ARMY RESEARCH LABORATORY

Simplifying Partitioning Complexities by Using a Common Data Hub

by Jerry A. Clarke and Raju R. Namburu

ARL-TR-2799 August 2002

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5067

ARL-TR-2799 **August 2002**

Simplifying Partitioning Complexities by Using a Common Data Hub

Jerry A. Clarke and Raju R. Namburu

Computational and Information Sciences Directorate, ARL

Abstract

Most scalable, high-performance simulations of interest to the Department of Defense involve calculations on a geometric structure. These topologies (unconnected points, unstructured meshes, rectilinear grids, etc.) are then partitioned across many processors. The simulation proceeds in parallel with information being communicated between processors as necessary. If the topology of the mesh changes (crack propagation, adaptive mesh refinement, etc.), the mesh must be repartitioned before efficient computation may continue. In addition, if data must be shared between various topologies (Coupled Eulerian–Lagrangian simulation), the complexities of transferring data between meshes, in parallel, can be significant.

The Interdisciplinary Computing Environment has defined a common data model and format that can efficiently consolidate large quantities of computed data during runtime. Initially used for runtime visualization of parallel simulations, this data hub is now being applied to the computational simulations themselves in order to alleviate the complexities of communicating data between different topologies and the repartitioning of evolving topologies. Data can be written to the data hub using one partitioning scheme and read back with a totally different scheme. In the situation of different topologies, programs need only be cognizant of their partitioning and the "overall" topology of the opposing mesh.

Contents

INTENTIONALLY LEFT BLANK.

List of Figures

INTENTIONALLY LEFT BLANK.

1. Introduction

Most scalable, high-performance simulations of interest to the Department of Defense (DOD) involve calculating values on some type of two (2-D) or three-dimensional (3-D) geometric structure. For example, classical molecular dynamics applications regularly calculate values on points in 3-D space (atoms). Finite-element structural dynamics simulations use beams (lines), tetrahedral, hexahedra, etc., while finite difference computational fluid dynamics codes use curvilinear grids. Values for these various classes of codes are calculated on the grid nodes or for some interior position in each cell (center, integration point, etc.).

Three examples of codes that use this traditional computational approach are shown in Figure 1. The first is a section of a Bacteriorhodopsin molecule simulated using fast molecular dynamics (FMD). The second is a simulation of an earth-penetrating projectile using Pronto3D, a finite element structural dynamics code. The third is a lateral jet thrust projectile using ZnsFlow, a finite difference fluid dynamics code. While these are only three examples, most High-Performance Computing (HPC) computational technology area (CTA) codes of interest to the DOD use similar computational techniques.

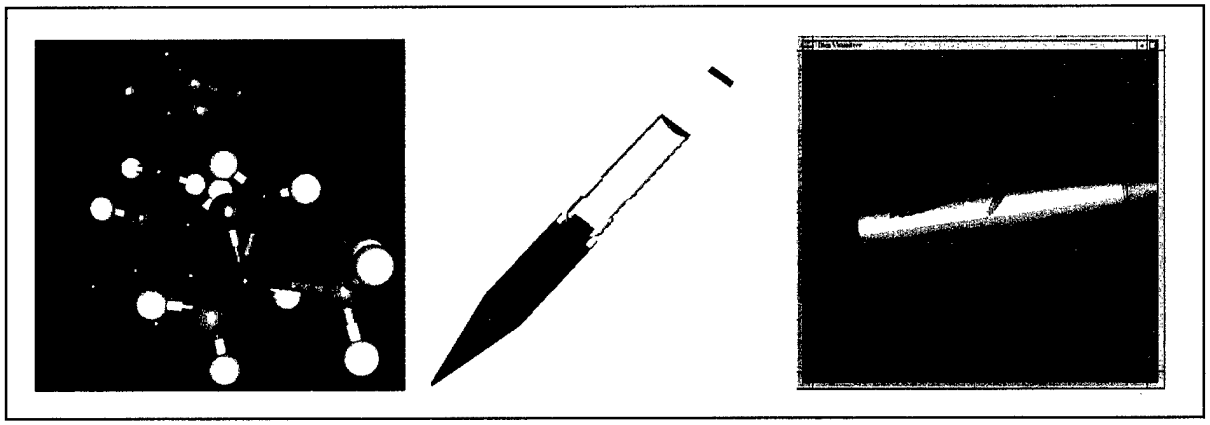

Figure 1. Computing values on meshes for computational chemistry and materials science (CCM), computational structural mechanics (CSM), and computational fluid dynamics (CFD).

A major concern for any of these computational techniques is how to efficiently compute the necessary values in parallel. These codes endeavor to optimize the calculation by evenly distributing work among processors and minimizing communication. Because various sections of the computational domain may require vastly different amounts of computational effort, simply decomposing the domain into equal partitions may not always be the optimal solution.

2. Efficient Mesh Partitioning

For this reason, much research and effort has been expended on developing efficient portioning utilities. ParaMeTIS from the University of Minnesota and CHACO from Sandia National Laboratories (SNL) are two examples of widely used mesh portioning software. Both of these tools are used in HPC codes of importance to the DOD. ParaMeTIS is used in ParaDyn from Lawrence Livermore National Laboratory and CHACO is used in Pronto3D from SNL. Other HPC codes use a variety of mesh portioning tools as either a preprocessing step or during runtime to dynamically distribute the workload after adaptive refinement.

Based on one or more criteria (computational load, communication, etc.), most of these partitioning software tools construct a weighted graph structure and then partition that structure using a variety of graph partitioning algorithms. Other tools like Zoltan (from SNL) include migration tools that perform the communication needed to move data and establish a new decomposition via message passing interface (MPI).

3. Complexities of Coupled Codes

All is well until an attempt is made to couple two or more HPC codes or tools. In addition to having vastly different mesh topologies, these codes or tools may be computing vastly different quantities that effect the computational weight assigned to each domain.

A good example of this problem is exhibited by the coupled Lagrangian–Eulerian code Zapotec as shown in Figure 2. This HPC code, which is part of the common high-performance computing software support initiative (CHSSI) portfolio project "Interdisciplinary Computing Environment for Weapon-Target Interaction" couples CTH (a finite volume Eulerian shock physics code) with Pronto3D (a finite element structural mechanics code). CTH uses a rectilinear mesh evenly decomposed among processors while Pronto3D uses an unstructured mesh partitioned using CHACO. To exchange information directly between processors, both the Lagrangian and Eulerian sections of the code must be cognizant of the other's partitioning scheme. If one wanted to use Zapotec with another finite element code, like ParaDyn (which uses MeTIS), a large potion of the code would need to be rewritten to accommodate the new partitioning scheme. So not only does the coupled code need to know about a foreign topology, it must also know the nuisances of the partitioning.

To simplify the exchange of information between codes (and tools) we have implemented an abstracted active data hub. The eXtensible Data model and Format (XDMF) provides efficient data transport and enough data format and metadata to describe a wide variety of information

Figure 2. Buried structure using the Zapotec couples Eulerian–Lagrangian code.

found in heavily used DOD HPC programs. This data hub has the advantage of freeing coupled codes and tools from the details of how the data are decomposed in order to perform parallel computation.

4. eXtensible Data Model and Format

At the heart of the system, is a unique heterogeneous shared memory system. Network Distributed Global Memory (NDGM) provides XDMF with a physically distributed, logically shared, unstructured memory buffer. But instead of handling the mapping and unmapping of memory pages automatically, NDGM is accessed through a subroutine interface. Although less automatic, this allows applications to form a "cooperative shared memory" that is simple yet efficient.

As shown in Figure 3, NDGM is a client-server layer that consists of multiple server processes and an Application Programmers Interface (API) for clients. Each server maintains a section of a virtual contiguous buffer and fields requests for data transfer and program synchronization. Clients use the API to transfer data in and out of the virtual buffer and to coordinate their activity.

NDGM has been used to develop parallel HPC applications, but it is particularly useful as a "data rendezvous" for a collection of applications. A parallel, computationally intensive code can write a snapshot of data to NDGM then continue its processing. The data can then be visually inspected, through 2-D plots and 3-D surfaces, while not inhibiting the progress of the code.

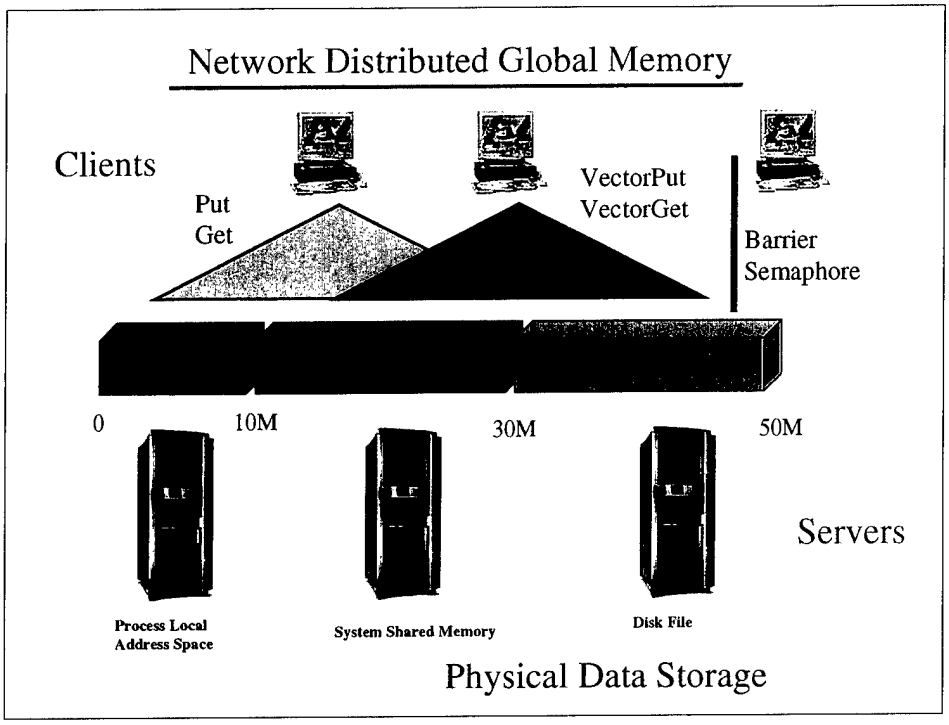

Figure 3. Emulating shared memory on distributed systems.

NDGM provides a distributed, heterogeneous unstructured buffer. To provide some structure to this buffer, XDMF uses the "Hierarchical Data Format Version 5" (HDF5) from the National Center for Supercomputing Applications (NCSA). HDF5, a well-known and widely used format, is designed to allow an orderly access to structured and unstructured datasets. All access is accomplished through a well-defined API. In addition to allowing access to disk files, HDF5 provides a "Virtual File Layer." This allows the addition of drivers for data access. XDMF includes an HDF5 *driver* for NDGM that allows NDGM access via the standard HDF5 API.

HDF5 defines a feature rich data format for structured and unstructured datasets as well as groups of data. XDMF primarily uses HDF5 for storage of enormous datasets or "Heavy" data. For description of the meaning of data or small amounts of data, XDMF uses the eXtensible Markup Language (XML). While HDF5 has an "attribute" facility for storing name=value pairs, the use of XML allows support of other heavy data formats as necessary. Additionally, the enormous amount of software available to process XML makes it a natural candidate for storage of this "Light" data.

The concept of separating the light data from the heavy data, as shown in Figure 4, is critical to the performance of this data model and format. HPC codes can read and write data in large, contiguous chunks that are natural to their internal data storage, to achieve optimal I/O performance. If codes were required to significantly rearrange data prior to I/O operations, data locality, and thus performance, could be adversely affected, particularly on codes that attempt to

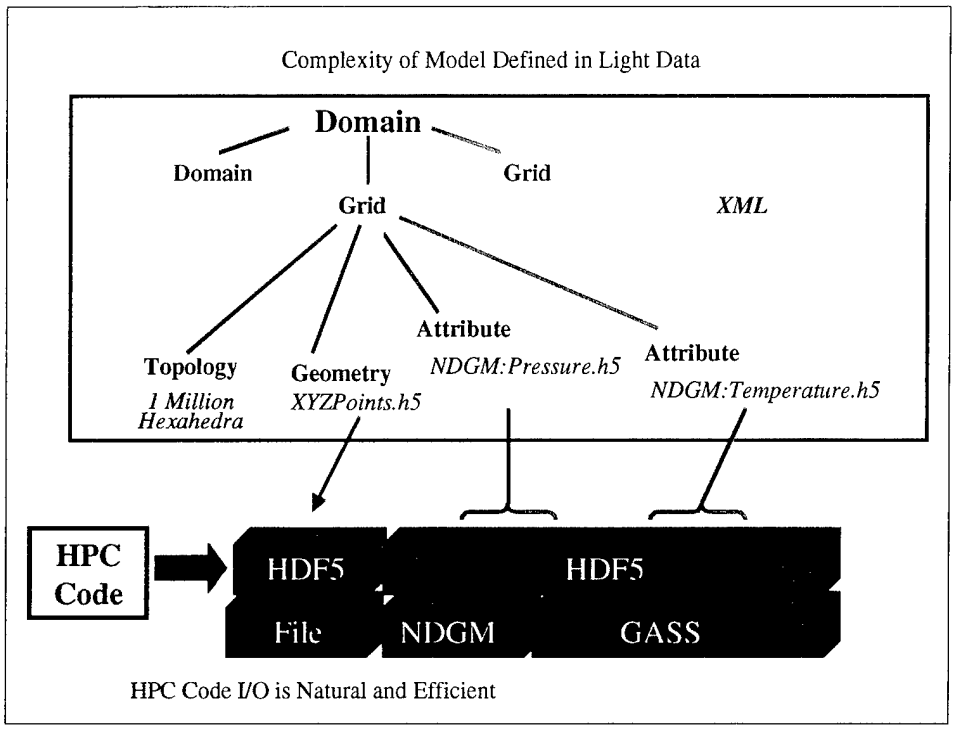

Figure 4. Separating light and heavy data.

make maximum use of memory cache. The complexity of the dataset is described in the light data portion, which is small and transportable. For example, the light data might specify a topology of one million hexahedra while the heavy data would contain the geometric XYZ values of the mesh and pressure values at the cell centers stored in large, contiguous arrays. This key feature allows reusable tools to be built that do not put onerous requirements on HPC codes. Despite the complexity of the organization described in the XML below, the HPC code only needs to produce the three HDF5 datasets for geometry, connectivity, and pressure values.

Through the use of NDGM, HDF, and XML, an abstracted active data hub is made possible for use in U.S. Army applications. XDMF provides a level of abstraction for enormous distributed datasets. Computational codes, visualization, and user interface can all interface with the data in a well-defined method without severely limiting performance. As mentioned previously, XDMF incorporates both a data model and format. It allows for a self-describing method of storing large data structures and the information necessary to tell how the data are to be used. XDMF provides a C++ class library mainly as a convenience layer. Codes and tools use this layer from system programming languages like C++, C, or FORTRAN or from scripting languages like Tcl, Python, and Java to easily access any XDMF functionality.

5. Benefits

The benefits of using XDMF for codes like Zapotec are twofold. First, a new exchange mechanism does not need to be developed for every code that is added to the system. In addition, other analysis tools, like runtime visualization, can be easily added because the data are available in a common data model and format. Access to the data hub by external analysis does not encumber the computational efficiency of the simulation.

With help from SNL, we have modified Zapotec to exchange node-centered information via XDMF instead of through its internal hard-coded scheme. This is in preparation to use other finite element Lagrangian codes like ParaDyn that have different capabilities. With help from CalTech, we have added the same capability to VTF (also part of the Interdisciplinary Computing Environment [ICE] for Weapon-Target Interaction CHSSI Portfolio project). In this situation, all data are transferred via XDMF. This allows the Eulerian fluid solver and Lagrangian solid solver to execute on the same machine or on different platforms seamlessly. It also gives VTF access to all of the ICE runtime visualization tools.

6. Conclusion

By freeing codes and tools from the complexities of understanding a variety of mesh partitioning schemes, we have simplified the access to runtime data for coupled codes and analysis tools. XDMF provides an efficient data exchange mechanism, along with a common, extensible data model and format to allow codes and tools to easily exchange computed values in a meaningful way.

Bibliography

Clarke, J. "Emulating Shared Memory to Simplify Distributed-Memory Programming." *IEEE Computational Science & Engineering*, vol. 4, no. 1, pp. 55–62, January-March 1997.

Clarke, J., and R. R. Namburu. "The eXtensible Data Model and Format: A High Performance Data Hub for Connecting Parallel Codes and Tools." ARL-TR-2552, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, July 2001.

Hoover, C. G., A. J. DeGroot, J. D. Maltby, and R. J. Procassini. "ParaDyn: DYNA3D for Massively Parallel Computers." UCRL 53868-94, Lawrence Livermore National Laboratory, Livermore, CA, 1995.

McGlaun, J. M., S. L. Thompson, and M. G. Elrick. "CTH: A Three-Dimensional Shock Wave Physics Code." *International Journal of Impact Engineering*, vol. 10, pp. 351–360, 1990.

Namburu, R. R., J. O. Balsara, T. L. Bevins, and P. P. Papados. "Large-Scale Explicit Simulations on Scalable Computers." *Advances in Engineering Software*, vol. 29, pp. 187–196, 1998.

Schraml, S., K. Kimsey, and J. Clarke. "High-Performance Computing Applications for Survivability-Lethality Technologies." *IEEE Computing in Science & Engineering*, vol. 4, no. 2, pp. 16–22, March-April 2002.

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	August 2002	Final, July 2001–July 2002

4. TITLE AND SUBTITLE
Simplifying Partitioning Complexities by Using a Common Data Hub

5. FUNDING NUMBERS
2U06CC

6. AUTHOR(S)
Jerry A. Clarke and Raju R. Namburu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-2799

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT*(Maximum 200 words)*

Most scalable, high-performance simulations of interest to the Department of Defense involve calculations on a geometric structure. These topologies (unconnected points, unstructured meshes, rectilinear grids, etc.) are then partitioned across many processors. The simulation proceeds in parallel with information being communicated between processors as necessary. If the topology of the mesh changes (crack propagation, adaptive mesh refinement, etc.), the mesh must be repartitioned before efficient computation may continue. In addition, if data must be shared between various topologies (Coupled Eulerian–Lagrangian simulation), the complexities of transferring data between meshes, in parallel, can be significant.

The Interdisciplinary Computing Environment has defined a common data model and format that can efficiently consolidate large quantities of computed data during runtime. Initially used for runtime visualization of parallel simulations, this data hub is now being applied to the computational simulations themselves in order to alleviate the complexities of communicating data between different topologies and the repartitioning of evolving topologies. Data can be written to the data hub using one partitioning scheme and read back with a totally different scheme. In the situation of different topologies, programs need only be cognizant of their partitioning and the "overall" topology of the opposing mesh.

14. SUBJECT TERMS	15. NUMBER OF PAGES
HPC, data model and format, distributed computing	12
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.