Optimization and Augmentation for Data Parallel Contour Trees

Hamish A. Carr, Oliver Rübel, Gunther H. Weber, James P. Ahrens

Abstract—Contour trees are used for topological data analysis in scientific visualization. While originally computed with serial algorithms, recent work has introduced a vector-parallel algorithm. However, this algorithm is relatively slow for fully augmented contour trees which are needed for many practical data analysis tasks. We therefore introduce a representation called the hyperstructure that enables efficient searches through the contour tree and use it to construct a fully augmented contour tree in data parallel, with performance on average 6 times faster than the state-of-the-art parallel algorithm in the TTK topological toolkit.

Index Terms—Computational Topology, Contour Tree, Parallel Algorithms

1 Introduction

Computational science and engineering depend on ever-larger simulations of physical phenomena in projects such as the US Exascale Computing Project (ECP). These in turn depend on visualization, which increasingly requires analytic tools to support interpretation of data beyond human comprehension.

One of the principal such analytic tools is the *contour tree* or *Reeb graph*, which summarizes the development of contours in the data set as the isovalue varies. Since contours occur in many visualisations, the contour tree and the related *merge tree* are of prime interest in automated data analysis. To date, the application of the contour tree has been limited by the algorithms available. While there is a standard serial algorithm [6] for merge and contour trees, distributed and data-parallel algorithms are another matter. Although some approaches exist, they either target a distributed model [2], have serial sections [25], or do not come with strong formal guarantees on performance.

Most recently, Gueuenet et al. [15], [16] have reported shared-memory improvements to the existing contour tree algorithms, while Carr et al. [9] reported a new, fundamentally data-parallel approach to contour tree computation known as *parallel peak-pruning (PPP)*. While PPP is faster than competing approaches for computing the contour tree, it is less efficient for the fully augmented contour tree, in which not only critical points are represented but also every other vertex in the input mesh.

Since the fully augmented contour tree is required for secondary computations, such as geometric measures [8] that are necessary for practical data analysis tasks, improving PPP's efficiency for the fully augmented contour tree is the necessary next step towards exascale contour tree analysis of data.

In this paper, we report on a data structure that we call the *hyperstructure* that provides efficient parallel access to the contour tree for augmentation and subsequent processing. This structure is related to both branch decomposition [31] and rake-and-compress [14]. Moreover, while it arises naturally from the PPP algorithm, it can be computed directly from a known contour tree and therefore combined with any other contour tree algorithm.

Formally, the hyperstructure can be as efficient as the logarithmic rake-and-compress, but not in the presence of W-structures [19]. In practice, however the hyperstructure requires sub-logarithmic additional cost, and the additional overhead to compute the fully augmented contour tree is therefore similar to the cost of computing the contour tree in the first place. Moreover, queries in the hyperstructure have guaranteed logarithmic cost and can be executed in parallel for use in augmentation.

We have fully implemented the hyperstructure in the open source VTK-m multicore visualization toolkit, and compare its performance both with a previous iteration of the PPP algorithm and with the rival TTK toolkit.

Our contributions are therefore:

- The definition of the hyperstructure for a contour tree
- A parallel algorithm to build the hyperstructure
- An algorithm to search for a point in the hyperstructure
- A parallel algorithm to fully augment the contour tree
- Formal analysis of the complexity of these algorithms
- An implementation of the hyperstructure in VTK-m
- Empirical performance reports for the hyperstructure

We review related work (Section 2), including the PPP algorithm (Section 2.4), then introduce the hyperstructure (Section 3), its computation, application and analysis, and its use to construct the fully augmented contour tree. We then study performance and scaling (Section 5), before concluding (Section 6).

2 Background

We will start with some basics of data-parallel computation in Section 2.1 and of contour trees in Section 2.2, discuss branch decompositions in Section 2.3, then introduce the recent parallel peak-pruning (PPP) algorithm in Section 2.4, and discuss the relationship with parallel tree contraction in Section 2.5.

- *Hamish A. Carr is with the University of Leeds, United Kingdom. E-mail: H.Carr@leeds.ac.uk.*
- *Oliver Rübel is with the Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, California, United States. E-mail: ORuebel@lbl.gov*
- *Gunther H. Weber is with the Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, California, United States and the Department of Computer Science, University of California, Davis, California, United States.*
- *James P. Ahrens is with the Los Alamos National Laboratory, Livermore, New Mexico, United States. E-mail: ahrens@lanl.gov.*

Manuscript received MMMM DD, 2020;

2.1 Data-Parallel Computation

Data-parallelism is an effective method for exploiting shared-memory parallelism on GPUs and multi-core CPUs. Blelloch [3] defined a scan vector model and showed that many algorithms can be implemented using a small set of "primitive" operators such as transform, reduce, and scan, which can each be implemented in a constant or logarithmic number of parallel steps.

Algorithms using this model run portably across multi-core and many-core architectures such as OpenMP, parallel C++ STL, NVIDIA's CUDA, and Intel's TBB, with architecture-specific optimisations isolated to the data-parallel primitives in the backends.

Under ECP, the next generation of the VTK [35] toolkit, VTK-m [27] (m for multicore), exploits this data-parallel approach to build tools for the next generation of exascale hardware. Together with its predecessor, PISTON [24], it has already been shown to be an effective tool for computing isosurfaces, cut surfaces, thresholds, Kd-trees [36] and halo finders [18], and most recently, the PPP algorithm for contour trees [9].

2.2 Contour Trees

Given $f : \mathbb{R}^d \to \mathbb{R}$, a *level set* is the inverse image $f^{-1}(h)$ of an *isovalue h*, and a *contour* is a connected component of a level set. The *Reeb graph* is the quotient space under continuous contraction of each contour to a point [34], preserving relationships between contours of different isovalues. For simply connected domains, this graph is acyclic and called the *contour tree*.

We assume that f is defined by an interpolant on a mesh M, most commonly simplicial or cubic, and that all vertices have unique isovalues, usually guaranteed by simulated simplicity [13]. For such a function, each contour can contain at most one mesh vertex, and each mesh vertex therefore belongs to a unique point in the tree. We refer to this as a *regular node* and observe that the regular nodes separate the tree into segments connecting pairs of regular nodes, which we refer to as *regular arcs*.

Most regular nodes and arcs are structurally redundant, so algorithms focus on the *critical points* in the mesh, where the number of contours change. Since in higher dimensions a critical point can change genus without changing connectivity, we refer to these as *supernodes* which are connected by chains of regular nodes, which are replaced by *superarcs*: each superarc is then treated as the *superparent* of the regular nodes and arcs on the chain. For convenience, we refer to the set of regular nodes and arcs as the *regular structure*, while the supernodes and superarcs form the *superstructure*.

While the contour tree is based on level sets, trees can also be defined based on the connectivity of super-level sets $\{x : f(x) \geq h\}$ or sub-level sets $\{x : f(x) \leq h\}$, typically using a value-ordered sweep or *filtration* to construct *merge trees*, which can be used independently for data analysis or for computing the contour tree.

In the contour tree, features are defined in terms of pairs of critical points, usually a saddle-extremum pair. This is of value for data analysis because the inverse image of the path in the contour tree between the pair of points defines a region in the data which is presumed to be a feature of spatial significance. The importance of a given feature can then be defined in terms of properties such as height, volume, integrable value, or summary statistics of the corresponding region [8]. Many of these properties can most readily be computed if the tree is *fully augmented* - i.e. it contains all regular points in addition to the critical points.

2.2.1 Sweep And Merge Algorithm for Contour Trees

For simplicial meshes on simply connected domains, the *sweep and merge* algorithm [6] performs a sorted sweep over the vertices, incrementally adding them to a union-find data structure [38]. As components are created or merged, critical points are identified, and a merge tree is constructed. After descending and ascending sweeps, two merge trees are combined to produce the contour tree.

This algorithm is a graph algorithm applied to the set of vertices and edges of a simplicial mesh, and topology graphs [5] were therefore introduced for arbitrary meshes and interpolants, and even for digital connectivity in images and volumes. A topology graph consists of a set of vertices which includes all critical points of the interpolant, and a set of edges such that any monotone path in the mesh between vertices can be mapped to a monotone path in the graph between the same vertices.

2.2.2 Scaling Sweep and Merge

While the sweep and merge algorithm is simple and efficient, it uses a sequential sweep through the contours, hindering its parallelization. Pascucci & Cole-McLaughlin [30] gave a distributed method that divides the data into spatial blocks, computes the contour tree separately for each block, then combines the contour trees for adjacent blocks into a topology graph, and repeats the process with larger blocks until a single compute node holds the entire contour tree. This was extended by Acharya & Natarajan [2], who combined it with the hill-climbing topology graph construction of Chiang et al. [11]. However, the parallel fan-in strategy forces the full contour tree to reside on the final compute node. For noisy or complex data sets, the contour tree size is nearly linear in input size. The resulting memory footprint on the central node is then impractical, since it prevents distributing storage cost, and only distributes compute costs partially.

One way to address memory constraints is the use of streaming algorithms. Pascucci et al. introduced a streaming algorithm for Reeb graphs [32], which served as basis for a streaming merge treee algorithm for large-scale combustion data sets [4]. Another approach consists of distributing data over multiple compute nodes. Morozov & Weber [28] introduced a distributed merge tree representation to avoid the large memory footprint as well as computation bottlenecks caused by a global fan-in. In this distributed representation, each compute node stores an exact merge tree for its own portion of the data set along with just enough supernodes from the full merge tree to support correlating these "local-global" representations with each other. To utilize on-node parallelism, they introduced skip trees to improve the computational efficiency of streaming merge tree computation algorithm by Bremer et al. [4]. In later work, they generalized this approach to contour trees [29] by computing distributed representations of merge trees and providing algorithms for common contour tree queries.

Similarly, Landge et al. [23] introduced segmented merge trees for segmenting data and identifying threshold-based features. Their approach constructs local merge trees and corrects them based on neighbouring domains. By considering features only up to a predefined size, this correction process requires less communication compared to the approach by Morozov & Weber [28]. Widanagamaachchi et al. [41] then described a data-parallel model for the merge tree, breaking the computation into a finite number of fan-in stages. This approach in effect quantised the merge tree, an effect that was acceptable for the task in hand.

Maadasamy et al. [25] find critical points then construct monotone paths [11] on GPU from saddles to extrema, use them to build topology graphs, then compute merge trees and contour trees in serial on CPU. While efficient in practice, the algorithm is greedy in nature with weak formal bounds. Moreover, this algorithm only computes the unaugmented contour tree.

Gueunet et al. [15] instead reduced the serial sweep cost by computing separate contour trees on different threads for subranges of the scalar value [15], and later introduced a task-based algorithm that decouples the sweep for separate peaks, with a common pool of small sweeps shared by all threads [16].

Smirnov & Morozov [37] have also described a shared memory algorithm based on compare-and-swap primitives rather than the vector primitives of the PPP algorithm [9] or the task-based parallelism of Gueunet etal [15].

2.3 Branch Decomposition

Before the contour tree can be used for analysis, it is typically simplified through *branch decomposition* [31], which builds monotone chains of superarcs called *branches* to connect extrema with saddles. Branches are built by assigning superarcs a priority, then sequentially removing the lowest priority superarc: if an interior supernode becomes regular in the process, it is also removed, and its remaining edges are concatenated into a larger branch.

The priority chosen for a superarc may be based on any of a variety of properties, such as, volume or hypervolume [7], but was originally defined to be the height of the edges, i.e. the difference in data values at the top and bottom, and this is often referred to as *the* branch decomposition. We note that recent work [10], [19] has identified *W-structures* that zig-zag horizontally across a contour tree. These correspond to nested ring-like structures such as volcanic calderas, alternating shells of high and low density, or ridges across a data set, and cause both theoretical and practical problems. For a simple example of a W-structure, we refer the reader to supernodes $284-35-68-67$ in Figure 2.

These branches generally correspond to cancellation pairs in persistent homology [12], but not if a W-structure is present [19]. We therefore use *height* of a feature rather than *persistence* to avoid confusion. Moreover, W-structures exist in practical data and cause both the branch decomposition and the PPP algorithm to serialise along the edges of the W-structure.

Any branch decomposition then defines a hierarchy of features for semantic analysis, but it is based on serial removal of branches, which poses problems in terms of parallel efficiency.

Previous work [31], [40] showed that branch decomposition can be used to search for an arbitrary point p in the contour tree, by identifying a monotone path passing through p, mapping it to a sequence of branches, identifying the branch where the monotone path passes through p's isovalue, then searching along the branch to identify the superarc on which the point lies.

More recent work varied this idea to support standard operations without computing the contour tree. Morozov & Weber [28], [29] use a distributed representation in which queries are performed for various properties, as do Klacansky et al. [22]. A key observation is that once a search structure has been constructed, searches for different points are independent of each other, allowing parallel search for all points in a data set simultaneously, which forms the basis for our contour tree augmentation. We have subsequently described [20] further operations on the hyperstructure that allow us to approximate the standard branch decomposition, and perform generalised operations over a contour tree with it.

2.4 Parallel Peak Pruning Algorithm

While other merge algorithms retain the serial notion of a sweep, the parallel peak pruning algorithm [9] discards it in favour of data parallel operations. For each maximum or *peak*, the closest saddle by height that connects the peak to another peak is the *governing saddle*. Unlike the pairing constructed in branch decomposition, this peak-governing saddle pairing is independent of any other pairing, and can therefore be found efficiently in parallel.

Parallel peak-pruning (PPP) prunes all peaks to their governing saddles in parallel. In doing so, existing governing saddles are transformed to regular points or new peaks, and subsequent iterations prune more and more of the tree in each pass, with a guaranteed logarithmic bound on the number of iterations. Figure 1 (right) shows this for a known merge tree used as its own topology graph. Here, all upper leaves $(7-13)$ are identified in parallel together with their governing saddles in the first iteration. In the second iteration, vertices $0-6$ remain to be processed, but form a simple chain with 6 as the peak (which used to be a saddle). The algorithm removes the entire chain in a single pass, guaranteeing logarithmic or better performance.

PPP modifies the second phase of sweep-and-merge to transfer all upper leaves at once, then alternates upper and lower leaf transfers in separate rounds, collapsing chains of regular nodes as described in Section 3.2. If no W-structures are present, the number of iterations is guaranteed to be logarithmic, but if they are present, the complexity depends on how many reversals or "kinks" are present in the W-structure. In practice, as we will see below, the number of iterations is typically sub-logarithmic.

One of the optimisations in the PPP algorithm is the reduction of the initial mesh to a topology graph in order to avoid repeated operations over the entire input data. If all edges in a simplicial mesh are used as the input topology graph, the algorithm will compute the fully augmented contour tree, but with a prohibitive performance penalty. Since full augmentation is required for many of the operations reported for the contour tree, accelerating the fully augmented contour tree is still needed.

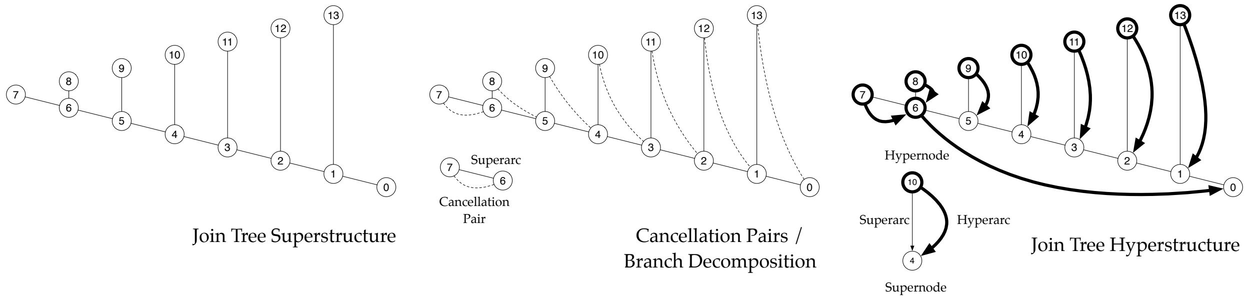
2.5 Generalized Superstructure

Branches are monotone chains of superarcs, which are monotone chains of regular arcs, so branches are also a generalisation of the superstructure. Unfortunately, branch decomposition is not parallel friendly, primarily because it removes branches sequentially.

Collapsing chains of edges in a graph is not restricted to contour trees: one general approach for efficient parallel traversals over rooted trees is parallel tree contraction [14], [26]. This alternates *rake* operations, which remove all leaves, and *compression* operations which replace all chains of degree-2 vertices with single edges. Each pair of rake-and-compress operations reduces the tree to a smaller tree with at most half as many vertices, guaranteeing logarithmic performance even for unbalanced trees.

While parallel-friendly, this does not guarantee the monotone chains that we need for search operations. In a merge tree, where the distance from the global root increases with isovalue, the first phase of PPP is the same as parallel tree contraction, and all chains generated are monotone, with the result that parallel tree contraction for a merge tree gives the hyperstructure (below).

For contour trees, there is no unambiguous global root, and parallel tree contraction may collapse chains in a W-structure, which are not monotone. Thus, while merge trees can exploit parallel tree contraction, efficient search operations in contour trees require a new, although closely related, search structure.



Join Tree Superstructure

Cancellation Pairs / Branch Decomposition

Join Tree Hyperstructure

Fig. 1: Merge Tree Superstructure, Branch Decomposition/Cancellation, and Hyperstructure. The superstructure connects supernodes with superarcs. The branch decomposition pairs peaks uniquely with saddles, but linearizes in this case. The hyperstructure gives efficient parallelism. Note that hyperarcs always start at hypernodes, but may end at supernodes.

2.6 Summary

While contour trees are a standard tool for data analysis, their use as data scales depends on parallel algorithms for construction, augmentation, and queries. Some scaling approaches with local SMP and distributed communication parallelism have been described, including the PPP algorithm, which computes the unaugmented contour tree. Further development requires fully augmented contour trees, which can be found if efficient parallel query operations are supported, but existing data structures are either not parallel friendly or do not generate monotone chains for query operations.

3 Hyperstructure

We now know that we want a parallel friendly branch decomposition with similar properties to parallel tree contraction. We therefore define a new variant - the *hyperstructure*, that

- is similar to parallel tree contraction [14], [26],
- is also a form of branch decomposition,
- arises naturally from the PPP algorithm,
- but does not depend on PPP,
- compresses monotone chains in the contour tree,
- can therefore be used for efficient isovalue queries, and
- can be used for efficient full augmentation in parallel.

3.1 Hyperstructure Definition

Assume an unrooted tree $T = \{E, S, f\}$ with supernode set S, edge set of superarcs E and a function f that assigns a unique value $f(s)$ to each vertex $s \in S$. Define an upper leaf edge of T to be a leaf edge $\{r, s\}$ where r is of degree 1 and $f(r) > f(s)$. Define lower leaf edges symmetrically to be $\{r, s\}$ where r is a leaf and $f(r) < f(s)$: given unique values, these two cases are exhaustive.

We reduce T by contracting alternate upper and lower leaf chains with a combined rake and compress operation. These leaf chains are not identical to those in parallel tree contraction, but similar. Algorithm 1 gives algorithmic expression to this idea.

An upper leaf chain $s_0, \ldots, s_k$ is a monotone chain from an upper leaf s_0 through regular (degree 2) vertices $s_1, \ldots, s_{k-1}$ to a critical point s_k such that $f(s_j) > f(s_{j+1}) \forall j \in \{0, \ldots, k-1\}$. The degree constraint and monotonicity mean that no vertex s may be part of two upper leaf chains except the lowest vertex s_k. Each upper leaf chain is replaced with a single *hyperarc* $\{s_0, s_k\}$. In serial the order we choose leaf chains is significant, but in parallel we transfer all available upper leaf chains simultaneously.

We define lower transfer symmetrically and orient all hyperarcs so that they originate at a leaf and terminate at a critical point. We alternate between upper transfer and lower transfer until the tree has no remaining edges, at which point the remaining vertex becomes the root of the tree. Since any tree must have either upper or lower leaves, the tree shrinks in each pair of passes, and the process is guaranteed to terminate. In the final pass, only one edge is left, and the “inner” end is taken as the root for the tree.

The hyperstructure is the set of all hyperarcs identified during transfer operations, with all edges in a leaf chain treated as children of the hyperarc. We call only the leaf end of each hyperarc a *hypernode*, giving a $1-1$ correspondence between hypernodes and hyperarcs, and we orient regular, super and hyperarcs accordingly: lower leaf transfers thus create ascending hyperarcs, while upper leaf transfers create descending hyperarcs, and all regular, super and hyperarcs point inwards towards the root of the tree, which is given a virtual hyperarc and treated as a hypernode.

Comparing with parallel tree contraction, upper reduction is a compression operation and immediate rake for upper leaf chains: in the first upper transfer, all hyperarcs will consist of single leaf edges, as will most hyperarcs in the first lower transfer, but not all, as upper transfers can create non-trivial lower leaf chains. Pairs of upper and lower passes are therefore *not* equivalent to a single rake and compression, although there is a strong similarity.

While the hyperstructure is *a* branch decomposition, it is not defined by a linear sequence of reductions, and may not correspond directly to a specific branch decomposition. However, like branch decomposition, all chains are monotone, which permits binary search along chains to find a specific edge (i.e., superarc).

3.2 Hyperstructure Construction

Algorithm 1 is a parallel algorithm for computing hyperstructure, starting with superarcs as candidates for hyperarcs. Since supernodes become regular in the process, we use “vertex” and “edge” during the process, to avoid clumsy terminology such as “regular supernode”. During compression steps, vertices are labeled with the hypernode for the edge as their *hyperparent*. During reduction steps, edges are identified as hyperarcs and transferred to the final set. We also track the iteration in which each hyperarc is transferred, as we will need this later.

We initialize the hyperstructure to empty in Step I, then alternate upper and lower transfers. Each iteration has three major stages: chain identification (Step IIA-1 and 2), hyperarc creation (Step IIA-3) and edge removal (Step IIA-4).

In Step IIA-1, since regular vertices have one ascending arc and one descending, we set their up and down neighbours accordingly. For upper leaves, we set the down neighbour similarly

Algorithm 1 Algorithm for Constructing Hyperstructure

```
Require: Contour Tree $T = \{E, V, f\}$
Require: That all edges $e = (u, v) \in E$ have $f(u) > f(v)$
  // Step I: Initialisation
  Let $N = \{\}$ be the set of hypernodes
  Let $A = \{\}$ be the set of hyperarcs
  Let $C = E$ be the initial set of potential chains
  Let $W = V$ be the working set of vertices
  Let $i = 0$ be the iteration
  while $\|W\| > 1$ do
    if $i$ is even then
      // Step IIA: Parallel Upper Transfer
      // Step IIA-1: Initialisation:
      for all Vertices $v \in W$ do
        if $v$ is regular with edges $(u, v), (v, w) \in C$ then
          $Up(v) = u$, $Down(v) = w$
        else
          if $v$ is an upper leaf with edge $(v, w) \in C$ then
            $Up(v) = v$, $Down(v) = w$
          else
            $Up(v) = v$, $Down(v) = v$
          end if
        end if
      end for
      // Step IIA-2: Pointer Doubling to Detect Chains
      for $\lg \|W\|$ iterations do
        for all Vertices $v \in W$ do
          $Up(v) = Up(Up(v))$
          $Down(v) = Down(Down(v))$
        end for
      end for
      // Step IIA-3: Hyperarc Creation
      for all Vertices $v \in W$ do
        if $v$ is an upper leaf then
          Let $c = (v, Down(v))$
          Set $Parent(v) = c)$
          Set $Iteration(c) = i$
          Add $c$ to $A$, $v$ to $N$
        end if
      end for
      // Step IIA-4: Edge Removal
      for all Vertices $v \in W$ with single edge $(v, w) \in C$ do
        if $Up(v)$ is a leaf then
          Set $Parent(v) = Parent(Up(v))$
          Remove $(v, w)$ from $C$
          Remove $v$ from $W$
        end if
      end for
    else
      // Step IIB: Parallel Lower Transfer is symmetric
    end if
    Increment $i$
  end while
  // Step III: The Root Vertex
  Let $w \in W$ be the root vertex
  Let $c = (w, NIL)$ be the virtual root edge
  Set $Parent(w) = c$
  Set $Iter(c) = i$
  Add $w$ to $N$ and $c$ to $A$
  Return $H = \{N, A, Iter, Parent\}$
```

but the up neighbour to the leaf itself. For any other vertex, we set both up and down neighbour to the vertex.

After this step, the down neighbour of upper leaf will be the lower end of its chain. Step IIA-3 therefore creates hyperarcs from all upper leaves, setting the hyperparent and iteration number.

Every upper leaf or regular vertex has a single downward edge, and Step IIA-4 removes any such vertex whose up neighbour is a leaf, leaving interior regular vertices in the tree. The hyperparent for each vertex is set to the leaf end of the new hyperarc, and edge and vertex removed from the working sets C and W.

After a finite number of iterations, the working set W of vertices will have one remaining vertex, which becomes the root of the tree. Step III then creates a virtual hyperarc and adds both root and virtual hyperarc to the hyperstructure, with hyperparent and iteration being set accordingly.

On completion the hyperstructure consists of $H = \{N, A, Iteration, Parent\}$ in addition to the underlying Contour Tree $T = \{E, V, F\}$. The PPP algorithm [9] executes these steps during batched parallel construction of the contour tree, identifying leaf chains in the merge trees rather than in the contour tree directly. As a result, constructing the hyperstructure during PPP is straightforward, but the hyperstructure can be constructed directly from the contour tree, so is independent of PPP.

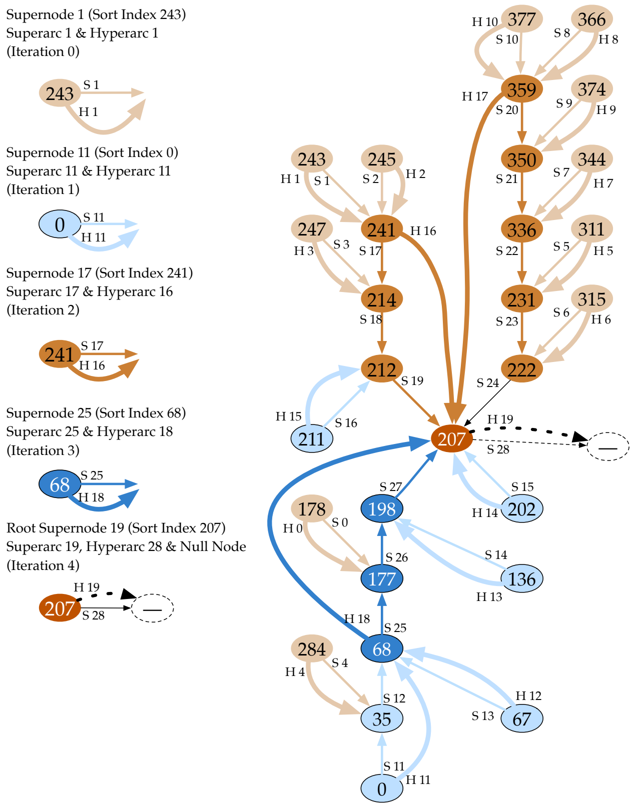
3.3 Hyperstructure Search

Once the hyperstructure has been constructed, we can search for the superarc or arc to which a point p belongs. As in Section 2.3, if p is in a cell of the mesh, two monotone paths from p are constructed in the cell to vertices u, v with higher and lower values than p. u and v are then located in the tree, and the monotone path between them searched using the "parent" relationships in a branch decomposition. We substitute hyperstructure for branch decomposition, and search by treating hyperarcs with higher iteration numbers as parents, resulting in Algorithm 2. Based on Section 2.4, we assume that we have regular nodes and arcs sorted along each superarc, and that we can determine the superparent for each regular vertex (i.e., the superarc to which the vertex belongs).

We start with a pair A, B of vertices on a monotone path through p, with $f(A) > f(p) > f(B)$. Step I extends this to a pair of supernodes then hypernodes on a monotone path through p, with $f(HNodeA) > f(p) > f(HNodeB)$. In general, these hypernodes will belong to different iterations. We take the hyperarc from the lower iteration in Step IIA and check whether its isovalues span $f(p)$: if so, we have found the correct hyperarc and jump to Step IV. If not, we substitute the inner end of the hyperarc for its outer end, update the spanning hypernodes and repeat. If the hyperarcs belong to the same iteration but are not identical, we can remove either, and Step IIA executes the other branch, whose details are symmetric. If the two hyperarcs match, the point must lie on them, and we fall out of the while loop at Step III to the binary search. Finally, since the iteration number of one of the spanning hypernodes increases in each pass, and there are a finite number of iterations, we are guaranteed termination.

3.4 Contour Tree Augmentation

This algorithm works for any point [40] including mesh vertices, and we use this to augment the contour tree. In the PPP algorithm, two pointer-doubling steps generate monotone paths from every vertex of the mesh to a local maximum and a local minimum. This is used to identify critical points and extract a topology graph


Supernode 1 (Sort Index 243)
Superarc 1 & Hyperarc 1
(Iteration 0)
243
S 1
H 1
Supernode 11 (Sort Index 0)
Superarc 11 & Hyperarc 11
(Iteration 1)
0
S 11
H 11
Supernode 17 (Sort Index 241)
Superarc 17 & Hyperarc 16
(Iteration 2)
241
S 17
H 16
Supernode 25 (Sort Index 68)
Superarc 25 & Hyperarc 18
(Iteration 3)
68
S 25
H 18
Root Supernode 19 (Sort Index 207)
Superarc 19, Hyperarc 28 & Null Node
(Iteration 4)
207
H 19
S 28
377
366
359
374
350
344
336
311
231
315
222
243
245
241
247
214
212
211
207
178
198
202
177
136
284
68
35
67
0

SUPERSTRUCTURE ARRAYS:
H-Hyper S-Super
A-Ascending D-Descending X-Neither

SN	Val	Idx	Srt	SA	HP	It	Flags
0	1	60	178	26	0	0	H D
1	4	334	243	17	1	0	H D
2	4	349	245	17	2	0	H D
3	4	377	247	18	3	0	H D
4	26	19	284	12	4	0	H D
5	34	101	311	23	5	0	H D
6	36	36	315	24	6	0	H D
7	60	152	344	22	7	0	H D
8	71	230	366	20	8	0	H D
9	86	178	374	21	9	0	H D
10	91	226	377	20	10	0	H D
11	0	0	0	12	11	1	H A
12	0	41	35	25	11	1	S A
13	0	81	67	25	12	1	H A
14	0	287	136	27	13	1	H A
15	1	312	202	28	14	1	H A
16	1	351	211	19	15	1	H A
17	4	328	241	18	16	2	H D
18	1	372	214	19	16	2	S D
19	1	352	212	28	16	2	S D
20	63	229	359	21	17	2	H D
21	60	223	350	22	17	2	S D
22	55	153	336	23	17	2	S D
23	4	146	231	24	17	2	S D
24	3	79	222	28	17	2	S D
25	0	82	68	26	18	3	H A
26	1	59	177	27	18	3	S A
27	1	286	198	28	18	3	S A
28	1	333	207	-1	19	4	H X

HYPERSTRUCTURE ARRAYS:
A-Ascending D-Descending
X-Neither

HN	SN	HA	Flags
0	0	26	D
1	1	17	D
2	2	17	D
3	3	18	D
4	4	12	D
5	5	23	D
6	6	24	D
7	7	22	D
8	8	20	D
9	9	21	D
10	10	20	D
11	11	25	A
12	13	25	A
13	14	27	A
14	15	28	A
15	16	19	D
16	17	28	D
17	20	28	D
18	25	28	A
19	28	-1	X

SN:	Supernode ID
Val:	Value
Idx:	Mesh Index
Srt:	Sort Index
SA:	Superarc Destination
HP:	Hyperparent
It:	Iteration
HN:	Hypernode ID
HA:	Hyperarc Destination

Fig. 2: Contour Tree Hyperstructure for the 18×21 data set vanc.txt in the supplementary material. Vertices are labelled by their sort order rather than data value. Shading indicates the order in which the supernodes are transferred (i.e. the iteration), in increasing order of darkness. Superarcs along each hyperarc are stored consecutively in the arrays.

for the algorithm. If the pair of extrema $Max(v), Min(v)$ for each vertex v is saved, a final step can be added in which every regular point searches for its superarc by calling Algorithm 2 with $A = Max(v), B = Min(v)$. The regular points are then sorted along the superarc as in PPP to establish the correct regular arcs.

3.5 Hyperstructure Analysis

The hyperstructure is not as formally efficient as the parallel tree contraction, since the alternating transfers do not give the same collapses, except for merge trees. Since the merge tree has a global extremum as its root, the hyperstructure of a merge tree is the same as the parallel tree contraction. If we consider the merge tree hyperstructure in Figure 1, we see that it will be identical to parallel tree contraction if we perform upper transfer operations only. If we also perform lower transfers, the lowest superarc may be transferred from below, giving a different result. However, if we restrict ourselves to upper transfer operations, the hyperstructure of a merge tree will be the same as the parallel tree contraction.

A trivial bound of $O(t)$ iterations can be shown, each of $O(t \lg t)$ work and $O(\lg t)$ time, where t is the size of the tree. No tighter bound has yet been established, due in part to W-structures, which serialize the PPP algorithm [9]. For contour trees without W-structures, however, recall that the first iteration can cause a lower chain to appear in the second iteration. For this to happen, however, there must be a W-structure. It then follows that if we have no W-structures, the paired iterations remove all leaves of the tree. Since the tree is perfectly balanced, this means that there are no regular vertices and the residue of the tree is still perfectly balanced. Inductively, therefore, we will need $O(\lg t)$ iterations. As with parallel tree contraction, the hyperstructure typically takes fewer than $\lg t$ iterations for imbalanced trees, and Section 5 gives details on the practical efficiency of the hyperstructure.

The PPP algorithm uses $O(\lg T)$ steps to construct the merge trees, and the hyperarcs in the hyperstructure all correspond to removing hyperarcs in a merge tree. The search in Algorithm 2 thus takes at most $O(\lg t)$ passes to find the correct hyperarc for a given point, followed by at most $O(\lg t)$ time for the binary search for the superarc and $O(\lg N)$ time for the binary search for the regular arc, where N is the number of regular nodes.

Full augmentation of the contour tree can then be done entirely in parallel, taking $O(N \lg t)$ work and $O(\lg t)$ time for search and $O(N \lg N)$ work in $O(\lg N)$ time for the final sort.

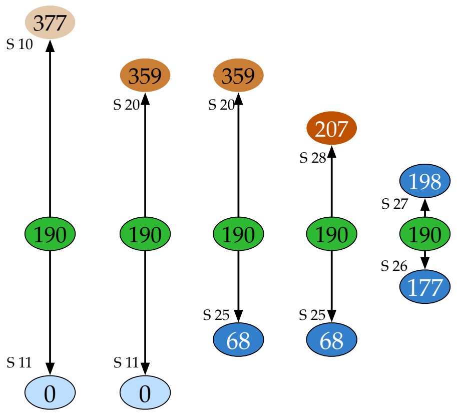
4 Hyperstructure Examples

Figure 2 illustrates the hyperstructure and its array implementation for a small 19×21 section of GTOPO30 around Vancouver, which is provided in the supplementary material. While small, this already shows several salient features. For example, the first pair of transfers (white, light grey) remove all of the leaves, with most hyperarcs being length 1, although $0-35-68$ is of length 2 is an example of the effect of a W-structure.

Next, the second pair (mid grey, dark grey) prunes chains of $3-5$ supernodes each, where pure logarithmic performance would predict doubling chain length instead. Finally, the last transfer

Algorithm 2 Algorithm for Searching in Hyperstructure

```
Require: Contour Tree $T = \{E, V, f\}$
Require: Hyperstructure $H = \{N, A, Iter, Parent\}$
Require: Search Isovalue $f(p)$
Require: Search Path $A, B \in V : f(A) > f(p) > f(B)$
  // Step I: Extending Arcs to Hyperarcs
  Let $SArcA$ be the superparent of $A$
  Let $SNodeA$ be the upper end of $SArcA$
  Let $HArcA$ be the hyperparent of $SNodeA$
  Let $HNodeA$ be the upper end of $HArcA$
  Let $SArcB$ be the superparent of $B$
  Let $SNodeB$ be the lower end of $SArcB$
  Let $HArcB$ be the hyperparent of $SNodeB$
  Let $HNodeB$ be the lower end of $HArcB$
  // Step II: Search until Hyperarcs Meet
  while $HArcA \neq HArcB$ do
    // Step IIA: Truncate Older Hyperarc
    if $Iter(HArcA) > Iter(HArcB)$ then
      Let $S$ be the inner end of $HArcB$
      if $f(S) > f(p)$ then
        Set $HArc = HArcB$
        Goto Step IV
      else
        Set $SNodeB = S$
        Set $HArcB$ to the hyperparent of $S$
        Set $HNodeB$ to the lower end of $HArcB$
      end if
    else
      Symmetric to the above
    end if
  end while
  // Step III: Fall through with Matching Hyperarcs
  Set $HArc = HArcA$
  // Step IV: Binary Search on Hyperarc and Superarc
  Search on $HArc$ for superarc $SArc$ spanning $f(p)$
  Search on superarc $SArc$ for arc $Arc$ spanning $f(p)$
  Return $SArc, Arc$
```



	Initial Path	Step #0	Step #1	Step #2	Binary Search
a	S10	S20	S20	S28	
f(a)	377	359	359	207	
H(a)	H10	H17	H17	H19	
I(a)	0	2	2	4	
b	S11	S11	S25	S25	
f(b)	0	0	68	68	
H(b)	H11	H11	H18	H18	
I(b)	1	1	3	3	

Fig. 3: Searching for a superarc in Figure 2. Each column represents the supernodes defining a monotone path through the search value 190 at a given stage of the search. We remove the ends iteratively until one passes the search value, then use a binary search to find the exact superarc.

removes 207 (in black), connecting it to a virtual supernode outside to the tree, and establishes 207 as the root supernode.

In Figure 3, we search for $f(c) = 190$ in the contour tree from Figure 2 along a monotone path from 377 to 0. Initially, $I(a) = 0$ and $I(b) = 1$, so we prune a first to 359. $I(a)$ is now 2, so we prune b to 68 and so on. This alternates between upper and lower ends, but it does not have to. The fourth prune (of 68) would pass 190, so we have found the correct hyperarc $H(c) = 18$.

To find the superarc $S(c)$, we take the supernode ID 25 for hypernode 18 and supernode ID 28 for the next hypernode 19, and observe that supernodes $25, 26, 27$ are in sorted order $68, 177, 198$ along hyperarc 18, terminating in supernode 28 with index 207. A binary search in this subsequence then identifies that since $198, 207$ span 190, c must belong on superarc $S(c) = 27$. In this instance, the destination supernode 28 was contiguous with the hyperarc's supernode sequence, but this will not always be true, which requires a minor modification to the standard binary search.

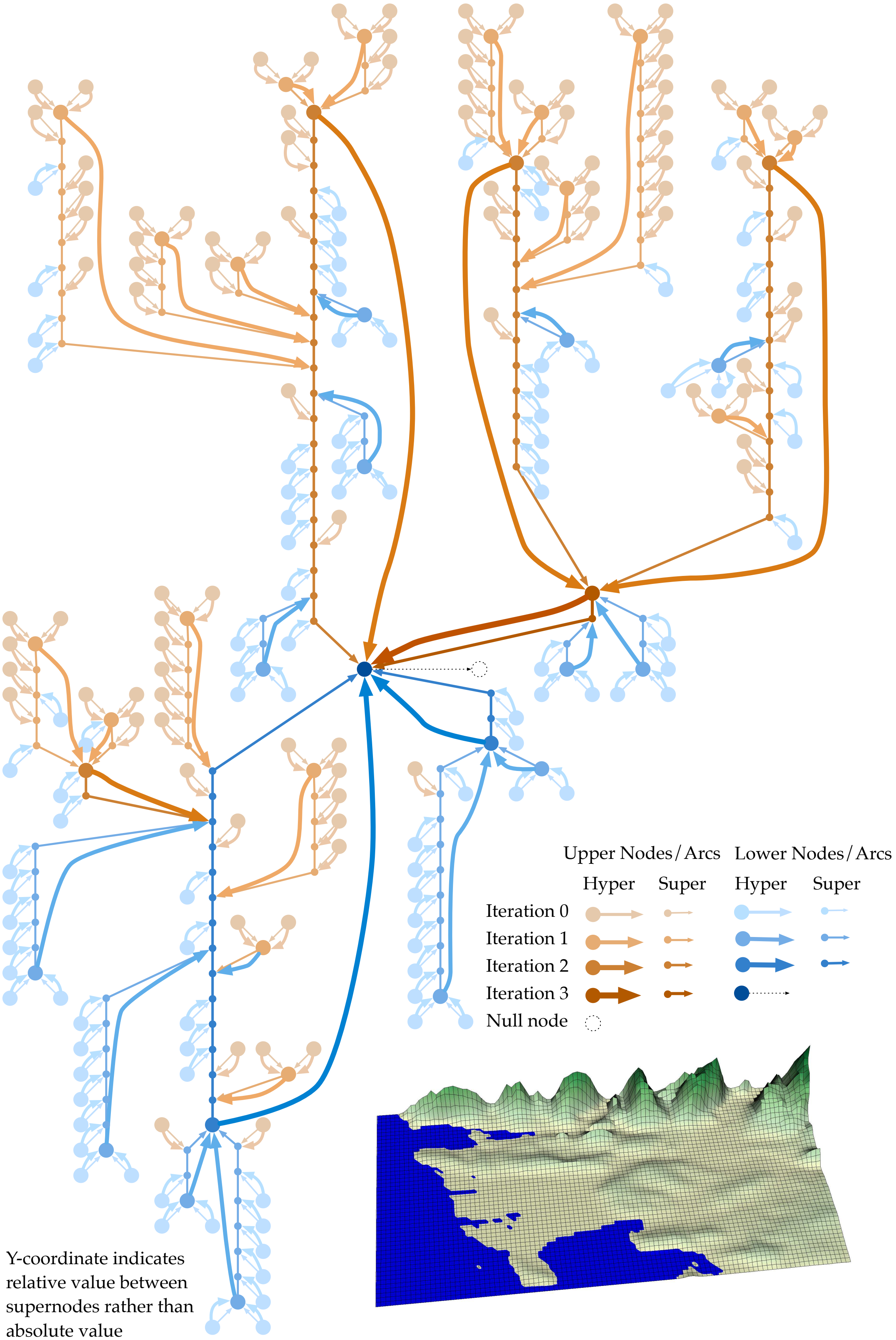
We show in Figure 4 a slightly larger 50×100 data set from near Vancouver. Here, we suppress node IDs for clarity and show superarcs as straight edges and hyperarcs as curved edges. As Table 1 shows, the hyperarcs generally increase in length in each iteration. We note that the merge phase transfers upper and lower leaves separately, so we treat each such pair as a single iteration with two sub-phases. However, since they are sequential in practice, the lower sweep sometimes removes saddle points if the upper sweep has reduced them to regular.

Again, we observe that in the first two iterations, nearly every hyperarc consists of a single superarc, but thereafter the hyperarcs consist of increasing numbers of superarcs. Clearly, this is data-dependent, but we can make several observations. First, the first (upper) iteration will always have hyperarcs of length 1, as will the last when the root supernode is transferred. The second (lower) iteration may be able to collapse chains, but will usually be dominated by hyperarcs of length 1. As a result, while there is no strict lower bound, the number of hyperarcs is often at least $1/2t$, but never more than t.

We also provide summary statistics for GTOPO30 (922M samples) and Pawpawsaurus (673M samples), where we see that the iterations tend to capture longer and longer chains of superarcs, resulting in a data structure with less than logarithmic cost, even though no such formal guarantee has been proved.

We note two patterns in this. First, the number of superarcs removed increases as side branches are removed, but many superarcs are removed late in the process. Thus, cost per iteration remains relatively high, but once the contour tree has been computed, our search cost is bounded by the number of iterations — in these cases at most 10 pairs iterations. In contrast, a logarithmic cost on $37-77M$ supernodes would be expected to be over 20.

Secondly, this emphasises why PPP collapses chains all at once rather than one supernode at a time: the second last hyperarc in GTOPO30 would have needed $1,805,490$ iterations if processed



Y-coordinate indicates relative value between supernodes rather than absolute value

Fig. 4: Hyperstructure for the 50 × 100 subset of GTOPO30 near Vancouver in the supplementary material. Statistics are shown in Table 1. The thick curved arcs of the hyperstructure capture longer and longer chains of superarcs as the computation progresses.

		Vancouver: 5000 Samples			**GTOPO30: 922M Samples**			**Pawpawsaurus: 673M Samples**		
Iteration		Hyperarcs	Superarcs	Max Path	Hyperarcs	Superarcs	Max Path	Hyperarcs	Superarcs	Max Path
0	Upper	96	96	1	9,314,516	9,314,516	1	19,287,297	19,287,297	1
0	Lower	100	100	1	9,365,583	9,367,520	4	19,348,617	19,348,617	1
1	Upper	18	64	10	1,376,996	2,647,426	438	3,641,312	6,484,774	24
1	Lower	13	48	10	1,112,917	1,921,963	233	3,675,455	6,581,455	29
2	Upper	4	51	21	223,138	1,378,580	10,663	624,168	1,898,808	205
2	Lower	2	18	15	129,122	620,453	2,031	645,501	2,007,563	413
3	Upper	1	2	2	37,005	1,191,714	97,781	77,107	388,181	2,147
3	Lower	1	1	1	14,494	338,611	7,116	87,570	469,584	2,887
4	Upper				6,352	1,769,558	834,985	4,733	186,475	19,604
4	Lower				1,498	223,372	17,827	7,588	188,725	15,445
5	Upper				1,151	1,351,895	335,520	174	194,693	36,664
5	Lower				121	147,021	35,822	561	284,623	29,291
6	Upper				225	1,079,846	64,619	15	205,624	72,572
6	Lower				13	110,648	39,616	60	341,333	107,526
7	Upper				43	1,178,495	181,958	1	12,247,346	12,247,346
7	Lower				1	44,080	44,080	12	1,569,487	1,357,541
8	Upper				6	2,217,728	890,829	1	3,750,263	3,750,263
8	Lower				1	203,606	203,606	3	929,837	531,545
9	Upper				1	1,805,490	1,805,490	1	8,650	8,650
9	Lower				1	1	1	1	1	1
Total:		**235**	**380**	**21**	**21,583,184**	**36,912,523**	**1,805,490**	**47,400,177**	**76,373,336**	**12,247,346**

TABLE 1: Statistics for Hyperstructure On Three Example Datasets. As the data size grows, the hyperstructure collapses longer chains of superarcs into hyperarcs. Although it does not come with strong formal guarantees, it outperforms logarithmic collapse in practice.

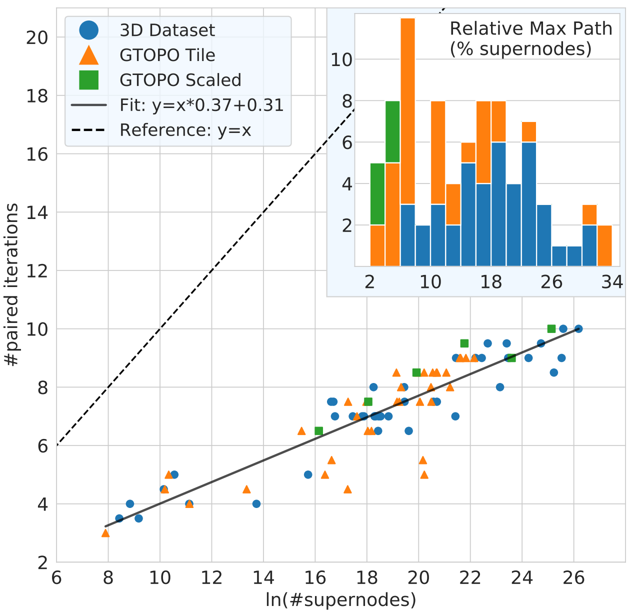


Fig. 5: Hyperstructure scaling: log-linear plot of contour tree size (x axis) against number of paired iterations in hyperstructure (y). Reference line (y=x) shows idealized logarithmic collapse for a perfectly balanced tree. In practice, the hyperstructure typically requires less than half as many iterations as the idealized collapse. **Inset:** Histogram of the relative maximum path length (y-axis) for all datasets, binned by logarithm of tree size. The longest hyperarc often captures 15% or more of the entire contour tree. See Supplement R for further details.

one at a time, causing a massive serial bottleneck.

5 Results & Performance Analysis

We have just described how to build a hyperstructure for efficient search in data-parallel contour trees. The key question is now the practical performance of PPP with hyperstructure and full augmentation added. We will describe our implementation and experiment design (Section 5.1 - 5.2), then discuss the results of our performance study in detail (Section 5.3 - 5.6).

5.1 Implementation

The hyperstructure computation is interwoven with the PPP algorithm rather than added as a post-process. As such, computing the hyperstructure with PPP required refactoring the original PPP code and cannot easily be separated out. We therefore elected to re-implement PPP with different detailed data structures (although primarily a superset of the original version), with support for 2D and 3D regular grids as well as arbitrary topology graphs. Our current version explicitly supports regular cubic lattices in both 2D and 3D, both by Freudenthal triangulation (2 triangles in 2D, 6 tetrahedra in 3D) and with the topology induced by Marching Cubes cases [9].

For other regular triangulations or for irregular meshes (including abstract simplicial complexes), we have abstracted the code to template on a Mesh type, and all that remains is to write appropriate classes. We have already implemented a class to handle arbitrary topology graphs, and do not anticipate problems in adding additional mesh types in the future.

Our algorithms can also be adapted to compute merge trees only, and we plan to expose this as a feature in the near future. On principle, there is no reason why the two merge trees should not be computed in parallel to each other, but we have not implemented that solution to date. We have released PPP as open source through the VTK-m library available at https://gitlab.kitware.com/vtk/vtk-m. Using VTK-m enables our algorithms to run a broad range of parallel compute architectures, from multi-core CPUs using TBB or OpenMP to many-core GPUs using CUDA.

5.2 Experiment Design

We compare the original PPP implementation *PPP1* with and without augmentation against our refactored implementation with hyperstructure *PPP2*, using *(aug)* for versions with augmentation and *(no aug)* for those without. As the core data structures and algorithm design are very similar for PPP1 and PPP2, we anticipate minimal impact from refactoring, but cannot isolate it due to tight integration of the hyperstructure into the merge tree computation. We also test how augmentation scales with data size and thread count. Finally, we compare performance with the algorithms in the open source TTK toolkit [16], [39], which supports contour tree computation from 2 and 3 simplicial complexes.

We therefore have three sets of performance tests: 1) Augmentation in PPP1 and PPP2 (Section 5.3), 2) Hyperstructure Scaling (Section 5.4), and 3) Comparison with TTK (Section 5.5). We start with details of the data and systems we used.

Data: GTOPO30 [1] is a global digital elevation model with a horizontal grid spacing of 30 arc seconds ($\approx 1km$) and a resolution of (21601×43201) pixel. To assess performance across data scales with similar topology, we rescaled by progressively reducing resolution in half, i.e., $G(1.0)$ to $G(0.03125)$ (Supplement A.2).

We also use 42 different 3D data sets from the visualization community for our scaling studies. Most of these come from the Open SciVis Dataset page (https://klacansky.com/open-scivis-datasets/), including many from the defunct VolVis page (http://volvis.org), such as the christmas tree dataset [21]. We also use a time step from the SciVis contest asteroid data set [33] available from the Los Alamos National Laboratory (https://dssdata.org). In addition we use the electric field and charge components of a single timestep of two 3D WarpX laser-driven, plasma-based particle accelerator simulations. Supplement A.1 describes these 3D data sets, covering many applications and sizes (from $68K$ to more than $934M$ data points) and complexity ($\mathrm{O}(10^2)$ to $\mathrm{O}(10^7)$ supernodes). Using this diverse collection of data allows us to assess performance across a broad range of scales and topologies.

Architecture: To evaluate performance across different compute architectures we used several systems at the National Energy Research Computing Center (NERSC):

- **Haswell:** One Haswell compute node of NERSC Cori with two 2.3 GHz 16-core Intel® Xeon ™ E5-2698 v3 (‘Haswell’) processors and 128 GB DDR4 2133 MHz memory. Each core supports two hyperthreads and has two 256-bit-wide vector units, supporting 32 physical threads and 64 hyperthreads.
- **Bigmem:** One large-memory login node of Cori with the same processor configuration as Haswell but 750 GB memory and no hyperthreading, i.e. with 32 threads.
- **KNL:** One KNL node of Cori with one 1.5Ghz 68-core Intel® Xeon Phi™ 7250 (‘Knights Landing’) processor with 272 hardware threads (4 per core), two 512-bit-wide vector processing units, and 96 GB DDR4 2400 MHz memory.
- **GPU:** One GPU node of Cori with two 2.4 GHz 20-core Intel Xeon Gold 6148 (‘Skylake’) CPUs, 384 GB DDR4 memory, 930 GB on-node NVMe storage, and 8 1.246 GHz NVIDIA V100 (‘Volta’) GPUs with 16 GB HBM2 memory each connected over NVLink. We used one shared GPU node with one GPU and 5 CPU cores allocated to the job.

Test Parameters: We measured compute time for PPP1 and PPP2 and their main sub-phases, and total time for TTK, using 1, 2, 4, 8, 16, 24, and 32 cores on Haswell and Bigmem, plus 64 threads with hyperthreading on Haswell, 1, 8, 16, 32, 64, 136, and 272 threads on KNL, and 1 full V100 plus 1 host core on GPU.

On shared compute systems like the Cori supercomputer, a primary source of variations in runtime performance is due variations in the overall state of the system and the often 100s of workloads running simultaneously on the system. To minimize the impact of such variations on our performance results, we repeated each test 5 times using the fastest run as reference.

In order to meet the memory needs of the algorithms, we tested the largest WarpX datasets (resolution of $6791 \times 371 \times 371$) on Bigmem, and therefore exclude these results when computing statistics to avoid mixing results from different systems in the box plots. We include the full results from all tests in the supplementary material.

5.3 Augmentation in PPP1 and PPP2

Our first set of tests focusses on the incremental cost of adding augmentation to the PPP algorithm, both for the original PPP1 and for our refactored PPP2. We consider the cost of adding augmentation to PPP1 first, then the cost when added to PPP2. We then compare PPP2 against PPP1 to see how much overhead was added to the underlying computation, and finally consider the overall cost of PPP2 (aug) vs PPP1 (no aug).

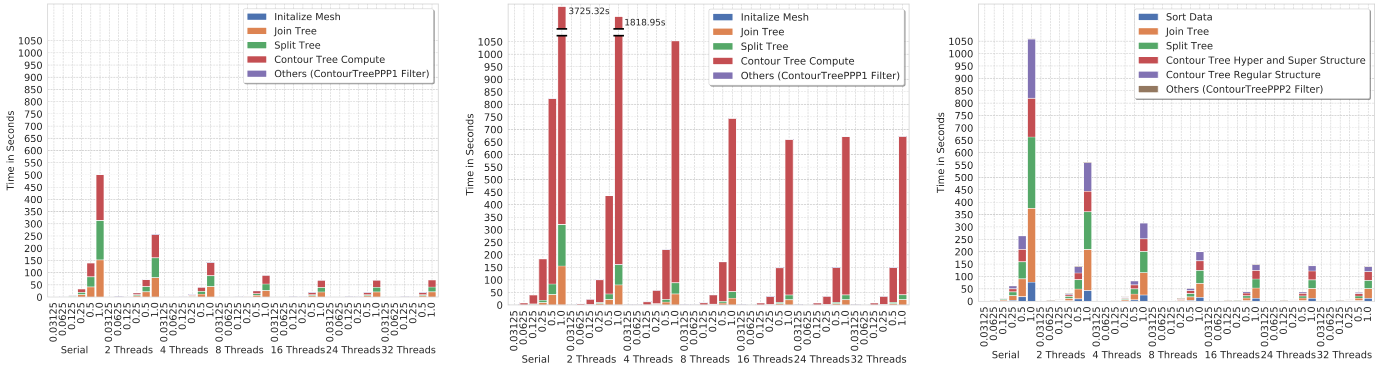
Augmentation in PPP1: Although PPP1 algorithm works with the supernodes of the data set, it can compute the fully augmented contour tree by including all regular nodes in the computation. Our first test is therefore to see the impact of this on PPP1, as shown in Figure 6(a,b). These plots and the box plot in Figure 7(a) show that PPP1 (no aug) took 500.18s for the full GTOPO30 in serial and 69.37s with 32 threads, whereas PPP1 (aug) took 3725.32s and 672.40s, a factor of 7.45× slower in serial and 9.66× slower with 32 threads. For the smallest version (0.03125), the overhead is 3.83× in serial, 2.53× with 32 threads. At medium scales, the slowdown ranges between $5.48 \times -5.92\times$ in serial and $5.11 \times -7.91\times$ with 32 threads. We conclude that augmenting PPP1 has a considerable impact on speed.

Augmentation in PPP2: Unlike PPP1, PPP2 always computes the hyperstructure, so the baseline performance is not the same, and may also be affected by refactoring. We can however measure the costs of the augmentation phase directly in a single run, and show both PPP2 (no aug) and PPP2 (aug) in Figure 6 (c). Here, the bars are broken down by phase: the topmost is augmentation, so the unaugmented cost consists of the lower portions of the bar. Here, it is obvious that the augmentation cost has gone from the dominant cost in the computation to a small additional cost.

For the full GTOPO30, the PPP2(aug) is 1.29× more expensive than PPP2(no aug) in serial and 1.17× with 32 threads. These are similar at all scales with serial cost for augmentation in the range of $1.22 \times -1.29\times$ and parallel cost in the range of $1.11 \times -1.17\times$ with 32 threads. We therefore see that PPP2 (aug) is an efficient way of adding augmentation to the contour tree, despite increasing the cost for computing the contour tree and the hyperstructure.

Hyperstructure Cost: If we compare the plots in Figure 6, it is clear that PPP1 (no aug) is the fastest, but that PPP1 (aug) is the slowest. Our next task is therefore to quantify the cost of moving to PPP2, some of which is due to adding the hyperstructure, but some of which may be due to the general refactoring performed. This is visible in Figure 7 (a), which shows that across all scales and thread counts, PPP2 (aug) runs $1.55 \times -2.25\times$ slower than PPP1 (no aug). If the augmented contour tree is not needed, we therefore recommend using PPP1 rather than PPP2.

Hyperstructure Advantage: Finally, we can consider what advantages we gain from using the hyperstructure to compute the augmentation, shown in Figure 7(b, c). Here, we can see that PPP2 (aug) is considerably faster than PPP1 (aug). In the serial

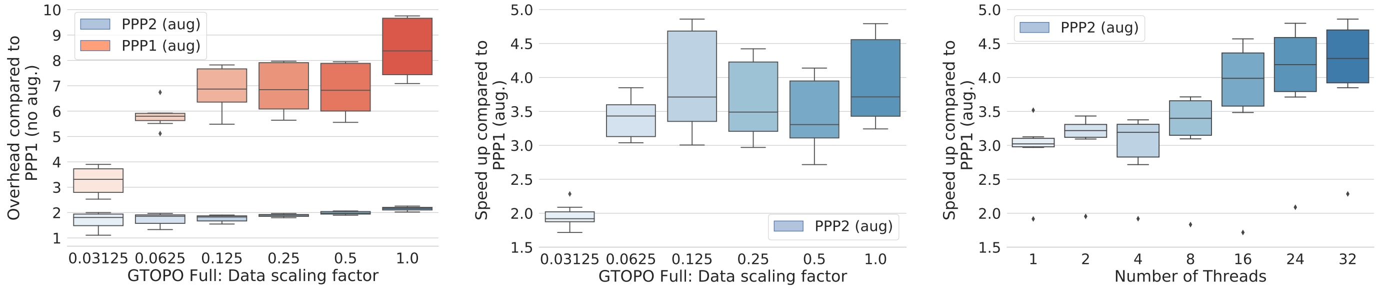


(a) Timings: PPP1 (no aug) Haswell

(b) Timings: PPP1 (aug) Haswell

(c) Timings: PPP2 (aug) Haswell

Fig. 6: Performance of (a) PPP1 (no aug), (b) PPP1 (aug), and (c) PPP2 (aug) on scaled GTOPO30 for varying numbers of threads on Bigmem. Removing the top segment from (c) gives the performance for PPP2 (no aug). See Fig. 7 for the corresponding speed ups.



(a) Overhead vs. PPP1 (no aug)

(b) Speed-up vs. PPP1(aug) by data scale

(c) Speed-up vs. PPP1 (aug) by threads

Fig. 7: **(a)** Overhead of PPP2 (aug) and PPP1 (aug) compared to PPP1 (no aug) on the scaled GTOPO datasets and varying numbers of threads grouped by dataset scale. See Fig. 6 for the corresponding timing results. **(b, c)** Speed-up of PPP2 (aug) compared to PPP1 (aug) on the scaled GTOPO datasets and varying numbers of threads grouped by dataset scale and number of threads, respectively.

implementation, PPP2 (aug) runs $1.92\times - 3.52\times$ faster than PPP1 (aug), while for 32 threads, PPP2 (aug) runs $2.28\times - 4.79\times$ faster than PPP1 (aug). We therefore recommend using PPP2 where the augmented contour tree is required.

5.4 Hyperstructure Scaling

We also want to ensure that the overall scaling demonstrated for PPP carries over, both with respect to the amount of parallelism available and with respect to the data size.

Thread Scaling In Figure 8, we show thread scaling for Haswell over all 3D data sets considered. Figure 8b shows PPP2 (aug) achieving median speedup of 12.48× with 64 threads, peaking at 16.89×. Equally, Figure 9 shows similar results for the many-core KNL system across all data sets. Using all 68 physical compute cores on KNL, we see a maximum speed-up of 42.33×, a median speed-up of 28.87×, and a minimum speed-up of 16.91×, with performance dropping off once hyperthreading kicks in.

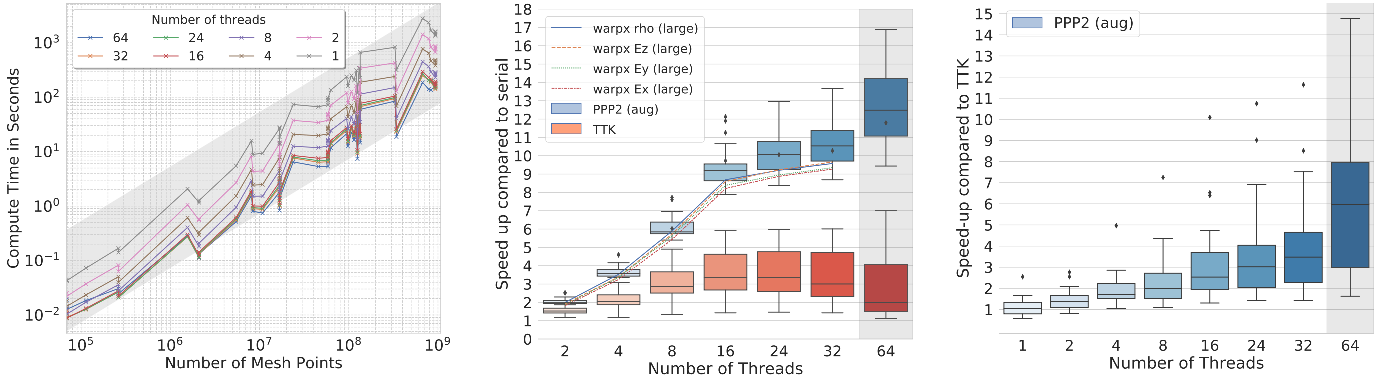
For GPUs, the picture is more nuanced. We tested PPP2 on GPU, comparing overall speedup against serial performance on KNL and Haswell. For medium-sized data with $118^3 < n < 684^3$ mesh points, average speed-ups were 23.35× and 104.07× and maximum speed-ups were 41.20× and 176.46× compared to serial Haswell and KNL, recognizing that clock rates differ.

A better comparison, however, involves the best execution time on each architecture, so Figure 10 compares GPU against Haswell with 64 threads and KNL with 68. Here, we observe maximum speed-ups of 3.00× and 5.55× and average speed-ups of 1.82× and 3.73×, for medium data (white area).

On GPU, data movement between the GPU and host memory is a major cost: we expect (and see) lower speed-ups for small datasets with $< 118^3$ points (tan area), likely due to data movement overhead and insufficient parallel workload. For large data files ($> 684^3$ points), the GPU runs out of memory, reducing performance.

Data Scaling: As with PPP1, we test scaling by taking GTOPO30 (our largest data set) and scaling it down, then considering how PPP2 scales as the mesh size increases (Figure 6c). Here, we see that between the half-resolution $G(0.5)$ and full-resolution $G(1.0)$, the cost increases by $3.80 - 4.90\times$, in line with the data increase of a factor of 4×. Similarly, between $G(0.25)$ and $G(0.5)$, the increase is between $4.26 - 4.70\times$ across all thread counts. This is slightly lower than PPP1 (no aug), whose scaling factor was $\approx 3.60 - 4.32\times$, but still quite reasonable.

Similarly, Figure 8a suggests that while the scaling for PPP2 (aug) may not be exactly linear with increasing data size, the scaling is still well behaved in practice. We expect large fluctuations in this plot, since the 3D datasets have greatly varying topological structure. We note, though, that the fluctuations are consistent between curves, indicating that PPP2 (aug) scales consistently well with increasing numbers of threads across the varying datasets. This observation is confirmed by the fact that the error bars in speed-ups in Figure 8b (blue boxplot) are well-behaved.



(a) Timings: PPP2 (aug) Haswell

(b) Speed up: PPP2 (aug) Haswell

(c) Speed up: PPP2 (aug) vs. TTK (aug)

Fig. 8: Scaling on Haswell for all 3D datasets. **(a)** PPP2 (aug) scaling against mesh size (log/log). Gray polygon indicates perfect weak scaling. **(b)** PPP2 (aug) scaling for thread count (blue) and TTK (aug) (red): WarpX using PPP2 (aug) on Bigmem. **(c)** PPP2 (aug) vs TTK by thread count. Gray area indicates hyperthreading (2 threads per core).

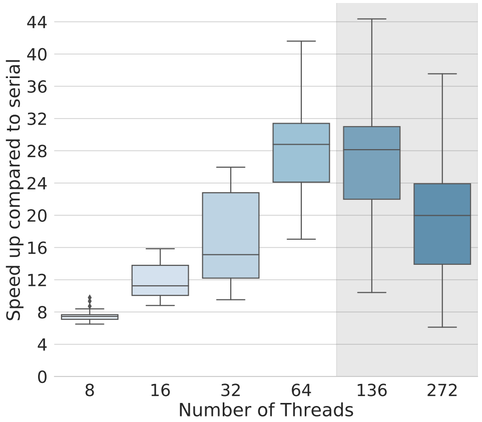


Fig. 9: Speed-up for PPP2 (aug) on KNL for the 3D datasets using varying numbers of threads. The gray area indicates the use of 2-4 threads per core (i.e., hyperthreading). See also Supplement M.

5.5 Comparison with TTK

Finally, we consider PPP2 (aug) against the augmented contour tree computation in TTK [39] (Figure 8b and 8c). For these benchmarks, we obtained the TTK source from GitHub (commit `008f0bcea116e8b5c46d2a73117ff30b248ccea0` as of 17:45:31 on August 8th 2019) and compiled it in `Release` mode, following discussions with Julien Tierny on the best compile options to use.

We saw significantly better scaling for PPP2 than for TTK, with consistently higher speedups at all thread counts on Haswell. Moreover, while the parallel efficiency drops off for both PPP2 and TTK, TTK peaks in efficiency at 16 threads, and drops significantly once hyperthreading kicks in. In contrast, PPP continues to take advantage of additional parallelism across all thread counts, outperforming TTK by a median factor of 5.96× and a maximum of 14.78×.

On the many-core KNL architecture—where efficient scaling is paramount—we saw further improved scaling of PPP2 compared to TTK (Figure 11a). In one extreme case, TTK took 1815.54s in serial and 444.19s with 16 threads for the (512 × 499 × 512) christmas tree dataset, but 7744.3s with 272

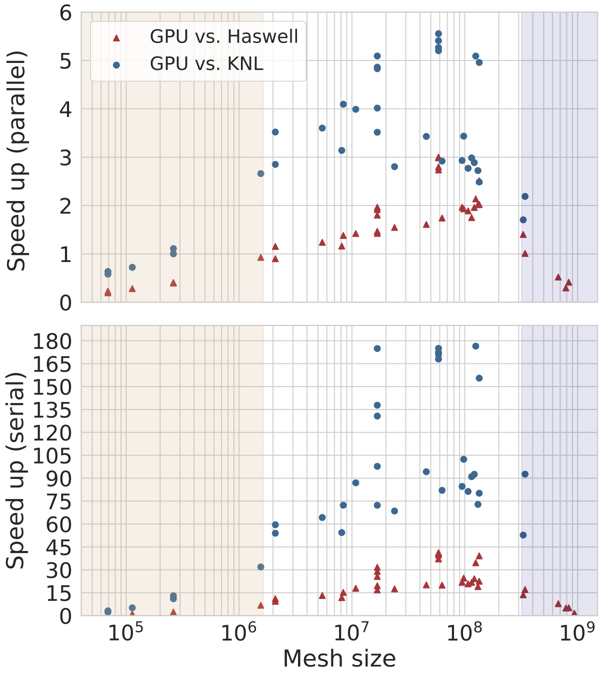


Fig. 10: Scatterplot showing the speed-up of PPP2 (aug) on GPU for all 3d datasets compared to Haswell and KNL in serial (bottom) and in parallel (top) using 64 threads on Haswell and 68 threads on KNL. The tan background indicates datasets smaller than 118^3 and the blue background indicates large datasets with more than 320 million (i.e., $\approx 684^3$). See Supplement N for details.

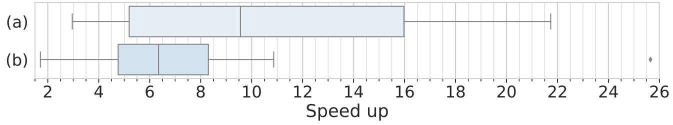


Fig. 11: (a) Speed-ups for PPP2 (aug) compared to TTK at 68 cores on KNL on 3D datasets. (b) Speed-ups for PPP2 (aug) on GPU compared to the best runtime for TTK on Haswell on 3D datasets . See also Supplements O and P for details.

threads (i.e., $\approx 4.27\times$ slower than serial). In Figure 11a we therefore compare TTK and PPP2 on KNL at 68 cores without hyperthreading, and see a median speed-up on KNL of 9.56× and a maximum speed-up of 21.74× across the 3D test datasets.

Since TTK does not run on GPUs, Figure 11(b) compares PPP2 (aug) on GPU against TTK on Haswell, with a median speed-up of 6.35× and a maximum speed-up of 25.65×.

5.6 Summary

PPP2 (aug) shows consistent scaling and good speed-ups with increasing numbers of threads and across varying data sizes on both Haswell and KNL, and further speedups on GPU. These results indicate that our algorithm is indeed able to utilize modern multi-core and many-core architectures. Finally, PPP2 (aug) has consistently shown significantly improved parallel scaling and runtime performance compared to the state-of-the-art parallel contour tree implementation available in TTK.

6 Conclusions & Future Work

We have extended parallel peak pruning to add full augmentation using hyperstructure to accelerate lookups. We expect to add more features to this computation such as branch decomposition, geometric measure computation, simplification and parallel isocontour extraction, for a full suite of data parallel analytic tools.

We also expect to build hybrid distributed-data parallel modules for architectures such as the Summit supercomputer at the Oak Ridge Leadership Computing Facility (OLCF). This will involve minimising network communication as well as processing time and memory lookup, but we are reasonably certain that our approach will scale with appropriate modifications.

Acknowledgments

We acknowledge EPSRC Grant EP/J013072/1 and the University of Leeds for the first author's study leave at Los Alamos National Laboratory. 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 under Contract No. DE-AC02-05CH11231 to the Lawrence Berkeley National Laboratory and under Award Number 14-017566 to the Los Alamos National Laboratory, and subcontract 7452335 to the University of Leeds.

This research used resources of the University of Leeds Advanced Research Computing facility and of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. We thank Jean-Luc Vay and Maxence Thevenet for making the WarpX dataset available to us. We would also like to thank Julien Tierny for his assistance in setting up the TTK runs for comparison.

References

[1] USGS 30 Arc-second Global Elevation Data, GTOPO30, 1997.

[2] A. Acharya and V. Natarajan. A Parallel and Memory Efficient Algorithm for Constructing the Contour Tree. In *Proceedings of the 2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 271–278, Apr. 2015.

[3] G. Blelloch. *Vector Models for Data-Parallel Computing*. PhD thesis, MIT, 1990.

[4] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. S. Day, and J. B. Bell. Interactive Exploration and Analysis of Large Scale Turbulent Combustion Using Topology-based Data Segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 17(9):1307–1324, 2011.

[5] H. Carr and J. Snoeyink. Representing Interpolant Topology for Contour Tree Computation. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 59–74. Springer, 2009.

[6] H. Carr, J. Snoeyink, and U. Axen. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003.

[7] H. Carr, J. Snoeyink, and M. van de Panne. Simplifying Flexible Isosurfaces with Local Geometric Measures. In *Proceedings of Visualization 2004*, pages 497–504, 2004.

[8] H. Carr, J. Snoeyink, and M. van de Panne. Flexible Isosurfaces: Simplifying and Displaying Scalar Topology Using the Contour Tree. *Computational Geometry: Theory and Applications*, 43(1):42–58, 2010.

[9] H. Carr, G. H. Weber, C. Sewell, O. Rübel, P. Fasel, and J. Ahrens. Scalable contour tree computation by data parallel peak pruning. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, November 2019.

[10] H. Carr, G. H. Weber, and J. Tierny. Pathological and Test Cases For Reeb Analysis. Accepted for publication., 2020.

[11] Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and Optimal Output-Sensitive Construction of Contour Trees Using Monotone Paths. *Computational Geometry: Theory and Applications*, 30:165–195, 2005.

[12] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological Persistence and Simplification. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 454–463. IEEE, 2000.

[13] H. Edelsbrunner and E. P. Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[14] J. Gibbons, W. Cai, and D. B. Skillicorn. Efficient parallel algorithms for tree accumulations. *Science of Computer Programming*, 23(1):1 – 18, 1994.

[15] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Contour forests: Fast multi-threaded augmented contour trees. In *6th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 85–92, Oct 2016.

[16] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-based Augmented Merge Trees with Fibonacci Heaps. In *7th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, 2017.

[17] F. Guo, H. Li, W. Daughton, and Y.-H. Liu. Formation of Hard Power Laws in the Energetic Particle Spectra Resulting from Relativistic Magnetic Reconnection. *Physics Review Letters*, 113:155005, Oct 2014.

[18] K. Heitmann, N. Frontiere, C. Sewell, S. Habib, A. Pope, H. Finkel, S. Rizzi, J. Insley, and S. Bhattacharya. The Q Continuum Simulation: Harnessing the Power of GPU Accelerated Supercomputers. To appear in the Astrophysical Journal Supplement., 2015.

[19] P. Hristov and H. Carr. W-Structures in Contour Trees. In submission.

[20] P. Hristov, G. H. Weber, H. Carr, O. Rübel, and J. Ahrens. Data parallel hypersweeps for in situ topological analysis. In *Proceedings of IEEE Large Data Analysis and Visualization*, 2020.

[21] A. Kanitsar, T. Theussl, L. Mroz, M. Sramek, A. V. Bartroli, B. Csebfalvi, J. Hladuvka, D. Fleischmann, M. Knapp, R. Wegenkittl, P. Felkel, S. Rottger, S. Guthe, W. Purgathofer, and M. E. Groller. Christmas tree case study: Computed tomography as a tool for mastering complex real world objects with applications in computer graphics. In *Proceedings of IEEE Visualization*, pages 489–492. IEEE, 2002.

[22] P. Klacansky, A. Gyulassy, P.-T. Bremer, and V. Pascucci. Toward Localized Topological Data Structures: Querying the Forest for the Tree. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):173–183, 2019.

[23] A. G. Landge, V. Pascucci, A. Gyulassy, J. C. Bennett, H. Kolla, J. Chen, and P. T. Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031, Nov. 2014.

[24] L.-T. Lo, C. Sewell, and J. Ahrens. PISTON: A Portable Cross-Platform Framework for Data-Parallel Visualization Operators. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 11–20, 2012.

[25] S. Maadasamy, H. Doraiswamy, and V. Natarajan. A hybrid parallel algorithm for computing and tracking level set topology. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, Dec. 2012.

[26] G. L. Miller and J. H. Reif. Parallel tree contraction part 1: Fundamentals. In S. Micali, editor, *Randomness and Computation*, pages 47–72. JAI Press, Greenwich, Connecticut, 1989. Vol. 5.

[27] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. Ma, H. Childs, M. Larsen, C. Chen, R. Maynard, and B. Geveci. VTK-m: Accelerating the visualization toolkit

for massively threaded architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, 2016. http://m.vtk.org/.

[28] D. Morozov and G. Weber. Distributed Merge Trees. *ACM SIGPLAN Notices*, 48(8):93–102, 2013.

[29] D. Morozov and G. Weber. Distributed Contour Trees. In P.-T. Bremer, I. Hotz, V. Pascucci, and R. Peikert, editors, *Topological Methods in Data Analysis and Visualization III*, Mathematics and Visualization, pages 89–102. Springer, 2014.

[30] V. Pascucci and K. Cole-McLaughlin. Parallel Computation of the Topology of Level Sets. *Algorithmica*, 38(2):249–268, 2003.

[31] V. Pascucci, K. Cole-McLaughlin, and G. Scorzell. Multi-resolution computation and presentation of contour trees. In *Proceedings of the IASTED conference on Visualization, Imaging and Image Processing (VIIP 2004)*, pages 452–290, 2004.

[32] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust On-line Computation of Reeb Graphs: Simplicity and Speed. *ACM Transactions on Graphics*, 26(3):58.1–58.9, 2007.

[33] J. Patchett and G. Gisler. Deep water impact ensemble data set. Technical Report LA-UR-17-21595, Los Alamos National Laboratory, 2017.

[34] G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de l'Acadèmie des Sciences de Paris*, 222:847–849, 1946.

[35] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit*. Kitware, 4th edition, 2006. https://vtk.org/.

[36] C. Sewell, L.-T. Lo, and J. Ahrens. Portable Data-Parallel Visualization and Analysis in Distributed Memory Environments. In *Proceedings of the IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, pages 25–33, 2013.

[37] D. Smirnov and D. Morozov. *Triplet Merge Trees*, volume V of *Topological Methods in Data Analysis and Visualization*, pages 19–36. Springer, 2020.

[38] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22:215–225, 1975.

[39] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2018. https://topology-tool-kit.github.io/.

[40] G. Weber, S. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-Controlled Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):330–341, March/April 2007.

[41] W. Widanagamaachchi, C. Christensen, P.-T. Bremer, and V. Pascucci. Interactive Exploration of Large-Scale Time-Varying Data Using Dynamic Tracking Graphs. In *2d IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 9–17, 2012.

Hamish Carr is a Professor in the School of Computing at the University of Leeds, having earned his PhD from the University of British Columbia in 2004. His research centers on computational topology and the mathematical foundations of scientific visualization, but also includes applications ranging from civil engineering to environmental sciences. He is a member of the IEEE, ACM and Eurographics, and currently serves as Corporate Secretary for Eurographics.

Oliver Rübel received the MS degree in computer science in 2006 and the PhD degree in computer science in 2009 from the University of Kaiserslautern, Germany. He is a Staff Scientist at Lawrence Berkeley National Laboratory. His research interests include high-performance data analysis and visualization, scientific data management, machine learning and query-driven visualization.

Gunther H. Weber is a Staff Scientist in the Computational Research Division at Lawrence Berkeley National Laboratory and an Adjunct Associate Professor of Computer Science at UC Davis. He earned his Ph.D. in computer science from the University of Kaiserslautern, Germany in 2003 and completed his post-doc in 2006 at UC Davis. His research interests include parallel algorithms, visualization, data analysis, machine learning and computer graphics.

James Ahrens is a senior scientist in the Applied Computer Science Group at Los Alamos National Laboratory. His primary research interests are visualization, computer graphics, data science and parallel systems. Dr. Ahrens is author of over 100 peer reviewed papers and the founder/design lead of ParaView, an open-source visualization tool designed to handle extremely large data. ParaView is broadly used for scientific visualization and is in use at supercomputing and scientific centers worldwide. Dr. Ahrens is the Chair of the IEEE Computer Society Technical Committee on Visualization and Graphics (VGTC). Dr. Ahrens received his B.S. in Computer Science from the University of Massachusetts at Amherst in 1989 and a Ph.D. in Computer Science from the University of Washington in 1996.

SUPPLEMENT A
DATASETS OVERVIEW

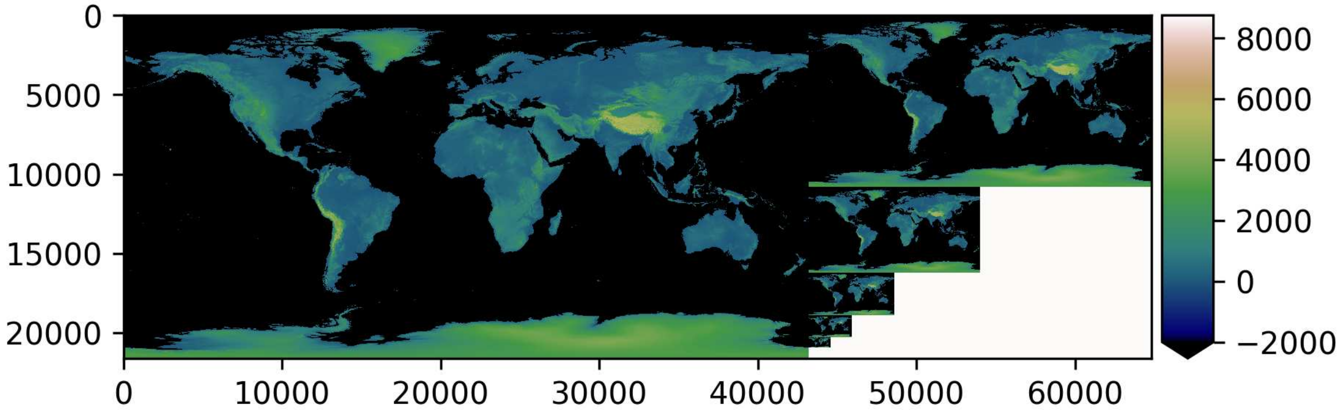
A.1 Shape and size of the 3D datasets

name	source	shape	# nodes	# supernodes
marschner lobb	Marschner and Lobb	[41, 41, 41]	68,921	1,506
nucleon	SFB 382 of the German Research Council (DFG)	[41, 41, 41]	68,921	579
silicium	VolVis distribution of SUNY Stony Brook, NY, USA	[98, 34, 34]	113,288	458
neghip	VolVis distribution of SUNY Stony Brook, NY, USA	[64, 64, 64]	262,144	2,242
fuel	SFB 382 of the German Research Council (DFG)	[64, 64, 64]	262,144	344
tooth	TBD	[103, 94, 161]	1,558,802	231,242
shockwave	TBD	[64, 64, 512]	2,097,152	1,133
hydrogen atom	SFB 382 of the German Research Council (DFG)	[128, 128, 128]	2,097,152	13,593
lobster	VolVis distribution of SUNY Stony Brook, NY, USA	[301, 324, 56]	5,461,344	323,349
mri ventricles	Dirk Bartz, VCM, University of Tbingen, Germany	[256, 256, 124]	8,126,464	1,562,438
engine	General Electric	[256, 256, 128]	8,388,608	467,702
statue leg	German Federal Institute for Material Research and Testing (BAM), Berlin, Germany	[341, 341, 93]	10,814,133	353,877
tacc turbulence	Gregory D. Abram and Gregory P. Johnson, Texas Advanced Computing Center, The University of Texas at Austin. Simulation by Diego A. Donzis, Texas A&M University, P.K. Yeung, Georgia Tech	[256, 256, 256]	16,777,216	313,281
aneurism	Philips Research, Hamburg, Germany	[256, 256, 256]	16,777,216	54,197
bonsai	S. Roettger, VIS, University of Stuttgart	[256, 256, 256]	16,777,216	179,627
skull	Siemens Medical Solutions, Forchheim, Germany	[256, 256, 256]	16,777,216	1,710,477
foot	Philips Research, Hamburg, Germany	[256, 256, 256]	16,777,216	719,091
mrt angio	Özlem Gürvit, Institute for Neuroradiology, Frankfurt, Germany	[416, 512, 112]	23,855,104	4,818,339
stent	Michael Meißner, Viatronix Inc., USA	[512, 512, 174]	45,613,056	2,856,825
warpx small Ez	WarpX collaboration	[425, 371, 371]	58,497,425	111,396
warpx small Ex	WarpX collaboration	[425, 371, 371]	58,497,425	358,203
warpx small rho	WarpX collaboration	[425, 371, 371]	58,497,425	106,908
warpx small Ey	WarpX collaboration	[425, 371, 371]	58,497,425	100,467
pancreas	Roth HR, Lu L, Farag A, Shin H-C, Liu J, Turkbey EB, Summers RM. DeepOrgan: Multi-level Deep Convolutional Networks for Automated Pancreas Segmentation	[240, 512, 512]	62,914,560	6,682,631
bunny	Stanford Radiology & Computer Science Departments	[512, 512, 361]	94,633,984	11,110,783
backpack	Kevin Kreeger, Viatronix Inc., USA	[512, 512, 373]	97,779,712	5,693,268
present	Christoph Heinzl, 2006	[492, 492, 442]	106,992,288	11,547,958
neocortical layer 1 axons	V De Paola, MRC Clinical Sciences Center, Imperial College London	[1464, 1033, 76]	114,935,712	9,289,314
prone	Walter Reed Army Medical Center, USA	[512, 512, 463]	121,372,672	12,087,883
asteroid	John Patchett and Galen Gisler, Los Alamos National Laboratory [33]	[500, 500, 500]	1250,00,000	804,757
christmas tree	Armin Kanitsar, 2002	[512, 499, 512]	130,809,856	19,962,839
vertebra	Michael Meißner, Viatronix Inc., USA	[512, 512, 512]	134,217,728	2,808,594
magnetic reconnection	Bill Daughton (LANL) and Berk Geveci (KitWare) [17]	[512, 512, 512]	134,217,728	27,860,405
marmoset neurons	Frederick Federer, Moran Eye Institute, University of Utah	[1024, 1024, 314]	329,252,864	48,399,592
stag beetle	Meister Eduard GrÏler, Georg Glaeser, Johannes Kastner, 2005	[832, 832, 494]	341,958,656	712,098
pawpawsaurus	Matthew Colbert, 4 February 2014	[958, 646, 1088]	673,328,384	76,373,336
spathorhynchus	Matthew Colbert, 17 February 2005	[1024, 1024, 750]	786,432,000	39,376,047
kingsnake	DigiMorph.org, The University of Texas High-Resolution X-ray CT Facility (UTCT), and NSF grant IIS-9874781	[1024, 1024, 795]	833,617,920	50,552,413
warpx large Ey	WarpX collaboration	[6791, 371, 371]	934,720,031	330,912
warpx large rho	WarpX collaboration	[6791, 371, 371]	934,720,031	322,028
warpx large Ez	WarpX collaboration	[6791, 371, 371]	934,720,031	378,067
warpx large Ex	WarpX collaboration	[6791, 371, 371]	934,720,031	242,442

TABLE S1: Overview of the shape and size of the 3D datasets used in the performance evaluation with datasets sorted by size. Most data sets are from the Open SciVis Dataset page (https://klacansky.com/open-scivis-datasets/). The WarpX data sets are courtesy of collaborators at Lawrence Berkeley National Laboratory and not publically available. The asteroid data set is available from the Los Alamos National Laboratory (https://dssdata.org). Gray color is used to indicate data sets excluded from summary statistics in the main manuscript.

A.2 Shape and size of the GTOPO datasets

The GTOPO30 [1] dataset is a global digital elevation model with a horizontal grid spacing of 30 arc seconds ($\approx 1km$). The dataset consists of 34 tiles; 27 at 6000×4800 pixel, 6 at 3600×4800 pixel, and 1 at 5400×5400 pixel. When combining the tiles, the full dataset consists of (21601×43201) pixel. We rescaled the full image by progressively reducing resolution by half, i.e., $G(1.0)$ to $G(0.03125)$ (Figure S1). Using the scaled GTOPO30 dataset allows us to assess the performance of our algorithms across a broad range of data scales with similar overall topology.



Scale	1.0	0.5	0.25	0.125	0.0625	0.03125
# Pixel x	21601	10800	5400	2700	1350	675
# Pixel y	43201	21600	10400	5400	2700	1350
Complexity	3.96%	5.44%	6.14%	6.8%	7.46%	7.93%

Fig. S1: Map of the full GTOPO dataset at various levels of resolution and table listing for each dataset its spatial resolution $(nx \times ny)$, number of supernodes s in the contour tree, and relative topological complexity $s/(nx * ny)$. Figure courtesy of [9].

SUPPLEMENT B
PPP2 (AUG.): GTOPO SCALED DATASETS ON BIGMEM

		Sort Data	Join Tree	Split Tree	Contour Tree	Contour Tree	Others
					Hyper and Super Structure	Regular Structure	VTKm Filter
# Threads	Data Scale						
1	0.03125	0.048630	0.207546	0.174923	0.087610	0.116136	0.000054
	0.06250	0.208212	0.943393	0.788297	0.383193	0.501342	0.000066
	0.12500	0.945953	4.230829	3.695311	1.778210	2.402060	0.000191
	0.25000	3.960810	18.555830	15.423140	12.402000	11.443600	0.000107
	0.50000	18.391400	72.203700	68.914100	50.245800	53.550400	0.000111
	1.00000	77.016400	298.479700	287.697700	156.281000	239.435000	0.000130
2	0.03125	0.026518	0.109904	0.098011	0.055616	0.058266	0.000050
	0.06250	0.109869	0.458176	0.410026	0.214655	0.255682	0.000067
	0.12500	0.519750	2.137011	1.985819	0.912534	1.151560	0.000148
	0.25000	2.208060	9.159630	8.371400	6.304780	5.468490	0.000134
	0.50000	9.784850	39.250110	38.179930	26.645400	26.911400	0.000136
	1.00000	43.636000	165.719300	152.017300	83.078400	116.565000	0.000233
4	0.03125	0.016977	0.065250	0.057854	0.045043	0.034323	0.000053
	0.06250	0.066391	0.262750	0.235714	0.137049	0.143752	0.000066
	0.12500	0.290299	1.203035	1.064462	0.567707	0.652800	0.000124
	0.25000	1.310450	5.158769	4.742222	3.682770	3.141820	0.000159
	0.50000	5.730630	23.007820	21.500300	15.882300	15.334000	0.000161
	1.00000	25.128900	90.958500	85.396700	50.666500	63.222300	0.000180
8	0.03125	0.012228	0.042967	0.038067	0.039583	0.020206	0.000054
	0.06250	0.044788	0.164449	0.146370	0.107772	0.088423	0.000075
	0.12500	0.185068	0.714842	0.669096	0.390392	0.393709	0.000130
	0.25000	0.777907	3.204646	2.989298	2.571620	1.857070	0.000125
	0.50000	3.542380	14.254480	13.456520	11.678600	9.055280	0.000121
	1.00000	15.930600	56.241650	52.697180	38.672200	36.815400	0.000134
16	0.03125	0.009786	0.031520	0.026727	0.037708	0.013078	0.000055
	0.06250	0.033263	0.103712	0.091446	0.087573	0.058305	0.000070
	0.12500	0.130710	0.501700	0.456125	0.282293	0.254270	0.000108
	0.25000	0.596698	2.278978	2.053212	2.150790	1.187160	0.000152
	0.50000	2.580320	10.312600	9.324330	10.428900	5.781710	0.000125
	1.00000	12.107800	41.030330	36.389440	35.205700	23.781500	0.000122
24	0.03125	0.009071	0.030089	0.023525	0.038087	0.011330	0.000056
	0.06250	0.029400	0.095509	0.083493	0.091515	0.053406	0.000070
	0.12500	0.138033	0.486331	0.424532	0.289395	0.229786	0.000102
	0.25000	0.563995	2.212921	1.909693	2.130750	1.076820	0.000137
	0.50000	2.742440	9.905690	8.759190	10.550800	5.186070	0.000120
	1.00000	12.044000	39.687610	34.671500	35.616900	21.624200	0.000121
32	0.03125	0.008698	0.029737	0.022863	0.038247	0.011206	0.000054
	0.06250	0.029856	0.091014	0.079083	0.092945	0.050020	0.000090
	0.12500	0.136423	0.496860	0.406772	0.293082	0.211175	0.000148
	0.25000	0.544278	2.181621	1.833596	2.136790	1.018630	0.000157
	0.50000	2.612600	9.625790	8.474760	10.522400	4.907740	0.000168
	1.00000	11.950700	39.046080	33.299630	35.680300	20.339000	0.000136

TABLE S2: PPP2 (aug.) runtime in seconds on Bigmem for all GTOPO scaled datasets. We repeated each evaluation 5 times and report here the best time.

Supplement C
PPP1 (aug.): GTOPO scaled datasets on Bigmem

# Threads	Data Scale	Initalize Mesh	Join Tree	Split Tree	Contour Tree Compute	Others (VTKm Filter)
1	0.03125	0.000005	0.100575	0.094306	1.021700	0.000095
	0.06250	0.000003	0.407041	0.383845	7.788960	0.000092
	0.12500	0.000003	2.089010	2.017418	35.119300	0.000095
	0.25000	0.000002	9.635480	9.383570	164.435000	0.000099
	0.50000	0.000003	41.686530	41.692720	739.475000	0.000091
	1.00000	0.000003	155.148300	166.514100	3403.660000	0.000148
2	0.03125	0.000007	0.056720	0.051478	0.572263	0.000079
	0.06250	0.000002	0.221375	0.206460	4.543540	0.000155
	0.12500	0.000003	1.145895	1.106653	20.080800	0.000135
	0.25000	0.000003	5.167188	5.008365	90.430900	0.000123
	0.50000	0.000003	21.909340	21.829680	391.718000	0.000155
	1.00000	0.000003	79.755100	81.975400	1657.220000	0.000216
4	0.03125	0.000002	0.036816	0.032898	0.351527	0.000063
	0.06250	0.000002	0.137344	0.126003	2.411640	0.000118
	0.12500	0.000002	0.618527	0.591995	11.543800	0.000126
	0.25000	0.000003	2.844511	2.700998	52.539700	0.000145
	0.50000	0.000002	11.346910	11.129190	198.775000	0.000085
	1.00000	0.000003	43.991320	44.724610	964.559000	0.000205
8	0.03125	0.000002	0.026167	0.023480	0.230922	0.000069
	0.06250	0.000002	0.091118	0.082178	1.535020	0.000128
	0.12500	0.000003	0.398153	0.384114	7.953110	0.000123
	0.25000	0.000003	1.733624	1.627038	36.428700	0.000143
	0.50000	0.000003	7.504660	7.154560	157.207000	0.000137
	1.00000	0.000003	27.072280	26.411040	690.607000	0.000212
16	0.03125	0.000003	0.021013	0.017789	0.165181	0.000076
	0.06250	0.000002	0.074256	0.063990	1.165910	0.000100
	0.12500	0.000003	0.305498	0.284500	6.834640	0.000123
	0.25000	0.000003	1.284466	1.179640	31.533300	0.000141
	0.50000	0.000003	5.469318	5.149704	137.804000	0.000137
	1.00000	0.000003	20.895730	19.005430	620.055000	0.000138
24	0.03125	0.000002	0.027949	0.023425	0.182752	0.000078
	0.06250	0.000002	0.079312	0.066255	1.166130	0.000097
	0.12500	0.000003	0.319575	0.299866	6.904570	0.000126
	0.25000	0.000003	1.304322	1.206192	31.772800	0.000147
	0.50000	0.000003	5.487210	5.157788	139.217000	0.000148
	1.00000	0.000003	21.080570	19.162100	630.622000	0.000158
32	0.03125	0.000002	0.034123	0.030086	0.188815	0.000098
	0.06250	0.000003	0.087637	0.077572	1.154750	0.000126
	0.12500	0.000003	0.329175	0.312924	6.862900	0.000168
	0.25000	0.000003	1.308381	1.214739	31.592000	0.000128
	0.50000	0.000003	5.507190	5.169020	138.891000	0.000201
	1.00000	0.000003	21.533150	19.342320	631.533000	0.000197

TABLE S3: PPP1 (aug.) runtime in seconds on Bigmem for all GTOPO scaled datasets. We repeated each evaluation 5 times and report here the best time.

SUPPLEMENT D
PPP1 (NO AUG.): GTOPO SCALED DATASETS ON BIGMEM

		Initalize Mesh	Join Tree	Split Tree	Contour Tree Compute	Others (VTKm Filter)
# Threads	Data Scale					
1	0.03125	0.000002	0.102815	0.096248	0.118469	0.000039
	0.06250	0.000003	0.444308	0.412492	0.591270	0.000062
	0.12500	0.000003	2.309496	2.326335	2.515640	0.000075
	0.25000	0.000003	9.936340	9.434210	13.144900	0.000087
	0.50000	0.000002	41.562140	41.517730	55.841900	0.000083
	1.00000	0.000002	152.172700	161.831300	186.175000	0.000083
2	0.03125	0.000003	0.056382	0.051353	0.066727	0.000043
	0.06250	0.000002	0.224634	0.210459	0.302244	0.000054
	0.12500	0.000003	1.114092	1.057573	1.354840	0.000085
	0.25000	0.000003	5.234457	4.933360	6.731400	0.000102
	0.50000	0.000003	21.761690	21.210900	28.569100	0.000118
	1.00000	0.000003	79.764500	81.119500	95.768500	0.000114
4	0.03125	0.000002	0.036262	0.033089	0.046967	0.000047
	0.06250	0.000003	0.137820	0.126286	0.189270	0.000057
	0.12500	0.000003	0.641808	0.597046	0.759918	0.000074
	0.25000	0.000003	2.839331	2.658884	3.846090	0.000101
	0.50000	0.000003	11.893560	11.642540	16.277100	0.000099
	1.00000	0.000003	43.318450	44.309400	54.196200	0.000104
8	0.03125	0.000002	0.024833	0.022069	0.037802	0.000051
	0.06250	0.000003	0.088441	0.081488	0.127415	0.000062
	0.12500	0.000003	0.393046	0.371047	0.507293	0.000086
	0.25000	0.000003	1.723394	1.616853	2.471840	0.000099
	0.50000	0.000003	7.368170	7.184440	10.643000	0.000092
	1.00000	0.000003	27.016650	26.308310	35.523000	0.000101
16	0.03125	0.000002	0.021759	0.018441	0.032648	0.000053
	0.06250	0.000003	0.068258	0.059885	0.096656	0.000060
	0.12500	0.000003	0.301079	0.283091	0.364978	0.000077
	0.25000	0.000003	1.283871	1.183246	1.851400	0.000096
	0.50000	0.000003	5.478969	5.156219	8.266200	0.000111
	1.00000	0.000003	20.729370	18.954340	28.828100	0.000097
24	0.03125	0.000002	0.027352	0.023455	0.032916	0.000054
	0.06250	0.000003	0.075671	0.065950	0.096192	0.000062
	0.12500	0.000003	0.308340	0.295856	0.359345	0.000082
	0.25000	0.000003	1.298662	1.196502	1.807200	0.000132
	0.50000	0.000003	5.480809	5.167340	8.211130	0.000099
	1.00000	0.000003	21.021010	19.130230	28.623700	0.000104
32	0.03125	0.000002	0.034054	0.030508	0.035395	0.000064
	0.06250	0.000003	0.085976	0.073355	0.098607	0.000075
	0.12500	0.000002	0.318661	0.315507	0.363226	0.000107
	0.25000	0.000003	1.299428	1.212303	1.780190	0.000134
	0.50000	0.000003	5.495380	5.172770	8.231250	0.000128
	1.00000	0.000002	21.510680	19.335010	28.528200	0.000136

TABLE S4: PPP1 (without augmentation) runtime in seconds on Bigmem for all GTOPO scaled datasets. We repeated each evaluation 5 times and report here the best time.

SUPPLEMENT E
PPP2 (AUG.): 3D DATASETS ON HASWELL

E.1 Timings: PPP2 (aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	0.019187	0.011769	0.010659	0.010856	0.013227	0.017858	0.025205	0.044322
nucleon	0.012679	0.009101	0.008827	0.008746	0.010345	0.014426	0.021983	0.042582
silicium	0.018394	0.012641	0.012780	0.013212	0.017135	0.023370	0.037345	0.072624
neghip	0.030582	0.026697	0.026378	0.027768	0.035993	0.050449	0.082300	0.168537
fuel	0.024609	0.021052	0.020963	0.022316	0.027895	0.039988	0.063270	0.141167
tooth	0.282559	0.273261	0.277170	0.299204	0.405078	0.609764	1.053600	2.070850
shockwave	0.111137	0.113883	0.116075	0.124427	0.181934	0.297688	0.553402	1.161640
hydrogen	0.131877	0.132082	0.136419	0.143512	0.208174	0.329792	0.582974	1.264610
lobster	0.516682	0.555891	0.582303	0.616003	0.945686	1.537280	2.698670	5.489420
mri ventricles	1.559560	1.715060	1.803890	1.927610	2.932370	4.707160	8.483680	15.830300
engine	0.804868	0.897163	0.936940	1.006380	1.518340	2.424080	4.346340	8.867550
statue leg	0.741967	0.866883	0.909443	0.993966	1.524920	2.477990	4.415470	9.345340
tacc turbulence	1.807190	2.235020	2.427950	2.757360	4.498750	7.797220	14.122400	26.666200
aneurism	0.831651	1.013960	1.066860	1.155160	1.808580	2.987330	5.425620	13.744700
bonsai	1.027380	1.245880	1.327040	1.428450	2.214260	3.638920	6.595480	14.636500
skull	2.356190	2.699400	2.870470	3.141280	4.858050	7.969280	14.265700	27.884100
foot	1.389460	1.633440	1.724670	1.860710	2.911700	4.840200	8.615040	18.566200
mrt angio	6.471770	7.626220	7.962700	8.497830	12.544600	20.703300	37.448500	73.334900
stent	5.325230	6.383460	6.805380	7.550070	11.849200	19.735000	34.320200	66.669200
warpx rho (small)	5.354010	6.527730	7.003840	7.808880	12.663700	21.197200	37.725100	72.694800
warpx Ez (small)	6.466830	7.963830	8.513250	9.460350	15.299800	25.871100	46.159800	87.888300
warpx Ey (small)	6.501510	8.040200	8.757870	9.811780	15.774100	26.757800	47.368800	89.664700
warpx Ex (small)	6.717990	8.263380	8.856520	10.052400	16.271500	27.783500	49.371700	95.379600
pancreas	11.790400	13.879800	14.642800	15.961100	24.480200	40.167800	70.989000	135.096000
bunny	21.316300	24.536600	25.813000	27.608700	40.686500	66.938400	120.183000	236.376000
backpack	12.520400	14.858400	15.707700	17.182600	26.030300	43.750000	78.192300	159.306000
present	22.685700	26.376200	27.600200	29.740100	43.414700	70.661800	126.933000	250.069000
neocort. layer	16.401500	19.348000	20.193700	21.521000	31.875700	51.892900	92.625200	204.158000
prone	25.150900	29.483200	31.613100	34.499800	53.187600	89.298600	160.691000	310.074000
asteroid	7.373470	9.210960	9.842410	10.618200	17.150600	28.733600	52.093300	119.469000
christmas tree	35.961200	39.073600	40.324900	42.336700	59.752100	94.397700	166.871000	339.325000
vertebra	14.641500	17.943200	19.215900	21.424000	35.118600	60.677100	112.848000	228.179000
mag. reconnection	59.661200	66.813100	70.408800	77.027700	113.828000	188.167000	340.721000	664.827000
marmoset neurons	84.569200	94.431700	98.481100	104.685000	150.220000	240.152000	426.093000	824.224000
stag beetle	18.681000	23.072900	24.363200	26.033300	40.847900	68.783000	126.085000	315.585000
pawpawsaurus	185.781000	251.325000	268.867000	293.805000	446.418000	769.598000	1414.310000	2793.680000
spathorhynchus	139.926000	205.566000	204.218000	230.328000	364.463000	643.461000	1185.180000	2346.190000
kingsnake	134.797000	188.818000	197.556000	210.112000	292.977000	462.468000	827.277000	1675.660000
warpx rho (large)*	—	140.703000	146.385000	155.079000	228.759000	381.784000	674.742000	1347.350000
warpx Ez (large)*	—	162.013000	169.519000	182.229000	273.583000	462.352000	818.371000	1564.750000
warpx Ey (large)*	—	173.035000	180.686000	192.892000	287.576000	481.855000	865.744000	1617.240000
warpx Ex (large)*	—	153.269000	160.034000	172.977000	261.810000	437.770000	777.557000	1420.540000

TABLE S5: PPP2 (aug.) runtime in seconds on Haswell for all 3D datasets. We repeated each evaluation 5 times and report here the best time. Datasets marked with * were evaluated on the Bigmem system.

E.2 Speed up: PPP2 (aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	2.310026	3.766059	4.158176	4.082907	3.351000	2.481885	1.758440	1.0
nucleon	3.358467	4.679048	4.824276	4.868746	4.116191	2.951815	1.937033	1.0
silicium	3.948244	5.744933	5.682585	5.496904	4.238343	3.107521	1.944668	1.0
neghip	5.510915	6.313027	6.389302	6.069447	4.682494	3.340740	2.047842	1.0
fuel	5.736467	6.705570	6.734039	6.325735	5.060602	3.530234	2.231170	1.0
tooth	7.328912	7.578286	7.471407	6.921198	5.112225	3.396150	1.965499	1.0
shockwave	10.452325	10.200293	10.007667	9.335916	6.384953	3.902206	2.099089	1.0
hydrogen	9.589314	9.574431	9.270043	8.811876	6.074774	3.834568	2.169239	1.0
lobster	10.624369	9.874993	9.427085	8.911353	5.804696	3.570865	2.034121	1.0
mri ventricles	10.150491	9.230173	8.775646	8.212398	5.398466	3.363026	1.865971	1.0
engine	11.017397	9.883990	9.464373	8.811334	5.840293	3.658109	2.040234	1.0
statue leg	12.595358	10.780394	10.275894	9.402072	6.128413	3.771339	2.116499	1.0
tacc turbulence	14.755615	11.931079	10.983010	9.670917	5.927469	3.419962	1.888220	1.0
aneurism	16.527005	13.555466	12.883321	11.898525	7.599719	4.600998	2.533296	1.0
bonsai	14.246433	11.747921	11.029434	10.246421	6.610109	4.022210	2.219171	1.0
skull	11.834402	10.329740	9.714123	8.876668	5.739772	3.498948	1.954625	1.0
foot	13.362169	11.366319	10.765074	9.978019	6.376412	3.835833	2.155092	1.0
mrt angio	11.331506	9.616153	9.209803	8.629838	5.845934	3.542184	1.958287	1.0
stent	12.519497	10.444054	9.796543	8.830276	5.626473	3.378221	1.942564	1.0
warpx rho (small)	13.577636	11.136306	10.379278	9.309248	5.740408	3.429453	1.926961	1.0
warpx Ez (small)	13.590631	11.035934	10.323707	9.290174	5.744408	3.397161	1.904001	1.0
warpx Ey (small)	13.791365	11.152048	10.238186	9.138474	5.684299	3.350974	1.892906	1.0
warpx Ex (small)	14.197639	11.542444	10.769422	9.488242	5.861758	3.432958	1.931868	1.0
pancreas	11.458135	9.733281	9.226104	8.464078	5.518582	3.363291	1.903055	1.0
bunny	11.088979	9.633609	9.157246	8.561649	5.809691	3.531247	1.966801	1.0
backpack	12.723715	10.721612	10.141905	9.271356	6.120022	3.641280	2.037362	1.0
present	11.023200	9.480858	9.060405	8.408479	5.760008	3.538956	1.970087	1.0
neocort. layer	12.447520	10.551892	10.109985	9.486455	6.404816	3.934218	2.204130	1.0
prone	12.328545	10.516972	9.808402	8.987704	5.829817	3.472328	1.929629	1.0
asteroid	16.202548	12.970309	12.138186	11.251342	6.965879	4.157815	2.293366	1.0
christmas tree	9.435864	8.684252	8.414776	8.014914	5.678880	3.594632	2.033457	1.0
vertebra	15.584401	12.716739	11.874489	10.650625	6.497383	3.760546	2.022003	1.0
mag. reconnection	11.143373	9.950549	9.442385	8.631012	5.840628	3.533175	1.951236	1.0
marmoset neurons	9.746149	8.728255	8.369362	7.873372	5.486779	3.432093	1.934376	1.0
stag beetle	16.893368	13.677734	12.953348	12.122359	7.725856	4.588125	2.502954	1.0
pawpawsaurus	15.037490	11.115806	10.390565	9.508620	6.257991	3.630051	1.975295	1.0
spathorhynchus	16.767363	11.413317	11.488654	10.186300	6.437389	3.646204	1.979606	1.0
kingsnake	12.430989	8.874472	8.481949	7.975080	5.719425	3.623299	2.025513	1.0
warpx rho (large)*	—	9.575844	9.204153	8.688152	5.889823	3.529090	1.996837	1.0
warpx Ez (large)*	—	9.658176	9.230529	8.586723	5.719471	3.384326	1.912030	1.0
warpx Ey (large)*	—	9.346317	8.950555	8.384174	5.623696	3.356279	1.868035	1.0
warpx Ex (large)*	—	9.268280	8.876489	8.212306	5.425843	3.244946	1.826927	1.0

TABLE S6: PPP2 (aug.) speed up compared to serial on Haswell. Datasets marked with * were evaluated on the Bigmem system.

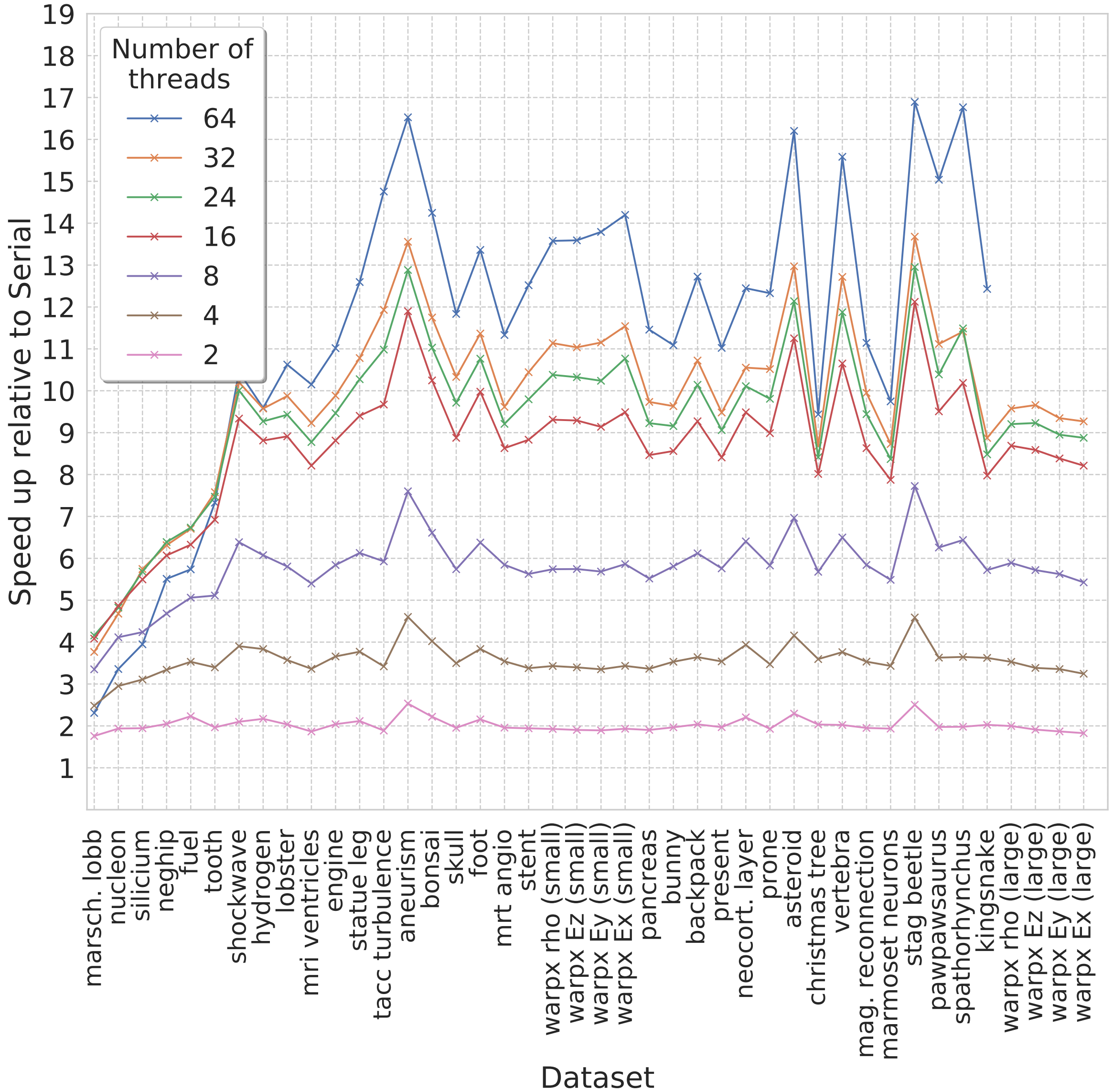


Fig. S2: PPP2 (aug.) speed up compared to serial on Haswell. See also Table S6

SUPPLEMENT F
TTK (AUG.): 3D DATASETS ON HASWELL

The timing results for TTK on Haswell included here have been published previously by the authors in [9] and are included here for reference to ease comparison with the results from our new algorithm.

F.1 Timings: TTK (aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	0.049862	0.030217	0.021816	0.020424	0.021291	0.028796	0.040681	0.062984
nucleon	0.042875	0.031371	0.021521	0.019456	0.020031	0.023282	0.029632	0.044781
silicium	0.102043	0.095027	0.088266	0.086240	0.040611	0.045389	0.058960	0.097439
neghip	0.060087	0.038009	0.040597	0.039004	0.046466	0.056421	0.077084	0.118688
fuel	0.097933	0.087743	0.085085	0.082077	0.085127	0.092970	0.106507	0.144186
tooth	2.626720	1.544830	1.328990	1.190950	1.227900	1.540450	1.893480	5.271760
shockwave	1.447200	0.969234	1.046470	0.795929	0.792436	0.850987	1.411480	1.711060
hydrogen	0.286965	0.261202	0.260882	0.276810	0.290801	0.379583	0.489017	0.897027
lobster	3.843540	2.287700	2.035030	1.843810	1.979520	2.754030	3.591520	5.494350
mri ventricles	17.827900	11.219400	9.730420	9.069210	9.191580	11.388500	14.368400	20.242700
engine	6.107600	3.232240	2.835140	2.598120	2.701860	3.525560	4.711360	6.789390
statue leg	4.236080	2.700060	2.360520	2.062490	3.153440	3.943640	5.647150	8.613430
tacc turbulence	5.014680	3.745910	3.636350	3.657250	5.271280	8.767310	12.792500	21.701800
aneurism	2.153240	2.196500	2.160850	2.197950	2.604890	3.207560	4.602450	8.633830
bonsai	2.779400	2.057730	1.906650	1.864640	2.420320	3.772570	5.308880	8.423000
skull	19.759800	12.478700	10.645100	9.790570	10.379700	14.066000	22.974500	35.141300
foot	8.985310	5.485160	5.023670	4.620460	5.114590	7.657740	9.003580	14.319000
mrt angio	59.381800	36.001600	31.195700	29.119300	31.687900	40.628400	52.898300	71.837000
stent	45.805100	37.771200	35.688400	35.734300	38.835400	51.775200	71.398900	96.321700
warpx rho (small)	16.450300	18.315400	18.300300	18.181100	24.474100	39.437200	57.300200	89.415300
warpx Ez (small)	23.403300	26.716600	26.585500	26.563900	33.289400	47.797600	71.774000	128.230000
warpx Ey (small)	10.581800	12.152500	12.408900	12.826600	20.718100	40.466900	64.301700	124.791000
warpx Ex (small)	19.844500	22.408400	22.677000	22.319200	26.031200	42.990200	67.780100	127.652000
pancreas	85.582200	61.432200	58.113900	58.978300	69.786700	103.438000	148.792000	206.390000
bunny	159.189000	114.513000	108.009000	108.468000	113.140000	159.588000	209.711000	298.799000
backpack	84.777800	52.132100	47.268200	42.080200	46.355000	72.303500	88.007000	127.822000
present	183.657000	137.897000	130.239000	128.413000	137.205000	159.860000	229.482000	297.514000
neocort. layer	151.719000	87.135400	80.580400	68.908500	68.542100	90.822700	115.003000	170.920000
prone	156.111000	92.651400	84.815800	80.490000	100.541000	139.140000	199.069000	300.989000
asteroid	22.478600	20.114100	19.870700	20.637600	25.492800	45.738400	59.076000	94.126300
christmas tree	231.260000	155.274000	138.467000	131.247000	145.541000	198.457000	246.902000	368.184000
vertebra	44.784300	30.756800	30.008600	29.982500	40.847800	65.843900	93.591300	157.115000
mag. reconnection	881.570000	777.465000	756.458000	777.343000	825.301000	934.525000	940.863000	1108.890000
marmoset neurons	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
stag beetle	31.648700	36.864300	37.601900	38.680500	52.992100	76.769800	119.961000	221.407000
pawpawsaurus	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
spathorhynchus	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
kingsnake	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx rho (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx Ez (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx Ey (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx Ex (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM

TABLE S7: TTK runtime in seconds on Haswell for all 3D datasets. We repeated each evaluation 5 times and report here the best time. OOM indicates that TTK did not complete computation of the contour tree in any of the 5 tries due to out-of-memory error. Similar to PPP, we attempted to process the OOM files on a Cori login node, with 512GB of main memory, to accommodate the larger memory requirements, but even with OMP STACKSIZE set to 1000M the files did not complete successfully.

F.2 Speed up: TTK (aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	1.263164	2.084396	2.887055	3.083838	2.958245	2.187233	1.548245	1.0
nucleon	1.044460	1.427460	2.080795	2.301667	2.235585	1.923409	1.511233	1.0
silicium	0.954880	1.025378	1.103921	1.129855	2.399320	2.146740	1.652626	1.0
neghip	1.975269	3.122637	2.923566	3.042962	2.554303	2.103621	1.539721	1.0
fuel	1.472291	1.643276	1.694607	1.756721	1.693777	1.550889	1.353770	1.0
tooth	2.006974	3.412518	3.966742	4.426517	4.293314	3.422221	2.784165	1.0
shockwave	1.182324	1.765373	1.635078	2.149765	2.159241	2.010677	1.212245	1.0
hydrogen	3.125911	3.434227	3.438440	3.240587	3.084676	2.363191	1.834347	1.0
lobster	1.429502	2.401692	2.699886	2.979889	2.775597	1.995022	1.529812	1.0
mri ventricles	1.135451	1.804259	2.080352	2.232025	2.202309	1.777468	1.408835	1.0
engine	1.111630	2.100522	2.394728	2.613193	2.512858	1.925762	1.441068	1.0
statue leg	2.033349	3.190088	3.648954	4.176229	2.731439	2.184132	1.525270	1.0
tacc turbulence	4.327654	5.793465	5.968017	5.933912	4.116989	2.475309	1.696447	1.0
aneurism	4.009692	3.930722	3.995571	3.928128	3.314470	2.691713	1.875920	1.0
bonsai	3.030510	4.093346	4.417696	4.517226	3.480118	2.232695	1.586587	1.0
skull	1.778424	2.816103	3.301171	3.589301	3.385580	2.498315	1.529578	1.0
foot	1.593601	2.610498	2.850307	3.099042	2.799638	1.869873	1.590367	1.0
mrt angio	1.209748	1.995384	2.302785	2.466989	2.267017	1.768147	1.358021	1.0
stent	2.102860	2.550136	2.698964	2.695497	2.480255	1.860383	1.349064	1.0
warpx rho (small)	5.435481	4.881974	4.886002	4.918036	3.653466	2.267283	1.560471	1.0
warpx Ez (small)	5.479142	4.799638	4.823306	4.827228	3.851977	2.682771	1.786580	1.0
warpx Ey (small)	11.792984	10.268751	10.056572	9.729079	6.023284	3.083780	1.940711	1.0
warpx Ex (small)	6.432614	5.696614	5.629140	5.719381	4.903808	2.969328	1.883326	1.0
pancreas	2.411600	3.359639	3.551474	3.499423	2.957440	1.995302	1.387104	1.0
bunny	1.877008	2.609302	2.766427	2.754720	2.640967	1.872315	1.424813	1.0
backpack	1.507730	2.451887	2.704186	3.037581	2.757459	1.767854	1.452407	1.0
present	1.619944	2.157509	2.284370	2.316853	2.168390	1.861091	1.296459	1.0
neocort. layer	1.126556	1.961545	2.121111	2.480391	2.493650	1.881908	1.486222	1.0
prone	1.928045	3.248618	3.548737	3.739458	2.993694	2.163210	1.511983	1.0
asteroid	4.187374	4.679618	4.736939	4.560913	3.692270	2.057927	1.593309	1.0
christmas tree	1.592078	2.371189	2.659002	2.805276	2.529761	1.855233	1.491215	1.0
vertebra	3.508261	5.108301	5.235666	5.240223	3.846352	2.386174	1.678735	1.0
mag. reconnection	1.257858	1.426289	1.465898	1.426513	1.343619	1.186581	1.178588	1.0
marmoset neurons	—	—	—	—	—	—	—	—
stag beetle	6.995769	6.006000	5.888187	5.723995	4.178113	2.884038	1.845658	1.0
pawpawsaurus	—	—	—	—	—	—	—	—
spathorhynchus	—	—	—	—	—	—	—	—
kingsnake	—	—	—	—	—	—	—	—
warpx rho (large)	—	—	—	—	—	—	—	—
warpx Ez (large)	—	—	—	—	—	—	—	—
warpx Ey (large)	—	—	—	—	—	—	—	—
warpx Ex (large)	—	—	—	—	—	—	—	—

TABLE S8: TTK speed-up compared to serial on Haswell for all 3D datasets. See Supplement F.1 for the corresponding timings.

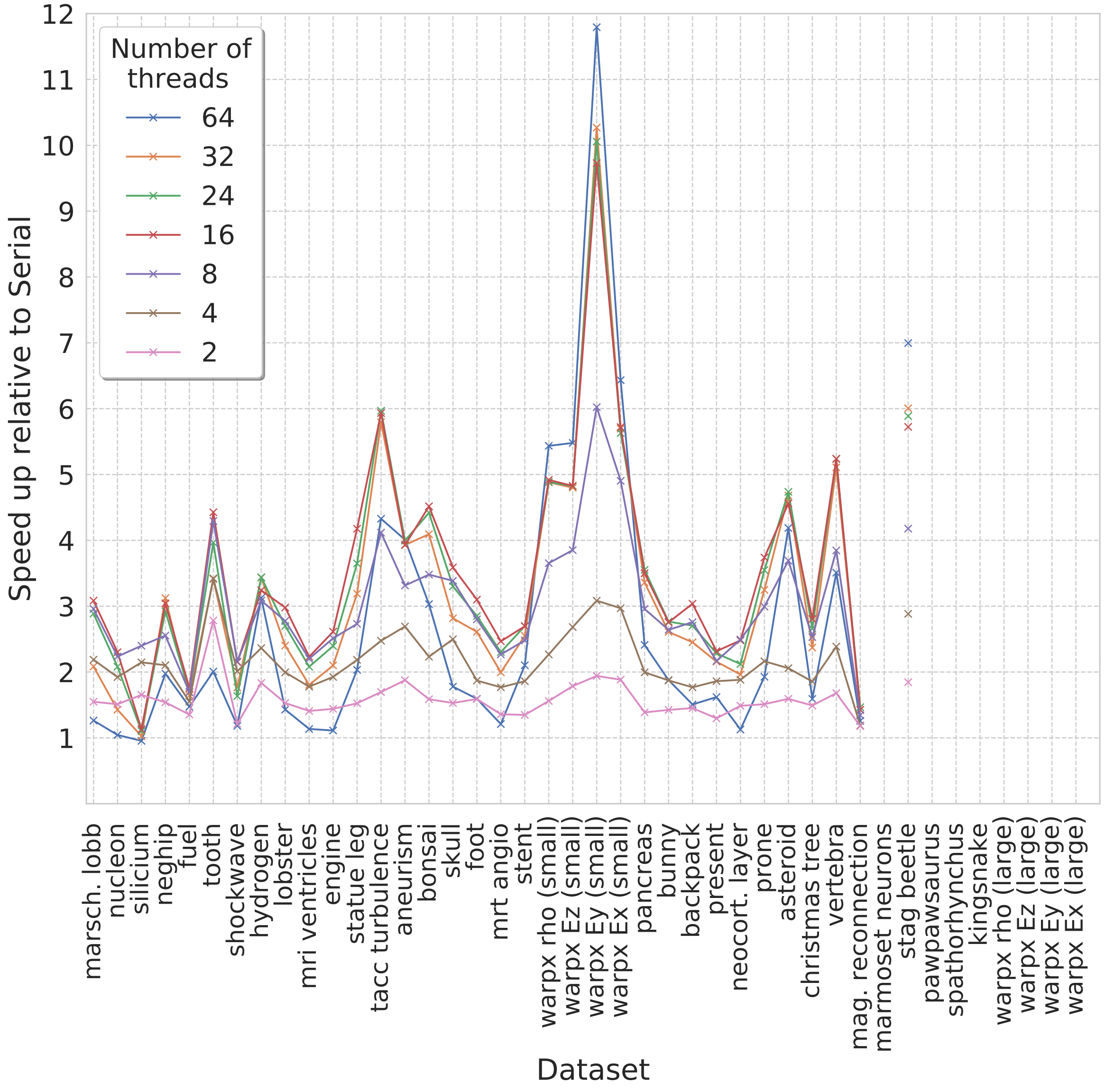


Fig. S3: TTK speed up compared to serial on Haswell. See also Table S8

SUPPLEMENT G
SPEED-UP PPP2 (AUG.) COMPARED TO TTK (AUG.): 3D DATASETS ON HASWELL

	64	32	24	16	8	4	2	1
marsch. lobb	2.598771	2.567543	2.046721	1.881433	1.609723	1.612492	1.613982	1.421055
nucleon	3.381560	3.447158	2.438207	2.224551	1.936298	1.613932	1.347949	1.051642
silicium	5.547624	7.517142	6.906527	6.527506	2.370061	1.942166	1.578784	1.341689
neghip	1.964758	1.423730	1.539048	1.404637	1.290970	1.118373	0.936626	0.704225
fuel	3.979613	4.167878	4.058789	3.677886	3.051657	2.324945	1.683362	1.021386
tooth	9.296182	5.653313	4.794855	3.980395	3.031268	2.526305	1.797153	2.545699
shockwave	13.021766	8.510787	9.015464	6.396755	4.355623	2.858654	2.550551	1.472969
hydrogen	2.176005	1.977575	1.912358	1.928828	1.396913	1.150977	0.838832	0.709331
lobster	7.438889	4.115375	3.494796	2.993183	2.093211	1.791495	1.330848	1.000898
mri ventricles	11.431365	6.541695	5.394132	4.704899	3.134523	2.419399	1.693652	1.278731
engine	7.588325	3.602734	3.025957	2.581649	1.779483	1.454391	1.083983	0.765644
statue leg	5.709257	3.114676	2.595567	2.075011	2.067938	1.591467	1.278947	0.921682
tacc turbulence	2.774849	1.676007	1.497704	1.326359	1.171721	1.124415	0.905830	0.813832
aneurism	2.589115	2.166259	2.025430	1.902723	1.440296	1.073721	0.848281	0.628157
bonsai	2.705328	1.651628	1.436769	1.305359	1.093060	1.036728	0.804927	0.575479
skull	8.386336	4.622768	3.708487	3.116745	2.136598	1.765028	1.610471	1.260263
foot	6.466764	3.358042	2.912830	2.483170	1.756565	1.582112	1.045100	0.771240
mrt angio	9.175511	4.720766	3.917729	3.426675	2.526019	1.962412	1.412561	0.979575
stent	8.601525	5.917042	5.244145	4.732976	3.277470	2.623522	2.080375	1.444771
warpx rho (small)	3.072519	2.805784	2.612895	2.328260	1.932618	1.860491	1.518888	1.230010
warpx Ez (small)	3.618976	3.354743	3.122838	2.807919	2.175806	1.847529	1.554903	1.459011
warpx Ey (small)	1.627591	1.511467	1.416886	1.307265	1.313425	1.512340	1.357469	1.391752
warpx Ex (small)	2.953934	2.711772	2.560487	2.220286	1.599803	1.547328	1.372853	1.338357
pancreas	7.258634	4.426015	3.968770	3.695128	2.850741	2.575147	2.095987	1.527728
bunny	7.467947	4.667028	4.184287	3.928762	2.780775	2.384102	1.744931	1.264083
backpack	6.771173	3.508594	3.009238	2.449001	1.780809	1.652651	1.125520	0.802368
present	8.095717	5.228084	4.718770	4.317840	3.160335	2.262326	1.807899	1.189728
neocort. layer	9.250312	4.503587	3.990373	3.201919	2.150293	1.750195	1.241595	0.837195
prone	6.206975	3.142515	2.682932	2.333057	1.890309	1.558143	1.238831	0.970701
asteroid	3.048578	2.183714	2.018886	1.943606	1.486409	1.591809	1.134042	0.787872
christmas tree	6.430820	3.973885	3.433784	3.100076	2.435747	2.102350	1.479598	1.085048
vertebra	3.058723	1.714120	1.561655	1.399482	1.163139	1.085152	0.829357	0.688560
mag. reconnection	14.776270	11.636416	10.743799	10.091733	7.250422	4.966466	2.761388	1.667938
marmoset neurons	—	—	—	—	—	—	—	—
stag beetle	1.694165	1.597732	1.543389	1.485809	1.297303	1.116116	0.951430	0.701576
pawpawsaurus	—	—	—	—	—	—	—	—
spathorhynchus	—	—	—	—	—	—	—	—
kingsnake	—	—	—	—	—	—	—	—
warpx rho (large)	—	—	—	—	—	—	—	—
warpx Ez (large)	—	—	—	—	—	—	—	—
warpx Ey (large)	—	—	—	—	—	—	—	—
warpx Ex (large)	—	—	—	—	—	—	—	—
Nyx	—	—	—	—	—	—	—	—

TABLE S9: Speed up of PPP2 (aug.) compared to TTK (aug.) on Haswell. See Supplements E and F for the corresponding performance tables for PPP2 (aug.) and TTK (aug.) on Haswell, respectively

SUPPLEMENT H
PPP1 (NO AUG.): 3D DATASETS ON HASWELL

The timing results for PPP1 (no aug.) on Haswell included here have been published previously by the authors in [9] and are included here for reference to ease comparison with the results from our new algorithm.

H.1 Timings: PPP1 (no aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	0.033055	0.030258	0.031103	0.027123	0.028659	0.031116	0.036576	0.039753
nucleon	0.031096	0.026896	0.026699	0.026854	0.024182	0.027759	0.034495	0.036596
silicium	0.034715	0.029465	0.029583	0.032216	0.028851	0.034026	0.043325	0.050460
neghip	0.043687	0.037691	0.037577	0.040677	0.041616	0.052291	0.068589	0.097084
fuel	0.039960	0.036719	0.037327	0.038628	0.040577	0.047113	0.063771	0.085372
tooth	0.267086	0.243260	0.248558	0.254357	0.342220	0.518939	0.816788	1.591930
shockwave	0.097280	0.094213	0.093131	0.095394	0.130586	0.187419	0.317553	0.587685
hydrogen	0.109471	0.108616	0.107407	0.111205	0.148911	0.216281	0.356003	0.621548
lobster	0.465318	0.478225	0.487549	0.512951	0.724602	1.132370	1.912070	3.564980
mri ventricles	1.481870	1.623060	1.691120	1.745950	2.498200	3.986720	6.927940	14.049000
engine	0.667967	0.717370	0.740272	0.785006	1.139780	1.787140	3.105430	6.028240
statue leg	0.603635	0.653635	0.673348	0.695886	1.036650	1.669470	2.894090	5.458320
tacc turbulence	0.891400	0.971992	1.011880	1.089030	1.668750	2.771720	4.841090	9.196700
aneurism	0.497326	0.598067	0.606651	0.630344	0.921900	1.537790	2.795060	5.215160
bonsai	0.640259	0.735186	0.757202	0.798286	1.228990	2.050770	3.577160	6.890100
skull	1.937480	2.164390	2.224800	2.357210	3.437780	5.590730	9.591030	19.672300
foot	1.081000	1.181380	1.222440	1.262570	1.898590	3.062520	5.324510	10.592500
mrt angio	5.615060	6.444860	6.636290	6.779310	9.555340	15.031600	25.812000	52.262700
stent	3.942580	4.485490	4.674960	4.850010	7.195720	11.500400	20.411000	41.098100
warpx rho (small)	1.747890	2.122500	2.189420	2.303060	3.623980	6.133930	11.015500	21.554800
warpx Ez (small)	1.863410	2.244090	2.319450	2.459290	3.870990	6.617620	11.867700	23.476900
warpx Ey (small)	1.822880	2.248890	2.339130	2.482620	3.924610	6.682670	12.007900	23.438100
warpx Ex (small)	2.188610	2.600140	2.712100	2.907770	4.653710	7.716100	13.842300	27.427500
pancreas	8.418060	10.037000	10.324500	10.664300	15.006200	23.516300	41.645100	81.438400
bunny	15.296000	17.878300	18.425300	18.800800	26.908300	42.765500	72.919300	144.755000
backpack	9.077100	10.402100	10.779200	11.118400	15.877700	25.664500	44.431200	86.976700
present	16.814100	19.730600	20.246400	20.893400	29.354200	45.740100	78.807700	161.914000
neocort. layer	13.962900	15.741300	16.328500	16.895600	24.072200	38.760300	67.291800	131.801000
prone	17.512400	20.680000	21.173900	21.929300	31.087700	49.612900	86.311500	169.273000
asteroid	3.915640	4.831200	4.939870	5.079580	7.670890	12.924200	23.023200	45.232200
christmas tree	28.032000	32.277800	33.110200	33.840500	46.654300	72.853800	125.615000	244.865000
vertebra	6.814380	8.012720	8.401800	8.921680	13.465800	21.907900	39.625500	77.141100
mag. reconnection	42.059000	47.922300	49.375900	51.038000	72.470400	115.074000	204.075000	394.356000
marmoset neurons	72.946000	83.369200	85.129100	87.734900	123.638000	200.107000	344.231000	678.229000
stag beetle	9.713370	11.887400	11.998100	12.258100	18.675600	31.438200	58.305700	112.556000
pawpawsaurus	116.667000	154.088000	158.855000	161.947000	237.537000	379.057000	659.068000	1301.130000
spathorhynchus	78.139400	80.754100	88.111600	102.787000	157.699000	255.936000	437.597000	844.489000
kingsnake	92.553600	121.540000	124.073000	130.367000	181.377000	295.014000	502.689000	999.994000
warpx rho (large)*	19.398200	22.225600	25.102500	30.858400	50.491600	94.365700	180.433000	325.116000
warpx Ez (large)*	19.698200	22.551600	25.811100	31.523200	51.664700	94.584300	182.051000	345.387000
warpx Ey (large)*	19.926100	22.028700	25.209700	31.117700	52.116100	99.732400	185.629000	332.137000
warpx Ex (large)*	19.514700	22.853700	24.664400	30.554600	49.821300	92.615300	176.965000	325.281000

TABLE S10: PPP1 (no aug.) runtime in seconds on Haswell for all 3D datasets. We repeated each evaluation 5 times and report here the best time. Datasets marked with * were evaluated on the Bigmem system.

H.2 Speed up: PPP1 (no aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	1.202617	1.313797	1.278088	1.465629	1.387084	1.277562	1.086851	1.0
nucleon	1.176891	1.360659	1.370709	1.362787	1.513382	1.318338	1.060895	1.0
silicium	1.453571	1.712511	1.705681	1.566298	1.748980	1.483001	1.164683	1.0
neghip	2.222263	2.575780	2.583574	2.386687	2.332853	1.856607	1.415441	1.0
fuel	2.136447	2.325041	2.287120	2.210095	2.103956	1.812084	1.338734	1.0
tooth	5.960365	6.544150	6.404662	6.258644	4.651774	3.067663	1.949012	1.0
shockwave	6.041170	6.237827	6.310318	6.160633	4.500368	3.135675	1.850667	1.0
hydrogen	5.677741	5.722435	5.786848	5.589209	4.173956	2.873798	1.745907	1.0
lobster	7.661384	7.454608	7.312045	6.949943	4.919915	3.148247	1.864461	1.0
mri ventricles	9.480589	8.655872	8.307512	8.046622	5.623649	3.523950	2.027876	1.0
engine	9.024757	8.403251	8.143277	7.679228	5.288950	3.373121	1.941193	1.0
statue leg	9.042418	8.350716	8.106239	7.843699	5.265345	3.269493	1.886023	1.0
tacc turbulence	10.317142	9.461703	9.088726	8.444855	5.511131	3.318048	1.899717	1.0
aneurism	10.486401	8.720026	8.596640	8.273514	5.656969	3.391334	1.865849	1.0
bonsai	10.761426	9.371914	9.099421	8.631117	5.606311	3.359762	1.926137	1.0
skull	10.153550	9.089074	8.842278	8.345587	5.722385	3.518735	2.051114	1.0
foot	9.798797	8.966209	8.665047	8.389634	5.579140	3.458753	1.989385	1.0
mrt angio	9.307594	8.109206	7.875289	7.709147	5.469476	3.476855	2.024744	1.0
stent	10.424164	9.162455	8.791113	8.473818	5.711465	3.573624	2.013527	1.0
warpx rho (small)	12.331897	10.155383	9.844982	9.359200	5.947825	3.514028	1.956770	1.0
warpx Ez (small)	12.598891	10.461657	10.121753	9.546210	6.064831	3.547635	1.978218	1.0
warpx Ey (small)	12.857731	10.422075	10.020007	9.440873	5.972084	3.507296	1.951890	1.0
warpx Ex (small)	12.531927	10.548470	10.113012	9.432486	5.893685	3.554581	1.981426	1.0
pancreas	9.674248	8.113819	7.887878	7.636544	5.426984	3.463062	1.955534	1.0
bunny	9.463585	8.096687	7.856317	7.699406	5.379567	3.384855	1.985140	1.0
backpack	9.581992	8.361456	8.068938	7.822771	5.477916	3.388989	1.957559	1.0
present	9.629656	8.206238	7.997175	7.749529	5.515872	3.539870	2.054545	1.0
neocort. layer	9.439371	8.372943	8.071838	7.800907	5.475237	3.400412	1.958649	1.0
prone	9.665894	8.185348	7.994418	7.719033	5.445015	3.411875	1.961187	1.0
asteroid	11.551675	9.362519	9.156557	8.904713	5.896604	3.499807	1.964636	1.0
christmas tree	8.735195	7.586174	7.395455	7.235856	5.248498	3.361046	1.949329	1.0
vertebra	11.320340	9.627330	9.181497	8.646477	5.728668	3.521154	1.946754	1.0
mag. reconnection	9.376257	8.229071	7.986811	7.726713	5.441615	3.426977	1.932407	1.0
marmoset neurons	9.297686	8.135247	7.967064	7.730436	5.485603	3.389332	1.970273	1.0
stag beetle	11.587739	9.468513	9.381152	9.182173	6.026901	3.580230	1.930446	1.0
pawpawsaurus	11.152511	8.444071	8.190677	8.034295	5.477589	3.432544	1.974197	1.0
spathorhynchus	10.807467	10.457537	9.584311	8.215913	5.355069	3.299610	1.929833	1.0
kingsnake	10.804485	8.227695	8.059723	7.670607	5.513345	3.389649	1.989290	1.0
warpx rho (large)	16.760112	14.627997	12.951539	10.535737	6.439012	3.445277	1.801866	1.0
warpx Ez (large)	17.533937	15.315410	13.381336	10.956597	6.685164	3.651631	1.897199	1.0
warpx Ey (large)	16.668440	15.077467	13.174968	10.673572	6.373021	3.330282	1.789252	1.0
warpx Ex (large)	16.668511	14.233188	13.188279	10.645893	6.528954	3.512173	1.838109	1.0

TABLE S11: PPP1 (no aug.) speed up compared to serial on Haswell. Datasets marked with * were evaluated on the Bigmem system.

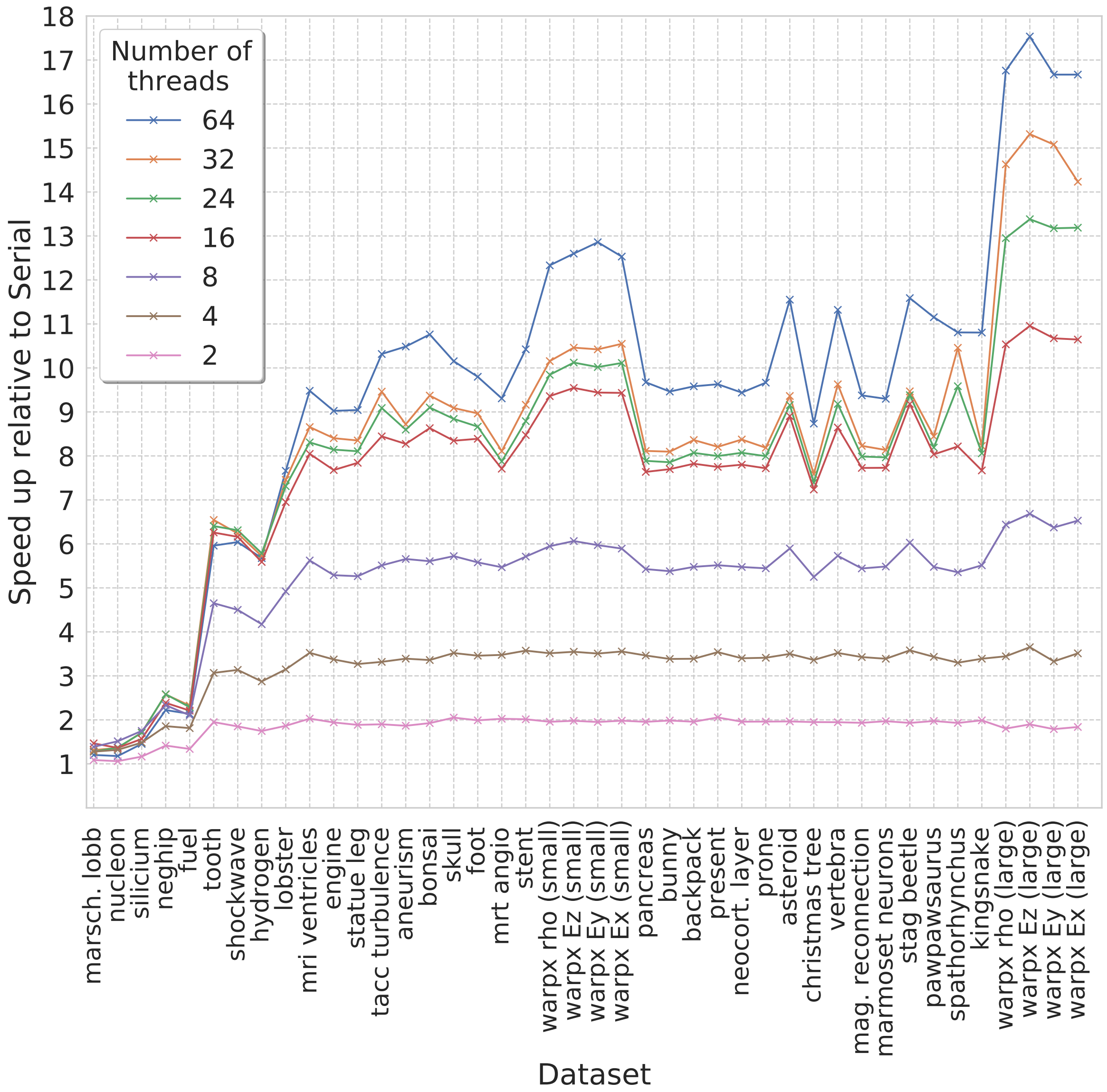


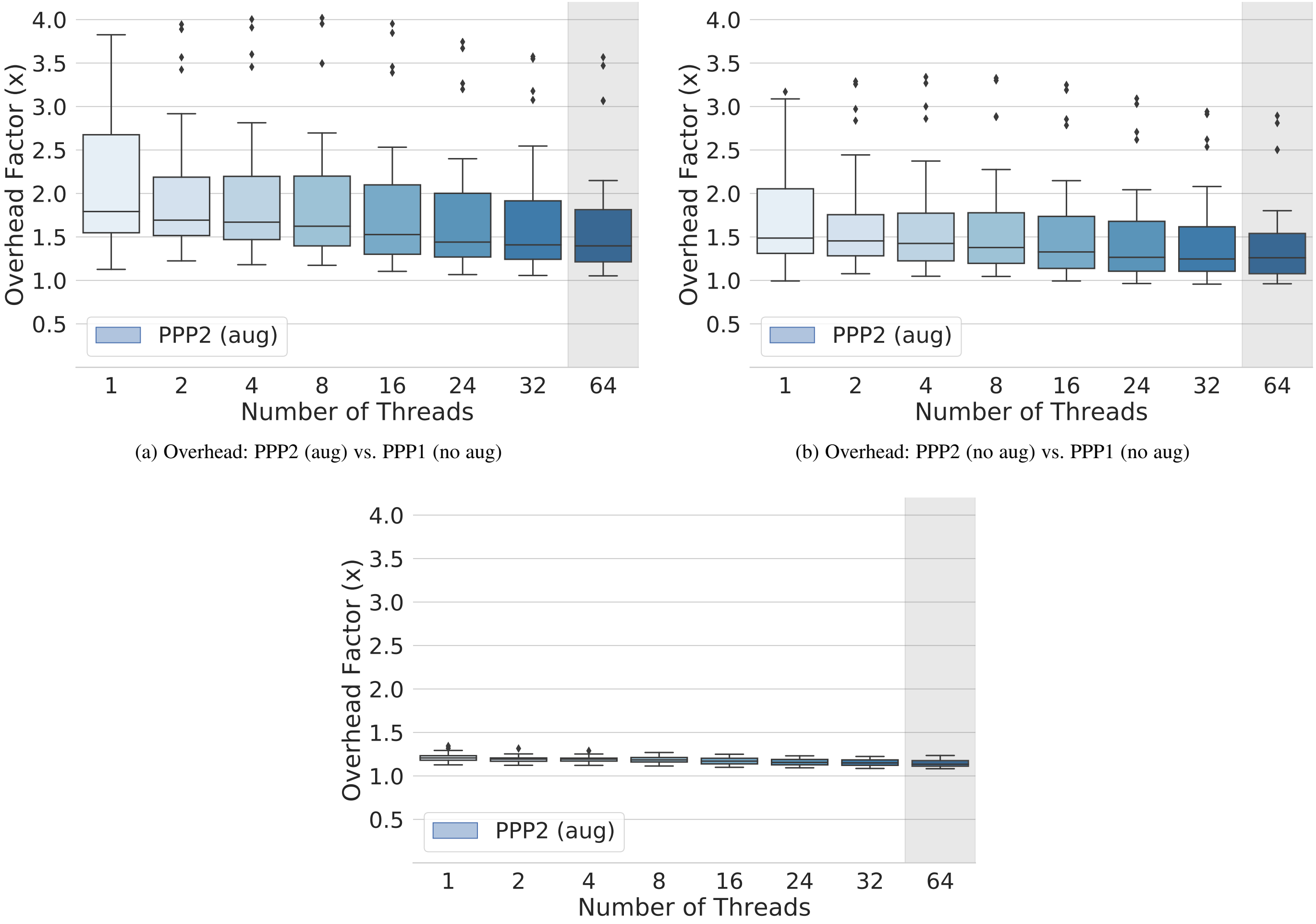
Fig. S4: PPP1 (no aug.) speed up compared to serial on Haswell. See also Table S11

SUPPLEMENT I
SPEED-UP PPP1 (NO AUG.) COMPARED TO PPP2 (AUG.): 3D DATASETS ON HASWELL

	64	32	24	16	8	4	2	1
marsch. lobb	0.580444	0.388947	0.342695	0.400225	0.461507	0.573918	0.689116	1.114935
nucleon	0.407744	0.338364	0.330602	0.325689	0.427805	0.519673	0.637276	1.163570
silicium	0.529865	0.429024	0.432001	0.410099	0.593911	0.686848	0.861976	1.439239
neghip	0.700034	0.708302	0.701964	0.682644	0.864884	0.964772	1.199894	1.735992
fuel	0.615836	0.573340	0.561605	0.577720	0.687467	0.848775	0.992155	1.653552
tooth	1.057933	1.123329	1.115112	1.176315	1.183677	1.175021	1.289931	1.300842
shockwave	1.142444	1.208781	1.246365	1.304354	1.393212	1.588356	1.742708	1.976637
hydrogen	1.204675	1.216046	1.270113	1.290518	1.397976	1.524831	1.637554	2.034614
lobster	1.110385	1.162405	1.194348	1.200900	1.305111	1.357577	1.411387	1.539818
mri ventricles	1.052427	1.056683	1.066684	1.104047	1.173793	1.180710	1.224560	1.126792
engine	1.204952	1.250628	1.265670	1.282003	1.332134	1.356402	1.399594	1.471001
statue leg	1.229165	1.326249	1.350629	1.428346	1.471008	1.484297	1.525685	1.712128
tacc turbulence	2.027361	2.299422	2.399445	2.531941	2.695880	2.813134	2.917194	2.899540
aneurism	1.672245	1.695395	1.758606	1.832587	1.961796	1.942612	1.941146	2.635528
bonsai	1.604632	1.694646	1.752557	1.789396	1.801691	1.774416	1.843776	2.124280
skull	1.216111	1.247187	1.290215	1.332626	1.413136	1.425445	1.487400	1.417430
foot	1.285347	1.382654	1.410842	1.473748	1.533612	1.580463	1.617997	1.752768
mrt angio	1.152574	1.183303	1.199872	1.253495	1.312837	1.377318	1.450817	1.403198
stent	1.350697	1.423135	1.455709	1.556712	1.646701	1.716027	1.681456	1.622197
warpx rho (small)	3.063128	3.075491	3.198948	3.390654	3.494418	3.455729	3.424729	3.372557
warpx Ez (small)	3.470428	3.548802	3.670374	3.846781	3.952426	3.909427	3.889532	3.743608
warpx Ey (small)	3.566614	3.575186	3.744072	3.952188	4.019278	4.004058	3.944803	3.825596
warpx Ex (small)	3.069524	3.178052	3.265558	3.457082	3.496458	3.600718	3.566727	3.477517
pancreas	1.400608	1.382863	1.418258	1.496685	1.631339	1.708083	1.704618	1.658873
bunny	1.393587	1.372424	1.400954	1.468485	1.512043	1.565243	1.648164	1.632938
backpack	1.379339	1.428404	1.457223	1.545420	1.639425	1.704689	1.759851	1.831594
present	1.349207	1.336817	1.363215	1.423421	1.478994	1.544855	1.610667	1.544456
neocort. layer	1.174649	1.229123	1.236715	1.273764	1.324171	1.338816	1.376471	1.548987
prone	1.436177	1.425687	1.493022	1.573229	1.710889	1.799907	1.861757	1.831798
asteroid	1.883082	1.906557	1.992443	2.090370	2.235803	2.223240	2.262644	2.641238
christmas tree	1.282862	1.210541	1.217900	1.251066	1.280742	1.295714	1.328432	1.385764
vertebra	2.148618	2.239339	2.287117	2.401341	2.607985	2.769645	2.847863	2.957943
mag. reconnection	1.418512	1.394196	1.425975	1.509223	1.570683	1.635183	1.669587	1.685855
marmoset neurons	1.159340	1.132693	1.156844	1.193197	1.214999	1.200118	1.237811	1.215259
stag beetle	1.923225	1.940954	2.030588	2.123763	2.187234	2.187880	2.162482	2.803804
pawpawsaurus	1.592404	1.631048	1.692531	1.814205	1.879362	2.030296	2.145924	2.147118
spathorhynchus	1.790723	2.545580	2.317720	2.240828	2.311131	2.514148	2.708382	2.778236
kingsnake	1.456421	1.553546	1.592256	1.611696	1.615293	1.567614	1.645703	1.675670
warpx rho (large)	—	6.330673	5.831491	5.025504	4.530635	4.045792	3.739571	4.144213
warpx Ez (large)	—	7.184102	6.567678	5.780790	5.295356	4.888253	4.495284	4.530425
warpx Ey (large)	—	7.854980	7.167321	6.198787	5.517988	4.831479	4.663840	4.869196
warpx Ex (large)	—	6.706529	6.488461	5.661242	5.254981	4.726757	4.393846	4.367116

TABLE S12: Speed up of PPP1 (no aug.) compared to PPP2 (aug.) on Haswell. See Supplements E and H for the corresponding performance tables for PPP2 (aug.) and PPP1 (no aug.) on Haswell, respectively

SUPPLEMENT J
OVERHEAD FOR AUGMENTATION FOR PPP2: 3D DATASETS ON HASWELL



(a) Overhead: PPP2 (aug) vs. PPP1 (no aug)

(b) Overhead: PPP2 (no aug) vs. PPP1 (no aug)

(c) Overhead: PPP2 (aug) vs (no aug)

Fig. S5: Overhead for augmentation for PPP2 on Haswell for a large collection of 3D datasets of varying size and using varying numbers of threads. **(a)** PPP2 (aug) vs PPP1 (no aug) **(b)** PPP2 (no aug) compared to PPP1 (no aug). **(c)** PPP2 (aug) vs PPP2 (no aug) Gray area indicates hyperthreading (2 threads per core).

SUPPLEMENT K
PPP2 (NO AUG.): 3D DATASETS ON HASWELL

K.1 Timings: PPP2 (no aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	0.018292	0.010871	0.009741	0.009898	0.011907	0.015712	0.021876	0.037747
nucleon	0.011908	0.008375	0.008117	0.007906	0.009234	0.012690	0.019001	0.033505
silicium	0.017087	0.011413	0.011597	0.011914	0.015336	0.020649	0.032475	0.058612
neghip	0.027898	0.024096	0.023673	0.024954	0.031794	0.043229	0.069767	0.139952
fuel	0.022008	0.018747	0.018577	0.019950	0.024678	0.035107	0.054410	0.116366
tooth	0.261562	0.251293	0.254806	0.273900	0.365017	0.544596	0.932156	1.817192
shockwave	0.093816	0.096914	0.099220	0.107725	0.156759	0.252183	0.474125	0.929722
hydrogen	0.114442	0.115217	0.118848	0.122597	0.176237	0.275476	0.483158	1.068324
lobster	0.462815	0.489209	0.510285	0.533678	0.804823	1.301611	2.283985	4.567149
mri ventricles	1.424414	1.553278	1.630600	1.734273	2.612086	4.178234	7.461250	13.955800
engine	0.715247	0.785761	0.815623	0.866256	1.301340	2.067189	3.688967	7.476970
statue leg	0.651403	0.747514	0.774644	0.832823	1.248477	2.020131	3.601291	7.477870
tacc turbulence	1.544296	1.904195	2.067211	2.338516	3.797030	6.577240	11.828930	21.907380
aneurism	0.715736	0.853830	0.889248	0.944720	1.447673	2.385957	4.330220	10.643400
bonsai	0.877712	1.052880	1.111614	1.182005	1.816339	2.969383	5.350600	11.343780
skull	2.117005	2.394886	2.520536	2.724763	4.178919	6.787870	12.144430	23.527120
foot	1.222084	1.419978	1.490146	1.580537	2.442183	4.055024	7.185590	15.080700
mrt angio	5.972979	7.023602	7.283457	7.732446	11.263060	18.465010	33.248510	64.591700
stent	4.652773	5.553726	5.899703	6.457600	10.012600	16.535950	28.946510	56.082000
warpx rho (small)	4.378856	5.383710	5.732620	6.411660	10.431160	17.546030	31.260190	59.158100
warpx Ez (small)	5.236400	6.534960	7.027210	7.845950	12.774280	21.636600	38.661760	72.490900
warpx Ey (small)	5.275090	6.613390	7.233620	8.064450	13.057950	22.322270	39.499800	74.295000
warpx Ex (small)	5.473190	6.812490	7.341210	8.297720	13.435340	23.154930	41.134400	78.329700
pancreas	10.605380	12.367570	12.971730	14.020680	21.129660	34.504460	60.717300	115.704600
bunny	19.287410	22.108810	23.166330	24.524870	35.680680	58.399850	104.651000	204.808000
backpack	11.112440	13.055490	13.678090	14.712030	21.770470	36.386570	64.901100	130.618400
present	20.403750	23.525650	24.562970	26.237510	37.643660	60.840950	109.007600	213.714300
neocort. layer	14.926440	17.403950	18.007710	18.915670	27.296620	44.031190	78.396000	169.165500
prone	22.573830	26.245800	28.015270	30.115910	45.663140	75.922200	135.610500	260.157200
asteroid	6.272360	7.811380	8.285280	8.833730	14.014400	23.563940	42.482560	92.402600
christmas tree	32.917580	35.591520	36.519520	38.231850	53.194530	84.078700	148.676300	301.052300
vertebra	12.277170	15.073210	16.183160	18.000690	29.214810	50.587000	93.959100	186.871100
mag. reconnection	54.955120	61.171540	64.139020	69.197990	100.367000	164.827900	296.068400	580.446300
marmoset neurons	78.110570	86.645380	89.947640	94.520700	132.935600	210.287400	375.172500	731.076200
stag beetle	15.668060	19.225640	20.228030	21.272250	32.934240	55.224600	100.655000	234.854000
pawpawsaurus	164.294800	216.207600	229.700500	246.828600	365.924100	618.895000	1137.393000	2258.561000
spathorhynchus	118.751100	167.972900	165.904200	184.352700	287.150000	499.207000	900.843000	1780.192000
kingsnake	119.049100	162.494200	169.260800	177.777300	243.641700	384.354200	690.087000	1402.796000
warpx rho (large)*	—	117.855600	123.357900	131.045500	194.070200	323.058200	568.198000	1112.813000
warpx Ez (large)*	—	139.537300	145.808600	156.915000	237.031500	399.823800	708.219000	1335.720000
warpx Ey (large)*	—	145.295100	152.589900	163.366500	245.022000	411.534300	735.784000	1359.286000
warpx Ex (large)*	—	135.213600	140.916400	152.229200	229.980800	385.338900	679.836500	1208.355000

TABLE S13: PPP2 (no aug.) runtime in seconds on Haswell for all 3D datasets. We repeated each evaluation 5 times and report here the best time. Datasets marked with * were evaluated on the Bigmem system.

K.2 Speed up: PPP2 (no aug.) for 3D datasets on Haswell

	64	32	24	16	8	4	2	1
marsch. lobb	2.063621	3.472351	3.875093	3.813791	3.170019	2.402440	1.725528	1.0
nucleon	2.813655	4.000802	4.127955	4.238074	3.628438	2.640241	1.763343	1.0
silicium	3.430210	5.135367	5.054022	4.919673	3.821860	2.838436	1.804829	1.0
neghip	5.016488	5.808086	5.911908	5.608377	4.401802	3.237456	2.005997	1.0
fuel	5.287513	6.207114	6.263915	5.832794	4.715449	3.314629	2.138672	1.0
tooth	6.947462	7.231367	7.131669	6.634509	4.978376	3.336768	1.949451	1.0
shockwave	9.910058	9.593288	9.370308	8.630513	5.930900	3.686696	1.960922	1.0
hydrogen	9.335094	9.272278	8.988994	8.714112	6.061863	3.878102	2.211128	1.0
lobster	9.868196	9.335792	8.950193	8.557874	5.674725	3.508843	1.999641	1.0
mri ventricles	9.797573	8.984741	8.558690	8.047061	5.342780	3.340119	1.870437	1.0
engine	10.453689	9.515578	9.167189	8.631363	5.745593	3.616975	2.026847	1.0
statue leg	11.479637	10.003652	9.653299	8.978943	5.989594	3.701676	2.076441	1.0
tacc turbulence	14.185998	11.504799	10.597554	9.368069	5.769609	3.330786	1.852017	1.0
aneurism	14.870567	12.465479	11.968990	11.266195	7.352075	4.460852	2.457935	1.0
bonsai	12.924262	10.774048	10.204783	9.597066	6.245409	3.820248	2.120095	1.0
skull	11.113398	9.823900	9.334173	8.634556	5.629954	3.466053	1.937277	1.0
foot	12.340150	10.620376	10.120284	9.541504	6.175090	3.719016	2.098742	1.0
mrt angio	10.813984	9.196378	8.868275	8.353333	5.734827	3.498059	1.942695	1.0
stent	12.053457	10.098086	9.505902	8.684651	5.601143	3.391520	1.937436	1.0
warpx rho (small)	13.509944	10.988352	10.319557	9.226643	5.671287	3.371595	1.892442	1.0
warpx Ez (small)	13.843652	11.092784	10.315744	9.239276	5.674754	3.350383	1.875003	1.0
warpx Ey (small)	14.084120	11.234027	10.270791	9.212656	5.689637	3.328291	1.880896	1.0
warpx Ex (small)	14.311526	11.497954	10.669862	9.439906	5.830124	3.382852	1.904238	1.0
pancreas	10.909991	9.355484	8.919751	8.252424	5.475933	3.353323	1.905628	1.0
bunny	10.618740	9.263637	8.840762	8.351033	5.740025	3.506995	1.957057	1.0
backpack	11.754250	10.004864	9.549462	8.878340	5.999797	3.589742	2.012576	1.0
present	10.474266	9.084310	8.700670	8.145373	5.677299	3.512672	1.960545	1.0
neocort. layer	11.333278	9.719949	9.394060	8.943141	6.197306	3.841947	2.157833	1.0
prone	11.524726	9.912336	9.286264	8.638530	5.697313	3.426629	1.918415	1.0
asteroid	14.731712	11.829229	11.152622	10.460202	6.593404	3.921356	2.175071	1.0
christmas tree	9.145639	8.458540	8.243600	7.874385	5.659460	3.580601	2.024884	1.0
vertebra	15.221024	12.397565	11.547257	10.381330	6.396451	3.694054	1.988856	1.0
mag. reconnection	10.562188	9.488829	9.049816	8.388196	5.783239	3.521529	1.960514	1.0
marmoset neurons	9.359504	8.437567	8.127797	7.734562	5.499476	3.476557	1.948640	1.0
stag beetle	14.989348	12.215666	11.610325	11.040393	7.130998	4.252706	2.333257	1.0
pawpawsaurus	13.747002	10.446261	9.832634	9.150321	6.172212	3.649344	1.985735	1.0
spathorhynchus	14.990952	10.598091	10.730241	9.656447	6.199519	3.566040	1.976140	1.0
kingsnake	11.783340	8.632899	8.287778	7.890749	5.757619	3.649748	2.032781	1.0
warpx rho (large)	—	9.442173	9.021011	8.491806	5.734075	3.444621	1.958495	1.0
warpx Ez (large)	—	9.572494	9.160777	8.512379	5.635200	3.340772	1.886027	1.0
warpx Ey (large)	—	9.355346	8.908099	8.320470	5.547608	3.302971	1.847398	1.0
warpx Ex (large)	—	8.936638	8.574978	7.937735	5.254156	3.135824	1.777420	1.0

TABLE S14: PPP2 (no aug.) speed up compared to serial on Haswell. Datasets marked with * were evaluated on the Bigmem system.

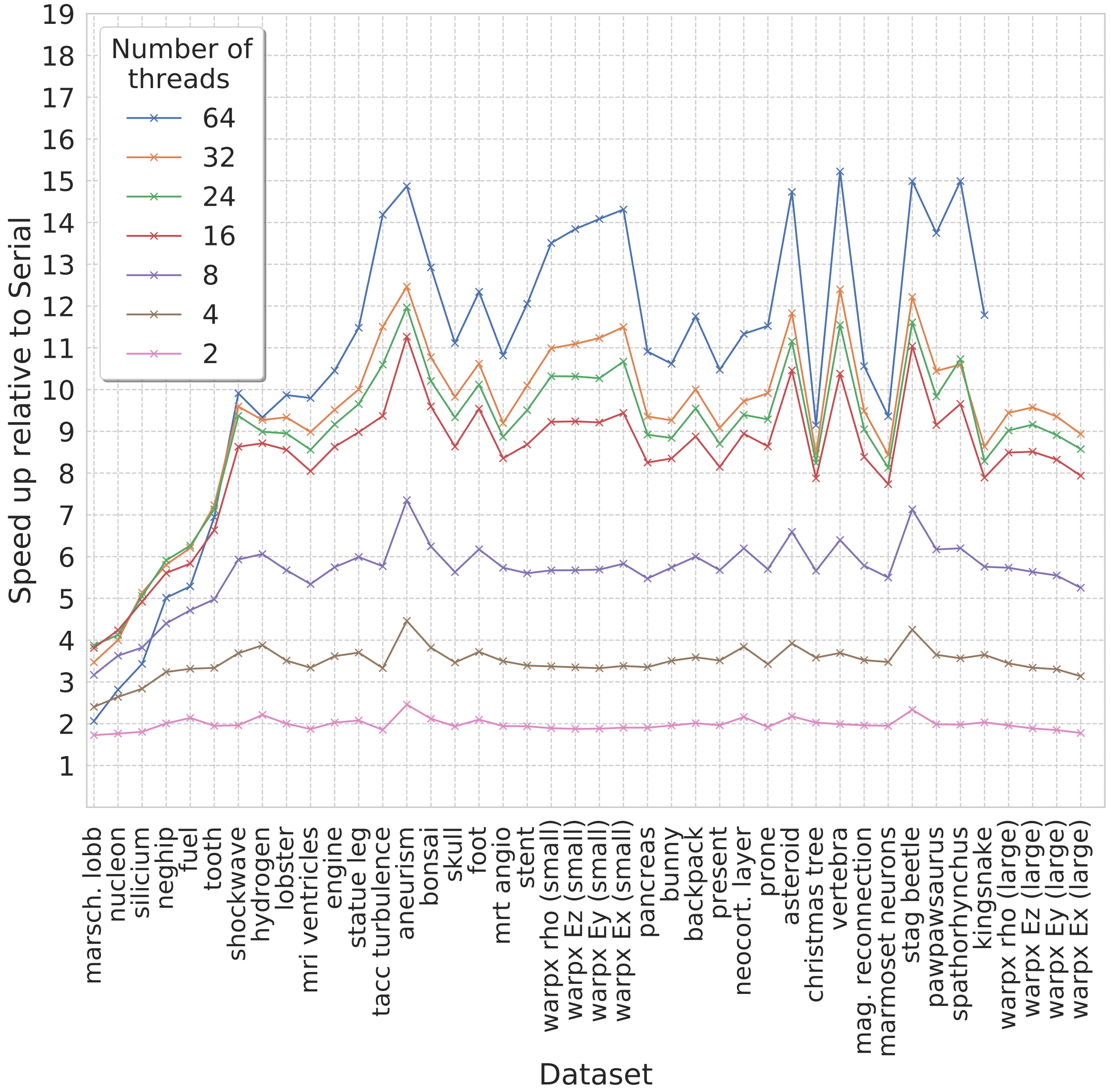


Fig. S6: PPP2 (no aug.) speed up compared to serial on Haswell. See also Table S14

SUPPLEMENT L
SPEED-UP PPP1 (NO AUG.) COMPARED TO PPP2 (NO AUG.): 3D DATASETS ON HASWELL

	64	32	24	16	8	4	2	1
marsch. lobb	0.553363	0.359267	0.313178	0.364905	0.415483	0.504942	0.598082	0.949538
nucleon	0.382949	0.311371	0.304009	0.294398	0.381861	0.457150	0.550822	0.915537
silicium	0.492215	0.387348	0.392012	0.369809	0.531557	0.606878	0.749568	1.161554
neghip	0.638597	0.639304	0.629977	0.613465	0.763991	0.826699	1.017169	1.441556
fuel	0.550746	0.510566	0.497684	0.516470	0.608169	0.745168	0.853219	1.363046
tooth	0.979318	1.033022	1.025137	1.076833	1.066615	1.049442	1.141246	1.141502
shockwave	0.964391	1.028666	1.065383	1.129269	1.200427	1.345557	1.493058	1.582007
hydrogen	1.045407	1.060774	1.106520	1.102441	1.183505	1.273695	1.357174	1.718812
lobster	0.994621	1.022967	1.046633	1.040407	1.110710	1.149457	1.194509	1.281115
mri ventricles	0.961227	0.957006	0.964213	0.993312	1.045587	1.048038	1.076980	0.993366
engine	1.070782	1.095336	1.101788	1.103502	1.141747	1.156702	1.187909	1.240324
statue leg	1.079134	1.143626	1.150436	1.196781	1.204338	1.210043	1.244360	1.369995
tacc turbulence	1.732439	1.959064	2.042941	2.147338	2.275374	2.372981	2.443444	2.382091
aneurism	1.439169	1.427649	1.465831	1.498737	1.570315	1.551549	1.549240	2.040858
bonsai	1.370870	1.432127	1.468055	1.480679	1.477912	1.447936	1.495768	1.646388
skull	1.092659	1.106495	1.132927	1.155927	1.215587	1.214129	1.266228	1.195952
foot	1.130512	1.201965	1.218993	1.251841	1.286314	1.324081	1.349531	1.423715
mrt angio	1.063743	1.089799	1.097519	1.140595	1.178719	1.228413	1.288103	1.235904
stent	1.180134	1.238154	1.261979	1.331461	1.391466	1.437859	1.418182	1.364589
warpx rho (small)	2.505224	2.536495	2.618328	2.783974	2.878371	2.860487	2.837837	2.744544
warpx Ez (small)	2.810117	2.912076	3.029688	3.190331	3.300003	3.269544	3.257730	3.087754
warpx Ey (small)	2.893822	2.940735	3.092440	3.248363	3.327197	3.340322	3.289484	3.169839
warpx Ex (small)	2.500761	2.620047	2.706836	2.853637	2.887017	3.000859	2.971645	2.855882
pancreas	1.259837	1.232198	1.256403	1.314730	1.408062	1.467257	1.457970	1.420762
bunny	1.260945	1.236628	1.257311	1.304459	1.326010	1.365583	1.435162	1.414860
backpack	1.224228	1.255082	1.268934	1.323215	1.371135	1.417778	1.460710	1.501763
present	1.213490	1.192343	1.213202	1.255780	1.282394	1.330145	1.383210	1.319925
neocort. layer	1.069007	1.105623	1.102839	1.119562	1.133948	1.135987	1.165016	1.283492
prone	1.289020	1.269139	1.323104	1.373318	1.468849	1.530292	1.571175	1.536909
asteroid	1.601874	1.616861	1.677226	1.739067	1.826959	1.823242	1.845207	2.042850
christmas tree	1.174286	1.102663	1.102969	1.129766	1.140185	1.154074	1.183587	1.229462
vertebra	1.801656	1.881160	1.926154	2.017635	2.169556	2.309076	2.371178	2.422458
mag. reconnection	1.306620	1.276473	1.298994	1.355813	1.384938	1.432364	1.450782	1.471884
marmoset neurons	1.070800	1.039297	1.056603	1.077344	1.075200	1.050875	1.089886	1.077919
stag beetle	1.613041	1.617312	1.685936	1.735363	1.763490	1.756608	1.726332	2.086552
pawpawsaurus	1.408237	1.403144	1.445976	1.524132	1.540493	1.632723	1.725760	1.735846
spathorhynchus	1.519734	2.080054	1.882887	1.793541	1.820874	1.950515	2.058613	2.108011
kingsnake	1.286272	1.336961	1.364203	1.363668	1.343289	1.302834	1.372791	1.402804
warpx rho (large)	—	5.302696	4.914168	4.246672	3.843614	3.423471	3.149080	3.422818
warpx Ez (large)	—	6.187468	5.649066	4.977762	4.587881	4.227169	3.890223	3.867314
warpx Ey (large)	—	6.595718	6.052825	5.249954	4.701465	4.126385	3.963734	4.092546
warpx Ex (large)	—	5.916486	5.713352	4.982202	4.616114	4.160640	3.841644	3.714804

TABLE S15: Speed up of PPP1 (no aug.) compared to PPP2 (no aug.) on Haswell. See Supplements K and H for the corresponding performance tables for PPP2 (no aug.) and PPP1 (no aug.) on Haswell, respectively

Supplement M
PPP2 (aug.): 3D datasets on KNL

M.1 Timings: PPP2 (aug.) for 3D datasets on KNL

	272	136	64	32	16	8	1
marsch. lobb	0.189881	0.0785264	0.0474332	0.053703	0.0465615	0.0500553	0.198165
nucleon	0.144268	0.059897	0.0390479	0.0519011	0.0436207	0.0459974	0.204289
silicium	0.162643	0.0702245	0.0461271	0.0629211	0.0579499	0.06547	0.335728
neghip	0.257693	0.116668	0.0786994	0.131563	0.140844	0.142237	0.828035
fuel	0.199577	0.109097	0.0678452	0.110688	0.111103	0.120299	0.805161
tooth	1.83919	1.13364	0.78989	1.37936	1.39045	1.67887	9.71478
shockwave	1.04893	0.590996	0.353622	0.63206	0.68136	0.894689	6.65973
hydrogen	1.11114	0.652397	0.398657	0.399654	0.536743	0.88667	6.79715
lobster	3.67317	1.89733	1.51845	2.8117	3.04032	3.8964	26.7833
mri ventricles	8.30477	4.89567	4.28627	4.83027	6.81417	11.2285	73.0076
engine	4.76578	2.81518	2.43148	4.32637	3.78793	6.09428	42.1098
statue leg	4.31418	2.38684	2.06183	2.4057	3.56858	6.08611	45.3627
tacc turbulence	7.11965	4.74068	4.3593	5.69743	9.19547	16.9212	126.803
aneurism	4.36623	2.46805	2.18222	5.04308	5.74792	7.75388	75.868
bonsai	5.00397	3.05811	2.73404	5.9876	6.62327	8.93747	74.5079
skull	9.50929	6.2636	5.75918	11.1969	13.1809	18.2322	119.585
foot	6.65373	4.21236	3.77614	7.78775	8.97822	11.9677	92.6252
mrt angio	19.5006	12.5129	11.881	24.9724	30.4036	41.7379	286.443
stent	17.6835	12.1931	11.4899	14.781	31.3253	43.4737	311.821
warpx rho (small)	14.3849	10.5728	10.4691	13.8336	23.1061	42.4533	328.7
warpx Ez (small)	16.6937	12.0491	12.1342	15.8755	26.6014	49.1662	371.132
warpx Ey (small)	15.863	11.8892	11.7691	16.1999	27.4346	50.3052	380.82
warpx Ex (small)	16.9818	12.581	12.6317	17.3323	29.4088	54.1208	410.633
pancreas	27.7912	20.5895	20.1015	45.9541	57.4003	79.4803	555.132
bunny	43.2206	32.5811	32.1878	74.6645	92.4612	129.168	916.63
backpack	30.4238	22.9332	22.5796	52.6615	64.3027	89.3918	663.134
present	44.003	33.9845	33.8971	79.9727	96.6904	134.46	975.878
neocort. layer	37.7874	28.8997	28.1961	36.5336	57.0429	101.63	852.669
prone	47.0094	36.8933	37.7486	93.604	118.126	169.279	1188.15
asteroid	23.4236	17.4278	17.7778	23.4432	38.4061	69.7962	608.646
christmas tree	66.927	50.8868	48.9899	61.8208	96.5295	172.081	1298.82
vertebra	37.9014	29.239	29.3374	70.6778	86.6153	125.563	906.271
mag. reconnection	95.3314	73.6443	74.5279	101.657	172.326	318.501	2363.61
marmoset neurons	128.224	103.111	104.406	139.615	229.758	419.654	3182.54
stag beetle	45.6242	38.6227	41.1813	111.131	134.483	183.342	1713.13
pawpawsaurus	OOM	OOM	OOM	OOM	OOM	OOM	OOM
spathorhynchus	OOM	OOM	OOM	OOM	OOM	OOM	OOM
kingsnake	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx rho (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx Ez (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx Ey (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM
warpx Ex (large)	OOM	OOM	OOM	OOM	OOM	OOM	OOM

TABLE S16: PPP2 (aug.) runtime in seconds on KNL for all 3D datasets. We repeated each evaluation 5 times and report here the best time. OOM indicates termination due to insufficient memory (i.e., out-of-memory error).

M.2 Speed up: PPP2 (aug.) for 3D datasets on KNL

	272	136	64	32	16	8	1
marsch. lobb	1.04363	2.52355	4.17777	3.69002	4.25598	3.95892	1
nucleon	1.41604	3.41067	5.23175	3.93612	4.6833	4.44132	1
silicium	2.0642	4.78078	7.27832	5.3357	5.79342	5.12797	1
neghip	3.21326	7.09736	10.5215	6.29383	5.87909	5.82152	1
fuel	4.03434	7.38023	11.8676	7.27415	7.24698	6.693	1
tooth	5.2821	8.56955	12.2989	7.04296	6.98679	5.7865	1
shockwave	6.34907	11.2687	18.8329	10.5365	9.77417	7.44363	1
hydrogen	6.11728	10.4187	17.0501	17.0076	12.6637	7.66593	1
lobster	7.2916	14.1163	17.6386	9.52566	8.80937	6.87386	1
mri ventricles	8.79104	14.9127	17.0329	15.1146	10.7141	6.50199	1
engine	8.83587	14.9581	17.3186	9.73329	11.1168	6.90973	1
statue leg	10.5148	19.0053	22.0012	18.8563	12.7117	7.45348	1
tacc turbulence	17.8103	26.7479	29.0879	22.2562	13.7897	7.49374	1
aneurism	17.3761	30.7401	34.7664	15.044	13.1992	9.78452	1
bonsai	14.8898	24.364	27.2519	12.4437	11.2494	8.33658	1
skull	12.5756	19.0921	20.7642	10.6802	9.0726	6.559	1
foot	13.9208	21.9889	24.5291	11.8937	10.3167	7.7396	1
mrt angio	14.6889	22.8918	24.1093	11.4704	9.42135	6.8629	1
stent	17.6334	25.5736	27.1387	21.0961	9.95429	7.17264	1
warpx rho (small)	22.8504	31.0892	31.3972	23.761	14.2257	7.74263	1
warpx Ez (small)	22.2319	30.8016	30.5856	23.3777	13.9516	7.54852	1
warpx Ey (small)	24.0068	32.0308	32.3576	23.5076	13.881	7.57019	1
warpx Ex (small)	24.1808	32.6391	32.5081	23.6918	13.9629	7.58734	1
pancreas	19.9751	26.9619	27.6164	12.0801	9.67124	6.98452	1
bunny	21.2082	28.1338	28.4776	12.2767	9.91367	7.09642	1
backpack	21.7966	28.9159	29.3687	12.5924	10.3127	7.41829	1
present	22.1775	28.7154	28.7894	12.2026	10.0928	7.25776	1
neocort. layer	22.5649	29.5044	30.2407	23.3393	14.9479	8.38993	1
prone	25.2747	32.205	31.4753	12.6934	10.0583	7.01889	1
asteroid	25.9843	34.9239	34.2363	25.9626	15.8476	8.72033	1
christmas tree	19.4065	25.5237	26.512	21.0094	13.4552	7.54772	1
vertebra	23.9113	30.9953	30.8913	12.8226	10.4632	7.21766	1
mag. reconnection	24.7936	32.0949	31.7144	23.2508	13.7159	7.42104	1
marmoset neurons	24.8202	30.8652	30.4823	22.7951	13.8517	7.58372	1
stag beetle	37.5487	44.3555	41.5997	15.4154	12.7386	9.3439	1
pawpawsaurus	—	—	—	—	—	—	—
spathorhynchus	—	—	—	—	—	—	—
kingsnake	—	—	—	—	—	—	—
warpx rho (large)	—	—	—	—	—	—	—
warpx Ez (large)	—	—	—	—	—	—	—
warpx Ey (large)	—	—	—	—	—	—	—
warpx Ex (large)	—	—	—	—	—	—	—

TABLE S17: PPP2 (aug.) speed up compared to serial on KNL.

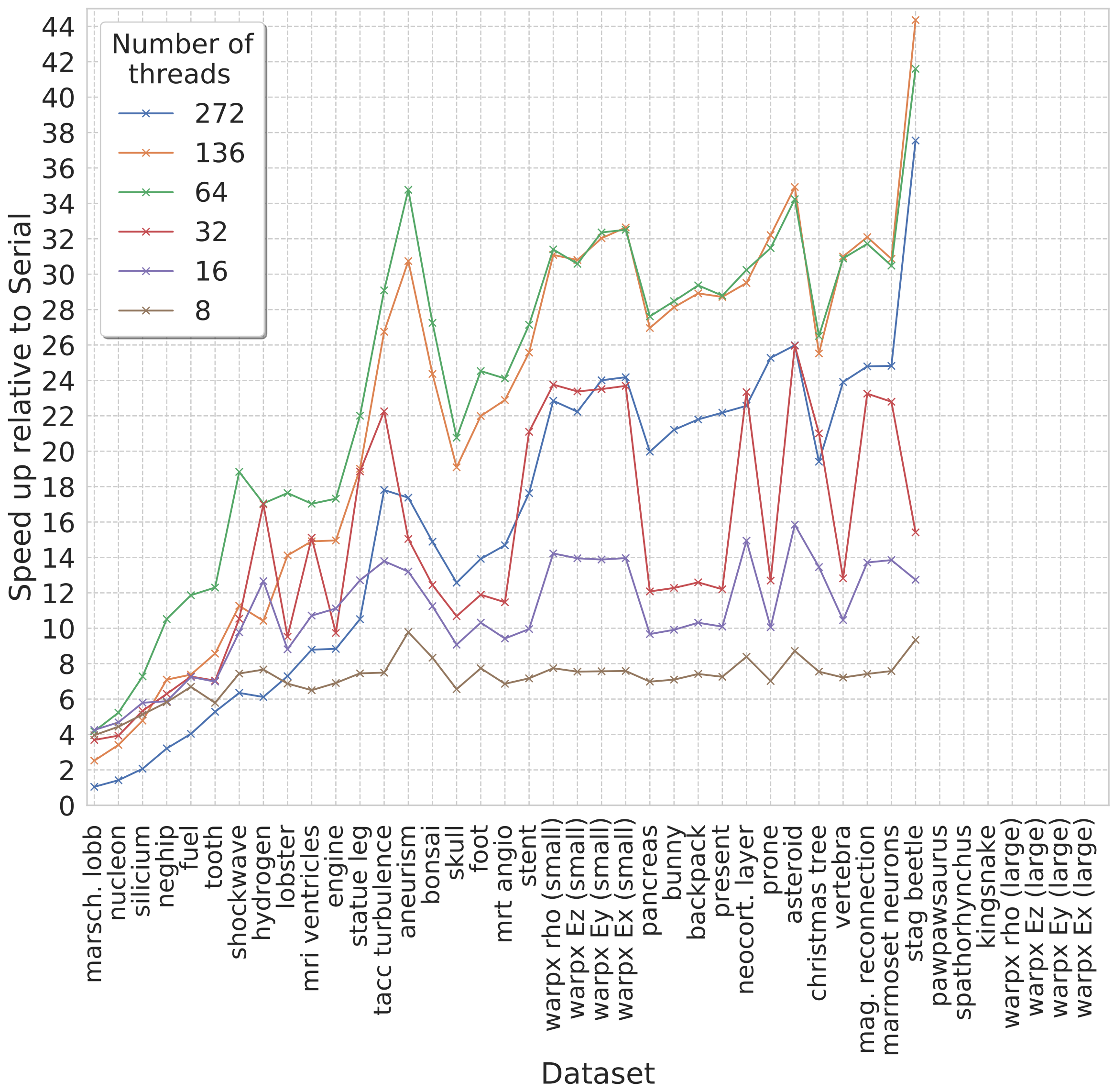


Fig. S7: PPP2 (aug.) speed up compared to serial on KNL. See also Table S17

SUPPLEMENT N
PPP2 (AUG.): 3D DATASETS ON GPU

Type	Runtime	Speed-up compared to serial		Speed-up compared to parallel	
Architecture	GPU	Haswell	KNL	Haswell (64 threads)	KNL (69 threads)
marsch. lobb (41x41x41)	0.0836726	0.529707	2.36834	0.229308	0.58047
nucleon (41x41x41)	0.0648119	0.657009	3.15203	0.195628	0.635984
silicium (98x34x34)	0.0652458	1.11308	5.14559	0.281919	0.722534
neghip (64x64x64)	0.0749097	2.24987	11.0538	0.408257	1.0006
fuel (64x64x64)	0.0619631	2.27824	12.9942	0.397151	1.10982
tooth (103x64x161)	0.304512	6.80055	31.9028	0.927908	2.66056
shockwave (64x64x512)	0.123419	9.41217	53.9603	0.900485	2.85045
hydrogen (128x128x128)	0.114205	11.0732	59.5171	1.15474	3.51957
lobster (301x324x56)	0.416843	13.169	64.2527	1.23951	3.60162
mri ventricles (256x256x124)	1.34234	11.7931	54.3883	1.16182	3.13917
engine (256x256x128)	0.582395	15.226	72.3045	1.382	4.09406
statue leg (341x341x93)	0.521648	17.915	86.9604	1.42235	3.99003
tacc turbulence (256x256x256)	0.920163	28.9799	137.805	1.96399	4.86069
aneurism (256x256x256)	0.433768	31.6868	174.905	1.91727	5.09168
bonsai (256x256x256)	0.570138	25.6719	130.684	1.80198	4.82995
skull (256x256x256)	1.65469	16.8516	72.2703	1.42395	3.51472
foot (256x256x256)	0.947463	19.5957	97.7613	1.46651	4.01465
mrt angio (416x512x112)	4.18093	17.5403	68.5118	1.54793	2.80328
stent (512x512x174)	3.30908	20.1474	94.2319	1.60928	3.42848
warpx rho (425x371x371)	1.95613	37.1626	168.036	2.73704	5.2657
warpx Ez (425x371x371)	2.15401	40.8022	172.298	3.00223	5.5535
warpx Ey (425x371x371)	2.17607	41.2049	175.004	2.98773	5.40833
warpx Ex (425x371x371)	2.40038	39.7352	171.07	2.79872	5.20113
pancreas (240x512x512)	6.76634	19.9659	82.0432	1.74251	2.91882
bunny (512x512x361)	10.8356	21.8148	84.5943	1.96725	2.9306
backpack (512x512x373)	6.47881	24.5888	102.354	1.93252	3.43446
present (492x492x442)	12.0027	20.8344	81.3049	1.89005	2.76979
neocort. layer (1464x1033x76)	9.36919	21.7904	91.0078	1.75058	2.9848
prone (512x512x463)	12.8359	24.1568	92.5646	1.95942	2.88724
asteroid (500x500x500)	3.44921	34.6366	176.46	2.13773	5.09027
christmas tree (512x499x512)	17.8289	19.0323	72.8491	2.01702	2.72267
vertebra (512x512x512)	5.82859	39.1482	155.487	2.51201	4.95756
mag. reconnection (512x512x512)	29.4913	22.5432	80.146	2.02301	2.48741
marmoset neurons (1024x1024x314)	60.3373	13.6603	52.7458	1.40161	1.70429
stag beetle (832x832x494)	18.4938	17.0644	92.6327	1.01012	2.18853
pawpawsaurus (958x646x1088)	356.473	7.837	—	0.521164	—
spathorhynchus (1024x1024x750)	472.074	4.96996	—	0.296407	—
kingsnake (1024x1024x750)	324.983	5.15615	—	0.414782	—
warpx rho (6791x371x371)	—	—	—	—	—
warpx Ez (6791x371x371)	1076.44	1.45363	—	—	—
warpx Ey (6791x371x371)	1097.96	1.47295	—	—	—
warpx Ex (6791x371x371)	1056.42	1.34467	—	—	—
Nyx (1024x1024x1024)	—	—	—	—	—

TABLE S18: PPP2 (aug.) runtime performance on GPU (first columns) and speed-ups compared to Haswell and KNL in serial (second and third column) and in parallel using 64 and 68 threads, respectively (fourth and fifth column).

Supplement O
PPP2 (aug.) on GPU vs TTK on Haswell: 3D datasets

	Runtime in Seconds		Speed-up Factor
	TTK best time Haswell	PPP2 (aug.) GPU	PPP2 (aug.) vs. TKK
marsch. lobb (41x41x41)	0.0204239	0.0836726	0.244093
nucleon (41x41x41)	0.0194559	0.0648119	0.30019
silicium (98x34x34)	0.040611	0.0652458	0.622431
neghip (64x64x64)	0.0380089	0.0749097	0.507396
fuel (64x64x64)	0.0820768	0.0619631	1.32461
tooth (103x64x161)	1.19095	0.304512	3.91101
shockwave (64x64x512)	0.792436	0.123419	6.4207
hydrogen (128x128x128)	0.260882	0.114205	2.28433
lobster (301x324x56)	1.84381	0.416843	4.42327
mri ventricles (256x256x124)	9.06921	1.34234	6.75627
engine (256x256x128)	2.59812	0.582395	4.4611
statue leg (341x341x93)	2.06249	0.521648	3.9538
tacc turbulence (256x256x256)	3.63635	0.920163	3.95185
aneurism (256x256x256)	2.15324	0.433768	4.96404
bonsai (256x256x256)	1.86464	0.570138	3.27051
skull (256x256x256)	9.79057	1.65469	5.91686
foot (256x256x256)	4.62046	0.947463	4.87667
mrt angio (416x512x112)	29.1193	4.18093	6.96479
stent (512x512x174)	35.6884	3.30908	10.785
warpx rho (425x371x371)	16.4503	1.95613	8.40961
warpx Ez (425x371x371)	23.4033	2.15401	10.865
warpx Ey (425x371x371)	10.5818	2.17607	4.8628
warpx Ex (425x371x371)	19.8445	2.40038	8.26723
pancreas (240x512x512)	58.1139	6.76634	8.58868
bunny (512x512x361)	108.009	10.8356	9.96798
backpack (512x512x373)	42.0802	6.47881	6.49505
present (492x492x442)	128.413	12.0027	10.6987
neocort. layer (1464x1033x76)	68.5421	9.36919	7.31569
prone (512x512x463)	80.49	12.8359	6.27069
asteroid (500x500x500)	19.8707	3.44921	5.76094
christmas tree (512x499x512)	131.247	17.8289	7.36147
vertebra (512x512x512)	29.9825	5.82859	5.14404
mag. reconnection (512x512x512)	756.458	29.4913	25.6502
marmoset neurons (1024x1024x314)	—	60.3373	—
stag beetle (832x832x494)	31.6487	18.4938	1.71131
pawpawsaurus (958x646x1088)	—	356.473	—
spathorhynchus (1024x1024x750)	—	472.074	—
kingsnake (1024x1024x750)	—	324.983	—
warpx rho (6791x371x371)	—	—	—
warpx Ez (6791x371x371)	—	1076.44	—
warpx Ey (6791x371x371)	—	1097.96	—
warpx Ex (6791x371x371)	—	1056.42	—
Nyx (1024x1024x1024)	—	—	—

TABLE S19: PPP2 (aug.) on GPU compared to the best time for TTK on Haswell for all 3D datasets.

SUPPLEMENT P
PPP2 (AUG.) AND TTK: 3D DATASETS ON KNL USING 68 THREADS

	Runtime in Seconds		Speed-up Factor
	TTK	PPP2 (aug.)	PPP2 (aug.) vs. TKK
marsch. lobb (41x41x41)	0.133379	0.0485694	2.74615
nucleon (41x41x41)	0.122145	0.0412193	2.9633
silicium (98x34x34)	0.517972	0.0471423	10.9874
neghip (64x64x64)	0.151461	0.0749543	2.02071
fuel (64x64x64)	0.41519	0.068768	6.03755
tooth (103x64x161)	10.0997	0.810173	12.4661
shockwave (64x64x512)	5.01708	0.3518	14.2612
hydrogen (128x128x128)	1.1904	0.401953	2.96154
lobster (301x324x56)	14.3544	1.50131	9.56125
mri ventricles (256x256x124)	73.2515	4.21383	17.3836
engine (256x256x128)	21.7249	2.38436	9.11142
statue leg (341x341x93)	17.3904	2.08139	8.35519
tacc turbulence (256x256x256)	20.2919	4.47263	4.53691
aneurism (256x256x256)	9.79005	2.20861	4.43267
bonsai (256x256x256)	11.272	2.75374	4.09334
skull (256x256x256)	76.3922	5.81578	13.1353
foot (256x256x256)	35.3593	3.80373	9.29595
mrt angio (416x512x112)	215.976	11.7203	18.4275
stent (512x512x174)	187.126	11.3451	16.494
warpx rho (425x371x371)	73.1801	10.3004	7.10459
warpx Ez (425x371x371)	115.892	11.9623	9.6881
warpx Ey (425x371x371)	47.9467	11.7689	4.07402
warpx Ex (425x371x371)	94.67	12.4847	7.58288
pancreas (240x512x512)	305.238	19.7497	15.4553
bunny (512x512x361)	630.242	31.7548	19.8471
backpack (512x512x373)	336.336	22.2512	15.1154
present (492x492x442)	608.947	33.2449	18.317
neocort. layer (1464x1033x76)	607.812	27.9652	21.7346
prone (512x512x463)	557.327	37.0603	15.0384
asteroid (500x500x500)	90.8985	17.5574	5.17722
christmas tree (512x499x512)	890.02	48.5423	18.3349
vertebra (512x512x512)	150.595	28.8956	5.21169
mag. reconnection (512x512x512)	OOM	73.3569	—
marmoset neurons (1024x1024x314)	OOM	102.832	—
stag beetle (832x832x494)	147.646	40.4743	3.6479
pawpawsaurus (958x646x1088)	OOM	OOM	—
spathorhynchus (1024x1024x750)	OOM	OOM	—
kingsnake (1024x1024x750)	OOM	OOM	—
warpx rho (6791x371x371)	OOM	OOM	—
warpx Ez (6791x371x371)	OOM	OOM	—
warpx Ey (6791x371x371)	OOM	OOM	—
warpx Ex (6791x371x371)	OOM	OOM	—
Nyx (1024x1024x1024)	OOM	OOM	—

TABLE S20: PPP2 (aug.) and TTK runtime in seconds on KNL using 68 cores (first two columns) and corresponding speed-up of PPP2 (aug.) compared to TKK (last column) for all 3D datasets. We repeated each evaluation 5 times and report here the best time. OOM indicates termination due to insufficient memory (i.e., out-of-memory error).

SUPPLEMENT Q
PPP2 (AUG.) AND TTK: 3D DATASETS ON KNL USING 272 THREADS

	Runtime in Seconds		Speed-up Factor
	TTK	PPP2 (aug.)	PPP2 (aug.) vs. TKK
marsch. lobb (41x41x41)	0.943476	0.189881	4.96878
nucleon (41x41x41)	0.506342	0.144268	3.50973
silicium (98x34x34)	0.783547	0.162643	4.81759
neghip (64x64x64)	0.601298	0.257693	2.33339
fuel (64x64x64)	0.590389	0.199577	2.9582
tooth (103x64x161)	90.4828	1.83919	49.1971
shockwave (64x64x512)	5.49364	1.04893	5.23738
hydrogen (128x128x128)	3.32058	1.11114	2.98844
lobster (301x324x56)	123.217	3.67317	33.5451
mri ventricles (256x256x124)	601.307	8.30477	72.405
engine (256x256x128)	168.677	4.76578	35.3934
statue leg (341x341x93)	138.453	4.31418	32.0925
tacc turbulence (256x256x256)	159.622	7.11965	22.4199
aneurism (256x256x256)	47.0601	4.36623	10.7782
bonsai (256x256x256)	77.0497	5.00397	15.3977
skull (256x256x256)	642.938	9.50929	67.6116
foot (256x256x256)	288.926	6.65373	43.4232
mrt angio (416x512x112)	1995.33	19.5006	102.321
stent (512x512x174)	1159.88	17.6835	65.5911
warpx rho (425x371x371)	76.1957	14.3849	5.29692
warpx Ez (425x371x371)	128.856	16.6937	7.71884
warpx Ey (425x371x371)	56.6231	15.863	3.56951
warpx Ex (425x371x371)	196.518	16.9818	11.5723
pancreas (240x512x512)	2481.49	27.7912	89.2905
bunny (512x512x361)	4256.58	43.2206	98.485
backpack (512x512x373)	2735.43	30.4238	89.9109
present (492x492x442)	5020.31	44.003	114.09
neocort. layer (1464x1033x76)	4980.15	37.7874	131.794
prone (512x512x463)	4771.93	47.0094	101.51
asteroid (500x500x500)	409.767	23.4236	17.4938
christmas tree (512x499x512)	7744.3	66.927	115.713
vertebra (512x512x512)	1321.15	37.9014	34.8576
mag. reconnection (512x512x512)	12431.0	95.3314	130.398
marmoset neurons (1024x1024x314)	OOM	128.224	—
stag beetle (832x832x494)	466.154	45.6242	10.2173
pawpawsaurus (958x646x1088)	OOM	OOM	—
spathorhynchus (1024x1024x750)	OOM	OOM	—
kingsnake (1024x1024x750)	OOM	OOM	—
warpx rho (6791x371x371)	OOM	OOM	—
warpx Ez (6791x371x371)	OOM	OOM	—
warpx Ey (6791x371x371)	OOM	OOM	—
warpx Ex (6791x371x371)	OOM	OOM	—
Nyx (1024x1024x1024)	OOM	OOM	—

TABLE S21: PPP2 (aug.) and TTK runtime in seconds on KNL using 272 threads (first two columns) and corresponding speed-up of PPP2 (aug.) compared to TKK (last column) for all 3D datasets. The KNL chip has 68 cores but supports 4 hyper-threads. We repeated each evaluation 5 times and report here the best time. OOM indicates termination due to insufficient memory (i.e., out-of-memory error).

Supplement R
Hyperstructure statistics

R.1 Summary Plots

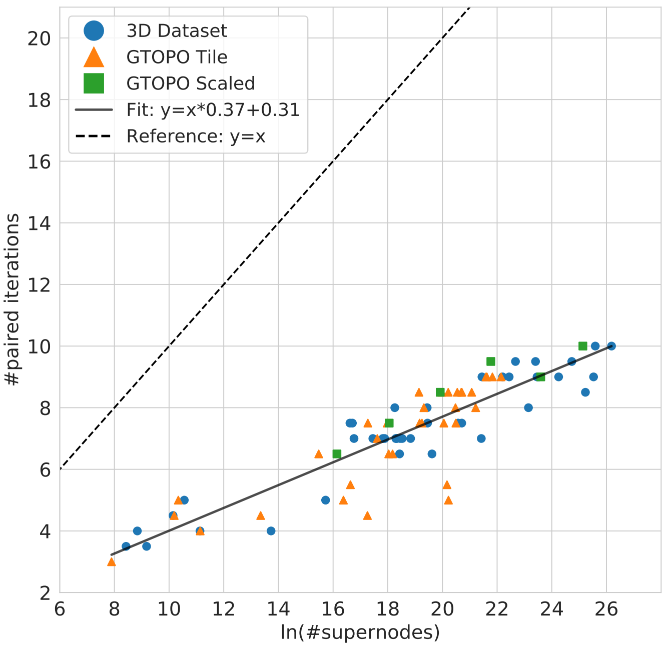


Fig. S8: The collapse of a perfectly balanced tree would require $ln(\#supernodes)$ pruning steps, with each step corresponding to a pruning of both upper and lower leafs. Since PPP2 alternates in each iteration between pruning either upper or lower leafs, a pruning step, hence corresponds to a pair of iterations in PPP2. The scatter plot compares the expected number of pruning steps versus the actual number of pruning steps required for all test datasets. We observe that in practice the number of pruning steps required is for all datasets significantly smaller than the $ln(\#supernodes)$ estimate and PPP2 requires at most 10 paired iterations for any of the datasets.

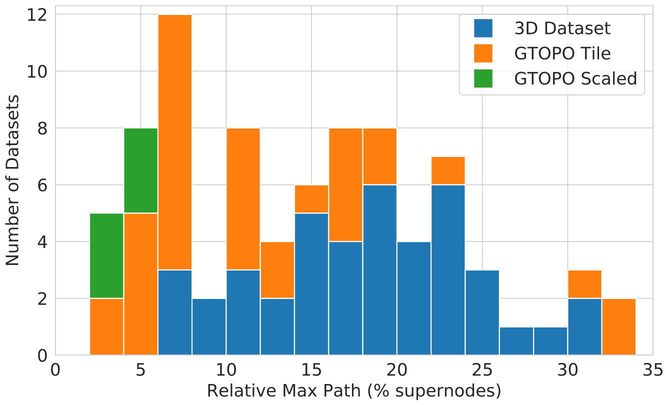


Fig. S9: Histogram showing the relative lengths of the longest hyperarc for all datasets. We observe that in most cases the longest hyperarc contains at least 5%, and in many cases 10% to 30%, of the total number of supernodes.

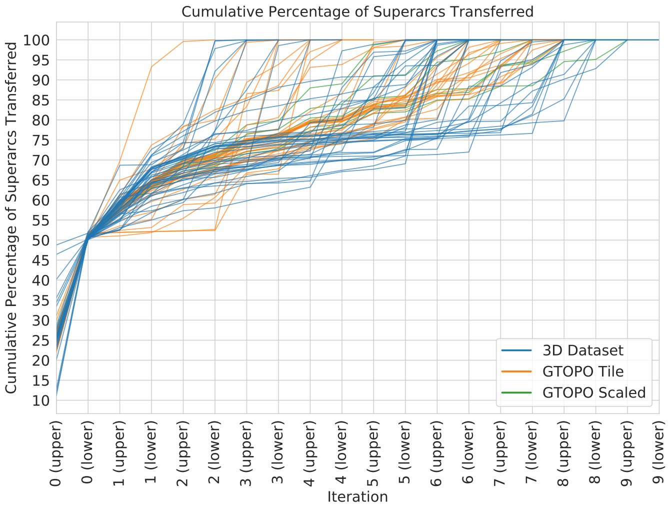


Fig. S10: Cumulative percentage of superarcs transferred across iterations of PPP2 for all datasets. We observe that roughly 50% to 55% of all supernodes are being transferred in the first iteration pair (i.e., iteration 0 upper/lower). After, the leaf supernodes have been transferred in the first iteration pair, the rate at which the algorithm can transfer supernodes often slows down. In many cases we then see large jumps later on, indicating that many supernodes are being transferred at once in some later iteration (which in turn typically correspond to long hyperarcs).

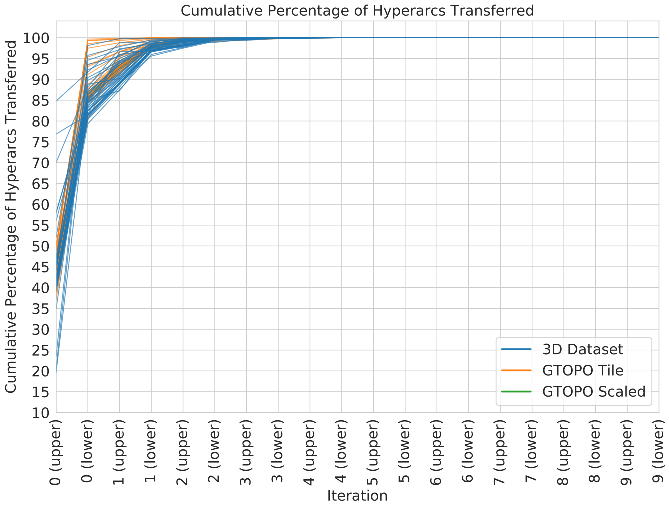


Fig. S11: Cumulative percentage of hyperarcs transferred across iterations of PPP2 for all datasets. We observe that by the end of the first iteration pair $\approx 80\%$, and by the end of the second iteration pair more than 95%, of all hyperacrs have been transferred for all datasets.

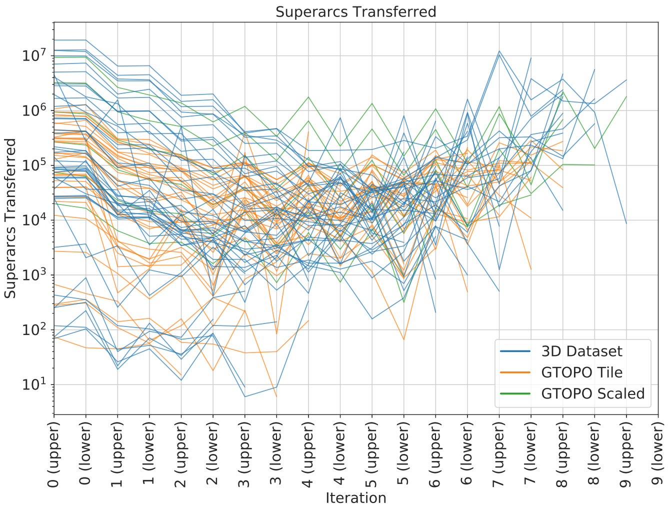


Fig. S12: Number of superarcs transferred per iteration for iterations $[0, i-1]$ for all datasets. In the last iterations only 1 superarc is being transferred in all cases. To reduce visual clutter we, therefore, only show iterations $[0, i-1]$.

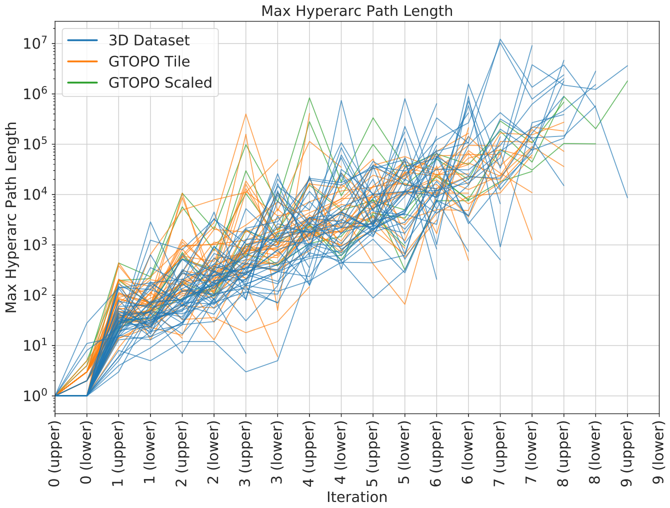


Fig. S14: Maximum hyperarc path length in number of supernodes per iteration for iterations iterations $[0, i-1]$ for all datasets. In the last iteration only 1 hyperarcs of length 1 is being transferred in all cases. To reduce visual clutter we, therefore, only show iterations $[0, i-1]$.

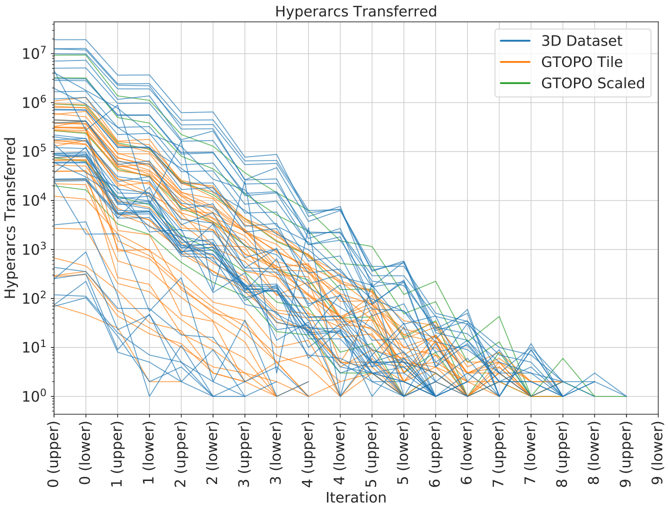


Fig. S13: Number of hyperarcs transferred per iteration for iterations $[0, i-1]$ for all datasets. In the last iteration only 1 hyperarc is being transferred in all cases. To reduce visual clutter we, therefore, only show iterations $[0, i-1]$.

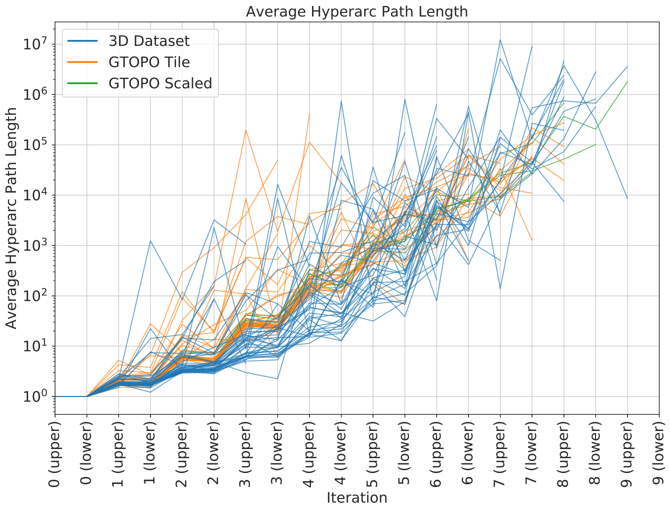


Fig. S15: Average hyperarc path length in number of supernodes per iteration for iterations iterations $[0, i-1]$ for all datasets. In the last iteration only 1 hyperarcs of length 1 is being transferred in all cases. To reduce visual clutter we, therefore, only show iterations $[0, i-1]$.

R.2 Hyperstructure statistics: 3D Datasets

marsch. lobb (41x41x41)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	431	431	1	1.00	1
0 (lower)	354	354	1	1.00	1
1 (upper)	63	119	1	1.89	8
1 (lower)	62	104	1	1.68	5
2 (upper)	18	67	1	3.72	12
2 (lower)	16	78	1	4.88	12
3 (upper)	2	6	3	3.00	3
3 (lower)	4	9	1	2.25	5
4 (upper)	1	337	337	337.00	337
4 (lower)	1	1	1	1.00	1
Total	952	1,506	1	1.58	1

nucleon (41x41x41)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	74	74	1	1.00	1
0 (lower)	224	224	1	1.00	1
1 (upper)	8	19	1	2.38	6
1 (lower)	5	71	1	14.20	67
2 (upper)	2	34	6	17.00	28
2 (lower)	1	156	156	156.00	156
3 (upper)	1	1	1	1.00	1
Total	315	579	1	1.84	1

silicium (98x34x34)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	119	119	1	1.00	1
0 (lower)	111	111	1	1.00	1
1 (upper)	20	44	1	2.20	5
1 (lower)	7	52	1	7.43	45
2 (upper)	5	36	1	7.20	17
2 (lower)	1	86	86	86.00	86
3 (upper)	2	9	2	4.50	7
3 (lower)	1	1	1	1.00	1
Total	266	458	1	1.72	1

neghip (64x64x64)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	249	249	1	1.00	1
0 (lower)	892	892	1	1.00	1
1 (upper)	23	40	1	1.74	6
1 (lower)	46	94	1	2.04	19
2 (upper)	5	73	5	14.60	26
2 (lower)	2	386	43	193.00	343
3 (upper)	1	507	507	507.00	507
3 (lower)	1	1	1	1.00	1
Total	1,219	2,242	1	1.84	1

fuel (64x64x64)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	69	69	1	1.00	1
0 (lower)	104	104	1	1.00	1
1 (upper)	17	26	1	1.53	3
1 (lower)	2	45	1	22.50	44
2 (upper)	4	12	1	3.00	7
2 (lower)	1	87	87	87.00	87
3 (upper)	1	1	1	1.00	1
Total	198	344	1	1.74	1

tooth (103x64x161)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	58,602	58,602	1	1.00	1
0 (lower)	58,986	58,986	1	1.00	1
1 (upper)	9,185	15,158	1	1.65	13
1 (lower)	9,504	16,471	1	1.73	15
2 (upper)	1,103	3,452	1	3.13	28
2 (lower)	1,311	4,876	1	3.72	183
3 (upper)	103	663	1	6.44	31
3 (lower)	163	2,140	1	13.13	155
4 (upper)	6	23,156	1	3,859.33	23,146
4 (lower)	23	1,593	1	69.26	327
5 (upper)	1	36,424	36,424	36,424.00	36,424
5 (lower)	6	1,784	17	297.33	1,070
6 (upper)	1	7,936	7,936	7,936.00	7,936
6 (lower)	1	1	1	1.00	1
Total	138,995	231,242	1	1.66	1

shockwave (64x64x512)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	255	255	1	1.00	1
0 (lower)	318	318	1	1.00	1
1 (upper)	9	22	1	2.44	4
1 (lower)	48	132	1	2.75	9
2 (upper)	2	29	3	14.50	26
2 (lower)	9	120	2	13.33	30
3 (upper)	1	116	116	116.00	116
3 (lower)	2	140	68	70.00	72
4 (upper)	1	1	1	1.00	1
Total	645	1,133	1	1.76	1

hydrogen (128x128x128)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	3,190	3,190	1	1.00	1
0 (lower)	3,682	3,682	1	1.00	1
1 (upper)	117	258	1	2.21	20
1 (lower)	1	1,242	1,242	1,242.00	1,242
2 (upper)	11	928	1	84.36	803
2 (lower)	1	3,225	3,225	3,225.00	3,225
3 (upper)	1	1,067	1,067	1,067.00	1,067
3 (lower)	1	1	1	1.00	1
Total	7,004	13,593	1	1.94	1

lobster (301x324x56)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	85,268	85,268	1	1.00	1
0 (lower)	77,558	77,558	1	1.00	1
1 (upper)	12,078	20,464	1	1.69	16
1 (lower)	9,809	14,627	1	1.49	13
2 (upper)	1,645	5,532	1	3.36	29
2 (lower)	580	1,919	1	3.31	310
3 (upper)	196	1,931	1	9.85	125
3 (lower)	23	529	1	23.00	292
4 (upper)	29	1,543	1	53.21	852
4 (lower)	1	61,354	61,354	61,354.00	61,354
5 (upper)	7	5,017	1	716.71	4,803
5 (lower)	1	46,770	46,770	46,770.00	46,770
6 (upper)	2	836	8	418.00	828
6 (lower)	1	1	1	1.00	1
Total	187,198	323,349	1	1.73	1

mri ventricles (256x256x124)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	383,162	383,162	1	1.00	1
0 (lower)	416,779	416,779	1	1.00	1
1 (upper)	54,579	91,948	1	1.68	43
1 (lower)	64,780	110,385	1	1.70	15
2 (upper)	6,657	24,015	1	3.61	65
2 (lower)	8,558	30,109	1	3.52	42
3 (upper)	684	9,868	1	14.43	367
3 (lower)	1,036	10,745	1	10.37	322
4 (upper)	78	9,394	1	120.44	2,760
4 (lower)	115	7,774	1	67.60	1,064
5 (upper)	14	10,885	12	777.50	5,323
5 (lower)	15	18,137	8	1,209.13	11,837
6 (upper)	1	333,994	333,994	333,994.00	333,994
6 (lower)	2	105,242	260	52,621.00	104,982
7 (upper)	1	1	1	1.00	1
Total	936,461	1,562,438	1	1.67	1

engine (256x256x128)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	120,929	120,929	1	1.00	1
0 (lower)	115,489	115,489	1	1.00	1
1 (upper)	17,205	28,498	1	1.66	30
1 (lower)	14,302	23,509	1	1.64	37
2 (upper)	1,839	6,414	1	3.49	65
2 (lower)	1,387	5,693	1	4.10	98
3 (upper)	160	1,695	1	10.59	167
3 (lower)	138	3,620	1	26.23	295
4 (upper)	15	1,280	2	85.33	235
4 (lower)	15	6,627	5	441.80	1,729
5 (upper)	3	2,844	47	948.00	2,745
5 (lower)	3	6,424	198	2,141.33	4,710
6 (upper)	1	144,679	144,679	144,679.00	144,679
6 (lower)	1	1	1	1.00	1
Total	271,487	467,702	1	1.72	1

statue leg (341x341x93)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	142,150	142,150	1	1.00	1
0 (lower)	36,793	36,793	1	1.00	1
1 (upper)	18,677	35,879	1	1.92	31
1 (lower)	2,284	3,541	1	1.55	129
2 (upper)	2,576	12,741	1	4.95	78
2 (lower)	132	1,253	1	9.49	417
3 (upper)	335	5,806	1	17.33	259
3 (lower)	14	13,345	1	953.21	9,829
4 (upper)	33	4,192	2	127.03	1,272
4 (lower)	5	88,439	7	17,687.80	85,604
5 (upper)	2	5,809	1,083	2,904.50	4,726
5 (lower)	1	3,928	3,928	3,928.00	3,928
6 (upper)	1	1	1	1.00	1
Total	203,003	353,877	1	1.74	1

tacc turbulence (256x256x256)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	36,937	36,937	1	1.00	1
0 (lower)	123,219	123,219	1	1.00	1
1 (upper)	4,379	12,344	1	2.82	48
1 (lower)	18,917	35,082	1	1.85	27
2 (upper)	834	5,845	1	7.01	170
2 (lower)	3,313	15,790	1	4.77	55
3 (upper)	139	2,265	1	16.30	158
3 (lower)	646	8,659	1	13.40	120
4 (upper)	19	1,124	1	59.16	193
4 (lower)	121	5,391	2	44.55	452
5 (upper)	4	884	62	221.00	441
5 (lower)	26	2,659	1	102.27	623
6 (upper)	1	58,809	58,809	58,809.00	58,809
6 (lower)	3	3,767	160	1,255.67	2,995
7 (upper)	1	505	505	505.00	505
7 (lower)	1	1	1	1.00	1
Total	188,560	313,281	1	1.66	1

aneurism (256x256x256)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	25,158	25,158	1	1.00	1
0 (lower)	2,062	2,069	1	1.00	8
1 (upper)	2,092	3,417	1	1.63	23
1 (lower)	55	424	1	7.71	354
2 (upper)	266	1,108	1	4.17	177
2 (lower)	2	4,516	12	2,258.00	4,504
3 (upper)	36	317	1	8.81	80
3 (lower)	1	16,413	16,413	16,413.00	16,413
4 (upper)	2	774	2	387.00	772
4 (lower)	1	1	1	1.00	1
Total	29,675	54,197	1	1.83	1

bonsai (256x256x256)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	60,016	60,016	1	1.00	1
0 (lower)	30,347	30,347	1	1.00	1
1 (upper)	11,491	19,897	1	1.73	17
1 (lower)	2,932	5,080	1	1.73	161
2 (upper)	1,779	5,553	1	3.12	33
2 (lower)	303	1,432	1	4.73	375
3 (upper)	196	1,377	1	7.03	262
3 (lower)	33	2,285	1	69.24	781
4 (upper)	23	463	1	20.13	164
4 (lower)	4	31,762	3	7,940.50	31,680
5 (upper)	3	15,852	2,487	5,284.00	10,357
5 (lower)	1	518	518	518.00	518
6 (upper)	2	5,044	124	2,522.00	4,920
6 (lower)	1	1	1	1.00	1
Total	107,131	179,627	1	1.68	1

skull (256x256x256)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	442,557	442,557	1	1.00	1
0 (lower)	423,008	423,008	1	1.00	1
1 (upper)	62,651	110,682	1	1.77	76
1 (lower)	55,770	98,654	1	1.77	57
2 (upper)	8,825	35,700	1	4.05	76
2 (lower)	7,102	30,143	1	4.24	248
3 (upper)	1,188	16,038	1	13.50	346
3 (lower)	784	17,024	1	21.71	1,224
4 (upper)	171	10,941	1	63.98	1,053
4 (lower)	66	9,894	1	149.91	3,241
5 (upper)	25	6,447	5	257.88	1,452
5 (lower)	8	37,075	28	4,634.38	35,006
6 (upper)	2	53,207	459	26,603.50	52,748
6 (lower)	1	419,106	419,106	419,106.00	419,106
7 (upper)	1	1	1	1.00	1
Total	1,002,159	1,710,477	1	1.71	1

foot (256x256x256)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	200,798	200,798	1	1.00	1
0 (lower)	163,071	163,071	1	1.00	1
1 (upper)	30,751	52,341	1	1.70	81
1 (lower)	24,975	43,154	1	1.73	17
2 (upper)	4,113	14,224	1	3.46	325
2 (lower)	3,483	12,125	1	3.48	134
3 (upper)	422	6,062	1	14.36	766
3 (lower)	398	15,158	1	38.09	10,881
4 (upper)	41	6,603	1	161.05	2,460
4 (lower)	43	27,473	1	638.91	24,388
5 (upper)	9	4,302	19	478.00	1,524
5 (lower)	5	20,968	10	4,193.60	20,169
6 (upper)	3	9,600	196	3,200.00	8,573
6 (lower)	1	143,211	143,211	143,211.00	143,211
7 (upper)	1	1	1	1.00	1
Total	428,114	719,091	1	1.68	1

mrt angio (416x512x112)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	1,189,720	1,189,720	1	1.00	1
0 (lower)	1,277,556	1,277,556	1	1.00	1
1 (upper)	222,819	393,396	1	1.77	15
1 (lower)	236,087	415,943	1	1.76	33
2 (upper)	38,889	123,649	1	3.18	48
2 (lower)	40,443	129,136	1	3.19	552
3 (upper)	5,573	34,577	1	6.20	166
3 (lower)	5,821	37,077	1	6.37	291
4 (upper)	581	10,659	1	18.35	543
4 (lower)	739	14,041	1	19.00	620
5 (upper)	44	10,028	1	227.91	2,195
5 (lower)	78	25,422	2	325.92	11,202
6 (upper)	5	21,619	1,037	4,323.80	12,050
6 (lower)	14	41,489	26	2,963.50	22,646
7 (upper)	2	21,642	553	10,821.00	21,089
7 (lower)	4	177,955	899	44,488.80	166,784
8 (upper)	1	894,429	894,429	894,429.00	894,429
8 (lower)	1	1	1	1.00	1
Total	3,018,377	4,818,339	1	1.60	1

stent (512x512x174)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	732,853	732,853	1	1.00	1
0 (lower)	715,803	715,813	1	1.00	11
1 (upper)	120,649	200,090	1	1.66	15
1 (lower)	120,132	206,725	1	1.72	76
2 (upper)	16,791	51,054	1	3.04	80
2 (lower)	18,386	62,510	1	3.40	168
3 (upper)	2,267	16,925	1	7.47	262
3 (lower)	2,688	25,880	1	9.63	806
4 (upper)	333	11,904	1	35.75	1,273
4 (lower)	398	17,636	1	44.31	2,238
5 (upper)	61	10,406	1	170.59	2,343
5 (lower)	56	85,945	1	1,534.73	54,268
6 (upper)	13	13,449	1	1,034.54	5,123
6 (lower)	9	130,615	9	14,512.80	95,126
7 (upper)	3	425,048	60	141,683.00	424,575
7 (lower)	3	134,878	773	44,959.30	124,937
8 (upper)	2	15,093	58	7,546.50	15,035
8 (lower)	1	1	1	1.00	1
Total	1,730,448	2,856,825	1	1.65	1

warpx rho (425x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	26,524	26,524	1	1.00	1
0 (lower)	27,015	27,015	1	1.00	1
1 (upper)	4,369	11,285	1	2.58	30
1 (lower)	4,489	11,386	1	2.54	23
2 (upper)	815	5,097	1	6.25	123
2 (lower)	806	3,937	1	4.88	106
3 (upper)	160	2,694	1	16.84	140
3 (lower)	142	1,372	1	9.66	70
4 (upper)	26	1,878	2	72.23	1,000
4 (lower)	19	1,283	1	67.53	423
5 (upper)	5	1,786	20	357.20	1,296
5 (lower)	5	700	7	140.00	279
6 (upper)	1	7,635	7,635	7,635.00	7,635
6 (lower)	2	4,315	441	2,157.50	3,874
7 (upper)	1	1	1	1.00	1
Total	64,379	106,908	1	1.66	1

warpx Ez (425x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	27,467	27,467	1	1.00	1
0 (lower)	28,265	28,265	1	1.00	1
1 (upper)	5,376	10,497	1	1.95	37
1 (lower)	5,854	11,097	1	1.90	26
2 (upper)	896	3,436	1	3.83	74
2 (lower)	1,127	4,273	1	3.79	66
3 (upper)	91	1,149	1	12.63	213
3 (lower)	187	2,822	1	15.09	409
4 (upper)	10	1,647	4	164.70	1,071
4 (lower)	30	1,644	1	54.80	927
5 (upper)	3	2,660	1	886.67	2,452
5 (lower)	4	3,386	2	846.50	1,709
6 (upper)	1	13,052	13,052	13,052.00	13,052
6 (lower)	1	1	1	1.00	1
Total	69,312	111,396	1	1.61	1

warpx Ey (425x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	24,707	24,707	1	1.00	1
0 (lower)	25,539	25,539	1	1.00	1
1 (upper)	5,171	11,356	1	2.20	32
1 (lower)	5,368	11,421	1	2.13	35
2 (upper)	1,069	4,490	1	4.20	50
2 (lower)	1,102	4,790	1	4.35	102
3 (upper)	152	2,964	1	19.50	260
3 (lower)	167	3,305	1	19.79	168
4 (upper)	25	1,473	1	58.92	697
4 (lower)	24	1,108	1	46.17	476
5 (upper)	5	158	1	31.60	88
5 (lower)	5	382	5	76.40	314
6 (upper)	1	7,775	7,775	7,775.00	7,775
6 (lower)	2	998	255	499.00	743
7 (upper)	1	1	1	1.00	1
Total	63,338	100,467	1	1.59	1

warpx Ex (425x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	89,763	89,763	1	1.00	1
0 (lower)	89,665	89,665	1	1.00	1
1 (upper)	14,228	27,121	1	1.91	44
1 (lower)	13,863	26,603	1	1.92	45
2 (upper)	2,449	9,417	1	3.85	43
2 (lower)	2,318	9,482	1	4.09	72
3 (upper)	370	4,086	1	11.04	117
3 (lower)	343	3,388	1	9.88	105
4 (upper)	42	4,448	1	105.91	3,511
4 (lower)	44	1,585	1	36.02	828
5 (upper)	5	3,115	15	623.00	1,908
5 (lower)	4	11,373	9	2,843.25	9,249
6 (upper)	1	78,156	78,156	78,156.00	78,156
6 (lower)	1	1	1	1.00	1
Total	213,096	358,203	1	1.68	1

pancreas (240x512x512)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	1,668,761	1,668,761	1	1.00	1
0 (lower)	1,708,812	1,708,812	1	1.00	1
1 (upper)	312,276	553,437	1	1.77	25
1 (lower)	324,615	586,364	1	1.81	637
2 (upper)	53,629	180,161	1	3.36	60
2 (lower)	58,224	205,393	1	3.53	196
3 (upper)	8,424	61,697	1	7.32	204
3 (lower)	9,716	78,677	1	8.10	1,700
4 (upper)	1,254	30,143	1	24.04	1,855
4 (lower)	1,547	47,714	1	30.84	5,677
5 (upper)	172	31,606	1	183.76	10,215
5 (lower)	221	42,397	1	191.84	11,563
6 (upper)	22	9,777	11	444.41	4,249
6 (lower)	35	103,692	10	2,962.63	15,355
7 (upper)	3	319,844	109	106,615.00	318,514
7 (lower)	10	331,070	48	33,107.00	133,734
8 (upper)	2	146,477	322	73,238.50	146,155
8 (lower)	1	576,608	576,608	576,608.00	576,608
9 (upper)	1	1	1	1.00	1
Total	4,147,725	6,682,631	1	1.61	1

bunny (512x512x361)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	2,838,433	2,838,433	1	1.00	1
0 (lower)	2,808,011	2,808,011	1	1.00	1
1 (upper)	565,885	971,368	1	1.72	152
1 (lower)	562,713	975,797	1	1.73	29
2 (upper)	95,426	278,481	1	2.92	62
2 (lower)	97,524	296,650	1	3.04	236
3 (upper)	12,605	70,238	1	5.57	600
3 (lower)	14,325	96,231	1	6.72	1,289
4 (upper)	1,261	21,505	1	17.05	2,047
4 (lower)	1,819	63,016	1	34.64	4,488
5 (upper)	75	8,815	1	117.53	1,923
5 (lower)	238	58,320	1	245.04	11,910
6 (upper)	4	25,565	123	6,391.25	9,017
6 (lower)	31	59,555	1	1,921.13	20,061
7 (upper)	1	71,645	71,645	71,645.00	71,645
7 (lower)	9	365,136	27	40,570.70	211,601
8 (upper)	1	463,238	463,238	463,238.00	463,238
8 (lower)	2	1,638,778	107,693	819,389.00	1,531,085
9 (upper)	1	1	1	1.00	1
Total	6,998,364	11,110,783	1	1.59	1

neocort. layer (1464x1033x76)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	4,530,953	4,530,953	1	1.00	1
0 (lower)	274,435	274,435	1	1.00	1
1 (upper)	890,996	1,575,333	1	1.77	35
1 (lower)	8,443	10,292	1	1.22	19
2 (upper)	162,689	513,528	1	3.16	86
2 (lower)	145	410	1	2.83	162
3 (upper)	24,852	151,843	1	6.11	469
3 (lower)	3	25,881	1	8,627.00	25,878
4 (upper)	2,807	47,350	1	16.87	599
4 (lower)	1	737,383	737,383	737,383.00	737,383
5 (upper)	211	14,528	1	68.85	2,633
5 (lower)	1	803,647	803,647	803,647.00	803,647
6 (upper)	12	10,558	6	879.83	9,066
6 (lower)	1	585,415	585,415	585,415.00	585,415
7 (upper)	2	7,757	1,185	3,878.50	6,572
7 (lower)	1	1	1	1.00	1
Total	5,895,552	9,289,314	1	1.58	1

backpack (512x512x373)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	1,959,245	1,959,245	1	1.00	1
0 (lower)	928,147	928,148	1	1.00	2
1 (upper)	312,680	519,833	1	1.66	60
1 (lower)	116,793	203,311	1	1.74	64
2 (upper)	45,439	137,423	1	3.02	513
2 (lower)	15,189	73,301	1	4.83	295
3 (upper)	6,316	37,973	1	6.01	185
3 (lower)	2,063	47,315	1	22.93	3,749
4 (upper)	990	15,653	1	15.81	348
4 (lower)	281	60,253	1	214.42	27,011
5 (upper)	186	10,974	1	59.00	2,489
5 (lower)	30	140,551	6	4,685.03	131,730
6 (upper)	44	3,505	1	79.66	1,016
6 (lower)	2	896,861	5,791	448,430.00	891,070
7 (upper)	9	1,249	1	138.78	916
7 (lower)	1	268,623	268,623	268,623.00	268,623
8 (upper)	2	389,049	198	194,524.00	388,851
8 (lower)	1	1	1	1.00	1
Total	3,387,418	5,693,268	1	1.68	1

prone (512x512x463)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	3,074,489	3,074,489	1	1.00	1
0 (lower)	3,060,968	3,060,968	1	1.00	1
1 (upper)	541,886	950,738	1	1.75	21
1 (lower)	552,040	983,743	1	1.78	172
2 (upper)	89,011	295,714	1	3.32	74
2 (lower)	94,420	327,227	1	3.47	674
3 (upper)	13,997	98,905	1	7.07	406
3 (lower)	15,192	120,452	1	7.93	2,277
4 (upper)	2,128	44,325	1	20.83	1,800
4 (lower)	2,279	56,230	1	24.67	2,766
5 (upper)	292	26,217	1	89.78	2,575
5 (lower)	291	49,763	1	171.01	11,919
6 (upper)	43	68,110	2	1,583.95	31,944
6 (lower)	30	63,513	6	2,117.10	30,995
7 (upper)	7	137,266	15	19,609.40	95,407
7 (lower)	5	720,851	15	144,170.00	624,198
8 (upper)	1	2,009,371	2,009,371	2,009,370.00	2,009,371
8 (lower)	1	1	1	1.00	1
Total	7,447,080	12,087,883	1	1.62	1

present (492x492x442)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	4,104,001	4,104,001	1	1.00	1
0 (lower)	1,783,906	1,783,906	1	1.00	1
1 (upper)	770,588	1,334,760	1	1.73	20
1 (lower)	231,234	377,146	1	1.63	24
2 (upper)	128,108	383,334	1	2.99	287
2 (lower)	27,355	88,138	1	3.22	3,354
3 (upper)	16,610	88,732	1	5.34	546
3 (lower)	3,017	44,456	1	14.74	20,448
4 (upper)	1,422	22,073	1	15.52	2,620
4 (lower)	354	70,809	1	200.03	54,118
5 (upper)	97	10,347	1	106.67	2,280
5 (lower)	55	390,482	1	7,099.67	225,918
6 (upper)	8	15,979	1	1,997.38	7,028
6 (lower)	11	923,444	7	83,949.50	728,853
7 (upper)	2	17,678	4,876	8,839.00	12,802
7 (lower)	2	81,059	72	40,529.50	80,987
8 (upper)	1	1,811,613	1,811,613	1,811,610.00	1,811,613
8 (lower)	1	1	1	1.00	1
Total	7,066,772	11,547,958	1	1.63	1

asteroid (500x500x500)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	218,927	218,927	1	1.00	1
0 (lower)	184,184	184,184	1	1.00	1
1 (upper)	8,811	24,483	1	2.78	143
1 (lower)	6,737	15,042	1	2.23	130
2 (upper)	1,229	18,724	1	15.24	524
2 (lower)	784	5,698	1	7.27	169
3 (upper)	177	14,045	1	79.35	1,196
3 (lower)	48	15,621	1	325.44	14,522
4 (upper)	22	11,809	9	536.77	4,961
4 (lower)	3	107,270	12	35,756.70	107,239
5 (upper)	5	13,542	223	2,708.40	9,665
5 (lower)	1	175,411	175,411	175,411.00	175,411
6 (upper)	1	1	1	1.00	1
Total	420,929	804,757	1	1.91	1

christmas tree (512x499x512)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	5,043,404	5,043,404	1	1.00	1
0 (lower)	5,120,184	5,120,185	1	1.00	2
1 (upper)	962,265	1,696,398	1	1.76	20
1 (lower)	978,319	1,740,623	1	1.78	186
2 (upper)	163,162	516,377	1	3.16	61
2 (lower)	168,267	552,928	1	3.29	922
3 (upper)	21,839	140,600	1	6.44	309
3 (lower)	22,503	163,634	1	7.27	2,636
4 (upper)	2,273	42,070	1	18.51	1,722
4 (lower)	2,188	48,491	1	22.16	4,594
5 (upper)	226	15,622	1	69.12	2,032
5 (lower)	184	14,993	1	81.48	4,312
6 (upper)	26	74,070	1	2,848.85	68,920
6 (lower)	19	7,956	1	418.74	2,637
7 (upper)	5	42,136	1,955	8,427.20	15,935
7 (lower)	3	80,149	3,080	26,716.30	68,467
8 (upper)	1	4,663,202	4,663,202	4,663,200.00	4,663,202
8 (lower)	1	1	1	1.00	1
Total	12,484,869	19,962,839	1	1.60	1

marmoset neurons (1024x1024x314)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	12,500,240	12,500,240	1	1.00	1
0 (lower)	11,914,404	11,914,404	1	1.00	1
1 (upper)	2,271,746	3,770,564	1	1.66	138
1 (lower)	2,206,798	3,645,308	1	1.65	20
2 (upper)	289,320	938,551	1	3.24	661
2 (lower)	292,900	858,662	1	2.93	79
3 (upper)	26,331	251,998	1	9.57	1,296
3 (lower)	27,662	256,467	1	9.27	1,289
4 (upper)	1,677	76,681	1	45.73	7,265
4 (lower)	2,693	103,047	1	38.26	1,665
5 (upper)	80	36,122	2	451.52	8,436
5 (lower)	216	57,456	1	266.00	24,197
6 (upper)	4	136,458	1,152	34,114.50	123,228
6 (lower)	12	291,210	57	24,267.50	266,389
7 (upper)	2	10,382,652	2,494	5,191,330.00	10,380,158
7 (lower)	2	786,384	274	393,192.00	786,110
8 (upper)	1	2,393,387	2,393,387	2,393,390.00	2,393,387
8 (lower)	1	1	1	1.00	1
Total	29,534,089	48,399,592	1	1.64	1

vertebra (512x512x512)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	710,313	710,313	1	1.00	1
0 (lower)	697,188	697,188	1	1.00	1
1 (upper)	125,016	238,245	1	1.91	21
1 (lower)	123,330	236,581	1	1.92	49
2 (upper)	22,641	77,378	1	3.42	37
2 (lower)	22,851	81,810	1	3.58	296
3 (upper)	2,995	18,202	1	6.08	84
3 (lower)	3,423	33,397	1	9.76	11,858
4 (upper)	198	2,256	1	11.39	155
4 (lower)	308	8,474	1	27.51	4,282
5 (upper)	5	45,661	5	9,132.20	45,537
5 (lower)	8	19,113	2	2,389.12	18,981
6 (upper)	1	639,975	639,975	639,975.00	639,975
6 (lower)	1	1	1	1.00	1
Total	1,708,278	2,808,594	1	1.64	1

stag beetle (832x832x494)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	173,400	173,400	1	1.00	1
0 (lower)	185,667	185,667	1	1.00	1
1 (upper)	24,489	41,769	1	1.71	17
1 (lower)	30,871	58,043	1	1.88	71
2 (upper)	3,202	11,874	1	3.71	82
2 (lower)	5,682	24,430	1	4.30	946
3 (upper)	421	6,395	1	15.19	148
3 (lower)	1,049	17,662	1	16.84	7,483
4 (upper)	69	7,948	1	115.19	1,870
4 (lower)	190	8,397	1	44.19	490
5 (upper)	7	2,391	32	341.57	1,795
5 (lower)	31	8,540	2	275.48	2,828
6 (upper)	2	18,895	439	9,447.50	18,456
6 (lower)	6	6,010	40	1,001.67	3,748
7 (upper)	1	140,676	140,676	140,676.00	140,676
7 (lower)	1	1	1	1.00	1
Total	425,088	712,098	1	1.68	1

mag. reconnection (512x512x512)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	7,051,861	7,051,861	1	1.00	1
0 (lower)	7,349,289	7,349,289	1	1.00	1
1 (upper)	1,165,087	2,003,273	1	1.72	55
1 (lower)	1,372,871	2,448,675	1	1.78	24
2 (upper)	174,425	760,621	1	4.36	170
2 (lower)	263,704	869,727	1	3.30	67
3 (upper)	26,507	356,027	1	13.43	292
3 (lower)	46,987	287,978	1	6.13	188
4 (upper)	3,741	126,162	1	33.72	1,041
4 (lower)	6,279	81,485	1	12.98	692
5 (upper)	358	34,419	1	96.14	2,075
5 (lower)	582	40,166	1	69.01	4,434
6 (upper)	24	141,934	2	5,913.92	56,694
6 (lower)	50	107,068	13	2,141.36	23,650
7 (upper)	1	199,929	199,929	199,929.00	199,929
7 (lower)	9	234,436	1,156	26,048.40	76,579
8 (upper)	1	132,100	132,100	132,100.00	132,100
8 (lower)	2	5,635,254	2,805,510	2,817,630.00	2,829,744
9 (upper)	1	1	1	1.00	1
Total	17,461,779	27,860,405	1	1.60	1

pawpawsaurus (958x646x1088)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	19,287,297	19,287,297	1	1.00	1
0 (lower)	19,348,617	19,348,617	1	1.00	1
1 (upper)	3,641,312	6,484,774	1	1.78	24
1 (lower)	3,675,455	6,581,455	1	1.79	29
2 (upper)	624,168	1,898,808	1	3.04	205
2 (lower)	645,501	2,007,563	1	3.11	413
3 (upper)	77,107	388,181	1	5.03	2,147
3 (lower)	87,570	469,584	1	5.36	2,887
4 (upper)	4,733	186,475	1	39.40	19,604
4 (lower)	7,588	188,725	1	24.87	15,445
5 (upper)	174	194,693	1	1,118.93	36,664
5 (lower)	561	284,623	1	507.35	29,291
6 (upper)	15	205,624	25	13,708.30	72,572
6 (lower)	60	341,333	7	5,688.88	107,526
7 (upper)	1	12,247,346	12,247,346	12,247,300.00	12,247,346
7 (lower)	12	1,569,487	2,470	130,791.00	1,357,541
8 (upper)	1	3,750,263	3,750,263	3,750,260.00	3,750,263
8 (lower)	3	929,837	37,489	309,946.00	531,545
9 (upper)	1	8,650	8,650	8,650.00	8,650
9 (lower)	1	1	1	1.00	1
Total	47,400,177	76,373,336	1	1.61	1

spathorhynchus (1024x1024x750)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	9,934,218	9,934,218	1	1.00	1
0 (lower)	9,869,552	9,869,552	1	1.00	1
1 (upper)	1,915,871	3,485,788	1	1.82	21
1 (lower)	1,911,896	3,489,802	1	1.83	2,859
2 (upper)	352,275	1,178,007	1	3.34	119
2 (lower)	357,062	1,201,884	1	3.37	155
3 (upper)	54,182	325,408	1	6.01	427
3 (lower)	57,418	345,133	1	6.01	306
4 (upper)	5,623	97,576	1	17.35	1,774
4 (lower)	6,672	85,980	1	12.89	2,192
5 (upper)	379	66,469	1	175.38	7,244
5 (lower)	533	20,761	1	38.95	2,051
6 (upper)	20	52,494	63	2,624.70	11,987
6 (lower)	14	36,333	13	2,595.21	17,110
7 (upper)	3	57,160	1,542	19,053.30	53,762
7 (lower)	1	9,129,481	9,129,481	9,129,480.00	9,129,481
8 (upper)	1	1	1	1.00	1
Total	24,465,720	39,376,047	1	1.61	1

kingsnake (1024x1024x750)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	12,632,583	12,632,583	1	1.00	1
0 (lower)	12,836,212	12,836,212	1	1.00	1
1 (upper)	2,427,078	4,396,435	1	1.81	19
1 (lower)	2,445,861	4,507,113	1	1.84	32
2 (upper)	437,962	1,466,466	1	3.35	53
2 (lower)	444,437	1,577,939	1	3.55	191
3 (upper)	64,617	411,830	1	6.37	294
3 (lower)	65,855	466,979	1	7.09	404
4 (upper)	6,271	94,345	1	15.04	586
4 (lower)	6,515	117,396	1	18.02	769
5 (upper)	452	36,334	1	80.39	3,024
5 (lower)	454	50,195	1	110.56	4,113
6 (upper)	50	19,758	9	395.16	2,555
6 (lower)	34	1,613,889	5	47,467.30	1,565,355
7 (upper)	10	45,681	101	4,568.10	33,433
7 (lower)	7	3,821,218	2	545,888.00	3,794,106
8 (upper)	2	1,495,163	1,445	747,582.00	1,493,718
8 (lower)	2	1,336,393	107,718	668,196.00	1,228,675
9 (upper)	1	3,626,483	3,626,483	3,626,480.00	3,626,483
9 (lower)	1	1	1	1.00	1
Total	31,368,404	50,552,413	1	1.61	1

warpx rho (6791x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	74,923	74,923	1	1.00	1
0 (lower)	86,496	86,496	1	1.00	1
1 (upper)	10,787	19,324	1	1.79	27
1 (lower)	12,369	21,942	1	1.77	49
2 (upper)	1,392	6,374	1	4.58	305
2 (lower)	1,604	7,299	1	4.55	71
3 (upper)	185	4,064	1	21.97	1,010
3 (lower)	197	4,666	1	23.69	1,310
4 (upper)	35	15,114	1	431.83	11,557
4 (lower)	37	6,214	1	167.95	2,239
5 (upper)	4	20,414	2	5,103.50	20,247
5 (lower)	6	933	1	155.50	658
6 (upper)	1	54,264	54,264	54,264.00	54,264
6 (lower)	1	1	1	1.00	1
Total	188,037	322,028	1	1.71	1

warpx Ez (6791x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	92,450	92,450	1	1.00	1
0 (lower)	97,316	97,316	1	1.00	1
1 (upper)	8,329	18,452	1	2.22	168
1 (lower)	8,823	18,353	1	2.08	71
2 (upper)	1,212	7,335	1	6.05	135
2 (lower)	1,302	9,027	1	6.93	283
3 (upper)	175	6,132	1	35.04	2,061
3 (lower)	178	5,293	1	29.74	901
4 (upper)	18	21,601	1	1,200.06	21,057
4 (lower)	21	19,434	1	925.43	17,347
5 (upper)	3	32,903	2	10,967.70	32,889
5 (lower)	2	49,563	1	24,781.50	49,562
6 (upper)	1	207	207	207.00	207
6 (lower)	1	1	1	1.00	1
Total	209,831	378,067	1	1.80	1

warpx Ey (6791x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	85,044	85,044	1	1.00	1
0 (lower)	81,477	81,504	1	1.00	28
1 (upper)	6,291	14,355	1	2.28	140
1 (lower)	6,128	12,460	1	2.03	176
2 (upper)	733	5,883	1	8.03	208
2 (lower)	674	4,827	1	7.16	99
3 (upper)	76	7,923	1	104.25	5,244
3 (lower)	53	1,779	1	33.57	1,318
4 (upper)	7	4,943	2	706.14	3,489
4 (lower)	6	4,357	1	726.17	4,041
5 (upper)	3	3,735	2	1,245.00	2,092
5 (lower)	2	8,176	3,193	4,088.00	4,983
6 (upper)	1	95,925	95,925	95,925.00	95,925
6 (lower)	1	1	1	1.00	1
Total	180,496	330,912	1	1.83	1

warpx Ex (6791x371x371)

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	60,849	60,849	1	1.00	1
0 (lower)	60,961	60,962	1	1.00	2
1 (upper)	5,625	12,497	1	2.22	39
1 (lower)	6,068	13,618	1	2.24	46
2 (upper)	912	6,727	1	7.38	270
2 (lower)	972	6,581	1	6.77	239
3 (upper)	132	2,331	1	17.66	342
3 (lower)	145	3,101	1	21.39	454
4 (upper)	20	4,376	1	218.80	3,487
4 (lower)	22	4,177	1	189.86	2,843
5 (upper)	2	38,774	10	19,387.00	38,764
5 (lower)	2	15,278	6	7,639.00	15,272
6 (upper)	1	13,170	13,170	13,170.00	13,170
6 (lower)	1	1	1	1.00	1
Total	135,712	242,442	1	1.79	1

R.3 Hyperstructure statistics: GTOPO Tiles

gt30antarcps

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	26,213	26,213	1	1.00	1
0 (lower)	26,620	26,622	1	1.00	3
1 (upper)	1,516	3,201	1	2.11	48
1 (lower)	747	1,968	1	2.63	39
2 (upper)	205	3,205	1	15.63	600
2 (lower)	66	1,233	1	18.68	469
3 (upper)	27	15,519	1	574.78	13,162
3 (lower)	7	1,169	2	167.00	1,056
4 (upper)	4	17,272	57	4,318.00	16,849
4 (lower)	1	5,344	5,344	5,344.00	5,344
5 (upper)	1	1	1	1.00	1
Total	55,407	101,747	1	1.84	1

gt30e020n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	1,076,545	1,076,545	1	1.00	1
0 (lower)	1,281,082	1,281,759	1	1.00	3
1 (upper)	163,735	303,509	1	1.85	52
1 (lower)	178,004	296,636	1	1.67	72
2 (upper)	23,103	140,772	1	6.09	973
2 (lower)	19,890	89,848	1	4.52	1,013
3 (upper)	3,187	112,126	1	35.18	1,249
3 (lower)	2,185	54,632	1	25.00	2,055
4 (upper)	454	108,853	1	239.76	4,353
4 (lower)	220	71,259	1	323.90	8,201
5 (upper)	73	141,945	2	1,944.45	33,820
5 (lower)	24	73,732	7	3,072.17	20,691
6 (upper)	16	189,532	452	11,845.80	64,619
6 (lower)	3	19,224	739	6,408.00	15,630
7 (upper)	5	258,791	272	51,758.20	166,686
7 (lower)	1	154,030	154,030	154,030.00	154,030
8 (upper)	1	273,905	273,905	273,905.00	273,905
8 (lower)	1	1	1	1.00	1
Total	2,748,529	4,647,099	1	1.69	1

gt30e020n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	799,473	799,473	1	1.00	1
0 (lower)	793,839	793,944	1	1.00	3
1 (upper)	134,578	258,863	1	1.92	56
1 (lower)	115,518	203,928	1	1.77	41
2 (upper)	22,305	138,388	1	6.20	157
2 (lower)	14,888	74,704	1	5.02	256
3 (upper)	3,785	109,206	1	28.85	746
3 (lower)	1,878	45,523	1	24.24	1,829
4 (upper)	703	103,008	1	146.53	2,592
4 (lower)	231	25,940	1	112.29	2,335
5 (upper)	133	104,260	4	783.91	9,970
5 (lower)	15	53,501	30	3,566.73	38,912
6 (upper)	26	112,489	51	4,326.50	30,550
6 (lower)	3	82,084	5,439	27,361.30	42,458
7 (upper)	6	109,843	9,524	18,307.20	23,227
7 (lower)	2	109,368	19,659	54,684.00	89,709
8 (upper)	2	39,075	3,056	19,537.50	36,019
8 (lower)	1	1	1	1.00	1
Total	1,887,386	3,163,598	1	1.68	1

gt30e020s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	326,497	326,497	1	1.00	1
0 (lower)	421,497	421,723	1	1.00	3
1 (upper)	48,940	90,030	1	1.84	43
1 (lower)	56,607	92,011	1	1.63	24
2 (upper)	6,330	39,016	1	6.16	291
2 (lower)	5,504	25,063	1	4.55	2,031
3 (upper)	795	35,722	1	44.93	1,793
3 (lower)	463	17,950	1	38.77	3,344
4 (upper)	116	38,558	1	332.40	7,186
4 (lower)	42	18,500	1	440.48	11,797
5 (upper)	17	43,763	15	2,574.29	13,974
5 (lower)	4	30,014	23	7,503.50	23,459
6 (upper)	5	129,893	2,837	25,978.60	72,218
6 (lower)	2	165,901	684	82,950.50	165,217
7 (upper)	1	1	1	1.00	1
Total	866,820	1,474,642	1	1.70	1

gt30e060n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	597,412	597,412	1	1.00	1
0 (lower)	513,903	513,993	1	1.00	3
1 (upper)	92,480	174,163	1	1.88	34
1 (lower)	60,883	108,207	1	1.78	72
2 (upper)	15,044	86,821	1	5.77	174
2 (lower)	7,592	43,355	1	5.71	1,139
3 (upper)	2,557	62,619	1	24.49	835
3 (lower)	877	24,824	1	28.31	609
4 (upper)	460	59,227	1	128.75	2,776
4 (lower)	87	22,252	1	255.77	6,373
5 (upper)	90	37,028	2	411.42	5,138
5 (lower)	10	46,109	5	4,610.90	28,377
6 (upper)	17	45,518	48	2,677.53	22,866
6 (lower)	3	192,834	2,500	64,278.00	142,283
7 (upper)	3	49,199	739	16,399.70	37,291
7 (lower)	1	139,028	139,028	139,028.00	139,028
8 (upper)	1	1	1	1.00	1
Total	1,291,420	2,202,590	1	1.71	1

gt30e060n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	952,677	952,677	1	1.00	1
0 (lower)	923,558	923,700	1	1.00	3
1 (upper)	160,578	302,251	1	1.88	47
1 (lower)	138,282	243,681	1	1.76	46
2 (upper)	26,003	150,902	1	5.80	191
2 (lower)	18,212	89,654	1	4.92	200
3 (upper)	4,260	115,795	1	27.18	900
3 (lower)	2,314	55,398	1	23.94	1,824
4 (upper)	749	110,216	1	147.15	3,348
4 (lower)	311	45,494	1	146.28	7,407
5 (upper)	136	103,333	3	759.80	24,254
5 (lower)	35	55,571	13	1,587.74	15,666
6 (upper)	28	77,885	100	2,781.61	13,016
6 (lower)	5	42,400	97	8,480.00	23,117
7 (upper)	5	49,687	2,987	9,937.40	19,577
7 (lower)	1	219,238	219,238	219,238.00	219,238
8 (upper)	2	184,863	1,422	92,431.50	183,441
8 (lower)	1	1	1	1.00	1
Total	2,227,157	3,722,746	1	1.67	1

gt30e060s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	293	293	1	1.00	1
0 (lower)	356	358	1	1.01	3
1 (upper)	56	141	1	2.52	19
1 (lower)	25	162	1	6.48	114
2 (upper)	10	59	1	5.90	33
2 (lower)	4	55	4	13.75	36
3 (upper)	3	38	9	12.67	18
3 (lower)	2	40	10	20.00	30
4 (upper)	1	147	147	147.00	147
4 (lower)	1	1	1	1.00	1
Total	751	1,294	1	1.72	1

gt30e060s60

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	305,496	305,496	1	1.00	1
0 (lower)	304,794	304,794	1	1.00	1
1 (upper)	928	1,414	1	1.52	29
1 (lower)	1,005	1,506	1	1.50	111
2 (upper)	46	1,517	1	32.98	621
2 (lower)	34	5,342	1	157.12	2,772
3 (upper)	2	398,776	298	199,388.00	398,478
3 (lower)	5	9,438	3	1,887.60	4,723
4 (upper)	1	113,801	113,801	113,801.00	113,801
4 (lower)	2	35,056	195	17,528.00	34,861
5 (upper)	1	1	1	1.00	1
Total	612,314	1,177,141	1	1.92	1

gt30e100n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	446,693	446,693	1	1.00	1
0 (lower)	419,319	419,377	1	1.00	2
1 (upper)	73,927	144,434	1	1.95	38
1 (lower)	54,310	98,433	1	1.81	51
2 (upper)	12,440	77,571	1	6.24	124
2 (lower)	7,118	37,304	1	5.24	125
3 (upper)	2,265	61,109	1	26.98	521
3 (lower)	944	22,732	1	24.08	397
4 (upper)	412	55,039	1	133.59	1,654
4 (lower)	125	11,449	1	91.59	996
5 (upper)	76	77,318	2	1,017.34	10,289
5 (lower)	18	6,948	3	386.00	3,086
6 (upper)	13	19,710	111	1,516.15	5,030
6 (lower)	1	4,830	4,830	4,830.00	4,830
7 (upper)	4	122,168	193	30,542.00	113,472
7 (lower)	1	111,363	111,363	111,363.00	111,363
8 (upper)	1	1	1	1.00	1
Total	1,017,667	1,716,479	1	1.69	1

gt30e100n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	848,764	848,764	1	1.00	1
0 (lower)	779,550	779,610	1	1.00	2
1 (upper)	143,564	273,816	1	1.91	35
1 (lower)	97,856	165,313	1	1.69	88
2 (upper)	24,276	140,541	1	5.79	164
2 (lower)	11,699	55,017	1	4.70	382
3 (upper)	4,317	109,355	1	25.33	787
3 (lower)	1,520	34,123	1	22.45	1,006
4 (upper)	803	94,877	1	118.15	3,330
4 (lower)	202	41,932	1	207.58	8,303
5 (upper)	147	99,704	7	678.26	8,790
5 (lower)	18	26,532	14	1,474.00	11,341
6 (upper)	33	107,602	29	3,260.67	22,721
6 (lower)	4	17,839	661	4,459.75	12,593
7 (upper)	8	71,931	920	8,991.38	24,025
7 (lower)	2	287,058	90,717	143,529.00	196,341
8 (upper)	2	82,796	10,667	41,398.00	72,129
8 (lower)	1	1	1	1.00	1
Total	1,912,766	3,236,811	1	1.69	1

gt30e100s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	75,139	75,139	1	1.00	1
0 (lower)	78,604	78,604	1	1.00	1
1 (upper)	5,415	13,332	1	2.46	202
1 (lower)	4,062	9,403	1	2.31	150
2 (upper)	818	9,085	1	11.11	567
2 (lower)	354	9,471	1	26.75	2,297
3 (upper)	136	6,977	1	51.30	693
3 (lower)	39	13,044	1	334.46	10,923
4 (upper)	22	3,582	2	162.82	1,031
4 (lower)	5	16,999	10	3,399.80	15,888
5 (upper)	4	8,802	409	2,200.50	5,617
5 (lower)	1	51,723	51,723	51,723.00	51,723
6 (upper)	1	1	1	1.00	1
Total	164,600	296,162	1	1.80	1

gt30e120s60

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	314,600	314,600	1	1.00	1
0 (lower)	315,210	315,210	1	1.00	1
1 (upper)	2,446	4,059	1	1.66	113
1 (lower)	1,994	2,962	1	1.49	60
2 (upper)	151	2,581	1	17.09	1,144
2 (lower)	56	518	1	9.25	303
3 (upper)	20	171,870	1	8,593.50	158,078
3 (lower)	5	84	2	16.80	50
4 (upper)	1	409,870	409,870	409,870.00	409,870
4 (lower)	1	1	1	1.00	1
Total	634,484	1,221,755	1	1.93	1

gt30e140n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	12,273	12,273	1	1.00	1
0 (lower)	10,695	10,696	1	1.00	2
1 (upper)	2,108	4,265	1	2.02	41
1 (lower)	944	1,649	1	1.75	24
2 (upper)	348	2,207	1	6.34	62
2 (lower)	84	412	1	4.90	56
3 (upper)	60	2,438	1	40.63	398
3 (lower)	10	972	1	97.20	322
4 (upper)	14	2,464	31	176.00	773
4 (lower)	1	2,031	2,031	2,031.00	2,031
5 (upper)	3	5,260	179	1,753.33	3,562
5 (lower)	1	856	856	856.00	856
6 (upper)	1	1	1	1.00	1
Total	26,542	45,524	1	1.72	1

gt30e140n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	319,977	319,977	1	1.00	1
0 (lower)	291,240	291,274	1	1.00	3
1 (upper)	57,114	109,796	1	1.92	30
1 (lower)	34,650	58,776	1	1.70	50
2 (upper)	10,320	60,331	1	5.85	122
2 (lower)	3,669	17,334	1	4.72	87
3 (upper)	1,913	49,966	1	26.12	442
3 (lower)	387	8,899	1	22.99	289
4 (upper)	359	45,262	1	126.08	2,323
4 (lower)	40	5,107	2	127.67	711
5 (upper)	66	41,724	1	632.18	6,231
5 (lower)	6	11,401	11	1,900.17	10,690
6 (upper)	10	33,769	18	3,376.90	10,907
6 (lower)	2	121,521	46,807	60,760.50	74,714
7 (upper)	2	28,430	1,843	14,215.00	26,587
7 (lower)	1	10,922	10,922	10,922.00	10,922
8 (upper)	1	1	1	1.00	1
Total	719,757	1,214,490	1	1.69	1

gt30e140s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	65,737	65,737	1	1.00	1
0 (lower)	65,808	65,820	1	1.00	3
1 (upper)	9,789	20,641	1	2.11	209
1 (lower)	7,205	13,753	1	1.91	63
2 (upper)	1,628	10,830	1	6.65	116
2 (lower)	701	4,006	1	5.71	222
3 (upper)	276	7,865	1	28.50	279
3 (lower)	58	2,439	1	42.05	656
4 (upper)	49	11,227	1	229.12	1,692
4 (lower)	4	4,081	3	1,020.25	1,855
5 (upper)	9	7,662	69	851.33	3,764
5 (lower)	1	1,222	1,222	1,222.00	1,222
6 (upper)	3	42,989	202	14,329.70	36,828
6 (lower)	1	492	492	492.00	492
7 (upper)	1	1	1	1.00	1
Total	151,270	258,765	1	1.71	1

gt30w000s60

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	39,736	39,736	1	1.00	1
0 (lower)	39,732	39,732	1	1.00	1
1 (upper)	268	472	1	1.76	16
1 (lower)	194	1,310	1	6.75	644
2 (upper)	19	5,591	1	294.26	4,931
2 (lower)	9	8,178	1	908.67	7,837
3 (upper)	3	12,390	206	4,130.00	11,061
3 (lower)	1	49,234	49,234	49,234.00	49,234
4 (upper)	1	1	1	1.00	1
Total	79,963	156,644	1	1.96	1

gt30w020n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	575,565	575,565	1	1.00	1
0 (lower)	663,282	663,477	1	1.00	3
1 (upper)	90,244	175,405	1	1.94	69
1 (lower)	90,509	157,208	1	1.74	233
2 (upper)	14,487	89,621	1	6.19	932
2 (lower)	10,941	54,503	1	4.98	1,039
3 (upper)	2,233	65,920	1	29.52	2,469
3 (lower)	1,275	32,146	1	25.21	7,116
4 (upper)	325	52,887	1	162.73	3,212
4 (lower)	118	19,507	1	165.31	3,003
5 (upper)	37	155,490	11	4,202.43	38,171
5 (lower)	10	59,924	21	5,992.40	56,941
6 (upper)	7	126,716	3,031	18,102.30	32,170
6 (lower)	3	116,326	3,827	38,775.30	95,135
7 (upper)	1	91,905	91,905	91,905.00	91,905
7 (lower)	1	1	1	1.00	1
Total	1,449,038	2,436,601	1	1.68	1

gt30w020n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	271,661	271,661	1	1.00	1
0 (lower)	247,004	247,055	1	1.00	3
1 (upper)	47,350	94,673	1	2.00	87
1 (lower)	30,451	53,534	1	1.76	54
2 (upper)	8,509	53,032	1	6.23	108
2 (lower)	3,274	16,041	1	4.90	119
3 (upper)	1,518	41,036	1	27.03	729
3 (lower)	309	10,805	1	34.97	1,046
4 (upper)	276	29,844	1	108.13	1,733
4 (lower)	25	10,461	2	418.44	2,633
5 (upper)	59	36,304	3	615.32	5,462
5 (lower)	3	17,397	1,003	5,799.00	10,097
6 (upper)	13	41,351	327	3,180.85	16,045
6 (lower)	1	3,616	3,616	3,616.00	3,616
7 (upper)	3	102,518	3,092	34,172.70	69,333
7 (lower)	1	1,266	1,266	1,266.00	1,266
8 (upper)	1	1	1	1.00	1
Total	610,458	1,030,595	1	1.69	1

gt30w020s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	128,765	128,765	1	1.00	1
0 (lower)	169,230	169,318	1	1.00	3
1 (upper)	19,912	36,970	1	1.86	28
1 (lower)	23,566	38,351	1	1.63	31
2 (upper)	2,764	15,185	1	5.49	98
2 (lower)	2,476	10,067	1	4.07	140
3 (upper)	357	10,571	1	29.61	937
3 (lower)	274	6,951	1	25.37	791
4 (upper)	50	11,895	1	237.90	1,501
4 (lower)	26	6,920	2	266.15	1,867
5 (upper)	9	29,373	257	3,263.67	14,860
5 (lower)	6	53,406	1	8,901.00	45,241
6 (upper)	3	43,909	199	14,636.30	24,877
6 (lower)	1	25,816	25,816	25,816.00	25,816
7 (upper)	1	1	1	1.00	1
Total	347,440	587,498	1	1.69	1

gt30w060n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	39,174	39,174	1	1.00	1
0 (lower)	40,801	40,801	1	1.00	1
1 (upper)	6,943	14,476	1	2.08	104
1 (lower)	3,949	7,622	1	1.93	71
2 (upper)	1,366	6,823	1	4.99	83
2 (lower)	526	2,969	1	5.64	835
3 (upper)	239	4,814	1	20.14	211
3 (lower)	51	1,921	1	37.67	1,335
4 (upper)	40	7,722	1	193.05	1,915
4 (lower)	1	4,618	4,618	4,618.00	4,618
5 (upper)	6	1,151	44	191.83	420
5 (lower)	1	66	66	66.00	66
6 (upper)	2	15,417	4,750	7,708.50	10,667
6 (lower)	1	10,683	10,683	10,683.00	10,683
7 (upper)	1	1	1	1.00	1
Total	93,101	158,258	1	1.70	1

gt30w060n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	52,828	52,828	1	1.00	1
0 (lower)	50,687	50,693	1	1.00	3
1 (upper)	5,348	12,897	1	2.41	386
1 (lower)	3,089	6,904	1	2.24	66
2 (upper)	977	10,111	1	10.35	1,291
2 (lower)	364	2,576	1	7.08	292
3 (upper)	165	22,247	1	134.83	18,634
3 (lower)	32	3,814	1	119.19	2,023
4 (upper)	25	25,153	5	1,006.12	21,228
4 (lower)	5	1,457	27	291.40	771
5 (upper)	5	8,362	225	1,672.40	3,069
5 (lower)	2	888	319	444.00	569
6 (upper)	1	3,123	3,123	3,123.00	3,123
6 (lower)	1	1	1	1.00	1
Total	113,529	201,054	1	1.77	1

gt30w060s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	164,241	164,241	1	1.00	1
0 (lower)	167,542	167,565	1	1.00	3
1 (upper)	27,563	51,527	1	1.87	61
1 (lower)	21,775	38,065	1	1.75	204
2 (upper)	4,386	23,449	1	5.35	154
2 (lower)	2,541	12,615	1	4.96	537
3 (upper)	673	15,544	1	23.10	668
3 (lower)	277	8,525	1	30.78	903
4 (upper)	115	17,503	1	152.20	1,543
4 (lower)	30	10,859	1	361.97	4,761
5 (upper)	25	25,937	2	1,037.48	14,454
5 (lower)	5	24,116	38	4,823.20	16,657
6 (upper)	4	13,667	593	3,416.75	9,733
6 (lower)	2	69,422	2,192	34,711.00	67,230
7 (upper)	1	12,194	12,194	12,194.00	12,194
7 (lower)	1	1	1	1.00	1
Total	389,181	655,230	1	1.68	1

gt30w060s60

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	2,712	2,712	1	1.00	1
0 (lower)	2,600	2,600	1	1.00	1
1 (upper)	203	1,063	1	5.24	186
1 (lower)	137	365	1	2.66	13
2 (upper)	37	1,005	1	27.16	214
2 (lower)	13	116	1	8.92	29
3 (upper)	6	841	2	140.17	462
3 (lower)	1	1,740	1,740	1,740.00	1,740
4 (upper)	1	1	1	1.00	1
Total	5,710	10,443	1	1.83	1

gt30w100n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	377,275	377,275	1	1.00	1
0 (lower)	363,585	363,615	1	1.00	2
1 (upper)	51,779	100,962	1	1.95	90
1 (lower)	39,050	67,745	1	1.73	135
2 (upper)	8,090	50,837	1	6.28	577
2 (lower)	4,287	19,325	1	4.51	283
3 (upper)	1,238	36,658	1	29.61	612
3 (lower)	393	7,358	1	18.72	558
4 (upper)	186	34,369	1	184.78	2,956
4 (lower)	33	33,795	1	1,024.09	26,703
5 (upper)	36	41,234	5	1,145.39	6,639
5 (lower)	2	27,443	1,639	13,721.50	25,804
6 (upper)	7	151,818	1,493	21,688.30	58,567
6 (lower)	1	62,046	62,046	62,046.00	62,046
7 (upper)	2	85,527	22,511	42,763.50	63,016
7 (lower)	1	1	1	1.00	1
Total	845,965	1,460,008	1	1.73	1

gt30w100n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	440,035	440,035	1	1.00	1
0 (lower)	412,883	412,975	1	1.00	4
1 (upper)	75,792	152,785	1	2.02	83
1 (lower)	49,852	88,947	1	1.78	59
2 (upper)	13,651	84,634	1	6.20	837
2 (lower)	5,681	26,169	1	4.61	140
3 (upper)	2,370	63,923	1	26.97	1,971
3 (lower)	564	11,897	1	21.09	675
4 (upper)	403	66,082	1	163.97	3,964
4 (lower)	43	5,065	1	117.79	1,184
5 (upper)	80	82,925	5	1,036.56	9,877
5 (lower)	3	12,177	219	4,059.00	11,292
6 (upper)	18	73,700	141	4,094.44	19,663
6 (lower)	1	6,537	6,537	6,537.00	6,537
7 (upper)	5	41,345	1,207	8,269.00	17,194
7 (lower)	2	118,768	35,397	59,384.00	83,371
8 (upper)	1	1	1	1.00	1
Total	1,001,384	1,687,965	1	1.69	1

gt30w100s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	280,101	280,101	1	1.00	1
0 (lower)	269,351	269,401	1	1.00	5
1 (upper)	40,546	74,060	1	1.83	36
1 (lower)	32,965	57,396	1	1.74	48
2 (upper)	5,612	35,325	1	6.29	307
2 (lower)	3,522	19,752	1	5.61	239
3 (upper)	822	25,734	1	31.31	756
3 (lower)	393	15,871	1	40.38	849
4 (upper)	123	17,602	2	143.11	3,469
4 (lower)	43	10,051	9	233.74	2,153
5 (upper)	23	14,234	14	618.87	2,609
5 (lower)	5	51,736	73	10,347.20	27,993
6 (upper)	3	2,927	25	975.67	1,677
6 (lower)	1	212,632	212,632	212,632.00	212,632
7 (upper)	1	1	1	1.00	1
Total	633,511	1,086,823	1	1.72	1

gt30w120s60

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	68,175	68,175	1	1.00	1
0 (lower)	69,187	69,187	1	1.00	1
1 (upper)	1,036	3,395	1	3.28	207
1 (lower)	649	1,916	1	2.95	97
2 (upper)	170	15,348	1	90.28	8,709
2 (lower)	51	1,010	1	19.80	185
3 (upper)	35	20,296	1	579.89	16,099
3 (lower)	5	2,632	19	526.40	1,082
4 (upper)	7	20,255	2	2,893.57	16,298
4 (lower)	2	13,617	134	6,808.50	13,483
5 (upper)	3	51,074	62	17,024.70	50,769
5 (lower)	1	1,578	1,578	1,578.00	1,578
6 (upper)	1	1	1	1.00	1
Total	139,322	268,484	1	1.93	1

gt30w140n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	171,759	171,759	1	1.00	1
0 (lower)	143,125	143,144	1	1.00	3
1 (upper)	22,406	44,070	1	1.97	29
1 (lower)	12,279	21,254	1	1.73	47
2 (upper)	3,355	28,126	1	8.38	184
2 (lower)	1,104	8,199	1	7.43	385
3 (upper)	571	23,286	1	40.78	696
3 (lower)	118	12,316	1	104.37	5,177
4 (upper)	100	13,740	1	137.40	1,313
4 (lower)	16	11,093	2	693.31	4,309
5 (upper)	19	14,661	35	771.63	2,783
5 (lower)	2	46,883	12,584	23,441.50	34,299
6 (upper)	4	45,786	245	11,446.50	24,511
6 (lower)	1	39,487	39,487	39,487.00	39,487
7 (upper)	1	1	1	1.00	1
Total	354,860	623,805	1	1.76	1

gt30w140n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	394,703	394,703	1	1.00	1
0 (lower)	375,273	375,343	1	1.00	3
1 (upper)	66,169	134,073	1	2.03	110
1 (lower)	41,669	73,130	1	1.76	71
2 (upper)	12,346	79,844	1	6.47	271
2 (lower)	4,360	21,479	1	4.93	177
3 (upper)	2,503	65,597	1	26.21	1,029
3 (lower)	428	13,467	1	31.46	1,496
4 (upper)	513	52,516	1	102.37	2,313
4 (lower)	38	9,357	3	246.24	1,766
5 (upper)	117	50,653	2	432.93	4,504
5 (lower)	7	5,891	81	841.57	2,505
6 (upper)	18	39,863	78	2,214.61	13,041
6 (lower)	3	77,122	2,456	25,707.30	53,001
7 (upper)	4	81,974	69	20,493.50	77,342
7 (lower)	1	53,093	53,093	53,093.00	53,093
8 (upper)	1	1	1	1.00	1
Total	898,153	1,528,106	1	1.70	1

gt30w140s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	74	74	1	1.00	1
0 (lower)	47	47	1	1.00	1
1 (upper)	25	45	1	1.80	9
1 (lower)	2	56	1	28.00	55
2 (upper)	2	15	1	7.50	14
2 (lower)	1	1	1	1.00	1
Total	151	238	1	1.58	1

gt30w180n40

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	280	280	1	1.00	1
0 (lower)	308	309	1	1.00	2
1 (upper)	44	110	1	2.50	14
1 (lower)	19	57	1	3.00	28
2 (upper)	12	159	1	13.25	71
2 (lower)	3	18	2	6.00	13
3 (upper)	2	227	65	113.50	162
3 (lower)	1	6	6	6.00	6
4 (upper)	1	1	1	1.00	1
Total	670	1,167	1	1.74	1

gt30w180n90

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	152,180	152,180	1	1.00	1
0 (lower)	138,739	138,748	1	1.00	2
1 (upper)	27,889	56,103	1	2.01	28
1 (lower)	14,076	23,438	1	1.67	34
2 (upper)	5,477	32,652	1	5.96	133
2 (lower)	1,209	5,532	1	4.58	98
3 (upper)	1,131	25,974	1	22.97	418
3 (lower)	117	3,151	1	26.93	436
4 (upper)	217	20,316	1	93.62	1,732
4 (lower)	5	2,010	2	402.00	964
5 (upper)	48	22,359	6	465.81	1,942
5 (lower)	2	1,016	358	508.00	658
6 (upper)	14	13,117	34	936.93	4,260
6 (lower)	1	9,116	9,116	9,116.00	9,116
7 (upper)	3	11,576	233	3,858.67	10,807
7 (lower)	1	61,163	61,163	61,163.00	61,163
8 (upper)	1	1	1	1.00	1
Total	341,110	578,452	1	1.70	1

gt30w180s10

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	675	675	1	1.00	1
0 (lower)	459	459	1	1.00	1
1 (upper)	144	331	1	2.30	14
1 (lower)	32	64	1	2.00	23
2 (upper)	22	119	1	5.41	16
2 (lower)	3	388	2	129.33	380
3 (upper)	2	218	2	109.00	216
3 (lower)	1	1	1	1.00	1
Total	1,338	2,255	1	1.69	1

gt30w180s60

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	22,034	22,034	1	1.00	1
0 (lower)	21,094	21,094	1	1.00	1
1 (upper)	586	2,502	1	4.27	438
1 (lower)	371	1,403	1	3.78	58
2 (upper)	105	12,749	1	121.42	10,663
2 (lower)	35	645	1	18.43	104
3 (upper)	13	15,764	2	1,212.62	11,483
3 (lower)	1	3,779	3,779	3,779.00	3,779
4 (upper)	2	5,244	299	2,622.00	4,945
4 (lower)	1	1	1	1.00	1
Total	44,242	85,215	1	1.93	1

R.4 Hyperstructure statistics: GTOPO Scaled

gtopo full scaled=0.03125

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	20,027	20,027	1	1.00	1
0 (lower)	16,520	16,520	1	1.00	1
1 (upper)	3,318	6,423	1	1.94	22
1 (lower)	2,037	3,762	1	1.85	48
2 (upper)	546	3,972	1	7.27	86
2 (lower)	213	1,617	1	7.59	138
3 (upper)	111	4,614	1	41.57	668
3 (lower)	21	725	2	34.52	383
4 (upper)	18	5,938	5	329.89	2,472
4 (lower)	3	738	33	246.00	497
5 (upper)	3	6,987	1,368	2,329.00	3,941
5 (lower)	1	952	952	952.00	952
6 (upper)	1	1	1	1.00	1
Total	42,819	72,276	1	1.69	1

gtopo full scaled=0.0625

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	74,271	74,271	1	1.00	1
0 (lower)	63,237	63,255	1	1.00	3
1 (upper)	11,936	23,036	1	1.93	29
1 (lower)	8,211	14,609	1	1.78	35
2 (upper)	1,963	13,079	1	6.66	183
2 (lower)	933	5,130	1	5.50	95
3 (upper)	348	14,297	1	41.08	1,042
3 (lower)	84	3,440	1	40.95	403
4 (upper)	65	14,073	1	216.51	1,066
4 (lower)	8	2,945	3	368.12	1,469
5 (upper)	12	19,167	52	1,597.25	8,124
5 (lower)	1	317	317	317.00	317
6 (upper)	3	16,621	4,086	5,540.33	7,556
6 (lower)	1	7,531	7,531	7,531.00	7,531
7 (upper)	1	1	1	1.00	1
Total	161,074	271,772	1	1.69	1

gtopo full scaled=0.125

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	266,554	266,554	1	1.00	1
0 (lower)	234,880	234,914	1	1.00	2
1 (upper)	43,612	83,124	1	1.91	26
1 (lower)	31,124	54,738	1	1.76	54
2 (upper)	7,191	44,932	1	6.25	710
2 (lower)	3,705	18,939	1	5.11	105
3 (upper)	1,219	39,676	1	32.55	1,937
3 (lower)	395	11,883	1	30.08	1,111
4 (upper)	236	43,293	1	183.44	2,580
4 (lower)	40	4,818	4	120.45	613
5 (upper)	42	44,299	8	1,054.74	4,273
5 (lower)	3	5,720	207	1,906.67	3,410
6 (upper)	8	83,204	505	10,400.50	25,307
6 (lower)	1	7,661	7,661	7,661.00	7,661
7 (upper)	2	18,872	708	9,436.00	18,164
7 (lower)	1	28,852	28,852	28,852.00	28,852
8 (upper)	1	1	1	1.00	1
Total	589,014	991,480	1	1.68	1

gtopo full scaled=0.25

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	939,798	939,798	1	1.00	1
0 (lower)	870,276	870,376	1	1.00	3
1 (upper)	151,085	291,325	1	1.93	198
1 (lower)	111,034	193,983	1	1.75	66
2 (upper)	25,138	157,142	1	6.25	530
2 (lower)	13,212	66,573	1	5.04	326
3 (upper)	4,262	151,060	1	35.44	29,745
3 (lower)	1,455	37,112	1	25.51	1,948
4 (upper)	769	139,616	1	181.56	15,439
4 (lower)	152	25,520	1	167.90	4,820
5 (upper)	156	124,467	2	797.87	7,679
5 (lower)	16	11,496	10	718.50	4,955
6 (upper)	35	126,737	6	3,621.06	22,918
6 (lower)	1	8,019	8,019	8,019.00	8,019
7 (upper)	8	199,108	1,388	24,888.50	79,566
7 (lower)	1	30,580	30,580	30,580.00	30,580
8 (upper)	2	104,272	587	52,136.00	103,685
8 (lower)	1	101,932	101,932	101,932.00	101,932
9 (upper)	1	1	1	1.00	1
Total	2,117,402	3,579,117	1	1.69	1

gtopo full scaled=0.5

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	3,247,221	3,247,221	1	1.00	1
0 (lower)	3,172,821	3,173,278	1	1.00	5
1 (upper)	497,462	948,572	1	1.91	197
1 (lower)	383,100	651,012	1	1.70	214
2 (upper)	80,605	510,225	1	6.33	5,707
2 (lower)	44,499	224,105	1	5.04	767
3 (upper)	13,724	400,672	1	29.20	10,523
3 (lower)	5,082	122,275	1	24.06	1,427
4 (upper)	2,379	647,850	1	272.32	280,895
4 (lower)	519	78,870	1	151.97	9,509
5 (upper)	457	461,739	1	1,010.37	99,352
5 (lower)	49	65,717	3	1,341.16	15,034
6 (upper)	86	451,359	63	5,248.36	50,450
6 (lower)	5	39,528	107	7,905.60	19,858
7 (upper)	13	871,952	3,308	67,073.20	290,849
7 (lower)	1	106,615	106,615	106,615.00	106,615
8 (upper)	1	687,679	687,679	687,679.00	687,679
8 (lower)	1	1	1	1.00	1
Total	7,448,025	12,688,670	1	1.70	1

gtopo full scaled=1

Iteration	Hyperarcs	Superarcs	Min Path	Avg Path	Max Path
0 (upper)	9,314,516	9,314,516	1	1.00	1
0 (lower)	9,365,583	9,367,520	1	1.00	4
1 (upper)	1,376,996	2,647,426	1	1.92	438
1 (lower)	1,112,917	1,921,963	1	1.73	233
2 (upper)	223,138	1,378,580	1	6.18	10,663
2 (lower)	129,122	620,453	1	4.81	2,031
3 (upper)	37,005	1,191,714	1	32.20	97,781
3 (lower)	14,494	338,611	1	23.36	7,116
4 (upper)	6,352	1,769,558	1	278.58	834,985
4 (lower)	1,498	223,372	1	149.11	17,827
5 (upper)	1,151	1,351,895	1	1,174.54	335,520
5 (lower)	121	147,021	3	1,215.05	35,822
6 (upper)	225	1,079,846	18	4,799.32	64,619
6 (lower)	13	110,648	215	8,511.38	39,616
7 (upper)	43	1,178,495	108	27,406.90	181,958
7 (lower)	1	44,080	44,080	44,080.00	44,080
8 (upper)	6	2,217,728	22,857	369,621.00	890,829
8 (lower)	1	203,606	203,606	203,606.00	203,606
9 (upper)	1	1,805,490	1,805,490	1,805,490.00	1,805,490
9 (lower)	1	1	1	1.00	1
Total	21,583,184	36,912,523	1	1.71	1