Component-based 2-/3-dimensional nearest neighbor search based on Elias method to GPU parallel 2D/3D Euclidean Minimum Spanning Tree Problem

Wen-Bao Qiao, Jean-Charles Créput

Component-based 2-/3-dimensional Nearest Neighbor Search based on Elias method to GPU parallel 2D/3D Euclidean Minimum Spanning Tree Problem

Wen-Bao Qiao[a,b,*], Jean-Charles Créput[b]

[a] *Computer School, Beijing Information Science and Technology University, China*
[b] *CIAD, Univ. Bourgogne Franche-Comté, UTBM, F-90010 Belfort, France*

Abstract

We present improved data parallel approaches working on graphics processing unit (GPU) compute unified device architecture (CUDA) platform to build hierarchical Euclidean minimum spanning forest or tree (EMSF/EMST) for applications whose input only contains N points with arbitrary data distribution in 2D/3D Euclidean space. Characteristic of the proposed parallel algorithms follows "data parallelism, decentralized control and $O(1)$ local memory occupied by each GPU thread". This research has to solve GPU parallelism of component-based nearest neighbor search (component-based NNS), tree traversal, and other graph operations like union-find. For exact NNS, instead of using classical K-d tree search or brute-force computing method, we propose a K-d search method working based on dividing the Euclidean K-dimensional space into congruent and non-overlapping square/cubic cells where size of points in each cell is bounded. For component-based NNS, with the uniqueness property based on 2D/3D square/cubic space partition, we propose dynamic and static pruning techniques to prune unnecessary neighbor cells' search. For tree traversal, instead of using breadth-first-search, this paper proposes CUDA kernels working with a distributed dynamic link list for selecting a local spanning tree's shortest outgoing edge since size of local EMSTs in EMSF can not be predicted.

* Corresponding author
Email address: rapidbao@outlook.com (Wen-Bao Qiao)

Source code is provided online and experimental comparisons are conducted on both 2D and 3D benchmarks with up to 10^7 points to build final EMST. Results show that applying K-d search with static pruning technique and the proposed operators totally working in parallel on GPU, our current implementation runs faster than our previous work and current optimal sequential dual-tree mlpack EMST library.

Keywords: Component-based nearest neighbor search, 3D Euclidean Minimum Spanning Tree, GPU Parallel 3D EMST, GPU Breadth first search, GPU union-find, GPU link list, decentralized control

---

1. Introduction

Given a point set $v_i \in V$ of N points in K–dimensional Euclidean space, the exact Euclidean Minimum Spanning Tree (EMST) problem is to find the lowest weight spanning tree in a complete un-directed graph $G = (V, \varnothing)$, $(v_0, v_1, ...v_{N-1} \in V)$, with implicit $\frac{N \times (N-1)}{2}$ edge list E and edge weights given by the Euclidean distance between any two points. A general minimum weight spanning tree (MST) is defined as: given a general weighted un-directed graph $G = (V, E)$ with N vertexes and explicit edge list E, finding a subset of E that connects all vertexes without any circles and with minimum total edges' weight[1]. A minimum spanning forest (MSF) consists of local MSTs on each of the connected graph components of that graph G. A *connected component (or just component)* of an un-directed graph G is a sub-graph of G in which any two vertexes are connected by a finite or infinite sequence of edges[1, 2].

MSF/MST is widely used in computer vision, pattern recognition and data mining applications, for example, researchers have applied MST to stereo matching and achieved top ranking result on Middlebury [3, 4]; MSF consisting various small MSTs naturally forms data clusters of an input[5, 6, 7], like image segmentation [8, 9]; an exact MST also provides upper bound for tour length estimation of traveling salesman problems [10]. In cases when the input only contains independent Euclidean points, building MSF/MST becomes EMSF/EMST problem.

Furthermore, with the development of big data and edge-computing in real 3-dimensional world [11], some applications require the ability to deal with big 3D data in real-time on graphics processing unit (GPU). This further requires the capability to build large-size 3D MSF/MST in real-time.

Three classical MST algorithms like Borůvka's [1] (see [12] for translation), Kruskal's (1956) [13] and Prim's (1957) [14] algorithms establish the basis of MST implementations. They all work sequentially on standard general graph with predefined distance matrix (or edge list), and take quadratic time complexity. The best time complexity for a sequential general MST solution is proposed by Bernard Chazelle [15], and it takes $O(E\alpha(E,V))$ where α is the functional inverse of Ackermann's function. However, it is time-consuming to apply these general MST algorithms to build 3D EMST on the implicit complete graph $G = (V, \varnothing)$ whose $\frac{N\times(N-1)}{2}$ edge list E is unprepared and N is large.

Very few approaches directly address exact EMSF/EMST, except for some efficient sequential algorithms like these algorithms using Delaunay triangulation [16], Voronoi diagram [17], K-d tree [18], dual tree [6], and a GPU parallel algorithm using sliced spiral search [2]. They mainly take advantage of the nearest neighbor search (NNS) [19] in Euclidean space since geometrically closest pair of points naturally satisfy the definition of MST[6][18][20]. While, when considering the 3D or higher K-d EMSF/EMST, previous solutions become even more rarer, the most recent optimal sequential EMST algorithm is the dual-tree EMST proposed by March et al.[6].

It is not an easy trick to make an integral GPU parallel 2D/3D EMSF/EMST algorithm and get actual computing acceleration over state-of-art sequential K-d EMSF/EMST algorithm. This is because building K-d EMST is an iterative procedure including K–d NNS, tree traversal, and other graphical operations. Parallelism of each sub-step should obey the characteristic of "data parallelism, decentralized control, and $O(1)$ local memory occupied by each GPU thread" in order to highly utilize the capability of NVIDIA GPU card series. Also, frequent data transfer between CPU and GPU is time-consuming. To our best knowledge, we do not find in literature other researchers' GPU parallel solutions

to 2D/3D EMSF/EMST problem.

2. Related Work

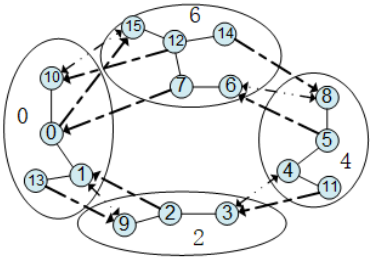
As we focus on GPU parallel EMSF/EMST algorithms, we ignore parallelism of general MSF/MST such as [21, 22, 23, 24, 25, 26], and mainly investigate related EMSF/EMST algorithms[17, 18, 16, 20, 6, 2] and their operators. Actually, the most recent state-of-art sequential EMSF/EMST is proposed by March et al. in 2010 [6], while the first GPU parallel EMSF/EMST was proposed by Qiao et al. in 2019 [2].

Since the input graph $G = (V, \varnothing)$ only contains vertexes, building EMSF/EMST naturally follows Borůvka's framework and consists of four steps illustrated in Fig.1, which begins with each vertex of the graph G being a component (or local MST) of a whole MSF, iteratively *finds minimum weighted outgoing edge for each component* and adds all such edges to current components in one iteration. During this procedure, size of each component grows arbitrarily and iteratively, and it mainly contains following operators:

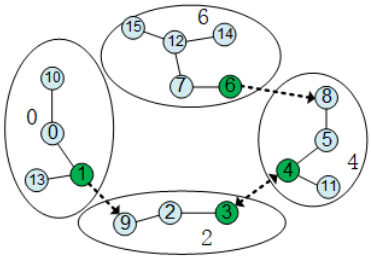
- *Find each component's closest outgoing vertex.* This includes two sub-steps:
  - *FindMin1: Find each vertex's closest outgoing point that belongs to another component;*;
  - *FindMin2: Find each component's shortest outgoing edge among all sub-vertexes' minimum outgoing weights.*
- *Union-find, to determine whether two nodes belongs to the same component, connect old components and merge them into new big component.*

Table 1 shows related operators to solve FindMin1 in literature, while table 2 shows related operators to solve FindMin2.

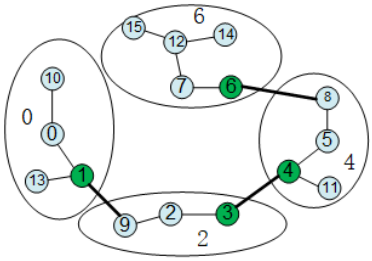
To solve FindMin1, EMST algorithms usually use exact K-d NNS algorithm by adding a judgment of whether the nearest points belong to different components [17, 18, 16, 20, 6], except for the GPU parallel 2D-EMSF/EMST algorithm



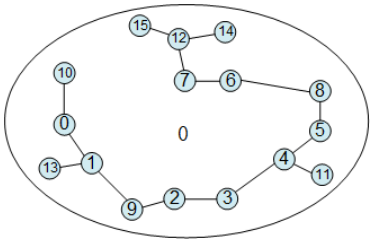
(a) *FindMin1*: Find each vertex's closest outgoing point. Number in white ellipses represents each component' root identifier.



(b) *FindMin2*: Find the node (dark blue) with the shortest outgoing edge of each component, called the winner node of the component.



(c) *Connect Graph*: Add link to each winner node's and to its correspondent node's (green) adjacency list .



(d) *Compact Graph*: Merge these connected local EMSTs by updating each vertex's root identifier.

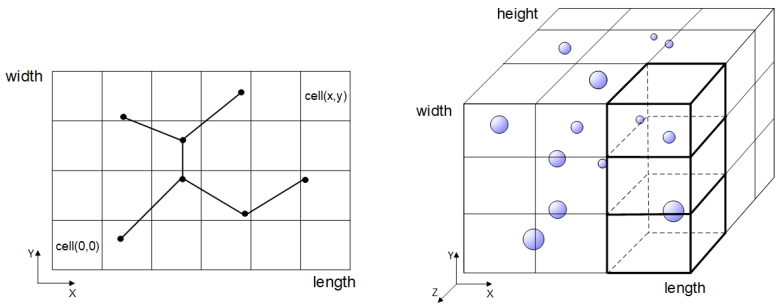
Figure 1: The four main steps of building EMST in Borůvka's framework [2].

Table 1: Operators used to solve FindMin1 in related work. Italic fonts mean that these operators are proposed in this paper.

-PU \ o	2D NNS	KD NNS	2D Component-based NNS	3D Component-based NNS
CPU	K-d tree[19] Bentley Spiral Search [27]	K-d tree [19] Dual tree [6]	Sliced Spiral Search [2]	
GPU	K-d tree [28] Bentley Spiral Search [29]	K-d tree [28] *K-d search*	Sliced Spiral Search [2] *2-d search with dynamic/static pruning*	*3-d search with dynamic/static pruning*

proposed by Qiao et al.[2], who proposed a sliced spiral search method that is a kind of component-based 2D NNS. Key difference lies in that many points will belong to the same component when local MST grows big, so that many points' exact K-d NNS become meaningless. And ***component-based K-d NNS*** regards all vertexes of the component as a whole, and finds the whole's closest outgoing point that belongs to another component.

For exact K-d NNS, except for brute force computing which we do not consider in this paper, researchers often prefer to use K-d tree NNS method. K-d tree NNS [19] and various improved tree data structures such as dual-tree [6], VP-tree [30], Ball-tree [31] work based on hierarchical binary search tree according to recursive K-d spatial partition. While, GPU parallelism of constructing high dimensional K-d tree data structure was recently solved by Hu et al. in 2015 [28], and it is complex to both consider the requirement of memory occupation and decentralized control with which each point works independently.



(a) Cellular partition of 2D plane

(b) Cellular partition of 3D Euclidean space

Figure 2: Cellular partition of 2D/3D Euclidean space with square topology.

Other kind of exact K-d NNS, like Elias' methods [32, 33], work based on partitioning the Euclidean K-dimensional space into congruent and non-overlapping sub-regions, cells, or bins, called **cellular partition**, which is conceptually shown in Fig.2. All input Euclidean points define an axes aligned bounding box with size of "width, height, depth, ..." of a whole Euclidean space. The quantity of cells, or the individual area occupied by each cell, is scalable. Research

Table 2: Operators used to solve FindMin2 in related work. Italic fonts mean that these operators are proposed in this paper.

-PU \ o	Tree Traversal	Union-find [35]
CPU	Deep-first Search (DFS)[36] Breadth-first Search (BFS)[36]	common method
GPU	Breadth-first Search [22] Two-direction Breadth-first Search [2] *Distributed dynamic link list*	Union-find with parallel BFS[1] [2] *Union-find with parallel Link list*

interests lie in that both construction of cellular partition and NNS can be parallelized with characteristic of "data parallelism, decentralized control and each thread occupies $O(1)$ local memory". With cellular partition, each cell contains a list of the points that fall within the cell's boundaries. When a query point q comes in, **Elias' NNS** approach firstly searches the cell where q is located, then passes to search these neighbor cells that are close to the starting cell [34]. **Bentley's spiral search** [27] belongs to a kind of Elias' approaches through accessing neighbor cells in a spiral manner. Once one point is found, it is guaranteed that there is no need to search any other cells that do not intersect the circle of radius equals to the distance to the first point found and centered at the query point [27, 2]. The primal GPU implementation of Bentley's spiral search has been mentioned in the work proposed by Zhang et al.[29]. Qiao et al. [2] combine Bentley spiral search with the uniqueness property in Euclidean space, and propose **sliced spiral search** to build 2D EMSF/EMST. This paper, we propose K-d search and 2D/3D component-based NNS for FindMin1, which works based on congruent and non-overlapping square/cubic space partition.

To solve FindMin2, since size of an independent component can not be predicted, GPU parallel tree traversal has to traverse all vertexes of a component and select the shortest outgoing edge. Traditional depth-first search (DFS) [36] cannot directly run on GPU since DFS works in a recursive manner which current CUDA platform can not support. Hu et al. [28] transfer DFS recursive

manner into an iterative manner using array-based priority queues on GPU for parallel K-d tree NNS. Classical Breadth-first Search (BFS) [36] can directly work on GPU [22], but needs a local variable to store frontier nodes whose size depending on the input tree size. While, GPU can not support launching large amount of independent BFSs with each thread occupying a large-size local variable. Harish et al.[22] explore parallelism between frontier nodes of BFS to work in a manner of "data parallel, decentralized control and each thread occupies $O(1)$ local memory on GPU". However, when each frontier node works independently and in parallel, communication problems arises for selecting the shortest outgoing edge. Qiao et al. [2] proposed GPU two-directional BFS, which also adopts an iterative manner similar to Harish et al. [22] to deal with frontier nodes from root to leaves, but adds a backtrack procedure from leaves to the root again to select the shortest outgoing edge of a local MST. While, this BFS iterative manner needs additional parallel reduction kernels to judge termination at each iteration, leading to the limit of acceleration factors. This paper, we proposes GPU distributed linked list and CUDA kernels to implement tree traversal.

After FindMin2, as shown in Fig.1, the shortest outgoing edge has been found for each component, the rest graphical operations need to connect these closest neighbor components and merge them into new big component. This procedure is generally accomplished by union-find algorithm [35]. GPU parallel union-find can be implemented based on GPU parallel BFS to firstly construct a root merging graph[2], but this manner has been proved slower than sequential union-find implementation [2]. In this paper, we implement GPU parallel union-find with linked list and CUDA kernels.

The following paper is organized as this: section 3 theoretically explain the proposed 2D/3D component-based NNS based on square/cubic cellular partition; section 4 explains the proposed GPU implementation of key operators; section 5 shows simulation results; and section 6 concludes this paper.

[2] $https://stanford.edu/\ rezab/classes/cme323/S15/projects/parallel\_union\_find\_presentation.pdf$

Table 3: Variables used to explain the proposed 2D/3D Component-based NNS.,

Variables	Purpose	Variables	Purpose
q	a query point	p, $p_{i,j}$, u, w	an arbitrary point
C, C_q, $C_{neighbor}$	Cell ceter	L, step length	length/width of a square/cubic cell

3. 2D/3D Component-based NNS based on Square/Cubic Cellular Partition

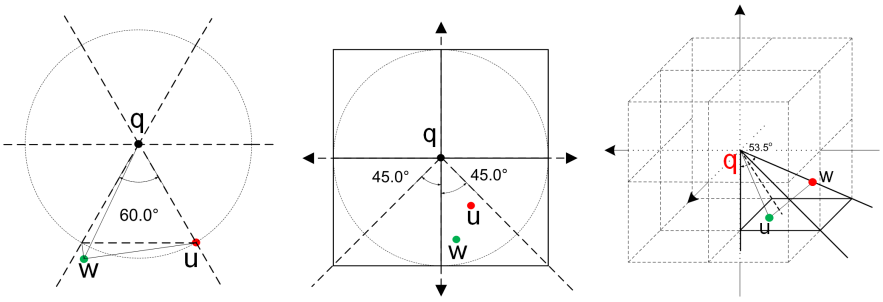
Based on congruent and non-overlapping cellular partition in K-d space, sort of Elias' NNS algorithms with respect to a query point q mainly concerns two aspects: the way to access neighbor cells centering at the current cell C_q where q is located, and the total quantity of neighbor cells needed to traverse for finding the nearest neighbor. In 2D space, the classical Bentley spiral search manner works well to find the nearest neighbor at a little cost of accessing the least quantity of neighbor cells. However, in 3D or higher K-d space, this spiral manner should be re-designed to achieve that goal. Table 3 shows the variable used to explain the proposed algorithms.

Basic idea of component-based K-d search is to combine the advantage of *uniqueness property* with NNS in K-d Euclidean space [2], if all the closest points in separate regions possessing uniqueness property with respect to point q belong to the same component, then q would never have chance to constitute this component's closest outgoing edge. While in higher K-d space, uniqueness property is not same to its 2D version. Based on uniqueness property, we propose two ways to prune unnecessary cells' search in both 2D and 3D space.

3.1. Uniqueness Property

As shown in Fig.3 (a), given a query point q in K-d Euclidean space, a partitioned region $\mathbb{R}$ centering at q has the *uniqueness property* with respect to q if for every pair of points $u, w \in \mathbb{R}$, $\|wu\| < max\left(\|wq\|, \|uq\|\right)$ [37]. In 2D space,

the maximum partitioned region centering at point q is the circulator sector with 60°, while the equal case where $\|uq\| = \|wq\|$ should be carefully treated [2]. In K-D Euclidean space, because every 3 points construct a 2D plane, the maximum partitioned region that has uniqueness property centering at point q is a partitioned region constructed by $K-1$ dimensional partitioned region that possesses uniqueness property.



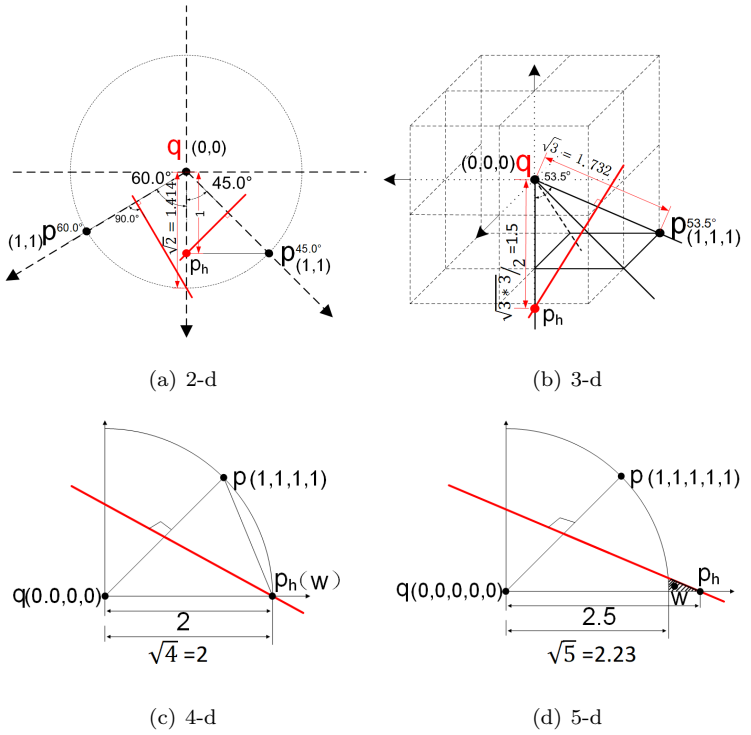
(a) (b) (c)

Figure 3: Slab spatial partition that possesses uniqueness property based on square/cubic cellular partition of the 2D/3D space.

Due to the square/cubic topology of 2D/3D cellular partition, it can easily educe other spatial partitions possessing uniqueness property. For example, in 2D square cellular partition shown in Fig.3(b), centering at the query point q and along the 4 coordinate axes from q, it exists 8 equal 45° circulator sectors possessing uniqueness property. While, in 3D cubic cellular partition, centering at q and along with the coordinate axes and vertexes of a cubic cell, a region with bold line shown in Fig.3(c) possesses uniqueness property, the maximum triangle with q as vertex is 53.5° in this region. Each 3D cubic cell has 6 faces and each face has 4 such regions. We call each 2D sector or 3D region possessing uniqueness property centering at q as a "slab".

However, the square/cubic topology has its limitations when expanding uniqueness property in higher 4D or K-d space. As shown in Fig.4, start from q, along with one axes direction in the K'th dimension and connect the furthest point p with coordinate (1,1) or (1,1,1) or (1,1,1,1) or (1,1,1,1,1), $\angle pqp_h$ defines

the maximum triangle located in the region. The red lines in Fig.4 indicate the hyperplane between q and p. Point p_h indicates the intersecting point between the hyperplane (red line) and the axes in the Kth dimension. $\angle pqp_h$ equals to 60° in 4D Euclidean space, while this angle is larger than 60° in 5D space.



(a) 2-d (b) 3-d

(c) 4-d (d) 5-d

Figure 4: Analysis of uniqueness property defined by 2D/3D/4D/5D square/cubic cellular partition.

3.2. K-d search

Generally, working on square/cubic cellular partition shown in Fig.2, from current cell C_q, one K-d search enters into neighbor cells along every one of the $2 \times K$ coordinate axes directions with step length $m * L$ ($m = 1, 2, 3..$), where L indicates the axes-parallel distance between every two closest cell centers, and then expands search to the rest $K - 1$ dimensional space with the same step length $m * L$ until the stop searching criteria are satisfied, as shown in Fig.5.

Take 3D search as an example and make each cell center as an abstract node, Fig.6 (a,c,e,g) shows the way to search cells along one of the 6 coordinate axes directions.

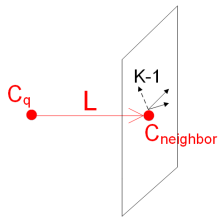


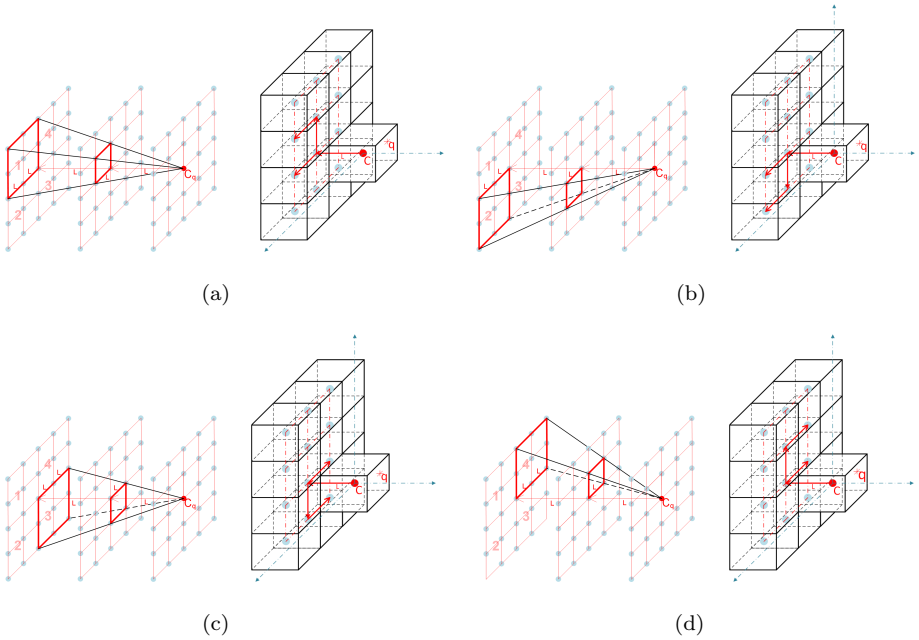
Figure 5: General K-d search manner

The overall stop searching criteria take advantage of hyperplane between q and the first point p found. Given the distance d_{max} between q and the first neighbor point p as shown in Fig.7, there is not need to search cells located further than $(d_{max}/L+1)*L$ to the cell center C_q. This is due to the hyperplane between q and p defines two regions: one shallow region where all points are closer to point p than to q, shown in Fig.7(a); another shallow region that does not need to be searched for q's NNS, shown in Fig.7(b).

3.3. Pruning techniques

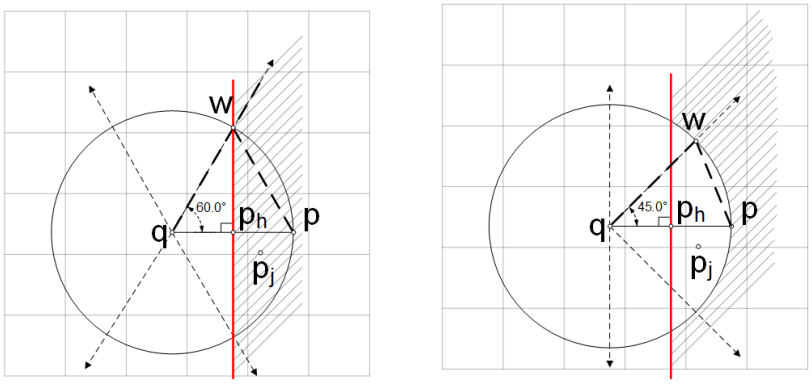
Using pure K-d search takes too much time to find a query point q's closest outgoing point when a local component is too much big. The component-based NNS prunes many points' closest outgoing point search according to the uniqueness property explained in section 3.1. While, there are two ways to prune unnecessary neighbor cells' access.

One is a dynamical way that the algorithm searches each point's closest outgoing point at each iteration, once one closest point p_i in one slab has been confirmed belonging to the same component with q, this slab is closed for later search. The other one is a static way that the algorithm firstly searches all the closest points p_i in all separate slabs, then in later iterations there is no K-d NNS procedure until final EMST is achieved. These two procedures are explained together in Algorithm 2.



(a) (b) (c) (d)

Figure 6: K-d search manner to access neighbor cells one face after another and one slab by one slab with same step length. (a-d) shows the algorithm searches four slabs after entering one face of the cubic cell C_q where the query point q locates. Round blue points indicate cell centers C_q, C of each cell.



(a) (b)

Figure 7: Hyper-plane (red) between the query point q and the first neighbor point p found in K-d Euclidean space.

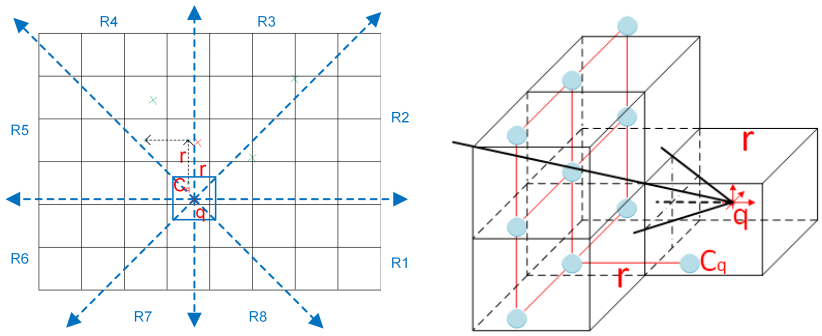


Figure 8: Component-based 2D/3D NNS based on uniqueness property defined by square/cubic cellular partition.

4. GPU Parallel Operators to Build EMSF/EMST

We implement all the proposed operators, like GPU parallel construction of cellular partition, 2D/3D component-based NNS, tree traversal and union-find, using different graph representations and CUDA kernels.

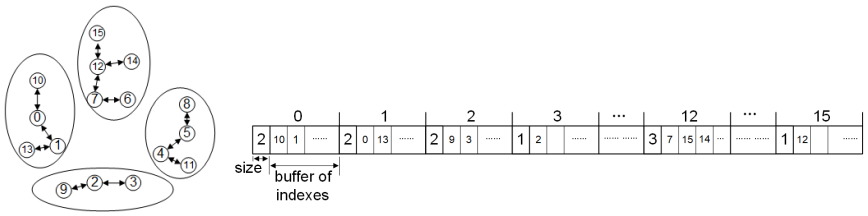
Since GPU models treat memory as general arrays [28, 21, 22], we also implement all graph representation and operators upon arrays, labeled as "grid", $g < node >$, of a certain type of *element* or *node*, for example, the input points are stored in a grid g_p, each point g_p has its id that is indice on the grid. Table 4 illustrates key variables used in this paper.

Taking an EMSF instance $G' = (V', E')$ shown in Fig.9(a) as an example. Firstly, the whole EMSF G' is represented by doubly linked vertex list (DLVL) shown in Fig.9(b), where the i's node $i \in [0, N)$ contains a bounded buffer to store its edge list (also called **links**), and a "size" to indicate number of links. Secondly, each local EMST component of G' is represented in two ways. One way is disjoint set data structure (DSS) shown in Fig.10(a-d). DSS is implemented as an array of indices shown in Fig.10(e), where the i's node $i \in [0, N)$ contains a pointer to i's parent node in the tree. The other way is link list connecting the root and all leaves one by one, until reaching the last node that is characterized by an index with a dummy value such as -1, as shown in Fig.11(a-d), where the i's node $i \in [0, N)$ stores a pointer to current node's next

Table 4: Variables used to illustrate the proposed GPU operators.

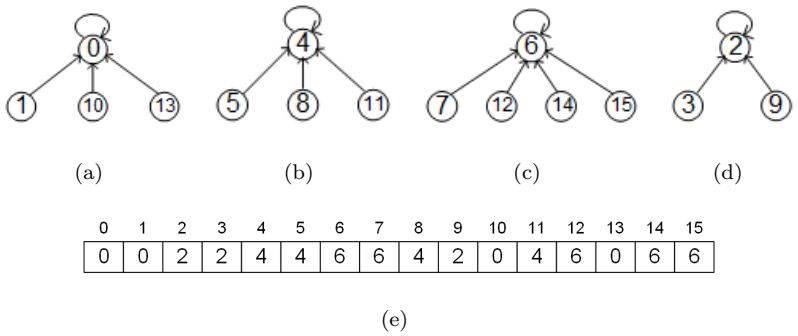
Variables	Purpose	Variables	Purpose
g_p	grid of N input Euclidean Points	id,tid	index of a point located in g_p
g_{cm}	cellular partition	$cellId$	index of a cell located in g_{cm}
g_{corr}	grid of each point's closest outgoing point, selected by FindMin1	i_{win} i_{nex} i_{corres}	index of a winner or next point, or the winner's corresponding point located in a grid
g_{win}	grid of each component's winner vertex possessing the component's shortest outgoing edge, selected by FindMin2	corres	index of a point's closest outgoing piont
g_{dss}	grid of disjoint set trees	root, $r_{i,j}$	root of a local EMST
g_{ddll}	grid of distributed dynamic link list	link	index of a point's link
g_{dist}	grid of each point's outgoing distance	dist	Euclidean distance between two points
g_{dlvl}	grid of doubly linked list	L	The searching step length from current cell C_q where q locates
g_{ssm}	grid of each point's component-based NNS results	L_{max}	The max topological distance need to be searched theoretically

node along the link list. Since local EMST of current EMSF grows dynamically at different iterations, and all local MEST's component lists are implemented in one array shown in Fig.11(e), it is called distributed dynamic link list (DDLL).



(a) (b)

Figure 9: (a) EMSF instance consists of local EMSTs; (b) Data structure of doubly linked vertex list (DLVL) to represent EMSF shown in (a).



(a) (b) (c) (d)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	2	2	4	4	6	6	4	2	0	4	6	0	6	6

(e)

Figure 10: (a-d) Disjoint set trees (DSS) to represent local EMSTs shown in Fig.9(a); (e) Data structure of DSS.

4.1. GPU Parallel Construction of K-d Cellular Partition

We construct the K-d cellular partition on GPU through assigning one thread to one input Euclidean point. As shown in Alg.1, each point associated with a thread tries to find the cell where it is located. All N points can be inserted into cellular partition in parallel.

4.2. GPU Parallel 2D/3D Search with Pruning Technique

For 2D/3D component-based NNS with dynamic/pruning technique based on square/cubic cellular partition, which is explained in section 3.3, we assign

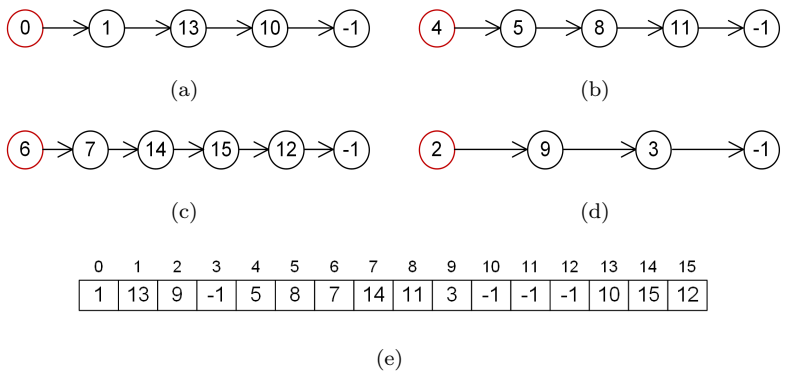


Figure 11: (a-d) Link list to represent local EMSTs shown in Fig.9(a); (e) Data structure of distributed dynamic link list.

ALGORITHM 1: Kernel: construct K-d cellular partition on GPU side with one thread assigned to one input point.

input: g_p with N input Euclidean Points

output: g_{cm}

$tid \leftarrow$ getThreadID ;

if $g_p(tid)$ *is valide* **then**

IndexType cellId = findCell($g_p(tid)$) ; `// find the cell where point` $g_p(tid)$ `locates.`

$g_{cm}(cellId)$ inserts ($g_p(tid)$);

end

each input point a CUDA thread to find each component's shortest outgoing point in parallel. As shown in Alg.2, the dynamic pruning has a procedure to find each point's closest outgoing point, while the static way only choose to find all separate closest point in independent slabs.

4.3. GPU Parallel Tree Traversal with Distributed dynamic link list

Instead of using GPU parallelism of BFS or DFS for tree traversal to collect the shortest outgoing edge of a local EMST component, we combine the disjoint set data structure with GPU lock-free parallel link list data structure proposed by Nyland et al. [38] to accomplish this task.

It contains two independent GPU CUDA kernels illustrated in Alg.3 and Alg.4. Alg.3 starts from independent point, finds root of this point and adds itself to the root's link list using GPU CUDA lock-free Atomic operation [38]. Alg.4 traverses independent link lists from their root nodes to collect the shortest outgoing edge within each component.

4.4. GPU Parallel Union-find with link list

We divide GPU parallelism of union-find algorithm into two independent CUDA kernels. First, *Union* operation updates the new root of the connected old root nodes, which can be executed in the same kernel with the *Connect* operation shown in Alg.5. Second, *Flattening* operation updates all leave nodes' root to the same new root, as shown in Alg.6.

5. Experiments

Due to the fact that few algorithms exist in current literature that produce an exact Euclidean MST, and the most optimal sequential EMST algorithm that in current literature is the state-of-the-art "dual-tree EMST" algorithm [6] from Mlpack library [3]. Experimental comparisons are designed as the following. We ignore sequential EMST implementations using Prim's, Borůvka's,

[3] https://www.mlpack.org/

ALGORITHM 2: 2D/3D component-based NNS based on square/cubic cellular partitions with dynamic/static pruning technique.

input: g_p, g_{cm}, g_{dss}

output: Closest outgoing point *corres*, all slab points slab[j], g_{ssm}, g_{dis}, g_{corr}.

$tid \leftarrow$ getThreadID ;

if $g_p(tid)$ *is valide* **then**
  while $L <= L_{max}$ **do**
    for *each face* **do**
      if $L > dist_{min}$ ***or*** *current face locates on the boundary of* g_{cm} **then**
        current face is closed to be searched;
      end
      if *current face has to be searched* **then**
        for *each slab of current face* **do**
          check slab is previously closesd or not ;
          if *current slab has to be search* **then**
            for *each cell in current slab* **do**
              for *each point* p_i *in current cell* **do**
                if *dynamic pruning* && *point* p_i, $g_p(tid)$ *in different component* **then**
                  update *corres*, $dist_{min}$;
                end
                if *static pruning* || *(dynamic pruning* && *point* p_i, $g_p(tid)$ *in same component)* **then**
                  check if p_i locates in a $slab[j]$;
                  if p_i *in slab[j]* **then**
                    update $slab[j].corres \leftarrow p_i$;
                    update $slab[j].dist \leftarrow dist(p_i, g_p(tid))$;
                  end
                end
              end
            end
          end
        end
      end
      L ++;
  end
end

ALGORITHM 3: Kernel: construct distributed dynamic link list.

input: g_{corr}, g_{dss}

output: g_{ddll}

$tid \leftarrow$ getThreadID;

if *tid is valid grid index* **then**

  if $g_{corr}(tid) \neq -1$ **then**

    $root \leftarrow g_{dss}(tid)$;

    if $root \neq tid$ **then**

      old, link = $g_{ddll}(root)$;

      do

        old $\leftarrow$ link;

        $g_{ddll}(tid) \leftarrow old$;

        link = atomicCAS($g_{ddll}(root), link, tid$);

      while $link \neq old$;

    end

  end

end

ALGORITHM 4: Kernel: find minimum outgoing edges by traversing each link list.

input: $g_{ddll}, g_{dss}, g_{corr}, g_{dist}$

output: g_{win}

$tid \leftarrow$ getThreadID;

if *tid is valid grid index* **then**

  $root \leftarrow g_{dss}(tid)$;

  if $root == tid$ **then**

    $i_{win} \leftarrow tid$;

    $i_{nex} \leftarrow tid$;

    while i_{nex} *exists* **do**

      $dist \leftarrow g_{dist}(i_{nex})$;

      $corres \leftarrow g_{corr}(i_{nex})$;

      Select minimum $dist$, update $i_{win} \leftarrow i_{nex}$;

      $i_{nex} \leftarrow g_{ddll}(i_{nex})$;

    end

    $g_{win}(i_{win}) \leftarrow true$;

  end

end

ALGORITHM 5: Kernel. Connect and union components.

input: g_{win}, g_{corr}, g_{dss}, $g'_{dss} \leftarrow g_{dss}$

output: g_{dlvl}, g_{dss}

$tid \leftarrow$ getThreadID;

if $g_{win}(tid)$ *is a winner node* **then**

  $corres \leftarrow g_{corr}(tid)$;

  $g_{dlvl}(tid) \leftarrow corres$; `// Add` *corr* `to` i_{win}`'s adjacency list.`

  $r_{win} \leftarrow g'_{dss}(tid)$;

  $r_{corres} \leftarrow g'_{dss}(corres)$;

  if *(corres is not a winner node)* $\|$ *(corres is winner node &&*

   $g_{corr}(corres) \neq tid$ **then**

    Atomic-Insert $g_{dlvl}(corres) \leftarrow tid$;

    $g_{dss}(r_{win}) \leftarrow r_{corres}$;

    else if $r_{win} < r_{corres}$ **then**

      $g_{dss}(r_{corres}) \leftarrow r_{win}$;

    end

  end

end

ALGORITHM 6: Kernel. Flattening component.

input: g_{dss}, $g'_{dss} \leftarrow g_{dss}$

output: g_{dss}

$tid \leftarrow$ getThreadID;

if *tid is valid* **then**

  $r \leftarrow tid$;

  while $r \neq g_{dss}(r)$ **do**

    $r \leftarrow g_{dss}(r)$

  end

  $g_{dss}(tid) \leftarrow r$; `// update root`

end

Kruskal's algorithm since their quadratic time complexity and they have been tested running much slower than sequential sliced spiral search [2]. While, GPU parallel Bentley spiral search [39, 40] and the sliced spiral search [2] all work in 2D space, only dual-tree mlpack EMST [6] can build 3D EMSF/EMST. We firstly compare the efficiency of our newly proposed GPU operators working on 2D input instances. Then, we provide an integral GPU parallel EMSF/EMST algorithm, and compare it mainly with dual-tree mlpack EMST.

Source code of our proposed GPU parallel 2D/3D EMSF/EMST algorithm in this paper has been put on GitHub[4]. We implemented all proposed algorithms using C/C++, CUDA Toolkit v9.1, and QT creator as a cross-platform compiler. Unless specified, code is compiled on laptop with CPU Intel(R) Core(TM) i7-4710HQ, 2.5 GHz with 8GB of RAM running Windows, GPU card GeForce GTX 850M. CUDA configuration is set fixedly as <<<N/128 + 1, 128>>>.

Test benchmarks include 23 2D uniform distributed data sets, 23 2D benchmarks for Euclidean National Traveling Salesman Problems (TSP) offered by TSPLIB [41], and 10 larger 3D uniform distributed data sets. These data sets simulate a wide range of possible data distribution in Euclidean space.

About evaluation items, since we focus on GPU parallelism, we put emphasis on firstly, the running time of different algorithms; secondly, whether the quantity of total edges of EMST result built for the same N points equals to $N-1$; thirdly, whether total length of the same instance's EMST result remains the same at different executions of an GPU parallel algorithm. The last two items have been confirmed to remain the same in all the following experimental comparisons. Each test instance has been executed more than 10 times to confirm the conclusion obtained.

[4] $https://wenbaoqiao.github.io/Component-based-2D-3D-NNS-to-GPU-parallel-2D-3D-EMST/$

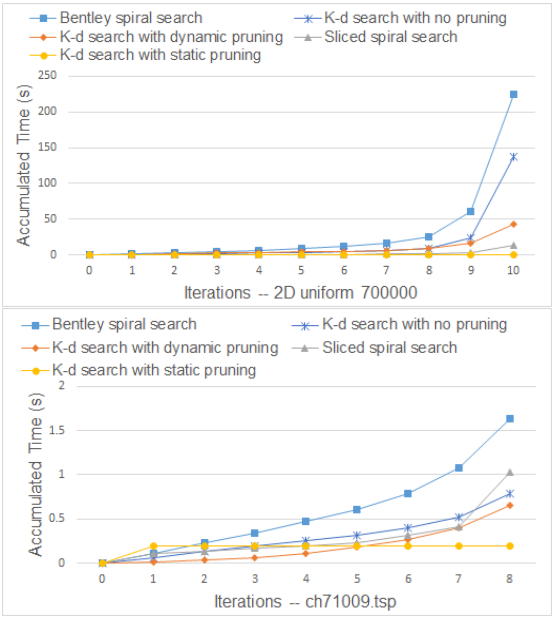


Figure 12: Accumulated time of Find Min 1 with different NNS algorithms in different iterations to build EMST for 2D uniform-700000 and ch71009.tsp benchmarks.

5.1. Evaluation of Independent GPU Operators

For operators to solve FindMin1, we compare GPU parallel EMST algorithms using separate Bentley spiral search, pure K-d search with no pruning, K-d search with dynamic pruning, K-d search with static pruning technique.

Fig.12 shows the accumulated time taken by these operators at different iterations of building EMSF/EMST for 2D uniform 700000 and ch71009.tsp [41] benchmarks. It is clear to see that using exact NNS to find each point's closest outgoing point takes more time in later iterations because size of each component grows large, such as Bentley spiral search and K-d search with no pruning. While, the component-based NNS take less time in later iterations than pure NNS, such as K-d search with dynamic pruning and sliced spiral search. The sliced spiral search has unstable performance when running on ch71009.tsp and takes more time than K-d search with no pruning, but it takes less time when running on uniform 700000 instance. This is because sliced

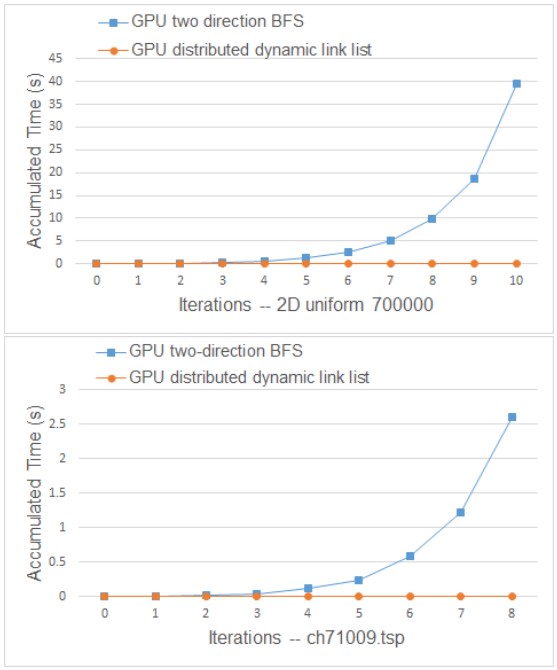


Figure 13: Accumulated time of Find Min 2 with GPU two-direction BFS and distributed dynamic link list at different iterations to build EMST for 2D uniform-700000 and ch71009.tsp benchmarks.

spiral search adopt a different manner to check slabs and cells that do not need to access, which is sensitive to input data distribution. However, the K-d search with static pruning technique performs much more stable in different iterations. This is because the static pruning strategy only perform component-based NNS at the first iteration, and does not in later iterations.

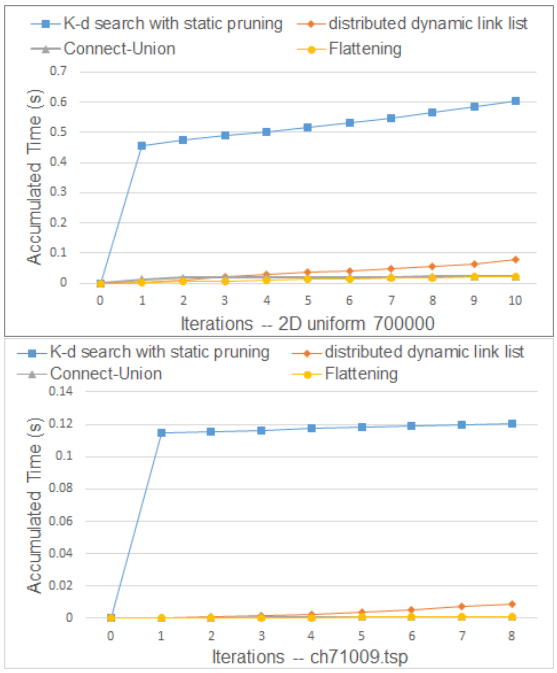


Figure 14: Accumulated time of the winner GPU kernel operators in different iterations to build EMST for 2D uniform-700000 and ch71009.tsp benchmarks.

For operators to solve FindMin2 using GPU tree traversal algorithms, since most related work in literature applies BFS, we mainly compare the GPU two directional BFS [2] and our newly proposed GPU distributed dynamic link list. Fig.13 shows the accumulated time taken by the two operators at different iterations of building EMSF/EMST for 2D uniform 700000 and ch71009.tsp [41] benchmarks. It is clear to see that GPU two-direction BFS takes much more time in later iterations when size of component grows large, while using dynamic link list takes much more less time. This is because the iterative GPU

parallel BFS proposed by Harish et al.[22] and applied by Qiao et al.[2] all need additional parallel reduction kernels to judge temination at each BFS iteration. While, the construction of distributed dynamic linked list uses CUDA lock-free atomic operations, and the traversal of linked list performs in single kernel, which is very fast when all linked lists perform in parallel.

For rest operators like connect-union and flattening, their performance mainly depends on parallel union-find operator, which can be implemented by GPU BFS [5] or dynamic link list. While, there is no need to compare GPU BFS and GPU distributed dynamic link list again. Fig.14 shows the accumulated time of these winner operators in different iterations to build EMST for 2D uniform 700000 and ch71009.tsp benchmarks. It is clear to see that time taken by these winner operators grow linearly.

5.2. Evaluation of the Whole Algorithm

Combine the proposed GPU parallel K-d search with static pruning technique, tree traversal and union-find operators with dynamic link list to be a whole GPU parallel EMSF/EMST algorithm, we name it as "K-d search with static pruning EMST" and compare it with current optimal sequential EMST algorithm, namely dual-tree mlpack EMST.

In 2D space, Fig.15 shows the overall running time of these two algorithms to build exact EMST for 23 instances with uniform data distribution and 23 national TSP instances with arbitrary data distribution. It is clear to see K-d search with static pruning EMST takes less time than dual-tree mlpack EMST.

In 3D space, Fig.16 also shows that K-d search with static pruning EMST runs faster than dual-tree mlpack EMST algorithm. And since some GPU card series share the same CUDA platform and our proposed GPU parallel algorithm follows "data parallelism, decentralized control, and each thread occupies $O(1)$ local memory", our proposed GPU parallel EMST can run on more advanced GPU cards. As shown in Fig.16, the running time comparison between GTX

[5] $https://stanford.edu/\ rezab/classes/cme323/S15/projects/parallel\_union\_find\_presentation.pdf$

850M, GTX 980M and GTX1080Ti shows our proposed algorithm can achieve more acceleration factors along with the development of GPU hardware.

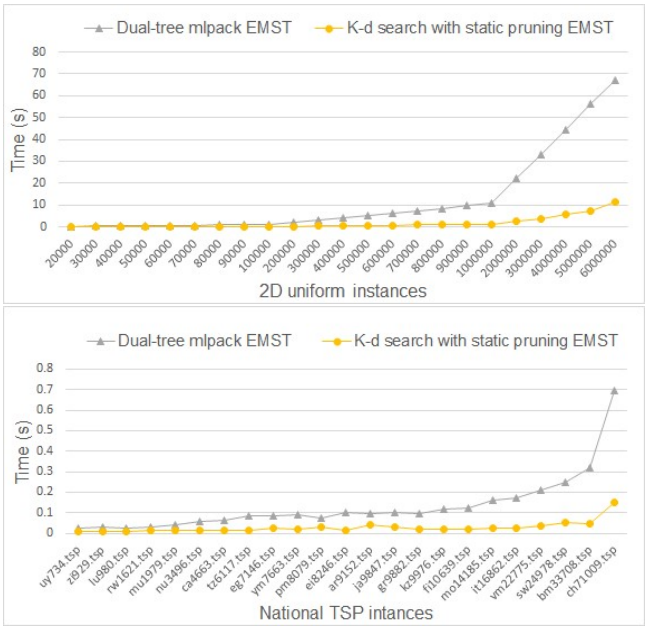
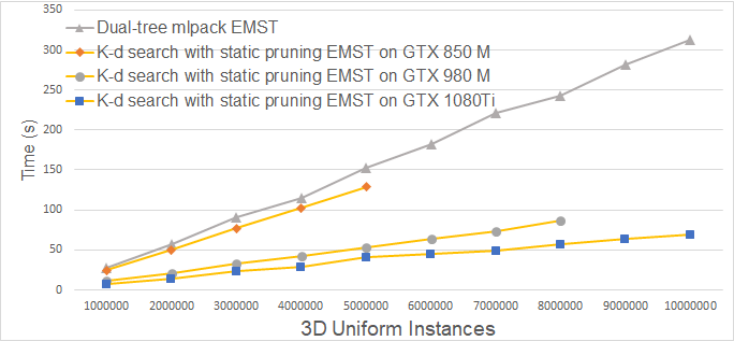


Figure 15: Overall running time comparison between of our proposed K-d search with static pruning EMST and dual-tree mlpack EMST in 2D space.

6. Conclusion

This paper explores GPU parallelism of building 2D/3D EMST/EMSF in Borůvka's framework with every key step working on GPU side and the integral implementation runs faster than current optimal sequential EMST and previous GPU parallel EMST algorithms. Parallelism of these operators follow "data parallelism, decentralized control and $O(1)$ local memory for each GPU thread", which enables further acceleration on more advanced GPU cards.

For FindMin1 operators, we propose a general K-d NNS search based on congruent and non-overlapping cellular partition with square topology, and improve it with pruning technique in 2D/3D space for finding a local component's closest outgoing point. Also, we explore two different pruning manners, one is



(a)

Figure 16: Overall running time comparison between our proposed K-d search with static pruning EMST and dual-tree mlpack EMST in 3D space. And the comparison of acceleration factors achieved along with the development of GPU cards.

dynamic and the other is static pruning. Experimental comparison proves that the static pruning technique save more running time.

For FindMin2 operators and union-find, we jump out of GPU parallel BFS or DFS algorithms for tree traversal, propose to apply linked list data structure on GPU and profit from the CUDA atomic operations to achieve higher running speed. Though GPU parallelism prefer coalesced memory access on arrays, here because size of independent component can not be predicted, linked list is a compromised option.

The proposed methods provide basis for some higher level distributed or parallel algorithms in both 2D and 3D Euclidean space. Application to other Euclidean problems can be envisaged with high efficiency and within divide-and-conquer scheme, for example 3D traveling salesman problem, hierarchical 2D/3D data clusters and apply them to 2D/3D image processing such as optical flow or stereo matching problems.

7. Acknowledgements

This paper is sponsored by Chinese Scholarship Council. Thank you very much.

References

[1] O. Boruvka, O jistém problému minimálním (1926).

[2] W.-b. Qiao, J.-C. Créput, Gpu implementation of borůvka's algorithm to euclidean minimum spanning tree based on elias method, Applied Soft Computing 76 (2019) 105–120.

[3] L. Li, X. Yu, S. Zhang, X. Zhao, L. Zhang, 3d cost aggregation with multiple minimum spanning trees for stereo matching, Applied Optics 56 (12) (2017) 3411–3420.

[4] D. Scharstein, R. Szeliski, A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, International journal of computer vision 47 (1-3) (2002) 7–42.

[5] Y. Xu, V. Olman, D. Xu, Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees, Bioinformatics 18 (4) (2002) 536–545.

[6] W. B. March, P. Ram, A. G. Gray, Fast euclidean minimum spanning tree: algorithm, analysis, and applications, in: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2010, pp. 603–612.

[7] C. Zhong, D. Miao, P. Fränti, Minimum spanning tree based split-and-merge: A hierarchical clustering method, Information Sciences 181 (16) (2011) 3397–3410.

[8] L. An, Q.-S. Xiang, S. Chavez, A fast implementation of the minimum spanning tree method for phase unwrapping, IEEE transactions on medical imaging 19 (8) (2000) 805–808.

[9] Y. Xu, E. C. Uberbacher, 2d image segmentation using minimum spanning trees, Image and Vision Computing 15 (1) (1997) 47–57.

[10] N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group (1976).

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet of Things Journal 3 (5) (2016) 637–646.

[12] J. Nešetřil, E. Milková, H. Nešetřilová, Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history, Discrete mathematics 233 (1-3) (2001) 3–36.

[13] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical society 7 (1) (1956) 48–50.

[14] R. C. Prim, Shortest connection networks and some generalizations, Bell system technical journal 36 (6) (1957) 1389–1401.

[15] B. Chazelle, A minimum spanning tree algorithm with inverse-ackermann type complexity, Journal of the ACM (JACM) 47 (6) (2000) 1028–1047.

[16] A. Lingas, A linear-time construction of the relative neighborhood graph from the delaunay triangulation, Computational Geometry 4 (4) (1994) 199–208.

[17] M. I. Shamos, D. Hoey, Closest-point problems, in: Foundations of Computer Science, 1975., 16th Annual Symposium on, IEEE, 1975, pp. 151–162.

[18] J. L. Bentley, J. H. Friedman, Fast algorithms for constructing minimal spanning trees in coordinate spaces, IEEE Trans. Comput. 27 (STAN-CS-75-529) (1975) 97.

[19] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.

[20] S. Rajasekaran, On the euclidean minimum spanning tree problem, Computing Letters, 2004 1 (1) (2004).

[21] V. Vineet, P. Harish, S. Patidar, P. Narayanan, Fast minimum spanning tree for large graphs on the gpu, in: Proceedings of the Conference on High Performance Graphics 2009, ACM, 2009, pp. 167–171.

[22] P. Harish, V. Vineet, P. Narayanan, Large graph algorithms for massively multithreaded architectures, International Institute of Information Technology Hyderabad, Tech. Rep. IIIT/TR/2009/74 (2009).

[23] S. Nobari, T.-T. Cao, P. Karras, S. Bressan, Scalable parallel minimum spanning forest computation, in: ACM SIGPLAN Notices, Vol. 47, ACM, 2012, pp. 205–214.

[24] S. Chung, A. Condon, Parallel implementation of bouvka's minimum spanning tree algorithm, in: Parallel Processing Symposium, 1996., Proceedings of IPPS'96, The 10th International, IEEE, 1996, pp. 302–308.

[25] K. W. Chong, Y. Han, T. W. Lam, Concurrent threads and optimal parallel minimum spanning trees algorithm, Journal of the ACM (JACM) 48 (2) (2001) 297–323.

[26] D. A. Bader, K. Madduri, Designing multithreaded algorithms for breadth-first search and st-connectivity on the cray mta-2, in: Parallel Processing, 2006. ICPP 2006. International Conference on, IEEE, 2006, pp. 523–530.

[27] J. L. Bentley, B. W. Weide, A. C. Yao, Optimal expected-time algorithms for closest point problems, ACM Transactions on Mathematical Software (TOMS) 6 (4) (1980) 563–580.

[28] L. Hu, S. Nooshabadi, M. Ahmadi, Massively parallel kd-tree construction and nearest neighbor search algorithms, in: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, 2015, pp. 2752–2755.

[29] N. Zhang, H. Wang, J.-C. Creput, J. Moreau, Y. Ruichek, Cellular gpu model for structured mesh generation and its application to the stereo-matching disparity map, in: Multimedia (ISM), 2013 IEEE International Symposium on, IEEE, 2013, pp. 53–60.

[30] P. N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, in: Acm-siam Symposium on Discrete Algorithms, 1993.

[31] S. M. Omohundro, Five balltree construction algorithms, International Computer Science Institute Berkeley, 1989.

[32] R. L. Rivest, On the optimality of elia's algorithm for performing best-match searches., in: IFIP Congress, 1974, pp. 678–681.

[33] J. G. Cleary, Analysis of an algorithm for finding nearest neighbors in euclidean space, ACM Transactions on Mathematical Software (TOMS) 5 (2) (1979) 183–192.

[34] M. Greenspan, G. Godin, J. Talbot, Acceleration of binning nearest neighbor methods, Proceedings of Vision Interface 2000 (2000) 337–344.

[35] R. E. Tarjan, J. Van Leeuwen, Worst-case analysis of set union algorithms, Journal of the ACM (JACM) 31 (2) (1984) 245–281.

[36] T. H. Cormen, Introduction to algorithms, MIT press, 2009.

[37] G. Robins, J. S. Salowe, On the maximum degree of minimum spanning trees, in: Proceedings of the tenth annual symposium on Computational geometry, ACM, 1994, pp. 250–258.

[38] L. Nyland, S. Johns, Understanding and using atomic memory operations, in: 4th GPU Technology Conf.(GTC'13), March, 2013.

[39] N. Zhang, Cellular gpu models to euclidean optimization problems: Applications from stereo matching to structured adaptive meshing and traveling salesman problem, Ph.D. thesis, Université de Technologie de Belfort-Montbeliard (2013).

[40] H. Wang, Cellular matrix for parallel k-means and local search to euclidean grid matching, Ph.D. thesis, Université de Technologie de Belfort-Montbéliard (2015).

[41] G. Reinelt, Tsplib—a traveling salesman problem library, ORSA journal on computing 3 (4) (1991) 376–384.