Architectural Challenges and Solutions for Petascale Postprocessing

Hank Childs

Computer Scientist, Lawrence Livermore National Laboratory

E-mail: childs3@llnl.gov

Abstract.

With petascale applications comes petascale data; gleaning knowledge and insight from this large-scale data is widely accepted to be a limiting factor in many fields of scientific endeavor. Large-scale data presents two incredible challenges: scale and complexity. The scale challenge refers to the problem of being able to process tremendous numbers of bytes of data within some time constraint. The complexity challenge is more subtle; petascale data is inherently complex and reducing this data to comprehensible forms is difficult.

For the scale challenge, it is often assumed that continued usage of techniques viable at the terascale will provide a recipe for success at the petascale. We argue the opposite: following this path will result in cost prohibitive solutions. Instead, we must pursue "smarter" techniques, such as in situ and multi-resolution processing, which are well established by a decade's worth of research.

The forms of postprocessing are very diverse, but they can be summarized in broad use cases: data exploration, presentation graphics, quantitative analysis, comparative analysis, and visual debugging. Together, these use cases are responsive to the complexity challenge. Moreover, these use cases are responsive to the demands of the user community. Making pretty pictures is just one component of the simulation community's postprocessing needs and all of these needs must be addressed for petascale data.

A major focus of this paper is to construct a road map for realizing this broad set of use cases at the petascale, and, further, to do this in the most economical way possible. None of the smart processing techniques we consider are a panacea; none are capable of supporting every use case. However, a software architecture that underpins these processing techniques and that can dynamically select between them will provide an economical way to meet these challenging requirements, and in this paper we discuss such an architecture.

1. Introduction

As noted in the abstract, gleaning knowledge and insight from large-scale data is widely accepted to be a limiting factor in many fields of scientific endeavor. The computational science community is approaching petascale level simulations, which will produce extremely large data sets. Postprocessing this data is both critical and challenging. Continuing with the current techniques for production, terascale postprocessing will result in prohibitive hardware costs. Instead, we propose a change in strategy: a system (that is, software architecture) that employs varied techniques based on situation. This system, in turn, will enable petascale postprocessing in a production setting and in many diverse forms, all with substantially reduced hardware costs.

In Section 2, we describe "pure parallelism", the technique commonly used for production visualization and analysis at the terascale. We also describe why this technique will likely be cost prohibitive at the petascale. In Section 3, we survey the "smart" alternatives to pure parallelism and discuss their relative costs. In Section 4, we argue that many use cases that go beyond traditional visualization should be considered for two reasons. One, these use cases are responsive to the mandate of the user community and they will enable users to reduce the incredible complexity of petascale data. Two, the increased cost of entry for petascale postprocessing will result in

economies from considering these use cases simultaneously. In Section 5, we survey the use cases themselves: data exploration, quantitative analysis, presentation graphics, visual debugging, and comparative analysis. In Section 6, we discuss the viability of our smart processing techniques for each of the use cases, and argue that a combination of techniques is needed to provide the most cost effective solution. Finally, in Section 7, we describe a system where algorithms can be implemented independently of processing technique (such as pure parallelism or its smart alternatives), and, further, where the system can adaptively switch between these techniques on an as-needed basis.

2. The problem with pure parallelism

"Pure parallelism" refers to the technique where many processors are used to read the entire data set into primary memory. This is the simplest technique to implement; the data is stored in its natural form and every byte of data is available on demand. At the terascale, production visualization tools, such as VisIt[4], EnSight[5], and ParaView[9], successfully used this technique for the processing of data. They partitioned the data and had each processor operate on its respective subset, with the tools primarily distinguished by the level of communication permitted amongst its processors during execution.

The hardware to enable this processing takes one of two forms. With one form, there is a dedicated mini-supercomputer for postprocessing tasks. In this case, the data is either transferred to a private disk for the machine, or the data is directly accessible from the supercomputer through a shared disk. With the other form, the supercomputer itself is used to visualize and analyze the results after the simulation has completed, either through dedicated postprocessing nodes or by using the same batch nodes used to run the simulation.

The problem with continuing on the pure parallelism path is that the hardware costs will be prohibitive. Currently, the massive amounts of data produced by Lawrence Livermore's 360 TeraFLOP BlueGene/L machine are processed by a separate, dedicated machine, Gauss. Gauss is a 512 processor Linux cluster which itself made the Top500 and cost nearly two million dollars. When planning for a similar setup with a five PetaFLOP machine, the estimate for a separate, dedicated postprocessing cluster was about *fifteen million dollars*. Although this cost *may* be justifiable in the context of a larger supercomputer purchase, it is probably not justifiable given that software alternatives can substantially reduce these hardware requirements.

The notion of co-opting a portion of the supercomputer to do interactive postprocessing is also not very palatable. Postprocessing is typically done interactively, resulting in an uneven and "bursty" usage of the computer. The computer utilization oscillates between zero and one hundred percent usage. During a "non-burst," is this portion of the supercomputer to sit idle, waiting to respond? Is that cost justifiable?

Another problem with co-opting the supercomputer is the trend toward multi-core processing. Postprocessing requires a different architectural balance of computing resources than simulation; it is typically a memory- and I/O-limited activity and is rarely compute-limited. As petascale platforms tend towards increasing compute power, with memory and I/O not keeping pace, the net increase in hardware capability delivered for postprocessing is not proportionate. Hence, postprocessors will need greater resources to handle the considerably larger data sets, and will need an increasing proportion of the supercomputer to do its job.

To summarize, planning on using pure parallelism at the petascale entails either making an expensive computer purchase or co-opting a significant portion of the supercomputer, diminishing its effectiveness.

3. Large Data Processing Techniques

In this section, we survey the techniques for large data processing that have already been investigated and/or developed by the research community. As we will see in later sections, many of these techniques perform well with certain postprocessing use cases, but not others.

For each section, we start by describing the technique, then describe its advantages, disadvantages, and challenges. Disadvantages are problems inherent with the technique; challenges are things the technique currently does not do well, but could be improved.

3.1. In situ processing

3.1.1. Description In situ processing consists of processing the data using the resources allocated for the simulation code. In this model, the simulation code advances in time for a while, then hands off a baton to a postprocessing algorithm, which generates results and hands the baton back[1].

[1] Note that this model blurs the line of the term "postprocessing", since analysis is done while the simulation code is running, rather than after it has finished. Regardless, we continue to use the term, because the analysis is still performed after the simulation code has calculated a result, even though the period of time afterwards is only fractional seconds.

3.1.2. Advantages This techniques sidesteps I/O costs and will always have sufficient compute resources.

3.1.3. Disadvantages In situ processing requires that the user knows exactly what analysis should be done as the simulation is running, which is often not the case. Also, some analysis is inherently time consuming, like with exploratory visualization. As in situ processing also occupies the entire supercomputer while it is running, this use of resources can be very expensive. Further, there is an entire class of operations that look at an event and want to trace *backwards in time* how that event came to be. This is not possible with in situ processing. Finally, there is a tendency to refrain from I/O altogether when in situ processing is employed. If this is the case, the only recourse for performing new analysis is to run the simulation again.

3.1.4. Challenges Visualization and analysis software is typically developed to work with many simulation codes, and there is an economy in this re-use among the codes. However, for in situ processing, developing a single solution that will be effective for many simulation codes is a major challenge. Postprocessing algorithms often fix the way they represent different mesh types and fields on those meshes. For example, a postprocessing algorithm may assume the fields are in "row-major order." But each simulation code has its own in-core representation. One code may have "row-major order", but another may have "column-major order." In this case, the postprocessing algorithm can directly operate on the data of the "row-major order" simulation, but, in the "column-major order" case, the algorithm must convert the simulation data to "row-major." This conversion step leads to memory bloat, which is a significant problem. Although, high end supercomputers have lots of memory, this resource is still often the limiting factor for the resolution of simulations; tolerating this conversion step will only worsen the dearth. (It should be emphasized that the row-major / column-major example is just one of *many, many* possible pitfalls. Experience shows that the odds of a fixed data model of a postprocessing tool matching that of a simulation code are very low, unless the postprocessing tool was built around the simulation code.)

The challenge, then, is to implement algorithms in a way that is indifferent to the simulation code's in-core layout, and, of course, still maintain high efficiency. One approach is to leave the in-core layout as is and use abstract interfaces (e.g. virtual functions) to hide the data layout from the postprocessing algorithms. Depending on the extent that virtual functions are used, this may be adequately efficient. However, the SCIRun library[8] demonstrated an even more successful approach to this problem, by using a templated data model. This templated approach allows for high efficiency, in addition to solving the data layout issue.

Finally, performing in situ processing on petascale computers would force common postprocessing algorithms to be run on an unprecedented numbers of processors. The extent to which these algorithms will be efficient at this scale is an open question.

3.2. Multi-resolution techniques

3.2.1. Description With multi-resolution techniques, the data set can be explored by starting with a coarse representation and then focusing in on portions of the data set to see finer representations. Finer representations may come automatically after coarse representations finish calculating, or they may be explicitly requested by the analyst. Ultimately, enough replacements with finer and finer representations will return the original data (at least desirably).

3.2.2. Advantages These techniques have been greatly studied by the research community (for example, [10] and [12]), so their viability is well established. For use cases where a coarse representation (potentially augmented by some fine representations) will suffice, this technique is a tremendous win, since it substantially reduces I/O and processing costs.

3.2.3. Disadvantages For other use cases, offering results on a coarse data set is not meaningful, because too much detail is lost. Further, the multi-resolution representation introduces costs, both to compute and to store. Given that most simulations output at a low rate (to save on slow I/O), however, these costs may be insignificant.

3.2.4. Challenges When a coarse version of the data set is displayed, it is important to understand the accuracy of the results. Some multi-resolution techniques emphasize performance through elegant indexing, but de-emphasize control over what data is displayed at coarse resolutions and de-emphasize understanding of the inaccuracy of the intermediate results.

Finally, some multi-resolution techniques lose their knowledge of the original data set. Some use cases require that results be returned in a format familiar to users. For example, a query that returns an element identifier for a block-structured simulation would want that identifier to include information like which block the element is contained in, as well as its logical I, J, and K indices. The challenge is to apply a multi-resolution hierarchy to this data with minimal overhead, and still be able to produce such information.

3.3. Out-of-core

3.3.1. Description "Out-of-core" techniques[15] consist of partitioning the data set into subsets, and then processing each subset one at a time. Each subset must be small enough to fit into primary memory. After processing each subset, it is discarded, in all or in part, before moving on to the next one.

3.3.2. Advantages This technique does not require parallel hardware. By reducing the memory needs and operating on subsets, it can process any amount of data with a simple desktop machine.

3.3.3. Disadvantages Out-of-core techniques often are not able to meet interactivity requirements. I/O is often the bottleneck for postprocessing, even when doing parallel processing, and serial out-of-core will only encounter this problem all the more so. Of course, out-of-core can be parallelized, but we do not consider this variant in this paper, as the purpose would be to increase interactivity and the number of processors to achieve this interactivity would approach that of pure parallelism. Another approach to address I/O issues would be to augment out-of-core techniques to read less data, for example only reading the portion of the data set that intersects a slice or lies on a contour (see [3]), but these approaches are not widely applicable to the diverse use cases we are considering.

3.3.4. Challenges Some algorithms require repeated access to data. These algorithms are, at a minimum, difficult in this setting, because data must be fetched from disk multiple times. However, clever implementations of these algorithms may be able to minimize re-reads, or even eliminate the re-reads altogether in some cases.

3.4. Pure parallelism

3.4.1. Description "Pure parallelism" uses many processors to read the entire data set into primary memory. The data is stored in its natural form and every byte of data is available on demand. We argued in Section 2 that regular use of this technique will be cost prohibitive. Despite these costs, pure parallelism will likely have a role in the petascale future, primarily as a backup. The question is to what extent.

3.4.2. Advantages This processing paradigm is suitable for all use cases we consider.

3.4.3. Disadvantages The hardware costs for pure parallelism have the potential to be large.

3.4.4. Challenges I/O represented the significant bottleneck at the terascale, and this problem will only worsen in the petascale regime, since processing power is advancing much more quickly than I/O rates. Further, many of the future petascale supercomputers being considered have lightweight operating systems that create barriers for deploying tools, for example precluding sockets, shared libraries, virtual memory, and/or threads. (This issue also exists for in situ processing.) Both of these issues must be addressed for a pure parallelism approach to be successful.

3.5. Cost comparison of techniques

Calculating exact costs for these techniques is difficult and highly dependent on the problem being solved. However, the differences in cost between the techniques are large enough that they can be assessed in gross terms. The two axes of comparison to consider are *machine costs* (the cost of using the necessary hardware to perform the processing) and *runtime costs* (the time a user must wait to do the required analysis):

- For pure parallelism,
  - the machines costs consist of procuring and maintaining a postprocessing-oriented supercomputer or of dedicating a portion of the supercomputer for these purposes.
  - the runtime costs are dependent on the amount of parallelized I/O and processing capabilities. Given adequate resources, this can be very fast.
- For in situ processing,
  - the machine costs are the overhead to have the supercomputer do the postprocessing work and *not* be advancing in the simulation. However, the costs are substantially reduced because the data is already loaded in main memory, saving on very expensive I/O costs.
  - the runtime costs are very low, since the adequate resources are guaranteed and I/O is less of a concern.
- For multi-resolution techniques,
  - the machine costs come from the overhead of outputting multi-resolution information (see Section 3.2.3). The rest of the processing is then done with a desktop computer or a small cluster.
  - the runtime cost is very cheap (provided that multi-resolution is a general match to the use case).
- For out-of-core techniques,
  - the machine costs are negligible.

- the runtime costs are often prohibitive (see Section 3.3.3).

It is difficult to combine our non-quantitative runtime and machine costs into a "total" cost. That said, in situ and multi-resolution techniques appear to be the most viable for use with petascale data, as out-of-core often has prohibitively high runtime costs and pure parallelism may incur prohibitively high machine costs. Table 1 summarizes the preceding paragraphs. For runtime costs, multi-resolution techniques were listed as cheaper than pure parallelism because they often can avoid severe I/O costs. It could be reasonably argued that these two should be reversed.

Technique	Machine Cost (lowest is best)	Runtime Cost (lowest is best)	Total Cost (lowest is best)
Pure Parallelism	4	3	2 (tie)
In Situ	3	1	1 (tie)
Multiresolution	2	2	1 (tie)
Out-of-core	1	4	2 (tie)

Table 1. The costs of the processing techniques.

4. The Argument for Diverse Use Cases

A principal assumption of this paper is that it is advantageous to both developers and users to consider a diverse set of use cases. In this section, we provide two arguments in support of this assumption. One, the user community relies on diverse use cases that go beyond classic scientific visualization and expect that these approaches will be available at the petascale. Further, petascale data is inherently complex and a diverse suite of techniques are needed to reach crucial insights. Two, an economy is achieved when the diverse use cases are addressed simultaneously.

4.1. Beyond Meat Grinders

"Meat grinder" is a term that describes tools that focus solely on handling the scale of data without considering the associated complexity. The analogy is that these tools simply change the form of the data (from meshes to images), cranking data through without bothering to make it easier to interpret. Complex isosurfaces often go beyond what our human brain's visual processing system can meaningfully understand. Following the analogy, then, we should spend more time "finding the right meat." We need to complement techniques such as isosurfacing, slicing, and volume rendering with techniques that more directly allow analysts to do the science they need to do. This often comes in diverse forms, such as:

- quantitative measures, like calculating a probability density function,
- comparative techniques, like studying why a high resolution simulation doesn't match smaller predecessors,
- the ability to dig down and study the behavior of just a handful of data points.

To truly understand petascale data, addressing the scale is necessary, but not sufficient. Focusing exclusively on data exploration will deny analysts what they need: a diverse suite of techniques capable of enabling crucial insights.

4.2. Economies Achieved by Considering Diverse Use Cases

To what extent should there be a common infrastructure for delivering these diverse use cases? At one extreme, does it make sense to target a single tool/library that can employ each of the postprocessing techniques from Section 3 for our target use cases? Or are there distinctions that make multiple tools/libraries useful and/or necessary? We believe the answer is that combining the solutions for these postprocessing areas into a single solution is beneficial from a cost perspective. We will make this argument in the following sections, from both a development and a user perspective.

4.2.1. Development Economies The techniques described in Section 3 all have higher software costs to implement than that of pure parallelism. Thus, the cost of entry for postprocessing will rise for the petascale regime. Further, many distinct use cases can be solved with the same or similar solutions. Hence, from a development perspective, there are economies that make a single tool/library (or small set of tools/libraries) desirable. The biggest example is that a large data infrastructure can be developed once and reused in all of these areas (the nature of this infrastructure will be discussed in Section 7). In addition, file format readers, algorithms, and data

models all benefit from this reuse. Also, many previous solutions in this space (for example, AVS[16], VTK[13], and OpenDX[1]) have shown that a flexible, extensible architecture is advantageous. This underlying architecture would also be reusable. The advantage, then, of a single tool/library is that all of these assets can be developed a single time and reused. Finally, the system we propose in Section 7 will allow algorithms to be implemented in a way that is indifferent to processing technique. So a developer can implement an algorithm one time, and achieve a great economy; the algorithm would then be deployable with pure parallelism, out-of-core, in situ, and multi-resolution processing[2].

4.2.2. User Economies From a user perspective, there are economies to having a single tool (*not library*) that can accommodate all of these use cases. First, visualization and analysis tools tend to have complex features, and hence complex interfaces. Learning one interface, as opposed to five or more, is often preferred by users. Further, users tend to jump back and forth between use cases. For example, a session focusing on comparative analysis often leads to some key discovery that then leads to visual debugging on one of the simulations. From within a single tool, they can do this seamlessly. With multiple tools, there is overhead.

4.2.3. Realities of General Solutions The argument from the preceding sections is that, especially for petascale postprocessing, there is an advantage to having a system (software infrastructure) where assets can be re-used, and, for users, there is an advantage to having a single interface to perform all of their use cases. This should be tempered by experience, however. In the world of visualization tools, there is no "one size fits all" solution. Some scientific domains are so specialized that there is little benefit from leveraging a generalized postprocessing infrastructure. Conversely, making a postprocessing infrastructure that is so general that it can support all scientific domains can substantially increase development time and reduce performance.

5. Postprocessing Use Cases

In Section 4, we motivated the power of a diverse set of use cases. They are what customers demand, they allow for better understanding of complex data, and we can achieve both developer and user economies by considering these use cases simultaneously. In this section, we explore each of the use cases: data exploration, quantitative analysis, comparative analysis, visual debugging, and presentation graphics. For each use case we will present a description, discuss requirements placed on processing techniques, and, for some of the use cases, discuss new challenges created by large scale data.

5.1. Data Exploration

5.1.1. Description Data exploration includes the standard scientific visualization techniques, for example slicing, contouring, and volume rendering. These techniques employ our sophisticated visual processing system to quickly identify trends and anomalies in data.

5.1.2. Requirements for Processing Techniques With respect to processing, we need to keep the "bursty" model of data exploration in mind. It is expected that data exploration is interactive. Further, it must be understood that data exploration has periods of interactivity, when results are considered, and that idle time must be weighed against the resources used to produce results.

5.1.3. Large Data Challenges There are only so many pixels on a computer screen (and only so many rods in your eyes). Petascale simulations have so much data that many, many data points will be encoded in a single pixel. Will this prevent analysts from locating anomalies, such as hot spots? We need to ask to what extent classic data exploration techniques must be adapted to guarantee that what is presented to an analyst accurately reflects the simulation.

5.2. Quantitative Analysis

5.2.1. Description "Quantitative analysis" is a broad term and it is useful to think of three areas:

- Basic techniques that span scientific domains
- Advanced techniques that span scientific domains
- Techniques that are tailored to a specific physics area.

The motivation for artificially distinguishing between basic and advanced techniques is that some people hear the phrase "quantitative analysis" and think of *only* basic operations such as integrating density over a volume to get a mass, calculating a surface area, or calculating a flux. These operations are often among the most useful, but, to many others, this represents only the beginning. Examples of advanced techniques are calculating characteristic functions, such as chord length distributions[11]. Techniques tailored to specific physics areas vary highly, but a

[2] Some algorithms place constraints on the types of processing they allow; this issue will be discussed further in Section 7.

good example is that of "virtual diagnostics," which allow for simulations to be compared to actual experiments, through comparison to real diagnostics.

5.2.2. Requirements for Processing Techniques Techniques tailored to a specific physics area create the most new requirements. These techniques must often be performed at the full resolution of the data, not on simplified versions. Further, some of these techniques may require incorporating outside knowledge, such as equation of state information. Finally, many of these techniques require development from personnel that have, for example, a background in physics. So the development environment must require that people with a different set of skills (i.e. non-visualization personnel) be able to effectively interact with the environment provided to them. This environment may have many instantiations. One example is a stripped down programming environment, where personnel with domain expertise can implement algorithms in the C programming language on arrays of data. Another example could be a scripting environment where experts use a language, like Python.

5.2.3. Large Data Challenges Even at the terascale, physics-based analysis was often addressed by serial tools written directly by simulation code developers and their customers. Anecdotally, these serial tools will run for a month to get an answer. A similar approach at the petascale, assuming a one thousand fold increase in data size, would lead to a runtime of one thousand months – over eighty years! Where one month may be an acceptable turnaround time, eighty years definitely will not be. We conclude that the standard operating procedure at the terascale will lead to poor results at the petascale. Of course, these serial, specialized analysis tools could be parallelized. But, again, petascale has raised the cost of entry. The point, of course, is that petascale simulations are creating a need for a customizable petascale analysis environment.

5.3. Comparative Analysis

5.3.1. Description Comparative analysis is also a broad term. Shen et al.[14] define three basic areas of comparative visualization: image-based comparison, data-level comparison, and topological comparisons. Image-based methods compare images that result from visualization algorithms. Most image-based comparisons[6, 17] perform image differencing algorithms on images from multiple inputs. These systems are limited to comparing visualizations where the entire data set can be represented by a single image. However, these techniques are extremely important in the context of comparison to experimental results. Data-level comparisons [14] place fields from one mesh onto another mesh and use derived quantities to examine differences between the two data sets. Topology-based (or feature-based) techniques compare features of the data sets [2, 7]. Typically, they survey the data set, create summaries of the data, and develop methods to visualize and compare these summaries.

5.3.2. Requirements for Processing Techniques Comparative analysis requires processing multiple data sets simultaneously, for example multiple time slices of a single simulation or multiple simulations from an ensemble. Finally, these techniques must also often be performed at the full resolution of the data, not on simplified versions.

5.4. Visual Debugging

5.4.1. Description For simulation code developers, visual debugging is often the number one use case for a postprocessing tool: it is the primary means for finding bugs in the simulation code. Examples of these techniques include locating a hot spot, finding an area of mesh tangling, or "picking" on an element and getting a report about it, such as the value of the element, the value of its neighbors, etc.

From the perspective of data processing, there are two major classes of visual debugging: aggregate queries and individual queries. An example of an individual query is a code developer noticing a hot spot and wanting to obtain more information about it. An example of an aggregate query is a code developer wanting to survey the entire data set and obtain a list of elements that meet some criteria. The difference between these two classes is the resources necessary to satisfy these queries.

5.4.2. Requirements for Processing Techniques Visual debugging requires information about the simulation to be produced in a manner that respects the original layout of the simulation. For example, any query that returns an element identifier must return the information in a way that makes sense to the simulation code developer.

5.5. Presentation Graphics

5.5.1. Description Presentation graphics is one of the most time consuming activities for visualization experts. This term includes moviemaking and making images for viewgraph presentations, publications, etc. Again, we distinguish between two classes of presentation graphics: "presentation-oriented graphics" and "exploration-oriented movies." "Presentation-oriented graphics" are typically high quality. Their purpose is to inform a large audience. "Exploration-oriented movies" are typically done by analysts for themselves. They make simple movies so they can observe, and hopefully ultimately understand, phenomena in the simulation. In many ways, this use case is very similar to data exploration, with the key differences being that the explorations are saved in a permanent

record (for example, an MPEG) and that various forms of automation of algorithms (e.g. animating a slice, an isovalue, or over time) are required. Presentation-oriented graphics also augment the images with annotations like time sliders, logos, and text.

5.5.2. Requirements for Processing Techniques With presentation-oriented graphics, it is assumed that the images represent the entire data set, as the images (even if they are frames of a movie) are often studied in great detail and there is often a desire to "show off" the full resolution of the data. This requirement is relaxed with exploration-oriented moviemaking; the movies often don't require the full resolution of the data.

6. Viability of Techniques for Use Cases

Having surveyed the processing techniques (Section 3) and the use cases (Section 5), we can see that not all of the smart processing techniques will be applicable to the use cases. The mode of failure always follows the same formula: processing technique "ABC" does not have property "IJK", which is required by use case "XYZ". In Section 6.1, we will enumerate the failure modes. This is followed by a summary of the viability of techniques for the use cases, in Section 6.2, which provides a road map for the petascale postprocessing of our uses cases.

6.1. Failure modes

For each of the following properties (the "IJK"s), the property is required by a use case, but is not supported by a processing technique. The properties are:

- Processing the data must take place in its full and native form
- Multiple data sets must be supported simultaneously
- Processing must occur interactively
- Idle time between processing requests will occur

6.1.1. Processing the data in its full and native form Multi-resolution techniques do not process the data in its full and native form[3]. But this representation is a requirement for all areas of quantitative analysis and comparative analysis, aggregate queries for visual debugging, and presentation-oriented graphics. Therefore none of those use cases can be fully solved with multi-resolution techniques.

It should be noted that multi-resolution may still have value added in the areas of comparative analysis and quantitative analysis. Here, the initial, "rough" answers from coarse data can guide more in depth queries in the data set. At some point, both comparative analysis and quantitative analysis almost always dictate results at the true resolution of the data. (Note that the system described in Section 7 will allow for these results to be taken on a coarse representation, putting the decision in the hands of the user.)

6.1.2. Supporting multiple data sets simultaneously In situ processing will only work with the simulation's current data set. So it is not able to support multiple data sets simultaneously. Further, comparative analysis requires processing multiple data sets (e.g. multiple time slices from one simulation, multiple simulations, etc.). It should be noted, however, that clever configurations could overcome these limitations, for example having a client connecting to two in situ-based servers simultaneously. Generally speaking, however, in situ processing and comparative analysis are not compatible.

Some advanced and physics-based quantitative analysis techniques have a time component, which can be difficult (going forward in time) or impossible (going backwards). So in situ processing should be considered on a case-by-case basis for these areas.

6.1.3. Interactivity Out-of-core processing is not able to provide interactivity (see Section 3.3.3). Further, data exploration and aggregate queries from visual debugging require interactivity. So out-of-core is not appropriate for these use cases. It should be noted, however, that individual queries from visual debugging also require interactivity, but these techniques do not require petascale data processing, making out-of-core a viable approach.

6.1.4. Idle time In situ processing should not be used when the postprocessing is "bursty". The inherent idle time is not a good use of the supercomputer. Data exploration and visual debugging often lend themselves to idle time, so, in general, in situ processing should not be used for these uses cases.

Of course, in some cases, the ability to ask questions interactively can be a time saver, for example when the alternative is to save the state of the simulation, analyze it offline and then later reload it. Or when the simulation is about to veer off track and interactive inspection can identify this and right the situation (i.e. computational steering). These examples argue that bursty use cases can be extremely useful with in situ processing. Yet, the

[3] They can do this with enough refinement, but this essentially becomes out-of-core in this case.

	Pure Parallelism	In Situ	Multires	Out-of-core
Data Exploration	Yes	No	Yes	No
Quantitative Analysis				
Basic	Yes	Yes	No	Yes
Advanced	Yes	Sometimes	No	Yes
Physics-based	Yes	Sometimes	No	Yes
Comparative Analysis				
Image-Based	Yes	No	No	Yes
Data-Level	Yes	No	No	Yes
Topological	Yes	No	No	Yes
Visual Debugging				
Individual Queries	Yes	No	Yes	Yes
Aggregate Queries	Yes	Yes	No	?
Moviemaking				
Presentation-oriented	Yes	Yes	No	Yes
Exploration-oriented	Yes	Yes	Yes	Yes

Table 2. The viability of postprocessing techniques for each of the use cases. If a technique is viable for a use case, it is labeled "Yes", "No" otherwise. For each use case, a preferred solution is selected and colored green. Back up solutions are colored yellow.

overarching point from this section is that we cannot plan on using this technique to serve these use cases on a regular basis.

6.2. Summarizing the viability of processing techniques for use cases

The last section established which processing techniques were suitable for our use cases. We summarize these findings in Table 2. We incorporated the cost table (Table 1) to decide which processing techniques are preferred for addressing each of our use cases. We color the preferred techniques green. In the case of in situ processing, we also choose a backup solution, colored yellow, since in situ processing can only be used when the user knows what analysis should be done a priori.

Table 1 declared a tie between in situ processing and multiresolution techniques. From Table 2, we see that "resolving the tie" between these techniques is not necessary as they are mutually exclusive for the use cases we consider. Further, in situ processing or multi-resolution techniques can be used to address many of our use cases, and only comparative analysis falls back to a "high cost" solution. Since each row has at least one "Yes", the set of use cases we want to consider is fully covered by our processing techniques; these techniques should be sufficient to do petascale processing. In the cases where in situ was selected, either out-of-core processing or pure parallelism play a larger role as a backup. Choosing between out-of-core and pure parallelism force a trade off to be made between runtime costs and machines costs, respectively, which should be taken on a case-by-case basis.

This analysis has taken place with somewhat broad strokes. Although the presentation of this information is black-and-white (Yes or No), there are exceptions, especially when a group is listed as "No." The purpose of Table 2 is to make a road map for petascale postprocessing for our diverse use cases in a production environment. Further, this table motivates some important points:

- We will need to employ *multiple* smart techniques adaptively, based on the work being done.
- We will still have to co-opt the supercomputer to do some of our required processing.
- By using smart techniques, we will be able to prevent this co-opting for the majority of the time.

The remaining question, then, is the form to deploy these techniques, which is the subject of the following section.

7. An Architecture for Supporting Petascale Postprocessing Use Cases

We will now describe a system that will allow us to realize, in an economical way, Section 6.2's road map to deliver all of our use cases at the petascale. The vision is to have a single system (or software architecture) that can

utilize different processing paradigms. In this proposed system, algorithms can be implemented in a manner that is indifferent to the processing paradigm, at least indifferent to the extent possible. This will be done by using data flow networks (described in Section 7.1) and then controlling how data flows through these networks (described in Section 7.2).

7.1. Data flow network overview

The most prevalent design for postprocessing libraries is data flow networks. In this design, algorithms are implemented in a manner that focuses only on its input and output. This approach allows for algorithms to be interoperable, to be "plugged together" in dynamic ways by users to fit their current needs. Further, it is easily extensible; a new algorithm can be implemented by simply creating a new module.

Data flow networks are not without faults. They normally require the data model to be somewhat fixed[4] and each algorithm must be aware of the data model, or, at a minimum, state which parts of the data model they support. Further, generally speaking, the modular design means that algorithms are executed one at a time, rather than all at one time. The latter is typically more efficient (for one, it avoids thrashing memory when the data being processed at that moment is sufficiently large), but takes much more development time.

7.2. Processing of data

In terms of processing data, the important requirement is that each algorithm is implemented in a way that accommodates processing of subsets of the larger data set. Each of the "smart" techniques can then be implemented using data flow networks and allowing the underlying system to control how data flows through.

- For in situ processing, the data flow network will be instantiated on the simulation code's processors and the data operated on will be each processor's portion of the data set.
- For out-of-core processing, one data flow network will be instantiated, and subsets of the larger data set will be read and sent through the network one at a time.
- For multi-resolution, the data at any given time will be some coarse representation of the larger data set, with previous executions of the network being wiped out as they get replaced by finer versions of the data set.
- For pure parallelism, each processor will instantiate an identical data flow network, the data set is partitioned, and each processor is assigned a subset.

Some algorithms can not be implemented in a setting that is unaware of how the data set, as a whole, is being processed. For example, algorithms that require all of the data to be in memory at one time cannot support out-of-core techniques. One solution to this problem is to have the algorithms communicate their needs to a data flow network "executive." The executive can then make decisions about which modes of operation are possible. The contracts mechanism proposed in [3] is ideal for this type of communication.

That said, resources for a given analysis are typically fixed while the analysis is running ("I will be running in situ", "I have to run out-of-core", etc.), so the executive may be limited to aborting the operation and recommending a different processing paradigm.

7.3. Viability

Although the proposed system is daunting, the good news is that work has taken place to establish its viability. The VisIt project (of which the author of this paper is a member) targets all five of these use cases and has demonstrated promising results in the area of dynamic processing paradigms. VisIt has full support for pure parallelism. It also supports out-of-core processing, although some of its algorithms are not possible in an out-of-core environment. Those algorithms set a field in a contract and an executive disables the out-of-core mode. (It then falls back to pure parallelism.) VisIt also is capable of in situ processing, although it suffers from the data model conversion issue described in Section 3.1. There is no multi-resolution support, however[5]. Regardless, the ability to provide an environment that allows for algorithms to be implemented indifferent of processing paradigms, combined with multiple processing paradigms being provided is promising.

8. Summary

The ability to postprocess petascale data will be crucial to the success of petascale simulations. The current path being used for terascale postprocessing (pure parallelism) will result in prohibitive hardware costs. Worse, we must

[4] Fixed in the sense that adding new fields type (like edge-centered data) or new meshes (like ying-yang meshes) is difficult.

[5] There is no support for level-of-detail processing. There is full support for Adaptive Mesh Refinement (AMR) data, but the user controls what level of refinement is viewed, based on the available resources.

consider many diverse use cases simultaneously. However, we can address this problem by employing a variety of "smart" processing techniques. None of the techniques is a panacea alone. Instead, we must deliver a system that can adaptively employ these techniques in the appropriate situations. And, further, we must understand when these situations occur. All of these ideas have been fleshed out in this paper, to ultimately form a cost effective road map for proceeding into the area of petascale postprocessing.

9. Acknowledgments

This work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). In addition, this work was performed under the auspices of the U.S Department of Energy by the University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. Lawrence Livermore National Laboratory, P.O. Box 808, L-159, Livermore, Ca, 94551 This document is UCRL-TR-232039.

References

[1] Greg Abram and Lloyd A. Treinish. An extended data-flow architecture for data analysis and visualization. Research report RC 20001 (88338), IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, February 1995.

[2] Rajesh K. Batra and Lambertus Hesselink. Feature comparisons of 3-D vector fields using earth mover's dist ance. In David Ebert, Markus Gross, and Bernd Hamann, editors, *IEEE Visualization '99*, pages 105–114, San Francisco, 1999.

[3] Hank Childs, Eric Brugger, Kathleen Bonnell, Jeremy Meredith, Mark Miller, Brad Whitlock, and Nelson Max. A contract based system for large data visualization. In *Proceedings of IEEE Visualization 2005*, 2005.

[4] Hank Childs and Mark Miller. Beyond meat grinders: An analysis framework addressing the scale and complexity of large data sets (to appear). In *Proceedings of HPC2006*, 2006.

[5] Computational Engineering International, Inc. *EnSight User Manual*, May 2003.

[6] Willem C. de Leeuw, Hans-Georg Pagendarm, Frits H. Post, and Birgit Walter. Visual simulation of experimental oil-flow visualization by spot noise images from numerical flow simulation. In *Visualization in Scientific Computing '95*, pages 135–148, 1995.

[7] Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Local and global comparison of continuous functions. In *Proceedings of the IEEE conference on Visualization (VIS-04)*, pages 275–280, October 2004.

[8] C.R. Johnson, S. Parker, and D. Weinstein. Large-scale computational science applications using the SCIRun problem solving environment. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000.

[9] C. Charles Law, Amy Henderson, and James Ahrens. An application architecture for large data visualization: a case study. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 125–128. IEEE Press, 2001.

[10] L. Linsen, V. Pascucci, M. A. Duchaineau, B. Hamann, and K. I. Joy. Wavelet-based multiresolution with $\sqrt[2]{n}$ subdivision. *Computing*, 72(1):129–142, 2004.

[11] Alain Mazzolo and Benoit Roesslinger. Monte-carlo simulation of the chord length distribution function across convex bodies, non-convex bodies and random media. *Journal of Mathematical Physics*, 44(12):6195–6208, 2003.

[12] Valerio Pascucci and Randall J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, pages 2–2, New York, NY, USA, 2001. ACM Press.

[13] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 93–ff. IEEE Computer Society Press, 1996.

[14] Qin Shen, Alex Pang, and Sam Uselton. Data level comparison of wind tunnel and computational fluid dynamics data. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 415–418, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[15] C. Silva, Y. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. In *Visualization 2002 Course Notes*, 2002.

[16] Craig Upson, Thomas Faulhaber Jr., David Kamins, David H. Laidlaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam. The application visualization system: A computational environment for scientific visualization. *Computer Graphics and Applications*, 9(4):30–42, July 1989.

[17] Hualin Zhou, Min Chen, and Mike F. Webster. Comparative evaluation of visualization and experimental results using image comparison metrics. In *VIS '02: Proceedings of the conference on Visualization '02*, Washington, DC, USA, 2002. IEEE Computer Society.