Parallel implementation of the CCBA for mesh generation

E.D. Batista, J.S. Otto and J.E. Castillo*
Computational Science Research Center, SDSU
* dbatista@sciences.sdsu.edu

Abstract

In this work we extend the CCBA to produce a parallel-iteratively mesh generator. The CCBA or Convex-Combination-Based-Algorithm (presented in ACSESS 2007) uses a mesh-size function and convex combinations on local polytopes to change the position of the nodes of an initial mesh. This process is repeated iteratively and stops when the max norm between two consecutives meshes is smaller than a given threshold. The parallelization is obtained by dividing the initial mesh into as many submeshes as processors available and dealing with each submesh in a different processor. This so-called mesh decomposition does not depend on the physical domain of the problem, which is the main difference and the main advantage over domain decomposition-like techniques for parallelization. We also present a way to reduce overhead while preserving the property of non-folding of the original sequential algorithm. Several examples of 2D meshes on complex domains show the potential of the technique for constructing non-uniform quad meshes.

CCBA

We start with an initial grid constructed over the domain, Ω. Then, for each point, p, of the mesh we consider a convex polytope $V \subset \Omega$ that contains p. If V is generated from points v_j then, p is updated according to

$$\hat{p} = \sum_{j=1}^{n} \lambda_j(p) v_j \qquad (1)$$

where λ_j are estimated with the mesh-size function f.

Since (1) defines a local convex combination mapping that maps V into a convex closed region, then (1) is one-to-one, and $\hat{p} \in V$. Moreover, $\hat{p} \notin \partial V$ because $f > 0$ and, in turn, $\lambda_j > 0$. This property guarantees that non-folding will be produced.

The mesh-size function

The mesh-size function, f, is a positive function defined on Ω. Hence, f is independent of the numbers of points of the grid, which provides simplicity and versatility to the model. For regions of high density, the mesh-size function f has small values, whereas for regions of small density, f has bigger values. The mesh-size function can be obtained through different ways: given *a priori*, based on the geometry of Ω, based on errors from differential equations or combination of these, for mention just a few.

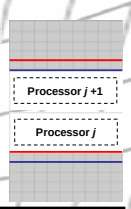
Example 1 shows a mesh-size function, f_1, obtained from

$$f_1(x,y) = \min\{g_1(x,y), g_2(x,y), g_3(x,y)\}$$

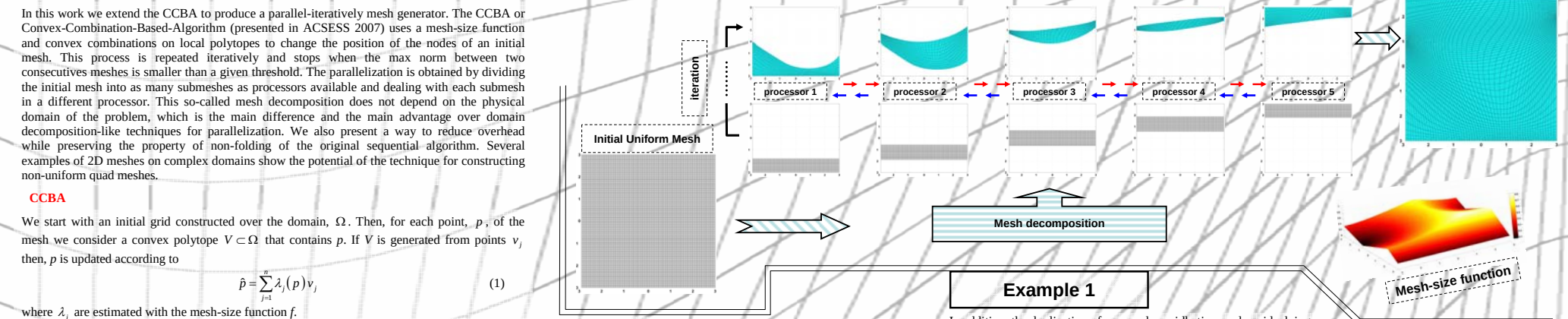
where

$$g_1(x,y) = .3\times|y+3|+.2$$
$$g_2(x,y) = .2\times\sqrt{(x-3)^2+(y-2)^2}+.2$$
$$g_3(x,y) = .15\times\sqrt{(x+3)^2+(y-1)^2}+.07$$

Example 2 shows a mesh-size function, f_2, obtained by implementing an inverse distant weighting algorithm with R-functions. An R-function works as a Boolean switching function and can be combined in a fully automatic way, to obtain an implicit representation of the domain.



Replicated rows
Red line only changes in processor $j+1$.
Blue line only changes in processor j.



Example 1

Parallelization

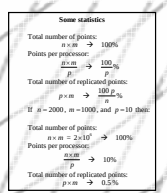
Because calculations in the convex-combination based algorithm are done locally, the CCBA can be easily parallelized. We work with a SIMD (Singular Instruction Multiple Data) problem and we consider the following information as input values:

P_t: total number of processors to be used,
n: rows of the grid,
m: columns of the grid.

P_t will be written as follow

$$P_t = P+1,$$

where P is the number of processors that will work modifying the mesh points, and the number 1 stands for the master processor.



The first we do is *mesh decomposition*, which consists of dividing the initial mesh in as many processors available are. Then, we distribute the P pieces of the initial mesh among P processors where they will be modified. During mesh decomposition we also replicate certain rows. The last two rows of processor j will be the first two rows of processor $j+1$, where $j = 1, \ldots, P-1$.

Each processor works independently and only interacts with other processors at the end of each iteration, to exchange the replicated rows. Processor j will update its last row with the second row of processor $j+1$; and processor $j+1$ will update its first row with the row prior to the last row of processor j. By proceeding this way, each processor interacts with its two closest neighbors (processors) and at each iteration no foldings are guaranteed.

Mesh decomposition

The idea is to divide the initial mesh into sub-meshes and assign each of these pieces to a processor. The size of each sub-mesh will be proportional to the computational power of the corresponding assigned processor. Thus, we are able to adapt the algorithm to heterogeneous clusters of computers.

Mesh decomposition has several advantages over mesh decomposition. Domain decomposition, as its name suggests, means dividing the given domain, Ω, into several pieces. This can be a very difficult task depending on the complexity of Ω. Mesh decomposition, on the other hand, is geometry independent. Since the initial mesh (and every mesh, actually) is stored in a matrix, dividing this mesh simply means selecting a number of rows and/or columns of the matrix. This is done automatically. For instance, example 2 presents a very complex non-convex domain. However, for this case mesh decomposition is as easy as it is for example 1, which has a very simple square-shape domain.

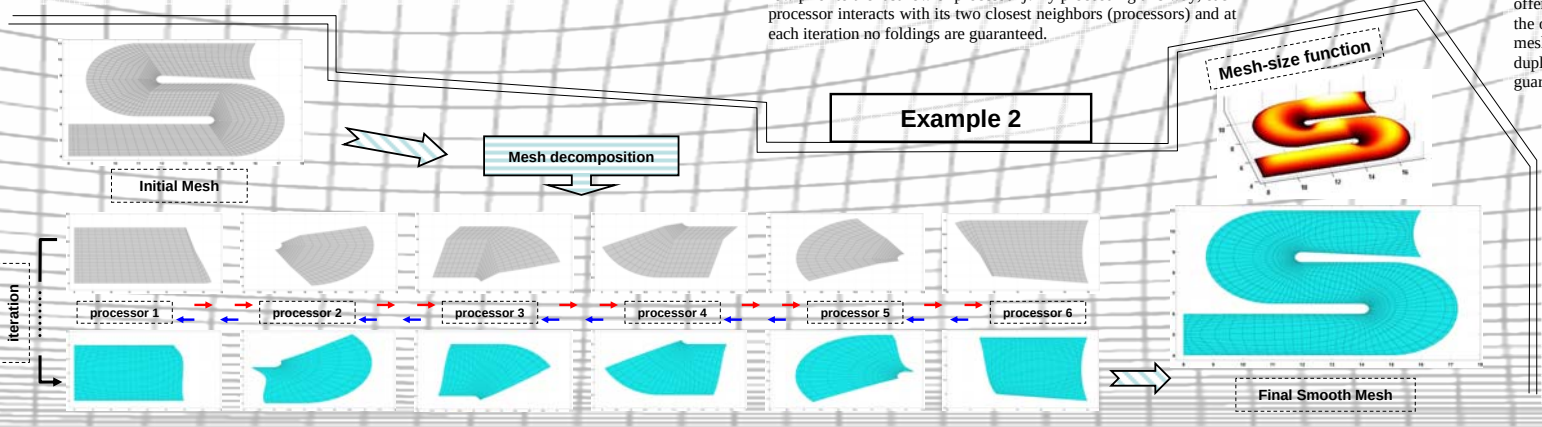
After doing domain decomposition each piece of the decomposed domain remains unchanged during the mesh generation process; local meshes are adapted to the sub-domains. In contrast, in the mesh decomposition case, each sub-mesh adapts itself to the domain. They can change independently, facilitating the refinement procedure. We can appreciate this in example 1. Each sub-mesh is a uniform rectangular-shape mesh but, at the end of the mesh generation process, each one presents a very different shape resulting from the mesh-size function used for mesh refinement.

In addition, the duplication of rows reduces idle time and avoids doing extra computations. It is worth to say that communication, idle time, and extra computation are the three main sources of overhead in parallel computation.

Conclusions and future works

The technique produces meshes of high quality that are smooth and can be adapted to different domains and different levels of refinement. The implementation of a mesh-size function offers flexibility to control the refinement of the mesh. The parallelization of the original CCBA can be done very easily by using a geometry-independent mesh decomposition technique, instead of using domain decomposition. The duplication of the boundaries of the sub-meshes, on the other hand, guarantees that unfolded meshes will be produced.

We expect to develop a deeper analysis of the speed up and scalability of this parallel mesh generator. Also, we will study the possibility of extending the algorithm for constructing 3D meshes.



References

- E.D. Batista, *A convex-combination based algorithm for grid generation*, poster presentation at the Applied Computational Science and Engineering Student Support (ACSESS), San Diego State University, USA, March 7, 2007.
- P.S. Pacheco, *Parallel programming with MPI*, Morgan Kaufmann Publishers, Inc. San Francisco, California, 1997.
- V.L. Rvachev, T.I. Sheiko, V. Shapiro, I. Tsukanov, *Transfinite*

Acknowledgements

Thank you very much Carny Cheng and Jennifer Winn for helping to translate the original MATLAB code to C.