Parallelization of a Multi-physics Code

William Dai, A.J. Scannapieco, Frederick Cochran, James Painter, Chong Chang
Los Alamos National Laboratory, Los Alamos, New Mexico, United States

Abstract - Roxane is a code recently developed at Los Alamos National Laboratory for multi-physics simulations. The physics capabilities include hydrodynamics, material mixing, radiation, magnetohydrodynamics, etc. This document is to present the main physics capability, basic data structures, parallelization, and IO. The focus will be on parallelization. The data structures include data layer-out and adaptive mesh refinement. Roxane is in the stage of active development, and some of material presented here will possibly be changed in the near future.

Keywords: numerical simulation, multiphysics, parallel

1 Introduction

Roxane is a package for multi-physics simulations on Eulerian grids. The physics includes hydrodynamics, advection, three-temperature (3-T) radiation diffusion, ideal and resistive magnetohydrodynamics with circuits, linear solvers, material mixing, and real equations of state. The code is mostly written in Fortran, and a small part in C language.

Simulations are performed through adaptive mesh refinement (AMR). Although there are other kinds of AMR, such as block-based AMR and patch-based AMR, Roxane uses cell-based AMR, in which numerical cells are refined cell by cell.

A typical simulation through Roxane involves many materials. To effectively use computer memory, material properties in mixing cells are stored as link lists, and only non-zero values of the properties are stored. Isotopes of any material are similarly compressed and stored.

Roxane is a parallel code run on massively parallel computers. The parallel capability was late added after most physics packages and data structures were developed. The design principle of the parallel capability was to minimize the number of communication during each time step and to minimize the modifications in physics packages we had to make for the parallel capability.

The IO capability in parallel environments is for restarting, visualization, onnection, and debugging, and it is built on top of MPI IO. Although Roxane is able to write arbitrary M files from N processors (N to M) for each dump event, currently N to 1 is the mode for users. The data in a restarting file are of two kinds, one for description of data, called attributes, for which processors have the same values, and the other for arrays that are distributed among processors. For visualization, meshes with AMR, unstructured meshes, and variables defined on the meshes are directly written into files.

To check the correctness of algorithms of physics packages and parallel implementation, a set of regression tests are run daily and weekly. Daily tests are for developers to check any modification of the codes and weekly ones are for more rigorous examinations.

2 Physics capability

In Roxane, hydrodynamics is described by the Euler equation. In our applications, physical viscosity may be ignored. Instead, in addition to numerical viscosity, artificial viscosity is typically introduced in calculations. Although the gamma law equation of state is included, Roxane uses real EOS tables to close the equations. Each of materials involved in simulations can have many isotopes. These materials are not necessarily in thermal equilibrium. Therefore, the mass density, pressure, energy, and flow velocity in the Euler equation are the collection of these materials.

Roxane accepts Cartesian, cylindrical, and spherical coordinates in one-, two-, and three-dimension. The hydro solver in Roxane is not sophisticated, so that various advanced mixing models could be easily incorporated into hydrodynamics. For a given mesh, all the variables are zonal, i.e., defined on volume. Roxane uses a dimension split technique to solve two- or three-dimensional hydrodynamics equation in a fixed grid. Therefore, one-dimensional solver is a building block for hydrodynamics. During each one-dimensional pass, momentum is defined on a staggered grid, called “panel”.

Each one-dimensional pass may be divided into two steps, flow motion and advection. The flow motion is due to initial flow velocity and acceleration caused by pressure gradient and external force. This step is simple in coding. Following the flow motion is advection, and advection is much more complicated in coding because of the nature of many materials and isotopes.

Before advection takes place, Roxane constructs material interface within any mixed cells based on the distribution of mass and volume of each material on each cell. Currently Young’s linear interface reconstruct procedure is used to handle materials in mixed cells, and this procedure is expected to be replaced by a modified interface reconstruct algorithm that will result in smoother interfaces between materials. The interface orientation in a mixed cell is determined by the
distribution of the material in neighboring cells, and the location of interface is calculated through volume fractions occupied by the materials within the cell.

The advection of materials and their isotopes making the hydrodynamics step a little more complicated. First, each material in each cell is advected through hydrodynamics and the reconstructed material distribution. Each isotope of a material is assumed uniformly distributed within the reconstructed volume of the material.

Magnetohydrodynamics in Roxane is mainly to model laboratory experiments that involve the interaction between MHD waves, diffusion of magnetic field in materials, and circuits in experimental devices.

Radiation in Roxane could be in the diffusion limit. Temperatures of radiation field, electrons, and ions are coupled and are allowed to be different from each other. Diffusion problems in Roxane, such as diffusion of magnetic field in MHD, diffusion of temperatures of radiation fields, electrons, and ions, result in large algebraic systems at each time step. These systems are nonlinear, and material properties involved in the system depend on solutions. The size of time step is such chosen that the material properties will not experience any significant change, and therefore, the material properties of the previous time step are used to calculate the solutions of the current time step.

The iterative conjugate gradient method converges slowly, and it is particularly slow for large linear systems. To increase the convergence, multigrid methods are used as preconditioner for the conjugate gradient method. Multigrid methods are particularly efficient because mathematical systems exhibit multiple scales of behavior, for which basic relaxation methods, such as conjugate gradient method, exhibit different rates of convergence for short- and long-wavelength components.

Since the diffusion problems in Roxane are described on unstructured meshes, the geometric multigrid is too difficult to implemented, an algebraic multigrid method is used. Unlike geometric multigrid method, algebraic multigrid method automates the coarsening process and interpolation without geometry information. Actually coarsening operators and interpolation can be derived directly from underlying matrices without any references to the grids.

3 Adaptive Mesh Refinement

The use of cell-based mesh refinement technique has quite long history in our institution. Cell-based AMR refines only those cells that are supposed to be refined. In cell-based AMR, cells, including the cells that are not refined, are treated cell by cell. The connectivity of meshes is described by connectivity arrays.

After a mesh of cell-based AMR is generated through the initial setup or refinement and coarsening during simulations, the connectivity between cells must be built before any calculation could precede, such as spatial derivatives of variables. There are several ways to establish the connectivity. Some take advantage of the nature of structured cells, but at the cost of much more memory footprint. Others describe the connectivity in the way for unstructured meshes.

Figure 1. Illustration for a radiation mesh. Different colors represent different materials.

To avoid unknown material properties in mixed cells, mixed cells in an AMR mesh are divided into a set of polygons through material interface reconstruction. Figure 1 shows an example of the “unstructured mesh”. In each of these polygons, the material is pure, and its material properties are known. Therefore, the linear solver is for a set of unknowns defined on these polygons, i.e., the diffusions or linear systems are solved on unstructured meshes. Although mixed cells are a small part of cells for many problems, all clean (unmixed) cells are treaded as unstructured cells in the linear solver.

Roxane takes advantage of the nature of structured cells but with very small memory cost. Two arrays, nlow(1:nzone,1:dim) and nhgh(1:nzone, 1:dim), are used for the purpose. Here nzone is the number of cells on my processor, including ghost cells (which will be discussed late),
and \( \text{dim} \) is the dimensionality of the problem. The value \( \text{nlow(nz,i)} \) gives the cell id of the lower side of the dimension \( i \). In the case of one-dimensional problem, this neighboring cell is uniquely determined. In the multi-dimensional case, \( \text{nlow(nz,i)} \) further refers to the cell that is at the lower end of any dimension other than the \( i \).th. Similarly, \( \text{nhgh(nz,i)} \) gives the neighboring cell of \( nz \) at the high end of the \( i \)th dimension, but at the lower end of any dimension other than the \( i \).th. This is illustrated in Fig.2.

The reason we could effectively use the concept of \( \text{nlow} \) and \( \text{nhgh} \) is that mesh is gradually refined, i.e., an immediate neighboring cell of any cell can only be one level higher or lower. This assumption is also guaranteed across processor interfaces. There are many methods to generate the connectivity arrays. \( \text{Roxane} \) uses a local KD-tree to set up \( \text{nlow} \) and \( \text{nhgh} \). Each identity in the KD-tree is a rectangular cell. After the array \( \text{nlow} \) and \( \text{nhgh} \) are setup, the tree is not needed anymore until the mesh is changed, typically after one time step. A mixing cell is always refined to the highest level. Furthermore, if any neighboring cell is a mixing or it has different material, this cell will be refined. To resolve shocks, the location of shock fronts is always refined to the highest level.

### 4 Data structure

\( \text{Roxane} \) has its own data structures, some are good, and some need to be improved. Some are good for traditional computer platforms, but not good for the (anticipated) future computers.

\( \text{Roxane} \) currently imposes two kinds of boundary condition, one is the reflection boundary condition, and the other is the continuation boundary condition. For the convenience of the code developers to write the code uniformly for both interior cells and cells on the domain boundaries, \( \text{Roxane} \) introduces one layer of boundary cells attached to each simulation boundary, and uses an array \( \text{irdx(nz)} \) to indicate whether each cell, \( nz \), is a boundary cell. The size of each boundary cell is the same as its corresponding interior cell.

As stated before, \( \text{Roxane} \) uses a linear list to identify cells, and each cell is specified by its center and widths in each dimension through two dimensional arrays, \( \text{xe(nz, dim)} \) and \( \text{dxe(nz, dim)} \). The connectivity between cells are described by two two-dimensional arrays, \( \text{nlow(nz, dim)} \) and \( \text{nhgh(nz, dim)} \). The values of \( \text{xe} \) and \( \text{dxe} \) may change after each time step, and \( \text{nlow} \) and \( \text{nhgh} \) are re-built after each time step.

Scalar variables defined on cells are described as one-dimensional array, such as \( \text{d(nz)} \). Flow velocity is defined on panels and described by two two-dimensional arrays \( \text{vl(nz, dim)} \) and \( \text{vh(nz, dim)} \).

We have to mention two points about the cells in \( \text{Roxane} \). First, every cell is an active cell, and there are no parent cells. Second, due to historic reason, cells in \( \text{Roxane} \) are sorted according to their levels. As the result of sorting, base cells, whose level is 0, are listed at the beginning in the list of \( nzone \) cells. After these cells are the cells with level 1, and so on.

To effectively use computer memory, \( \text{Roxane} \) stores only non-zero values of material properties, and these material properties include mass density fraction \( df \), volume fraction \( vf \), energy fraction \( ef \), material strength fraction, such as \( \text{txxf, tyxf, } \) magnetic field intensity fraction \( hf \), etc.

The integer array in \( \text{Roxane, } \text{ist} \), is used first to determine whether each cell is a clean cell (i.e., with one material) or a mixed cell (with more than one material). If \( \text{ist(nz)} \) is bigger than 0, then this cell is a clean cell, and the value is also id of the material. If for a particular cell \( nz \), \( \text{ist(nz)} \) gives a negative integer, this indicates a mixed cell, and the integer, \( \text{iax (} = - \text{ist(nz)} \) gives the index which is used in other arrays for the first material in this cell. For example, \( \text{dft(iax), vft(iax), and eft(iax) } \) gives the mass, volume, and internal energy fractions of the first material in the cell. For the second material in the cell and its material properties, the value of \( \text{nxtf(iax)} \) gives the index of the second material in the arrays \( df, vf, ef \). etc. This procedure could continue until \( \text{nxtf(iax)} \) gives 0.

Materials in \( \text{Roxane} \) may have different isotopes. The properties of these isotopes have to be stored in memory during calculations. Since it is possible to have many isotopes, but each of them spreads only over a fraction of simulation domain, \( \text{Roxane} \) stores only non-zero values of material properties of isotopes. Similar to compressed material data, such as \( df \) and \( vf \), the fractions of mass density of isotopes changes after advection in each dimensional pass.

### 5 Parallelization

Design and implementation of parallelization were introduced after the framework and most physics capabilities in \( \text{Roxane} \) were implemented and tested. Therefore, one constraint in parallelization is the minimum change in the framework and physics modules, so that authors of the framework and modules feel comfortable when further debugging or modifications on the framework and physics capabilities are needed.

The parallelization includes partitioning, identifying constructing ghost cells, communication, link from another code project, and comparison between serial and parallel calculations. The design principle of the parallelization includes the following three aspects: avoid arrays that has the global size, such as \( \text{map_global_to_local } \text{globalid} \), avoid the use of global id, and use less number of communications.

When a simulation runs on more than one processor, mesh and its associated variables are distributed among the processors. The initial mesh used in \( \text{Roxane} \) is generated through Rage, another code project at our institution. Although Rage and \( \text{Roxane} \) both deal with structured cell-based AMR, they order cells differently. As a result of the different ordering, the part of the initial mesh on each
processor may be disconnected in Roxane. Therefore, the first task in parallelization was to make the simulations correctly run on this disconnected partition.

At the early stage of development of parallel, for domain partition we intended to depend solely on the open software Zoltan, developed by Sandia National Laboratories. But, during the implementation of our parallel strategy, we also developed our own tool for domain partition. There were a couple of reasons for us to do this. First, even if we use Zoltan, we have to pack, unpack, implement communications, and reconstruct Roxane data structures, which is the majority work of domain partition. Second, a simple partition tool is desired for possible future modification of codes on future computers. There are several common approaches for domain partitions in Zoltan, and all the approaches work well for Roxane. The home-grown tool uses only bisection, which results in a rectangular domains for each processor.

To avoid additional communications, Roxane requires that all the cells generated from a same base cell (i.e., a cell whose level is 0), will be always on the same processors. This constraint eliminated the complexity and many communications. Otherwise, if an interface between processors is along with an interface between refined cells at one instant, the coarsening of the cells will be different from a serial calculation unless more communications are introduced. As expected, this constraint makes load balance imperfect, particularly when the number of base cells on each processor is small. Considering that any load balance in real problem depends on many factors, such as number of material, the number of isotopes of the materials on cells, and the load balance of physics packages imported from other codes, a perfect balance is almost impossible for real problems, unless other parallel strategies, such as task list, are used.

For domain partition, the following procedure is executed after each time step.

- Check load balance.
- If load is balanced, do nothing. Otherwise, do the following.
- Re-partition to determine the future owner of each cell.
- Create a set of buffers, pack coordinates, variables, and material data of those cells that will go to other processors, and send each of buffers to a targeting processor.
- Create another set of buffers to receive data from a set of processors for new cells.
- Unpack the buffers received from other processors add new cells into the remaining list of cells.
- Build link lists for material and isotope data for the cells coming from other processors.

Partition or re-partition deals with only real cells (contrary to ghost cells). After re-partition at the initial time or after one time step, ghost cells are to be included in each processor before any physics calculation.

Several considerations were in the design to build ghost cells. First, we allow any domain partition as an input, rectangular or non-rectangular domains, connected or disconnected domains, base-cell partition or others. Second, each processor communicates only to its neighboring processors. Third, at no instant, one processor owns more cells than those cell on its own domain, and no global arrays and indices are involved. Fourth, for the need of future numerical solvers, the number of layers of ghost cells is a parameter or an input of a subroutine (function) call. Last, to reduce the number of communication, we pack all the variables together before the only communication, including coordinates of ghost cells, the variables with different types (integer, float, etc), compressed material and isotope data, and link lists, etc.
For AMR in parallel environments, special care must be taken for the constraint of smooth refinement and coarsening near interfaces between processors. In Roxane, this constraint is enforced on each refinement level, from the highest to the lowest levels.

For certain problems, Roxane has to get initial variables and material data from an output file of another code project. The variables and material data are defined on a set of triangles. Therefore, this capability is often called trilink. The cells in Roxane are refined based on material interfaces and shock fronts. Simulation boundaries in trilink are always initially refined because of the consideration of possible reflection boundary conditions, and they will be gradually coarsened during the next few time steps if the boundary condition is of reflecting.

The input file in trilink is often large in size. When it is linked to a parallel calculation, Roxane first partitions the simulation domain solely according to area of domains. At that time, Roxane has not built the simulation mesh, and has not read variables and material data yet. On the base of this partition, each processor reads a set of triangles and variables defined on the triangles. The area covered by the set of triangles is slightly larger than the domain of the processor, so that valid interpolation could be made for all the cells within the domain.

Correctness of physics models, numerical methods, and implementation of codes is of paramount importance. After we have the confidence on the models, methods, and implementation of serial calculations, the correctness of parallelization becomes important. The errors in parallelization are often subtle and difficult to identify. For the possible differences in simulation results between serial and parallel calculations, what is the expectation we can have?

Two observations have helped us shape our expectation. First, the difference in results of two serial calculations but on different computers is of the order $O(10^{-12})$. The possible reason for this is the different mechanisms for machine round-off errors. Second, the difference of two serial calculations on a same computer but compiled with any optimization flag, such as $-O$, and with the flag $-g$ is also of the order $O(10^{-12})$. A flag of optimization in compiler would possibly change the order of mathematical operations. Furthermore, any tiny difference in solutions, no matter how small it is, will possibly trigger different mesh refinement and coarsening, thus results in different meshes, and finally gives significantly different results.

Therefore, we have asked for no difference between serial and parallel calculations when the flag $-g$ in compiling is used. Specifically, we have used $10^{-14}$ as the criterion for the difference. The process to guarantee the difference within $10^{-14}$ is the one to find subtle bugs in both parallelization and serial code. Almost every difference has led us to find a new bug.
To compare the result between two calculations, for each cell in the parallel run, we have to first find the corresponding cell in the serial run through space coordinates, which is accomplished through KD-tree, and then compare the variables on the two cells. The tool for this comparison is available to users and developers. We find such a tool is extremely useful to detect any tiny difference.

6 IO

For a long time, Roxane didn’t have parallel IO capabilities, but had the one for data analysis and visualization tool POP, and had limited restart capabilities for certain problems through the IO module pio in rage. During the development of Roxane, particularly its parallel capabilities of various physics packages, parallel IO capabilities for restart and visualization became necessary and urgent. It seems that it will take a much long time to apply the module pio of rage to Roxane.

Current IO capabilities in Roxane are based on the module hio, which consists of two layers, bio and meshio. The layer of bio is a set of functions for writing, querying, and reading, which is based on MPI-IO. On the top of bio, there is another layer called meshio for AMR mesh, unstructured mesh, possible particles, and their associated variables.

Several users have asked for the structure of the header in a file written through bio. Actually, the structure of the header or tail depends on how users use bio, and it is designed to hide from users. Instead, querying functions are supplied to get all the data in file without knowing how the file was generated.

As stated before, the module hio consists of two layers. The first layer includes bio functions for writing, querying, and reading meta data and arrays. The second layer, meshio, is built on the top of bio for writing, querying, reading users’ high level data structures, such as AMR meshes, unstructured meshes, particles, scalar and vector variables defined on these meshes and particles, and the relationship between variables and meshes.

Certain capabilities in bio and meshio are for possible future use. Currently, Roxane uses the mode N-to-I in hio and meshio, in which N processors in a parallel run write to I file at one dump event, such as restart dump and visualization dump. Another mode is N-to-M, through which N processors in a parallel run write to M files.

The module bio also has a buffering mechanism for problem-size arrays. The purpose of this method is to reduce the number of writing into a disk. The buffering IO could improve the IO performance by a factor between 4 and 20, depending on machines and problem sizes. For a typical run on 64 processors, the factor is above 10.

The buffering is the default IO in Roxane. To change the default, developers only have to change the parameter, io_buffering, from true to false. All the tools related to Roxane could read restart files generated through either buffering or non-buffering IO.

High-level data structures in Roxane are directly written into visualization files through the module meshio, which is developed on the top of bio. The high-level data structures include AMR mesh, unstructured mesh in radiation solvers, triangle meshes in trilink, particles, and the variables defined on the meshes. For cell-based AMR meshes, meshio directly write x(nzone, dim) and dx(nzone, dim). The association between variables and meshes is automatically built and stored in files.

The module meshio supports a broad range of unstructured meshes, which include meshes with fixed shapes, arbitrary polygons, and arbitrary polyhedrons. Cells with a fixed shape may be triangles, quadrangles, pentagons, tetrahedrons, pyramids, wedges, and pentagon prisms. A cell may be a zone, or face, or edge, i.e., a mesh may be a zone-mesh, face-mesh, edge-mesh, and points. An edge-mesh may be one-, or two-, or three-dimensional, and a face-mesh may be either two- or three-dimensional. Cells of a zone-mesh may be made directly from nodes, or the cells may be made from edges, or the cells may be made from faces and the faces are then made from either edges or nodes.

The module meshio also supports ghost mesh elements, boundary faces, boundary edges, boundary nodes, slip faces, slip edges, slip nodes, etc. The variables associated with unstructured meshes may be node-variables, or edge-variables, or face-variables, or zone-variables, and variables may be scalars, or vectors, or tensors.

We should point out that the IO capability in Roxane has not gone through any optimization for file systems and hardware. Roxane writes restart files through bio functions listed in the section above. Some data are written the bio_attr_write and others through bio_write. Although bio_write doesn’t have to have valid offset and global size of array as inputs, writing one array through bio_write will result in one communication. To reduce the number of communications, Roxane calculates all the offsets and global sizes through one communication, and then call bio_write to write all the arrays.

Roxane writes only necessary but sufficient data for restart, and writes only data on real cells. To restart with the same number of processors, each processor just reads original data back from a file without reconstructing compressed material data. To restart with different number of processors, each processor has to read a part of original data or more than the original data, and thus has to reconstruct compressed material data. We have not implemented this yet. But, to restart with more processors than the original number, some processors read nothing from the file, and then all processors take advantage of repartition to restart the calculation.
Roxane writes its data in the subroutine Roxane_write, and reads the data in the routine Roxane_read. The restart files of Roxane are in the binary format, and could be read on either big-endian or little-endian computers.

An important part of a visualization file is the simulation mesh and its associated variables. The mesh is described by two-dimensional arrays, \texttt{x(nzone,dim)} and \texttt{dxe(nzone,dim)}. Roxane could directly write \texttt{xe} and \texttt{dxe} into the file for the mesh. Currently our main tool for visualization is EnSight. Visualization tool developed by Computational Engineering International, Inc. Since EnSight doesn’t have any capability to directly accept arrays \texttt{xe} and \texttt{dxe} for visualization, Roxane also could convert \texttt{xe} and \texttt{dxe} to a representation of an unstructured mesh, and then write the unstructured mesh into file.

A reader is developed to visualize Roxane visualization files, no matter whether the mesh in the file is of AMR described by \texttt{xe} and \texttt{dxe} or is unstructured. Currently, the files could be visualized by both EnSight and VisIt, a visualization tool developed in Lawrence Livermore National Laboratory.

Visualization files also could contain some redundant data for convenience in visualization. For example, the working variable, pressure, is never in a restart file, but it may in a visualization file. Also, Roxane writes several other meshes. Each of the meshes is actually a part of the simulation mesh occupied by one material. These meshes for materials overlapped with each other because of mixing. Roxane also writes all the mixing cells as another mesh so that users could easily view the cells that are mixed. On these meshes for materials and mixed cells there are also all the variables.

To reduce the size of visualization files, variables of floating point are written into files as single precision by default. But, considering that some developers may need variables of double precision for debugging, Roxane also could write the variables as double precision. For the size of files, by default, Roxane writes only a basic set of variables into files, but more variables could be added into files through commands in input files.

Roxane could write particles and variables associated with the particles into visualization files, and the reader is able to visualize the particles and their variables.

The reader of Roxane directly connects the data files to visualization tools, such as EnSight and VisIt without explicit transformation of data files. To meet the requirement of some visualization tools, Roxane could also write ghost cells and valid values of variables on ghost cells into visualization files.

7 Regression tests

Roxane goes through its regression tests on main computers daily and weekly. There are three sets of tests. One is for serial calculations to test the correctness of algorithms and modifications of codes, the second set is to test correctness of parallel implementations, and the third is to test those parallel implementations involving third-party software.

For the tests of serial calculations, there is a standard file, called gold file, for each problem. These gold files are actually restart files. We use the same gold file for different computer platforms. Each test will generate a restart file, which will be compared against the gold file of the problem for all the variables. Since there are differences in solutions between different computers, and these differences are typically accumulated with time, the tolerance used in this set of tests may not be as small as it should be for certain test problems. For significant changes in algorithms, we have to re-produce or update those gold files.

For the second set of tests, there are no gold files. Each test on each machine will generate two files, which are restart files too, one from a serial calculation, and the other from parallel calculation. These two files then compare against each other for all the variables. Since numerical results of mathematical operations depend on the order of operations, such as a summation on cells, we are trying to avoid such operations in this set of tests. This is achieved through a command \texttt{“use_global_sum = .false.”} in input files. Since some global sums in Roxane feed back to the dynamics of solutions, such as the refinement/coarsening criteria discussed before, this command actually changes the solutions. But it changes the solutions in both serial and parallel calculations, and in the exactly same way. This command is for parallel regression tests only, and the flag \texttt{use_global_sum} should be left as \texttt{true.} (by default) in users’ calculations.

In principle, the differences in solutions between serial and parallel calculations should be exactly vanishing for the second set of test problems. But, the executable of Roxane in daily regression tests is generated through compiling with optimization flags on. The optimization changes the order of mathematical operation, and changes the order differently on serial and parallel calculations. The optimization results in slightly different results between serial and parallel calculations during each time step. Furthermore, the difference between serial and parallel calculations could be accumulated with time, trigger different refinement and coarsening of meshes, and causes obvious mismatch in final solutions. This is the reason the tolerance is not as small as it should be in some of regression tests.

8 Acknowledgement

Elise Sandford worked on regression tests, Paul Weber worked on the connection between Roxane data files and visualization tools, and Lori Pritchett worked on the integration of the partition tool Zoltan with Roxane and performance of calculations. The authors greatly appreciate their effort. LA-UR-12-23138