Eurographics Symposium on Parallel Graphics and Visualization (2019)
H. Childs, S. Frey (Editors)

Efficient Point Merging Using Data Parallel Techniques

Abhishek Yenpure[1], Hank Childs[1], and Kenneth Moreland[2]

[1]University of Oregon, USA
[2]Sandia National Laboratories, USA

Abstract

We study the problem of merging three-dimensional points that are nearby or coincident. We introduce a fast, efficient approach that uses data parallel techniques for execution in various shared-memory environments. Our technique incorporates a heuristic for efficiently clustering spatially close points together, which is one reason our method performs well against other methods. We then compare our approach against methods of a widely-used scientific visualization library accompanied by a performance study that shows our approach works well with different kinds of parallel hardware (many-core CPUs and NVIDIA GPUs) and data sets of various sizes.

CCS Concepts

• ***Computing methodologies*** → ***Shared memory algorithms; Scientific visualization;*** *Computer graphics;*

1. Introduction

With this work, we contribute a many-core approach for the point merge algorithm i.e. merging nearby points for scientific visualization. The primary challenge for this algorithm is efficiently identifying which points are close to each other. Although the point merge algorithm does not receive as much attention as algorithms like iso-surfacing or volume rendering, it is used regularly in scientific visualization tools.

Point merging typically complements visualization algorithms that iterate over cells like iso-surfacing or slicing. These algorithms iterate over input cells to generate triangles; the vertices of these output triangles are interpolated and not part of the source data set. Consider a case where an iso-surface operation is applied on two neighboring cells, C_1 and C_2, to produce two abutting triangles T_1 and T_2, as shown in Figure 1. T_1 comprises vertices V_{11}, V_{12}, and V_{13}, and T_2 is composed of vertices V_{21}, V_{22}, and V_{23}. As abutting triangles, two sets of their vertices should be coincident. Without loss of generality, assume V_{11} and V_{21} are coincident and V_{13} and V_{22} are coincident. If these pairs are not merged, then connectivity-based operations will fail. In particular, rendering this triangle data will result in flat shading, since the normal of V_{11} would reflect only T_1 (and not T_2), the normal of V_{21} would reflect only T_2 (and not T_1), and V_{13} and V_{22} would suffer similarly. However, if V_{11} and V_{21} are merged to make a new point V' and if V_{13} and V_{22} are merged to make a new point V'', then the lighting will appear smooth.

Point merging is useful in other settings as well. When visualization algorithms generate triangles with small areas, then lighting issues can again arise (among other issues). The vertices of triangles with small areas are typically very close to one another and thus can be reduced to a single vertex through point merging. Further, numerical errors can sometimes cause interpolated vertex positions to be slightly offset. So in the previous iso-surface example, even vertices that should be considered exactly coincident might have inexact point coordinates. Point merging solves this problem as well.

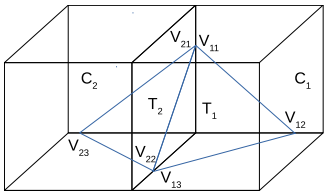


Figure 1: *Example of abutting triangles generated by applying an iso-surface operation.*

Point merging is typically done by organizing points into a spatial data structure and traversing that data structure to locate nearby points. Our algorithm works in this vein, although we are able to arrange our operations so that no explicit data structure is needed. It is for this reason that we refer to our approach as "Virtual Grid" point merging. A particular focus for our algorithm is on many-core architectures. Our code is designed to use the parallel building blocks available from the VTK-m library that ensure good performance over varying architectures. We evaluate this code, comparing to another module in VTK-m, and to both parallel and serial modules in VTK. Overall, we find that our Virtual Grid point merging algorithm is competitive with other parallel point merging techniques, more resilient to irregular distributions of input points, scalable within shared memory domain, and performs well on both CPU and GPU devices.

DOI: 10.2312/pgv.20191112

2. Related Work

We divide this section into two parts. In the first part we discuss previous works that deal with merging of points. In the second part we discuss previous works for developing scientific visualization algorithms using data-parallel techniques.

2.1. Merging Points

Many previous works present approaches for merging points for the related application of mesh simplification. This section is further divided into three parts. The first part discusses works that use tree-based search structures to discover spatially close points. The second part discusses works that use spatial binning data structures to merge points. The third part discusses works that focus on improving the accuracy of merging points.

2.1.1. Merging Points Using Search Structures

Rock and Wozny [RW92] provided the one of the first approaches for reconstructing the topology of a model. The first step in their approach involves merging spatially close points, which they term "vertex merging," as the points that they merge are vertices of triangular facets. To locate close points that need to be merged, they used an AVL search tree constructed with the vertices of the facets. Kanaya et al. [KTKN05] presented a related approach where they perform vertex clustering using an octree and offer multiple degrees of mesh simplification. They used a depth-first approach on this octree to locate the connected components of the desired degree, and simplify them using vertex merging.

2.1.2. Merging Points Using Spatial Binning

The construction of search structures for large data is known to have a significant computational overhead. Rossignac and Borrel [RB93] addressed this issue with a mesh simplification approach that uniformly subdivides a 3D volume containing an input mesh into smaller 3D regions, or "bins." All vertices that occur within the same region are merged together. The input mesh is then corrected to remove all degeneracies that were introduced by the merge operation, yielding a simplified output mesh. While this approach is fast, it has very little control over the accuracy and quality of the simplification. Shin et al. [SPC*04] modified the technique used by Rossignac and Borrel to improve the accuracy of vertex merging. They aimed to only merge vertices that exist within a user-defined tolerance. Vertices within this tolerance may occur in spatially adjacent bins. To limit searching for neighbors in all adjacent bins, Shin et al. described a way to decompose a cell in multiple regions. This modification significantly reduces the number of bins that must be searched sequentially. The primary focus of this was the construction of topology from triangle data, given no prior connectivity information.

2.1.3. Improving Accuracy of Point Merging

This set of works focus on minimizing the error introduced by merging vertices. Barequet and Kumar [BK97] merged pairs of vertices that exist on different edges. In this method, edges are selected based on the cost of moving the endpoint vertex of one edge to the endpoint vertex of another edge. This cost has a user-provided upper bound, and a pair of vertices are merged by averaging their coordinates. Garland and Heckbert [GH97] choose vertex pairs to merge by considering the effect on the mesh. A pair of vertices are merged based on a per-vertex error function that calculates the sum of squared distances to the planes of the triangles that meet at the vertex. To merge a pair (v_1, v_2), the position v_{new} is calculated as the minimum error point. Low and Tan [LT97] described a way to produce a more consistent mesh simplification via a cell clustering approach that assigns weights to vertices based on the probability of the vertex lying on the mesh boundary, and on the size of the faces bounded by the vertex. Lindstrom and Turk [LT98] minimized the impact of point merging by assigning a cost to the mesh edges, where the cost is a function of the volume, boundary, and shape preservation properties of the mesh vertices. Finally, Lindstrom [Lin00] proposed a hybrid scheme where they used the vertex clustering algorithm proposed Rossignac and Borrel [RB93] and improved it by using error minimization techniques proposed by Garland and Heckbert [GH97], and Lindstrom and Turk [LT98]

2.2. Data-Parallel Techniques

Recently, a growing body of literature has investigated the design of data-parallel algorithms for scientific visualization applications using data-parallel primitive (DPP) [Ble90] operations, such as sort, gather, scatter, map, reduce, copy, etc. The DPP approach inspires our own research of techniques for performing point merging in a data-parallel setting.

Lessley et al. [LMLC17] described approaches for data-parallel searching for duplicate elements in a 3D mesh topology. Point merging has similar elements, in that coincident points need to be searched and eliminated. That said the work by Lessley et al. is not applicable to the problem of merging points, because their algorithm can only be used to compare indices and find unique elements. The point merging problem, however, considers spatial locations (not indices) and must be capable of identifying nearby points (not only duplicates). Regardless, their results motivate the use of data-parallel techniques for search problems.

Miller et al. [MMM14] presented a data-parallel method that generates the output topology of a visualization operator using the knowledge of the input mesh topology. One of the visualization operators they studied was the Marching Cubes algorithm, where all output vertices occur on the edges of the cells of the input voxel grid. Abutting triangles from the iso-surface will contain redundant (coincident) points, and the authors located and removed these points via a Reduce-By-Key DPP. However, this approach cannot be directly applied to a mesh when the information about the original topology is unavailable.

Finally, the emergence of platform-portable libraries, such as NVIDIA Thrust [NVI18] and VTK-m [MSU*16], have made it convenient to write algorithms in a single code implementation for execution across multiple platforms (e.g., both CPUs and GPUs). These high-level libraries provide a set of core DPPs that are optimized for each target platform of execution using low-level, platform-specific libraries, such as OpenMP, Intel TBB, and NVIDIA CUDA. Algorithms can then be written in terms of these

core DPPs or user-defined DPPs ("worklets" in the case of VTK-m). This has been demonstrated in previous works in the context of scientific visualization operations like iso-surface and cut surface extraction [LSA12], threshold [LSA12, MMA*13], and contour-tree construction [CWSA16]. Specifically in the case of VTK-m, a number of research works have demonstrated competitive performance to algorithms designed for a specific platform for rendering [LLN*15, LMNC15, SM15] and other scientific visualization operations [LBMC16, LLCC17, PYK*18, LPM*17]. This collection of work justifies the merit of the platform-portable framework. In this work, we design our algorithm with VTK-m and contribute the implementation as a standard, open-source filter within the library.

3. Formal Definition of Point Merging

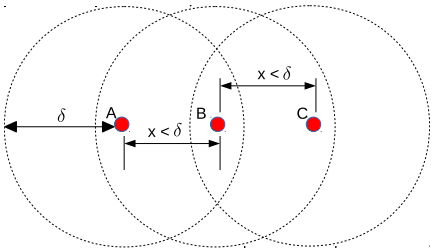


Figure 2: *Points to be merged, where* $x < \delta$

Informally, the point merging operation finds all points within a distance δ and merges those points together. However, it is necessary to define correctness of merging points in ambiguous cases when we have tolerance $\delta \neq 0$. Figure 2 presents a scenario where we have 3 points to be merged together, points A, B, and C. Points A and B satisfy the distance criteria, and points B and C satisfy the distance criteria with respect to tolerance δ, but A and C do not satisfy the distance criteria. This is an ambiguous case because it is unclear whether A and C should be merged together to satisfy all of B's distance criteria. In this work we consider any group of points that satisfy the following properties to be a proper point merge:

1. In a group of points to be merged, every point in the group is within distance δ from at least one point in the group; and
2. After the merging is complete, none of the resulting points are within distance δ from one another.

Note that our definition of point merging allows for multiple, equally correct solutions. For example, Figures 3(a), 3(b), 3(c) present all the possible, correct solutions to merge the points from Figure 2. Since there are multiple correct solutions to merge points in an ambiguous case, the outputs of different approaches can be slightly different from each other, i.e., they might not contain the same points in the final output. However, they must satisfy the properties discussed earlier.

4. Technical Approach

This section provides details on our Virtual Grid approach. The key benefit of our approach compared to previous approaches is that our approach is designed for many-core architectures and is composed of data parallel techniques. At a high level, our approach works as follows. It begins by binning points into the cells of a 3D uniform grid, which is the same first step taken by some previous approaches. However, these previous approaches have relied on an explicit mapping from cells to lists of points within the cells. Our approach, instead, focuses on using data parallel operations to rearrange a large list of points. After rearranging, each thread of a many-core device operates on one cell at a time, and the points that lie within that cell are readily available in the thread's memory. This is the reason we term our approach as a "Virtual Grid" approach – while we conceptually use a grid (i.e., 3D bins) to guide our process, our DPP-based sequence of operations obviates the need to explicitly represent this grid. Figure 4 provides a 2D schematic of how this process occurs in our Virtual Grid approach.

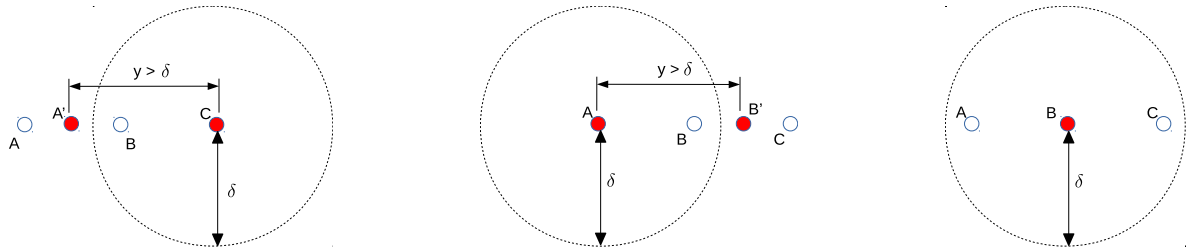
Another difference between our algorithm and previous approaches is in the merge step. Previous approaches have merged all points that occur in the same cell or have employed methods to look for close points in adjacent cells. It is useful to merge all points binned together in cases like mesh decimation, where accuracy is not of the utmost importance. We support this as a special case of merging points as this helps us to avoid expensive computations that are necessary to obtain high accuracy. In cases where higher accuracy is desired, our approach works differently; we perform distance computations between pairs of points within a cell to ensure that only the points that occur within the tolerance δ are merged. However, this operation by itself is not sufficient to merge all points.

Occasionally, close points (i.e., points that are within δ and should be merged) are mapped into adjacent cells of the Virtual Grid. It is important to identify when this happens and ensure they are merged. We deal with this case by performing multiple iterations of our point merge operation. Each iteration uses its own Virtual Grid, with the Virtual Grids from each iteration being offset from others by a small amount. This way, the points that satisfy the tolerance criteria, but were binned into different cells in one iteration, can be binned into the same cell in a later iteration. Specifically, we run eight iterations. We calculate the eight bounding boxes of the eight Virtual Grids as:

1. Expanding the bounding box of the original data set by δ in all directions (1 *count*);
2. Shifting the bounding box from step 1 by δ along each axis (3 *counts*);
3. Shifting the bounding box from step 1 by δ diagonally along planes XY, YZ, and XZ (3 *counts*); and
4. Shifting the bounding box from step 1 by δ along the diagonal of the bounding box (1 *count*).

Our algorithm also treats a tolerance of 0 and mesh decimation as special cases where only a single iteration is required. This is because, in the case of merging exactly coincident points (tolerance of 0), the issue of close points in adjacent cells is not relevant, and in the case of mesh decimation only an approximate representation is desired.

A single iteration of our approach works as follows:



(a) Merge points A and B. $y > \delta$ (b) Merge points B and C. $y > \delta$ (c) Merge points A, B, and C

Figure 3: *Possible, correct solutions to merging points*

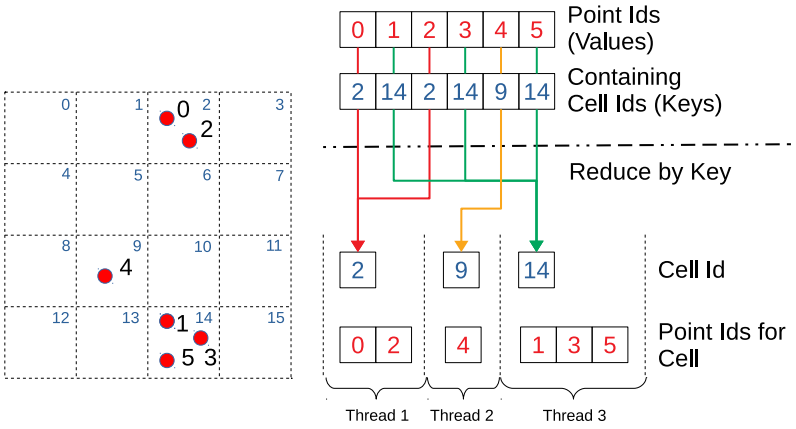


Figure 4: *Notional example of the Virtual Grid approach in action.*

1. For each point, identify which cell in the Virtual Grid contains it. This is done via a "Map" data parallel operation.
2. Rearrange the layout of the points so that points contained in the same cell are grouped together in an array.
3. For each group, calculate the set of points that are within the tolerance δ of each other. There may be multiple such sets. For a set of points that satisfy the tolerance criteria, output the point with the minimum index as the neighborhood identifier. This is done via a "Reduce-By-Key" data parallel operation, which is provided by VTK-m.
4. Rearrange the layout of the points so that points contained in the same neighborhood are grouped together in an array.
5. Reduce each neighborhood of points by calculating its centroid. This is done via a "Reduce-By-Key" data parallel operation, which is provided by VTK-m.

Miller et al. [MMM14] show how to perform Steps 2 and 4 efficiently in parallel by sorting, and VTK-m provides this grouping as a basic feature [Mor18]. The steps described above are performed once for each of the Virtual Grids described earlier. Algorithm 1 shows these steps in pseudocode.

This approach performs at its best when the cells of the Virtual Grid have small numbers of points binned into them. Binning fewer points in the cells of the Virtual Grid translates to performing fewer

Algorithm 1 Virtual Grid approach

```
    // Step 1
1   for all i ∈ [0.. | Points |], do in parallel
2       Cell[i] = GET-CELL-FOR-POINT(Points[i])
    // Step 2 : Performed internally by VTK-m
    // using Cell array from Step 1.
3   Bins = {(bin, P_bin) :
                P_bin are indices of all points in bin}
    // Step 3
4   for all (bin, P_bin) ∈ Bins, do in parallel
5       for i ∈ P_bin
6           nearest[i] = MIN(i, GETNEAREST(i, P_bin))
    // Step 4: Performed internally by VTK-m
    // using Nearest array from Step 3.
7   Clusters = {(cluster, P_cls) :
                P_cls are indices of all points in cluster}
    // Step 5
8   for all (cluster, P_cls) ∈ Clusters, do in parallel
9       centroid = GET-CENTROID(Points, P_cls)
    // Centroids collected from Step 5
    // become the set of merged points.
```

computations to correctly group points that satisfy the tolerance δ. Also, since the Virtual Grid is represented sparsely in memory, i.e., no state information for the grid is stored, it can use much smaller bins and therefore reduce the cost of searching for neighbors in step 3 of Algorithm 1. To achieve this, we calculate the optimal dimensions for the uniform grid based on the tolerance δ using the following equation:

$$dimension_t = \frac{length_t}{2 \times \delta} \quad (1)$$

where $dimension_t$ is the dimension of the grid along axis t and $length_t$ is the length of the bounding box along axis t in the original data set. This enables us to create the smallest bins for which all pairs of points within the tolerance δ of each other are guaranteed to bin in the same bin or in adjacent bins. However, if the tolerance δ is very small, then the dimensions of the Virtual Grid can become so large that it requires more than 2^{32} or 2^{64} bins, which means they cannot be indexed by 32-bit or 64-bit numbers. In such cases, we limit the dimensions of the grid using the following equation:

$$dimension_t = floor(\sqrt[3]{|MaxValue|}) \quad (2)$$

Where $MaxValue$ is the maximum number of bins that can be indexed using the data type we choose for binning points. We also use Equation 2 to calculate dimensions of the grid when δ is zero.

Section 6 presents the impacts of the data type choice on performance.

5. Experimental Overview

To better compare and study the performance characteristics of our method, we performed tests with the following variables:

- 6 algorithms;
- 3 data sets;
- 4 values of tolerance δ, 2 absolute and 2 relative to the bounding box; and
- 2 different hardware architectures.

We ran performance tests with select combinations of these variables. Table 1 details these combinations. Note that we tested each of these combinations with all three of our data sets. In total we compared 171 unique combinations of these variables. Limitations of the features from our comparators restricted us from testing all possible combinations. We elicit these limitations further in this section.

5.1. Algorithms

We consider a total of six algorithms: two algorithms are from the VTK library, and four algorithms are implemented using VTK-m.

5.1.1. VTK Algorithms

The two algorithms from VTK come from the "vtkSMPMergePoints", and "vtkCleanPolyData" classes. Both of these classes have limitations. The "vtkSMPMergePoints" point locator merges coincident points in parallel,[†] but requires the user to explicitly take care of threading. The "vtkCleanPolyData" filter is capable of merging points within a user provided tolerance. This filter also performs additional tasks, such as removing degenerate triangles. However, the vtkCleanPolyData filter operates only in serial.

5.1.2. VTK-m Algorithms

From the four algorithms implemented in VTK-m, two algorithms come from variants of our Virtual Grid approach, one algorithm comes from a point locator based approach for merging points, and one other comes from the "VertexClustering" filter from the VTK-m library [MSU*16, Mor18].

The variants in the Virtual Grid approach result from the user's choice in data type to store the bin indices while binning points. For this paper, we chose to test with two different data types: 32-bit integers and 64-bit integers.

The point locator based approach was implemented as a VTK-m counterpart to the point locator based module in VTK, namely vtkSMPMergePoints. The solution for this study was implemented using a modification of the "PointLocatorUniformGrid" module from the VTK-m library, which uses spatial binning to locate the nearest neighbor of a point. The modification was performed to support queries for nearest neighbors for the same points that were used to build the search structure. Without this modification, for a given query point, the points locator returns the same query point as the nearest neighbor. Other point locator modules based on a k-D tree and on two-level uniform grid are also available in the VTK-m library but were not used for this study. The algorithm that we developed uses multiple iterations to merge points. For each iteration it finds the nearest neighbor for every point, and merges the neighbors by reducing them to their centroid. These iterations are performed until no new neighbors are discovered in the set of the residual points.

The "VertexClustering" filter from VTK-m is intended for the use of mesh decimation and we use it as a parallel comparator for the Virtual Grid approach for the application of mesh decimation.

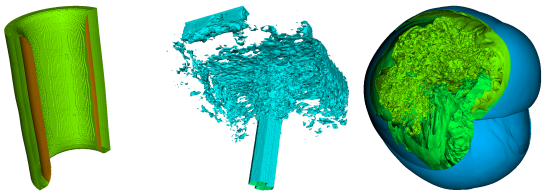
5.2. Data Sets

We explored three data sets for this study. In each case, we took an existing data set and applied an iso-surface operation. The fusion data set comes from the NIMROD [SGG*04] simulation code, which is used to model the behavior of burning plasma. The thermal hydraulics data set comes from the NEK5000 [FLPS08] code, which is used for the simulation of computational fluid dynamics. Finally, the supernova data set comes from a supernova simulation made available by Blondin and Mezzacappa [BMD03]. We used the VTK-m library to perform the iso-surfacing and disabled the option for merging coincident points. This resulted in a triangle

[†] We encountered a bug in this module, and so the experiments performed in our study came from a revised version provided by Kitware. The bug fix will be available in a future version of VTK. The bug report can be found at the link: https://gitlab.kitware.com/vtk/vtk/issues/17386.

	Tolerance				Parallel			
	Point Merge		Decimation		CPU			
Algorithm	$\delta = 0$	$\delta \neq 0$	$\delta = 1\%$	$\delta = 10\%$	TBB	Scaling	GPU	Count
vtkSMPMergePoints	✓				✓	✓		18
vtkCleanPolyData		✓			✓			3
VTK-m Vertex Clustering			✓	✓	✓		✓	12
VTK-m Point Locator	✓	✓			✓	✓	✓	42
VTK-m Virtual Grid (32 bit)	✓	✓	✓	✓	✓	✓	✓	54
VTK-m Virtual Grid (64 bit)	✓	✓			✓	✓	✓	42
Total								171

Table 1: *Combinations that we tested. As an example of how to interpret this table, consider the row for "VTK-m Virtual Grid (32-bit)," which we used for both point merge and mesh decimation experiments, with all types of parallelism. For different hardware architectures we ran 1 experiment for both CPU and GPU execution, and performed 5 scaling experiments for TBB with varying number of CPU threads, making the total 7. For point merging, we have 2 tolerances $\times$ 7 parallelizations $\times$ 3 data sets $=$ 42 experiments. For mesh decimation, we have 2 tolerances $\times$ 2 parallelizations $\times$ 3 data sets $=$ 12 experiments. In total, we have 54 total experiments for this case.*



Fusion Thermal Hydraulics Supernova

Figure 5: *Data sets used for the performance tests. An additional clip operation was applied to generate the images for the Fusion and Supernova data to reveal more intricate details.*

soup, where no triangles shared any common vertices. Explicitly, if there were N triangles, then the soup would have $3 \times N$ vertices, with many of the vertices being coincident and replicated in the vertex list. Figure 5 provides the visuals for the test data sets, and Table 2 provides additional information for reproducibility purposes.

		Output		
Data set	Iso-Values	Points	Cells	Bounds
Fusion	2.4, 3.2	4125540	1375180	0 – 1
Hydraulics	42, 64	15686430	5228810	0 – 1
Supernova	0.02, 0.05, 0.07	24493224	8164408	0 – 431

Table 2: *Details of the data sets we used for our experiments.*

5.3. Tolerance for Merge

We test our algorithms with four values for tolerance, two as absolute values to study point merging and two as fractions of the data set extents used to study mesh decimation.

We used zero as the tolerance to merge exactly coincident points present in the data set. Choosing the tolerance as $\delta = 0$ enabled us to compare our parallel algorithms implemented using VTK-m with the parallel features to merge coincident points from the VTK library. We used a non-zero tolerance $\delta = 0.0001$ to merge points that occur within the specified distance of each other. This value was arbitrarily chosen, but yielded sufficient reduction for the test data sets. This enabled us to test cases where it is needed to merge points that are not strictly coincident but are separated by some insignificant distance between them.

The fractional tolerances were chosen to make it easier to study mesh decimation with the Virtual Grid approach. We chose to have bins that were sized 1% and 10% of the bounding volumes of the data set. These values represent 100^3 and 10^3 spatial bins for mesh decimation respectively.

5.4. Hardware Architectures

The biggest advantage of using the VTK-m library is its ability to provide portable performance over multiple architectures that are comparable to platform specific solutions. We tested our implementations on two different hardware:

CPU: Dual IBM Power9 CPU, each with 22 cores running at 3.8 GHz, capable of running 4 threads per core, and equipped with 512 GBytes of DDR4 memory.

GPU: NVIDIA Tesla V100 GPU of the Volta family with 5120 CUDA cores, 6.1 TeraFLOPS of double precision performance, and equipped with 16 GBytes of HBM2 memory.

VTK-m uses the TBB (Thread Building Blocks) library as a backend threading library for execution on many-core CPUs, and uses CUDA as a backend threading library for execution on NVIDIA GPUs. Henceforth, all parallel CPU execution times are reported using TBB, and GPU execution times are reported using CUDA.

6. Results

We present our evaluation of the Virtual Grid approach in four parts. The first two parts each contain an application: merging of exactly coincident points (Section 6.1) and mesh decimation (Section 6.2). In both parts, we compare performance with the applicable algorithms listed in Sections 5.1.1 and 5.1.2. In Section 6.3

we evaluate the strong scaling characteristics on multi-core CPUs. In Section 6.4 we present the performance portability when executing on multi-core CPUs and NVIDIA GPUs. In Section 6.3 and Section 6.4, we quantify performance and scaling in terms of the rate of processing points, which is calculated using the following equation:

$$processing\ rate = \frac{number\ of\ points}{execution\ time\ (seconds)} \quad (3)$$

This gives us the total number of points that a program is able to process in a second. We use processing rate as a measure because it is a better indicator of parallel speed-up than execution time [MO15].

6.1. Merging Points

This section is divided into two parts. In the first part, we study the performance of the Virtual Grid approach for merging exactly coincident points. In the second part, we study the performance of the Virtual Grid approach for merging points with a non-zero tolerance.

Our Virtual Grid approach is designed to merge points within a specified distance, δ, of each other. Merging exactly coincident points is a special case of this problem, i.e., the case where δ is zero. For our study, we compare against a VTK module that is designed specifically for merging coincident points; this comparator is somewhat imperfect, since the coincident point problem can be solved with fewer calculations then when δ is non-zero. In particular, our VTK-m based approach performs additional distance computations to make sure we group points only within the given tolerance δ. Also, we calculate the centroid for all the points grouped together to merge them, coincident or otherwise. Despite this asymmetry, we feel the comparison is valuable since it lets us test against another parallel implementation.

6.1.1. Merging Coincident Points ($\delta = 0$)

			VTK-m		
			Point	Virtual Grid	
Data Set	Device	VTK	Locator	32-bit	64-bit
Fusion	TBB	0.16	0.38	0.11	0.11
	CUDA		0.07	0.04	0.04
Thermal Hydraulics	TBB	1.01	4.88	0.39	0.41
	CUDA		0.31	0.12	0.14
Supernova	TBB	1.14	3.97	0.61	0.67
	CUDA		0.35	0.19	0.21

Table 3: *Execution times in seconds for merging points with $\delta = 0$. The execution times for TBB are presented using 40 CPU cores.*

Table 3 presents the execution times for this part of the study. In all cases the CPU execution times for the Virtual Grid approach are better than the comparator from the VTK library. The point locator based approach, implemented in VTK-m, performs slower than the other approaches in all cases. The reason for this behavior, apart from the extra computations, is the cost associated with the construction of the point locator data structure for every iteration until the merge converges.

Table 3 also shows the difference between the two versions of our Virtual Grid approach. The version using 32-bit integers consistently performs better than the one using 64-bit integers. Intuitively, the 32-bit integers results in larger bins, which results in more points binned together, which in turn should result in more time spent computing distances between points. However, 32-bit integers can be sorted faster than 64-bit integers, and the reduced sort time more than compensates for the extra distance comparisons, which are embarrassingly parallel.

When considering performance across data sets, the algorithms behaved differently. The execution times for the Virtual Grid are consistently proportional to the number of points being processed. For example, the Virtual Grid algorithm using 32-bit integers running with TBB computes at a rate of about 40 million points per second for all three data sets. In contrast, the algorithms based on point locators, which includes the VTK algorithm, process points at a much slower rate for thermal hydraulics data than the other two data sets. This is because the thermal hydraulics data is not spatially distributed as evenly as the other two data sets. We observe, therefore, that the Virtual Gird approach is more resilient to the spatial distribution of the points in the data and a better choice for unstructured data at different scales.

6.1.2. Merging Close Points ($\delta = 0.0001$)

The execution times for our comparator from the VTK library come from the "vtkCleanPolyData" module. This is the only module available in the VTK library that readily supports merging of points that are separated by some distance. As mentioned in Section 5, it does not support parallel execution on any architecture. In addition to merging close points, the vtkCleanPolyData module removes degenerate triangles; we modified our algorithms implemented using VTK-m to also remove these triangles to have a fair comparison.

			VTK-m		
			Point	Virtual Grid	
Data Set	Device	VTK	Locator	32-bit	64-bit
Fusion	Serial	2.68	7.17	2.43	2.49
	TBB		0.47	0.22	0.23
	CUDA		0.09	0.16	0.17
Thermal Hydraulics	Serial	76.79	121.43	10.85	11.19
	TBB		5.99	0.75	0.81
	CUDA		0.58	0.56	0.50
Supernova	Serial	64.13	87.96	15.21	15.96
	TBB		4.67	0.84	1.02
	CUDA		0.42	0.40	0.55

Table 4: *Execution times in seconds for merging points with $\delta =$ 0.0001. The execution times for TBB are presented using 40 CPU cores.*

Table 4 presents the execution times for this part of the study. There is a significant increase in the execution times for both the VTK library and the VTK-m implementations compared to the

times presented in Section 6.1.1. The increase in execution time for the VTK library is a result of now having to perform distance checks and calculating the centroid for merging points that are within the distance δ. The modules for merging coincident points from the VTK library only perform a comparison for equality. The increase in the execution times for the VTK-m implementations is a result of having to perform multiple iterations to merge points that are within the distance δ.

The Virtual Grid approach in VTK-m is the most performant of all the algorithms that we tested for this case. As we bin points into the smallest possible cells of the Virtual Grid, we have to perform only a few distance checks between pairs of points that belong to the same cell in order to merge them correctly. This reduces the computations that are performed by each thread when executing in parallel. Also, in contrast to the point locator based algorithm implemented in VTK-m, the Virtual Grid approach does not have the overhead of maintaining and updating a search structure to locate close points.

6.2. Mesh Decimation

		$\delta = 1\%$		$\delta = 10\%$	
		Vertex	Virtual	Vertex	Virtual
Data Set	Device	Clust.	Grid	Clust.	Grid
Fusion	TBB	0.07	0.09	0.06	0.15
	CUDA	0.16	0.04	0.16	0.06
Thermal Hydraulics	TBB	0.22	0.32	0.23	0.51
	CUDA	0.24	0.11	0.21	0.43
Suprenova	TBB	0.36	0.48	0.35	0.76
	CUDA	0.27	0.17	0.26	0.33

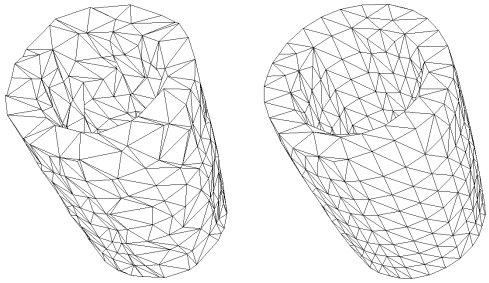
Table 5: *Execution times in seconds for performing mesh decimation. The execution times for TBB are presented using 40 CPU cores.*

Table 5 provides a comparison of our Virtual Grid approach against the Vertex Clustering filter from the VTK-m library. The numbers from the table suggest that the Vertex Clustering filter outperforms the Virtual Grid approach in most cases. This is an expected result as after clustering the vertices of the data, the Vertex Clustering filter chooses a random point from the clustered vertices as a representative point of the cluster. As mentioned in section 4, in the Virtual Grid approach we calculate the centroid of the clustered points as a representative point. The Virtual Grid approach incurs a significant performance cost for the centroid computations.

However, the extra cost helps the Virtual Grid approach produce smoother output meshes. Figure 6 shows the difference in the output of the Vertex Clustering filter and the Virtual Grid approach for the fusion data.

6.3. CPU Scaling Study

In addition to comparing our Virtual Grid approach to the reference VTK counterparts and the point locator based VTK-m algorithm, we also studied the strong scaling characteristic of the algorithms



(a) Vertex Clustering (b) Virtual Grid

Figure 6: *Decimation outputs using VTK-m Vertex-Clustering and the Virtual Grid approach for the Fusion data. The output data was obtained using 10^3 subdivisions for the Vertex Clustering filter and $\delta = 10\%$ for the Virtual Grid method.*

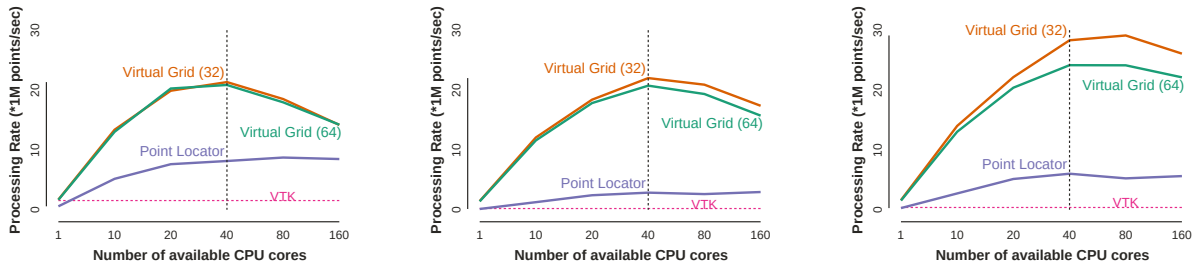
for merging close points. Figure 7 plots the results for this part of the study. Since there is no parallel module in VTK for this task, the plots show a flat line for the corresponding VTK module. The plots present the comparison of strong scaling results of our implementations when executing with varying number of CPU threads.

In general, the plots indicate that our VTK-m implementations are able to benefit from an increase in the number of available CPU cores. That said, we observe that although this scaling is good, it does level off as we approach the number of cores available. We believe this is because the point merging algorithm, which by its nature must load points from disparate memory locations, is memory bandwidth bound. Consequently, we observe diminishing returns for more cores and almost no benefit from engaging hyperthreading. These observations suggest that increasing the number of threads for execution beyond the available physical CPU cores does not result in any performance improvements for our implementations. This can be used as a heuristic to schedule programs to yield the best performance.

6.4. Performance Across Devices

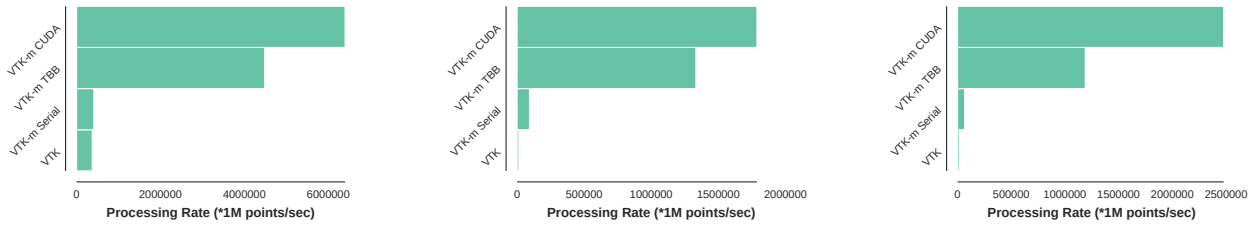
Figure 8 presents the comparison of processing rates for the Virtual Grid approach across devices. The processing rate for the VTK library comes from the "vtkCleanPolyGrid" filter and is used as a comparator for the Virtual Grid approach when executing in serial. This section reinforces our findings from Section 6.1.2 that the Virtual Gird approach is very efficient for a general case of merging points. When executing in serial the Virtual Grid approach offers better throughput compared to the VTK comparator. Additionally, the Virtual Grid approach is able to use available parallelism efficiently. When executing on GPUs the Virtual Grid approach is able to achieve a much higher processing rate.

The observed performance portability of our approach between GPU and CPU is just short of the ideal. The Power9 CPUs are capable of a throughput of 2.1 TFLOPS, and the Nvidia V100 GPUs are capable of a throughput of 6.1 TFLOPS, so the ratio of the theoretical throughput between the GPU and the CPU is about 3. In our study, the maximum observed ratio was about 2 for the supernova dataset. This result is consistent with previous findings,



(a) CPU strong scaling for fusion data. (b) CPU strong scaling for thermal hydraulics data. (c) CPU strong scaling for supernova data.

Figure 7: *CPU strong scaling for merging close points* ($\delta = 0.0001$) *: The dashed vertical line at 40 threads is where we start using logical CPU cores. The VTK data shown is the processing rate for the serial version.*



(a) Processing rates for fusion data (b) Processing rates for thermal hydraulics data (c) Processing rates for supernova data

Figure 8: *Comparison of processing rates for Virtual Grid (32-bit) approach across devices. These rates are based on the tolerance* $\delta = 0.0001$. *The TBB results are provided using 40 CPU cores.*

as GPUs are more memory bound than CPUs and our algorithm stresses memory.

7. Conclusions

We presented our Virtual Grid algorithm, a fast and scalable solution for merging points in a data set that occur within a certain user provided tolerance. This algorithm leverages VTK-m and its data parallel techniques.

Overall, our algorithm performs well on varied parallel environments and workloads. Our experiments reveal that the Virtual Grid approach works better than its competitors for merging points from the VTK and VTK-m libraries. Further, while our approach is slower for the application of mesh decimation, it yields smoother output meshes.

For future work, we expect further optimizations are possible with respect to optimizing memory accesses. We will also be exploring alternatives to the sort used for grouping points in the same Virtual Grid bins. We are considering hash-table–based approaches similar to that used by Lessley et al. for external facelist calculation [LBMC16, LPM*17].

8. Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[BK97] BAREQUET G., KUMAR S.: Repairing cad models. In *Proceedings. Visualization '97 (Cat. No. 97CB36155)* (Oct 1997), pp. 363–370. 2

[Ble90] BLELLOCH G. E.: *Vector models for data-parallel computing*, vol. 75. MIT press Cambridge, 1990. 2

[BMD03] BLONDIN J. M., MEZZACAPPA A., DEMARINO C.: Stability of standing accretion shocks, with an eye toward core-collapse supernovae. *The Astrophysical Journal 584*, 2 (February 2003), 971–980. 5

[CWSA16] CARR H. A., WEBER G. H., SEWELL C. M., AHRENS J. P.: Parallel peak pruning for scalable smp contour tree computation. In *2016 IEEE 6th Symposium on Large Data Analysis and Visualization (LDAV)* (2016), IEEE, pp. 75–84. 3

[FLPS08] FISCHER P., LOTTES J., POINTER D., SIEGEL A.: Petascale algorithms for reactor hydrodynamics. In *Journal of Physics: Conference Series* (2008), vol. 125, IOP Publishing, p. 012076. 5

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216. 2

[KTKN05] KANAYA T., TESHIMA Y., KOBORI K.-I., NISHIO K.: A topology-preserving polygonal simplification using vertex clustering. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2005), ACM, pp. 117–120. 2

[LBMC16] LESSLEY B., BINYAHIB R., MAYNARD R., CHILDS H.: External facelist calculation with data-parallel primitives. In *Proceedings of the 16th Eurographics Symposium on Parallel Graphics and Visualization* (2016), Eurographics Association, pp. 11–20. 3, 9

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 259–262. 2

[LLCC17] LI S., LARSEN M., CLYNE J., CHILDS H.: Performance impacts of in situ wavelet compression on scientific simulations. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2017), ISAV'17, ACM, pp. 37–41. 3

[LLN*15] LARSEN M., LABASAN S., NAVRÁTIL P. A., MEREDITH J. S., CHILDS H.: Volume rendering via data-parallel primitives. In *EGPGV* (2015), pp. 53–62. 3

[LMLC17] LESSLEY B., MORELAND K., LARSEN M., CHILDS H.: Techniques for data-parallel searching for duplicate elements. In *Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on* (2017), IEEE, pp. 1–5. 2

[LMNC15] LARSEN M., MEREDITH J. S., NAVRÁTIL P. A., CHILDS H.: Ray tracing within a data parallel framework. In *2015 ieee pacific visualization symposium (pacificvis)* (2015), IEEE, pp. 279–286. 3

[LPM*17] LESSLEY B., PERCIANO T., MATHAI M., CHILDS H., BETHEL E. W.: Maximal clique enumeration with data-parallel primitives. In *Large Data Analysis and Visualization (LDAV), 2017 IEEE 7th Symposium on* (2017), IEEE, pp. 16–25. 3, 9

[LSA12] LO L.-T., SEWELL C., AHRENS J.: Piston: A portable cross-platform framework for data-parallel visualization operators. 3

[LT97] LOW K.-L., TAN T.-S.: Model simplification using vertex-clustering. In *Proceedings of the 1997 symposium on Interactive 3D graphics* (1997), ACM, pp. 75–ff. 2

[LT98] LINDSTROM P., TURK G.: Fast and memory efficient polygonal simplification. In *Visualization'98. Proceedings* (1998), IEEE, pp. 279–286. 2

[MMA*13] MAYNARD R., MORELAND K., ATYACHIT U., GEVECI B., MA K.-L.: Optimizing threshold for extreme scale analysis. In *Visualization and Data Analysis 2013* (2013), vol. 8654, International Society for Optics and Photonics, p. 86540Y. 3

[MMM14] MILLER R., MORELAND K., MA K.-L.: Finely-threaded history-based topology computation. In *Proceedings of the 14th Eurographics Symposium on Parallel Graphics and Visualization* (2014), Eurographics Association, pp. 41–48. 2, 4

[MO15] MORELAND K., OLDFIELD R.: Formal metrics for large-scale parallel performance. In *International Conference on High Performance Computing* (2015), Springer, pp. 488–496. 7

[Mor18] MORELAND K.: *VTK-m User's Guide (Version 1.3)*. Tech. Rep. SAND 2018-13465 B, Sandia National Laboratories, November 2018. 4, 5

[MSU*16] MORELAND K., SEWELL C., USHER W., LO L., MEREDITH J., PUGMIRE D., KRESS J., SCHROOTS H., MA K., CHILDS H., LARSEN M., CHEN C., MAYNARD R., GEVECI B.: Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications 36*, 3 (May 2016), 48–58. 2, 5

[NVI18] NVIDIA CORPORATION: Thrust, Nov. 2018. https://developer.nvidia.com/thrust. 2

[PYK*18] PUGMIRE D., YENPURE A., KIM M., KRESS J., MAYNARD R., CHILDS H., HENTSCHEL B.: Performance-portable particle advection with vtk-m. In *Eurographics Symposium on Parallel Graphics and Visualization* (2018), Childs H., Cucchietti F., (Eds.), The Eurographics Association. 3

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in computer graphics*. Springer, 1993, pp. 455–465. 2

[RW92] ROCK S. J., WOZNY M. J.: Generating topological information from a bucket of facets. In *1992 International Solid Freeform Fabrication Symposium* (1992). 2

[SGG*04] SOVINEC C., GLASSER A., GIANAKON T., BARNES D., NEBEL R., KRUGER S., SCHNACK D., PLIMPTON S., TARDITI A., CHU M., ET AL.: Nonlinear magnetohydrodynamics simulation using high-order finite elements. *Journal of Computational Physics 195*, 1 (2004), 355–386. 5

[SM15] SCHROOTS H. A., MA K.-L.: Volume rendering with data parallel visualization frameworks for emerging high performance computing architectures. In *SIGGRAPH Asia 2015 Visualization in High Performance Computing* (2015), ACM, p. 3. 3

[SPC*04] SHIN H., PARK J. C., CHOI B. K., CHUNG Y. C., RHEE S.: Efficient topology construction from triangle soup. In *Geometric Modeling and Processing, 2004. Proceedings* (2004), IEEE, pp. 359–364. 2