned at 0.2m after the tube entry. Further the flow field is set up with a velocity v_0 in the right direction and in the cavitated region with $v_1 = v_0\rho_0/\rho_1$. If the velocity is set up near to the speed of sound, the cavity must be transported in the right direction. Therefore it is an excellent benchmark for checking the load balance algorithm, because the position of the cavity moves from subdomain to subdomain. Results and the domain decomposition for a parallel run with eight processors are shown in Fig. 4.

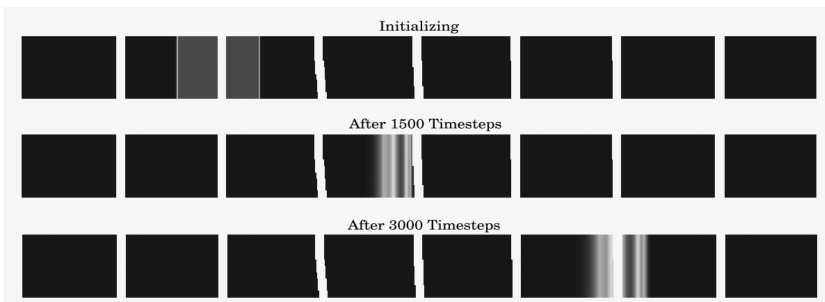
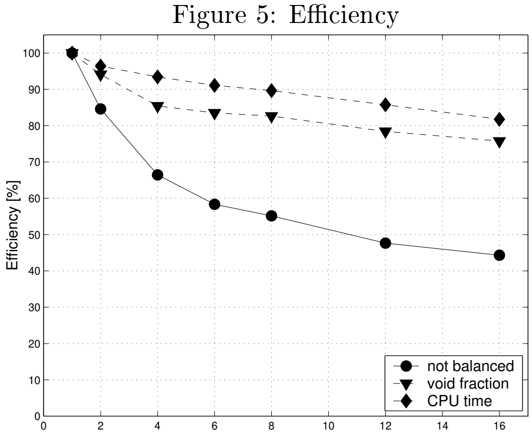


Figure 4. Void fraction (bright spots) and domain decomposition of the benchmark

Table 1
CPU Time in seconds and Speedup

	# Prozesse	1	2	4	6	8	12	16
not load-balanced	CPU Time	9622.34	5685.98	3620.49	2748.64	2180.57	1682.86	1356.80
	Speedup	1	1.692	2.658	3.501	4.413	5.718	7.092
load balanced void fraction	CPU Time	-	5112.81	2816.00	1920.45	1455.54	1022.48	793.907
	Speedup	-	1.882	3.417	5.010	6.611	9.411	12.120
load balanced CPU time	CPU Time	-	4990.32	2574.45	1760.46	1341.62	935.26	735.52
	Speedup	-	1.928	3.738	5.466	7.172	10.288	13.082

Figure 5: Efficiency



The computing platform is a cluster consisting of dual-CPU PCs, with Intel Xeon 2 GHz processors. The results are summarized in Tab. 1 and Fig. 5 for several parallel runs. Further, they are compared with the load balancing approach, we used in [10] where the decision-making was based on the void fraction. From these results it is obvious that the gain in efficiency, achieved by the algorithm of this work is greater than the previous one. Without the scpecific load balancing to handle cavitating cells, only a poor efficiency (58% to 44% for 6 to 16 processors) could be reached. With the new approach the efficiency is over 91% up to 6 processors and remains over 81% up to 16 processors. Whereas, using the void fraction approach the efficiency drops below 84% when using 6 processors and below 76% at 16 processors.

6. Conclusion

In many industrial applications cavitation occurs and cannot be calculated detailed in a proper response time on a single processor computer. In order to reduce this time the use of parallel computing architectures is necessary. Nevertheless, the parallel algorithm has to deal with load imbalance introduced by the cavities. The most time consuming part of the algorithm is the iterative calculation of pressure and temperature. In this work a new load balancing algorithm has been developed, in which not cells, but the work done on the cells during determination of pressure and temperature, is redistributed across processors. The initial partitioning of the mesh is not changed. The decision-making for migrating workload is based upon the CPU time needed by a process to calculate pressure and temperature. The results of this algorithm show that there is a significant gain in efficiency. With the new approach it is possible to treat industrial problems, in the area of simulating the flow in injection systems, in a reasonable response time.

REFERENCES

[1] U. Iben, F. Wrona, C.-D. Munz, M. Beck, Cavitation in Hydraulic Tools Based on Thermodynamic Properties of Liquid and Gas, Journal of Fluids Engineering, 2002, Vol. 124, No. 4, pages 1011-1017.

[2] R. Saurel, J.P. Cocchi, P.B. Butler, Numerical Study of Cavitation in the Wake of a Hypervelocity Underwater Projectile, J. Prop. Pow., 1999, Vol. 15, No 4.

[3] W. Wagner et. al., The IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam, J. Eng. Gas Turbines and Power, 2000, Vol. 12.

[4] P. Batten, N. Clarke, C. Lambert, D.M. Causon, On the choice of wavespeeds for the HLLC Riemann solver, SIAM J. Sci. Comp., 1997, Vol. 18, No. 6, pp. 1553-1570.

[5] E.F. Toro, Riemann Solvers and Numerical Methods for Fluid Dynamics, Springer New York Berlin Heidelberg, 1997.

[6] C. Walshaw, M. Cross, M. G. Everett, Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes, Journal Parallel Distrib. Comput, pp. 102-108, Vol.47, No. 2, 1997.

[7] Kirk Schloegel, George Karypis, Vipin Kumar, Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes, Journal of Parallel and Distributed Computing, Vol. 47, pp. 109-124, 1997.

[8] M. Snir, S. Otto, S. Huss–Ledermann, D. Walker, J. Dongarra MPI The Complete Reference, The MIT Press, 1996.

[9] G. Karypis, V. Kumar, METIS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, University of Minnesota, Department of Computer Science / Army HPC Research Center, 1998.

[10] Frank Wrona, Panagiotis A. Adamidis, Uwe Iben, Rolf Rabenseifner, Dynamic Load Balancing for the Parallel Simulation of Cavitating Flows, In Recent Advances in Parallel Virtual Machine and Message Passing Interface, Jack Dongarra, D. Laforenza, and S. Orlando (Eds.), Proceedings of the 10th European PVM/MPI Users' Group Meeting, EuroPVM/MPI 2003, Sep. 29 - Oct. 2, Venice, Italy.