**IEEE International Conference on Cluster Computing,
CLUSTER 2000, November 28.–December 2., 2000,
Chemnitz University of Technology, Saxony, Germany.**

Efficient Parallelization of Eulerian–Lagrangian Approach for Disperse Multiphase Flow Calculations on MIMD Computer Architectures

Th. Frank, K. Bernert, K. Pachler, H. Schneider

Chemnitz University of Technology
Faculty of Mechanical Engineering and Process Technology
Research Group of Multiphase Flow
Reichenhainer Straße 70, 09107 Chemnitz, Germany
Email: frank@imech.tu-chemnitz.de
Tel.: +49 (371) 531 46 43, Fax: +49 (371) 531 46 44

Abstract

This paper deals with the presentation and comparison of two different parallelization methods for the Lagrangian (PSI–Cell) approach, a frequently used method for the numerical prediction of disperse multiphase flows, e.g. dilute gas–particle and gas–droplet flows. Both presented algorithms are based on the Domain Decomposition method applied to a blockstructured numerical grid, as it is widely used for the parallel computation of fluid flows. The first algorithm applies the same static assignment of grid partitions to the processors of the parallel machine (PM) also to the Lagrangian particle trajectory calculation (SDD – Static Domain Decomposition). All particle trajectories that cross a certain grid partition are calculated by the processor this partition is staticly assigned to. As our results show, the parallel efficiency of such a parallelization method can be dramatically deteriorated by e.g. non–homogeneous particle concentration distribution in the flow.

In the second parallelization method — the so–called Dynamic Domain Decomposition (DDD) — a dynamic assignment of grid and fluid flow information to PM processor nodes is used. A better load balancing between the processors of the PM can be established, leading to considerably increased parallel efficiency and a higher degree of flexibility in the application of the computational method to different flow conditions.

Results of performance evaluations are provided for two different typical test cases and for MIMD computer architectures of a AMD/Athlon based Linux workstation cluster and the Cray T3E as well.

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. Difficulties 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. 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:

$$\frac{\partial}{\partial x}(\rho_F\, u_F\, \Phi)+\frac{\partial}{\partial y}(\rho_F\, v_F \Phi)+\frac{\partial}{\partial z}(\rho_F\, w_F \Phi) = \frac{\partial}{\partial x}\left(\Gamma_\Phi\, \frac{\partial \Phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(\Gamma_\Phi\, \frac{\partial \Phi}{\partial y}\right)+\frac{\partial}{\partial z}\left(\Gamma_\Phi\, \frac{\partial \Phi}{\partial z}\right)+S_\Phi+S_\Phi^P \quad (1)$$

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.

Φ	S_Φ	S_Φ^P	Γ_Φ
1	0	0	0
u_F	$\frac{\partial}{\partial x}\left(\Gamma_\Phi \frac{\partial u_F}{\partial x}\right)+\frac{\partial}{\partial y}\left(\Gamma_\Phi \frac{\partial v_F}{\partial x}\right)+\frac{\partial}{\partial z}\left(\Gamma_\Phi \frac{\partial w_F}{\partial x}\right)-\frac{\partial p}{\partial x}+\rho_F f_x$	$S^P_{u_F}$	μ_{eff}
v_F	$\frac{\partial}{\partial x}\left(\Gamma_\Phi \frac{\partial u_F}{\partial y}\right)+\frac{\partial}{\partial y}\left(\Gamma_\Phi \frac{\partial v_F}{\partial y}\right)+\frac{\partial}{\partial z}\left(\Gamma_\Phi \frac{\partial w_F}{\partial y}\right)-\frac{\partial p}{\partial y}+\rho_F f_y$	$S^P_{v_F}$	μ_{eff}
w_F	$\frac{\partial}{\partial x}\left(\Gamma_\Phi \frac{\partial u_F}{\partial z}\right)+\frac{\partial}{\partial y}\left(\Gamma_\Phi \frac{\partial v_F}{\partial z}\right)+\frac{\partial}{\partial z}\left(\Gamma_\Phi \frac{\partial w_F}{\partial z}\right)-\frac{\partial p}{\partial z}+\rho_F f_z$	$S^P_{w_F}$	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

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 :

$$\frac{d}{dt}\begin{bmatrix} x_P \\ y_P \\ z_P \end{bmatrix} = \begin{bmatrix} u_P \\ v_P \\ w_P \end{bmatrix}\,, \quad (2)$$

$$\frac{d}{dt}\begin{bmatrix} u_P \\ v_P \\ w_P \end{bmatrix} = \frac{3}{4}\frac{\rho_F}{(\rho_P+\frac{1}{2}\rho_F)\,d_P}\left(v_{rel}C_D(Re_P)\begin{bmatrix} u_F-u_P \\ v_F-v_P \\ w_F-w_P \end{bmatrix} \right.$$

$$+ \frac{v_{rel}}{\omega_{rel}}C_M(\sigma)\begin{bmatrix} (v_F-v_P)(\omega_z-\Omega_z)-(w_F-w_P)(\omega_y-\Omega_y) \\ (w_F-w_P)(\omega_x-\Omega_x)-(u_F-u_P)(\omega_z-\Omega_z) \\ (u_F-u_P)(\omega_y-\Omega_y)-(v_F-v_P)(\omega_x-\Omega_x) \end{bmatrix}$$

$$\left. + \frac{2\nu_F^{1/2}}{\pi|\vec{\Omega}|^{1/2}}C_A\begin{bmatrix} (v_F-v_P)\Omega_z-(w_F-w_P)\Omega_y \\ (w_F-w_P)\Omega_x-(u_F-u_P)\Omega_z \\ (u_F-u_P)\Omega_y-(v_F-v_P)\Omega_x \end{bmatrix} \right) + \frac{\rho_P-\rho_F}{\rho_P+\frac{1}{2}\rho_F}\begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} \tag{3}$$

with: $\vec{\Omega} = \text{rot}\,\vec{v}_F\,, \quad Re_P = \dfrac{d_P\,v_{rel}}{\nu_F}\,, \quad Re_\omega = \dfrac{1}{4}\dfrac{{d_P}^2\omega_{rel}}{\nu_F}\,, \quad \sigma = \dfrac{1}{2}\dfrac{d_P\,\omega_{rel}}{v_{rel}}\,,$

$$v_{rel} = \sqrt{(u_F-u_P)^2+(v_F-v_P)^2+(w_F-w_P)^2}\,, \quad \omega_{rel} = \sqrt{(\omega_x-\Omega_x)^2+(\omega_y-\Omega_y)^2+(\omega_z-\Omega_z)^2}\,.$$

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

$$\frac{d}{dt}\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = -\frac{15}{16\pi}\frac{\rho_F}{\rho_P}\,\omega_{rel}\,\xi_m(Re_\omega)\begin{bmatrix} \omega_x-\Omega_x \\ \omega_y-\Omega_y \\ \omega_z-\Omega_z \end{bmatrix}\,. \tag{4}$$

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, 8]. 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, 7, 8, 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.

The iterative algorithm for the numerical simulation of the coupled two–phase flow is summarized as follows:

1. calculation of a converged solution for the fluid flow field without taking the source terms of the disperse phase S_Φ^P into account
2. tracing a large number of particles through the flow field and computing the source terms S_Φ^P simultaneously
3. recalculation of the fluid flow field considering the source terms S_Φ^P of the disperse phase
4. repeating Steps 2 and 3 until the solution of the coupled equations has converged.

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 (Fig. 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 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.

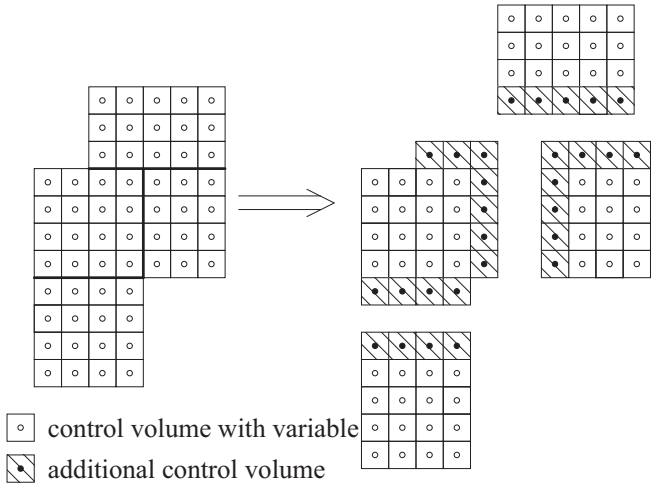

Fig. 1: Domain decomposition for the numerical grid.

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].

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.

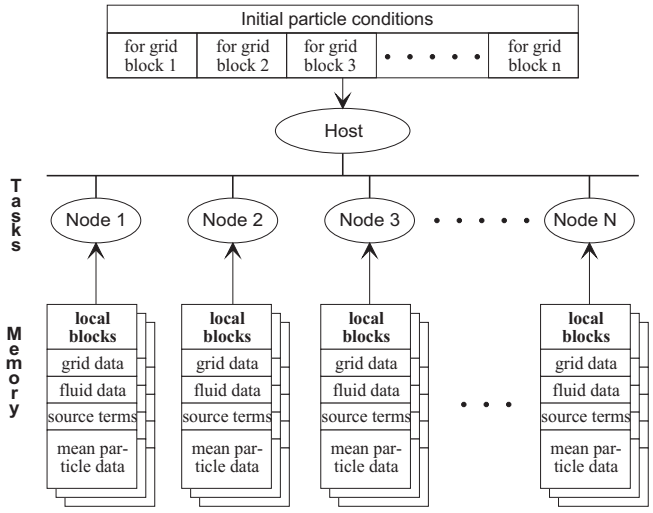

Fig. 2: Static Domain Decomposition (SDD) method for the Lagrangian solver.

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 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. in a 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.

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.

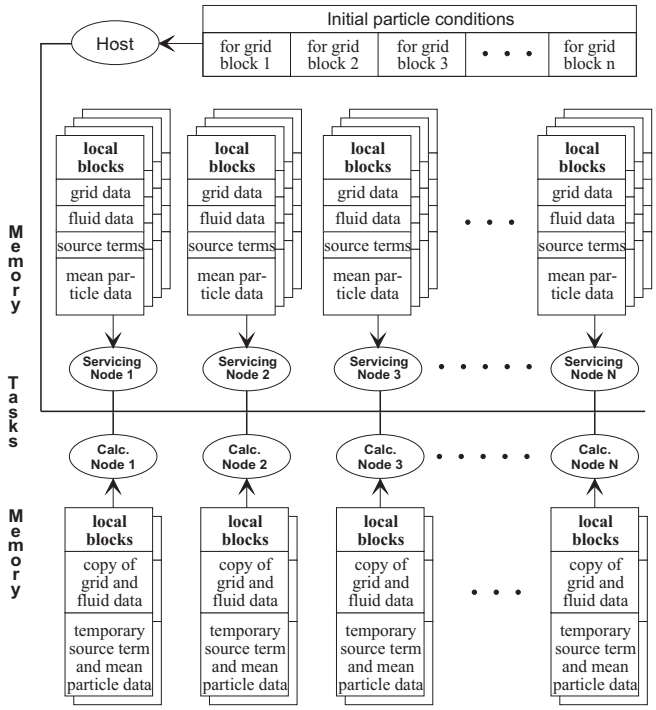

Fig. 3: Dynamic Domain Decomposition (DDD) method for the Lagrangian solver.

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.

There are some further details of the DDD method that should be pointed out. For predicting a trajectory segment a calculating node needs the complete geometry data of the actual grid partition but only the fluid data for the grid cell in which the particle is currently located. Hence there are two thinkable algorithms concerning the treatment of the fluid data :

1. If a particle trajectory crosses a certain control volume/grid cell, the fluid flow data corresponding to this grid cell and to the nearest neighbouring grid cells in each coordinate direction are required for further calculation. Now the fluid flow data for these 7 control volumes can be delivered at a time (further referred to as DDD–Point method). Even this method introduces more frequent communications with lower volume of transfered data it can be advantageous in dependence on the computer hardware and the size of the grid partitions of the numerical mesh because only a 1–dimensional subset of the fluid flow data has to be transfered for the calculation of the particle trajectory.
2. Otherwise the full information about the geometry and fluid flow data corresponding to the grid partition of the assigned particle initial condition can be transfered on the first request from the calculating node (further referred to as DDD–Block method). This algorithm implies a single communication while transferring a large amount of data. This method is more advantageous for parallel computer architectures with fast communication network and high bandwith of communication.

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 Results and Discussion

4.1 MPI Implementations and MIMD Computer Architectures

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 be 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 have been carried out on two different computer architectures. Most of the test case calculations have been performed on a AMD/Athlon 600 MHz based cluster of workstations (COW) running under RedHat Linux 6.0, Kernel Vers. 2.2.14. The COW has up to 12 processors with 512 MBytes RAM on each node. The cluster nodes are interconnected by 100 Mbit/s FastEthernet over a Cisco Catalyst Switch C3524–XL. For parallel computing we used MPI distributions of MPICH 1.2.0 and LAM–MPI 6.3.2.

For performance comparison we used the Cray T3E system at the University of Technology, Dresden. 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, with each of the six links from each node simultaneously supporting hardware transfer rates of up to 300 MBytes/s. The system is running under Unicos/mk 2.0.4 with the Message Passing Toolkit MPT 1.2.1.0 (containing MPI and PVM message passing libraries).

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 Fig. 4). The vanes are modeled 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 Multi–grid method. This means that the coarsest grid with only two cells between the blades has $10*10*62 = 6\,200$ finite control volumes. Due to memory limitations most of the investigations has been carried out on the second finest grid level with 396.800 CV's. At the inlet an uniform velocity profile with $u_F = u_P = 10.0\,m/s$ is given ($Re = 156\,000$), at the outlet a zero gradient condition is implemented. The particle 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. Similiar configurations are used for e.g. pneumatical conveying of granular material in channels and pipes.

The second test case differs from the first test case configuration by omiting the vanes in the channel bends. The omission of the corner vanes together with the 3 sucessive channel bends 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 imediately 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

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 Fig. 9 and Fig. 10).

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 (Fig. 5). In experiments carried out with LAM–MPI we could not observe significant differences in measured execution 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. 5 shows a great sensitivity of MPICH in respect to such a superposition of processes. Comparable results with MPICH could only achieved in the regime of a single MPI process per physical processor node of the PM. Therefor all further numerical experiments have been performed with LAM–MPI.

Fig. 6–8 show the total execution times, the speed–up and the parallel efficiency for calculations on both test cases with SDD, DDD–Block and DDD-Point 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. Fig. 6 show the remarkable reduction in computation time with all of the 3 different parallelization methods. It can also be seen from the figures that in all cases the Dynamic Domain Decomposition (DDD) method has a clear advantage over SDD method with a slight gain in performance for the DDD–Block method. The larger amount of inter-processor communications with small amount of transfered data in the DDD–Point method in comparison with the DDD–Block method leads to a slighly decreased performance of this algorithm although the total amount of transfered data is less for the DDD–Point method. This behavior is dependent on the resolution of the numerical grid and needs further investigation for grid levels with large numbers of CV's.

Further the advantage for the DDD methods for 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 homogenous 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 methods is not as large as for the second test case, where the gas–particle flow is strongly separated and 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 (Fig. 8).

The efficiency of the work load balancing introduced to the Lagrangian approach by the Dynamic Domain Decomposition (DDD) methods can also clearly be seen from the work load distribution diagramms in Fig.'s 9 and 10. 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–Block 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 Fig. 10 it can be seen for the DDD–Block 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.

5 Conclusions

The paper presents two parallelization methods for the Eulerian–Lagrangian approach for disperse multiphase flows calculations together with their MPI implementations. Performance results are given for two typical test cases. The obtained results show besides the general applicability of the SDD and DDD parallelization methods for parallel computers with distributed memory and message passing paradigm the importance of homogenous 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 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.

Acknowledgment

The authors are indebted to Prof. M. Perić for allowing the use of his laminar CFD code FAN–3D, which was the starting point for the presented code Mistral/PartFlow–3D. Further 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., Bernert K., Schneider J. :** "Numerische Untersuchungen der Gas–Partikel–Strömung in symmetrischen Doppelzyklon–Abscheidern", VDI–Berichte, Nr. 1511, 1999

[7] **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.

[8] **Sommerfeld M. :** "Modellierung und numerische Berechnung von partikelbeladenen turbulenten Strömungen mit Hilfe des Euler/Lagrange–Verfahrens", Berichte aus der Strömungstechnik, Shaker Verlag, Aachen, Germany, 1996.

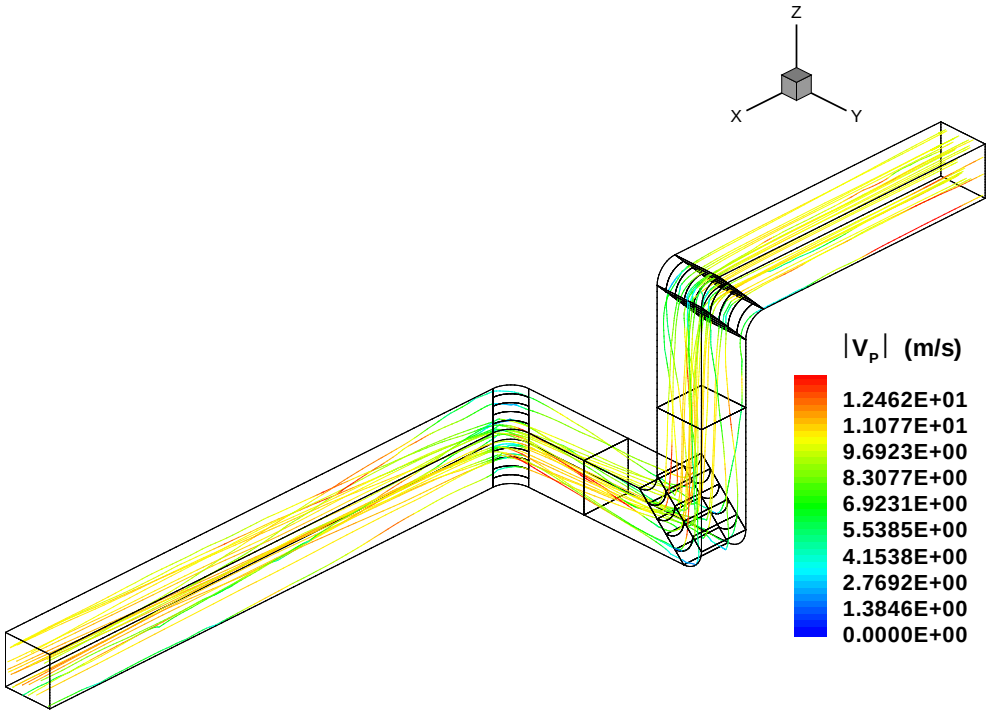

Fig. 4: Particle trajectories in the bended channel with corner vanes (test case 1).

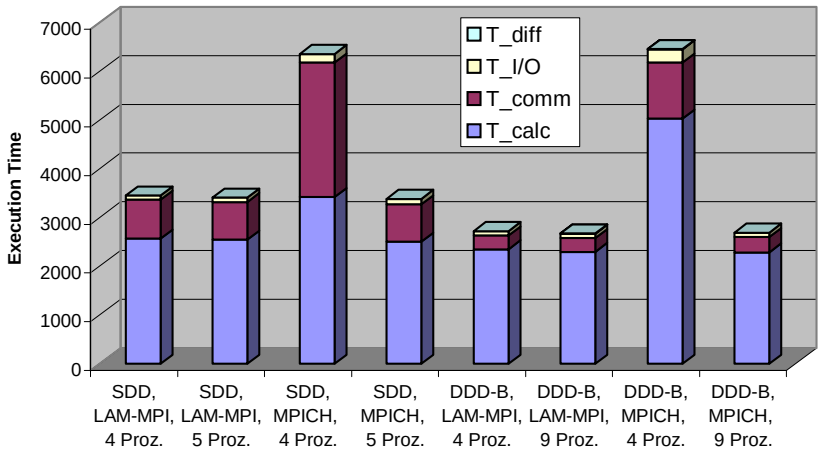

Fig. 5: Comparison of LAM–MPI and MPICH libraries in case of superposition of MPI processes on the same physical processor node of the PM.

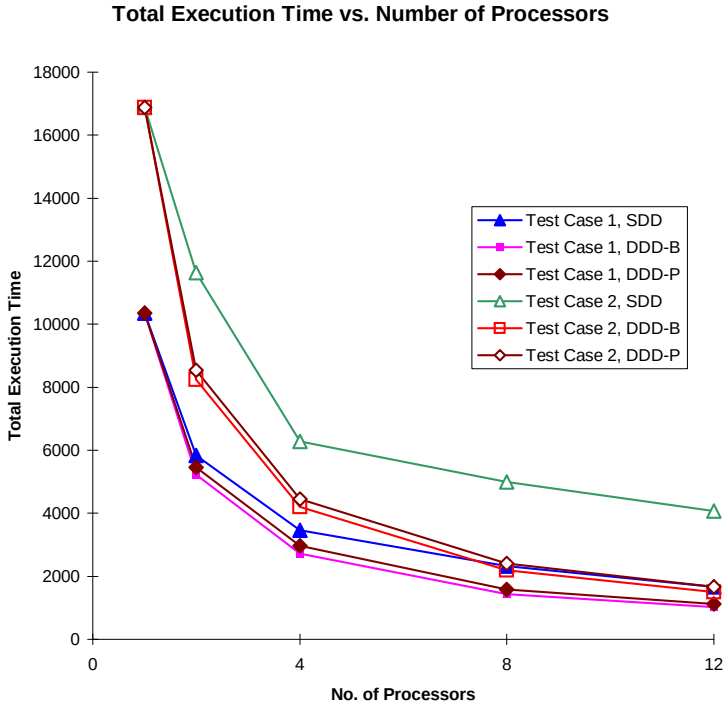

Fig. 6: Total execution times vs. number of processor nodes; comparison of parallelization methods for both test cases.

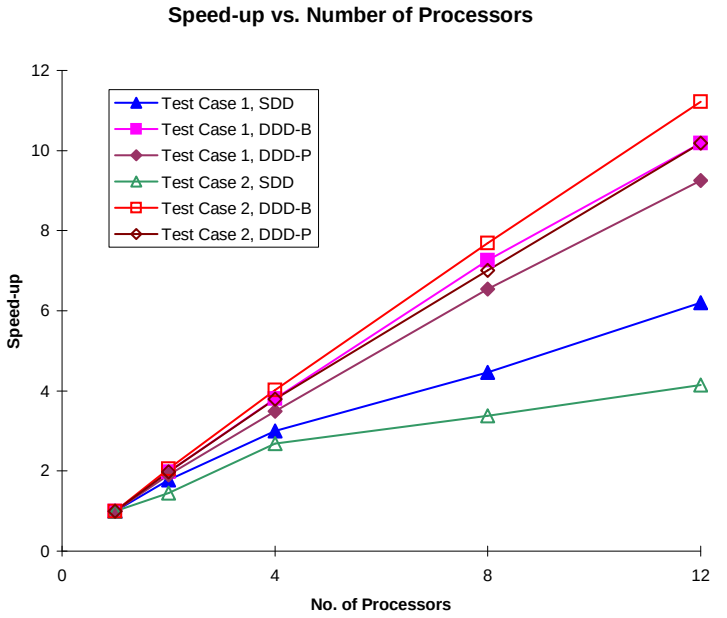

Fig. 7: Speed–up vs. number of processor nodes; comparison of parallelization methods for both test cases.

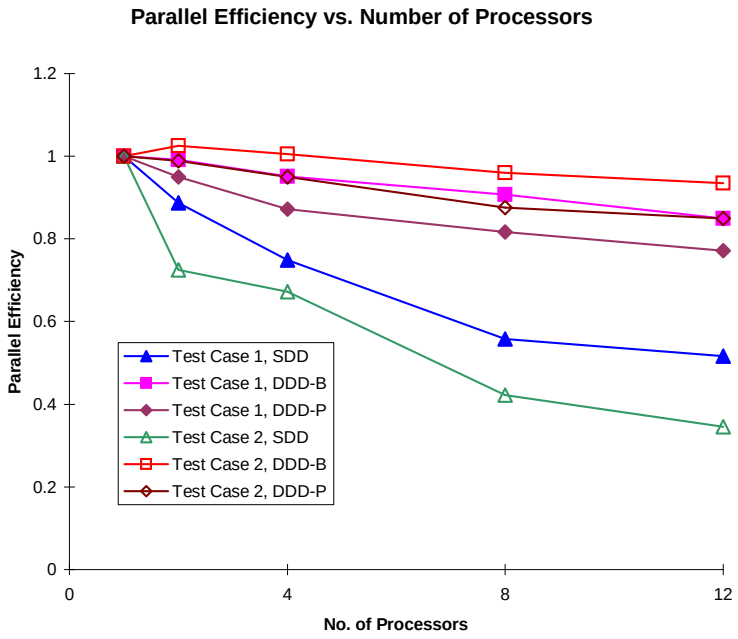

Fig. 8: Parallel efficiency vs. number of processor nodes; comparison of parallelization methods for both test cases.

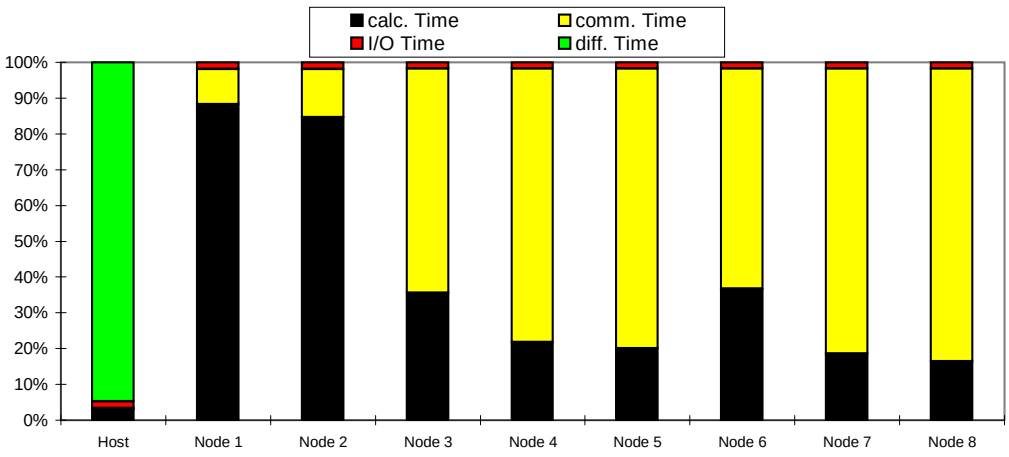

Fig. 9: Work load distribution for SDD method on test case 2 for execution on 8 processors.

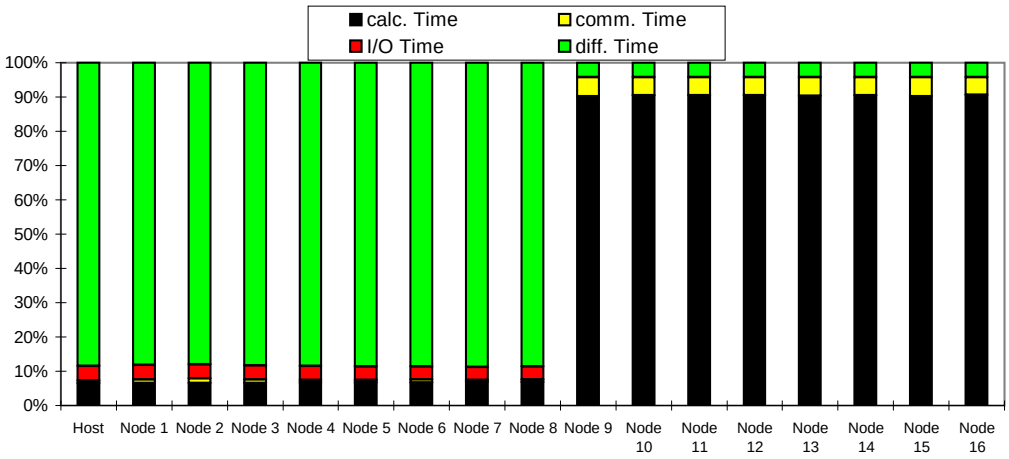

Fig. 10: Work load distribution for DDD–Block method on test case 2 for execution on 8 processors.