PARALLEL OCTREE MESH GENERATION FOR IMMERSED BOUNDARY SIMULATIONS

JABER J. HASBESTAN∗, INANC SENOCAK†
Department of Mechanical Engineering and Materials Science
University of Pittsburgh, Pittsburgh, PA
e-mails: jaber@pitt.edu, senocak@pitt.edu

Key words: Octree Generation, Red-Black Tree, Binarized Octree, Parallel Computing

Abstract. We present a software for dynamic adaptive mesh refinement of Cartesian domains for complex geometry flow simulations. We introduce Z-curve compliant red-black tree and utilize it as the fundamental data structure for the algorithm. The generated tree is in compliance with 2:1 balance restriction to allow for smooth transitioning from high resolution to low resolution. Tree topology is combined with weighted Hilbert space filling curve (HSFC) to generate a balanced distribution of work among the processes. The MPI-3.0 non-blocking collectives are used to overlap the communication with computation and hence, hide the latency. Geometry encoding is introduced to improve the search algorithm. Some preliminary strong scaling is presented. Extreme Weak scaling is under study and will be available in the final paper. This package is developed in C++ using object-oriented programming.

1 INTRODUCTION

Geometry projected mesh generation is a time consuming task. Numerically speaking, the discretization (Grid) and the corresponding solution are inter dependent. In turbulent flows, one has to determine mesh spacing from the wall ($y^+$) before the simulation according to the Reynolds number. Therefore, at its core every mesh is specifically customized for a certain simulation. The final solution will also depend on the quality of the mesh. This becomes even more crucial where the geometry undergoes large movements or large deformations. Such as fluid structure interaction (FSI) problems and inverse problems. In all above mentioned examples moving the geometry in three dimensional space and sustaining the quality of the mesh are very challenging. The immersed boundary method is designed to address these shortcomings at least for certain classes of fluid flow. This approach, relaxes the condition of the mesh projection on the geometry at the expense of mathematically treating the boundaries. Combining this method with Cartesian adaptive mesh refinement will enable one to minimize the simulation turn around time tremendously. In other words, no pre processing is involved and switching from one geometry to the other is trivial. The goal of the present study is to develop an open-source software package to generate an adaptive mesh for a given complex geometry. It should be noted that, the presented methodology here is not specific to immersed boundary calculations as one can easily modify the method to project the mesh on a geometry.
2 OCTREE CONSTRUCTION

Mesh generation using octree starts with one element and generates the mesh by successively dividing the parent element. Octree is generally created by dividing each edge in half, this is also referred to as isotropic refinement. This recursive refinement continues until a desired resolution is achieved. Morton encoding is a substantial part of our methodology. Here, we elaborate on it in detail. Every node at each level in an octree can be identified by 3 bits, with every bit corresponding to the location of the element in x, y and z directions, respectively. Figure 1 presents the Morton ordering for a mesh in three dimensions, which is refined only one level. Note that we colored the four lower elements on the bottom surface while coloring the top four elements on their top surfaces to avoid clutter in the visual representation. The bit representation holds for the individual cube.

![Morton code construction at a single level based on element location in x, y, z directions](image)

Figure 1: Morton code construction at a single level based on element location in x, y, z directions

3 FUNDAMENTAL DATA STRUCTURES

Different types of data structures have been used to store the tree data. The selection of data structure is a crucial step in developing linear octree generation algorithms due to the fact that it will directly impact the execution time as well as memory consumption. In this section, we elaborate on the two important data structures used in octree generation. We briefly explain the Z-curve enhanced hash table and Z-curve compliant red-black tree.

3.1 Hash Table

The most popular methods for tree generation used in the scientific computing is hashing using Morton encoding. This is also classified under the pointerless octree generation techniques. There are several ways to construct the Morton code. In this part of our implementation, we make use of hash maps, which is a well-known associative container data structure in the field of computer science. A hash map uses a hash function to produce an index for each key. An efficient implementation of a hash map unordered_map is readily available as part of the C++ standard library. When solving a PDE, each key is
mapped to a pointer to the solution vector, which is designed according to the numerical formulation and discretization method. This method can be used in combination with Z-curve to enhance data locality. This is achieved by converting the bitwise representation of every node to an integer.

3.2 Red-Black Tree

Red-black tree is widely used in different areas of computers science due to its ability to guarantee worst-case situations for insertion, removal and search operations. It is used from computational geometry to process scheduling in Linux kernels. We explore its potential for scientific large scale mesh generation and adaptation. Red-Black tree only needs a comparison function and hence the real value of the integer does not matter and it is only used to compare two elements.

4 Strong Scaling

Strong scaling is performed on a sports car geometry for 18 levels of adaptation. The results are presented in Fig. 8. The efficiency of bigger than 1.0 is due to the cache effect.

![Graph](image)

**Figure 2:** Illustration of strong scaling for the parallel version, a sports car is used for this analysis, nLevel=18, nElements=189,220,200.