Using XML with Large Parallel Datasets: Is There Any Hope?

Renato Elias*, Vanessa Braganholo, Jerry Clarke***, Marta Mattoso****, Alvaro Coutinho***

*High Performance Computer Center, COPPE/UFRJ, Brazil
(e-mail: rnelias@nacad.ufrj.br, alvaro@nacad.ufrj.br)
**Department of Computer Science, IM/UFRJ, Brazil
(e-mail: braganholo@dcc.ufrj.br)
*** US Army Research Laboratory, USA
(e-mail: clarke@arl.army.mil)
**** Computer Systems and Engineering, COPPE/UFRJ, Brazil
(e-mail: marta@cos.ufrj.br)

Abstract: This paper discusses data formats to represent large volumetric datasets. We claim that XML-based formats associated with binary formats are appropriate to this scenario because of the self-descriptive nature of XML. Thus, XML files can describe the structure of the binary files, which improves data handling by applications. This also improves interoperability and allows queries to be posed over data.

Keywords: XDMF, Large Volumetric Data Sets, Data Formats.

1. INTRODUCTION

It is well known that High Performance Computing (HPC) applications deal with large volumes of data, which has increased significantly over time. Such large amounts of data are hard to handle, especially in what concerns moving them from one site to another so they can be used by different applications (flow solvers, visualization, etc.). Time spent in moving data through the network could be prohibitive, and then researchers and application developers are trying their best to avoid this as much as possible [11]. To solve this problem, besides developing new access methods and tools, it is essential that data is represented and stored in a way that contributes to this. It has to be, at the same time, compact and self-descriptive, so that no much effort is put to access it. In this paper, we investigate the issue of representing and storing large parallel datasets.

Our main motivation to investigate this issue is data interoperability among several applications for the simulation of oil and gas problems in the Galileo Network, an alliance of five Universities in Brazil developing innovative parallel applications to face new exploration and production challenges in the recently discovered pre-salt areas in ultra deep waters (2,000s meters) southeast of Brazil. These new parallel applications encompass simulation software for computational solid, fluid and structural mechanics that have to share data. Therefore, besides improving data movement between different applications, data formats should enable data interoperability.

There are several ways to represent and store data. HPC applications usually use binary and compressed data provided by libraries such as netCDF, HDF5, ExodusII. They are efficient, but difficult to interoperate among different programs. One of the limitations is its lack of descriptive information on its binary data. Even after finding the required data, accessing it is not simple. This access might be different for each program that interoperates with the binary data. Ross et al. [11] share our point when they say "High-level libraries such as HDF5 and Parallel netCDF have been developed to provide more natural and efficient interfaces when used properly, but adopting these interfaces is a time consuming task, and even with these libraries significant tuning is often necessary". Additionally, parallel simulation applications that use these formats usually require several input files, one for each solution variable, time step and process. For large scale parallel problems, managing the volume and number of files can be prohibitive: several operating systems have difficulties in dealing with many of files in the same

directory. This is thus one more issue that needs to be addressed when choosing a data format.

Text-based formats like XML are self descriptive and flexible but are often taken aside, since they are too verbose and would largely increase the size of the data set, just making things worse. However, the self-description of XML can add consistency, reliability among other desired features to HPC applications. With this in mind, we investigated existing XML-based data formats for HPC applications, and found three main ones: VTK/XML [14], XDMF [1] and XDTM [10].

VTK/XML files can be of two types: structured (for topologically regular datasets such as arrays of pixels or voxels) and unstructured (for topologically irregular sets of points and cells). However, this format does not allow storing heavy data in separate binary files. It encodes all binary data using a Base64 scheme in order to not violate the XML standard. XDTM, on the other hand, has no predefined schema, and thus each application can define their own. The idea is to map XML types into a physical representation. Applications would deal with XML types directly, thus avoiding the difficulties of accessing binary data. The problem of this approach is the need to re-implement applications and no uniformity between schemas. XDMF, on the other hand, classifies data as light and heavy according to the amount of information it represents. Light data is allowed to be stored in the XDMF file's body following the XDMF DTD rules while heavy data is stored in HDF5 format and described in the XDMF file. This approach takes advantage of the XML proposal for a self described data without producing large XML files. XDMF is currently supported by VTK and consequently Paraview and Vislt.

In this work, we present a case study for the XDMF file format in a parallel finite element simulation for a natural convection problem. This problem is a simplified model problem for complex viscous flow and temperature phenomena occurring in salt tectonics [7]. Velocity, pressure and temperature data for several time steps were represented in XDMF and HDF5 for visualization with ParaView. The remainder of this paper is structured as follows. Section 2 presents XDMF in more details. Section 3 presents a case study with XDMF. Finally, Section 4 concludes and discusses future work.

2. XML AND LARGE VOLUMETRIC DATASETS

XDMF stores light data directly in XML format, while heavy data is described in XML but actually stored in HDF5 files. HDF5 alone is not enough, since it is not self-descriptive. The advantage of using XDMF together with HDF5 is that the structure is explicitly represented in the XDMF file, thus making access much easier. In summary, an XDMF file is composed of one or more *Domain*. Domains are composed of Grids, each of which represents a collection of homogeneous elements. Each *Grid* can have temporal data (*Time*), other Grids, descriptions of the data organization (*Topology*), XYZ mesh values (*Geometry*) and mesh values (*Attribute*). Data itself is stored in lower level elements called *DataItem*. There is also a special element called *Information* that can store application specific data.

When HDF5 files are used, DataItem points to positions inside the HDF5 files where data is actually stored. Figure 1 shows an example of XDMF file that points to different portions of an HDF5 file. The example is truncated due to space restrictions, but the following lines repeat the same block pointing to other HDF5 files. Notice that, despite the use of HDF5 the metadata information is explicit in the XDMF file. This makes data access much easier. Moreover, this example demonstrates the concept of spatial collections. Spatial collections can be employed to assemble different geometrical parts of the computational grid. Note that, when working in a domain decomposition context, the domain is spatially partitioned in smaller pieces and individually assigned to MPI processes. In this sense, the XDMF spatial collection is employed to recover the non-partitioned geometrical model. Alternatively, spatial collections could be used to distinguish different geometrical parts of the same model in a serial or parallel run. In this way, a car model could be understood as a collection of spatial parts representing the car doors, tires, wind shield and so on.

Another desirable concept in scientific file formats is the ability to represent transient data. In XDMF this is accomplished by the use of “temporal collections”. In Figure 2 we may observe that a “temporal

collection" is used to list which XDMF file corresponds to each time value. Moreover, in this example, each time step is stored in a different XDMF "spatial collection" file, as previously described and shown in Figure 1, and referenced by the use of *<xi:include href="..."/>* clauses. It gives rise to a file arrangement corresponding to a "temporal collection of spatial collections". Note that, in practice, several possibilities in terms of data storage and arrangements could be made without losing portability and readability across different software and systems. Actually, this is one of the main XML goals and naturally present in XDMF format.

```
<XDMF xmlns:xi="http://www.w3.org/2001/XInclude" Version="2.0">
  <Domain Name="EdgeCFD">
    <Grid Name="foo_00000" GridType="Collection" CollectionType="Spatial">
      <Time TimeType="Single" Value="0.00"/>
      <!-- GRID FOR THE FIRST PARALLEL (SUB)DOMAIN -->
      <Grid Name="foo_000" GridType="Uniform">
        <!-- MESH INCIDENCE -->
        <Topology Type="Tetrahedron" NumberOfElements="   296882 " BaseOffset="1">
          <DataItem Dimensions="1187528 " NumberType="Int" Format="HDF">foo_000_00000.h5:/incid</DataItem>
        </Topology>
        <!-- NODAL COORDINATES -->
        <Geometry Type="XYZ">
          <DataItem Dimensions="1472283" NumberType="Float" Precision="8" Format="HDF">
                    foo_000_00000.h5:/coords</DataItem>
        </Geometry>
        <!-- TRUNCATED LINES -->
      </Grid>
      <!-- GRID FOR THE SECOND PARALLEL SUBDOMAIN -->
      ...
   <!-- TRUNCATED LINES -->
   </Grid>
  </Domain>
</XDMF>
```

Fig. 1: *XDMF for a spatial collection (file foo_00000.xmf).*

```
<XDMF xmlns:xi="http://www.w3.org/2001/XInclude" Version="2.0">
   <Domain Name="EdgeCFD">
      <Grid GridType="Collection" CollectionType="Temporal">
         <Time Type="Single" Value="0.00"/>
         <xi:include href="foo_00000.xmf" xpointer="xpointer(//XDMF/Domain/Grid)" />
         <Time Type="Single" Value="0.01"/>
         <xi:include href="foo_00001.xmf" xpointer="xpointer(//XDMF/Domain/Grid)" />
      <!-- TRUNCATED LINES -->
      </Grid>
   </Domain>
</XDMF>
```

Fig. 2: *XDMF example for a temporal collection (file foo.xmf).*

3. CASE STUDY

The case study presented in this work employs the EdgeCFD software. It is a parallel Fortran90 finite element code for coupled Navier-Stokes and transport problems. EdgeCFD features stabilized and variational multiscale finite element formulations. EdgeCFD is being used for Newtonian and non-Newtonian fluid flows, free-surface flow simulations with interface tracking approaches (volume-of-fluid/level sets), gravity currents and turbulence [see 7 and references therein]. The two equations are solved by a staggered approach. In this software, most of the computational cost comes from the **u**-p coupled solution of the incompressible flow equations while the cheaper part is due to the transport equation. Time integration is a predictor-multicorrector algorithm with adaptive timestepping [11]. Within the flow solution loop, the multi-correction steps correspond to the Inexact-Newton method as described in [3]. As a linear solver, EdgeCFD employs the Generalized Minimal Residual Method (GMRES) since both equation systems, stemming from the incompressible flow and transport, are non-symmetric. Furthermore, a nodal block-diagonal and diagonal preconditioner are used respectively for flow and transport. Most of the computational effort spent in the solution phase is devoted to matrix-vector products. In order to compute such operations more efficiently, we have used an edge-based data structure as detailed by Elias et al [3]. This data structure reduces indirect memory access, memory requirements to hold the coefficients of the stiffness matrices and the number of floating point operations

when compared to other data structures. The computations are performed using the message passing interface library (MPI). The parallel partitions are generated by Metis/ParMetis library, while the information regarding the edges of the computational grid is obtained from the EdgePack library [2]. EdgePack also reorders nodes, edges and elements to improve data locality, exploiting efficiently the memory hierarchy of current processors.

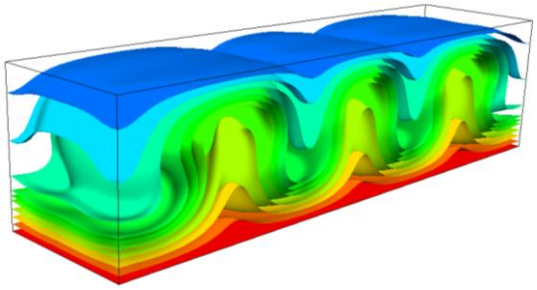

Fig. 3. Iso-temperature contour plots for the three-dimensional Rayleigh-Benard problem.

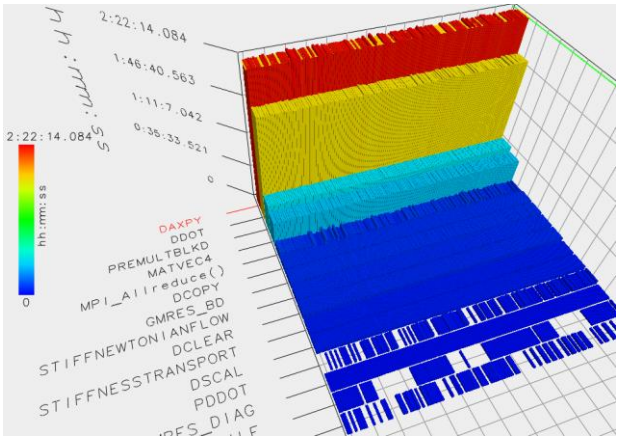

Fig.4. EdgeCFD parallel performance on 128 cores of SGI Altix ICE cluster.

A case study of 3D Rayleigh–Benard convection has been carried out to investigate the XDMF use in a large scale simulation. The benchmark corresponds to a rectangular 3D domain of aspect ratio 4:1:1 aligned with the Cartesian axes and subjected to a temperature gradient [4]. Simulations are made on a 501×125×125 mesh, resulting in 39,140,625 tetrahedral elements using an edge-based SUPG scheme with pressure stabilization. A converged stationary solution, shown in Figure 3, shows the convective rolls obtained at Rayleigh number Ra=30,000 and Prandt number Pr=0.71.This solution was obtained on 128 cores of a SGI Altix ICE 8200 cluster with a fixed time step. Every time step we solve, by the Inexact Newton method, two nonlinear systems of equations, for flow and temperature, respectively with 31M and 7.8M equations.

Code performance has been profiled using TAU on 50 time steps. The three-dimensional execution profile is shown in Figure 4. We may observe the excellent load distribution by the uniform size of the CPU time bars. In this case we used non-blocking point-to-point communication between subdomains. It is worth to emphasize that most of the time is spent in parallel DAXPY BLAS primitives (red bar), which does not involve communication. It indicates that the same problem could be run in much more CPUs without losing scalability.

Following the XDMF concepts, in this case study the heavy information (nodal coordinates, element incidence, pressure, temperature and velocity nodal solution) is hierarchically stored in HDF5 files while all information required to access the HDF5 files (number of elements and nodes) are stored in XDMF files. Moreover, each process stores its own solution temporal collection as described in Section 2, while rank 0 creates the spatial collection to collect all subdomain data. With this arrangement, the solution is rendered and visualized using ParaView in a client-server scheme. The system used for the solution process and rendering is the SGI Altix ICE cluster, with no special graphics hardware. In other words, the visualization is off-screen rendered by the SGI Altix ICE numerical processors using the Mesa library, connected to a ParaView client session. The number of files generated in this XDMF scheme (spatial collection of temporal collections) is proportional to the number of processors and transient output frequency as well and can be computed by the formula $n_f = n_s(1 + n_p) + 1$, where n_f, n_s and n_p are the number of files, time steps saved (output frequency) and processors respectively. Consequently, for a run

using 128 processors storing 15 time steps are created a total of 1,936 files. Clearly, for a large number of saved time steps and processor counts, the number of files can be very large. With our solution however, handling large number of files poses no difficulty at all.

4. CONCLUDING REMARKS

In this paper, we discussed the advantages of using an XML-based format to represent data in HPC applications. In this case study heavy information is hierarchically stored in HDF5 files while all information required to access the HDF5 files are stored in XDMF files following the XML standard, thus allowing flexibility, interoperability between different applications, self-description and easy-of-access. However, several other advantages can be foreseen when data is represented in XML. In our case study, XDMF allowed a significant reduction on the number of generated files. Additionally, generating XDMF files from our simulator was trivial. We just added some code, but no real change was needed. This code extension has also been included in other simulators we have. XDMF has all data types to support CFD applications and all our solvers now share the same valid schema. Note though that any other application using VTK can benefit from our solution.

On the other hand, as we generate more XML documents sharing the same schema we can proceed to submit XML queries that would return "all files that have *n* NodesPerElement" or "files use Tetrahedron as topology type". The discussion of proper formats for visualization of volumetric datasets is far from reaching an end. There are several other formats being proposed [6], and no standard yet. Furthermore, depending on the available resources and/or problem size, the end-to-end approach of Tu et al [10], where no intermediate results from simulations are saved, may be preferable.

Acknowledgements. Authors would like to thank CNPq and Petrobras for partially supporting this work.

REFERENCES

1. Clarke, J., Mark, E., Enhancements to the eXtensible Data Model and Format (XDMF). In: DoD High Performance Computing Modernization Program Users Group Conference (HPCMP), pp. 322-327. 2007.
2. Coutinho, A.L.G.A., Martins, M.A.D., Sydenstricker, R.M., Elias, R.N., Performance comparison of data-reordering algorithms for sparse matrix–vector multiplication in edge-based unstructured grid computations. International Journal for Numerical Methods in Engineering, 66:431–460, 2006.
3. Elias, R.N., Martins, M.A.D., Coutinho, A.L.G.A., Parallel edge-based solution of viscoplastic flows with the SUPG/PSPG formulation. Comput. Mech., 38:365-381, 2006.
4. Griebel, M., Dornseifer, T., Neunhoeffer, T., Numerical Simulation in Fluid Dynamics—A Practical Introduction. SIAM: Philadelphia, PA, 1998.
5. Hudec, M. R., Jackson, M. P.A., Terra infirma: Understanding salt tectonics, Earth-Science Reviews 82 (2007) 1–28
6. Kruger, J., Potter, K., MacLeod, R., Johnson, C., UFV – Unified Volume Format: A General System for Efficient handling of Large Volumetric Datasets. In: IADIS Computer Graphics and Visualization 2008 (CGV), 2008.
7. Lins, E.F., Elias, R.N., Guerra, G., Rochinha, F.A., Coutinho, A.L.G.A., Edge-based finite element implementation of the residual-based variational multiscale method. International Journal for Numerical Methods in Fluids, 2009.
8. Moreau, L., Zhao, Y., Foster, I., Voeckler, J., Wilde, M., XDTM: The XML Data Type and Mapping for Specifying Datasets. In: European Grid Conference (EGC), LNCS 3470, 495-505, 2005.
9. Ross, R., Peterka, T., Shen, H.-W., Hong, Y., Ma, K.-L., Yu, H., Moreland, K., Visualization and Parallel I/O at Extreme Scale, Preprint ANL/MCS-P1520-0708, July 2008.
10. Tu, T., Yu, H., Ramirez-Guzman, L., Bielak, J., Ghattas, O., Ma, K-L., O'Hallaron, D., From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing. In: SC'06, Tampa, FL, 2006.
11. Valli, A.M.P., Elias, R.N., Carey, G.F., Coutinho, A.L.G.A., PID adaptive control of incremental and arclength continuation in nonlinear applications. International Journal for Numerical Methods in Fluids, 2009.
14. VTK/XML. Available at http://www.vtk.org/VTK/img/file-formats.pdf.