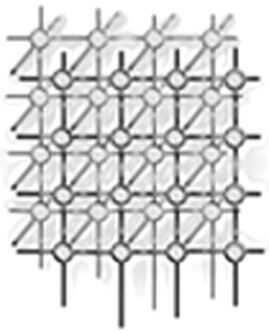
Parallel Isosurface Extraction for 3D Data Analysis Workflows in Distributed Environments

Daniele d'Agostino, Andrea Clematis, Vittoria Gianuzzi

Parallel Isosurface Extraction for 3D Data Analysis Workflows in Distributed Environments

Journal:	*Concurrency and Computation: Practice and Experience*
Manuscript ID:	CPE-07-0063.R3
Editor Selection:	Prof. Luc Moreau
Wiley - Manuscript type:	Research Article
Date Submitted by the Author:	03-Dec-2010
Complete List of Authors:	D'Agostino, Daniele; CNR, IMATI Clematis, Andrea; CNR, IMATI Gianuzzi, Vittoria; Univeristà di Genova, DISI
Keywords:	Parallel Isosurface Extraction, Remote Visualization

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*



Parallel Isosurface Extraction for 3D Data Analysis Workflows in Distributed Environments

D. D'Agostino[1,*,†], A. Clematis[1], V. Gianuzzi[2]

[1] *IMATI-CNR, Via de Marini 6, 16149 Genova, Italy*
[2] *DISI University of Genova, Via Dodecaneso 35, 16146 Genova, ITALY*

SUMMARY

In this paper we discuss the issues related to the development of efficient parallel implementations of the Marching Cubes algorithm, one of the most used methods for isosurface extraction, which is a fundamental operation for 3D data analysis and visualization. We present three possible parallelization strategies and we outline pros and cons of each of them, considering isosurface extraction as stand-alone operation or as part of a dynamic workflow. Our analysis shows that none of these implementations represents the most efficient solution for arbitrary situations. This is a major issue, because in many cases the quality of the service provided by a workflow depends on the possibility of selecting dynamically the operations to perform and, consequently, the more efficient basic building block for each stage. In this paper we present a set of guidelines that permits to achieve the highest performance for the extraction of isosurface in the most common situations, considering the characteristics of the data to process and of the workflow. These guidelines represent a suitable example to support the efficient configurations of workflows for 3D data processing in a dynamic and complex computing environment.

KEY WORDS: Parallel Isosurface Extraction, Remote Visualization

1. INTRODUCTION

Nowadays scientific instruments, medical equipments such as X-ray based Computed Tomography (CT) and scientific simulations, produce an increasing amount of huge volumetric data, that is 3D matrices of values. In order to extract information from these datasets it is

*Correspondence to: IMATI-CNR, Via de Marini 6, 16149 Genova, Italy
†E-mail: dagostino@ge.imati.cnr.it

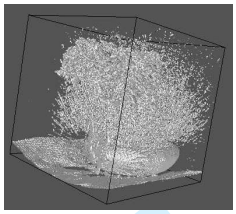
Received
Revised

important to analyse and visualize them in a proper way. *Isosurface extraction* is a basic operation that allows implementing many kinds of queries on these data. In particular it enables to determine the shape of organs and features of tissues in medical analysis, or the shape of molecules in bioinformatics applications, on the basis of a given threshold value (called *isovalue*). Isosurfaces are represented as a Triangulated Irregular Networks. Three examples of isosurfaces are shown in Figure 1.

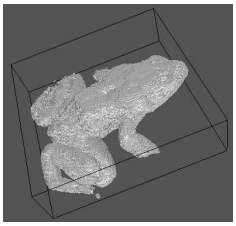
The Marching Cubes [1] is the most widely used algorithm for isosurface extraction. Many efficient implementations of this algorithm exist, in particular different powerful and freely available toolkits, like VTK [2] and AVS [3], provide it with a basic component. In the recent past most of the research activities about visualization and 3D data analysis started to consider the issues deriving from the impressive growth of data size, and from the use of basic components in complex workflows, often executed in a distributed and dynamic context such as the Grid [4]. With regard to the first issue, parallel processing represents a suitable approach to improve the performance of basic components, while different technologies such as component based software development, can be adopted to properly configure complex workflows.

Different examples of visualization tools that provide innovative features oriented towards parallel processing and distributed execution of complex workflows, now exist or are in development. ParaView [5] and VisIt [6] are open source, multi-platform visualization environments, which support distributed computation to process large datasets. Both of them provide a professional environment based on VTK as data processing and visualization kernel combined with MPI to provide parallel processing capabilities. In particular VisIt was designed with a strong emphasis on extremely large datasets. The SCIRun [7] project and tool provides an open source software, which allows developing dynamic problem solving environment for visualization applications [8]. COllaborative VIsualization and Simulation Environment (COVISE) [9] is an example of extendable distributed software environments, which puts the emphasis on high performance and parallel processing to deal with complex models and huge datasets. IRIS Explorer ™[10] is a visualization product developed by NAG, which is used for 3D visualization in different contexts such as life science, chemistry, multimedia, aerospace, medicine and others. The Grid Visualization Kernel (GVK) [11] project provides services to connect visualization clients and Grid applications. It relies on Globus [12] to implement functionalities of visualization pipelines. The DataCutter [13] framework provides an infrastructure that enables to develop data intensive applications in Grid environments, using a filter streams programming model especially for data visualization purposes.

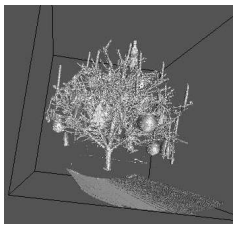
Despite the above mentioned wide activity, different aspects have still to be addressed, as noticed in [14], to develop fully satisfactory environments for visualization in true distributed environments. In particular, a common practice is a manual staging of components on resources into an execution pipeline. This approach is impractical in grid environments, where a single application might use many heterogeneous resources. Moreover, while the performance of a single component is straightforward to quantify, it is very difficult to evaluate the overall performance of a visualization pipeline. In fact parallel processing is often adopted for visualization tasks pursuing a "per component" optimisation strategy (e.g. by algorithm designers). The whole workflow optimisation is rarely considered. Instead we observe a lack of analysis aiming at a combined approach that considers different alternatives for single



(a) Isosurface extracted from the bonsai dataset for the isovalue 2.



(b) Isosurface extracted from the frog dataset for the isovalue 75.



(c) Isosurface extracted from the Christmas tree dataset for the isovalue 180.

Figure 1. Three examples of Isosurfaces.

component implementations and the effect of their use in different workflows, to process datasets with different characteristics.

This is a complex but also a frequent problem that involves performance analysis, algorithm design and evaluation, and software engineering aspects for dynamic component binding. In this paper we provide a contribution to address these problems, presenting our experience regarding the isosurface extraction and its parallelization strategies in common applicative scenarios. In particular, we focus on the analysis of a set of parallel components for isosurface extraction and the discussion of the guidelines for selecting the best solution considering their use within widely adopted workflows, which for a given volumetric dataset, produce in most cases one or few isosurfaces. As shown below, a general solution to the problem does not exist. Instead it is useful to consider different implementation strategies of the same component, and to define guidelines for the exploitation of the best suited implementation within a range of different situations.

1.1. Parallel Components and Workflow Optimization: An Example

To better clarify the issue, we shall consider a system for distributed visualization that we have implemented [15, 4]. The basic services it provides are the extraction of isosurfaces from volumetric datasets and their further processing for suitable visualization on remote clients. The high level architecture is represented in Figure 2.

The system is an example of a workflow that uses parallel isosurface extraction as a basic component together with other parallel and sequential operations on 3D data. Many problems arising from the remote nature of our service and from the data characteristics have to be taken into account. The most important are the following:

- The heterogeneity of the client devices used to analyse the results, because the size of computed isosurfaces may overcome their graphic capabilities. This problem can be

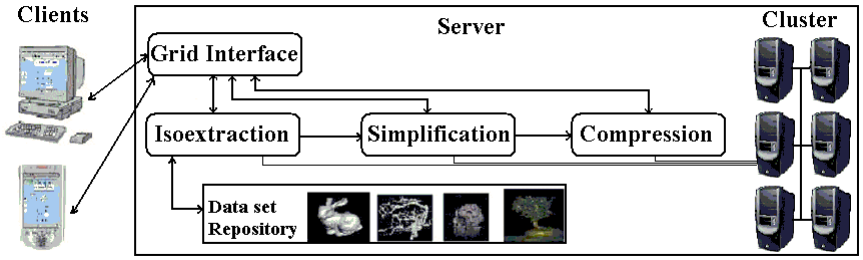


Figure 2. The high level architecture of our distributed 3D data analysis and visualization system.

tackled by providing a *simplification operation*, which produces a good approximation of the original surface made by less triangles.

- The performance of the interconnection network, because the transfer of the isosurface is perhaps the most costly part of the process. A *compression operation* that reduces the size of the data to be transferred, can be adopted to reduce the costs.
- The possible huge amount of data to process. The design of *parallel algorithms* for the considered operations represents a suitable solution to improve performance.

The composition of parallel algorithms is a major issue, because there is the need to consider the possible different parallelization strategies. For example, a very fast parallel algorithm based on the farm on demand paradigm (i.e. a worker requires a new task from the master when it finishes the previous one) may be followed by another one based on the data parallel paradigm (i.e. parallel processes subdivide among them the whole dataset at the beginning of the computation) . If the number of resulting tasks produced by the farm is greater than those required by the second algorithm, it is necessary to redistribute data, in order to produce a new, balanced subdivision. This *rebalancing* step may be computationally intensive, and this may lead to worsen the overall performance.

For this reason parallel algorithms have to be designed to achieve a good performance, considering both their execution as stand-alone programs and their composition in the workflow. This means that the parallelization strategy of each operation has to achieve two objectives:

- to provide high performance figures;
- to produce a partitioning of the output data that can be directly exploited by the next stage.

In many cases, as in our system, it is difficult to achieve both the objectives because it is not possible to know in advance the next step to execute [16]. The workflow in fact has not a fixed configuration; it is dynamically determined by considering the size of I/O data and other parameters as the available bandwidth and the kind of client devices. For example the

simplification operation may be required if the size of the isosurface overcomes the graphic capabilities of a client device, but most of the times this condition can be verified only after the isosurface extraction operation.

1.2. Issues in Pipeline Assembling

This example motivates the type of analysis we propose in the paper, because the quality of composite services depends not only on the selection of the more efficient and effective building block for each stage, but also on their flexible and efficient integration in a dynamic pipeline.

For this purpose we present an in-depth analysis of the Marching Cubes algorithm and three possible parallel approaches, each of them representing the best choice for a particular combination of the characteristics of the I/O data and the environmental parameters, in order to define the guidelines for the exploitation of the best suited implementation in a range of different situations.

We also consider other approaches based on the use of search structures, in order to speed up the isosurface extraction process. However, in cases similar to our workflow, where only few isosurfaces are extracted for a given dataset, the pre-computation of a data structure represents an overhead that decreases the performance.

The three parallel algorithms will be evaluated considering two measures: the *temporal efficiency*, which is the achieved speed up values considering only the single operation, and *the partitioning efficiency and quality*. Partitioning efficiency is measured considering the balancing of the number of triangles among processes, while quality is aimed at minimizing the number of shared (or border) triangles. The evaluation of the partitioning on the basis of these two aspects is motivated by the fact that the performance and the quality of the results of the simplification and the compression operations, as well as most of the operations working with triangular meshes, depend on them.

We show that it is not possible to determine one optimal general solution. Therefore we propose an isosurface extraction component with three parallel implementations and a set of guidelines to select the best solution on the basis of the particular situation.

The component (software is available on request) can be reused in other workflows and contexts. For example, we have used it in a bioinformatics application, the parallel system for the reconstruction of molecular surfaces [17]. Usually a molecule is represented through the set of its 3D atomic coordinates, as in the Protein Data Bank (PDB) [18], one of the most important repositories. Such formats are well suited for structural analysis operations, but not for those based on the molecular shape. In particular for molecular docking, which is one of our research fields, other representations such as the Connolly [19] surfaces, can be derived by modeling the atoms with spheres in a 3D volume, and extracting isosurfaces corresponding to three specific isovalues.

The rest of the paper is organized as follows. In Section 2 the Marching Cubes algorithm is described. In Section 3 alternative approaches based on the use of search structures are presented. The related works about the parallelization of the Marching Cubes algorithm and the three parallel versions we considered are reported respectively in Sections 4 and 5. Performance analysis results and the guidelines for parallel algorithm selection are discussed respectively in Sections 6 and 7, while conclusions and future works are outlined in Section 8.

2. THE MARCHING CUBES ALGORITHM

First of all, let us introduce the isosurface extraction operation. A **scalar volumetric dataset** is a pair (P, W), where $P = \{p_i \in \mathbb{R}^3,\ i = 1 \dots n\}$ is a finite set of points spanning a domain $\Omega \subset \mathbb{R}^3$ and $W = \{w_i \in \mathbb{R},\ i = 1 \dots n\}$ is a corresponding set of values of a scalar field f, sampled at the points of P, i.e. $w_i = f(p_i)$. A mesh Σ subdividing Ω into polyhedral cells having their vertices at the points of P is given. We assume that Σ is made up by parallelepipeds. This means that a cell $\sigma_j \in \Sigma$ is defined by eight points, $p_{j0}, \dots, p_{j7}$.

Given an **isovalue** $q \in \mathbb{R}$, the set $S(q) = \{p \in \Omega \mid f(p) = q\}$ is called the **isosurface** of the field f for the value q. The isosurface is normally approximated by a triangular mesh.

The **Marching Cubes** algorithm [1] is the most popular method used to extract triangulated isosurfaces from volumetric datasets. Let us briefly describe the I/O data format and an efficient implementation of the algorithm, called *Watt&Watt Marching Cubes*, which is the basis of the parallel algorithms that are the subject of this work.

It is worth noticing that we are not proposing a new sequential algorithm for isosurface extraction having a lower complexity. Instead, we describe an implementation that includes a set of optimization that none, at the best of our knowledge, considered in such a complete way.

2.1. Input and Output Data Format

If we denote with $Xmax$, $Ymax$ and $Zmax$ the size of the volume representing the generic dataset, the **input** of the algorithm is represented by a set of *Xmax*Ymax*Zmax* values. Each value is represented using B bytes, where B depends on the range in which the values can vary: with one byte it is possible to represent 256 values, with two bytes 65,536 values and so on.

The datasets are normally stored as a set of planes, called **slices**. Here we assume that datasets are made up of Zmax slices and each of them is made up of Xmax*Ymax values. Cells are made up of a pair of slices having subsequent values of the Z coordinate.

Volumetric datasets are produced by different types of devices, i. e. medical instruments, or by simulations, i. e. by fluid dynamics or Bioinformatics applications. Volumetric datasets are stored in one or multiple files following one out of different possible formats. For example the *Digital Imaging and Communications in Medicine* (DICOM) format [20] represents one of the most common standard for medical image storing, handling and transmission.

The **output** of the algorithm is represented by a triangular based representation of the isosurface. An efficient way to represent a triangular mesh is through the use of two tables, the **Vertex table** and the **Triangle table**. The Vertex table represents the geometry, i.e. it contains the coordinates of the vertices, while the Triangle table represents the connectivity, i.e. it contains the indices of the three vertices forming each triangle with respect to their position in the previous table.

This format is well suited both for visualization purpose and for further processing steps. With regard to the visualization, it is easy to convert the tables in most of the common file formats for 3D visualization, such as the *Object File Format* (OFF) [21] and the *Virtual Reality Model Language* (VRML 1 or 2) [22]. With respect to further processing operations, it has to

be considered that the incidence relation represented by the Triangle table implicitly contains all the other incidence relations among vertices, edges and triangles. This means that it is possible to derive any other representation of the mesh from these two tables.

Here below we remain as much as possible independent from a specific format, both for input and output data. We assume that considered datasets are composed only of the value of the points, so without any header, and furthermore the isosurfaces are represented by the Vertex and the Triangle tables.

2.2. The Original Marching Cubes Algorithm

In the Marching Cubes algorithm the triangular mesh representing the isosurface is defined piecewise over the cells of Σ. A cell $\sigma_j \in \Sigma$ is intersected by the isosurface represented by the isovalue q if the isovalue is between the minimum and the maximum of the values assumed by the cell points, that is $\min_i(w_{ji}) \leq q \leq \max_i(w_{ji})$. This kind of cells is called **active cells**. An active cell contributes to approximate the isosurface for a patch made of triangles. The union of all the patches represents the isosurface.

The algorithm consists of two main operations, the *Cell Classification* and the *Active Cell Triangulation*.

The **Cell Classification** is the operation that determines if a cell is intersected by the isosurface or not. This is done using a *bit vector* of 8 fields of one bit, each of them corresponding to one point of the cell. Points with values greater or equal to the isovalue are marked with 1, otherwise with 0: therefore a cell is an active cell if the bit vector has a value different from 0 (all points values lower than the isovalue) and 255 (all points values greater than or equal to the isovalue).

In these cases the **Active Cell Triangulation** operation is performed, consisting in the approximation of the intersection with the isosurface, using a triangular patch. Considering that a surface may intersect a cell in 254 ways, represented by the values of the bit vector except 0 and 255, a **look-up table** is used to enumerate all the possible connectivity schema. The coordinates of the vertices of the triangles are computed as a linear combination of the values of intersected edges.

2.3. The Watt&Watt Marching Cubes Algorithm

The original Lorensen and Cline's proposal presents some issues. Each of them was widely discussed in literature and many papers suggested solutions and improvements.

The first issue concerns the correctness of the results. It was demonstrated [23, 24] that the use of the original look-up table introduces topological inconsistencies in certain configurations of shared faces between cubes. A large number of techniques have been proposed in literature, aimed at producing consistent isosurfaces, to avoid holes and possible cracks, or to produce correct isosurfaces, meaning the real isosurface corresponds to the extracted one. We decided to adopt a solution based on the modified look-up table proposed in [25], which allows to efficiently produce consistent triangulations.

The second issue concerns the algorithm performance. Considering the cost of a single operation, we experimentally found that the Active Cell Triangulation operation costs one

magnitude order more than the Cell Classification operation. However, in many datasets less than 10% are active cells, therefore most of the execution time is spent examining non-active cells. A preprocessing step may allow the construction of many kinds of search structures, which permit to disregard parts of the volume that do not contain active cells, and consequently to reduce the execution time [26]. However, this kind of solution is suitable in the case of multiple queries on the same dataset, because the time necessary to create and access a search structure have also to be taken into account. This means that if a dataset is considered once or only a few times, the most efficient solution is the original exhaustive covering of all the cells, as discussed in Sections 3 and 6.3.

The third issue concerns the fact that each active cell is processed independently. This means that a vertex may be recalculated up to four times in adjacent cells. This approach may have a considerable impact on the computing time and on the size of the Vertex table. The best solution was proposed in [27], therefore we adopted it. With this technique the coordinates of a vertex are computed only the first time the vertex is considered. Then they are inserted in the Vertex table and the corresponding index is stored in one of the auxiliary data structures required by the technique. In this way the other cells sharing the vertex will use the stored index to form triangles.

The last issue concerns the efficient management of I/O data. In order to be able to process efficiently datasets and isosurfaces larger than the available amount of memory, we adopted a solution based on the use of buffering techniques. We use an input buffer to contain $NS \leq Zmax$ slices of the dataset and two output buffers to contain respectively vertices and triangles. All the buffers are allocated in the main memory, and their size is dynamically determined on the basis of the size of the dataset and the available RAM.

In [15] we presented an implementation of the algorithm in which all these issues were analysed and tackled. Hereafter we define it **Watt&Watt Marching Cubes (WWMC)**, because the first three issues were included in the version proposed in [27].

If we define with $numC$ the number of all the cell of the dataset, and with $numAC$ the number of the active cell, the analytic formulation of the computational cost of the WWMC algorithm is:

$$T_{WWMC} = T_I(numC) + T_O(numAC) + K_{CC} * numC + K_{ACT} * numAC \quad (1)$$

In this Equation $T_I(numC)$ and $T_O(numAC)$ represents the I/O times, $K_{CC} * numC$ the times for the Cell Classification operation and $K_{ACT} * numAC$ the time for the Active Cell Triangulation operation. The times for the acquisition of the data and the Cell Classification operation depend on the size of the dataset, represented by the number of cells, while the other two times depend on the size of the isosurface, represented by the number of active cells.

We analysed K_{CC} and K_{ACT} on a workstation whose characteristics are described in Section 6. While K_{CC} is truly a constant value equal to $8.36 * 10^{-8} sec.$, K_{ACT} varies between $5.7 * 10^{-7} sec.$ and $3.1 * 10^{-7} sec.$, depending on the number of triangles produced by the different kinds of active cells.

Thus, we can conclude that the WWMC algorithm is an improved version of the Marching Cubes algorithm that is able to treat efficiently arbitrary datasets and isosurfaces. However, its execution time may be high, because it is proportional to the size of the input dataset, and

to the percentage of active cells. Even if some further improvements can be obtained using threads to overlap computation and I/O, these are not sufficient to get a high performance level. This is the reason why we investigated possible parallelization strategies of the Marching Cubes algorithm.

3. APPROACHES BASED ON SEARCH DATA STRUCTURES

As widely demonstrated in literature, the use of external search structures allows speeding up the isosurface extraction process when multiple queries are performed on the same dataset. This result is achieved by creating an index on the data that permits to disregard large regions containing, for a given isovalue, only non-active cells. The various proposals may be conveniently classified in:

- **Range-based**: each cell c (or group of cells with a fixed size) is associated with the continuous set of values considering the lowest and the greatest value taken on by the function over its vertices: $R(c) = [\min_{p \in C} f(p), \max_{p \in C} f(p)]$. The isosurface extraction query concerns the enumeration of all cells c such as the isovalue $q \in R(c)$. The main search data structures adopted are the *kd-trees* [28] and the *interval trees* [29].
- **Domain-based**: the domain spanned by the dataset is hierarchically partitioned to disregard the parts of the volume not containing active cells. The main search data structure adopted is the *octree* [30].
- **Contour propagation**: on the basis of the continuity of the scalar field an active cell, called *seed cell*, is determined for each connected component of each possible isosurface. The seed cells are grouped in sets, considering the different isovalues. In this manner the entire isosurface can be traced by a breadth-first traversal through the face-adjacencies, starting from the cells of the *seed sets* corresponding to the required isovalue, as described in [31, 32, 33].

It is worth noting that a correct performance evaluation of these techniques has to take into account also the time and the space necessary to create and access the search structure. Therefore in many cases the adoption of search structures does not represent a feasible solution. In particular, two are the most representative situations:

- If a dataset is examined only once, the time necessary to create any of these indices plus the execution of the isosurface extraction is normally greater than the traversal of all the cells. This case occurs for example if the data represent the results of a medical examination or a computational fluid dynamics simulation. In other cases, only one specific isosurface is of interest, such as for protein surface reconstruction.
- If a dataset needs to be transferred for the analysis on a different resource. In this case, the additional time necessary to download the index has to be taken into account too.

Only a few examples of related work, such as [34, 35] suggest a comparison among some of the previous techniques, with contrasting results. For this reason, we decided to experiment two of these data structures, the *octrees* and the *kd-trees*. For the octrees we exploited an

application[†] based on the algorithm proposed in [30], while the implementation of the kd-trees is courtesy of Dr. Jürgen Toelke, one of the authors of [34].

An **octree** is a data structure that partitions the dataset, considering the recursive and even subdivision of the volume into eight equal parts at each step. The result is a tree, having at each level eight children. They are ordered considering the regular subdivision and labelled with the minimum and maximum values among the cell points that lie within. It is possible to speed up the extraction phase by pruning large branches of the tree, in order to traverse only the octants, whose values include the isovalue. In fact the range associated to an octant represents the union of the ranges of all the descendants. If the isovalue is not included, it is possible to disregard the corresponding branch of the tree.

In the range-based techniques the cells (or groups of them) are associated with their extreme values, but they are grouped considering these values instead of their spatial positions. In the algorithm proposed in [28] the cells are mapped into a 2D space, and a **kd-tree** is created on them, to allow an efficient range search. The tree organizes the points in such a way that, during the traversal, only one among the min and the max values is tested for each level of the tree.

It is worth noting that using the kd-tree, we have to store explicitly the position of the cells corresponding to a node, while in the octree this information is implicit. For this reason, the size of the kd-tree is larger than the octree. However the kd-tree enables to determine more quickly which are the active cells, as they are grouped taking into account their intervals and not their spatial positions.

Both of the programs we considered are composed by two main parts, the creation of the index and the effective extraction of the isosurface. In the first part the whole volume is acquired and processed.

In both cases we consider a block of 8x8x8 points as the smallest data unit, and we use buffers for the acquisition of large datasets, as explained in Section 2.3.

The resulting tree is then written by each program on the disk as a binary file, which is accessed in the following part. Both the programs then use the index to read and process only the blocks containing active cells. Regarding the isosurface extraction, we modified them to perform the same optimized operations we implemented for WWMC. We will present the performance of the two programs in Section 6.3.

4. RELATED WORKS ON THE PARALLELIZATION OF THE MARCHING CUBES ALGORITHM

Several parallel versions of the Marching Cubes algorithm were proposed on literature. They can be classified in four classes on the basis of the adopted parallelization strategy.

The first class represents the proposals based on the use of pure data parallelism. The aim is to provide a parallel module for isosurface extraction to an existing visualization system,

[†]The code can be found at ftp://ftp.cse.ucsc.edu/pub/avg/Scivi/

like the one proposed in [36] for the Visualization Toolkit (VTK) [2] and in [37] for the IRIS Explorer [10]. The main aspect is that only little attention to load balancing issues is paid in these modules.

The second class represents the proposals based on the use of search structures. The aim is to exploit the information provided by these structures to balance the workload among the parallel processes and to avoid the processing of large regions of non-active cells. Many proposals belong to this class. The most interesting papers are [38, 39] with the partitioning strategy based on interval trees, [40] on kd-trees and [41, 42] on octrees.

The last two classes are described respectively in [43] and [44].

In [43] the load balancing is achieved by a preliminary analysis step of the volumetric dataset, which counts the number of the vertices that will be produced in each pair of slices.

On the basis of the results, the partitioning of the dataset is determined with the purpose to distribute and maintain the isosurface in the available aggregate memory. In this way it is feasible to efficiently perform possible further operations, because the target of this algorithm is represented by interactive environments for image processing.

In [44] a static and random subdivision of the volumetric dataset among the nodes used for the isosurface extraction is proposed. The volume is subdivided into fine grain blocks, which are examined to extract a graph representing the range of values assumed by the cells points and the number of active cells intersected for each value. All the blocks having a similar graph are distributed among the nodes, in order to ensure that each parallel process will have the same amount of active cells for whatever isovalue. Please note that the preprocessing analysis is executed mostly in parallel.

We decided to evaluate separately the use of search structures, therefore we designed three parallelization techniques inspired by the remaining classes. We considered a pure data parallel algorithm, an algorithm with a load balancing step based on the spatial distribution of the active cells and an algorithm with the dynamic distribution of the blocks.

The first algorithm represents exactly the first class, while the other two are inspired by the last two classes, in order to exploit their positive features and overcome their limitations.

The main drawbacks of [43] are particularly the need of an amount of sufficient aggregate memory to maintain the whole isosurface, and the number of necessary data movements among the parallel processes to rebalance the volumetric dataset, according to the spatial mesh distribution, after the analysis step.

On the contrary, the main drawback of [44] is the production of a large number of duplicated data. We found the implementation of this algorithm made by the authors of the paper[‡], hereafter denoted as *Bajaj's parallel algorithm*, and we will present its performance in Section 6.3

[‡]The code is available at http://cvcweb.ices.utexas.edu/cvc/projects/project.php?pageIndex=1&proID=15

5. THREE PARALLEL IMPLEMENTATIONS OF THE MARCHING CUBES ALGORITHM

The key idea for the parallelization of the Marching Cubes algorithm is the subdivision of the cells among several parallel processes. This solution is possible because each cell can be treated in an independent manner. With this approach the main problem becomes the selection of a suitable partitioning strategy of the volumetric dataset.

We decided to adopt the pair of slices such as the "data unit" for partitioning, corresponding to *(Xmax-1)*(Ymax-1)* cells. This means that a partition is made up of intervals of two or more contiguous slices. Considering that datasets are stored per slice, this choice has the advantage to keep I/O overheads low and also to permit the production of fine grain partitions. We will discuss this choice in more detail by presenting the experimental results.

As mentioned above, the suitability of the partitioning strategy depends on the achievement of two goals, the balancing of the workload and a suitable spatial subdivision of the isosurface among the parallel processes. A perfect balancing of the workload can be obtained through an even distribution of both active and non-active cells among the parallel processes. This goal is very difficult to achieve due to the usually uneven distribution of the active cells in the volume.

A subdivision of the isosurface is suitable when each parallel process receives a part of the volume containing about the same number of triangles of the others, and at the same time the amount of duplicated vertices is low. The problem of duplicated vertices may arise because, whatever partitioning strategy is adopted, there will be many cell faces shared among processes. In this case two consecutive intervals share a slice, and therefore the vertices that possibly lie on it. Herafter we call them **shared vertices**.

Triangles lie instead within a cell, therefore they belong to only one process. However, we consider the triangles with one or two shared vertices as **shared triangles**, because they have to be managed carefully if the mesh partitions are directly exploited for further processing operations. For example, if a process deletes or modifies one of these triangles, it has to send this information to the other processes sharing the triangle vertices, in order to avoid topological inconsistencies.

Let us discuss these aspects by analysing the characteristics of the following three parallel algorithms:

- Load Balanced on Cells (LBC) algorithm: it is based on the data parallel paradigm with a load balancing strategy based on the total number of cells. With respect to the classification described in the previous Section, it represents the first class of parallel algorithms.
- Load Balanced on Active Cells (LBAC) algorithm: it is based on the data parallel paradigm with a load balancing strategy based on the total number of active cells. This information is obtained by a preprocessing phase executed "on the fly". It represents the third class, and it is in particular an improvement of the algorithm proposed in [43].
- Farm on Demand (FD) algorithm: it is based on the processor farm paradigm in which the balancing of the whole computation is obtained dynamically, providing that enough partitions (tasks) are generated. It is similar to [44] and therefore it represents the fourth

class. The main difference is that the random task distribution among the workers is made on demand, therefore the distribution is nondeterministic.

The input and the output have the same format for all the algorithms. The input is the same of the sequential algorithm, i.e. the volumetric dataset, which is subsequently subdivided by each algorithm with a different strategy.

Also, the output is the same isosurface of the sequential algorithm, that is partitioned among the parallel processes.

In particular, the parallel processes produce distinct Vertex and Triangle tables for each interval of slices they treat. The various Vertex tables may contain shared vertices, because of the aforementioned surface partitioning: in Section 5.1 we briefly present the problem and a technique that enables the efficient merging of the various tables in a unique file. This solution was adopted for all the parallel algorithms.

Concerning the I/O data, one must consider that parallel processes usually share them using a distributed file system. If the data are not stored on a local file system, the performance of the I/O operations are lower, because of the higher latency and a lower bandwidth with respect to local I/O operations. Another degradation factor is represented by the possible serializations of the read operations, due to the characteristics of the distributed file system and the contention of the network. Processes in fact may acquire/produce the data at about the same time. Hereafter we consider the use of a parallel file system for the input acquisition, while the output data are produced by each process on the disk of the machine on which it is executed.

5.1. The Management of Shared Vertices

The problem of shared vertices arises from the partitioning of the isosurface, which is a consequence of the volumetric dataset partitioning strategy. Shared vertices represent only a drawback, because they result in an unnecessary duplication of data increasing the size of the isosurface without adding any information. This duplication may represent a problem for further processing operations and also for the visualization of the isosurface, so there is the need to tackle the problem of removing multiple copies of a vertex. In this Section we present a brief description of our solution.

We considered a partition of the datasets in intervals of slices. This means that two processes share a slice, and consequently all the cell faces that lie on it. The situation is shown in Figure 3, where C_1 and C_2 represents a pair of cells sharing a face. If each process computes independently the intersection for the shared faces the resulting vertices will be duplicated in the two Vertex tables.

None of the triangles computed by P_i and P_{i+1} are duplicated, but all the triangles having at least one vertex on the shared slice Z_b have to be considered as shared if the mesh partitions are directly exploited for further processing operations, as mentioned before.

To avoid this situation, we let P_i compute the coordinates of the vertices, while P_{i+1} marks the vertices with labels. Process P_i inserts the vertices in its Vertex table and uses the correct indices to form triangles. Process P_{i+1} on the contrary, uses the labels to form triangles. At the end of the isosurface extraction process P_i will produce, besides the Vertex and the Triangle

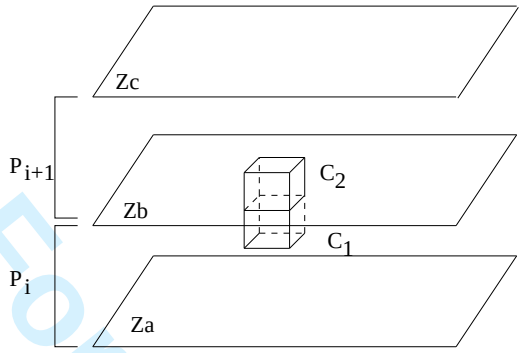


Figure 3. With the adopted partitioning strategy, two processes having contiguous intervals of slices share precisely one slice. This means for example that cells C_1 and C_2, belonging to the two different processes, share a face and the possible intersections.

tables, an **Index table** containing the indices of all the shared vertices. These indices will replace the labels in the Triangle table of process P_{i+1} when all the tables will be merged together in a single Vertex table and in a single Triangle table.

It is possible to implement this technique with little modifications with respect to the WWMC algorithm using the auxiliary data structures that allows avoiding the recalculation of the vertices. A complete description of the technique can be found in [15].

5.2. The Load Balanced on Cells Algorithm

This algorithm adopts the data parallel paradigm and load balancing is based on a homogeneous distribution of the cells of the dataset, therefore the name of algorithm **Load Balanced on Cells (LBC)**. This strategy is motivated by the fact that, usually, the highest time of the algorithm corresponds to the Cell Classification operation.

The slices of the dataset are subdivided into a number of intervals equal to the number of processes $numP$, so each interval is composed of about $\frac{Zmax}{numP}$ slices and each process receives only one interval. After the partitioning, each parallel process works independently by applying the WWMC algorithm, which has been slightly modified to manage the shared vertices on its interval. The output of each process is represented by a Vertex table, a Triangle table and, possibly, an Index table.

Considerations on the Partitioning Strategy
In this algorithm each process analyses $numC^i \simeq \frac{numC}{numP}$ cells, and consequently $numC^i \simeq numC^j \forall i, j \in [0 \dots numP - 1]$. This logically implies an *almost* linear speed up for the times of the input acquisition and the Cell Classification operations. The speed up may be limited by

the degradation of the input times, due to the use of a distributed file system. This degradation may vary according to the file system characteristics.

On the contrary, the distribution of the active cells among the parallel processes is unpredictable. The active cells may be very unevenly distributed and in the worst case they may belong to a single or very few processes. This means that the speed up for isosurface production and the Active Cell Triangulation operations is normally limited.

This aspect is also relevant in most cases, where the isosurface distribution is non-homogeneous.

5.3. The Load Balanced on Active Cells Algorithm

This algorithm adopts the data parallel paradigm and the load balancing strategy based on a homogeneous distribution of the active cells of the dataset, which gives the name of the algorithm **Load Balanced on Active Cells (LBAC)**. This strategy is motivated by the fact that the Active Cell Triangulation operation has the highest unitary cost.

In the LBC algorithm, the dataset partitioning is immediate: it is based on the number of slices and on the number of parallel processes, both known at runtime. The present partitioning strategy on the contrary, needs to know the number and the spatial distribution of the active cells in the volume. As previously mentioned, we do not assume to have a pre-computed index on the data, so this information has to be computed at runtime.

We logically subdivide the $numP$ parallel processes into a *coordinator* and $numW$ workers, with the coordinator being only responsible for the load balancing on the basis of the information collected by the workers. The parallel algorithm is composed of the following three phases.

1. Analysis phase
The aim of this phase is to determine the slices that contribute to the formation of active cells. These slices are defined as **active slices** and they are associated with a weight corresponding to the number of intersections they contain. This number is computed considering the values of the vertices of each edge. Logically this phase is equivalent to the Cell Classification operation but it is faster, because the bit vector is not computed explicitly.

It is performed on a data parallel partitioning of the volume, that is equal to that adopted in the previous algorithm. At the end, every worker sends to the coordinator the results for its slices.

2. Balancing phase
In this phase the coordinator, on the basis of the information provided by the workers, subdivides the slices into intervals, balancing the active cells. It is possible to achieve a very fine load balancing of the active cells because the granularity is represented by the single slice. At the end, the coordinator communicates the extremes of the assigned interval to each worker.

3. Isoextraction phase
In this phase each worker executes the same isosurface extraction algorithm of the LBC algorithm.

Listing 1. Pseudocode of the LBAC algorithm.

```
1 /********* Coordinator *********/
  if CoordinatorProcess() {
3     foreach worker w do
          ReceiveAnalysisRes(weight[w])
5     ActiveSliceBalancing(weight, &start, &stop)
      foreach worker w do
7         SendIsoextractionInterval(start[w], stop[w])
  }
9 /********** Worker **********/
  else {
11    DetermineAnalysisInterval(&start, &stop)
      ReadSlice(start, stop)
13    foreach slice s in the interval do
          AnalyzeSlice(s, &weight[s])
15    SendAnalysisRes(weight)
      ReceiveIsoextractionInterval(&start, &stop)
17    ReadNewSlice(start, stop)
      ... Cell Classification and Active Cell Triangulation as in WWMC ...
19  }
```

With respect to the I/O data, the considerations made for the previous algorithm are still valid. This algorithm is less straightforward than the others, therefore we present its pseudocode in Listing 1.

Considerations on the Partitioning Strategy

In this algorithm each worker W_i analyses $numAC_i \simeq \frac{numAC}{numW}$ cells, and consequently $numAC_i \simeq numAC_j \ \forall i,j \in [0 \dots numW-1]$. This logically implies a linear speed up for the times of the isosurface production and the Active Cell Triangulation operations.

Furthermore, considering that the balancing of the active cells means the balancing of vertices and triangles, the isosurface distribution among the parallel processes is nearly perfectly balanced.

However, the performance will be limited by the input acquisition and the Cell Classification operations. These operations in fact may be performed twice, once in the Analysis phase and then in the *Isoextraction* phase. In the Analysis phase we have a situation similar to that of the LBC algorithm: both the operations are balanced, because the cells are evenly distributed among the workers. This results in an almost linear speed up, that is limited by the use of a distributed file system, as in the previous case.

The re-execution of these operations in the *Isoextraction* phase represents the major overhead of the algorithm. The input acquisition in particular has a considerable impact, due to the reduced performance of the distributed file system. The incidence of the overhead may be reduced in two ways. Firstly, by exploiting the information on the non-active slice, it is possible to disregard pairs of non-active slices and consequently many non-active cells. Then, considering that normally the intervals for the Analysis phase and the *Isoextraction* phase have a non-empty intersection, only few slices will be read twice. This means that, on the average, the degradation resulting from this overhead may be relevant, but very much lower that the time of a complete re-read and re-scan of all the cells.

Another overhead is represented by the Balancing phase, but it is negligible.

5.4. The Farm on Demand Algorithm

This algorithm adopts the processor farm paradigm and the load balancing strategy is based on the distribution on demand of small intervals of slices, hence the name of the algorithm **Farm on Demand (FD)**.

In an algorithm based on the processor farm paradigm, the $numP$ parallel processes are subdivided into a *master* and $numW$ workers. The master is responsible for the assignment of the chunks of computation to the workers that, from time to time, require them.

The dataset is subdivided into a number of intervals greater than the number of workers. We call each interval *task*, and we denote with nT the total number of tasks. Normally tasks are made up by the same amount of slices (i.e. $\frac{Zmax}{nT}$), with the possible exception of the last one. This subdivision represents a finer grain partitioning with respect to the previous algorithms and it is able to allow a more efficient load balancing. Different tasks have different amounts of active cells and therefore different computational costs. For this reason, a worker that receives a task with few active cells will require a new task more quickly than a worker that receives a more expensive task. The computation is balanced without considering explicitly the number of active and non-active cells given to a process.

Workers execute for each task the same isosurface extraction algorithm of the LBC algorithm. In fact, we may consider the LBC algorithm as an instance of this algorithm where $nT = numP$.

With respect to the I/O data, in this algorithm the number of resulting files is greater, because the Vertex, Triangle and Index tables have to be produced for each task.

Considerations on the Partitioning Strategy

This algorithm theoretically allows an almost perfect and cheap load balancing, but the suitability of the isosurface partitioning is rarely completely satisfactory. The on demand subdivision of the tasks in fact may result in a not completely balanced subdivision of the isosurface and, in most cases, in a high number of shared vertices.

This last aspect is very important. Processes very often receive non-contiguous tasks, so there are up to $nT - 1$ shared slices and *probabilistically* much more shared vertices with respect to the previous algorithms, where the shared slices are only $numW$. It is worth noting that this result is highly probable but not always true: in some cases, for example, if the volume contains a set of distinct objects, the shared slice may be all non-active, and in this

case we will not have any shared vertices. However, a volumetric dataset normally represents the wrapping of an object, so a high number of shared slice results in a high amount of shared vertices.

6. PERFORMANCE ANALYSIS

We have performed experiments with the presented algorithms for isosurface extraction on different datasets, and we have collected performance data as well as information related to the quality and efficiency of the obtained mesh partitioning. We present in particular an in-depth analysis of LBC, LBAC and FD. For the sequential algorithms based on the use of the search structures and the Bajaj's parallel algorithm exploiting a static balancing of the cells, we show only the most relevant results instead.

All the algorithms are implemented in C, and using MPICH2 message passing library [45] for parallelism. We executed the parallel programs on a Linux-based Beowulf Cluster of 16 PCs equipped with 2.66 GHz Pentium IV processor, 512 MB of Ram and two EIDE 80 GB disks interfaced in RAID 0. The nodes are linked using a dedicated switched Gigabit network, and they share a PVFS2 [46] file system.

We exploited the existing PVFS2 partition for the sharing of data, in particular for the input datasets. PVFS2 is a good compromise among costs, installation simplicity and I/O performance, even if commercial solutions as Lustre or GPFS probably allow much faster I/O operations and therefore best speed up values. However, a comparison among different parallel file systems and optimal set-up of architectural parameters are outside the scope of this paper. The considered parallel algorithms in fact were designed to be portable on different architectural solutions, and therefore to exploit whatever kind of shared partition.

The sequential programs are executed using a single node of the cluster, and they use the local disk for I/O operations. We consider both synthetic and real datasets, and we evaluate the results on the basis of the following aspects:

- the temporal efficiency, that is the achieved speed up;
- the partitioning efficiency and quality, that is
  - the balancing of the produced triangles among the parallel processes;
  - the minimization of shared triangles among parallel processes.

The speed up is normally the most suitable measure to evaluate the performance of a single operation. On the contrary, to evaluate the overall performance of the workflow we have to consider the characteristics of the following steps.

The simplification and compression operations, as most of the operations performed on a triangular mesh, have a computational cost proportional to the number of triangles which have to be processed. This means that if triangles are badly balanced among the parallel processes, it is very difficult to achieve good speed up values for the parallel simplification and/or compression algorithms, and consequently the overall performance will be poor.

Considering the quality of the spatial subdivision of the isosurface it is important to minimize the number of the shared vertices. Very often the processing of a triangle having one or

two shared vertices requires the cooperation of the processes that share them; otherwise the results may not be topologically correct. The cooperation implies the need for these processes to communicate, and this leads to performance degradation proportional to the number of shared vertices. Another possibility is to disregard shared triangles during simplification or compression, but this normally lowers the quality of the results. In both cases, the management of shared vertices represents an issue, so they should be as few as possible.

In order to minimize the number of shared triangles, we do not use a quantitative measure, but we evaluate the number of shared triangles using a probabilistic approach. It is possible to show, in particular, that using a minimum number (with respect to the desired parallelism degree) of contiguous partitions of the mesh, we probabilistically minimize the number of shared triangles. This condition holds because isosurface is defined using local triangular patches. Hence we prefer partitions with a minimum number of contiguous subregions. Obviously it is possible to adopt a different strategy based on the exact counting of shared triangles. But this procedure has a cost that in most cases largely exceeds the obtained benefits. However, we plan to include in our future work ad hoc components such as "METIS" [47], a library of algorithms for partitioning unstructured graphs and hypergraphs.

6.1. Characteristics of Used Data

The characteristics of the volumetric datasets considered for the experiments are shown in Table I.

We used three synthetic datasets to analyse borderline situations that are not normally found using real datasets, but may be relevant to understand the performance of the algorithms. We considered a dataset producing an empty isosurface, because all the points have the same value 0, a dataset made up by all active cells for the isovalue 1, because the points assume alternately the values 0 or 2, and a non-homogeneous dataset, where the isosurface for the isovalue 1 is contained within the first 1000 slices. We call these datasets respectively Zero, ZeroTwo and NonH.

We considered five real datasets representing CT scans of a bonsai, a frog, a Christmas tree [§] a female cadaver [¶] and the volumetric description of the Connolly surface of the large ribosomal subunit from Deinococcus Radiodurans, identified as 1NKW in the Protein Data Bank [‖]. These datasets were selected on the basis of the size of I/O data, to test the algorithms on real cases. We denoted these datasets respectively with B, F, XT, FC and 1NKW.

The isovalues considered for the previous datasets and the characteristics of the resulting isosurfaces are shown in Table I together with the percentage of active cells and the execution times, in seconds, collected with the C implementation of the WWMC sequential algorithm using one node of the cluster. Three of these isosurfaces are shown in Figure 1.

[§]Generated from a real world Christmas Tree by the Department of Radiology, University of Vienna and the Institute of Computer Graphics and Algorithms, Vienna University of Technology [48]
[¶]Courtesy of the Visible Human Project [49] of the National Library of Medicine (NLM), Maryland
[‖]http://www.rcsb.org/pdb

Table I. Characteristics of the volumetric datasets and of the isosurfaces considered in the experiments. We denoted with $XxYxZ$ the grid size, with B the number of bytes used to represent the values, with $\%AC$ the percentage of active cell for a given isovalue, with $numT$ and $numV$ the number of triangles and vertices that compose the resulting isosurface and with T_{WWMC} the sequential execution times in seconds of the WWMC algorithm. With regards to the datasets, we denote with B the bonsai, with F the frog, with XT the Christmas tree, with FC the female cadaver and with 1NKW the protein.

Dataset	XxYxZ	B	Size	Isov.	% AC	numT	numV	T_{WWMC}
Zero	512x512x1000	2	500 MB	1	0	0	0	54.4
ZeroTwo				1	100	1,043,439,516	523,513,856	457.5
NonH	512x512x3996	2	1 GB	1	15.2	365,843,375	178,455,728	266.3
B	256x256x256	1	16 MB	29	4.32	1,435,852	722,820	2.0
				2	11.67	3,896,986	1,962,635	2.7
F	136x470x500	1	30 MB	75	4.1	2,655,552	1,334,280	3.7
XT	512x512x999	2	500 MB	180	0.89	4,514,539	2,294,042	30.0
				52	4.14	21,719,037	10,959,134	35.0
				25	60.07	365,459,601	178,339,350	135.4
FC	512x512x1734	2	867 MB	1000	7.37	69,600,216	35,594,711	83.5
1NKW	1031x1385x1204	2	3.8 GB	3018	0.01	72,001,372	35,999,636	275.3

For experiments with real datasets we considered those isovalues that produce large isosurfaces. Furthermore, we considered for the Bonsai and the Christmas tree an isovalue that produces about the same percentage of active cells as the Frog, to compare the influence of this percentage on the global computing time. We also considered another isovalue for the Christmas tree, corresponding to a low percentage of active cells, to compare the computational time of the algorithm with three different isosurfaces for a large input dataset.

6.2. The Temporal Efficiency

For the synthetic datasets we report the tests relative to the use of all the 16 nodes. For the real datasets instead, we performed a set of runs considering 4, 8 and 16 nodes. With the LBC algorithm each parallel process has its dedicated node, while the coordinator process of the LBAC algorithm and the master process of the FD algorithm share a node with a worker. These are lightweight processes, hence the sharing of CPU does not introduce any considerable overhead.

In Figures 4, 5 and 6, the speed up obtained is analysed considering both the *CPU time* and the *total time* (CPU + I/O) to outline the I/O influence on the performance.

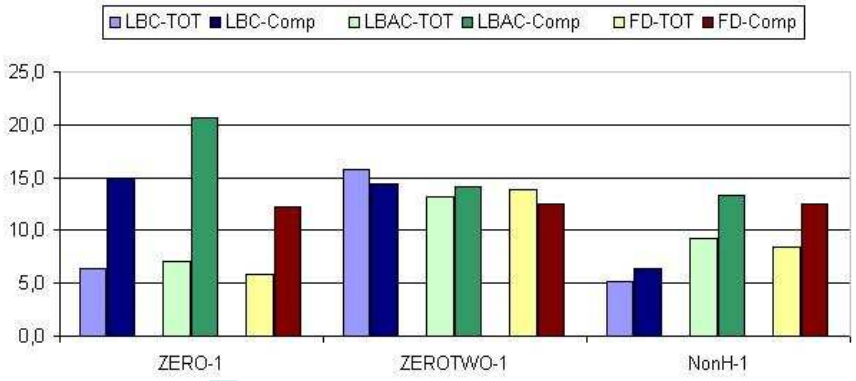


Figure 4. Graphic of the speed up (as $\frac{T_{seq}}{T_{par}}$) values obtained for the synthetic datasets considering 16 nodes. We indicated with "TOT" the speed up obtained for the total time, with "Comp" the speed up considering only the CPU time.

In Figure 4 we present the speed up values of the three algorithms considering the synthetic datasets. This analysis is useful because we are able to outline the behaviour of the algorithms in boundary cases, in order to determine the incidence of all the possible parameters on the isosurface extraction process in a quantitative way.

Empty Isosurfaces

If the required isovalue does not produce any isosurface (Zero-1) the I/O times are predominant. The performance of FD is limited by the greater number of I/O operations with respect to LBAC, which shows the highest performance, and also to LBC.

If we consider the speed up values of the CPU time, then LBAC achieves a superlinear result. This is due to the different structure of this algorithm with respect to the others. The LBAC algorithm performs at first the Analysis phase, in order to determine which are the active slices. After that, it considers again only the active slices to find the active cells and to compute the isosurface. If the result of the Analysis phase is that there are no active slices, as in this case, it is useless to go on with the isosurface extraction, because no isosurface exist.

On the contrary, the other sequential and parallel algorithms directly start the research of the active cells. Considering that the Cell Classification operation is more costly with respect to the Analysis phase, the optimal result of LBAC is evident. However, it is worth noticing that this behavior is limited only to this case, which is generally very rare, while in the other instances the Analysis phase represents an overhead that lowers the performance. For this reason is not convenient to implement an Analysis phase also for the other algorithms.

Large and Homogeneous Isosurfaces

When the isosurface is large and homogeneously distributed (ZeroTwo-1), the highest

computation time is represented by the Active Cell Triangulation. Considering that the required amount of computation is evenly distributed in the volume, LBC is able to present the highest performance. However, the performance of the three algorithms are quite close. This is due to the large size of the isosurface and the consequent lower incidence of overheads for LBAC and FD.

If we compare the speed up values of the total and the CPU times, we can see that I/O operations surprisingly do not represent a considerable overhead. This aspect is due to the large size of the isosurface, its balanced distribution and the use of the local disk to store the resulting tables. In this case in fact the time of the isosurface production is much greater than the time of the input acquisition, so that the overhead to acquire the input data is negligible.

Large and Non-Homogeneous Isosurfaces

When the isosurface is non-homogeneously distributed (NonH-1), the highest performance is still provided by LBAC. The highest computation time is represented by the Active Cell Triangulation, and the achieved speed up allowed by the Balancing phase of LBAC is able to overcome the overhead due to the Analysis phase, and consequently to present a better result with respect to FD. Clearly LBC presents the worst performance, because the entire isosurface is assigned to only a few processes.

Small-Medium Isosurfaces

For the real datasets the speed up values obtained considering the CPU times are shown in Figure 5, while those considering the total time are shown in Figure 6.

Table II also shows the times in seconds effectively spent to compute the isosurface extraction and to perform I/O operations for the most interesting cases.

If we consider only the execution of the isosurface extraction algorithm, we can see that FD is, with very few exceptions, the best possibility, especially if we take into account also the I/O cost. This is due to the normal uneven distribution of the isosurfaces and to their small size. For the same reasons, LBC has in the average a good performance. For small-medium isosurfaces, the highest time is the Cell Classification and this is balanced by its partitioning strategy. LBAC on the contrary, does not provide the expected high performance, with the only exception of XT-25, where it achieves the best time. For large isosurfaces, as we saw before, it provides the best results. However, for small-medium cases, the low level of efficiency is due to the relatively high cost of the rebalancing step with respect to the cost of active cell processing.

Further Considerations

We can see that LBC with 16 processes outperforms LBAC and FD for the largest dataset 1NKW-3018, mainly because it keeps the I/O overheads low and because of the use of the pair of slices such as the "data unit" for partitioning. Each process in fact has to consider about 240 MB of data, but while with LBC these data are acquired with a single operation, the processes of FD acquire them subdivided in multiple chunks, and the processes of LBAC may have to read more data due to the rebalancing.

When considering FC-1000 instead, FD behaves better than LBC, because of a better load balancing of active cells provided by a random assignment of tasks to workers. We implemented

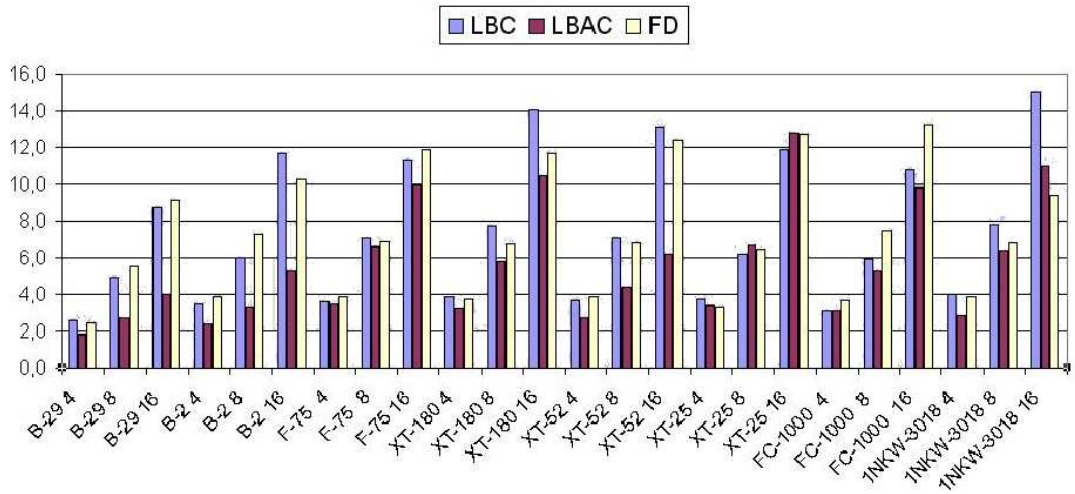


Figure 5. Graphic of the speed up values considering the CPU time, obtained for the real datasets with 4, 8 and 16 nodes.

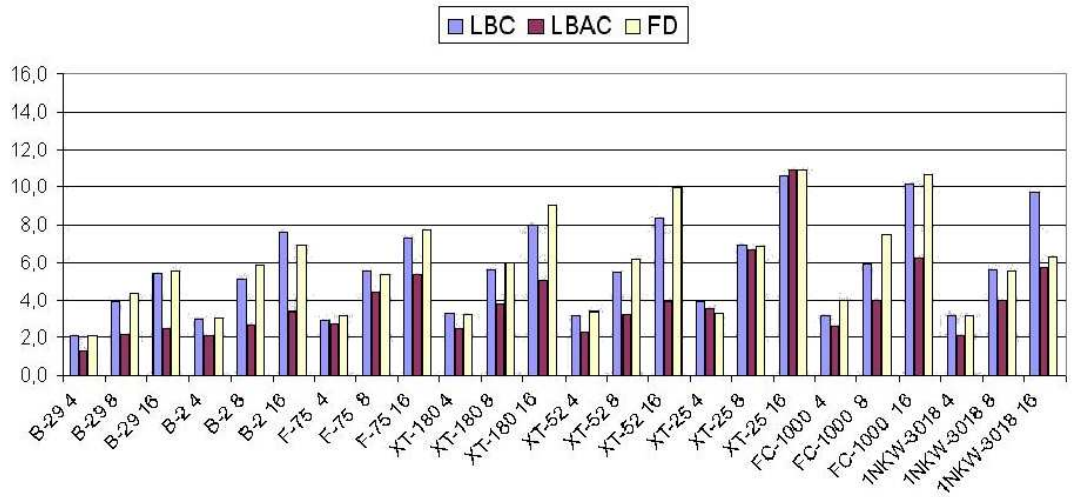


Figure 6. Graphic of the speed up values considering the total time obtained for the real datasets with 4, 8 and 16 nodes.

Table II. This table shows the times in seconds spent by the sequential and parallel algorithms to compute the isosurface extraction and to perform I/O operations for the most interesting cases: a small dataset and isosurface (B-29), a medium dataset and isosurface (XT-52), a medium dataset and a large isosurface (XT-25) and a large dataset and a medium isosurface (1NKW-3018).

Dataset	Algorithm	1		4		8		16	
		CPU	I/O	CPU	I/O	CPU	I/O	CPU	I/O
B-29	T_{WWMC}	1.70	0.30						
	LBC			0.66	0.30	0.34	0.17	0.19	0.17
	LBAC			0.93	0.61	0.62	0.31	0.43	0.29
	FD			0.68	0.28	0.31	0.15	0.19	0.18
XT-52	T_{WWMC}	26.55	8.45						
	LBC			7.18	3.76	3.74	2.69	2.04	2.17
	LBAC			9.74	5.67	6.06	4.68	4.29	4.34
	FD			6.88	3.41	3.92	1.74	2.14	1.67
XT-25	T_{WWMC}	94.20	41.20						
	LBC			25.13	9.72	15.21	4.34	7.94	3.84
	LBAC			27.77	10.60	14.08	6.36	7.34	5.03
	FD			28.10	12.63	14.63	5.19	7.40	4.91
1NKW-3018	T_{WWMC}	231.14	44.16						
	LBC			57.79	26.25	29.63	15.53	15.41	12.97
	LBAC			79.70	47.39	36.12	30.71	21.01	25.29
	FD			59.27	26.76	33.99	16.06	24.59	17.11

a similar partitioning strategy also for LBC: we subdivided the dataset into blocks of size of 128x128x256, which is a single block scattered among 256 slices, and we assign them to the LBC processes in a round robin way. Even providing the same amount of data with respect to the original partitioning, we found that the input time is more than 5 times greater than in the previous case. It is worth noticing that we can obtain about the same ratio using this partitioning strategy also with FD: in the original case a task is made up by contiguous slices, while in this case the data are scattered in the volume.

These results obviously depend on the organization of the data within the input dataset. If the isosurface extraction operation is inserted into a pipeline where the previous step produces the volumetric data subdivided into blocks, it is obviously more convenient to acquire them, rather than rearranging the data in interval of slices. For this reason, we plan to add specialized I/O routines to our algorithm implementation to tackle these cases.

Another issue is represented by the partitioning grain. Blocks can be composed by an arbitrary number of cells, while an interval of slices is composed by at least $(Xmax - 1) * (Ymax - 1)$ cells. The question remains if such a fine grain level is suitable for the isosurface extraction operation. The sequential processing of a dataset like the Bonsai, which may be considered as a unique block of 255x255x255 cells, requires less than 3 seconds. It is

Table III. A comparison among the execution times of the WWMC Marching Cubes algorithm with respect to the use of an octree or a kd-tree. Times are expressed in seconds. In brackets we indicated the sum of the times necessary for the construction of the search structure plus the isosurface extraction time.

	Structure Size	Structure Creation	XT-180	XT-25
WWMC	-	-	30	135.4
Octree	219 MB	34	10.2 (44.2)	124.8 (158.8)
Kd-tree	988 MB	72.6	9.1 (81.7)	119 (191.6)

therefore useless to subdivide it among a large number of processes, because the I/O and the communication overheads will limit the achievable speed up.

6.3. The Use of External Search Structures and Other Parallel Approaches

In the experiments we considered the time to compute and to store the octree and the kd-tree for XT, which represents a medium size dataset, and the improvement using them with respect to WWMC for XT-180 and XT-25. These two isosurfaces represent the two extreme cases that may occur. With XT-180 these indices permit in theory to skip the analysis of up to 99% of the cells, while for XT-25 the gain is less considerable, because more than half of the cells are active.

The results are presented in Table III. As expected the use of both trees allow the extraction of XT-180 in about one third of the time with respect to WWMC, while the performance for XT-25 is comparable. However, in both cases, if we consider the sum of the times necessary to build the search structure and to extract a single isosurface, we obtain a worse result than the exhaustive covering of all the cells made by WWMC.

If we consider the extraction of both surfaces one after the other, using the same index, we find again that WWMC presents a better performance (165.4 sec.) than respectively the use of the Octree (169 sec.) and the Kd-tree (200.7 sec.).

To the best of our knowledge, a parallel version of these algorithms is not available, but we would expect similar results nevertheless.

In particularly, it is true that it is possible to subdivide the analysis of the volume among the parallel processes; however, the construction of the data structure requires the global management of a coordinator process, therefore the speed up is likely to be sublinear.

We can conclude that the use of these indices is convenient if at least three isosurfaces are extracted from the same dataset. As said before, this is not always required, therefore in these cases WWMC represents the most efficient choice.

A similar situation arises if the indices are already available, but the processing of the data has to be performed on different computational resources. When using the octree, we have to download another 219 MB, while the size of the KD-tree is about twice the size of the dataset. If we consider for example the system for the reconstruction of molecular surface we described

Table IV. A comparison among the execution times of LBC, LBAC and FD with respect to the use of the static load balanced partitioning proposed in [44]. In this case, we indicated in brackets the sum of the time needed by the parallel construction of the search structure added to the isosurface extraction time.

	Structure Size	Struct. Creation	XT-180	XT-25
Worst parallel alg.	-	-	LBAC : 6	LBC : 12.7
Best parallel alg.	-	-	LBC : 3.3	LBAC : 12.3
Static Balancing	493 MB	11.4	0.1 (11.5)	19.1 (30.5)

in Section 1.2 and the 1NKW dataset, it is obviously more convenient to download the PDB file with a size lower than 1 MB, and to create the dataset on-the-fly on the remote cluster, rather than to download the pre-computed volume and its index.

With regard to the implementation of the Bajaj's parallel algorithm, we compared its execution time with the most and the least performant algorithms for the previously considered two isosurfaces using 16 nodes. The best algorithm for XT-180 in particular is LBC and the worst is LBAC, while the opposite applies for XT-25.

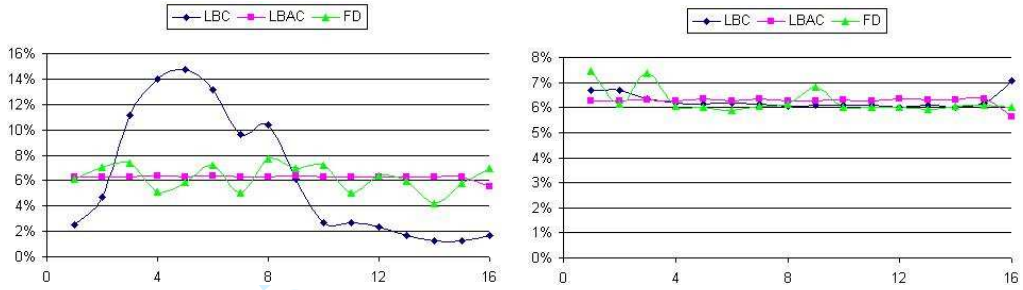
Results are shown in table IV. We can see that for XT-25 the isosurface extraction time is also larger than LBC, which has the worst result with respect to LBAC and FD. This is due to the overhead of accessing the data after the partitioning. The parallel algorithm in fact modifies the storage format of the dataset. Each parallel process, during the preprocessing step, produces locally a new file containing the data of the blocks statically assigned to it and other information. The result is that the original dataset is disregarded, and that 16 new files, with an average size of about 60 MB, are created. Each file contains cells belonging to different parts of the volume, therefore their spatial position has to be explicitly computed or acquired for each isosurface extraction.

As mentioned above, generally speaking, the main feature of this technique is that if the partitioning is sufficiently fine grained, the distribution will be almost balanced, also considering different isovalues for a dataset. Moreover, the data are acquired exploiting a local file, and this allows avoiding the overheads due to the use of a parallel file system.

However, besides the need to acquire the cells spatial position, which in some cases badly worsen the performance, it is worth noticing that the static partitioning fixes the parallelism degree. If we want to use 8 or 32 processes, and consequently 8 or 32 nodes, there is the need to produce a new set of 8 or 32 files.

6.4. The Partitioning Efficiency and Quality

The speed up values represent only one of the measures we use to evaluate the algorithms. The second one is the partitioning efficiency and quality, measured considering the balancing of the triangles among the parallel processes and the amount of shared vertices.



(a) The most common situation, represented by FC-1000.

(b) An homogeneously distributed isosurface, represented by XT-25.

Figure 7. Graphic of the distribution of the active cells, and consequently of the triangles, among 16 parallel processes for the three algorithms in two meaningful cases.

The influence of the resulting partitioning on the overall performance is clear, considering that the resulting parts of the isosurface are processed by the following steps of the workflow. Therefore, if a redistribution of the triangles is necessary due to their unbalancing, or if there is a high number of shared vertices, the total time for executing the workflow may considerably increase.

LBC and LBAC produce an isosurface that is partitioned on the base of the number of parallel processes $numW$ (equal to the number of used nodes), while FD on the number of tasks $nT > numW$. Only the partitioning strategy of the LBAC algorithm takes into consideration the problem of balancing the active cells, and consequently the triangles, among the processes. None of them instead explicitly takes into consideration the shared vertices in the partitioning of the volume. Generally speaking, we can say that LBC and LBAC partition the isosurface in large, contiguous regions along the Z axis, while the FD algorithm produces more fine grained partitions that are distributed in a random way, hence non contiguously, among the workers. Therefore, probabilistically, the partitioning of the FD algorithm will produce more shared vertices than the other two algorithms.

We analysed the partitioning resulting from the processing of the real datasets using 16 nodes, that is the most interesting case. Instead we disregarded the synthetic datasets. The results can be grouped in two classes, represented by Figures 7(a) and 7(b).

Figure 7(a) presents a comparison of the distribution of the active cells resulting from the three algorithms for the isosurface FC-1000. Many real cases are similar to this one. Considering the partitioning of the LBC algorithm, we can see that 80% of the isosurface is subdivided among the first eight processes, because most of the isosurface is included between slices 0 and 500. By using LBC we will have process 5 with about seven times the load of process 1 or processes from 9 to 16. FD presents a significatively better result, but also in this case the unbalancing can be high, since the load of processor 8 is twice the load of processor 14.

Finally, LBAC is able to produce quite a perfect balancing, because the percentage of active cells received by a worker varies between 5.5% and 6.4%.

Looking at the number of shared vertices, we found that LBC produces 195,804 of them, LBAC 318,472 and FD even 1,007,205. We may say that LBC produces less shared vertices but the load balancing of the triangles is very poor. On the contrary, FD and even more LBAC subdivide the mesh in balanced partitions, but the number of shared vertices produced by FD is very large. This is because, as mentioned earlier, contiguous tasks are normally assigned to different processes (in an experiment we see that for the case FC-1000 worker 1 processed tasks 10, 22, 35 and 52, worker 2 processed tasks 13, 30, 38 and 56 and so on), therefore there are up to $nT - 1 \gg numW$ shared slices with respect to LBC and LBAC, where the shared slice are only $numW$.

Figure 7(b) presents the same comparison considering the case XT-25. In this instance, the three algorithms present about the same results. It is worth noting that similar cases are not so frequent, because they depend on an high percentage of active cells and/or on their homogeneous distribution in the volume. However, also in this case the number of shared vertices produced by FD is very large: they are 11,847,754 with respect to the 2,111,077 of LBC and to the 2,117,112 of LBAC. LBC and LBAC present about the same number of shared vertices because the isosurface is well balanced in the volume, therefore their partitioning is almost the same.

7. Guidelines for Parallel Implementation Selection

To summarize the pros and cons of the three parallel algorithms we can say that the speed up of the *LBC algorithm* depends on the percentage of active cells, in particular on the incidence of the Active Cell Triangulation and Isosurface Production times on the whole execution time. The LBC algorithm will provide an acceptable performance for medium-large datasets with isovalues producing not very large isosurfaces and also for large datasets with isovalues producing large but homogeneously distributed isosurfaces. Only in this last case also the isosurface partitioning will be acceptable.

The opposite situation arises considering the *LBAC algorithm*. Its achievable speed up depends on the incidence of the double execution of the input acquisition and the Cell Classification operations on the whole execution time. The incidence is in inverse relationship to the size of resulting isosurfaces, so the algorithm achieves a poor performance, with medium-small isosurfaces. For large isosurfaces, however, this algorithm is able to provide better performance than the LBC algorithm. Obviously these cases are not so frequent, but they are interesting and can benefit much more from the parallel processing than the other cases. Furthermore, it is worth noting that the partitioning strategy is able to distribute the isosurface in a quite perfectly balanced way among the processes in any situation.

As a matter of fact, even if the *FD algorithm* is generally able to provide a good performance, because the partitioning strategy allows a load balanced computation to be achieved, its mesh partitioning probabilistically will not be satisfactory, mainly because of the large amount of shared vertices.

Table V. A summary of the characteristics of the algorithms.

	LBC	LBAC	FD
speed up	it degrades in proportion to the size and the non homogeneous isosurface distribution	it improves in proportion to the size and the heterogeneous isosurface distribution	good if the grain size is suitable
balancing of triangles	depends on the isosurface distribution	quite perfect	good in the average
shared vertices	probabilistically few	prob. few	prob. many

On the basis of these results, summarized in Table V, we can affirm that none of the parallel algorithms represents the optimal solution in whatever case. However, it is possible to consider in a combined way temporal efficiency and partitioning efficiency and quality to derive some guidelines that can be adopted in practical situations, to select the more adequate implementation for isosurface extraction.

The situations that can arise are characterized by both the workflow configuration and the kind of isosurface.

Four are the pipeline configurations that can arise:

I: workflow where the isosurface is the last operation;
II: workflow with data streaming;
III: workflow where the next operation requires a load balanced partition of the data;
IV: workflow where the next operation requires data coherence, as well as load balancing.

Regarding the kind of isosurface, we based our considerations distinguishing among the four most relevant cases:

- a large and homogeneous isosurfaces, as XT-25;
- a large and non-homogeneous isosurfaces, as FC-1000;
- a small and homogeneous isosurfaces, as F-75;
- a small and non-homogeneous isosurfaces, as B-2.

It is worthwhile to note that in most of the cases, we know for sure the characteristics of an isosurface only after its extraction and therefore it may look difficult or impossible to derive guidelines starting from an information that it is not available, when it is necessary. In these situations we will select always the algorithm that is able to behave well also in the worst case, which is a large and non-homogeneous isosurface, where a considerable uneven distribution of the active cell normally results in poor performace.

However, in many practical cases the user knows what he/she is looking for when extracting isosurfaces and it is possible to exploit an advanced rough classification for the selection. For example a user may require for the female body dataset an isovalue corresponding to the attenuation value of metallic implants as plates, pacemakers and screws. The isosurface will

Table VI. The efficiency values of LBC, LBAC and FD using 16 nodes for the four kinds of isosurfaces we considered. The values in brackets represent the efficiency considering only the CPU times, the others the efficiency considering the total times.

Dataset	LBC	LBAC	FD
XT-25	0.7 (0.7)	0.7 (0.8)	0.7 (0.8)
FC-1000	0.6 (0.7)	0.4 (0.6)	0.7 (0.8)
F-75	0.5 (0.7)	0.3 (0.6)	0.5 (0.7)
B-2	0.5 (0.7)	0.2 (0.3)	0.4 (0.6)

be small and localized only in a few slices. If the isovalue corresponds to the bones instead, the isosurface will be large and nearly homogeneously distributed.

7.1. Workflow I - End Position

The simplest situation is represented by the execution of the isosurface extraction as the last operation of the pipeline. This case is equivalent to the execution of the isosurface as a stand-alone operation, because we do not take care of the requirement of the following operations.

This means that for the selection we can disregard all the aspects related to the partitioning efficiency and quality, because no other operation will exploit the produced mesh partitioning, and we only examine the performance of the algorithms that we presented in Figures 4, 5 and 6. We focus our attention in particular on the efficiency values obtained for the four kinds of isosurfaces. These values are shown in Table VI.

We can see that the performance of FD and LBC are nearly the same. This is due to the high performance of the I/O system and the Gigabit network of our cluster, otherwise the performance of FD may be reduced due to the I/O and communication overheads; on the contrary, the high overheads of LBAC make its selection unsuitable, except for very large isosurfaces as XT-25. We may conclude that FD represents the best choice if the homogeneity degree is low, because its partitioning strategy ensures an optimal load balancing of the workload among the parallel processing, whereas when the homogeneity degree is high, LBC has to be preferred.

7.2. Workflow II - Data Streaming

In certain cases the volumetric dataset may not be completely available at the beginning of the isosurface extraction operation. For example, users may require to visualize the isosurface representing the shape of an object during the effective creation of volumetric data, which results from a complex simulation or from the acquisition using equipments or sensors and that requires a considerable amount of time to be produced.

In these cases the choice is practically fixed and it is represented by the FD algorithm, which is able to subdivide the available data, into small parts and to dynamically assign them to the workers.

7.3. Workflows III and IV - Data Partitioning Quality

The last two situations concern the execution of the isosurface extraction operation as an intermediate step of a pipeline of operations. Therefore we have to base the choice, beside the temporal efficiency, also on the partitioning efficiency and quality of the parallel algorithms.

In both the workflows the main objective is to ensure the balancing of the triangles among the parallel processes executing the following step of the pipeline. The difference between workflows III and IV is represented instead by the need to assure or not also the coherence of the data, to reduce as much as possible the number of shared vertices.

The two situations may be represented by considering a pipeline made up by the isosurface extraction and the compression operation for the workflow III, by the isosurface extraction and the simplification for the workflow IV. We refer for the discussion to parallel algorithms for the the compression operation [51] and the simplification operation [50] that we developed.

Parallel Compression Operation

We proposed a parallel compression algorithm in which each part of the isosurface can be encoded independently from the others, based on Edgebreaker [52]. We experimentally evaluated that the size of the results obtained with the parallel algorithm increases of almost 10% with respect to that obtained by the sequential version, but the achieved speed up overcomes the greater time necessary to transfer this larger file.

For example, we considered the case XT-52: the size of the uncompressed isosurface is of 374 MB; with the sequential compression we can reduce it to 31 MB, and with the parallel compression using 16 processes the resulting file has a size of 32 MB. We transmitted the data on a geographic network connecting our institute in Genoa with the Institute for Biotechnologies of CNR in Milan. The bandwidth is of about 1.14 Mbps and the client is a PC with characteristics similar to our cluster nodes. The measured total time to transmit the uncompressed file was 44min:37sec., 17min:36sec. considering the sequential compression and the transmission of the resulting file of 31 MB, while only 4min:31sec. with the parallel compression and the transmission of the resulting file of 32 MB. For this reason we can say that with this kind of operations it is important to ensure the data balancing, because the cost to process each triangle is a constant time, while it is possible to disregard the problem of data coherency, because it has only a small impact on the quality of results.

Parallel Simplification Operation

The situation is different if we consider the simplification operation. The aim of this operation is to reduce the number of triangles, and consequently vertices and edges of the original mesh, producing a new mesh representing a good approximation of it. We considered the Garland-Heckbert sequential simplification algorithm [53], one of the best solution to obtain high quality results. It is based on an iterative process where, for each iteration, the triangle introducing the minimum approximation error is globally selected and deleted.

The main problem in the design of a parallel version of this algorithm is that its structure does allow an easy achievement of a good speed up maintaining high quality results. However, there are two main problems: *the necessity of a global selection of the edges to be collapsed among the parallel processes* and *the necessity of accessing non-local data, in the case of shared triangles.*

With respect to the first problem, if there is not an overall selection and consequently the same amount of simplification is performed independently by each parallel process on each part, the quality of the resulting mesh will be very poor. If we consider a terrain, triangles belonging to the less regular parts of the meshes as mountains should be preserved, while large planar regions can be hardly simplified with a very little degradation.

Regarding the second one, the evaluation and the possible deletion of shared triangles involves the collaboration of the two processes sharing it; therefore there is an increase of communications and synchronization points, which lower the performance. In the current parallel solution we disregard the shared triangles avoiding to collapse them, obtaining however a very good compromise between the achieved speed up and the production of high quality simplified meshes at least until the deletion of 90% of the triangles.

Better results can be obtained if the number of shared triangles is very little. For this reason, we may say that with this kind of operations it is important to ensure also the data coherency, because the quality of the result and the efficiency of the computation are inversely proportional to the number of shared data.

From these considerations we derived the following guidelines.

Guidelines for Workflows III and IV

The use of FD is suitable for *workflow III*. In fact, besides the high speed up values, the FD partitioning is sufficiently balanced and the high number of shared triangles produced do not have a considerable importance. If the isosurface is homogeneous also LBC may be selected. LBAC instead ensures the production of an optimally balanced partitioning in the cases, but its performance in general suffers from high overheads.

Instead for the *workflow IV* FD does not represent a good solution because of the great importance of data coherency.

However, considering that the FD algorithm produces an overpartitioned isosurface, that is composed by nT parts, we can implement a rebalancing step based on the merging of contiguous parts [16]. This solution can be applied only if the required parallelism degree is lower than nT but, if the task subdivision is sufficiently fine, it is possible to achieve about the same partitioning of LBAC. It is worth noticing that we do not know for certain that this operation reduces the number of shared vertices, but this occurred in the experiments we made, and this condition holds in most cases.

The execution of FD followed by the rebalancing, denoted as FD+R, is more expensive than LBC. The rebalancing is fast, but still remains an overhead that overcomes the difference among the execution times of the two algorithms when FD is faster than LBC. Therefore we may conclude that LBC represents the best choice if the homogeneity degree is high, because in these cases its partitioning strategy guarantees good performance, an optimal mesh partitioning and probabilistically less shared vertices than FD.

Table VII. A comparison among the performance of LBAC and FD followed by the rebalancing step (indicated with "R") when performed in pipeline with an operation where the data coherence is important, as the parallel simplification. We considered the two kinds of non-homogeneous isosurfaces and we used 16 nodes.

Dataset	Time (sec)			Speed Up (Efficiency)			Time (sec)	
	LBAC	FD	R	LBAC	FD	FD+R	LBAC+Simpl.	FD+R+Simpl.
B-2	0.8	0.4	1.5	3.4 (0.2)	6.9 (0.4)	1.4 (0.1)	14.2	15.9
FC-1000	13.5	7.8	6.8	6.2 (0.4)	10.7 (0.7)	5.7 (0.3)	47.5	50.7

Instead, when the homogeneity degree is low, we have to examine if the execution of FD+R is faster than LBAC, considering that they produce about the same partitioning and amount of shared vertices. We experimented the performance of FD+R for the non-homogeneous isosurfaces using 16 nodes. In Table VII the total execution times, the speed up and the efficiency values are presented and compared with the performance of LBAC.

We can see that, even if the rebalancing is a fast operation, LBAC achieves the best performance. Therefore we may conclude that if the isosurface extraction is executed within a workflow where the data coherency is a major requirement, the LBAC algorithm ensures a balanced distribution of the triangles among the parallel processes for the following steps with the higher speed up. However, when the homogeneity degree is high, the LBC component has to be preferred because it is able to produce the same partitioning in a faster way.

A summary of the guidelines for all the Workflows

To summarize, the guidelines for selecting the most suitable parallel algorithm considering all the workflow configurations and the kind of isosurfaces are depicted in Figure 8.

We stress again the fact that if the user knows in advance the characteristics of the resulting isosurface, he/she may specify it, in order to allow the selection of the most suitable parallel algorithm. However, as mentioned above, this information is often not available. In these cases we decided to consider the isosurface as non-homogeneous.

8. CONCLUSIONS AND FUTURE WORKS

In this paper we describe three different parallelization strategies for the isosurface extraction operation based on the Marching Cubes algorithm. We analyse these algorithms considering their temporal and partitioning efficiency to determine which of them is able to achieve the best performance figures, considering both the execution as stand-alone programs and their composition in a workflow like ours.

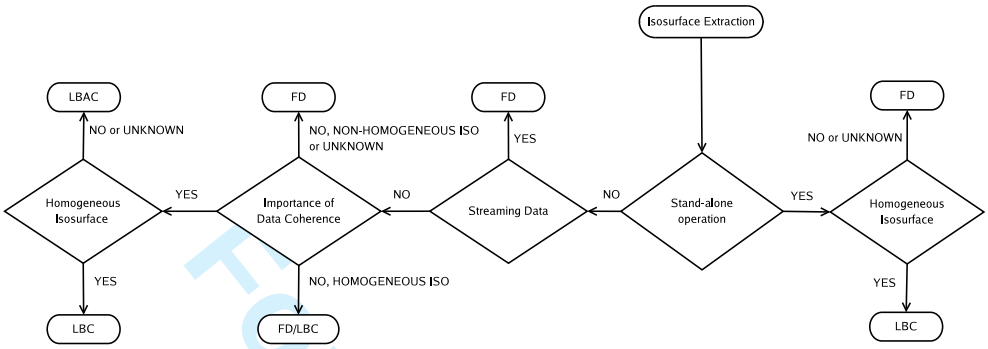


Figure 8. Selection of the parallel isosurface extraction algorithm on the basis of the workflow configurations and the kind of isosurfaces.

The experimental results show that none of them represent the optimal solution in arbitrary case.

In general the FD algorithm achieves the best performance if it is executed as a stand-alone program or as the last stage of a workflow. On the contrary, if a balanced partitioning of the resulting isosurface is required by the next stages of a workflow, the best overall performances are obtained using LBAC or, in certain cases, LBC.

Considering the future works, we plan to develop this work in two directions: we will analyse thoroughly the possibility to reduce I/O overheads by exploiting the specialized routines provided by MPI2 and to improve the partitioning strategy by designing an ad hoc solution which takes into account also the shared vertices problem.

The second aspect is represented by the experimentation of high level tools to simplify the composition of our algorithms with other modules. Presently we produce a library which has to be linked with an application written in C and MPI to be utilized. The goal is to provide the algorithms as a set of *components* that can be dynamically assembled with other heterogeneous components at runtime. For this purpose we have already done some experiments using software development frameworks that should be well suited for Grid aware software development and deployment. In particular, we have considered the use of ASSIST [54], a high level structured parallel programming environment, and Ccaffeine [55], a Common Component Architecture (CCA) compliant framework for parallel computing.

REFERENCES

1. Lorensen WE, Cline HE. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of SIGGRAPH 87)* 1987; **21**(4): 163–169.
2. The Visualization Toolkit (VTK). http://public.kitware.com/VTK/
3. Advanced Visual Systems (AVS). http://www.avs.com/

4. Clematis A, D'Agostino D. Parallel Remote Visualization for the Grid. In *Engineering the Grid: Status and Perspective,* Dongarra, Zima, Hoisie, Yang, Di Martino (eds.). American Scientific Publishers, 2006; 521–536.
5. ParaView: Parallel Visualization Application. http://www.paraview.org/HTML/Index.html
6. VisIt Visualization Tool. https://wci.llnl.gov/codes/visit/
7. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI). http://software.sci.utah.edu/scirun.html
8. Freire J, Silva CT, Callahan SP, Santos E, Scheidegger CE, Vo HT. Managing Rapidly-Evolving Scientific Workflows. In *Proceedings of the 2006 International Provenance and Annotation Workshop (LNCS, vol. 4145)*. Springer, 2006; 1018.
9. COVISE: A New approach to Distributed Integrated Product Development. http://www.hlrs.de/organization/vis/covise/executive_summary/
10. IRIS Explorer ™. http://www.nag.co.uk/welcome_iec.asp
11. Heinzlreiter P, Kranzlmüller D. Visualization Services on the Grid: The Grid Visualization Kernel. *Parallel Processing Letters* 2003; **13**(2): 135–148.
12. The Globus Toolkit. http://www.globus.org/
13. Beynon M, Chang C, Catalyurek UV, Kurc TM, Sussman A, Andrade H, Ferreira RA, Saltz JH. Processing Large-Scale Multidimensional Data in Parallel and Distributed Environments. *Parallel Computing, special issue on Parallel data-intensive algorithms and applications* 2002; **28**(5): 827–859.
14. Shalf J, Bethel EW. The grid and future visualization system architectures. *Computer Graphics and Applications* 2003; **23**(2): 6–9.
15. D'Agostino D. Designing Parallel Programs for 3D Data Processing on Distributed Architectures. PhD Thesis, DISI-TH-2006-03, Università di Genova, 2006.
16. Clematis A, D'Agostino D, Gianuzzi V. Load Balancing and Computing Strategies in Pipeline Optimization for Parallel Visualization of 3D Irregular Meshes. In *Proceedings of the 12th European PVM/MPI Users' Group Meeting (Euro PVM MPI 2005), (LNCS, vol. 3666)*. Springer, 2005; 457–466.
17. Merelli I, Orro A, D'Agostino D, Clematis A, Milanesi L. A Parallel Protein Surface Reconstruction System. *International Journal of Bioinformatics Research and Applications* 2008; **4**(3): 221–239.
18. Berman HM, Bhat TN, Bourne PE, Feng Z, Gilliland G, Weissig H, Westbrook J. The Protein Data Bank and the challenge of structural genomics. *Nature Structural Biology* 2000; **7**(11): 957–959.
19. Connolly ML. The molecular surface package, *J.Mol.Graphics* 1993; **11**(2): 139–141.
20. The Digital Imaging and Communications in Medicine standard (DICOM). ftp://medical.nema.org/medical/dicom/
21. The Object File Format (OFF). http://www.dcs.ed.ac.uk/home/mxr/gfx/3d/OFF.spec
22. The Virtual Reality Model Language (VRML). http://www.web3d.org/
23. Durst MJ. Additional Reference to Marching Cubes. *Computer Graphics* 1988; **22**(2): 72–73.
24. Wilhelms J, Van Gelder, A. Topological considerations in isosurface generation. Tech. Rep. UCSC-CRL-90-14, University of California, 1990.
25. Montani C, Scateni R, Scopigno R. A modified look-up table for implicit disambiguation of Marching Cubes. *The Visual Computer* 1994; **10**(6): 353–355.
26. Bajaj CL, Pascucci V. Accelerated IsoContouring of Scalar Fields. *Data Visualization Techniques*, Bajaj CL (ed.). John Wiley & Sons Ltd, 1998.
27. Watt A, Watt M. *Advanced Animation and Rendering Techniques Theory and Practice*. Addison-Wesley/ACM Press, 1992.
28. Livnat Y, Shen HW, Johnson CR. A near optimal isosurface extraction algorithm for unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 1996; **2**(1): 73–84.
29. Cignoni P, Montani C, Puppo E, Scopigno R. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics* 1997; **3**(2): 158–170.
30. Wilhelms J, Gelder AV. Octrees for faster isosurface generation. *ACM Transactions on Graphics* 1992; **11**(3): 201–227.
31. Itoh T, Yamaguchi Y, Koyamada K. Volume thinning for automatic isosurface propagation. In *Proceedings of IEEE Visualization '96*. IEEE Computer Society, 1996; 303–310.
32. van Kreveld M, van Oostrum R, Bajaj CL, Pascucci V, Schikore DR. Contour trees and small seed sets for isosurface traversal. In *13th ACM Symposium on Computational Geometry*. ACM, 1997; 210-220.
33. Carr H, Snoeyink J. Path seeds and flexible isosurfaces using topology for exploratory visualization. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*. Eurographics Association, 2003; 49–58.

34. Saupe D, Toelke J. Optimal Memory Constrained Isosurface Extraction. In *Proceedings of the Vision Modeling and Visualization conference*, 2001; 351–358.
35. Sutton P, Hansen C, Shen H, Schikore D. A Case Study of Isosurface Extraction Algorithm Performance. In *Data Visualization 2000*. Springer, 2000, 259–268.
36. Ahrens J, Law C, Schroeder W, Martin K, Papka M. A parallel approach for efficiently visualizing extremely large,time-varying data sets. Tech. Rep. LAUR-001630, Los Alamos National Laboratory, 2000.
37. Lombeyda S, Aivazis M, Rajan M. Parallel Isosurface Calculation and Rendering of Large Datasets in IRIS Explorer. In *Proceedings of the Visualization Development Environments meeting 2000*.
38. Chiang YJ, Farias R, Silva CT, Wei B. A unified infrastructure for parallel out-of-core isosurface and volume rendering of unstructured grids. In *Proceedings of IEEE Parallel Visualization and Graphics Symposium*. IEEE Computer Society, 2001; 59–66.
39. Zhang X, Bajaj C, Blanke W, Fussell D. Scalable isosurface visualization of massive datasets on COTS clusters. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*. IEEE Computer Society, 2001; 51-58.
40. Shen HW, Hansen CD, Livnat Y, Johnson CR. Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *Proceedings of Visualization '96*. IEEE Computer Society, 1996; 287–294.
41. Hanson C, Hinker P. Massively parallel isosurface extraction. In *Proceedings of Visualization '92*. IEEE Computer Society, 1992; 77–83.
42. Gao J, Shen HW. Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *Proceedings of IEEE Parallel Visualization and Graphics Symposium*. IEEE Computer Society, 2001; 64–74.
43. Miguet S, Nicod JM. A load-balanced parallel implementation of the marching-cubes algorithm. Tech. Rep. 95-24, Ecole Normale Superieure de Lyon, 1995.
44. Zhang X, Bajaj C, Ramachandran V. Parallel and out-of-core view-dependent isocontour visualization using random data distribution. In *Proceedings of Joint Eurographics-IEEE TCVG Symposium on Visualization*, 2002; 1–10.
45. Gropp W, Lusk E, Doss N, Skjellum A. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 1996; **22**(6): 789–828.
46. Ligon III WB, Ross RB. PVFS: Parallel Virtual File System. *Beowulf Cluster Computing with Linux*, Sterling T (ed.). MIT Press, 2001; 391–430.
47. Karypis G, Kumar V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 1998; **20**(1): 359–392.
48. The Christmas tree data set. http://www.cg.tuwien.ac.at/xmas/
49. The Visible Human Project. http://www.nlm.nih.gov/research/visible/visible_human.html
50. Clematis A, D'Agostino D, Gianuzzi V, Mancini M. Parallel Decimation of 3D Meshes for Efficient Web based Isosurface Extraction. In *Parallel Computing: Software, Technology, Algorithms, Architecture & Applications, Advances in Parallel Computing Series*, 2004; **13**: 159–166.
51. Clematis A, D'Agostino D, Gianuzzi V. Parallel Compression of 3D Meshes for Efficient Distributed Visualization. In *Parallel Computing: Current & Future Issues of High-End Computing*, *John von Neumann Institute for Computing Series*, 2005; **33**: 655–662.
52. J. Rossignac: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 1999; **5**(1), 47–61.
53. M. Garland, P. S. Heckbert: Surface Simplification Using Quadric Error Metrics. *Computer Graphics*, 1997; **31**: 209–216.
54. Vanneschi M. The programming model of ASSIST, an environment for parallel and distributed portable applications. *Parallel Computing*, 2002; **28**(12): 1709–1732.
55. Allan BA, Armstrong RC, Wolfe AP, Ray J, Bernholdt DE, Kohl JA. The CCA Core Specification in a Distributed Memory SPMD Framework. *Concurrency and Computation: Practice and Experience*, 2002; **14**(5): 323–345.