AMATOS – A FLEXIBLE ENGINE FOR ADAPTIVE GRID COMPUTATIONS

JÖRN BEHRENS

Abstract. amatos stands for Adaptive Mesh generator for Atmospheric and Oceanic Simulations. It is a software library that eases to a great extend the implementation and development of applications based on adaptive grid refinement. amatos does not provide any ready built method for solving PDEs as they occur in geophysical fluid dynamics, but it completely frees the developer from complicated data management tasks, usually connected to adaptive computations. This article describes the basic principles and simple examples of the functionality of amatos.

1. Introduction and basic notations

Adaptive modelling becomes more and more interesting for numerical simulation in geophysical, atmospheric and oceanic simulation. Recently, several project proposals have been made to include adaptive techniques into atmospheric models [6, 7]. There is also a number of research activities aiming at adaptive simulations [3, 9, 8, 4, 5, 1, 10].

A broad introduction of adaptive methods into atmospheric modelling, however, has not yet been taken place. One reason for the slow adoption is the cumbersome implementation of adaptive techniques. Complicated data management is the main keyword here. A second reason for the deferred acceptance of adaptive methods is the potential inefficiency on current high performance computing architectures. Indirect addressing is connected to this argument.

amatos provides measures to overcome both of the above mentioned obstacles. It implements a Fortran 95 module for adaptive computations in vectors. Thus, by using a simple programming paradigm, namely vector oriented data management, optimization of numerical code on different types of computer architectures is supported. By providing several methods for data handling and manipulation, an easy development of diverse numerical integration schemes is possible.

There are two philosophies which are key to the understanding of amatos:

1991 Mathematics Subject Classification. 65M50, 65Y05, 68U20.

Key words and phrases. adaptive grid generation, finite elements, partial differential equations, time-dependent problems, software library.
Think of the adaptive algorithm as a two phase procedure: In the first phase, the mesh is generated/adapted. Each mesh item keeps associated data. In the second phase, numerical calculations are performed. To achieve this, first gather all required data from mesh items into vectors, perform the calculations on vectors (utilizing consecutive storage positions for efficient pipelined or vectorized execution), and finally scatter the results back to mesh item storage positions. This is illustrated in Figure 1.

Think of the program as a data-flow, with methods acting and manipulating the data. A data structure (called grid.handle) represents a specific instance of the mesh. Methods (routines in amatos’ programming interface) act on the instance, manipulating it. Different methods can be applied to the mesh more or less independently.

The programming interface provides routines that implement methods in both of the above circumstances. Gathering (grid.getinfo) and scattering (grid.putinfo) accept the mesh handle without manipulating the mesh topology. Other methods, like grid.adapt alter the topology.

When performing numerical calculations on data gathered from the mesh into vectors, there MUST NOT be any change to the mesh topology, before scattering the results back to the mesh.

2. Principles and Methods in amatos

2.1. Mesh refinement technique. In principle, many different geometric mesh layouts are allowable. In practice, for two-dimensional computational domains two
different mesh types are prevalent: quadrilateral and triangular meshes. **amatos** in its current state supports triangular meshes.

Two refinement strategies are common in triangular mesh refinement: regular refinement and bisections (see figure 2). Regular refinement by construction guarantees self similarity of triangles constructed by refinement and therefore a lower bound on the inner angles of all triangles. This is a desirable property for numerical stability reasons. However, local regular refinement causes *hanging nodes* to occur in the mesh that have to be treated separately.

**amatos** uses a bisection refinement strategy, proposed by Rivara resp. Bänsch (Lit!). One advantage of this strategy is a simple recursive implementation. It also provides bounded inner angles and no hanging nodes. However this type of refinement also has some disadvantages, namely the number of neighboring nodes varies even in uniform refinements.

The bisection refinement strategy can be easily extended to three-dimensional domains, and a 3D α-Version of **amatos** is currently under development.

### 2.2. Data management

Data management within **amatos** is well hidden from the application programmer. **amatos** provides interface routines to access either a complete set of data items from the mesh or even single mesh items. The principle access pattern is comparable with a gather/scatter operation (fig ..). An application developer designs his scheme either point-wise, edge-wise or element-wise, i.e. computations are performed in a loop over all nodes, edges or elements. A simple finite difference scheme would probably loop through all nodes, while a finite element scheme would loop through elements and a finite volume scheme would calculate fluxes across edges.

An abstract algorithm of a computation with **amatos** would consist of three basic steps:

1. gather all required data from mesh
2. perform computations

![Figure 2. Triangular mesh refinement: a: regular, b: bisection](image)
scatter data back to mesh

While there seem to be two overhead steps in this algorithm, the computations can be carried out in vectors with high efficiency and optimized for either vector or cache based micro-processor architectures. Practical tests show the overhead corresponding to the gather/scatter to be less than one percent.

amatos can be seen as a container for irregularly structured data. It provides interfaces to access data in a structured fashion. In fact, by using space filling curves for ordering data items, the majority of data items that are adjacent to each other in the computational domain, are also close to each other in gathered vectors. Neighborhood relations are preserved in vectors, also easing parallelization.

Internally, amatos’ design is object oriented. There are three types of data items: nodes, edges and triangles (i.e. elements). A node is defined by its coordinates, while edges and triangles are defined by their corresponding node indices. Nodes have knowledge on their patch, i.e. the indices of the triangles they belong to. Nodes can carry different types of physical data, depending on the application, function representation, and initial configuration.

Edges know the triangle indices they separate. They have knowledge on their children and their parents (if they were constructed by refining an edge). Edges can also carry different types of data and even several unknowns, depending on the function representation (see subsection 2.3). Finally, triangles know their edges, their level of refinement, their state (refined, unrefined, or flagged for refinement/unrefinement). Triangles are oriented (counter-clock-wise) and an edge lies opposite of the corresponding node. Similar to edges, triangles can carry different types data for the physical computation.

This object oriented design facilitates parallelization, since all required data can move from one processor to another in one sweep, when redistribution of partitions has to be carried out.

In order to support time stepping schemes, each data item carries a time stamp. With the time stamp, different grids can be stored, reusing the same data items for several (time) stages of the simulation.

To access data objects efficiently, amatos allocates lists. There are lists for all nodes/edges/triangles of the mesh at current, past and future time. Additionally a list for all boundary nodes/edges is maintained.

2.3. Function representation. amatos supports finite element (FE) function representation of data. However, this feature has to be configured at compile time by the application developer. While the developers of amatos can not foresee the requirements of the application, it is the task of a user to provide the necessary data and definitions for higher order finite element representations.

The basic tool for defining FE in amatos is a signature data structure. The signature contains information on the number of unknowns per node/edge/triangle for the corresponding FE, a definition of positions of unknowns, the order of approximation, and an optional name. In a user modified module, the initialization of FE has to be performed. In principle, an unlimited number of different FE can be supported by amatos.

Once, FE have been specified, these data can be accessed by the application built on top of amatos. Note that amatos itself does not provide functionality for setting up stiffness/mass matrices, calculating basis functions, etc. This functionality has
2.4. Programming interface. The programming interface consists of approximately 25 routines or functions that allow to control mesh creation, termination, and adaptive refinement, saving and restoring meshes, retrieving and storing data, and performing numerical calculations on mesh items.

It is not in the scope of this article to explain every single interface routine, therefore the reader can refer to the documentation available for amatos [2].

3. A simple example: Adaptive triangulation of a sphere

In this section we want to give an example of the programming style induced by amatos. The aim is to create an adaptive grid on the sphere.

The initialization of amatos is depicted in figure 3. An application programmer has to use the module implementing amatos by calling use GRID_api. Data structures are initialized by the routine grid_initialize, while the refinement parameters are set by grid_setparameters. Finally an initial grid has to be created from a coarse mesh defined in a file. This is accomplished by grid_createinitial. Note that amatos is not a grid generator in the sense that it automatically generates a mesh for arbitrary domains but it refines a given coarse mesh.

In order to work with physical data in amatos, we need to initialize for example, a scalar (tracer) field. The p_grid data structure provided by amatos holds information on the size of the grid, namely the number of nodes that will store the data. The code fragment in figure 4 shows how data can be calculated, depending on the coordinate. While coordinate values are retrieved from the mesh with grid_getinfo, newly calculated data are stored back to the mesh by grid_putinfo.

The main part of an adaptive program is the adaptive loop, shown in figure 5. In the adaptive loop, the first step is to calculate the new values (in this case the cosine hill shaped scalar field, in a time-stepping scheme, the time integration takes this task). The second step is the calculation of an appropriate error criterion. This is done by slm_errorest here and we do not describe the function in detail. One could use the gradient of the scalar field as a refinement criterion. In a third step, all those triangles that have a large error (i.e. above a certain threshold r_trsh) are marked for refinement. Finally, a call to grid_adapt makes amatos to refine the grid correspondingly, automatically taking care for hanging nodes and making sure to stay within the previously given refinement limits.

At the end of each amatos program, internal data structure have to be deallocated, and possibly a backup of the grid has to be saved. Figure 6 shows how to achieve that.

The short program described in this section creates a triangulation of the sphere shown in figure 7. The initial grid consists of 92 nodes and 180 triangles shaped like a football (or bucky ball).

References

PROGRAM gridtest

!---------- uses

USE IO_utils ! IO routines
USE SLM_errorestimate ! refinement criterion
USE GRID_api ! this is amatos

!---------- local declarations

IMPLICIT NONE

REAL, DIMENSION(:,), ALLOCATABLE :: r_trac
REAL, DIMENSION(:,,:), ALLOCATABLE :: r_cords
REAL, DIMENSION(GRID_dimension) :: r_xy, r_kcentr, r_grad
REAL, DIMENSION(GRID_dimspherical) :: r_lcentr

!---------- read command line options

CALL io_getcmdline(p_contr)

!---------- initialize grid generator

CALL grid_initialize

!---------- initialize grid parameters

CALL grid_setparameter(p_grid, i_coarselevel= p_contr%phy%i_crslevel, &
i_finelevel= p_contr%phy%i_reflevel)

!---------- create initial triangulation

CALL grid_createinitial(p_grid, c_filename=p_contr%io%c_triangfile)

Figure 3. amatos main program header

2. J. Behrens, amatos – Adaptive mesh generator for atmosphere and ocean simulation, Technische Universität München, TUM, Center for Mathematical Sciences, D-80290 Munich, Germany, 2002, API Documentation Version 1.2.
initialize a tracer (cosine hill)

\[
\begin{align*}
\text{r_lcentr} &= (-\text{GRID}\_\text{PI}*0.5, 0, 0) ! \text{the center in sph. coordinates} \\
\text{r_kcentr} &= \text{grid}\_\text{geokart}(\text{r_lcentr}) ! \text{the center in kart. coordinates}
\end{align*}
\]

\text{ALLOCATE}(\text{r_cords}(\text{GRID}\_\text{dimension}, \text{p}\_\text{grid}(\text{i}\_\text{timeplus})\%i\_\text{nnumber}))
\text{ALLOCATE}(\text{r_trac}(\text{p}\_\text{grid}(\text{i}\_\text{timeplus})\%i\_\text{nnumber}))

retrieve node coordinates

\text{CALL grid}\_\text{getinfo}(\text{p}\_\text{grid}(\text{i}\_\text{timeplus}), \text{p}\_\text{grid}(\text{i}\_\text{timeplus})\%i\_\text{nnumber}, & \text{r_nodecoordinates}= \text{r_cords})

loop over all nodes

\text{DO i_cnt=1, p_grid(i_timeplus)\%i_number}
\text{r_xy(():= r_cords(:,i_cnt) - r_kcentr(():
\text{r_tmp= DOT_PRODUCT(r_xy,r_xy)
\text{IF(r_tmp < r_rad) THEN}
\text{r_trac(i_cnt)= 400.*((1.+ \cos((\text{GRID}\_\text{PI}* r_tmp)/ r_rad))}
\text{ELSE}
\text{r_trac(i_cnt)= 0.}
\text{ENDIF}
\text{END IF}
\text{END DO}

write back nodal tracer values

\text{CALL grid}\_\text{putinfo}(\text{p}\_\text{grid}(\text{i}\_\text{timeplus}), \text{p}\_\text{grid}(\text{i}\_\text{timeplus})\%i\_\text{nnumber}, & \text{r_nodevalues}= \text{r_trac})
\text{DEALLOCATE(\text{r_trac}, \text{r_cords})}

\textbf{Figure 4. amatos} program fragment initializing a cosine hill shaped tracer


\text{Technische Universität München, Center for Mathematical Sciences, Boltzmannstr. 3, 85747 Garching, Germany} \\
\text{E-mail address: behrens@ma.tum.de}
!-------------- adaptation loop

adapt_loop: DO WHILE(p_grid(i_timeplus)%i_maxlvl < p_grid(i_timeplus)%i_reflvlbnd)

!--------------- calculate tracer (cosine hill) on new nodes

CALL cosine_hill(p_grid(i_timeplus))

!--------------- estimate error... reuse r_trac here

ALLOCATE(r_trac(p_grid(i_timeplus)%i_enumfine))
CALL slm_errorest(p_grid(i_timeplus), p_grid(i_timeplus)%i_enumfine, r_trac)

!--------------- mark some elements for refinement

ALLOCATE(i_mark(p_grid(i_timeplus)%i_enumfine))
i_mark = 0
DO i_cnt=1, p_grid(i_timeplus)%i_enumfine
   IF (r_trac(i_cnt) > r_trsh) THEN
      i_mark(i_cnt)= GRID_pleaserefine
   END IF
END DO

CALL grid_putinfo(p_grid(i_timeplus), p_grid(i_timeplus)%i_enumfine, 
   l_finelevel= .TRUE., i_elementstatus= i_mark)

DEALLOCATE(r_trac,i_mark)

!---------- adapt grid

CALL grid_adapt(p_grid(i_timeplus), l_grd)

END DO adapt_loop

Figure 5. amatos program fragment with the adaptive loop

!--------------- write a save set

write(c_mfile,*) 'amatos_saveset.dat'
CALL grid_writesaveset(c_mfile,p_grid)

!--------------- finish grid generator

CALL grid_terminate

END PROGRAM gridtest

Figure 6. End of amatos program
Figure 7. Triangulated sphere