

# Parallelizing an Unstructured Grid Generator with a Space-Filling Curve Approach

Jörn Behrens<sup>1</sup> and Jens Zimmermann<sup>2</sup>

<sup>1</sup> Munich University of Technology,  
D-80290 München, Germany,  
Internet: [www-m3.ma.tum.de/m3/behrens/](http://www-m3.ma.tum.de/m3/behrens/),  
Email: [behrens@mathematik.tu-muenchen.de](mailto:behrens@mathematik.tu-muenchen.de)

<sup>2</sup> Ludwig-Maximilians-Universität,  
D-80539 München, Germany,  
Email: [jens.zimmermann@physik.uni-muenchen.de](mailto:jens.zimmermann@physik.uni-muenchen.de)

**Abstract.** A new parallel load distribution algorithm for unstructured parallel grid generation is presented. This new approach is based on a space-filling curve. The space-filling curve's indices are calculated recursively and in parallel, thus leading to a very efficient, and fast load distribution. The partitions resulting from this distribution are simply connected and the processor boundaries are sufficiently short.

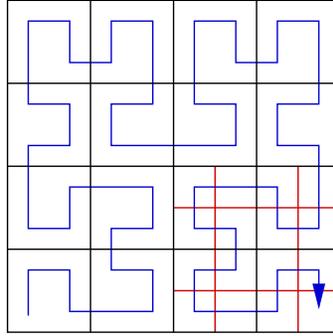
## 1 Introduction

Adaptive unstructured grid generation poses a nontrivial problem to parallelization approaches. However, a good parallel unstructured grid generator is desperately needed in several disciplines of scientific computing.

In [1] grid generation has been used in its serial form, because at that time no adequate parallel grid generator was available, that could be used for dynamic time-dependent flow problems. A new parallel hierarchical adaptive grid generator (`amatos`: Adaptive mesh Generator for Atmospheric and Oceanic Simulations) has been implemented, based on modular software techniques. `pamatos` is the name of the parallel implementation of `amatos`. `pamatos` can generate (i.e. refine, coarse, and adapt corresponding to a given error criterion) dynamically changing grids in two space dimensions. It is available as a software library, providing a simple programming interface (API). The grid generator's API remains almost unchanged for the parallel and serial versions respectively. Parallelization is based on the message passing paradigm.

Besides the nontrivial problems associated to the data management in unstructured grid generation on parallel computers, one major problem is the partition of the mesh. Data partitioning is required to be parallelizable, in order not to become a bottleneck for the whole grid generation process. Several authors have proposed different approaches, c.f. [4, 7, 10]. However, most of these methods require rather complicated calculations on the graph of the mesh.

Our approach follows a different line, as it is based on the geometry of the mesh items. We modify an algorithm for constructing a space-filling curve (SFC), originally



**Fig. 1.** Hilbert's space-filling curve for an adaptively refined rectangular grid

proposed by Hilbert for proving a set theoretical problem [6]. A space-filling curve passes through all points of a multi-dimensional domain, thereby, mapping the multi-dimensional index space to a one-dimensional set. This allows to find a unique enumeration of all elements of the unstructured mesh. Numbering the nodes of the curve (i.e. the elements of the mesh) consecutively, one can easily obtain a distribution of the mesh with almost optimal load balancing.

Roberts et al. [9] proposed several space-filling curve mechanisms for key generation in a hash table based mesh. Their bit-manipulation based algorithm also leads to a natural decomposition of the mesh. Griebel and Zumbush [5] introduced space-filling curves to sparse grid adaptive methods. However, both approaches do not easily extend to non-rectangular domains. Here we propose a modified space-filling curve approach for distributing a hierarchical triangular grid to several processors. We extend the approach to cover more complicated shapes of domains.

In the following section we give a detailed description of the recursive calculation of the space-filling curve and some modifications for non-rectangular domains. In section 3 we describe the grid generator which has been tailored for advection problems in meteorological and oceanic applications. Section 4 provides some numerical results and a conclusion is given in section 5.

## 2 Recursive calculation of the space-filling curve for triangle bisection

For two-dimensional rectangular domains Hilbert's space-filling curve has the shape shown in figure 1. A recursive construction mechanism is given, for example, by Breinholt and Schierz [3]. There are a lot of different forms of space-filling curves for different purposes. Lafruit and Cornelis [8] use so called dove-tail space-filling curves for the parallelization of fast wavelet transforms.

Figure 1 shows a space-filling curve for a rectangular grid. The space-filling curve is not really constructed, but the index of each cell is calculated with a recursive algorithm. When each cell is indexed, the index set is ordered consecutively. Now, a distribution

**Table 1.** Table of states for the space-filling curve algorithm.

original state	state description variables			resulting states	
	bisection vector	direction of marked edge	direction of SFC segment	new state $i$	new state $ii$
0	d	b	+	5, add	3
1	d	b	-	2	4, add
2	a	c	+	7, add	1
3	a	c	-	0	6, add
4	c	a	+	1, add	6
5	c	a	-	7	0, add
6	b	d	+	3, add	4
7	b	d	-	5	2, add

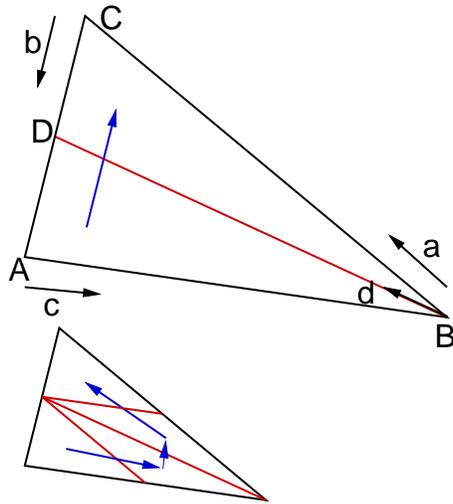
to the processors is easy: each processor receives an equally sized chunk of ascending indices.

Nearly optimal load balancing can be achieved. The question is, how good this distribution is with respect to other measures like the partition boundary length, etc. Griebel and Zumbusch [5] showed as well as Roberts et al. [9] that the partitions obtained by space-filling curve approaches are in deed well behaved. Our results in section 4 support this observation.

The algorithm for finding an index for a triangle of the finest grid depends on three basic properties of our grid. First, the procedure mimics the bisection of triangles. Second, the algorithm uses the center coordinates of each triangle as nodes for the space-filling curve. Finally, in order to find new indices, a tabulated indexing scheme is used (see table 1). There are three variables describing the state, and corresponding to that original state, two new resulting states. Each triangle of a given macro triangulation has state 1 by definition.

To find the index for a refined triangle, we have first to describe the coarse triangles state in detail. Figure 2 shows a triangle  $\overline{ABC}$  in state 1, with an edge marked for bisection  $\overline{CA}$ . Vectors  $a$ ,  $b$ ,  $c$ , and  $d$ , denote different directions, used to describe the state. The triangle is divided by a line defined by  $d$  (red line in figure 2). The blue arrow indicates the direction of the SFC.

Let us assume, we were interested in the new index of triangle  $\overline{DBC}$ . We have the following information on the state of the original triangle: The SFC's direction is oriented from  $A$  to  $C$ , thus vector  $b$  with negative sign defines the direction of the SFC, and vector  $d$  bisects the triangle. Reading these values in the table, we find that triangle  $\overline{ABC}$  is in state 1. Finally, as triangle  $\overline{DBC}$ 's center is in the opposite direction as the marked edge's orientation, we have new state  $ii$  as our new value. In some cases, the original index of the triangle has to be updated by an appropriate offset, indicated by an "add" in the table. The offset consists of the number of SFC-nodes "upstream" to the current position. Addition has to take place in cases where the current position is "downstream" in the SFC. Figure 3 shows the corresponding space-filling curve for a locally refined triangular grid.

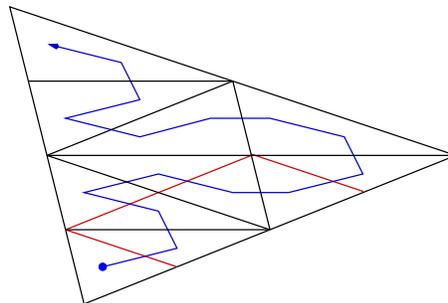


**Fig. 2.** Denotation of a triangle for the space-filling curve algorithm

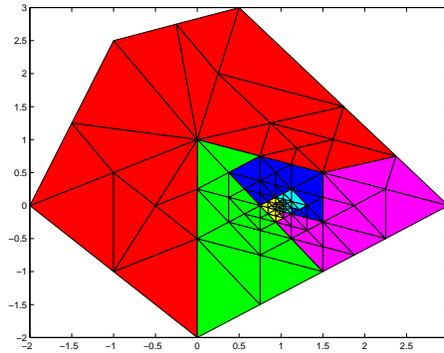
Table 1 contains all possible states. Applying the above given algorithm recursively, yields a unique index for each triangle on the finest level of triangulation.

When using a hierarchical grid generator (as in our case), we start with a coarse macro triangulation. Utilizing the above mentioned algorithm for each triangle of the macro triangulation, adding an offset to the indices of each macro triangle, and supposing that the children of each macro triangle will not be deformed too much, this space-filling curve can be used for non-rectangular domains. An example is given in figure 4.

The given algorithm can be executed to an arbitrary level of refinements recursively. By adding the corresponding state numbers to the index, we can obtain a unique index for each triangle of the mesh. This can be done in parallel for each triangle of the finest triangulation. Sorting of the indices can be done in parallel with a parallel sorting algo-



**Fig. 3.** Space-filling curve in a locally refined triangular grid



**Fig. 4.** Distribution of a locally refined triangulation over a non-rectangular domain to 6 processors

rithm. However, in our implementation this step is still serial. Given a consecutive list of indices, each processor can now decide independently which of its triangles have to be moved to which processors. Thus in principle the distribution can be done completely parallel.

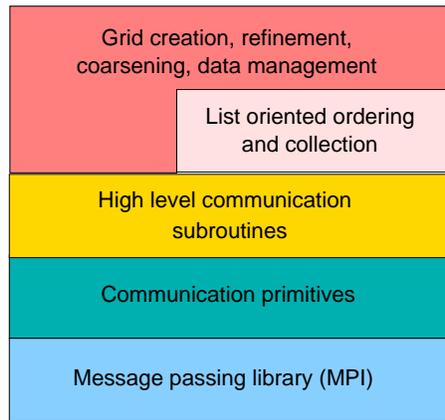
**Algorithm 1 (Recursive space-filling curve distribution)**

1. Calculate indices for each triangle of the finest level
2. Sort global index set
3. Cut global index set into chunks of equal size
4. Move all triangles which do not belong to the own chunk

### 3 The parallel grid generator

This section will give a brief description of the parallel mesh generator `pamatos`, using the new distribution method. `pamatos` is a two-dimensional grid generator for atmospheric and oceanic simulations. It implements a bisection refinement technique for the elements. `pamatos` is implemented in Fortran 90 utilizing a modular object oriented programming paradigm. A relatively small number of interface routines controls the behaviour of the grid. There is a serial sister of `pamatos`, called `amatos`, that has been used in atmospheric simulations in [2]. In principle, the programming interface for the serial and the parallel mesh generator are similar, so the application programmer does not need to care about parallelization aspects. It is the task of `pamatos`, to distribute the work load evenly among the processors, to provide the necessary communication, etc.

The internal structure of `pamatos` is characterized by a hierarchy of different software layers (see figure 5). Communication primitives build on top of a standard message passing library. As `pamatos` is in an early state of development, only MPI is available, but it is easy to exchange it by PVM, or other message passing libraries. The communication primitives layer hides the system dependent software parts away from the actual grid generator. On top of these, high level communication subroutines do the



**Fig. 5.** Software layers in the parallel implementation of the grid generator

work required for movement and information update in an unstructured grid. High level communication subroutines are engaged both by the grid generation layer and by the list layer.

The list layer facilitates programming in an unstructured grid generation process. An abstract procedure can be given as follows:

**Algorithm 2 (Abstract list oriented procedure)**

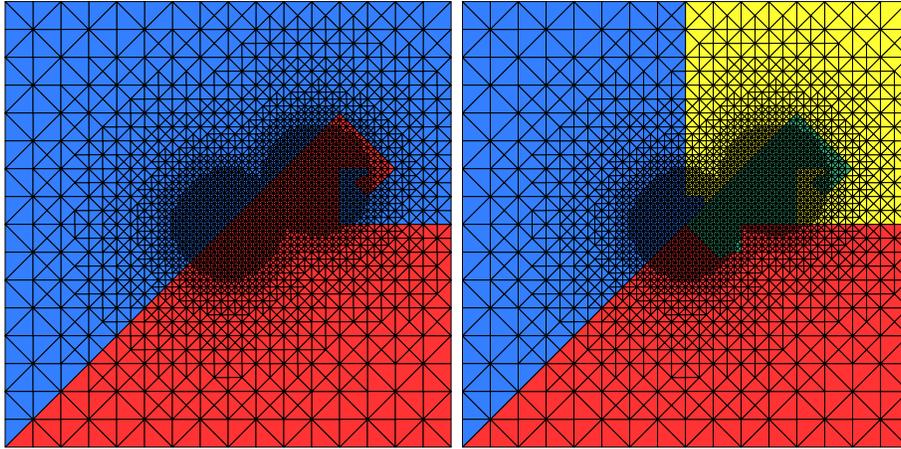
1. *Do the normal serial work until a data item is encountered that resides on a remote processor*
2. *Skip the part of work, depending on non-available data item*
3. *Put the pointer to the required data item into the collection list and continue with serial work*
4. *If no serial work is left, communicate all data items in the collection list*
5. *resume work with (now available) data items*

With this mechanism, the methods developed for the serial grid generator have to be extended only by calls to the collection list and a communication step. Communication takes place seldomly and consists of long messages instead of short frequent updates. Data management corresponding to the update of grid connectivity can be hidden away from the programmer.

A typical adaptation step is given in algorithm 3. Note that all the steps can be performed in parallel, however, there are several synchronization points required. Efficiency can be achieved only, if the grid is distributed evenly among the processors.

**Algorithm 3 (Adaptation of the grid)**

1. *[in parallel] Refine those elements, flagged for refinement*
2. *[synchronization point] Exchange edge information required for creation of an admissible triangulation*
3. *[in parallel] Coarse those elements, flagged for coarsening*



**Fig. 6.** Locally refined mesh, used in the test case described in the text. The meshes consist of 5,946 elements, the finest level of refinement is 14. Distribution to two (left) and four processors (right) is shown.

**Table 2.** Execution times of the space-filling curve indexing.

no. of processors	2	3	4
time [s]	0.303	0.176	0.135

4. [synchronization point] *Exchange edge information required for creation of an admissible triangulation*
5. [in parallel] *Calculate new distribution*
6. [synchronization point] *Move data items to achieve load balancing, update connectivity information*

## 4 Numerical examples

In order to show some numerical results a test case has been chosen, that contains two regions of local refinement (see figure 6). The reported tests were conducted with 18 levels of refinement, leading to a grid of 16,468 elements (the figure shows only 14 levels of refinement for visual reasons). As these are very fresh results, we are only able to give numbers for up to four processors (tested on an SGI Origin 200, with four MIPS R10K processors with 225 MHz clock frequency).

We first want to show the scalability of the indexing algorithm. Indexing of 16,468 elements has been tested on two, three, and four processors. Times are given in table 2 and show a good scalability of the indexing. As the sorting step in our implementation is still serial, and takes most of the time of the whole algorithm, scalability measurements have to be added later.

Let us therefore look at the load balancing results. Table 3 gives some values. Because of the nature of the algorithm, we achieve optimal load balancing of the element loads. As edges and nodes are copied at partition boundaries, we cannot expect optimal load balancing of these grid items, however, the table shows that even these numbers do not diverge too much. Note also, that the partition boundaries are relatively short, compared with the total number of edges in each partition.

## 5 Conclusions

In this article we have introduced a new recursive space-filling curve algorithm for the dynamic distribution of adaptively refined triangular meshes to many processors. The recursive indexing algorithm proves to be fast and easily parallelizable.

The partitions resulting from our space-filling curve have nice properties. The space-filling curve does not require rectangular domains, so it is more flexible than most other algorithms. We still have the property that each element of the finest level has a unique index. The edge length of the partition boundaries does not grow too much, thus minimizing communication among neighboring processors.

However, some work remains to be done. The index sorting has to be parallelized, detailed measurements of the runtime, communication overhead, and scalability of the complete grid generator are needed.

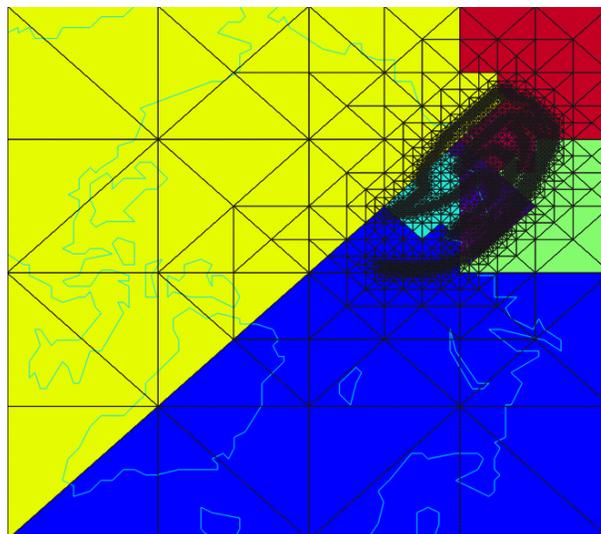
We intend to use the new parallel grid generator `pamatos` in our atmospheric simulations. Figure 7 shows a grid from the adaptive tracer simulations with colored partitions to eight processors. Again the partition domains show short boundaries, the partitioning is easily calculated, even though the grid is already quite complicated.

## References

1. J. Behrens. An adaptive semi-Lagrangian advection scheme and its parallelization. *Mon. Wea. Rev.*, 124(10):2386–2395, 1996.
2. J. Behrens, K. Dethloff, W. Hiller, and A. Rinke. Evolution of small scale filaments in an adaptive advection model for idealized tracer transport. Report TUM-M9812, Technische Universität München, Technische Universität München, Fakultät für Mathematik, D-80290 München, October 1998.

**Table 3.** Load balancing for a model problem grid for four, resp. two processors (percentage in brackets)

processor id	elements	edges	nodes	edges at part. bound.
1	4117 (25.0)	6256 (23.0)	2135 (22.1)	116
2	4117 (25.0)	8128 (29.9)	3090 (31.9)	175
3	4117 (25.0)	6575 (24.2)	2315 (23.9)	178
4	4117 (25.0)	6242 (22.9)	2126 (22.0)	109
1	8234 (50.0)	13717 (52.4)	4859 (53.4)	207
2	8234 (50.0)	12477 (47.6)	4244 (46.6)	207



**Fig. 7.** A refined grid from a meteorological application, distributed to eight processors

3. G. Breinholt and C. Schierz. Algorithm 781: Generating hilbert's space-filling curve by recursion. *AMS Trans. Math. Softw.*, 24:184–189, 1998.
4. R. Diekmann, D. Meyer, and B. Monien. Parallel decomposition of unstructured FEM-meshes. In *Proceedings of IRREGULAR 95*, volume 980 of *Lecture Notes in Computer Science*, pages 199–215. Springer-Verlag, 1995.
5. M. Griebel and G. Zumbusch. Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization. *Contemporary Mathematics*, 218:279–286, 1998.
6. D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Math. Ann.*, 38:459–460, 1891.
7. M. T. Jones and P. E. Plassmann. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In *Proceedings of the Scalable High Performance Computing Conference*, pages 478–485. IEEE Computer Society Press, 1994.
8. G. Lafruit and J. Cornelis. A space-filling curve image-scan for the parallelization of the two-dimensional fast wavelet transform. In *Proceedings of the 1995 IEEE Workshop on Nonlinear Signal and Image Processing*, <http://poseidon.csd.auth.gr/Workshop/>, 1995. Aristotle University of Thessaloniki, Aristotle University of Thessaloniki, Thessaloniki 540 06, P.O Box 451, Greece.
9. S. Roberts, S. Kalyanasundaram, M. Cardew-Hall, and W. Clarke. A key based parallel adaptive refinement technique for finite element methods. Technical report, Australian National University, Canberra, ACT 0200, Australia, 1997.
10. C. Walshaw, M. Cross, M. Everett, and S. Johnson. A parallelizable algorithm for partitioning unstructured meshes. In A. Ferreira and J. D. P. Rolim, editors, *Parallel Algorithms for Irregular Problems: State of the Art*, pages 25–46. Dordrecht, 1995. Kluwer Academic Publishers.