Adaptive Data Structure Management for Grid Based Simulations in Engineering Applications

Jérôme Frisch, Ralf-Peter Mundani, and Ernst Rank

***Abstract*—This paper describes a hierarchical adaptive data structure management used for typical engineering simulations such as temperature diffusion problems or computational fluid dynamic problems. Sketches for using an adaptive non-overlapping block structured grid in a distributed manner are deployed and sample simulations are computed to underline the used concepts. Furthermore, a small outlook is given to future work planned in this area, how to improve the implemented version of the code, as well as how a parallel concept might look like.**

***Index Terms*—data management, adaptive grid, non-overlapping block structured grid, ghost cells, transient temperature diffusion equation, computational fluid dynamics**

I. Motivation

In modern engineering simulations of any kind, accurate geometric representation is playing a key role for describing a certain problem. Figure 1 shows a huge, detailed power plant model containing more than 12,5 million triangles. It can be seen that both very small but also large triangles are present. If a detailed computational fluid dynamics simulation around this power plant should be computed, there is the necessity of simulating a large volume of surrounding air in order to reduce the effects of boundary conditions from the enclosing domain to the plant itself. The easiest way to perform this task is to uniformly refine the complete computational domain until the smallest triangle is included or until a given geometric accuracy is reached. A reasonable resolution would contain more than $2 \cdot 10^9$ uniform hexahedral cells for which the solution of the CFD problem would take even on huge super computers a quite long time. The consequence of this uniform refinement is a very fine grid on places where it is not mandatory from a geometric point of view. A solution is an adaptive refinement only in areas where more information is necessary or helpful to increase simulation results, whereas a coarse grid can be used in areas of low information density. Unfortunately, this adaptive handling of data asks for a more complex data structure to manage geometry and boundary conditions.

In this paper, an adaptive data structure management framework based on non-overlapping block structured grids is presented, in which two engineering applications are tested. The construction of the block structured grid is based on a recursive hierarchical build-up. The concept is explained and demonstrated for a transient temperature distribution and for a computational fluid dynamics scenario.

This paper describes work in progress in order to construct a data structure and a software framework which is able to deal with grid refinement and is prepared in such a way that a future parallel distribution to multiple systems for running a massive parallel application is possible.

Adaptive grids are quite well studied in literature (c. f. Samet [2], Barequet et al. [3]) and applied to specific problems (c. f. Coelho et al. [4]), but in contrast to Coelho et al., the sub-grids are surrounded with so called ghost cells as described later in section II.A, even if they would not be necessary for a computation running on a machine using a shared memory approach. The term 'distributed' refers to the fact that the data structure is not allocated as one block in memory, but as a hierarchy of grids that are all maintained separately and that are 'coupled' via update functions (described in section II.B) between two hierarchical levels. In future, different grids reside on different machines using a distributed computing approach, as state-of-the-art solutions for engineering simulations, such as fluid dynamic problems, are almost always using a parallel approach in order to cope with the high data amount.

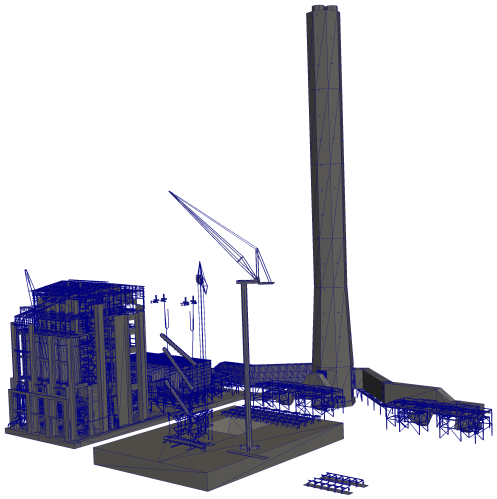


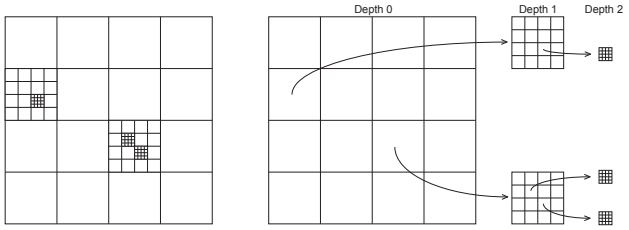
Figure 1. View of a power plant model [1] consisting of 12,748,510 triangular surfaces organised in 1,185 groups.

As geometric representation the authors chose a block structured approach as a trade-off between geometric accuracy and complexity in data handling. On the one hand, a fully detailed geometric description using unstructured grids can represent the geometry with a very high level of detail, using not too much cells. Unfortunately the data management handling is very complex and the performance is not so high. On the other hand, structured orthogonal grids have a very easy data handling and thus, very high performance regarding computation time but cannot represent the geometry quite accurate. Furthermore, the generation of input data for a structured block oriented mesh from an arbitrary surface mesh using an octree based space partition scheme is much easier to automatise than the generation of an arbitrary unstructured mesh.

Chair for Computation in Engineering, Technische Universität München, Arcisstrasse 21, 80290 München, Germany. Corresponding author contact: Phone: +49 89 289–25128, e-mail: frisch@tum.de

II. Adaptive Data Structure Management

The concept of the adaptive data structure management is based on non-overlapping block structured orthogonal grids. Each block is constructed out of orthogonal, equidistant pseudo-cells which can be regarded as real data cells describing fluids, solids, etc. or they may contain a link to a sub-grid. The possibility of local refinement gives the code the ability to adapt quite good to a complex geometry while still using orthogonal grids on which finite difference or finite volume schemes can be adapted fast.



(a) hierarchical 2D block oriented data structure

(b) recursive construction sequence of the 2D data structure from the left hand side

Figure 2. 2D block oriented data structure

The implemented code is designed for managing 3D grids. For the sake of simplicity some of the following examples are given for 2D grids only even if they can be applied to 3D grids. A scheme of the block structured grid can be seen in Figure 2. In this case, the main grid (identifiable by the depth zero), as well as the sub-grids, have a size of 4x4. These numbers result from a mere choice for an adequate visualisation. To reduce the overhead of grid management, a higher choice of cell amounts in the main grid level is reasonable. The arrows represent links through a pointer data structure from the respective pseudo-cell to the sub-grid and back.

In this case, a block structured approach is preferable to a standard octree, as a high depth would be necessary to acquire the desired detailed geometry. Furthermore a neighbouring search algorithm is called very often as a result of using finite difference stencils which is quite costly for octrees. Hence, we chose a non-overlapping block structured grid where the sub-grids are regular and neighbouring relations of finite difference stencils reduce to index shifting in data array access.

In order to keep the data structure as flexible as possible according to adaptive refinements, no links with pointers from single cells to neighbouring cells were established, but a ghost cell scheme was used.

A. Ghost Cell Scheme

The ghost cell scheme introduces one layer of cells all around the sub-grids as indicated in Figure 3 by gray-shaded cells around a 4x4 sub-cell grid. From depth zero to depth one there are two links to different sub-grids. The arrows from cells to ghost cells on the same depth level are not pointer links, but a mere indication which cell contents is copied during the update step described in the next sub-section.

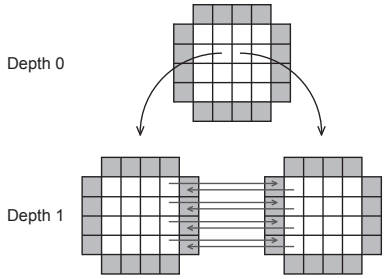


Figure 3. Example of the ghost cell scheme: ghost cells are marked in gray.

B. Update Step

The necessity of performing an update step and the ghost cell scheme resembles to a parallel computation approach. Using such a scheme for the design of the code – even in a serial case – is speeding up the later process of advancing to a parallel version.

The update step is performed after each computational step, meaning that after a specific computational algorithm has been performed on a sub-grid without following links to sub-cells e. g., the update function is called. Some inherent synchronisation is implied by the order of execution of update functions. Hence, to respect the order, it is necessary to treat the complete block structured grid in a bottom-up manner, starting on the deepest sub-grid and ending on the main grid at level zero.

To order the sub-grids in a bottom-up fashion, two data structures, namely a queue (FIFO) and a stack (LIFO) are used. At first, both stack and queue are empty and the main grid is added to the structures. While the queue is used for iterating through the different sub-grids into the depth, the stack is accumulating links to the complete data structure in such an order that the last elements pushed to the stack will be removed first, which delivers the bottom-up approach.

After the ordering of the pointers to the different sub-grids, different update procedures have to be chosen according to the computational desires. In case of a finite difference scheme, the total mean values for one sub-grid are computed and passed on to the corresponding parent cell for further processing. By starting from the deepest sub-grid, one can assure that only updated values are taken into account for performing computations on the current sub-grid.

Afterwards, cell values are copied to the corresponding neighbour ghost cells if they exist, as depicted by the arrows in depth one in Figure 3. Depending on a further subdivision of the neighbour cells, different copying techniques with or without averaging are applied. Having the surrounded cells as well as mean valued cells for the next step, a new calculation using only local values can be performed.

Thus, the main time stepping algorithm can be divided into two parts: one purely local part where only computation is taking place and one global part where communication is involved. Having built up the simulation in such a way, a parallelisation can be done without big changes.

III. Engineering Simulations

In the following section, the above mentioned basics will be applied to typical engineering simulations, namely temperature diffusion equation (III.A) for solving transient temperature distribution problems and Navier-Stokes equations (III.B) for solving computational fluid dynamics problems.

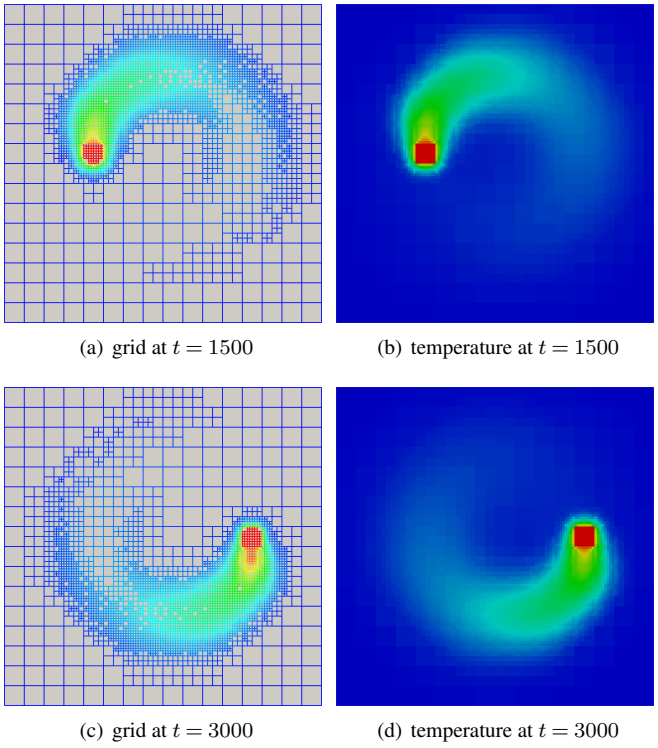
A. *Temperature Diffusion Equation*

One example of grid based engineering simulations is the time-dependent temperature diffusion equation

$$\frac{\partial}{\partial t} T = \alpha \cdot \Delta T \tag{1}$$

where T represents the temperature, depending on the time t and the spatial location, α the thermal diffusivity in [m^2/s] and Δ denoting the Laplace operator. If only a stationary solution is required, equation (1) reduces to the Laplace equation $\Delta T = 0$.

As numerical discretisation of equation (1), a forward Euler scheme in time and a central difference scheme in space is used, corresponding to a FTCS scheme.



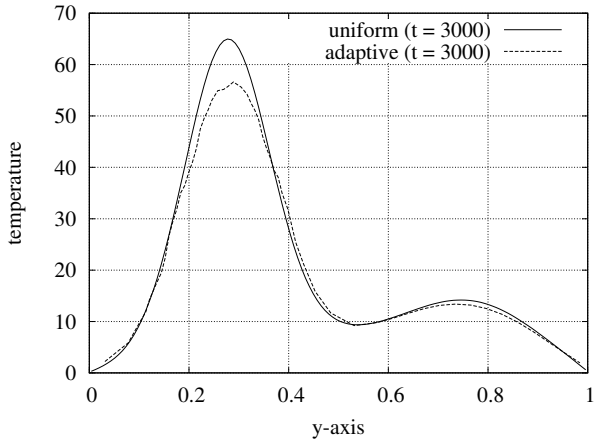
(a) grid at $t = 1500$ (b) temperature at $t = 1500$

(c) grid at $t = 3000$ (d) temperature at $t = 3000$

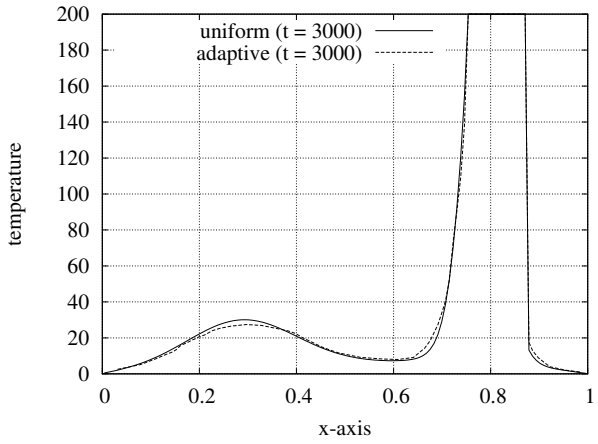
Figure 4. Computation of the time-dependent temperature diffusion equation (1) with adaptive grid refinement and time-dependent boundary conditions.

Example

An example of an adaptive computation can be seen in Figure 4, where the time-dependent temperature equation (1) is solved on a rectangular domain of 16x16x1. In this case, the computational domain is embedded in z-direction between two plates with Dirichlet boundary conditions $T = 0$. This setting was explicitly chosen over a setup with periodic boundary conditions in z-direction, as more energy is dissipated from the system and



(a) Temperature along a vertical line through the geometric centre of cavity



(b) Temperature along a horizontal line through the geometric centre of cavity

Figure 5. Comparison of temperature simulation results from a uniform computation (full lines) with results from an adaptive computation (dashed lines) at the time step $t = 3000$ of Figure 4(c) and Figure 4(d).

the adaptive coarsening can be better observed. For the sake of simplicity, only x and y coordinates are mentioned further on, even if a 3D computation was performed. As material, steel with a thermal diffusivity $\alpha = 1.172 \cdot 10^{-5}$ m^2/s was chosen.

The boundary cells of the domain have uniform temperature boundary conditions set to $T = 0$. Only a given obstacle of the size of one cell on the top level has a Dirichlet temperature boundary condition of $T = 200$. The boundary condition of the 'hot' cell is updated every time step t according to the relation $i_{\text{hot}} = 8 + 5 \cdot \lfloor \cos(2\pi t/1000) \rfloor$, $j_{\text{hot}} = 8 + 5 \cdot \lfloor \sin(2\pi t/1000) \rfloor$ and for all indices k_{hot}, which means that the 'hot' cell rotates counter-clockwise in the domain and has a period of $t_{\text{period}} = 1000$ s. If according to the index $(i_{\text{hot}}, j_{\text{hot}}, k_{\text{hot}})$ a new cell is selected, all other cells are set to 'free flow' meaning that the fixed Dirichlet boundary is removed, and the temperature of the cell can change again according to the values computed by the central difference stencil.

If a higher accuracy than 16x16x1 cells is desired, there are generally two possibilities. The easiest way is to increase the domain size uniformly to 128x128x1 e. g. As consequence, the domain consists now of 16,384 cells but no big changes to the code have to be made as only a different grid size has to be used.

A second, more complex way is an adaptive refinement as discussed in section II. Here only cells of interest get a higher resolution. In this example, cells from the top-level are refined by sub-grids of 2x2x1. This value is chosen for better visualisation results. From the point of view of computational overhead regarding grid management, it would be better to choose a higher value of cells per sub-grid.

According to the computation of the maximal and minimal temperature gradient between neighbouring cells, adaptive refinement or coarsening is applied to give a high accuracy regarding computational results using minimal cell amounts which reduces computational time. The example shown in Figure 4 uses three sub-levels of grids 2x2x1 with an average total amount of cells of 3,400. This is around 4.8 times less cell usage than in the uniform case at the same level of accuracy. Unfortunately the grid management is also more complicated and some averaging of values are applied in regions of coarsening.

Figure 5 shows a qualitative comparison between uniform and adaptive computation methods in terms of accuracy. The maximal temperature error from the adaptive to the uniform computation method in Figure 5(a) is around 12.5% and in Figure 5(b) about 8.0%. However, the adaptive version is approximately 1.5 times faster than the uniform computation.

These values can be even sped up by using a numerical more reasonable block size. As stated before, this example used a main grid size of 16x16x1 and the sub-grid size was chosen to 2x2x1 with three subdivision steps. When a sub-grid size of 4x4x1 is used with two subdivisions, compared to a uniform computation using 256x256x1 cells, the computation of the adaptive grid is 4.3 times faster than the uniform grid using 7.1 times less cells with a maximal error in temperature under 10%. Choosing a higher sub-grid size will be even more reasonable and give better results.

B. Navier-Stokes Equations

As a second example for grid based engineering simulations, an incompressible, isothermal Newtonian fluid flow without any acting external forces is simulated using the Navier-Stokes equations:

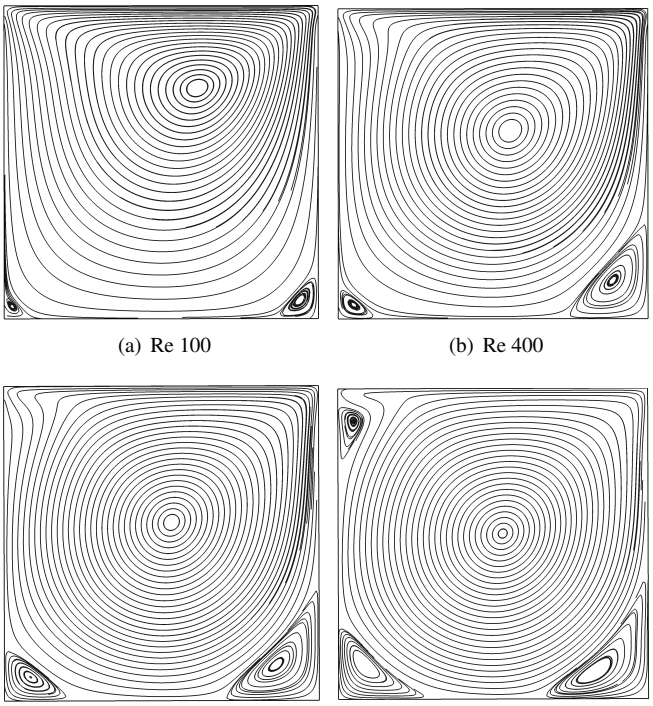
$$\vec{\nabla} \cdot \vec{u} = 0 \quad , \tag{2}$$

$$\frac{\partial}{\partial t}\vec{u} + \left(\vec{u} \cdot \vec{\nabla}\right)\vec{u} = -\frac{1}{\rho}\vec{\nabla}p + \nu\Delta\vec{u} \quad . \tag{3}$$

where $\vec{u}$ and p are the unknown velocities and pressure, t represents the time, ρ the density and ν the viscosity of the fluid. Further detailed information might be found in Hirsch [5] or Ferziger and Peric [6].

B.1 Numerical Discretisation Schemes

In the example at hand, a finite volume scheme is used for spatial discretisations and a finite difference scheme for temporal discretisations. For the sake of simplicity, the first tests are performed using an explicit Euler scheme for the temporal discretisations in order to test the above described adaptive block-oriented data structure. In a later stage it is planned to adopt a semi-implicit temporal discretisation. Furthermore, a fractional step or projection method is applied for solving the time-dependent incompressible flow equations. This method is based on a iterative procedure between velocity and pressure during one time step.



(a) Re 100 (b) Re 400

(c) Re 1000 (d) Re 3200

Figure 6. Streamline patterns in the lid-driven cavity with a grid resolution of 101x101 for different Reynolds numbers.

Omitting the pressure term, the momentum equations are solved for intermediate velocities $\vec{u}^*$:

$$\frac{\vec{u}^* - \vec{u}^n}{\Delta t} = -\left(\vec{u}^n \cdot \vec{\nabla}\right)\vec{u}^n + \nu\Delta\vec{u}^n \quad . \tag{4}$$

The superscript * denotes intermediate values and the superscript ^{n} values at time step n, which are fully known. In the second step, the pressure term at the next time step $n+1$ is used to correct the resultant intermediate velocity field leading to the velocity field at the new time step $n+1$:

$$\frac{\vec{u}^{n+1} - \vec{u}^*}{\Delta t} = -\frac{1}{\rho}\vec{\nabla}p^{n+1} \quad . \tag{5}$$

The divergence free velocity field at step $n+1$ can be guaranteed by computing the divergence of (5) and applying the continuity equation (2):

$$\frac{\rho}{\Delta t}\left(\vec{\nabla} \cdot \vec{u}^{n+1} - \vec{\nabla} \cdot \vec{u}^*\right) = -\Delta p^{n+1} \tag{6}$$

$$\Delta p^{n+1} = \frac{\rho}{\Delta t}\vec{\nabla} \cdot \vec{u}^* \quad . \tag{7}$$

Equation (7) represents a Poisson equation for the pressure, which has to be solved to compute the velocity field for the next time step, using (5).

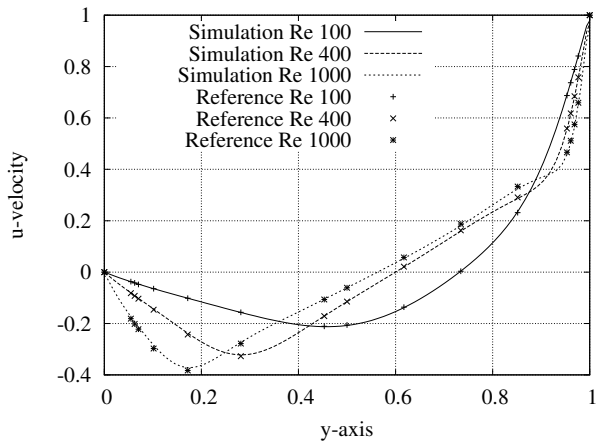
B.2 Staggered versus Collocated Grid Arrangements

While applying a spatial discretisation scheme, it is possible to choose between different settings. In a staggered grid

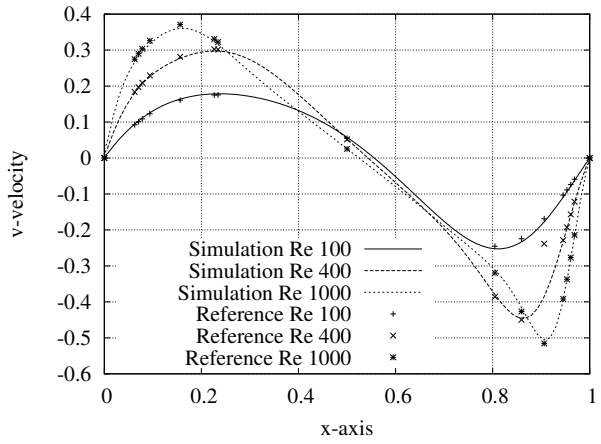
approach, not all variables are represented at the same point in space. Usually partially staggered arrangements are used, where pressure and other scalar terms are situated at the cell centre, whereas the velocities are positioned at the respective cell surfaces. This arrangement has the major advantage that the velocities are strongly coupled to the pressure values and no oscillations occur applying the projection method. Disadvantages are the more complicated handling in case of non-orthogonal grids or when applying multigrid solvers.

In collocated grids, all variables are defined in the cell centre which simplify the usage of non-orthogonal grids or advanced multigrid solvers. On the other hand, it can be shown that the solution of equation (7) leads to a so called odd-even decoupling which introduces non physical pressure oscillations if no special care is taken. More details can be found in Hirsch [5] or Ferziger and Peric [6].

The code described here uses a collocated grid arrangement due to the fact that in later phases of this project the authors plan to use sophisticated numerical solvers such as multigrid methods.



(a) u-velocities along a vertical line through the geometric centre of cavity



(b) v-velocities along a horizontal line through the geometric centre of cavity

Figure 7. Comparison of simulation results (lines) with results of Ghia et al. (points) for the lid-driven cavity

B.3 Validation Using the Lid-Driven Cavity Example

For the validation of the above described code, the lid-driven cavity example is used. This example consists of a square domain of unit length where the upper boundary wall moves with constant velocity $u = 1$. Thus, only shear driven forces from the no-slip boundaries are transferred to the initially resting fluid. Reference solutions for comparison where taken from Ghia et al. [7].

Figure 6 shows a streamline plot for different Reynolds numbers of $Re = 100$, $Re = 400$, $Re = 1000$, and $Re = 3200$ computed using the above mentioned code and a grid spacing of 101x101. All the validations and computations were done in a first step only in two dimensions, even if the data structure is designed for three dimensions.

For getting a better view of the numerical errors introduced by the discretisation technique e. g., detailed comparisons were made in Figure 7(a) and 7(b). It can be seen, that for Reynolds number $Re = 100$ the computed values match the reference values used by Ghia et al. quite well. But the higher the Reynolds number is, the higher the divergence between the computed values and the reference values gets, even if the characteristic behaviour can still be observed.

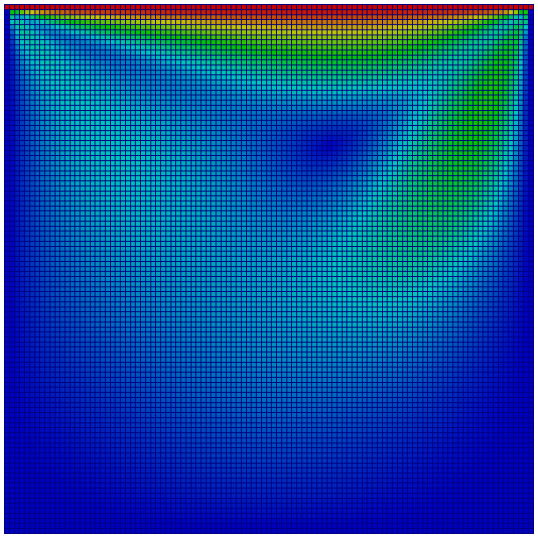
Computations with a higher geometric resolution and a smaller time step size for different Reynolds numbers show that a finer time step has a much higher impact as soon as the spatial discretisation is reasonably small. This is a numerical artefact of using the explicit Euler time scheme for temporal discretisation and shows that for simple tests of the data structure, the explicit scheme is adequate, but for later real case studies, a higher temporal discretisation technique has to be used.

Figure 8 shows the magnitude of the velocity vector $\vec{u}$ for an uniform and an adaptive computation of the lid-driven cavity example at $Re = 100$. The base grid is chosen to 21x21 and the sub-grid size to 5x5 for display reasons and the time step is set to 10^{-4} s.

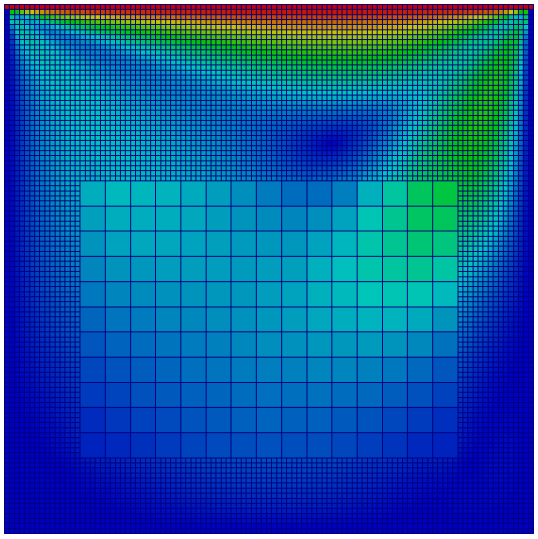
The accuracy of the computation is indicated in Figure 9. It can be seen, that the coarse grid of 21x21 is not reaching the reference values of Ghia et al., as the grid is too coarse to deliver accurate results. An adaptive mesh refinement as depicted in Figure 8(b) results in much better accuracy, even if the reference values are not quite reached. This is due to the interpolation effect of the grid changes from coarse to fine. In order to keep the computation algorithms as simple as possible, some trade-off was accepted and a numerical error was introduced. At the moment, the authors are working on reducing the numerical error while still keeping a simple scheme regarding numerical computation and data exchange from the different grid levels.

First parallel computations were done using a shared memory OpenMP concept. In this first implementation, only computational intensive nested loops of the Navier-Stokes equations were parallelised. Hence the update step mentioned in II.A is still running as serial procedure and is dominating the possible speedup as well as the parallel efficiency.

Computational results of the parallel speedup and efficiency are depicted in Figure 10. In order to get a better comparison, three different architectures and different grid sizes were used. The used architectures include an Intel Core 2 Quad Q9650 (3.00 GHz), an Intel Core i7 870 (2.93 GHz), and an Intel Xeon

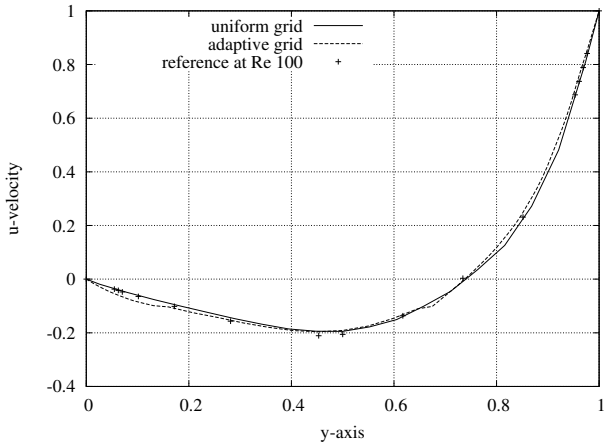


(a) uniform grid

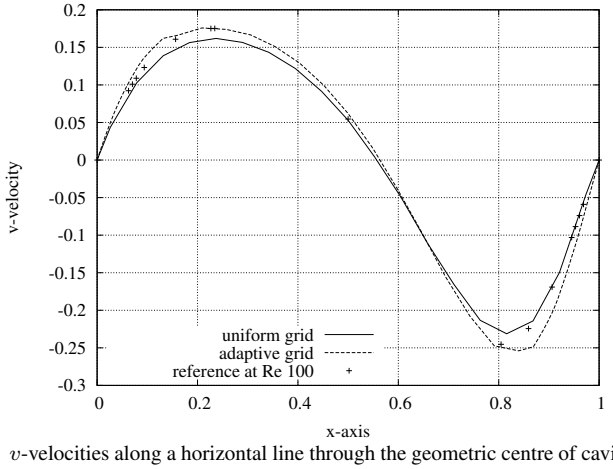


(b) adaptive grid

Figure 8. Magnitude of the velocity vector $\vec{u}$ of an example computation of the lid-driven cavity on a uniform grid using 105x105 cells with a time step of $\Delta t = 10^{-4}$ s and an adaptive computation using a base grid of 21x21 and sub-grids of size 5x5.



(a) u-velocities along a vertical line through the geometric centre of cavity



(b) v-velocities along a horizontal line through the geometric centre of cavity

Figure 9. Plots comparing velocities with the reference solution of Ghia et al. with an adaptive computation on a 21x21 coarse grid only (marked uniform grid in plots) and an adaptive computation depicted in Figure 8(b) (marked adaptive grid in plots)

E3-1245 (3.30 GHz). Furthermore the same optimisation flags were used for the Intel compiler on all architectures.

Figure 10 shows that this kind of parallelisation is not optimal as the efficiency is dropping quite fast as soon as more processes are used. Hence, another method has to be deployed when more cores or processes are involved, for which the data structure was designed to distribute the sub-grids to different processes using a message passing concept. This parallelisation will be subject to further investigations.

IV. Outlook to planned work for the future

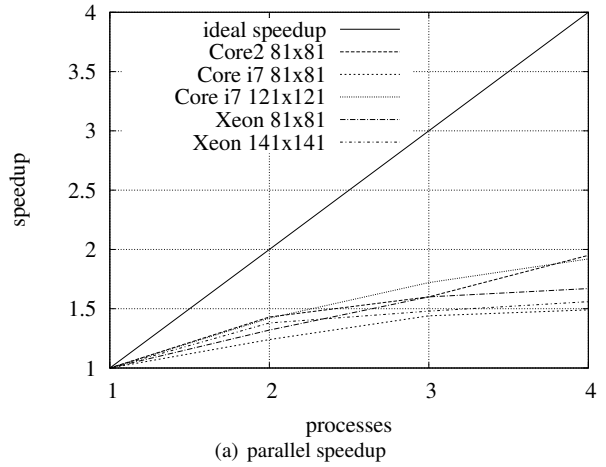
As this paper describes work in progress, the numerical error using an adaptive grid discretisation scheme has still higher errors than expected. The next steps will accordingly be, to improve the numerical scheme for the distribution of the values from one grid part to the other, especially in between coarse and fine cells.

As mentioned before, a higher order temporal discretisation scheme has to be implemented in order to increase the time step size and still get reasonable results.

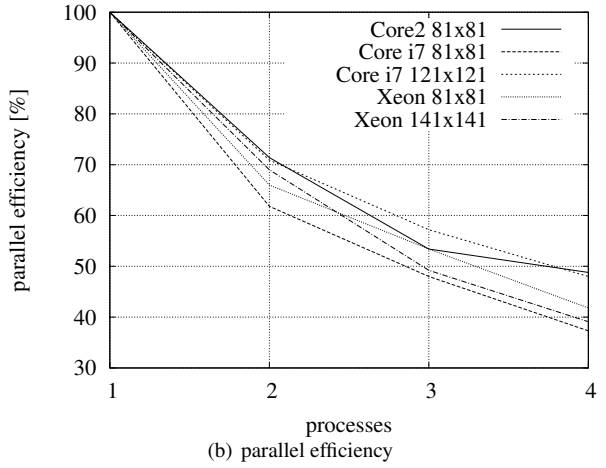
A next step is to exploit the special design of the code in order to implement a parallel concept. As mentioned in section II, the local computations on the grid can be executed in parallel while the communication step needs global access and, thus, synchronisation between the sub-grids is necessary. A good distribution of sub-grids to different processes depending on the communication layout has to be chosen to ensure minimal communication effort. One master process should not be handling all the communications but delegate them to separate handlers who organise communication between the working nodes to ensure an excellent load balancing and efficient communication patterns as depicted by Mundani et al. [8].

V. Conclusion

In this paper, we have presented an adaptive data structure management for the simulation of engineering problems such as the temperature diffusion equation or the Navier-Stokes equa-



(a) parallel speedup



(b) parallel efficiency

Figure 10. parallel speedup and efficiency computed on shared memory machines using different architectures and different grid sizes

tions. Example applications were computed as far as the presented code is implemented at the moment. As soon as the tasks described in section IV have been finished, next steps will comprise the improvement of the numerical algorithms. The adaptive implementation of the finite difference grid as well as the finite volume grid have shown promising results and the authors look forward to further increase efficiency and handle real world problems rather than test case scenarios.

The ultimate goal is to compute an adaptive fluid-flow simulation around the power plant model introduced in the motivation part in order to compare the results in terms of accuracy and performance between a parallel adaptive computation versus a pure uniform one.

VI. Acknowledgment

This publication is based on work supported by Award No. UK-c0020, made by King Abdullah University of Science and Technology (KAUST).

References

[1] University of North Carolina at Chapel Hill, *Power Plant Model*, 2001.

[2] H. Samet, "The quadtree and related hierarchical data structures," *ACM Comput. Surv.*, vol. 16, pp. 187–260, June 1984.

[3] G. Barequet, B. Chazelle, L. J. Guibas, J. S.B. Mitchell, and A. Tal, "Boxtree: A hierarchical representation for surfaces in 3d," *Computer Graphics Forum*, vol. 15, no. 3, pp. 387–396, 1996.

[4] P. Coelho, J. C. F. Pereira, and M. G. Carvalho, "Calculation of laminar recirculating flows using a local non-staggered grid refinement system," *Int. J. Numer. Meth. Fluids*, vol. 12, no. 6, pp. 535–557, 1991.

[5] C. Hirsch, *Numerical Computation of Internal and External Flows, Volume 1*, Butterworth–Heinemann, 2nd edition edition, 2007.

[6] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*, Springer, 3rd, rev. ed edition, 2002.

[7] U. Ghia, K. N. Ghia, and C. T. Shin, "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method," *Journal of Computational Physics*, vol. 48, no. 3, pp. 387 – 411, 1982.

[8] R.-P. Mundani, A. Düster, J. Knezevic, A. Niggl, and E. Rank, "Dynamic load balancing strategies for hierarchical p-FEM solvers," in *Proc. of the 16th EuroPVM/MPI Conference*. 2009, Springer.