SAND2012-7304C

In situ Fragment Detection at Scale

Nathan Fabian*
Sandia National Laboratories

ABSTRACT

We explore the problem of characterizing fragments using ParaView *in situ* with an explosion simulation. By running *in situ* we can see a much higher temporal view of the data as well as potentially compress the output to only those statistics about fragments we care about. However, the fragment finding must be able to scale as well as the simulation. In order to achieve the necessary scales, we borrow operations the simulation is already doing and take advantage of them within ParaView, demonstrating the resulting improvement in scaling performance.

Keywords: coprocessing, in situ, simulation, scaling

Index Terms: H.5.2 [User Interfaces (D.2.2, H.1.2, I.3.6)]: Prototyping—User interface management systems (UIMS); I.3.6 [Methodology and Techniques]: Interaction techniques

1 INTRODUCTION

In this paper, we explore the problem of characterizing fragments in an explosion simulation. A few examples of possible fragments are shown in Figure 1. We are interested in determining properties such as their size and shape as well as direction of travel and momentum as they come away from the main mass during the explosion. This is important, for instance, in survivability studies. In this paper, the particular problem we are working with is a simulation of an exploding pipe, Figure 2.

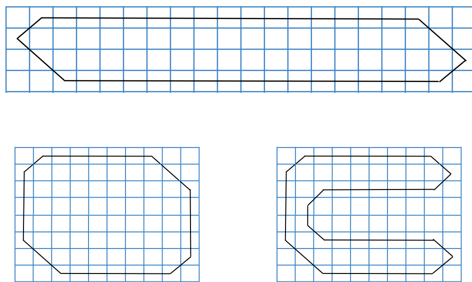

Figure 1: Examples of potential fragments that we'd like to characterize with this algorithm. The top fragment is long and sharp. The bottom left fragment is rounded and potentially safe. The u-shaped fragment may be harmful depending on the scenario.

The simulation code we use is CTH [3]. It is an Eulerian shock physics code that uses an adaptive mesh refinement (AMR) data model. These adaptive finite volumes can take up different amounts of space depending on where they are in the model and how closely the simulation is refining the space.

In order to correctly find fragments, we must first determine what is and is not a fragment. The simulation operates on a finite volume

*e-mail: ndfabia@sandia.gov

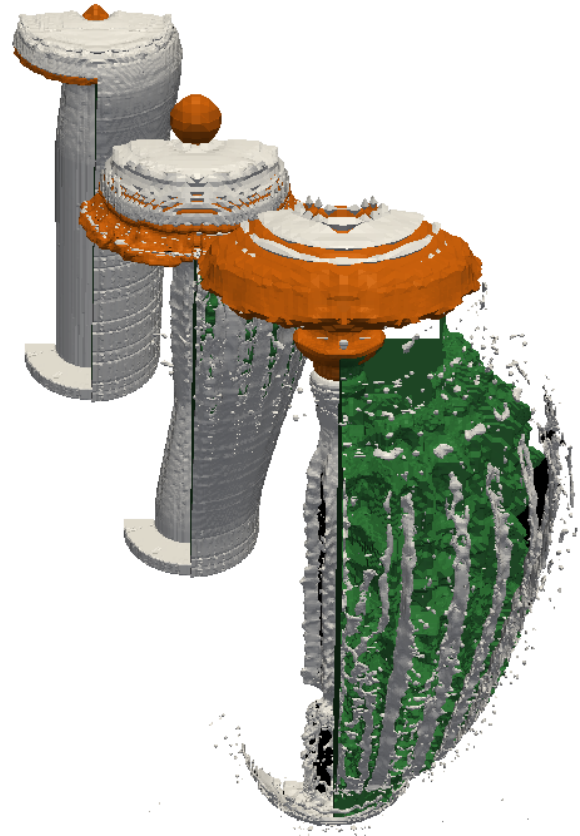

Figure 2: We are interested in finding and characterizing the fragments coming away from this exploding pipe.

and consists of a set of simulated materials which each take up a certain fraction of finite cells within that volume. We treat any connected region of cells with material volume fraction above a given threshold as a fragment of that material. Generally speaking, when the simulation begins, each material consists of one connected region, which we refer to as the main mass. As the simulation progresses, this region breaks apart and gaps occur between pieces of material, filled either by another material or by the surrounding air. Once there is a gap at least as large as one cell, i.e., material fraction below the threshold, we determine that a fragment has formed. The challenge when finding these fragments on a large scale parallel system is that regions that make up the fragments straddle process boundaries, requiring communication between the processes to determine the full shape of a fragment.

We find these fragments using algorithms developed in ParaView [5] linked *in situ* [2] with and running in the same memory space as CTH. Because we are running *in situ* we have closer access to the data without having to wait for it to be written to disk, but there is a necessary cost since the simulation must wait for the analysis algorithms to finish before it can proceed. Thus it is critical for the analysis to scale as efficiently as the simulation and run well on

the tens and hundreds of thousands of cores the simulation will be running on, and further as we move from petascale into exascale. While the algorithm for finding fragments already existed in ParaView, this work takes advantage of having closer access to CTH by reusing neighborhood information from within the simulation to avoid neighborhood discovery communications and to make the fragment finding algorithm work effectively on high performance machines.

The result of the algorithm is a collection of fragments represented as water-tight polyhedral mesh surfaces containing no gaps. Alone these can be much smaller than the original AMR mesh, especially after a visually preserving decimation of the triangle surface. In some cases, where an analyst is only concerned with a histogram of fragment quantities, such as number of fragments within a certain size range, the data written out can be many orders of magnitude smaller than the original mesh.

In the following sections, we will discuss the implementation of fragment detection algorithm currently implemented in ParaView and how it was modified to take advantage of operations already performed by the simulation. Also we'll discuss different representations and the reasoning behind the water-tightness. Finally, we will show the performance improvement as a result of the optimizations done in this work and discuss comparisons in terms of running the analysis *in situ* versus writing the data to disk.

2 Related Work

A number of algorithms for material interface reconstruction exist in the literature. One of the closest to the method use in here is described by Bonnell et al [1], which uses a material dependent coordinate to find surfaces around multiple materials easily. By contrast, the algorithm we use here finds fragment surfaces for each material independently, treating all other material fractions as air or void.

In work similar to this, Masters et al lciteMasters2008, are reconstructing interfaces to characterize fragments and debris in an adaptive mesh refinement simulation. While the goals are similar, the works by Masters feeds these interfaces back into an arbitrary Lagrangian-Eulerian simulation in order to more correctly model the fragments. In this work, the simulation proceeds independently of the analysis, though it may be possible to use the interfaces determined by this algorithm for a coupling back into a Lagrangian simulation.

3 Implementation

The method we use reconstructs water-tight fragment surfaces from the volume fractions on the cell. While it is possible some of the information we are concerned with is available considering only bounding boxes on the fragments, there may be some useful information lost, as can be seen in the similarity of the bounding box in Figure 1 between the rounded fragment and the u-shaped one. Additionally, more of the shape information would be available, if we simply thresholded the cells by volume and stored the cells as-is that make up the fragment. The general shape of the fragment would be available without needing to reconstruct the interfaces. However, this representation is much less compact than a water-tight surface. Because the water-tight surface is fully connected we can decimate the surface very effectively while preserving the topological structure, which results in a data reduction, roughly 9.5 time smaller in number of elements than thresholding cell regions of fragments. This large difference can be demonstrated in Figure **??**, where for instance the long pointed fragment would require saving 38 thresholded cells, but its surface representation would only require four polygons, when decimated.

The challenge in finding a water-tight isosurface values in an AMR mesh is in stitching the surfaces between different resolutions between blocks. When there is this difference in resolution, as shown in Figure 3, the isosurfaces generated for the two different resolutions will not meet up, so it is necessary to create degenerate shapes at the seams in order to correctly close the surface. This problem is exacerbated in distributed processing as these seams can bridge processor bounds and requires ghost cell information which is at a different resolution than the resident data. Unfortunately, the ghost cells contained in the CTH data set, are not sufficient for our needs here. Thus, it becomes necessary to do an all-to-all broadcast to determine where neighboring blocks are located in order to pass the ghost data.

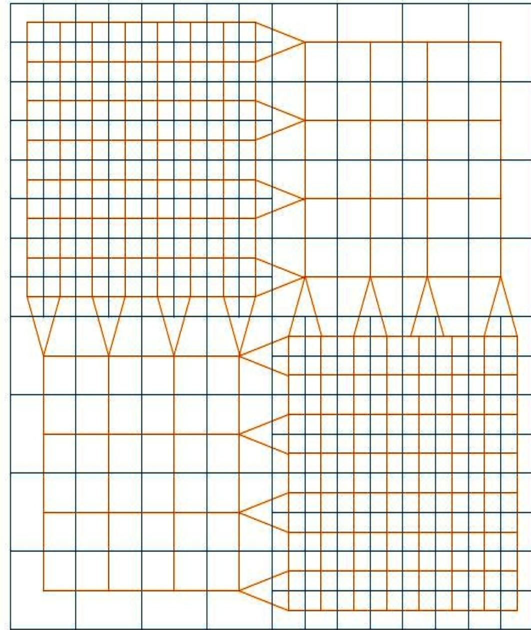

Figure 3: Image representing the isosurface at two different resolutions. Because the isosurfaces do not align, degenerate shapes must be stitched in to close the seams and make the surface water tight.

While this algorithm runs effectively when post-processing data on disk using ParaView, unfortunately the all-to-all broadcast makes the algorithm scale poorly, as shown in Figure 4. While it may be possible to amortize the cost by performing the all-to-all only when the AMR surface changes, this change happens often enough as to make it not worth doing. Therefore, we implement a mechanism to pull the block-neighbor information directly from CTH and take advantage of it within the algorithm.

Normally when linking *in situ* we can minimize the impact of changes to the original simulation code by linking into the IO layer. This is a point in the code where the simulation already expects to hand over a copy of the data to a long running part of the process while it waits for that data to be written to disk. So it is here that we can insert the analysis calls before the data gets written out to disk, without heavily modifying the simulation code. However, in order to pass the block-neighbor information, we must access a set of data that is not passed through the IO layer by CTH. In order to minimize the impact to the CTH API, we pass this new neighbor information at the end of the call to IO. The original code in CTH that writes information out to disk will ignore this extra information, but our code that splices in between these two layers, can access that information and pass it over to ParaView. Although we could allow that information to be written to disk along with everything else, it is not useful because the process neighborhood is no longer valid

once execution ends. Thus it would be a waste of space to write it out to file. It is a particular example of advantage received by running ParaView *in situ* as opposed to only executing post-process on the file output.

Finally, we modify the contouring algorithm in ParaView to take advantage of this array if it is available. Since we're running ParaView in the same job as the simulation all the process numbers are identical, and we can use these numbers as a replacement for the all-to-all communication.

4 RESULTS

We ran both the unmodified version of the algorithm and the version with the extra neighbor array passed by CTH on the same pipe problem. In CTH, problem size is managed by setting the max refinement depth of the AMR mesh. Each time the simulation refines, the problem can increase by up to 8 times larger, in practice it less. Thus when scaling up we run with a particular max depth and increase only when we fall too low in the number of blocks per core. In the results shown we change this depth at 32 cores and at 256 cores, repeating the runs at those points with the larger size.

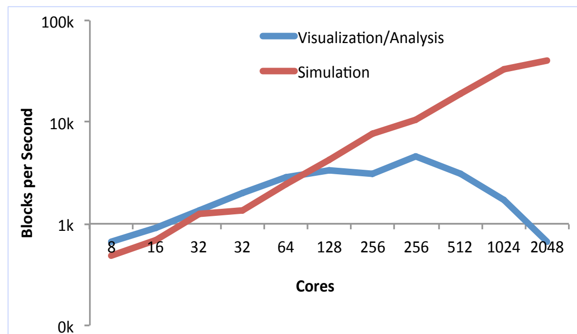

Figure 4: Performance of the normal AMR contouring algorithm. Because it must do an all-to-all communication in order to determine neighbors, it does not scale well beyond 128 cores. Note, the repeated values for 32 cores and 256 cores are because we change the max refinement depth in the simulation. Between those points the refinement depth is unchanged, so the number of total blocks is also unchanged.

As seen in Figure 4 the all-to-all broadcast begins dominating the time at around 128 cores. This is not necessarily a problem when running post-processing for two reasons. The first is that visualization clusters have been traditionally only about 5 percent of the size of the simulation, thus analysis algorithms like this one do not need to scale as well. The other reason is that although it may not be scaling well at 1024 or 2048 processors running offline it is only a difference of a few seconds.

However, once we do pass the neighbor information over from CTH and take advantage of it in the contouring algorithm, we are no longer dominated by the all-to-all broadcast, Figure 5.

max refinement depth	original	fragment surfaces only
3	165	14.6
4	391	32.2
5	919	68.2

Table 1: Comparison of file sizes for the original simulation file generated from 1000 steps of the simulation and the corresponding files containing fragment mesh surfaces. All file sizes are in megabytes.

Finally, we compare the size of the original file to the number of fragments found. Although we do not consider the statistics being measured and output on these results, we can assume that it will be a

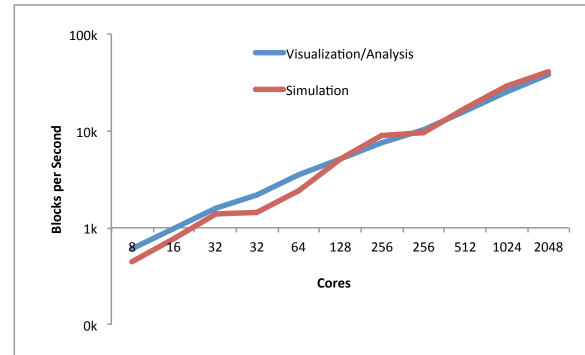

Figure 5: Performance of the specialized AMR contouring which takes advantage of the neighbor information already being processed and passed from CTH. This algorithm is no longer bound by the all-to-all as in Figure 4. Also note, the 32 and 256 core counts are repeated in the same way as in Figure 4

constant multiple of the number of fragments in the simulation. As can be seen in Table 1, the fragment surfaces files are much smaller than the original files and will use less disk bandwidth to write out. Although there is a cost to running the fragment detection *in situ* with the simulation, we expect that as we move into exascale the time taken to write increasingly large full results will be much larger than the fragment detection cost. And, more importantly, it will be outweighed by the savings because writing only the fragment data will be so much smaller.

5 CONCLUSION AND FUTURE WORK

We have shown that by taking advantage of the processing in CTH, we are able to scale up fragment analysis and run it along side the simulation without slowing down and still getting the analysis results we need, thereby saving the time necessary to write out the full simulation results. *In situ* processing will be a useful way of performing analysis at exascale [4], but it will be necessary for the analysis algorithms to scale effectively to these large scales, so that it does not become a bottleneck to running the simulations. For those algorithms that do not necessarily scale well, as in the case of the AMR contour algorithm, we may need to forward some of the processing the simulation is already doing so that it can be taken advantage of by the analysis code.

Although passing the neighbor information is effective when running *in situ* because we are in the same memory space, it is not something we can immediately take advantage of to improve post-processing. In addition, in-transit processing is which is an *in situ* type method of processing where data is passed through the network from the simulation job to a separate MPI job space running the analysis. While in-transit has advantages over in-memory *in situ* because in-transit runs out-of-core and asynchronously, it cannot as effectively take advantages of information like these block-neighbors because the process numbers are different between the two MPI jobs. Thus it must either be ignored in favor of neighbor discovery via all-to-all in the second job, or a mapping must be created between processes in the original simulation job and those being passed data in the analysis job.

We will continue to examine the entire pipeline of algorithm processing for fragment characterization for any other bottlenecks or opportunities to take advantage of close coupling nature of *in situ*.

ACKNOWLEDGEMENTS

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department

of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

[1] K. Bonnell, D. Schikore, K. Joy, M. Duchaineau, and B. Hamann. Constructing material interfaces from data sets with volume-fraction information. In *Visualization 2000. Proceedings*, pages 367 –372, oct. 2000.

[2] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Geveci, M. Rasquin, and K. E. Jansen. The ParaView coprocessing library: A scalable, general purpose in situ visualization library. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization*, pages 89–96, October 2011. DOI 10.1109/LDAV.2011.6092322.

[3] E. S. H. Jr, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington. Cth: A software family for multi-dimensional shock physics analysis. In *Proceedings of the 19th International Symposium on Shock Waves*, pages 377–382, 1993.

[4] K.-L. Ma, C. Wang, H. Yu, K. Moreland, J. Huang, and R. Ross. Next-generation visualization technologies: Enabling discoveries at extreme scale. *SciDAC Review*, (12):12–21, Spring 2009.

[5] A. H. Squillacote. *The ParaView Guide: A Parallel Visualization Application*. Kitware Inc., 2007. ISBN 1-930934-21-1.