4th International Conference on Multiphase Flow,
ICMF – 2001, May 27.–June 1. 2001,
New Orleans, Louisiana, U.S.A.

ASPECTS OF EFFICIENT PARALLELIZATION OF DISPERSE GAS–PARTICLE FLOW PREDICTIONS USING EULERIAN–LAGRANGIAN APPROACH

Th. Frank, K. Bernert, K. Pachler, H. Schneider
Chemnitz University of Technology, Faculty of Mechanical Engineering,
Research Group on Multiphase Flow & SIVUS gGmbH, 09107 Chemnitz, Germany.
Frank@imech.tu–chemnitz.de

Abstract—The paper deals with different methods for the efficient parallelization of Eulerian–Lagrangian approach which is widely used for the prediction of disperse gas–particle and gas–droplet flows. Several aspects of parallelization like e.g. scalability, efficiency and dynamic load balancing are discussed for the different kinds of Domain Decomposition methods applied to both the Eulerian and Lagrangian parts of the numerical prediction. The paper shows that remarkable speed-up's can be achieved on dedicated parallel computers and clusters of workstations (Beowulf systems) not only for idealized test cases but also for "real world" applications. Therefor the developed parallelization methods offer new perspectives for the computation of strongly coupled multiphase flows with complex phase interactions.

Key Words : CFD, disperse multiphase flows, Eulerian–Lagrangian approach, PSI–Cell method, parallel computing, Domain Decomposition

1 Motivation

Over the last decade the Eulerian–Lagrangian (PSI–Cell) simulation has become an efficient and widely used method for the calculation of various kinds of 2– and 3–dimensional disperse multiphase flows (e.g. gas–particle flows, gas–droplet flows) with a large variety of computational very intensive applications in mechanical and environmental engineering, process technology, power engineering (e.g. coal combustion) and in the design of internal combustion engines (e.g. fuel injection and combustion). Considering the field of computational fluid dynamics, the Eulerian–Lagrangian simulation of coupled multiphase flows with strong interaction between the continuous fluid phase and the disperse particle phase ranks among the applications with the highest demand on computational power and system recources. Massively parallel computers provide the capability for cost–effective calculations of multiphase flows. In order to use the architecture of parallel computers efficiently, new solution algorithms have to be developed. While there has been a larger effort in investigation of Domain Decomposition methods applied to fluid flow predictions there are only a very few publications on parallelization of Lagrangian particle tracking algorithms. Difficulties in parallelization arise from the complex data dependence between the fluid flow calculation and the prediction of particle motion, and from the generally non–homogeneous distribution of particle concentration in the flow field. Direct linkage between local particle concentration in the flow and the numerical work load distribution over the calculational domain often leads to very poor performance of parallel Lagrangian solvers operating with a Static Domain Decomposition method. Good work load balancing and high parallel efficiency for the Lagrangian approach can be established with the Dynamic Domain Decomposition method presented in this paper.

2 Physical And Mathematical Fundamentals

2.1 Basic equations of fluid motion

The fluid phase considered here is assumed to be Newtonian and to have constant physical properties. The fluid flow is 3–dimensional, steady, incompressible, turbulent and isothermal.

Φ	S_Φ	S_Φ^P	Γ_Φ
1	0	0	0
u_F	$\frac{\partial}{\partial x}\left(\Gamma_\Phi \frac{\partial u_F}{\partial x}\right) + \frac{\partial}{\partial y}\left(\Gamma_\Phi \frac{\partial v_F}{\partial x}\right) + \frac{\partial}{\partial z}\left(\Gamma_\Phi \frac{\partial w_F}{\partial x}\right) - \frac{\partial p}{\partial x} + \rho_F f_x$	$S_{u_F}^P$	μ_{eff}
v_F	$\frac{\partial}{\partial x}\left(\Gamma_\Phi \frac{\partial u_F}{\partial y}\right) + \frac{\partial}{\partial y}\left(\Gamma_\Phi \frac{\partial v_F}{\partial y}\right) + \frac{\partial}{\partial z}\left(\Gamma_\Phi \frac{\partial w_F}{\partial y}\right) - \frac{\partial p}{\partial y} + \rho_F f_y$	$S_{v_F}^P$	μ_{eff}
w_F	$\frac{\partial}{\partial x}\left(\Gamma_\Phi \frac{\partial u_F}{\partial z}\right) + \frac{\partial}{\partial y}\left(\Gamma_\Phi \frac{\partial v_F}{\partial z}\right) + \frac{\partial}{\partial z}\left(\Gamma_\Phi \frac{\partial w_F}{\partial z}\right) - \frac{\partial p}{\partial z} + \rho_F f_z$	$S_{w_F}^P$	μ_{eff}
k	$P_k - \rho_F\,\varepsilon$	0	$\mu + \frac{\mu_t}{\sigma_k}$
ε	$\frac{\varepsilon}{k}(c_{\varepsilon_1} P_k - c_{\varepsilon_2}\,\rho_F\,\varepsilon)$	0	$\frac{\mu_t}{\sigma_\varepsilon}$
$P_k = \mu_t \left\{ 2 \cdot \left[\left(\frac{\partial u_F}{\partial x}\right)^2 + \left(\frac{\partial v_F}{\partial y}\right)^2 + \left(\frac{\partial w_F}{\partial z}\right)^2 \right] + \left(\frac{\partial u_F}{\partial y} + \frac{\partial v_F}{\partial x}\right)^2 + \left(\frac{\partial u_F}{\partial z} + \frac{\partial w_F}{\partial x}\right)^2 + \left(\frac{\partial w_F}{\partial y} + \frac{\partial v_F}{\partial z}\right)^2 \right\}$			
$\mu_{eff} = \mu + \mu_t\ , \quad \mu_t = \rho_F c_\mu \frac{k^2}{\varepsilon}$; $c_\mu = 0.09\ , \quad c_{\varepsilon_1} = 1.44\ , \quad c_{\varepsilon_2} = 1.92\ , \quad \sigma_k = 1.0\ , \quad \sigma_\varepsilon = 1.3$			

Table 1: Source terms and diffusion coefficients for different variables Φ.

Fluid turbulence is modelled using the standard k–ε model and neglecting the influence of particle motion on fluid turbulence. Under these assumptions the time–averaged equations describing the motion of the fluid phase are given by the following form of the general transport equation:

$$\frac{\partial}{\partial x_j}\left(\rho_F\, u_{F\,j}\, \Phi\right) - \frac{\partial}{\partial x_j}\left(\Gamma\, \frac{\partial \Phi}{\partial x_j}\right) = +S_\Phi + S_\Phi^P \tag{1}$$

Here Φ is a general variable, Γ_Φ a diffusion coefficient, S_Φ a general source term and S_Φ^P symbolizes the source term due to momentum exchange between the fluid and the particle phase. The variables u_F, v_F and w_F represent the fluid velocity components, k is the turbulent kinetic energy and ε is the rate of dissipation of k. A detailed description of all terms and their correlations is shown in Table 1. In this table ρ_F is the fluid density and μ is the laminar viscosity.

2.2 Equations of motion of the disperse phase

The disperse phase is treated by the application of the Lagrangian (PSI–Cell) approach, i.e. discrete particle trajectories are calculated. Each calculated particle represents a large number of physical particles of the same physical properties. This is achieved by a particle number flow rate $\dot{N}_P$ prescribed to each calculated trajectory. The prediction of the particle trajectories is carried out by solving the ordinary differential equations for the particle location and velocities. Assuming that the ratio of fluid density to particle density is small ($\rho_F/\rho_P \ll 1$) these equations read :

$$\frac{d}{dt}\boldsymbol{x}_P = \boldsymbol{u}_P, \tag{2}$$

$$\begin{aligned} \frac{d}{dt}\boldsymbol{u}_P &= \frac{3}{4}\,\frac{\rho_F}{(\rho_P + \frac{1}{2}\rho_F)\, d_P}\left(u_{rel}\, C_D(Re_P)\, (\boldsymbol{u}_F - \boldsymbol{u}_P) \right. \\ &+ \frac{u_{rel}}{\omega_{rel}}\, C_M(\sigma)\, (\boldsymbol{u}_F - \boldsymbol{u}_P) \times (\boldsymbol{\omega} - \boldsymbol{\Omega}) \end{aligned}$$

$$+ \quad \frac{2\nu^{1/2}}{\pi|\boldsymbol{\Omega}|^{1/2}} C_A \left(\boldsymbol{u}_F - \boldsymbol{u}_P\right) \times \boldsymbol{\Omega} \Big) + \frac{\rho_P - \rho_F}{\rho_P + \frac{1}{2}\rho_F} \boldsymbol{g} \tag{3}$$

with: $\boldsymbol{\Omega} = \mathrm{rot}\, \boldsymbol{u}_F \,, \quad Re_P = \frac{d_P\, u_{rel}}{\nu} \,, \quad Re_\omega = \frac{1}{4} \frac{{d_P}^2 \omega_{rel}}{\nu} \,,$

$\sigma = \frac{1}{2} \frac{d_P\, \omega_{rel}}{u_{rel}} \,, \quad u_{rel} = |\boldsymbol{u}_F - \boldsymbol{u}_P| \,, \quad \omega_{rel} = |\boldsymbol{\omega} - \boldsymbol{\Omega}|$

where the rotation of the particle can be calculated from the following equation:

$$\frac{d}{dt}\boldsymbol{\omega} = -\frac{15}{16\pi} \frac{\rho_F}{\rho_P} \omega_{rel}\, \xi_m(Re_\omega) \left(\boldsymbol{\omega} - \boldsymbol{\Omega}\right) . \tag{4}$$

In these equations the subscript P indicates *Particle* and the subscript F indicates *Fluid.* ν is the fluid kinematic viscosity, d_P the paticle diameter and ω_{rel} the absolute value of the relative rotational velocity between fluid and particle. The terms on the right hand side of Eq. (3) represent the drag force exerted on the particle by the fluid, the lift force due to particle rotation (Magnus force), the lift force due to fluid velocity shear (Saffman force), the gravitational and added mass forces respectively. The values for the coefficients C_D, C_A, C_M and ξ_m can be found in [3, 2]. The effect of fluid turbulence on the motion of the disperse phase is modelled by the Lagrangian Stochastic–Deterministic (LSD) turbulence model. The particle's influence on the fluid phase is modelled by the PSI–Cell (Particle-Source-In-cell) method proposed by C.T. Crowe [2]. A more detailed description of all particular models involved in the Lagrangian particle trajectory calculation can be found in [5, 6, 2].

2.3 Solution Algorithm

For the numerical solution of the equations described in the above sections the physical space has to be discretized. Therefore a boundary–fitted, non–orthogonal numerical grid is used. The grid is blockstructured and consists of hexahedral cells. The equations of fluid motion (1) are numerically solved on the basis of a colocated, finite volume discretization. A pressure correction technique of SIMPLE kind (**S**emi–**I**mplicite **P**ressure **L**inked **E**quations) with convergence acceleration by a full multigrid method is applied (see [1]). When a converged solution for the fluid flow field has been calculated, the prediction of the particle motion is carried out. Therefore Eq.'s (3) and (4) are solved by using a standard 4th order Runge–Kutta scheme. In case of two-way-coupled multiphase flow systems the source terms S_Φ^P according to the PSI–Cell method are predicted simultaneously during trajectory calculation. After all particle trajectories are calculated the source terms are included in the fluid momentum equations and a new converged solution for the fluid flow field is computed. In the case of neglectable phase interaction (so called one-way-coupling) a single iteration step is sufficient to obtain the solution for the fluid and particle motion.

3 The Parallelization Methods

3.1 The parallel algorithm for fluid flow calculation

The parallelization of the solution algorithm for the set of continuity, Navier–Stokes and turbulence model equations is carried out by parallelization in space, that means by application of the domain decomposition or grid partitioning method. Using the block structure of the numerical grid the flow domain is partitioned in a number of subdomains (figure 1). Usually the number of grid blocks exceeds the number of processors, so that each processor of the PM has to handle a few blocks. If the number of grid blocks resulting from grid generation is too small for the designated PM or if this grid structure leads to larger imbalances in the PM due to large differences in the number of control volumes (CV's) per computing node a further preprocessing step enables the recursive division of largest grid blocks along the side of there

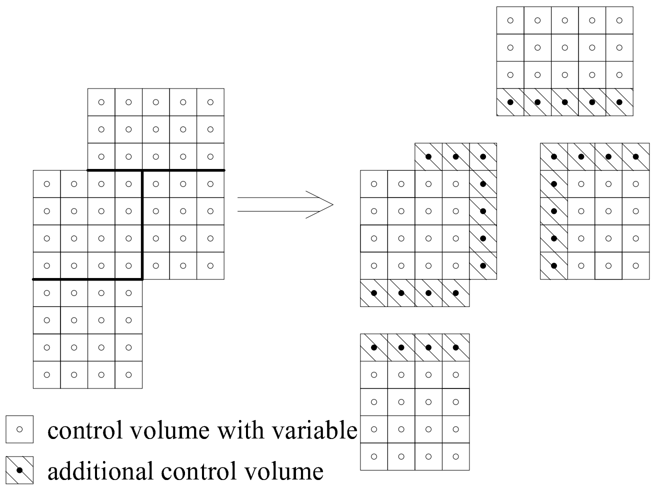


Fig. 1: Domain decomposition for the numerical grid.

largest expansion. The grid-block-to-processor assignment is given by a heuristicly determined block–processor allocation table and remains static and unchanged over the time of fluid flow calculation process.
Fluid flow calculation is then performed by individual processor nodes on the grid partitions stored in their local memory. Fluid flow characteristics along the grid block boundaries which are common to two different nodes have to be exchanged during the solution process by inter–processor communication, while the data exchange on common faces of two neighbouring grid partitions assigned to the same processor node can be handled locally in memory. More details of the parallelization method and results for its application to the Multi–grid accelerated SIMPLE algorithm for turbulent fluid flow calculation can be found in [1, 9].

3.2 Parallel algorithms for the Lagrangian approach

The prediction of the motion of the disperse phase is carried out by the application of the Lagrangian approach as described in Section 2.2. Considering the parallelization of this algorithm there are two important issues. The first is that in general particle trajectories are not uniformly distributed in the flow domain even if there is a uniform distribution at the inflow cross–section. Therefore the distribution of the numerical work load in space is not known at the beginning of the computation. As a second characteristic parallel solution algorithms for the particle equations of motion have to deal with the global data dependence between the distributed storage of fluid flow data and the local data requirements for particle trajectory calculation. A parallel Lagrangian solution algorithm has either to provide all fluid flow data necessary for the calculation of a certain particle trajectory segment in the local memory of the processor node or the fluid flow data have to be delivered from other processor nodes at the moment when they are required. Considering these issues the following parallelization methods have been developed :

Method 1: Static Domain Decomposition (SDD) method
The first approach in parallelization of Lagrangian particle trajectory calculations is the application of the same parallelization scheme as for the fluid flow calculation to the Lagrangian solver as well. That means a Static Domain Decomposition (SDD) method. In this approach geometry and fluid flow data are distributed over the processor nodes of the PM in accordance with the block–processor allocation table as already used in the fluid flow field calculation of the Navier–Stokes solver.
Furthermore an explicit host–node process scheme is established as illustrated in Figure 2. The trajectory calculation is done by the node processes whereas the host process carries out only management tasks. The node processes are identical to those that do the flow field calculation. Now the basic principle of the SDD method is that in a node process only those trajectory

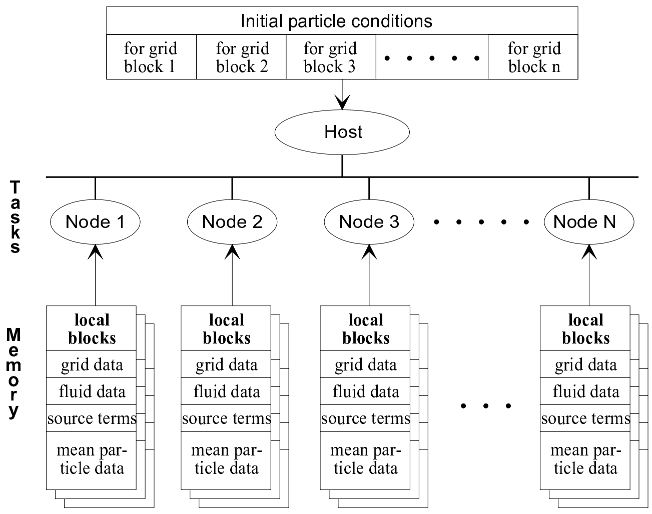


Fig. 2: Static Domain Decomposition (SDD) method for the Lagrangian solver.

segments are calculated that cross the grid partition(s) assigned to this process. The particle state (location, velocity, diameter, ...) at the entry point to the current grid partition is sent by the host to the node process. The entry point can either be at an inflow cross section or at a common face/boundary to a neighbouring partition. After the computation of the trajectory segment on the current grid partition is finished, the particle state at the exit point (outlet cross section or partition boundary) is sent back to the host. If the exit point is located at the interface of two grid partitions, the host sends the particle state to the process related to the neighbouring grid partition for continuing trajectory computation. This redistribution of particle state conditions is repeatedly carried out by the host until all particle trajectories have satisfied certain break condition (e.g. an outlet cross section is reached). During the particle trajectory calculation process the source terms for momentum exchange between the two phases are calculated locally on the processor nodes $1, \ldots, N$ from where they can be passed to the Navier–Stokes solver without further processing.
An advantage of the domain decomposition approach is that it is easy to implement and uses the same data distribution over the processor nodes as the Navier–Stokes solver. But the resulting load balancing can be a serious disadvantage of this method as shown later for the presented test cases. Poor load balancing can be caused by different circumstances, as there are :

1. Unequal processing power of the calculating nodes (e.g. heterogenous workstation cluster).
2. Unequal size of the grid blocks of the numerical grid. This results in a different number of CV's per processor node and in unequal work load for the processors.
3. Differences in particle concentration distribution throughout the flow domain. Situations of poor load balancing can occur e.g. for flows around free jets/nozzles, in recirculating or highly separated flows where most of the numerical effort has to be performed by a small subset of all processor nodes used.
4. Multiple particle–wall collisions. Highly frequent particle–wall collisions occur especially on curved walls where the particles are brought in contact with the wall by the fluid flow multiple times. This results in a higher work load for the corresponding processor node due to the reduction of the integration time step and the extra effort for detection/calculation of the particle–wall collision itself.
5. Flow regions of high fluid velocity gradients/small fluid turbulence time scale. This leads to a reduction of the integration time step for the Lagrangian approach in order to preserve accuracy of the calculation and therefore to a higher work load for the corresponding processor node.

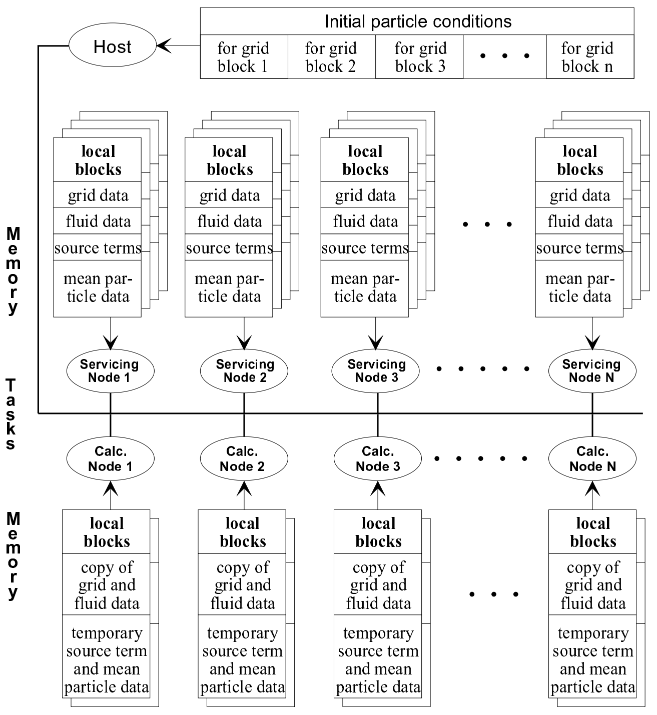


Fig. 3: Dynamic Domain Decomposition (DDD) method for the Lagrangian solver.

The reasons 1–2 for poor load balancing are common to all domain decomposition approaches and apply to the parallelization method for the Navier–Stokes solver as well. But most of the factors 3–5 leading to poor load balancing in the SDD method cannot be foreseen without prior knowledge about the flow regime inside the flow domain (e.g. from experimental investigations). Therefore an adjustment of the numerical grid or the block-processor assignment table to meet the load balancing requirements by redistribution of grid cells or grid partitions inside the PM is almost impossible. The second parallelization method shows how to overcome these limitations by introducing a load balancing algorithm which is effective during run time.

Method 2: Dynamic Domain Decomposition (DDD) method

This method has been developed to overcome the disadvantages of the SDD method concerning the balancing of the computational work load. In the DDD method there exist three classes of processes : the host, the servicing nodes and the calculating nodes (Figure 3). Just as in the SDD method the host process distributes the particle initial conditions among the calculating nodes and collects the particle's state when the trajectory segment calculation has been finished. The new class of servicing nodes use the already known block-processor assignment table from the Navier–Stokes solver for storage of grid and fluid flow data. But in contrast to the SDD method they do not performe trajectory calculations but delegate that task to the class of calculating nodes. So the work of the servicing nodes is restricted to the management of the geometry, fluid flow and particle flow data in the data structure prescribed by the block-processor assignment table. On request a servicing node is able to retrieve or store data from/to the grid partition data structure stored in its local memory.

The calculating nodes are performing the real work on particle trajectory calculation. These nodes receive the particle initial conditions from the host and predict particle motion on an arbitrary grid partition. In contrast to the SDD method there is no fixed block-processor assignment table for the calculating nodes. Starting with an empty memory structure the calculating nodes are able to obtain dynamically geometry and fluid flow data for an arbitrary grid partition from the corresponding servicing node managing this part of the numerical grid.

The correlation between the required data and the corresponding servicing node can be looked up from the block-processor assignment table. Once geometry and fluid flow data for a certain grid partition has been retrieved by the calculating node, this information is locally stored in a pipeline with a history of a certain depth. But since the amount of memory available to the calculating nodes can be rather limited, the amount of locally stored grid partition data can be limited by an adjustable parameter. So the concept of the DDD method makes it possible 1. to perform calculation of a certain trajectory segment on an arbitrary calculating node process and 2. to compute different trajectories on one grid partition at the same time by different calculating node processes.
It has further to be mentioned that a servicing node process does not have to be executed on a separate physical processor, since the work load is quite neglectable. In current MPI implementations the servicing node process is implemented as separate node process and is executed in parallel to the corresponding calculating node process on the same physical processor. Furthermore the host process is also executed on one of the N processors of the PM keeping the number of used processors constant in comparison with the Navier–Stokes solver. But results show that efficiency of calculation can be effected with some MPI distributions, e.g. such as MPICH 1.2.0. Another possible implementation is the execution of a calculating node process as a thread of the corresponding servicing node process. But this requires fragile mixed message passing and thread programming and leads to a not as portable solution as for the pure message passing implementation strictly based on MPI standards.

4 Numerical Experiments

4.1 MIMD computer architectures and MPI implementations

The different parallelization methods are based on the paradigm of a MIMD computer architecture with explicit message passing between the node processes of the PM. The implementation uses an encapsulated communication layer which can operate on top of standard MPI or PVM communication libraries (the latter from historical reasons). Usable communication libraries have to be in compliance with MPI 1.1 or PVM 3.2 standard.
Investigations presented in this paper were carried out on three different computer architectures. Most of the calculations have been performed on an AMD/Athlon PC cluster (12 AMD/Athlon, 600 MHz, 512 Mb RAM per node, FastEthernet) or on the **C**hemnitz **Li**nux **C**luster **CLIC** (528 Intel/Pentium III, 800 MHz, 512 Mb RAM per node, 2 x FastEthernet) [11, 10]. For performance comparison we used the CRAY–T3E at the Dresden University of Technology. This Cray T3E system consists of 64 DEC Alpha 21164 processors with 128 MBytes memory on each processor node. Nodes are arranged in a 3–dimensional torus (GigaRing), with each of the six links from each node simultaneously supporting hardware transfer rates of up to 300 MBytes/s. The calculations on the PC clusters were performed with MPI distributions of MPICH 1.2.0 and LAM–MPI 6.3.2. The CRAY was running under Unicos/mk 2.0.4 with the Message Passing Toolkit MPT 1.2.1.0.

4.2 Description of the test cases

Two test cases are investigated. The first test case is a dilute gas–particle flow in a three times bended channel with square cross section of $0.2 \times 0.2\, m^2$. In all three channel bends 4 corner vanes are installed, dividing the cross section of the bend in 5 separate corner sections (see Figure 4). The vanes are modelled as infinitely thin solid walls within the flow region (non slip condition). The duct has been subdivided into 64 blocks, the number of finite volumes for the finest grid is $80{*}80{*}496 = 3\,174\,400$; because of the blades no more than three coarser grids can be used for the multigrid method. This means that the coarsest grid with only two cells between the blades has $10{*}10{*}62 = 6\,200$ finite control volumes. At the inlet an plug velocity profile with 10.0 m/s is given ($Re_F = 156\,000$), at the outlet a zero gradient condition is implemented. The particle

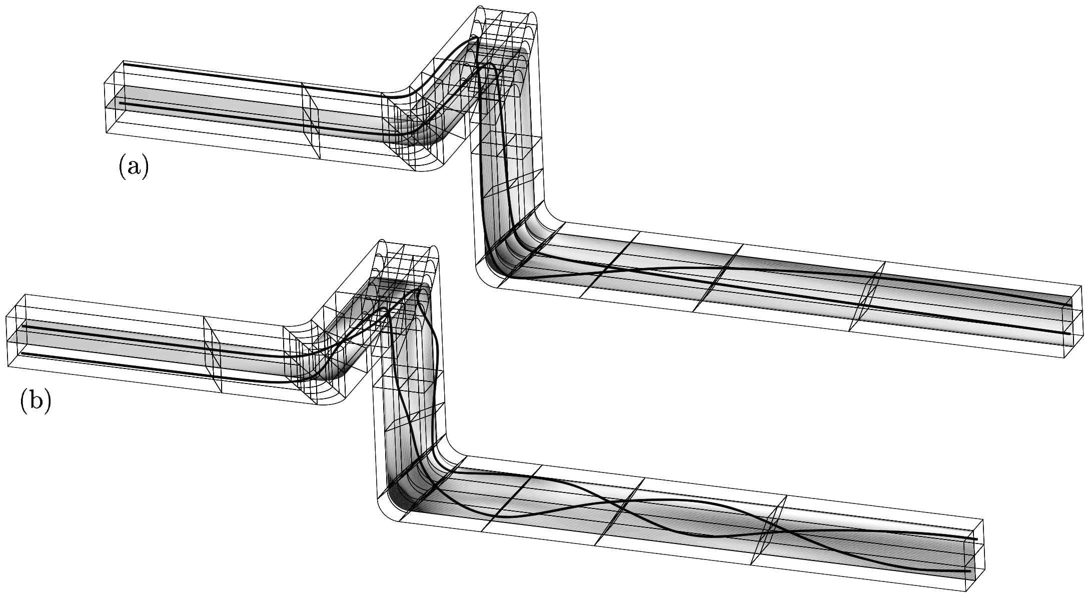


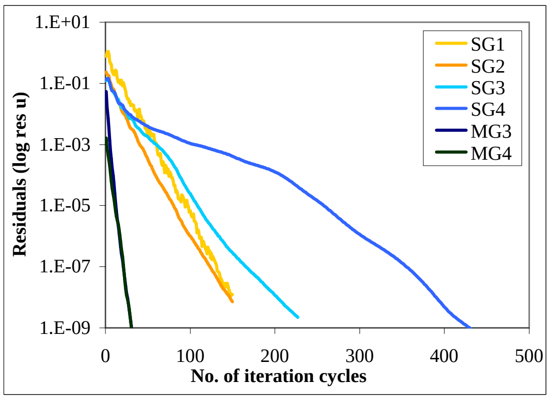
Fig. 4: Flow through a bended duct: grid blocks, absolute gas velocity and two particle trajectories; (a) bended duct with corner vanes, (b) bended duct without vanes.

phase with particle diameters of $d_P = 20, \ldots, 60\ \mu/m$ and a density of $\rho_P = 2500\ kg/m^3$ has initially a uniform concentration distribution over the inlet cross section of the duct. For each of the test case calculations 5000 particle trajectories have been calculated. Similar configurations are used for e.g. pneumatical conveying of granular material in channels and pipes.
The second test case differs from the first one by omitting the vanes in the channel bends. This leads to the development of a counter clockwise swirling fluid flow and due to the centrifugal forces acting on the particles to a strong separation of the particle phase from the fluid flow. Demixing of the particle phase starts immediately after the first bend and leads to the formation of a particle rope after the third channel bend. This test case has been introduced as an example of a strongly separated gas–particle flow in order to proof the suitability and performance of the developed load balancing algorithms for such kind of separated multiphase flows leading to poor parallel efficiency with the so far used SDD method.

4.3 Results and discussion

4.3.1 Fluid flow calculation

The first test compares the convergence of two algorithms which use the multigrid (MG) method in different way for the calculation of the fluid flow field. The first algorithm, denoted as single–grid method (SG), is the common SIMPLE method where only the pressure calculation is accelerated with the MG method. Although this inner MG method leads to a much faster convergence of the iteration for the complete system the dependence of the number of iterations needed for a prescribed accuracy on the number of unknowns cannot be overcome in this way. The second algorithm, denoted as multigrid method, applies the MG–method to the complete system of equations and uses the SIMPLE–algorithm as smoothing method and coarse grid solver. The MG–acceleration for the pressure calculation is used additionally.
In the test the flow through the bended duct with blades is calculated on a sequence of refined grids. Figure 5 includes single–grid runs on four grids (SG1 – SG4) with 10∗10∗62 (coarsest grid) up to 80∗80∗496 finite volumes and multigrid runs starting on the two finest grids (MG3, MG4). N denotes the number of SG iterations or MG cycles. The curves show that the number of iterations for the SG method increases while the number of MG cycles remains constant if the grid is refined. The total calculation times are decreased by the multigrid technique up to 1%



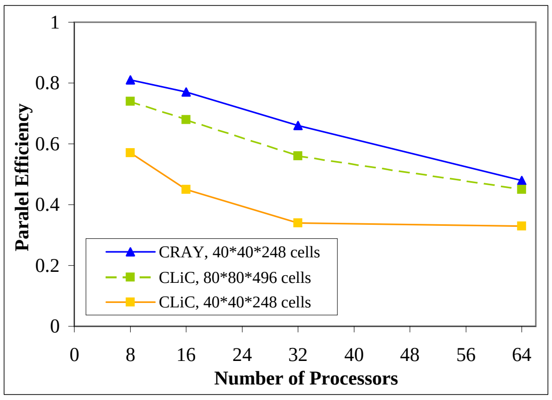


Fig. 5: Convergence of the single– and multi–grid method on a sequence of refined grids.

Fig. 6: Parallel Efficiency of the MG algorithm on the CLIC vs. Cray T3E.

of the original SIMPLE method. Figure 6 summarizes the parallel efficiency of the MG method on the CRAY–T3E and on the CLIC for a series of runs on an increasing number of processors. All calculations are performed for the bended duct with 64 blocks and 396 800 or 3 174 400 finite volumes on the finest grid. Generally the parallel efficiency is better for the calculations on the CRAY. This is due to the faster communication network. Table 2 shows calculation times T_cal and the times needed for data exchange T_e for the runs on the coarser grid. While with a growing number of nodes the ratio T_e/T_cal remains constant on the CRAY it grows from 0.42 to 0.8 on the CLIC. The run on the fine grid demonstrates that on the CLIC the parallel efficiency is fairly good as long as the number of finite volumes per node is not too small.
More details of the parallelization method and further results for fluid flow calculation can be found in [1].

Number of nodes	8	16	32	64
CRAY: T_cal	1661	863	499	349
T_e	396	199	103	80
T_e/T_cal	**0.24**	**0.23**	**0.21**	**0.23**
CLIC: T_cal	1022	669	449	249
T_e	434	400	330	200
T_e/T_cal	**0.42**	**0.60**	**0.73**	**0.80**

Table 2: Calculation and data exchange times for an increasing number of processor nodes.

4.3.2 Lagrangian particle trajectory calculation

For the test case calculations the total execution time, calculation time, communication time and I/O time have been measured for the execution of one iteration cycle of the Lagrangian solver (calculation of 5000 particle trajectories, one-way-coupling). From these measurements the difference time ($T_{diff} = T_{total} - T_{calc} - T_{comm} - T_{I/O}$) has been calculated. This difference time contains mainly the waiting time for the processor nodes in receive operations and global barriers, what can also be established from figure 9.
First of all the sensitivity of MPICH and LAM–MPI in respect to multi–processing/multi–tasking has been investigated. As has been pointed out earlier it is not necessary from the point of view of the amount of numerical work to spend an extra physical processor node for management tasks like the host process or the servicing node processes in DDD methods. But in our numerical experiments we could find different behavior of existing MPI libraries/distributions in the case of superposition of MPI processes on the same physical processor (figure 7). In experiments carried out with LAM–MPI we could not observe significant differences in measured execution

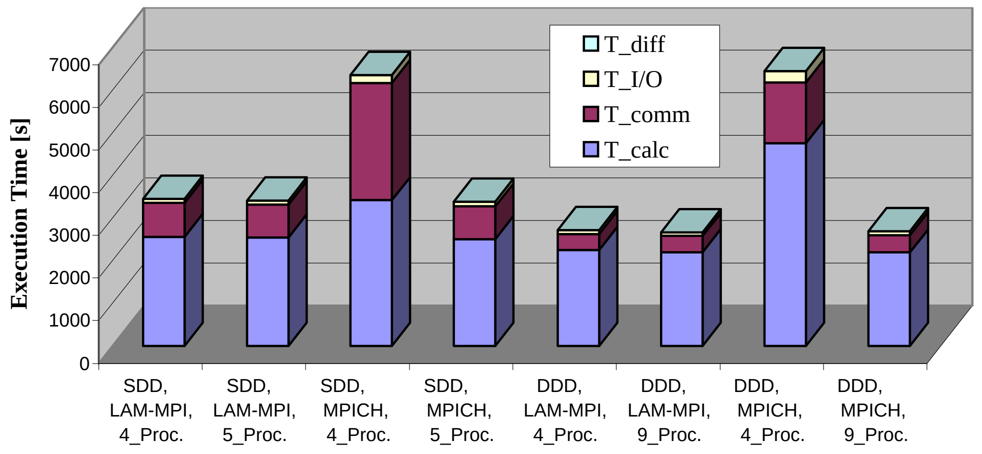


Fig. 7: Comparison of LAM–MPI and MPICH libraries in case of superposition of MPI processes on the same physical processor node of the PM.

or communication times when we run the SDD and DDD methods with superposition of the host and the N servicing node processes together with the N calculating node processes on N physical processors in comparison with an execution of the algorithm using a single process per processor node. But Fig. 7 shows a great sensitivity of MPICH in respect to such a superposition of processes. Comparable results with MPICH could only be achieved in the regime of a single MPI process per physical processor node of the PM. Therefor all further numerical experiments on clusters of workstations have been performed with LAM–MPI.

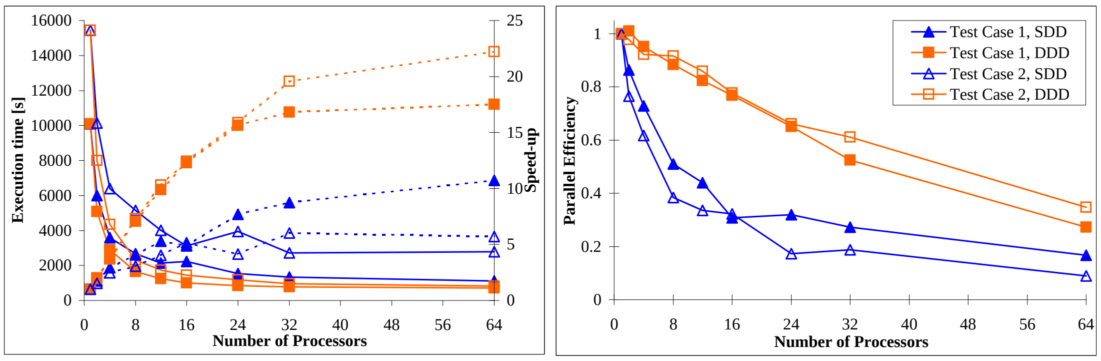


Fig. 8: Total execution times, speed–up and parallel efficiency vs. number of processor nodes (CLIC); comparison of parallelization methods for both test cases.

Figure 8 show the total execution times, the speed–up and the parallel efficiency for calculations on both test cases with SDD and DDD methods vs. the number of processor nodes. All test case calculations in this experiments had been carried out on the second finest grid level with 396.800 CV's. Figure 8 show the remarkable reduction in computation time with both parallelization methods. It can be seen from the figures that in all cases the Dynamic Domain Decomposition (DDD) method has a clear advantage over the SDD method. But this advantage for the DDD method in the first test case is not as remarkable as for the second test case. This is due to the fact, that the gas–particle flow in the first test case is quiet homogeneous in respect to particle concentration distribution which leads to a more balanced work load distribution in the SDD method. So the possible gain in performance with the DDD method is not as large as for the second test case, where the gas–particle flow is strongly separated and where we can observe particle roping and sliding of particles along the solid walls of the channel leading to a much higher amount of numerical work in certain regions of the flow. Consequently the SDD method shows a very poor parallel efficiency for the second test case due to poor load balancing between the processors of the PM (figure 8).

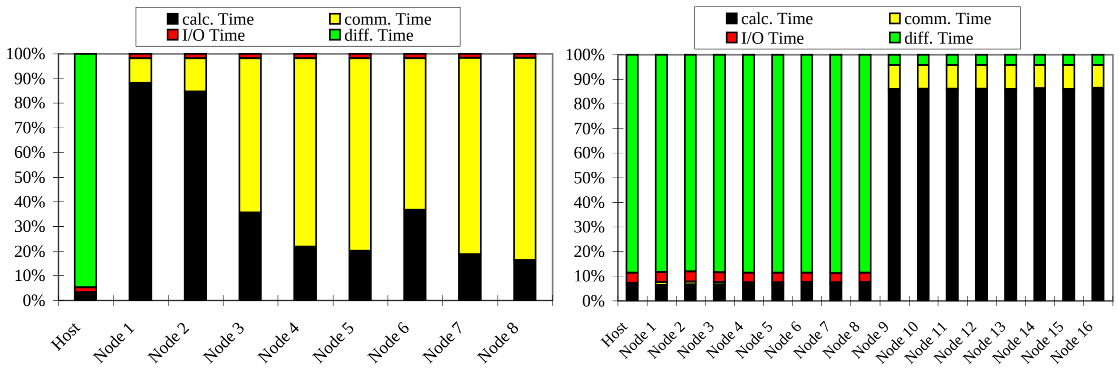


Fig. 9: Work load distribution for SDD and DDD methods on test case 2 for execution on 8 processor nodes (9 and 17 processes respectively).

The efficiency of the work load balancing introduced to the Lagrangian approach by the Dynamic Domain Decomposition (DDD) method can also clearly be seen from the work load distribution diagramms in figure 9. The diagrams show the ratio of calculation, communication and I/O times for the $(N + 1)$ and $(2N + 1)$ processes involved in a calculation for the second test case on $N = 8$ processors using the SDD and DDD method. In the SDD method the largest amount of calculation time is spent only by processors 1 and 2 while the other processors show a large amount of communication (waiting) time. From figure 9 it can be seen for the DDD method that the distribution of work load over the calculating node processes 9–16 is very uniform. Furthermore the management tasks (host process and servicing node processes) show a very low amount of calculation time of less then 10% of the total execution time. The amount of difference time for the calculating nodes arrises from the necessary waiting time of these processes during file I/O at start-up and end of the calculation process performed by the node processes 1–8.

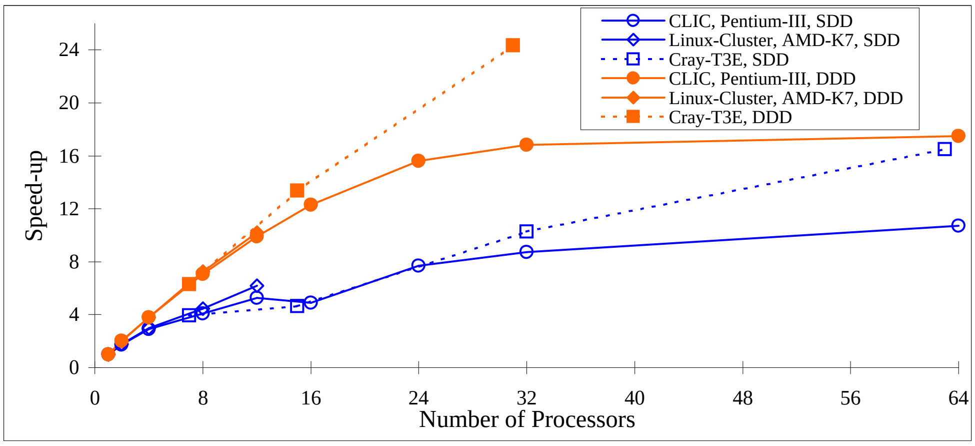


Fig. 10: Comparison of parallel performace on Chemnitz Linux Cluster(s) vs. Cray T3E

Figure 10 shows the comparison of test case calculations between the CLIC, an AMD–Athlon based workstation cluster and the Cray T3E for the test case 1. Again we can observe the clear advantage of the DDD method with the implemented dynamic load balancing scheme leading to an acceleration of the Lagrangian particle trajectory calculation by a factor of up to 3 on the Cray T3E in comparison with the SDD method. The impact of the Cray high-bandwith-low-latency interconnection network can clearly be seen from the figure. So the speed–up for the test case calculations on the Cray increases almost linearly with increasing number of processors up to 32 nodes. On the CLIC we observe minor speed–up values and reach saturation (effect of higher latency, bandwith limits of the FastEthernet communication network) for calculations on more than 32 processor nodes where a further substantial decrease of the total execution time for the Lagrangian solver could not be achieved. But similiar scaling behavior of the presented Eulerian–Lagrangian approach as it was observed for the Cray T3E should be within reach on workstation clusters with highspeed interconnects like e.g. Myrinet, GigaNet, SCI or other new emerging network technologies.

5 Conclusions

The paper presents two parallelization methods for the Eulerian–Lagrangian approach for disperse multiphase flow calculations together with their MPI implementations. Performance results are given for two typical test cases. The obtained results show the importance of homogeneous work load distribution, which has to be treated differently in comparison with Static Domain Decomposition (SDD) methods for common single phase flow computations. With the presented Dynamic Domain Decomposition (DDD) method remarkable speed–up can be achieved for the Eulerian–Lagrangian computation of disperse multiphase flows on MIMD computers and clusters of workstations. The main advantages of the DDD method are :

- dynamic work load distribution among the processors of the parallel machine; work load balancing is effective even on heterogeneous computer systems like eg. workstation clusters with different computational power of processor nodes;
- independence of performance from the subdivision of the numerical grid into grid blocks,
- independence of the algorithm performance from the flow regime (e.g. phase separation, non–homogeneous concentration distribution of the dispersed phase, local occurence of strong particle–wall interaction).

So the developed parallelization method with dynamic work load balancing offers new perspectives for the computation of strongly coupled multiphase flows with complex phase interactions and higher particle concentrations.

Acknoledgements

This work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG) in the framework of the Collaborative Research Centre SFB–393 under Contract No. SFB 393/D2.

References

[1] **Bernert K., Frank Th. :** "Multi–Grid Acceleration of a SIMPLE-Based CFD-Code and Aspects of Parallelization", IEEE International Conference on Cluster Computing — CLUSTER 2000, November 28.–December 2., 2000, Chemnitz, Germany.

[2] **Crowe C.T., Sommerfeld M., Tsuji Y. :** "Multiphase Flows with Droplets and Particles", CRC Press, 1998.

[3] **Frank Th. :** "Numerische Simulation der feststoffbeladenen Gasströmung im horizontalen Kanal unter Berücksichtigung von Wandrauhigkeiten", PhD Thesis, Techn. University Bergakademie Freiberg, Germany, (1992).

[4] **Frank, Th., Wassen, E.:** "Parallel Efficiency of PVM– and MPI–Implementations of two Algorithms for the Lagrangian Prediction of Disperse Multiphase Flows", JSME Centennial Grand Congress 1997, ISAC '97 Conference on Advanced Computing on Multiphase Flow, Tokyo, Japan, July 18–19, 1997

[5] **Frank, Th., Schneider, J., Yu, Q. Wassen, E.:** "Experimental and Numerical Investigation of Particle Separation in a Symmetrical Double Cyclone Separator", 8th Int. Symposium on Gas–Particle Flows, ASME Fluids Engineering Division Summer Meeting, San Francisco, CA, U.S.A., July 18–22, 1999, CD–ROM Proceedings, Paper No. FEDSM99–7865, pp. 1–10

[6] **Frank Th. :** "Application of Eulerian–Lagrangian Prediction of Gas–Particle Flows to Cyclone Separators", VKI, Von Karman Institute for Fluid Dynamics, Lecture Series Programme 1999–2000, "Theoretical and Experimental Modeling of Particulate Flow", Bruessels, Belgium, 03.–07. April 2000.

[7] **Perić M.:** "A Finite Volume Method for the Prediction of 3–dimensional Flows in Complex Ducts", Ph. D. Thesis, Imperial College, University of London, 1985.

[8] **Perić M.:** "Ein zum Parallelrechnen geeignetes Finite–Volumen–Mehrgitterverfahren zur Berechnung komplexer Strömungen auf blockstrukturierten Gittern mit lokaler Verfeinerung", Abschlußbericht zum DFG–Vorhaben Pe 350/3–1 im DFG–Habilitandenstipendiumprogramm, Stanford University, USA, 1992.

[9] **Schreck E., Perić M.:** "Parallelization of implicit solution methods", ASME Fluids Engineering Conference, June 22–23, 1992, Los Angeles (CA), USA.

[10] **TOP500** — List of Top 500 Supercomputer Sites in the World : http://www.top500.org

[11] **Web site** of the **C**hemnitz **Li**nux **C**luster (**CLIC**), Chemnitz University of Technology, Germany http://www.tu–chemnitz.de/urz/anwendungen/CLIC

[12] **Web site** of the Research Group on Multiphase Flow, Chemnitz University of Technology, Germany http://www.imech.tu-chemnitz.de/index.html – Index, List of Publications.