Reviewer: 1

Comments to the Author

I am attaching a PDF with a ton of comments on it for the authors.

*Responses to comments in attached PDF below.*

Reviewer: 2

Comments to the Author

This paper is well written and an interesting read on the evolution of vtk-m and its impact on science and visualization. My only comments would be to better explain the visualization in figure 5 (what is this bar with what looks like a christmas tree on top?), please describe what we are seeing in figures 6 and 7 (fig 6 is super boring since its mostly black, and Im not sure how to compare 7 to 6). Otherwise the paper is ready for publication.

*Text has been added next to Figure 5 to explain the simulation these data come from and why it is significant.*

*Figures 6 and 7 were explained more fully in the Laser Wakefield Acceleration section. The figures have been moved to that section and arranged to be more easily compared. The captions explain that Figure 6 (now Figure 8) is an earlier stage of the simulation whereas Figure 7 (now Figure 9) is a later stage at a lower resolution. This is easier to discern with the captions closer together.*

Attached PDF comments:

Introduction:

implementing classic scientific visualization algorithms that have been redesigned for heavily threaded environments: Is VTKm really just providing a bucket of redesigned algorithms that map better to GPU accelerators? Or, is it providing a lower-level, device nuetral bucket of operations that enable the expression of visualization algorithms that map will to any kind of accelerator?

*VTK-m is doing both of these things. This is made clear in the following sentence that starts with “VTK-m also provides…”*

promote: ....to productize VTKm?

*We were using “promote” to mean “further the progress of”. To make this more clear, we have replaced this word with “advance”.*

VTK-m is currently: This suggest that each of these tools when run on GPU-enabled HPC systems do the majority of their work with VTKm. Is that really true?

*These tools use VTK-m to varying degrees. The sentence has been rephrased to avoid suggesting VTK-m provides the complete functionality.*

Overview of VTK-m:

practitioners almost: What do we mean by "practitioners" here? Visualization tool developers or consumers of visualization tools trying to do actual science. I think it is the later in which case it might make sense just to use the term "users".

*Changed.*

Is table 1 a statement about the STARTING point of VTKm or the ENDING point?

*It is the current state of VTK-m, but not meant to be an exhaustive list. Rephrased to make this clearer.*

many threads: I dunno if it helps but a naive reader may not realize that the basic model for an accelerator device VTKm is suited for is that it is designed to operate on MANY threads. That point may be obvious to most people working in HPC but is otherwise potentially lost without some emphasis.

*A sentence was added to describe that these devices expect a great many threads.*

The entirety of VTK-m can be ported with a change to the device adapter: I assume this has already actually been demonstrated with, for example, supporting both AMD and NVIDIA GPUs (or, is that not the right way to think about it?).

*Yes, this is what we meant, and it is demonstrated by the devices discussed in this paper and others outside the scope of the paper that we have not.*

with standard C++14 and can be compiled for any device that supports C++: I find this a tad confusing...any device that also supports the C++-14 standard, right? I mean a device that supports C++ but only C++-03 or C++-11 wouldn't cut it, right?

*This is correct, but this wording was confusing. The sentence has been rephrased to make it clear we are talking specifically about C++14.*

A second approach: I am not understanding what fundamentally is different about these "two approaches". I believe one is referring to decomposing an input data object into smaller pieces that are then divvied out, inter-node parallel compute resources and the other is referring to breaking the operations to be performed into their most fundamental primitives which map well to MANY, intra-node threads. This could use a tad more explanation I think.

*The use of “approach” was confusing here. It has been replaced with “technique.”*

To simplify the implementation further, VTK-m provides meta data parallel primitives (Moreland et al. 2021) that incorporate common patterns for scientific data that were demonstrated to be efficient.: I am not understanding what fundamentally is different about these "two approaches". I believe one is referring to decomposing an input data object into smaller pieces that are then divvied out, inter-node parallel compute resources and the other is referring to breaking the operations to be performed into their most fundamental primitives which map well to MANY, intra-node threads. This could use a tad more explanation I think.

*The phrase meta data parallel primitives was misread. It means meta versions of data parallel primitives, not parallel primitives of metadata. Some hyphenation was added to clarify that as well as an example of a meta data-parallel primitive.*

Porting Challenges:

Adopting Kokkos:

supports is “write once, compile anywhere”: FYI..."write once, compile everywhere" was not mentioned earlier. Perhaps it should be as I think it is a key goal VTKm is achieving which is different from merely reimplementing everything everywhere every time we encounter a different hardware architecture.

*The theme of portability is visited several times before here. We added this specific statement of writing once and running everywhere to make it easier to tie back to the theme.*

wrap a parallel device: Can you say anything about the high level conceptual model for such a device? I mean, in broad terms, what are some of the basic assumptions that must be true in order for this 'parallel device' concept to be compatible with VTKm's design?

*The remainder of the sentence already lists all the things that a device is expected to do through a device adapter. This has been made more clear by replacing “and provide” with “into a common API that provides”.*

scan, sort, and reduce: There is that trio again. Its kind of fundamental. I highly recommend elaborating this concept just once upon first mention.

*Done in the first use in the overview section.*

but many are not well optimized: Instead of "many are not well optimized", maybe "but genericity precludes sufficient optimization for target architectures.

*Changed as recommended.*

back end: are "back end" and "device adapters" the same thing?

*That is what we meant. Changed instances of back end to device adapter to avoid confusion.*

Is "feature configuration" some kind of C++ templating design principle? If not, I don't know what this means other than Kokkos is initialized once. And, "once" when, upon build, installation within a single run of an executable?

*This has been reworded to be made clearer.*

Addition and Then Removal of Virtual Methods:

problems: Ok, what WAS the problem...the gathered code was too bloated? Gathering the code took too long...what was the problem?

*We added a sentence describing the specific problems encountered.*

the implementation, the new design based the array management on known buffers: if this design is in any way informed by other technologies (e.g. conduit), I think that should be stated somewhere.

*“known buffers” does not mean the design was based on another known method. Rather, it means that the array management has access to the raw memory buffer. This has been rephrased to make it more clear.*

fallback for the cases when the type is unexpected: Out of curiosity, how does a developer or user know or interrogate VTKm to discover that it is "falling back" to something far less optimal and then what to do to avoid that?

*Generally, no, but the system might give output to the log if an inefficient fallback happens. This is a bit more detail than we would like to get into here.*

Filter Interface Overhaul:

required exposed definitions that must be recompiled with every use: What? I have no idea what it means to recompile something with every use.

*Rephrased to make it clear that the compiler had to recompile the code each time the function was used.*

A broader aspect of this issue that is not touched on really at all here but is highly relevant IMHO is that VTKm is intended to support data coming from ALL MANNER of data producer (e.g. simulation). It cannot (or should not) make assumptions about float vs. double vs. long double, etc. or it will otherwise have less reach or substandard performance. As such, it requires sophisticated software engineering to enable that in a way that is practical both for developers and users of VTKm. I think this is a really critical aspect of visualization tool development that is not appreciated by most of the community and I think it would be a very useful thing to mention early on in this article.

*Good point. This explanation is added to the beginning of Addition and Then Removal of Virtual Methods.*

limits the portion of code to be instantiated: Is the main (or only) driver for this reducing compile time? Or, is saving on GPU memory consumed by code also part of the equation. If one or the other or both, it might be worth including a sentence to that effect.

*Rational added to the end of the paragraph.*

Because the filter execution methods are no longer template methods, they can be virtual methods.: This statement has meaning for me only if "filter execution methods" are synonymous with executing on the GPU. But, it would be clearer just to say that too.

*The word “execution” is overloaded in a confusing way. Rephrased to make this more clear.*

compiled into a .so file: Why is that important? I am not asking because I don't know. I am asking because I don't think readers will always connect the dots you are assuming they will.

*A description of the benefits was added.*

GPU-to-GPU Transfers

encapsulated in a newly introduced “blob” data type: It seems like here you are accepting the need for a "catch-all" data type. So, why isn't that ok in the Filter interfaces described in the previous section?

*Added an explanation that DIY’s blob is essentially the same concept as VTK-m’s raw buffer.*

The corresponding VTK-m changes: This is really down in the weeds. What is the upshot of all of this?

*Removed some extraneous information to make the point clearer.*

Operation on Exascale Hardware:

Frontier:

I think this article would do better if you introduced the main architectures you have run VTKm on successfully and their key architectural similarities and differences. A lot of the details of the full system are less relevant than those details that impact VTKm design or show of VTKm flexibility.

*Descriptions of exascale scaling studies have been added to the abstract and introduction.*

The total time to render this dataset was 300 ms: So, this statement does say "render" but I wonder if this 300 ms includes the time to read it into memory, the time to extract the iso-contour and/or just the time to draw the image once all the facets were computed.

*This has been clarified to be just the rendering time.*

Integration into Visualization Tools:

ParaView:

Highly parallel hardware: What does "highly parallel hardware" mean? Do you mean MANY THREADS?

*This has been rephrased.*

Interfacing with Applications:

The article is about VTKm and here we talk about integrating VTKm directly into some applications. But, doesn't Ascent do that too? And, Ascent can turn around and employ VTKm for its work. I think a high level 3-5 sentences on the target usage scenarios for this may be worth adding for clarity. I am not suggesting a bake-off comparison between them but I was left wondering wny do I need Ascent if I can do in-situ with VTKm and vise versa. I think the two capture different usage scenarios and it is probably worth ensuring the reader understands those.

*Added a sentence to make it clear that VTK-m is often used in conjunction with other visualization tools to be applied to science applications. Also changed the order of the subsections to make this point easier to follow.*

Tokamak Fusion Reactor:

So, this section is a case of integrating VTKm directly into an application and not via something like Ascent, right? Maybe make that point clearer?

*Added some text to spell this out.*

Figure 9: I think you want to add somewhere in this caption that this was computed using VTKm. I mean, maybe I should have just assumed that but it wasn't clear until I got to the end of this section.

*Added*

Figure 9 shows some examples of Poincaré plots: So, these are just examples of what this kind of plot looks like and have nothing to do with VTKm to compute them? Or, did VTKm compute these pictures?

*Added clarification that these were created using VTK-m.*

A lot of white space here. Missing picture maybe?

*This is just an artifact of LaTeX making the top and bottom of the two columns even.*

As a simulation step completes, EFFIS launches a Poincaré analysis task on GPUs in the node in a round-robin fashion: How is the application data communicated to the WDMApp on the additional nodes? I/O, MPI, DIY, something else?

*These details are handled by EFFIS and are well beyond the scope of this paper. See the Suchyta et al. 2022b reference for details.*

Laser Wakefield Acceleration:

So, here, you do use the phrase "in-situ" and did not in the tokomak case. Any reason for this?

*For practical reasons, these in-situ implementations can also be run post hoc on saved data. We are often vague because the work could happen either place, and often in the final production picture we work with captured snapshots. In this particular highlighted instance, these pictures come from images that were specifically captured in-situ from a running simulation.*

Conclusion:

anomalous? Really? I would say maybe "novel" for compute anyways. They were quite common especially in visualization apps and hardware just not for compute.

*Changed*