Parallel implicit matrix-free CFD solver using AMR grids

Pavel Pavlukhin and Igor Menshov
Keldysh Institute for Applied Mathematics RAS, Moscow 125047, Russia
E-mail: pavelpavlukhin@kiam.ru

Abstract. A novel parallel algorithm for the LU-SGS method with adaptive mesh refinement (AMR) is proposed. Domain decomposition and dynamic load balancing algorithms for spatial discretizations with AMR are described. For improving execution efficiency on targeted GPU-accelerated systems, corresponding coarsening/refining and memory defragmentation parallel algorithms are developed.

1. Introduction
The Free Boundary Method ([1]) which we develop as extension to the Navier-Stokes equations requires high mesh resolution for problems with complex geometry. In case of Cartesian grids it leads to unnecessarily huge increase of computational costs and memory consumption whereas high resolution is required only near solid surfaces and high-gradient regions. Therefore, one needs to exploit a kind of adaptive local mesh refinement (AMR) approach. The most suitable and robust approach for our methods is the AMR based on octree graph representation of three-dimensional Cartesian grids with recursive multi-level cell subdividing. One of our main goals is to implement a highly-scalable framework for solving CFD problems on HPC clusters with massively-parallel accelerators (such GPU and Xeon Phi). Thus, we deal with domain decomposition and dynamic load balancing challenges. Initially the computational domain is discretized with a coarse Cartesian grid referred to as base grid, and then necessary refinements are performed so each base grid cell becomes the root of corresponding octree for its subcells. The Z-like space filling curve (SFC) over all octrees is used for domain decomposition on distributed-memory systems with the only restriction: any octree is fully located in one process, i.e., splitting of octree over processes is forbidden. It considerably decreases complexity of mesh and octree inter-process management with just small limitation on dynamic load balancing: minimal portion for exchange between processes is a base grid cell with whole its octree. The matrix-free LU-SGS method [2] used in the iterative solver for discrete equations of implicit scheme leads to the graph coloring problem in the case of parallel implementation for one. If all grid cells are colored such that any two neighboring (by face) cells are different in color, then all cells of the same color can be processed simultaneously in parallel keeping full correctness of the LU-SGS method. In the case of Cartesian grids, only two colors with “chess”-like pattern are needed to implement highly efficient CFD solver with CUDA and MPI [3]. In the case of three-dimensional 2:1 balanced octree grids one needs four colors, e.g., the coloring algorithm presented in [6]. One its important property is simple local recoloring in case of the mesh coarsening and refinement that permits to effectively solve problems with implicit time steps
even in case of dynamically modified grids strictly conforming to LU-SGS method over whole distributed mesh. Applying this coloring scheme, we achieve good scalability with thread- and process-level parallelism. Our approach to AMR handling on octree grids is similar to that used in the library p4est [5]. However, this library lacks of thread-level parallelism (MPI-only) and doesn’t provide API for low-level data manipulation which is necessary for implementing such type of parallelism and graph coloring algorithms. In the present paper we deal with these issue. Following sections describe in details proposed algorithms and approaches.

2. Numerical methods and grid types.
Consider gas dynamics equations solved with the Free Boundary Method (FBM):

\[
\frac{\partial \mathbf{q}}{\partial t} + \frac{\partial \mathbf{f}_i}{\partial x_i} = -\mathbf{F}_w .
\]  

(1)

The only difference from its conventional form is the right-hand side additive called compensation flux (see details in [1]). By the finite volume method and implicit time integration scheme (1) leads to the system of linear equations which is solved by Newton’s iterative method with correspond sparse block matrix \(A\):

\[
A\delta \mathbf{q} = \mathbf{R},
\]  

(2)

where \(\delta \mathbf{q}\) is iterative residual. This system is then solved by the matrix-free Lower-Upper Symmetric Gauss-Seidel (LU-SGS) approximate factorization method [2]. By this method \(A\) is replaced by approximate factorization \(A \approx (D + L)D^{-1}(D + U)\) containing diagonal \((D)\), low-\((L)\) and upper-triangle \((U)\) parts of \(A\) that finally leads to solving the following systems (by corresponding forward and backward sweeps):

\[
\begin{align*}
(D + L)\delta \mathbf{q}_i^n & = -\mathbf{R}_i^{n+1} \\
(D + U)\delta \mathbf{q}_i^n & = D\delta \mathbf{q}_i^n .
\end{align*}
\]  

(3)

Using this FBM method for solving of 3-dimensional problems with complex (possibly moving) body geometries requires high resolution grids. In case of Cartesian grids it leads to unnecessarily huge increasing of computational costs and memory consumption whereas high resolution is actually required only near solid surfaces and for high-gradient flow regions. So one needs to exploit some kind of adaptive mesh refinement (AMR) approach. As FBM requires rectangular grid cells the most suitable and robust approach for one is AMR based on octree graph representation of 3-dimensional Cartesian grids with recursive multi-level cell subdividing (Fig. 1(a)): each rectangular cell of initial Cartesian grid can be recursively divided in 8 subcells (4 subcells in 2-dimensional case). Such constructing grid can be defined as the octree (quadtree) graph (each internal node in such tree has exactly eight children). With the Z-order space-filling curve (SFC) connecting nodes in this graph, the data on such adapted grid can be arranged in line which enables to perform simple partitioning and dynamic load balancing for parallel multi-process computations.

One of the major octree properties is k-to-1 balance. This property states that each grid cell has no more than \(k\) neighbors over any its face (Fig. 1(b)). 2-to-1 balanced grids simplify cell-to-cell solver interface and allow to avoid interpolation issues so such type of octree grids is the most suitable for the FBM.

3. Parallel algorithm on 2-to-1 balanced octree grids.
Our main goal is to implement highly-scalable framework based on the FBM for solving CFD problems on HPC clusters. The LU-SGS is the core solver method and its parallelization
Figure 1. (a) Octree-based grid (left) and corresponding graph representation with space-filling curve (right); (b) 2-to-1 balance status for a 2D quadtree mesh: unbalanced (left) and face balanced (right).

Figure 2. Parallel algorithm for LU-SGS: cells treating (marked by bold borders) and subsequent copying (marked by dashed arrows) between processes in coloring order (3 stages): red (left), black (center), white (right). "Ghost" cells are marked by dashed borders.

determines overall framework scalability. It’s quite easy to show that parallel algorithm construction for the LU-SGS method leads to graph coloring problem: one need to color the grid cells so that any two neighbor grid cells (with the common face) would have different colors [3]. In case of Cartesian grids the solution is simple chessboard coloring with only 2 colors. Obviously 2 colors for quadtree/octree grids aren’t already enough. For this case solution is found in [6]: 2-to-1 balanced octree requires 4 different colors for grid coloring. Color of each cell in octree is defined by only its node position in graph. For the sake of simplicity further algorithm description is provided for quadtrees (which require only 3 colors), additional explanations for octree happen if needed.

So by applying coloring scheme from [7] for 2-to-1 balanced quadtree (with 3 colors referred as c1, c2, c3) we obtain the following parallel algorithm: one performs computations over all cells with c1 color in parallel; wait until all c1 cells have been treated; next all c2 cells are swept in parallel; wait until all c2 cells have been treated; finally, all c3 cells are swept in parallel.

This algorithm is for the upper subsystem in (3), for the lower subsystem cells are treated the same way but in reverse color order (c3, c2 and then c1). All described steps are straightforwardly implemented in shared memory programming models like CUDA or OpenMP. Distributed memory models (multiple parallel running processes without shared global address space) like MPI require some modifications in proposed algorithm. Each process is assigned some grid subregion i.e. domain decomposition is performed for whole initial AMR grid. Border subregion interface consists of so called "ghost" cells which correspond to ones in neighboring domains (Fig. 2). After completing of computations on c color cells (c = c1, c2, c3) borders exchange (consisting of c color cells) is performed with necessary synchronizations thus updating "ghost" sets.

As one can see border cell transferring is the only addition to initial parallel algorithm so implementation based on MPI is also quite straightforward.
4. Domain decomposition and dynamic load balancing

Although proposed parallel algorithm for LU-SGS method seems pretty easy there are serious challenges associated with one. Some are domain decomposition and dynamic load balancing problems. For performance reasons not only grid data itself but corresponding quadtree metadata structures need to be decomposed and distributed. If we permit a quadtree splitting to two or more processes it leads to complicated scheme describing connections between distributed parts of single tree graph which must be updated during dynamic load balancing procedure. So we restrict each quadtree to be entirely in one process.

Initially Cartesian grid which we call base one is generated in whole domain (its cells referred later as macrocells). Then AMR procedure is performed in such a manner that some macrocells become root node for corresponding quadtree (Fig. 3) keeping 2-to-1 balance over all cells. Direct using of Z-order SFC for domain decomposition restricts base grid resolution to be power of two in each direction and this is unacceptable limitation. In case of arbitrary resolutions one needs to develop domain decomposition algorithm. The ones based on slicing lead to unbalances in partitioning and/or huge amount of border cells need to be transferred between processes so we just exploit Z-order SFC by virtually extending to power of two base grid resolution in each direction (Fig. 3). Each macrocell is assigned weight value equal to total number of its subcells. After that domain decomposition and dynamic load balancing are performed at base grid level the same way as with ordinary Z-order SFC adjusted for weights i.e. minimal relocatable part between processes is macrocell with whole its quadtree.

Although balancing is performed at quite coarse (macrocell) level proposed approach considerably decreases complexity of mesh and octrees inter-process management with small limitation on dynamic load balancing.

5. Parallel grid modification algorithms

Our framework is aimed to HPC clusters equipped with massively parallel accelerators like GPU. During task processing one needs to perform coarsening/refining procedures on grid cells which require quadtree metadata arrays modification in addition to applied data array (state vectors) update. Typical approach is to perform these procedures on CPU in single-thread mode whereas GPU executes only solver kernels. This leads to GPU stalls and huge data synchronization between CPU and GPU via multiple pci-e transfers what finally results in poor resource utilisation. In contrast, our approach includes porting of all quadtree procedures to GPU thus CPU is used only for inter-process data transfers. To the authors’ knowledge there is still no such attempts: any existing quadtree/octree based software exploits GPU at best for constructing such graphs but not for them subsequent transformations. So computational model is the following: after initial load balancing procedure described above all grid data is copied to GPU memory; CPU-GPU transfers are performed only for ghost/border cells which are needed to exchange between processes according to the proposed parallel algorithm for the LU-SGS method; all solver and grid manipulation procedures are executed entirely in GPU.

First challenges in porting quadtree processing on GPU are grid refinement and coarsening:
depending on intermediate solution analysis one needs to combine or subdivide some quadtree cells (Fig. 4). For this problems three arrays are allocated in GPU memory: base grid macrocells ordered by SFC (denoted as “□”), anchored/root nodes for all macrocells (denoted as “●”) and (denoted as “○”) dangling nodes (which are real grid cells) with state vectors (solver actually exploits only this array); all necessary node pointers are also initialized. Each thread in parallel performs solution analysis for coarsening/refinement on four/one dangling nodes (grid cells). If cell(s) is(are) needed to be coarsed/refined its(them) location(s) in array of “○” is(are) marked as empty and new corresponding cell(s) is(are) written to free space in the end of this array by each thread. Since some writings may occur simultaneously one uses global mutex for its separation. The same modification procedure is performed on array of “●”. In the final coarsening/refinement stage all necessary node pointers are updated fully independently in parallel.

It should be noted that after coarsening/refinement steps grid cells topology has been changed so one needs to perform grid recoloring for keeping correctness of the parallel LU-SGS algorithm. According to [coloring octrees] this procedure is in fully coherence with grid local adaptation steps: there is no need for global interprocess recoloring (in contrast to another coloring algorithms on unstructured grids), one is performed locally and independently only in having changed cells i.e. right in grid coarsening/refining procedure. It’s because cell color is determined only by its position in quadtree/octree (assuming 2-to-1 balance keeping).

Above-noted recoloring properties are crucial for efficient and scalable GPU implementation. As one can see coarsening/refinement procedures lead to holes (denoted as “×” on Fig. 4) inception in nodes arrays. Multiple their callings result in poor efficiency (less opportunities for coalescing load/store transactions from/to GPU RAM) and exhaustion of available memory so some sort of array defragmentation must be performed. In our approach this operation is also executed in GPU entirely. Each thread is assigned one macrocell and traversing over correspond quadtree based on local Z-space SFC is performed (Fig. 5). During this process new
local array consisting of pointers to dangling nodes in SFC order is formed. Then all these local arrays are merged in a global array with necessary synchronization between threads via global mutex. Finally each thread moves dangling node taken by pointer in global array to new position determined by index in this array. This algorithm not only compacts arrays but also relocates ones improving in some degree (as Z-space SFC permits) cell (and memory transactions) locality according to its neighboring.

Using global mutexes, of course, decreases efficiency of proposed algorithms but last ones after all are more preferable comparing with CPU-only grid modifications and corresponding back-and-forth GPU→CPU data transfers.

6. Conclusions
Our framework is based on Free Boundary Method. For exploiting resources of supercomputers equipped accelerators like GPU or Xeon Phi we proposed parallel algorithm for the LU-SGS method used as iterative solver in FBM on octree-based grid. During computations grid needs to be transformed via coarsening/refinement procedures. For efficient execution of last ones corresponding parallel algorithms were developed. Implementation of all proposed algorithms is in progress.

Acknowledgments
This research was supported by the Grant No 17-71-30014 from the Russian Science Foundation.

7. References
[1] Menshov I S and Kornev M A 2014 Free-Boundary Method for the Numerical Solution of Gas-Dynamic Equations in Domains with Varying Geometry Mathematical Models and Computer Simulations 6 612
[2] Jameson A and Turkel E 1981 Implicit schemes and LU decomposition Math. of Comp. 37 385
[3] Pavlukhin P and Menshov I 2015 On implementation high-scalable CFD solvers for hybrid clusters with massively-parallel architectures. In: Parallel Computing Technologies: 13th International Conference, PaCT 2015, Petrozavodsk, Proceedings
[4] Adamy U, Hoffmann M, Solymosi J and Stojakovic M 2004 Coloring Octrees. In: Chwa KY., Munro J.I.J. (eds) Computing and Combinatorics. COCOON 2004. Lecture Notes in Computer Science, vol 3106. Springer, Berlin, Heidelberg
[5] Burstedde C, Wilcox L C and Ghattas O 2011 p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees SIAM Journal on Scientific Computing 33 no. 3 (2011), pages 1103-1133
[6] Adamy U, Hoffmann M, Solymosi J and Stojakovic M 2004 Coloring Octrees. In: Chwa KY., Munro J.I.J. (eds) Computing and Combinatorics. COCOON 2004. Lecture Notes in Computer Science, vol 3106. Springer, Berlin, Heidelberg
[7] Pavlukhin P and Menshov I 2017 Parallel Algorithms for an Implicit CFD Solver on Tree-Based Grids In: Parallel Computing Technologies: 14th International Conference, PaCT 2017, Nijniy Novgorod, Proceedings