Published 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.

Dynamic Load Balancing for the Parallel Simulation of Cavitating Flows

Frank Wrona[2], Panagiotis A. Adamidis[1], Uwe Iben[2], Rolf Rabenseifner[1], Claus-Dieter Munz[3]

[1] High-Performance Computing-Center Stuttgart (HLRS), Allmandring 30, D-70550 Stuttgart, Germany {adamidis, rabenseifner}@hlrs.de
[2] Robert Bosch GmbH, Dept. FV/FLM, P.O. Box 106050, D-70059 Stuttgart {frank.wrona, uwe.iben}@de.bosch.com
[3] Institute for Aero- and Gasdynamics (IAG), Pfaffenwaldring 21, D-70550 Stuttgart, Germany munz@iag.uni-stuttgart.de

Abstract. 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. These equations are solved by an explicit finite volume approach. In opposite to the ideal gas, after each time step fluid properties, namely pressure and temperature, must be obtained iteratively for each cell. This is the most time consuming part, particularly if cavitation occurs. For this reason the algorithms 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

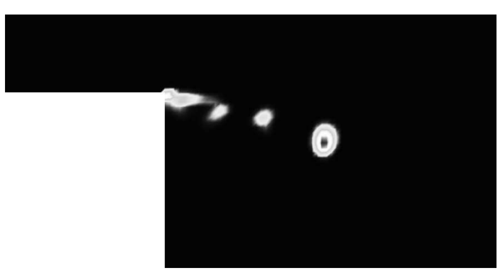

Fig. 1. Cavity formation behind a backward-facing step

Cavitation is the physical phenomenon of phase transition from liquid to vapor. 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. The work in this paper is extended for high pressure injection systems. 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. 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].

Further, 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. Therefore such simulations are very CPU time consuming and parallelization is unavoidable if one wants to calculate large problems.

2 Governing equations

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 $\mathbf{u} = (\rho, \rho v, \rho w, E)^T$, $\mathbf{f}(\mathbf{u}) = (\rho v, \rho v^2 + p, \rho v w, v(E+p))^T$ and $\mathbf{g}(\mathbf{u}) = (\rho w, \rho v w, \rho w^2 + p, w(E+p))^T$. Here, derivatives are denoted by an index. Further, ρ is the density, v and w the velocity in x-, respectively in y-direction. The property E is also 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 and are expressed as mixture properties

$$1/\rho = \mu/\rho_{\mathrm{G}} + (1-\mu)/\rho_{\mathrm{L}} \quad \text{and} \quad e = \mu e_{\mathrm{G}} + (1-\mu) e_{\mathrm{L}}. \tag{2}$$

The regarded fluid is water, where the gaseous phase (subscript G) is treated as ideal gas and the functions of the liquid phase (subscript L) are obtained from the IAPWS97 [2]. The mass fraction μ, the void fraction ε and their relation are defined by

$$\mu = m_{\mathrm{G}}/(m_{\mathrm{G}} + m_{\mathrm{L}}), \quad \varepsilon = V_{\mathrm{G}}/(V_{\mathrm{G}} + V_{\mathrm{L}}) \quad \text{and} \quad \varepsilon = \mu\rho/\rho_{\mathrm{G}} \tag{3}$$

For solving the governing equations numerically, eq.(1) is discretized as

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n + (\Delta t/\Omega_i) \sum_{j \in \mathcal{N}(i)} \mathcal{L}(\mathbf{u}_i^n, \mathbf{u}_j^n) \tilde{f}(\mathbf{u}_i^n, \mathbf{u}_j^n) l_{ij}, \tag{4}$$

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 underlying mesh is unstructured and consists of triangles and rectangles. The fluxes $\tilde{f}$ are calculated by approximate Riemann solvers, namely the HLLC-Solver [3]. Finally, the mass fraction – which describes the fractional mass portion of the gaseous phase to the total mass in each cell – must also be expressed as a function of pressure and temperature. 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 ρ, v, w, e can be computed analytically for each cell from the conserved quantities. However the pressure and the temperature for every cell cannot be calculated directly. Their values can be obtained iteratively from equations (2), because the internal energy and the density are already known

$$h_1(p,T) = 1/\rho - \mu/\rho_{\mathrm{G}} - (1-\mu)/\rho_{\mathrm{L}} = 0 \quad \text{and} \quad h_2(p,T) = e - \mu e_{\mathrm{G}} - (1-\mu) e_{\mathrm{L}} = 0. \tag{5}$$

These two equations are iterated by a two dimensional bisection method for p and T, until this two values converge. This step is the most time consuming part in the whole solution algorithm and even takes much more time if in a cell cavitation arises. Therefore simulations of realistic problems take several days. A more detailed description of the equations is presented in [1].

3 Parallel Algorithm

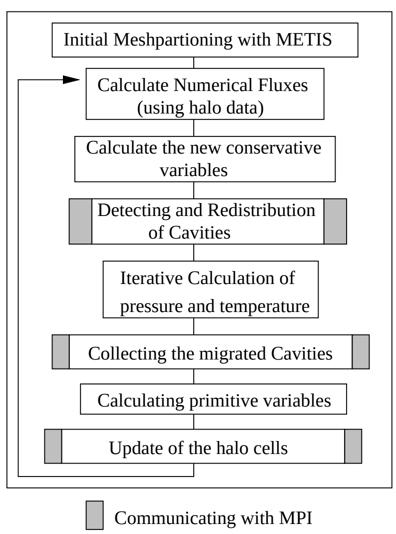

Fig. 2. Flow chart of parallel algorithm with dynamic load balance

In this work, we parallelize the algorithm by domain decomposition using the MPI paradigm. The target computing platform is a PC cluster. The phenomenon of cavitation affects the parallel algorithm in two ways. On one hand, the cavitating cells are not distributed homogeneously over the subdomains. This causes a very heavy load imbalance. On the other hand, the locations of the cavities move across subdomains. For this reason, using state of the art strategies [4] results in repartitioning at every time step. In our algorithm, only the work done on cavitating cells, during the iterative calculation of pressure and temperature (see Fig. 2), is redistributed at each time step. The parallel algorithm starts with an initial mesh partitioning, using the tool METIS [5] (see Fig. 2). After calculating the fluxes and conservative variables on the subdomains, cavities are detected by checking the void fraction ϵ. After this, every process knows how many cavitating cells reside in its subdomain. Denoting with cav_i this number for process i, the optimal number of cavities in each subdomain cav_{opt} is determined by

$$cav_{\mathrm{opt}} = 1/nprocs \sum_{i=1}^{nprocs} cav_i, \tag{6}$$

where $nprocs$ is the number of processes. Now the processes are classified into senders and receivers of cavities in the following manner: Depending on whether

$cav_i - cav_{\mathrm{opt}}$ is greater than, less than or equal zero, the ith process is going to migrate part of its cavitating cells, or will be a receiver of such cells, or will not redistribute any cell. 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.

The implementation is based on an all-gather communication which sends the cav_i values to all processes. The computation time needed for a cavitating cell is about 1 ms and only about the sixth part is needed for a non cavitating cell, on a PentiumIII 800 MHz processor. Furthermore, the senders take for each cavitating cell, which they move to a receiver, a corresponding non cavitating cell from the specific receiver. With this approach the number of cavitating and non cavitating cells is the same on each process, which leads to a well balanced system. Due to the small number of bytes for each cell, the approach of transferring non cavitating cells as compensation implies only a very small communication overhead. After this, the calculation of pressure and temperatures is carried out, and afterwards, 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. For this redistribution, 128 Bytes for each cell must be communicated, which means on a 100 Mbit/s Ethernet a communication time of nearly 10 μs. In this way we avoid expanding the halos, because we outsource only the time-consuming part of the computation done on the cavitating cells, and recollect the results.

4 Results

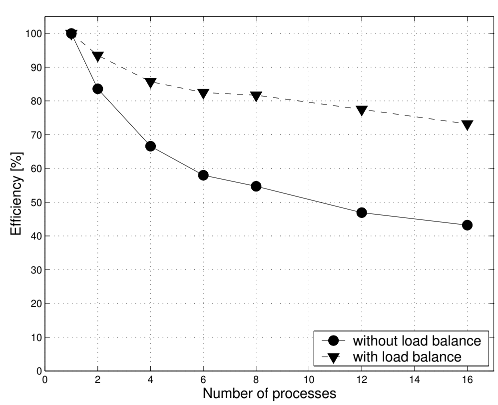

Fig. 3. Efficiency

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. At the initial step the computational domain consists of a cavity, which is embedded in pure liquid, and the initial velocity is directed in positive x-direction. Therefore it is an excellent benchmark for checking the load balance algorithm, because the position of the cavity moves from subdomain to subdomain. The computing platform was a cluster consisting of dual-CPU PCs, with Pentium III 1GHz processors. The results are summarized in Tab. 1 and Fig. 3 for several parallel runs. From these results it is obvious that a tremendous gain in efficiency has been achieved by the implemented load balancing strategy. In the runs without dynamic load balancing the efficiency drops down to 66.6% even when using only 4 processors and getting worse in the case of 16 processors (43.2%). In contrast the efficiency, with dynamic load balancing is over 80%, up to 8 processors, and in case of 16 processors still at 73.2%, which is a profit of 30% compared to the case without load balancing.

	# Processes	1	2	4	6	8	12	16
not load	CPU Time	12401.4	7412.12	4649.84	3558.7	2830.16	2201.29	1791.68
balanced	Speedup	1	1.67	2.66	3.48	4.38	5.63	6.92
load	CPU Time	-	6626.94	3616.51	2503.19	1897.14	1333.28	1058.19
balanced	Speedup	1	1.87	3.42	4.95	6.53	9.30	11.72

Table 1. CPU Time in seconds and Speedup

5 Conclusion

The simulation of cavitating flows is a CPU time demanding process. To obtain results in an adequate time it is necessary to use parallel computing architectures. In order to achieve high performance, the parallel algorithm has to deal with the problem of load imbalance, introduced by the cavities. In this work a new dynamic load balancing algorithm was developed, which treats this problem very efficiently. Future work will be concentrated on new criterions to detect the cavitating regions, and simulating 3D problems.

References

1. U. Iben, F. Wrona, C.-D. Munz, and M. Beck. Cavitation in Hydraulic Tools Based on Thermodynamic Properties of Liquid and Gas. *Journal of Fluids Engineering*, 124(4):1011–1017, 2002.
2. W. Wagner et al. The IAPWS Industrial Formulation 1997 for the Thermodynamic Properties of Water and Steam. *J. Eng. Gas Turbines and Power*, 12, January 2000. ASME.
3. P. Batten, N. Clarke, C. Lambert, and D.M. Causon. On the choice of wavespeeds for the HLLC Riemann solver. *SIAM J. Sci. Comp.*, 18(6):1553–1570, November 1997.
4. C. Walshaw, M. Cross, and M. G. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, 1997. (originally published as Univ. Greenwich Tech. Rep. 97/IM/20).
5. G. Karypis and V. Kumar. METIS:A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Technical report, University of Minnesota,Department of Computer Science / Army HPC Research Center, 1998.