Proceedings of ALGORITMY 2005
pp. 42–52

ADVANCES IN PARALLEL ADAPTIVE SIMULATION ON UNSTRUCTURED MESHES

STEFAN LANG *

Abstract. Parallel simulations of time-dependent problems on unstructured meshes using h-adaptation require the cooperation of numerical methods, distributed mesh management and dynamic load balancing and migration. Parallel-adaptive solution schemes, e.g. for tracking a front, incorporate mesh changes preferably in regions of significant solution phenomena.

This work focuses on advances in parallel adaptive computing on unstructured meshes. We present recent extensions to a parallel programming model, which allow a more efficient and stable realization of small mesh changes caused by grid adaptation. For reasons of algorithmic complexity processor local changes result only in incremental operations preserving a global and consistent view onto the distributed mesh.
While tracking fronts with parallel grid adaptation processor local computation load varies significantly. Thus dynamic repartitioning of the already distributed computation load is needed. We present and compare two schemes PRCB and MCAR capable to do a rebalancing in the context of multiplicative multigrid as optimal-complexity solver inside the numerical scheme.

Two applications from reservoir engineering demonstrate how these new capabilities are efficiently used in `2D` and `3D` parallel-adaptive simulations which require mesh adaptation and dynamic load balancing and migration during run-time.

Key words. Two-Phase Flow, Parallel Computation, Multigrid Methods, Mesh Adaptation, Dynamic Load Balancing

AMS subject classifications. 65Y05, 65M55, 65M50

1. Introduction.

The numerical simulation of complex physical phenomena with finite element or finite volume methods can benefit from a number of techniques which each in itself can accelerate computation time considerably. Most notably these are grid adaptation, unstructured grids and multigrid methods. The increasing utilization of MIMD architectures has given rise to the question of how these techniques can be implemented on modern supercomputers to solve large scale three dimensional problems.

Advances in dynamic load balancing and the development of the Programming Model *DDD*, which is a fundamental tool for parallelization of unstructured mesh applications, are investigated. Their benefits are discussed by analyzing parallel-adaptive simulations in 2 and 3D. Progress in the development road map of *DDD* is discussed by example considering two features: the newly introduced *DDD*-ObjMgr Environment and the capability to maintain interfaces for communication in an incremental way.

In this paper we consider two problems from petroleum reservoir engineering, where water displaces oil in a porous medium. These problems are realized inside the MUFTE-UG framework [2, 4]. The solution develops a sharp front, therefore simulations can greatly benefit from mesh adaptation. The simulation process requires grid adaption in each time step, which involves element refinement and coarsening based on error indicators and an adjustment of the load balancing by dynamic load migration. Thus they are perfectly suited for parallel-adaptive computations.

*Interdisciplinary Center for Scientific Computing, University of Heidelberg, Im Neuenheimer Feld 368, 69120 Heidelberg, Germany

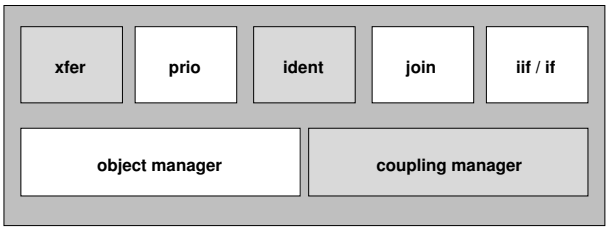


FIG. 2.1. *The DDD Programming Model with its structure into modules. Modules with white colored boxes are new, redesigned or enhanced significantly. Gray colored boxes have not been changed to considerable extend.*

While there are many developments which combine subsets of the aforementioned features, the number of software frameworks that aim at the same generality as the presented parallel-adaptive scheme realized in UG, is substantially smaller. As an ongoing project SIERRA [6] can be mentioned as a development effort for parallel-adaptive finite element simulations.

Even though the capabilities of 3D parallel adaptive methods with dynamic load balancing have been well realized, it seems like the complexity of the software engineering effort has prohibited more widespread use. We hope that the results presented in this paper help to enhance the acceptance of these methods.

2. A Programming Model for Unstructured Hierarchical Meshes. To enable unstructured mesh adaptation and load migration, elaborate data structures and capabilities for dynamic mesh changes are of great importance. Only with the availability of such functionalities it becomes possible to build up a complex mesh hierarchy, like a locally adapted *and* dynamically redistributed multigrid. It becomes immediately apparent that straightforward usage of message-passing programming models, e.g. MPI, will certainly be doomed to failure due to the lack of abstraction of communication and distribution mechanisms.

Therefore, an abstract programming model supporting high-level operations on the distributed grid objects has been designed and integrated into the *UG*-library. From the architecture's viewpoint this *Dynamic Distributed Data* (*DDD*) model is implemented as a *UG* subsystem; a standalone version is nevertheless available in order to allow parallelization of other projects as well.

The basic abstraction used by *DDD* is the notion of *distributed objects*. We call the data structures in the application, which are distributed on other processors, DDD objects. Each DDD object is assigned a *global unique identifier* (GID) and a set of information tuples $(proc, prio)$, so called *couplings*. The coupling tuples store the owning processor id of a copy of the same object and its priority on that processor [3].

The current *DDD* component design is shown in Figure 2.1. In this figure modules `join, iif, prio` and `object manager` have been introduced newly or have been subject to major redesign. All these modules can be used to manipulate the distributed graph structure of references between distributed objects and to invoke efficient communication procedures for collections of distributed objects. A short description of *DDD* components with emphasis on new or enhanced functionality follows:

- Interfaces (if): Supports communication operations on existing static data topologies. Interfaces are subsets of distributed objects at inter-processor boundaries and can be used in a transparent manner after their initial definition. They are kept consistent despite all dynamic data changes by other

DDD-Obj MgrEnv	P	closure	adaptgridL	unify	overlap
no	32	5.60	1.65	25.15	47.70
yes	32	5.10	1.79	6.23	8.05
no	64	3.62	0.87	15.94	27.03
yes	64	3.23	0.99	4.65	5.25
no	128	2.77	0.48	13.60	19.95
yes	128	2.50	0.60	4.92	4.32

TABLE 2.1

Time in seconds needed for several phases of parallel mesh adaptation with(out) using DDD's `ObjMgr` *Environment. While some overhead is introduced in local mesh modification (adaptgridL), improvements can clearly be seen in phases unify (ident + join) and overlap.*

DDD-Module	P	closure	adaptgridL	unify	overlap
if	16	22.78	1.52	8.61	2.64
iif	16	6.82	1.52	2.72	2.72
if	32	20.58	0.71	8.99	1.99
iif	32	5.83	0.71	2.61	2.03

TABLE 2.2

Time in seconds needed for several phases of parallel mesh adaptation with(out) using DDD's `iif` *Module. In both cases DDD's* `ObjMgr` *Environment is used. Improvements can clearly be seen in phases closure and unify (ident + join).*

DDD components.

- Incremental Interfaces (iif): The incremental update version for interface rebuild reduces the algorithmic complexity to renew a single interface from $O(Nlog(N))$ to $O(N_{old}) + O(N_{new}log(N_{new}))$, where $N = N_{old} + N_{new}$ is the sum of old, already present couplings in this interface and newly created couplings, which have to be integrated into it. In many examples with local phenomena (e.g. tracking of two-phase flow fronts) N_{new} is compared to N_{old} quite small: $N_{new} \ll N_{old}$, thus the interface update procedure has nearly optimal complexity.
- Priority (prio): Changing the priority of distributed objects in a consistent way is needed during transfer and coarsening. This can be done efficiently using this module.
- object manager: During grid adaptation or dynamic load migration distributed objects are created or migrated between processors. To keep track of references between mesh objects access from GIDs to local objects has to be provided. For efficiency reasons references are reconstructed by iterating over distributed objects in increasing GID ordering. The reconstruction itself is performed by localizing (substituting) a GID into a reference to the corresponding copy of the distributed object in the local address space. Maintaining the GID ordering in an incremental way results in a $O(N)$ complexity algorithm for the interesting case of parallel adaptivity. This is done inside the object manager environment marked with `DDD_ObjMgrBegin()` and `DDD_ObjMgrEnd()` calls.
- Transfer (xfer): Provides procedures to create object copies on remote processors or to delete local object copies. This enables dynamic changes of the data topology at run time. This feature allows an easier implementation of a

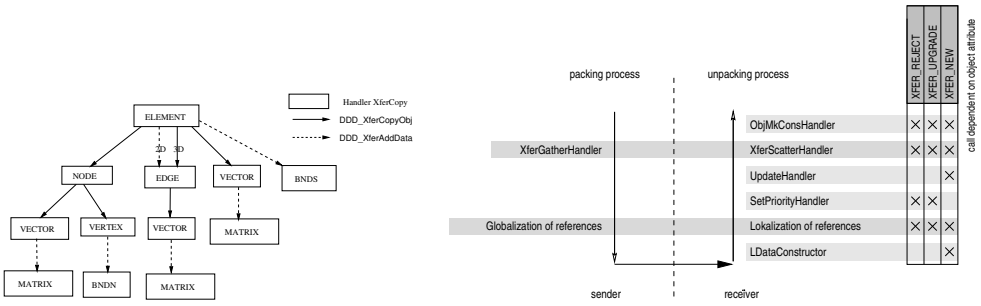


FIG. 3.1. *Object copy tree for object selection (left) and handler stack (right) called during the migration process.*

dynamic load migration facility with flexible grid overlapping strategy.

- Identification (ident): Creation of new distributed objects is performed via identification of local objects. This is possible during the complete program run and enables dynamic grid changes (e. g. for adaptive grid refinement).
- Join (join): An extension of the identification functionality. Join allows new objects to join into already existing distributed objects. For objects belonging to the mesh overlap a join operation ensures an extension or deletion of couplings via indirect messaging. This is crucial for code stability since missing couplings or duplicated distributed objects can lead to fatal situations in distributed grid adaptation and dynamic load migration.

Tables 2.1 and 2.2 show results for 3D fixed-sized computations of the Buckley-Leverett experiment at different architectures. The savings of using the *DDD*-ObjMgr Environment can clearly seen in phases unify and overlap, whilst using *DDD* incremental interface facility reduces closure and unify times, where most interfaces have to be renewed.
Unify is the added time needed for building new distributed objects using ident and join modules of *DDD*. Overlap measures the timings to transfer ghost elements in grid adaptation, since we use a one-element ghost overlap that has to be updated both in grid adaptation and load migration. Closure is the time to compute a conforming grid closure and adaptgridL counts time needed for all local mesh changes: creation and deletion of objects.

3. Dynamic Load Balancing and Migration. In a parallel environment, local grid adaptation involves the need to rebalance the computational load. This stage involves both, determining a new load balancing and dynamically redistributing the objects of the mesh in a separate migration step. This section discusses both parts of the rebalancing process in the context of multigrid.

3.1. Dynamic Load Balancing. Load balancing a single grid level means finding a mapping $\mathcal{LB} : \mathcal{E} \to \mathcal{P}$, where $\mathcal{E}$ is the element set and $\mathcal{P}$ is the set of processors. Via the dual graph $G = (V, E)$ of the grid, V is defined by the element centers and E is given by the element neighborships. Load balancing onto $|\mathcal{P}|$ processors can be formulated as graph partitioning problem: Minimize the edge separator (objective) whilst building $|\mathcal{P}|$ equal-sized partitions (constraint), each of size $|\mathcal{E}|/|\mathcal{P}|$. This optimization problem is NP-complete and is referred to as single-objective/single-constraint (SOSC) partitioning problem. Because of the NP-completeness partitioning is performed by applying heuristics, which tend to reduce necessary communication dur-

ing parallel computation to a reasonable volume. Many partitioning methods for the static SOSC partitioning problem have been proposed. All these partitioning methods take into account only the interface length of interprocessor boundaries as objective to minimize the total communication volume. Further costs of communication, e.g. message startup times, are not considered by these schemes.

The quality of a multigrid load distribution (constraint) is determined by

$$ElImbal = \sum_{l=0}^{L} \frac{|\mathcal{E}^l|}{|\mathcal{E}|} \left(\frac{\underset{p\in\mathcal{P}}{\text{Max}}\ (|\mathcal{E}_p^l|)}{\frac{|\mathcal{E}^l|}{|\mathcal{P}|}} - 1 \right) \qquad NdImbal = \sum_{l=0}^{L} \frac{|\mathcal{N}^l|}{|\mathcal{N}|} \left(\frac{\underset{p\in\mathcal{P}}{\text{Max}}\ (|\mathcal{N}_p^l|)}{\frac{|\mathcal{N}^l|}{|\mathcal{P}|}} - 1 \right) \tag{3.1}$$

for element sets $\mathcal{E}_p^l$ and node sets $\mathcal{N}_p^l$, where p is a processor id and l a specific mesh level. For a complete list of quality measures to evaluate the load balancing of a multigrid see [5].

Communication, present in parallel multigrid methods, can be categorized into horizontal communication inside grid levels during smoothing and vertical communication between grid levels in prolongation and restriction. Since we use multiplicative multigrid methods, the smoothing process has to be performed grid level by grid level. Because of efficiency reasons, each grid level has to be distributed over all processors. This leads to a multi-constraint load balancing problem with L constraints, one for each grid level. Furthermore communication implies $2L-1$ objectives to minimize communication during smoothing and defect transfer. This multi-objective/multi-constraint (MOMC) partitioning problem, which arises when multiplicative multigrid is used as solver, is much harder to solve than the single-objective/single-constraint (SOSC) load balancing problem to partition a flat grid.

Comparable differences exist, when load balancing requirements of multiplicative and additive multigrid schemes are analyzed. Since additive multigrid allows smoothing on all grid levels at the same time, load balancing can be formulated as SOSC partitioning problem of a weighted graph, when we restrict to load balancing of whole element trees[1].

To compute a load balancing we apply in this paper two distinct methods: A Parallel Recursive Coordinate Bisection variant (PRCB) and a graph-based methods, Multi-Constraint Adaptive Repartitioning (MCAR). Both schemes minimize the communication volume with the difference that the first one handles mesh levels of a multigrid independently, whilst the second one works on a single graph and is especially adapted to needs of multiplicative multigrid methods. In regard to the above categorization, PRCB is a SOSC method and MCAR belongs to class of SOMC methods. A more detailed description of these schemes can be found in [5] and is not focus of this paper.

3.2. Dynamic Load Migration. A key feature of the *UG* framework is its capability to dynamically migrate grid objects between processors during run-time. Thus a computation needs not to be interrupted, but continues after load transfer with a balanced work load on each processor. This difficult task is supported by *DDD* to a large extend.

The migration process is transaction oriented: First all transfer commands are passed to *DDD*, which performs bookkeeping. Selection of objects, that have to be transfered, starts at element objects. Elements store the load balancing information,

[1] An element tree is defined by a coarse grid element and all child elements created through refinement.

which has been computed in the load balancing process. Via the object copy tree shown in Figure 3.1 transfer commands for dependent objects are generated. In a second step these commands are executed by a single migration call to a migration module. This migration phase itself has various stages. Packing the data objects into buffers, sending and receiving of message buffers and unpacking objects of the data structure. Both the whole structure and complicated technical aspects of these stages are supported by *DDD*'s xfer module in a generalized way for arbitrary irregular data structures. Keeping this powerful functionality behind *DDD*'s well defined interfaces leads to a stable, maintainable and extendable migration subsystem.

4. Two-phase flow equations in porous media. The flow of two immiscible fluid phases in a porous medium is described by the equations for the conservation of mass and the generalized Darcy's law. We denote by w the wetting phase and by n the non-wetting phase. The system of nonlinear partial differential equations can be written in phase pressure–saturation formulation with the unknowns p_w and saturation S_n.

$$\begin{aligned} \frac{\partial(\Phi\varrho_w(1-S_n))}{\partial t} - \nabla\cdot\left(\varrho_w\frac{k_w}{\mu_w}K(\nabla p_w - \varrho_w g)\right) - \varrho_w q_w = 0 \\ \frac{\partial(\Phi\varrho_n S_n)}{\partial t} - \nabla\cdot\left(\varrho_n\frac{k_n}{\mu_n}K(\nabla p_w + \nabla p_c - \varrho_n g)\right) - \varrho_n q_n = 0. \end{aligned} \tag{4.1}$$

Here, Φ is the porosity, ϱ_w and ϱ_n are the densities of phase w and n, q_w and q_n are the source and sink terms for each phase, K the tensor of absolute permeability, k_w and k_n the relative permeabilities, μ_w and μ_n the dynamic viscosities and g the vector of gravitational force. The capillary pressure $p_c = p_n - p_w$ establishes the connection between the phase pressures p_w and p_n and is a function of S_w. There are several p_c–S_w correlations known in the literature, we use the Brooks-Corey model

$$p_c(S_w) = p_d\left(\frac{S_w - S_{wr}}{1 - S_{wr}}\right)^{-\frac{1}{\lambda}}, \tag{4.2}$$

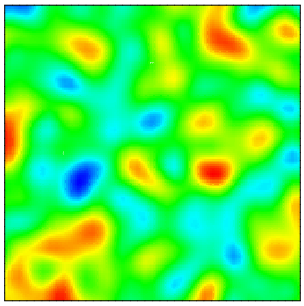
with the entry pressure p_d and the residual wetting phase saturation S_{wr}.

Together with a suitable domain $\Omega \subset \mathbf{R}^n$, $n = 2, 3$ and adequate boundary values and initial values equation (4.1) describes a well posed initial value problem.

The vertex-centered finite volume discretization of (4.1) and a detailed description of the derivation of the equations, modeling aspects, and solution strategies for the fully coupled and fully implicit method are described in [1] and [4]. Here an implicit time discretization is employed. The nonlinear system that arises in each time step is solved iteratively with a Newton method and a multigrid solver for the linearized system. The Newton method employs line search to achieve global convergence and the multigrid method is a standard V cycle with truncated restriction.

For local grid adaptation we need to apply some kind of error indicator to detect error critical areas. Here, the basic idea for an error indicator is to locate the regions with a sharp interface between the phases. A sharp interface can be identified by regarding the gradient of the saturations.

As error indicator to control refinement and coarsening we adopt a simple min-max indicator. The element local indicator $\eta_j, j = 1 \ldots N$ is defined by the difference between maximum and minimum value of the saturation value S_n of each element. The global maximum $max(\eta_j)$ is used to define tolerances for mesh adaptation.



DLB Method	P	1	2	4	8	16	32
PRCB	*TNLS*	406	220	123	74	45	43
MCAR	*TNLS*	406	230	124	66	46	39

FIG. 5.1. *Permeability field used in the fivespot experiment (left side). Time [min] for complete fivespot simulation consisting of 51 time-steps. Compared are two dynamic load repartitioning schemes PRCB and MCAR. Number of unknowns varies from 87000 to 123,000. Computations are performed on a Intel-based cluster with ethernet network (right side).*

These tolerances are given by

$$tol_{refine} = ref^{max} max(\eta_j) \qquad tol_{coarsen} = cor^{max} max(\eta_j). \tag{4.3}$$

An element is considered either for refinement, if the error indicator η_j for element e_j is above tol_{refine}, or for coarsening, if η_j is below $tol_{coarsen}$. For practical purposes the results obtained using this heuristic indicator are accurate enough.

5. The fivespot problem. The fivespot problem is a classical model problem from petroleum reservoir engineering. In order to exploit an oil reservoir, water is pumped into the reservoir and displaces the oil. The model configuration consists of a square in whose corners oil is produced. Water injection happens in the center of the square. The symmetry of the problems allows for the reduction of the problem to the right upper quadrant of the domain.

The numerical fivespot experiment **fivehet2d** exhibits a very distinct front traversing the domain. If the permeability is not homogeneous throughout the domain, the front will develop a complex shape. For the experiment the heterogeneous permeability distribution shown in figure 5.1 was used. The developing front is used to investigate how different dynamic load balancing schemes are able to capture and repartition it. The two load balancing methods of concern, PRCB and MCAR, are described in section 3.

The pictures in figure 5.2 visualize different qualitative behavior at the front. The quality in terms of interface lengths depends visibly on the load balancing method. The graph based dynamic load balancing variant (`MCAR`) produces much smoother and more compact partition shapes, since it considers the connectivity information of front elements on each level. `PRCB` doesn't capture the shape of the front and produces some unfavorable cuts through the front. This creates a long processor interface which requires more communication. Table 5.1 shows simulations times on different processor numbers for both load balancing schemes in comparison. While the simple and fast PRCB is favorable on moderate processor numbers, the much more complicated MCAR scheme pays off for larger processor counts with higher communication demands.

[2]

[2]Note that in figure 5.2 the load balancing is shown from the top of the grid hierarchy, i.e. several distinct grid levels are visible in one picture.

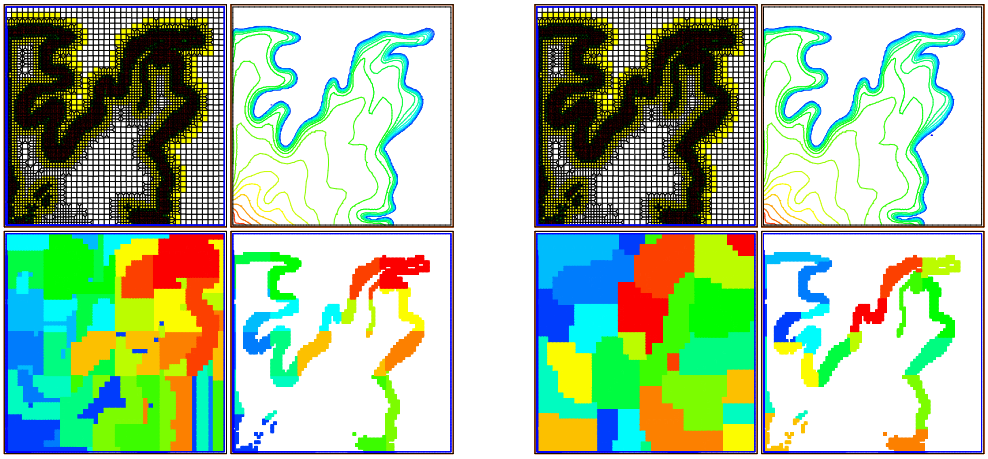


FIG. 5.2. *Development of the adaptive front at the phase boundary for load balancing with PRCB and MCAR as dynamic load-balancing methods: adapted mesh, phase saturation in log-scale, load balancing of whole multigrid (left) and front (right).*

6. The Buckley-Leverett problem in 3d. Displacement processes of immiscible fluids are characterized by the viscosity and density differences of the fluids and the surface tension forces at the interfaces of the fluid phases. The interface tends to be unstable and under certain circumstances viscous fingering can develop. This behavior can be observed in simulations too, if the grid resolves the front sufficiently fine. On coarse grids fingering can not develop because of numerical diffusion. Many time steps have to be solved before viscous fingering develops, it is therefore an ideal phenomena in which to combine mesh adaptation and parallel computing.

The Buckley-Leverett experiment describes the displacement of oil by water in a quadratic tunnel under consideration of gravitational forces. The left side of the domain has an inflow condition associated, the right side an outflow condition. The experiment **bl3d** was chosen so that it is still possible to run on a single processor computer with sufficient memory for one million unknowns ($\sim$ 500000 grid nodes).

A first study compares scalability behavior of the bl3d experiment on different architectures. The Cray T3E-1200 as classical HPC-machine (T3E), and two off-the-shelf PC-based architectures - with Myrinet-network (HELiCS) and with ethernet (PPC).

Results in table 6.1 show that on a Cray T3E and the HELiCS comparable speedups can be achieved for this fixed-sized problem. On moderate processor counts up to 32 processors PPC gives reasonable speedups, but clearly scalability limitations become apparent on higher processor counts. Table 6.2 shows a comparison which investigates how dynamic load migration influences computing time. The initial load balancing is chosen equally distributed work load by load balancing into stripped partition in flow direction, see figure 6.1. During the first time steps the front develops only in the direction along the load balancing stripes and work load stays equally distributed among processors. But when fingering occurs because of instabilities in the phase front, the work load differences between processors increase. This experiment shows how an already good load balancing can be improved by dynamic load migration and thus the time for the solution process can be reduced.

Symbols in table 6.2 are number of processors (P), used load balancing method

ARCH	P	1	4	16	64	128	256
	N	1.1M	278K	72K	19K	10.6K	5.9K
T3E	*TNLS*[s]	233 921	72 575	17 597	4 599	2 423	1 354
HELICS	*TNLS*[s]	-	23 535	6216	1638	889	448
PPC	*TNLS*[s]	-	35 048	9 364	3 019	-	-
T3E	$S_{NLSOLVE}$	1	3.22	13.3	50.9	96.6	173
HELICS	$S_{NLSOLVE}$	-	3.22!	12.2	46.3	85.2	169
PPC	$S_{NLSOLVE}$	-	3.22!	12.1	37.4	-	-

TABLE 6.1

Comparison of fixed-sized problem scalability on different architectures: T3E, HELICS and PPC. N indicates the number of unknowns per processor, TNLS the total time for numerical solution in seconds, S_x denotes the speedup.

P LB	E N	AvgE AvgN	ElImbal NdImbal	HNIFSUM HNIFMAX/HNIFIMB	TADAPT TLB/TMIG	TNLS TIT/NIT
32	564277	21053	56.4	144664	27.6	13766
STAT	455766	17725	45.3	6526/40	0.0/0.0	51.0/4.84
32	595991	22211	12.3	184324	33.3	13825
RCBXY 0.2	477271	18565	19.8	7701/31	0.9/50.4	39.6/4.81
32	592823	22095	16.8	186711	34.4	13804
RCBXY 0.3	475218	18484	21.7	7956/33	1.0/54.7	40.9/4.81
64	659219	12263	70.7	228138	23.1	14925
STAT	514336	10036	54.7	5531/51	0.0/0.0	31.8/4.91
64	685768	12763	14.7	287570	27.1	14427
RCBXY 0.2	526698	10305	28.3	5858/29	0.8/35.5	23.4/4.87
64	683470	12719	20.4	287161	26.8	14638
RCBXY 0.3	525797	10284	30.9	5970/31	0.8/38.4	24.5/4.87
128	739935	6891	74.9	385184	20.4	15268
STAT	549129	5408	63.4	4689/54	0.0/0.0	18.3/4.89
128	755752	7039	29.4	452317	25.9	13534
RCBXY 0.4	556744	5486	46.9	4680/31	0.8/34.0	14.6/4.86

TABLE 6.2

Parallel scalability of experiment ***bl3d*** *for load balancing with* `RCBXY` *in flow direction and fixed load balancing* `STAT`*. The table entries are averaged over all time steps.*

(LB), total number of elements/nodes (E/N), averaged number of elements/nodes per processor (AvgE/AvgN), imbalance in element/nodes (ElImbal/NdImbal). Measures to evaluate communication are total communication volume (HNIFSUM), maximal volume (HNIFMAX) and volume imbalance (HNIFIMB). Only the horizontal interfaces are evaluated, since these interfaces dominate the communication during multigrid solution of the linearized problem. Further columns list timings for grid adaption (TADAPT), load balancing (TLB) and load migration (TMIG) and timings for nonlinear solution (TNLS), one multigrid cycle (TIT) and number of multigrid cycles to convergence (NIT).

The results in table 6.2 show averaged values of a simulation with a fixed load balancing compared to a one with dynamic load balancing using `RCBXY`. `RCBXY` is a variant of `RCB` that takes the flow direction into consideration. Dynamic load balancing is only performed if a given value of load imbalance (ElImbal) is reached. Thus load balancing is performed only on demand after a couple of time steps.

In the example analyzed we used an tolerated imbalance of 0.2 and 0.3. Even though the initial load balancing is very satisfactory, the dynamic load migration can improve the solution time, which is the dominant part of the whole simulation process, by ∼25%. Remember that in most applications local phenomena appear in much more complex shapes, see e.g. fivespot example. Thus dynamic load balancing and migration are obligatory to perform a complete simulation over many time steps with a significant speedup and without getting into trouble with out-of-memory situations on single processors.

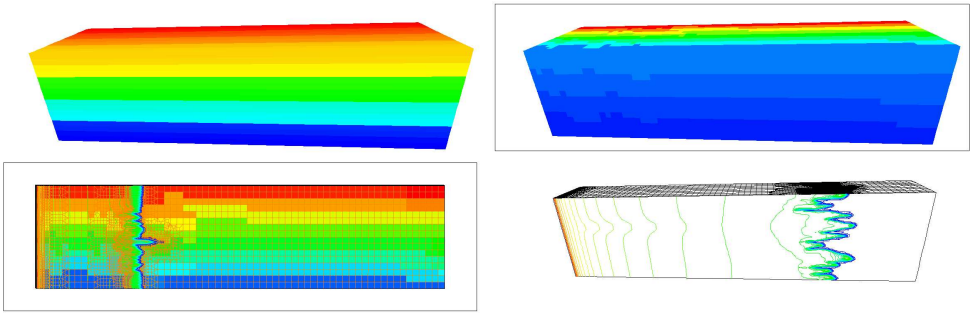


FIG. 6.1. *Load balancing for experiment* ***bl3d*** *in the static case and for the dynamic case (framed pictures).*

7. Conclusion. In this paper we present methods and concepts for parallel adaptive simulation of time-dependent, nonlinear partial differential equations.

Parallelization is realized with the innovative programming model *DDD*. Only by changes inside the programming model implementation we could improve performance of parallel mesh adaptation and load migration significantly. Since the simulation code itself has not been altered, this shows that a high abstraction level of the parallelization approach helps to overcome important hurdles of software engineering: efficiency, maintainability, and stability of the whole parallel-adaptive scheme.

Both dynamic load balancing schemes PRCB and MCAR show significant speedups on the fixed-sized fivespot example, even on a architecture with slow interconnection network. For higher number of processors the multi-constrained MCAR scheme has proven to be preferable over the simpler and faster PRCB scheme.

Further we present a comparison of architectures for a fixed-sized 3D parallel-adaptive simulation. Results show that a commodity based cluster with fast interconnection network shows similar scalability than an expensive vendor-specific architecture up to 256 processors. An ethernet-based cluster scales in this case only up to about 32 processors.

In a further experiment we apply dynamic load balancing and migration to treat load imbalance during evolution of the two-phase front. Results show that even for this worst-case situation a gain of 25% in the iteration time of the multigrid solver can be obtained by dynamic load balancing schemes compared to static load balancing.

REFERENCES

[1] P. Bastian. 1999. Numerical Computation of Multiphase Flows in Porous Media. Technischer Bericht, Technische Fakultät der Universität Kiel. Habilitiationsschrift.

[2] P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert, and C. Wieners. 1997. UG - a flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science*, 1:27–40, 1997.

[3] K. Birken. 1998. Ein Modell zur effizienten Parallelisierung von Algorithmen auf komplexen, dynamischen Datenstrukturen. Ph. D. thesis, Universitaät Stuttgart.

[4] R. Helmig. 1997. Multiphase Flow and Transport Processes in the Subsurface: A Contribution to the Modeling of Hydrosystems. Springer.

[5] S. Lang, G. Wittum. 2004. Large-Scale Density-Driven Flow Simulation Using Parallel Unstructured Grid Adaptation and Local Multigrid Methods. *Concurrency and Computation: Practise and Experience. To be published.*

[6] J.R. Stewart, C. Edwards. 2002. Mathematical Abstractions of the SIERRA Computational Mechanics Framework. Proc. of 5th World Congress on Comput. Mech.