International Conference on Parallel Computing,
ParCo – 2001, 4.–7. September 2001,
Naples, Italy, pp. 1–9.

DYNAMIC LOAD BALANCING FOR LAGRANGIAN PARTICLE TRACKING ALGORITHMS ON MIMD CLUSTER COMPUTERS

TH. FRANK, K. BERNERT AND K. PACHLER
Chemnitz University of Technology,
Research Group of Multiphase Flow
Reichenhainer Straße 70, 09107 Chemnitz, Germany
E-mail: frank@imech.tu-chemnitz.de

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. Presented algorithms are based on a modification of Domain Decomposition method applied to a blockstructured numerical grid, as it is widely used for the parallel computation of fluid flows. Traditional algorithms apply the same static assignment of grid partitions to the processors of the parallel machine (PM) also to the Lagrangian particle trajectory calculation, which can lead to a dramatic deterioration of parallel efficiency by e.g. non-homogeneous particle concentration distribution in the flow. In the newly developed parallelization method 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 typical test cases. Results obtained on the 528-node **C**hemnitz **Li**nux **C**luster (**CLIC**) are compared with those for the Cray T3E.

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 new Dynamic Domain Decomposition method presented in this paper.

2 The Eulerian–Lagrangian Approach

Due to the limited space it is not possible to give a full description of the fundamentals of the numerical approach. A detailed description can be found in [4]. The numerical approach consists of a Navier–Stokes solver for the solution of the fluids equations of motion [1] and a Lagrangian particle tracking algorithm (Particle-Source-In-cell method) for the prediction of the motion of the particulate phase in the fluid flow field (see eq. 1).

$$\frac{d}{dt}\vec{x}_P = \vec{u}_P \quad ; \quad m_P \frac{d}{dt}\vec{u}_P = \vec{F}_D + \vec{F}_M + \vec{F}_A + \vec{F}_G \quad ; \quad I_P \frac{d}{dt}\vec{\omega}_P = -\vec{T} \qquad (1)$$

A more detailed description of all particular models involved in the Lagrangian particle trajectory calculation can be found in [3,4]. The equations of fluid motion are solved on a blockstructured, boundary–fitted, non–orthogonal numerical grid by pressure correction technique of SIMPLE kind (**S**emi–**I**mplicite **P**ressure **L**inked **E**quations) with convergence acceleration by a full multigrid method [1]. Eq.'s (1) are solved in the Lagrangian part of the numerical simulation by using a standard 4th order Runge–Kutta scheme. Possible strong interactions between the two phases due to higher particle concentrations have to be considered by an alternating iterative solution of the fluid's and particles equations of motion taking into account special source terms in the transport equations for the fluid phase.

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. 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.

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].

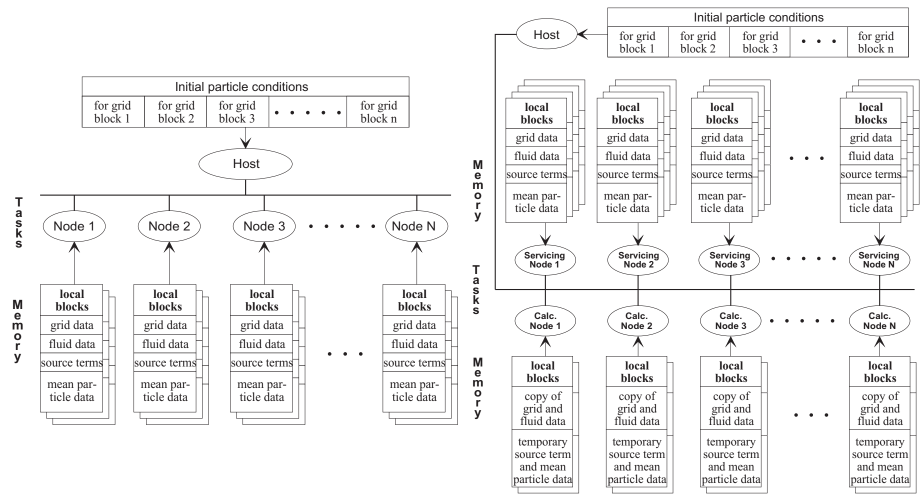

Figure 1. SDD and DDD method for the Lagrangian solver.

3.2 Parallel Algorithms for the Lagrangian Approach

Considering the parallelization of the Lagrangian particle tracking 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 1. 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.

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 1). 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.

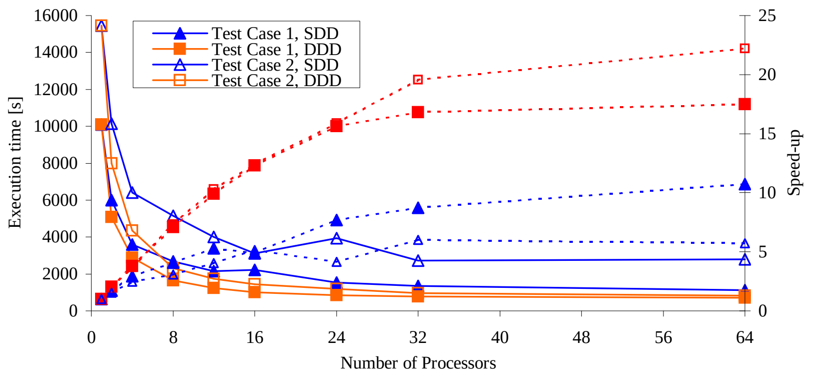

Figure 2. Parallel efficiency vs. number of processor nodes; comparison of parallelization methods for both test cases.

4 Results and Discussion

Results for the parallel performance of the multigrid–accelerated Navier–Stokes solver MISTRAL-3D has been recently published.[1] So we will concentrate here on scalability and performance results for the Lagrangian particle tracking algorithms. Implementations of the SDD and DDD methods were based on the paradigm of a MIMD computer architecture with explicit message passing between the node processes of the PM using MPI. For performance evaluation we used the Chemnitz Linux Cluster (CLIC) with up to 528 Pentium-III nodes, 0.5 Gb memory per node and a FastEthernet interconnect.

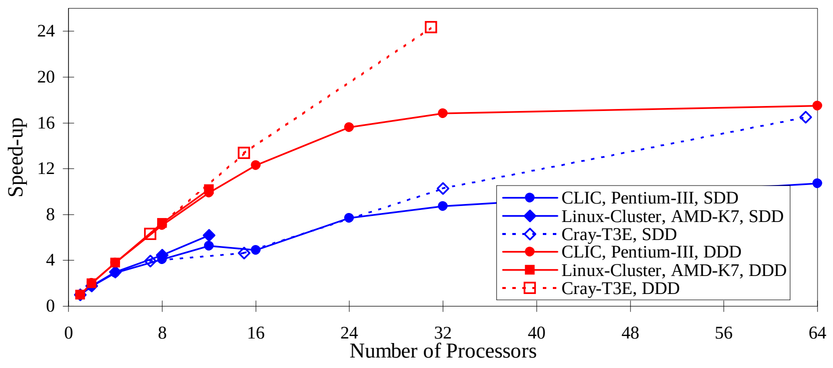

Figure 3. Comparison of parallel performace on Chemnitz Linux Cluster vs. Cray T3E.

These data were compared with results obtained on a Cray T3E system with 64 DEC Alpha 21164 processors with 128 Mb node memory.

The first test case is a dilute gas–particle flow in a three times bended channel with square cross section of $0.2 \times 0.2\,m^2$ and inlet velocities $u_F = u_P = 10.0\,m/s$ ($Re = 156\,000$). In all three channel bends 4 corner vanes are installed, dividing the cross section of the bend in 5 separate corner sections and leading to a quite homogeneous particle concentration distribution. This corner vanes have been omitted for the second test case providing a typical strongly separated gas–particle flow. The numerical grid has been subdivided into 64 blocks, the number of finite volumes for the finest grid is $80{*}80{*}496 = 3\,174\,400$. For each of the test case calculations 5000 particle trajectories have been calculated by the Lagrangian solver.

Fig. 2 shows the total execution times, and the speed–up values 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. Fig. 2 shows the remarkable reduction in computation time with both parallelization methods. It can also be seen from the figure that in all cases the Dynamic Domain Decomposition (DDD) method has a clear advantage over SDD method.

Further the advantage for the DDD method 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 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 (Fig. 2).

Figure 3 shows the comparison of test case calculations between the CLIC, an AMD–Athlon based workstation cluster and the Cray T3E. 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 for more than 32 processor nodes where a further substantial decrease of the total execution time for the Lagrangian solver could not be achieved.

Acknowledgements

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 Int. Conference on Cluster Computing — CLUSTER 2000, Nov. 28.–Dec. 2., 2000, Chemnitz, Germany.
2. **Crowe C.T., Sommerfeld M., Tsuji Y. :** "Multiphase Flows with Droplets and Particles", CRC Press, 1998.
3. **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.
4. **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.
5. Web site of the Research Group on Multiphase Flow, TUC, Germany. http://www.imech.tu-chemnitz.de/index.html – Index, List of Publications.