Progressive Exact Distance Transform

Dongxu Wang
State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China
and College of Information Engineering, Xiangtan University, Hunan, China

Wencheng Wang*
State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China

Yuexiang Shi
College of Information Engineering, Xiangtan University, Hunan, China

Abstract

An algorithm is proposed to perform exact distance transform in a progressive manner, based on the popular dimension reduction (DR) algorithm [8], for a binary image in 2D and higher dimensions. It first computes some columns of the image with an interval between them, and then progressively treats the columns in between the computed columns with the interval reduced gradually. In this way, coherences between neighbouring columns can be used to save computation considerably. By experiments, our new algorithm can be faster than the DR algorithm on the CPU by 49.2% ~181.9% for their sequential executions, and by up to 137.1% for their parallel executions. When our algorithm is executed on the GPU with the initial interval being 2 for achieving its highest level of parallelism, it may be even faster than the parallel banding (PB) algorithm [1] in some cases, the fastest one to date, especially when the image has high resolutions, ours tends to be faster. Therefore, our algorithm is very suitable for high precise applications, where images are large and there are more coherences.

Key words: distance transform, Voronoi diagrams, computational geometry

1. Introduction

Euclidean Distance Transform (EDT) plays an important role in many applications such as extracting medial axes and producing Voronoi diagrams. For an N-dimensional grid with some grid points colored, EDT is to compute for every grid point the Euclidean distance from itself to the closest colored grid point. It is closely related to the Voronoi diagram computation where for each grid point we are interested in the actual closest colored grid point or site [1].

Till now, a lot of algorithms have been proposed for EDT computation, some by accurate computation [8] while some by approximation for fast computation[3]. In earlier days, sequential approaches were studied a lot, and two recent papers [4, 5] made a survey on them. It is highlighted that exact EDT can be computed in linear time using a dimensionality reduction approach [8]. For fastness, one way is by hierarchical structures to perform EDT computation progressively[3]. However, it has to run in an approximative manner, because a coarser grid point and its son grid points may have different closest sites, and so causing inconsistency on distance computation and comparison. Thus, for exact EDT computation on large grids, always required in high precise applications, it is very time consumming. For acceleration, many parallel EDT algorithms have

* Corresponding author email: whn@ios.ac.cn

been proposed [6], and much progress has been achieved, especially with the GPU [7, 10]. In 2010, a parallel banding (PB) algorithm was proposed to compute exact EDT with the GPU [1]. It is inspired by the algorithm [6] and can be regarded as a parallelization implementation of the DR algorithm, with grid points partitioned into many bands in every dimension. It is more practical to implement on GPUs, and outperforms all existing EDT algorithms, no matter whether they are exact or approximated. However, to efficiently take adavantages of the GPU, it has to tune the bands for each dimension manually, and in treating very large grids, its efficiency would lower significantly due to limited processors.

In this paper, we present a progressive algorithm to perform exact EDT. It is based on the coherence that if two grid points, say P_1 and P_2, have the same closest site, the site must be the closest one to any a grid point on the straight line segment connecting P_1 and P_2. This is because by Voronoi diagram computation every Voronoi cell is convex, so that for two points inside a Voronoi cell the straight line segment connecting them must be inside the Voronoi cell. According to the strategy of the DR algorithm [4], we may first compute some columns with an interval between them, and then the columns between the computed columns progressively with the interval reduced gradually. In this way, if a grid point of the column under computation is on a line segment connecting two grid points that share a same closest site and from two computed columns, the grid point can share the same closest site also. Therefore, much distance computation and comparison can be saved. While using the measures of the DR algorithm, we further exploit more coherences to reduce the computation cost. By experiments on 2D binary images, our new algorithm can be faster than the DR algorithm on the CPU by 49.2% ~181.9% for their sequential executions, and by up to 137.1% for their parallel executions. As for their execution on the GPU, they can be sometimes faster than the PB algorithm, the fastest exact EDT algorithm to date, especially when the image has high resolutions, ours can get more support from coherences and tend to be faster, even by 61.43%. As a result, our algorithm is more helpful to perform high precise EDT, because more grid points will share a closest site when grids are generated more densely. Another advantage is that unlike the PB algorithm, our algorithm doesn't need to manually tune a lot of parameters for good performance.

In the following, we will first introduce the DR algorithm [8] and the PB algorithm [1] in Section 2, because ours integrates many measures of the DR algorithm, and will be compared with it and the PB algorithm. Then we describe our new algorithm in Section 3, and discuss its parallelization implmenttaion in Section 4. For simplicity, we will discusss our algorithm in the 2D case, i.e. we discuss the computation done in one dimension (for each row) and then in the second dimension (for each column). As for its extension to higher dimensions, it is easy by repeating the computation for each additional dimension. In section 5, we list and discuss the experimental results. And in Sections 6, conclusions are drawn finally.

2. Two Related Algorithms

2.1 Dimension Reduction Algorithm

Consider a 2D binary image of size $N=n\times n$. It is wanted to determine the closest colored pixel (site) to each pixel. This algorithm runs by treating the dimensions one by one.

In the first dimension, every row is treated respectively, where each pixel of a row gets its closest site in this row by performing the left-right sweeps as introduced in [9]. Let $S_{i,j}$ be the nearest site, among all sites in row j, of the pixel (i, j), and let $S_i = \{S_{i,j} \mid S_{i,j} \neq \text{NULL}, j = 0, 1, 2, \ldots, n-1\}$ be the collection of such closest sites for all pixels in column i. Note that $S_{i,j}$ is NULL when there is no site in row j. As illustrated in Figure 1, the circle points represent the nearest sites for the pixels of column i.

Afterwards, every colume is handled respectively, where every pixel gets its closest site in the image. It is separated into two steps. In the first step, for each column i in the image, the set of sites whose Voronoi regions intersect column i is computed from S_i, and the resulted sites form a set, P_i, called the set of the *proximate sites* of column i. Then, in the second step, each pixel of column i gets its the closest site from P_i.

As illustrated in Fig. 1, the sites of S_i are investigated sequentially to produce P_i. For a site in S_i, it is determined whether its related Voronoi cell intersects column i. If it is, the site is included in P_i; otherwise, it is excluded. This is by checking the intersections between column i and the perpendicular bisectors of the investigated site and its neighbouring sites, where perpendicular bisectors are drawn in green in Fig.1. For

example, since intersection In_j is above intersection In_{j-1}, in accordance with that $S_{i,j}$ is above $S_{i,j-2}$, $S_{i,j-1}$ is included in P_i. As for $S_{i,1}$, its related intersection In_2 is below its other related intersection In_1, not in accordance with that S_2 is above S_0, so it is excluded. The correctness of this measures is proved and explained clearly in [1, 8]. Interesting readers please refer to them.

As the sites in P_i and the pixels of column i are ordered by their y coordinates, they can be processed sequentially from top down or from bottom up to compute the closest site to each pixel of column i. At each pixel q, we check the distance of q to the two sites p_1 and p_2, where p_1 is at the front and p_2 is just after p_1 in the list of P_i. If p_1 is closer, then p_1 is the closest site to q, and the process is repeated for the pixel after q. If not, p_1 is removed from the list since it can no longer affect any other pixel from q onward, and use p_2 in place of p_1 as the front of the list to compute the closest site to q.

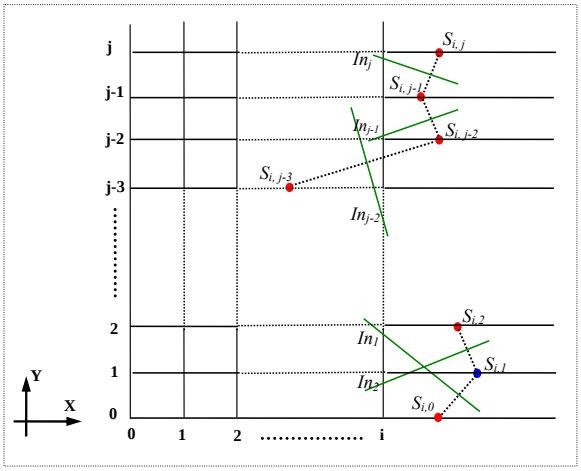

Fig.1. Illustration of the steps of the dimension reduction algorithm.

2.2 Parallel Banding Algorithm

The PB algorithm can be regarded as a parallelization implementation of the DR algorithm. In treating each dimension and even in each operation step, it divides the image into many bands to process the bands respectively in parallel, and then checks the information of the bands to combine them for getting the final results. If more bands are produced, the level of parallelism can be increased, but the cost for combining bands may be also increased to lower the computation efficiency. With regard to this, the number of bands in each step have to be tuned to effectively use processors for obtaining good performance. This is much dependent on the number of processors, and the authors of Ref. [1] made a lot of tests and suggested some good choices for the band numbers in some cases.

3. Progressive Algorithm

EDT computation is closely related to the underlying Voronoi diagram. As illustrated in Fig.2, the image is divided into many voronoi cells represented with convex polygons, and every column is separated into many segments by these polygons. Obviously, the pixels of a same segment must have the same closest site. If two segments from two columns respectively are inside a same voronoi cell, their formed quadrangle must be inside the voronoi cell, as the green, purple and blue quadrangles shown in Fig.2. Thus, the site for such a quadrangle must be a proximate site of these two columns respectively.

By the above discusssion, we design a progressive algorithm to save computation for EDT computation. Our improvements are on columns after rows are treated as in the DR algorithm. At first, some columns are colored with an interval between these columns, using the corresponding techniques in [8]. Then, columns are selected progressively to compute with the interval reduced half by half, till all columns are colored, where each column is computed by using coherence from its two neighbouring computed columns for acceleration.

Without loss of generality, we describe the techqniues by the image in Fig.2. Suppose column *i-3k* and column *i+k* have been handled, we start to treat column *i-k*. At first, we get the intersection set of the set of proximate sites for column *i-3k* and column *i+k*, P_{i-3k} and P_{i+k}. Clearly, every site of the intersection set must have related intersection points with column *i-3k* and column *i+k*, e.g., In_{11}, In_{12}, In_{21} and In_{22} shown in Fig.2, and their formed quadrangle can cover a segment of column *i-k*. The pixels of this segment, say the pixels from In'_1 to In'_2 in Fig.2, can get their closest sites directly. Afterwards, we treat the pixels of column *i-k* that have not been colored, which are organized into many blocks with each consisting of some consecutive such pixels, called *uncolored blocks*, as marked with red ellipses in Fig.2. For each uncolored block of column *i*, we compute its local S_i^b and then its P_i^b from S_i^b, and finally get the closest sites for the pixels of the uncolored block.

3.1 Efficient Computation of S_i^b

According to the analysis in [8], if pixel *A* is above pixel *B* in a column *i*, the closest site of pixel *A* would not be below that of pixel *B*, and vice versa. Thus, S_i^b for an uncolored block can be the set of $\{S_{i,j} \mid S_{i,j} \neq$ NULL, $j = j_1, j_1+1, j_1+2, \ldots, j_2-1, j_2\}$, under the supposition that Q_{h1} and Q_{h2} are the topmost and bottommost pixels of an uncolored block and the closest sites for Q_{h1+1} and Q_{h2-1} are pixels (i', j_1) and (i'', j_2) respectively, as illustrated in Fig.2, where *h1*, *h1+1* and *h2*, *h2-1* represent the *y* coordinates of these pixels. Fortunately, the sites of S_i^b can be reduced a lot by the followed analysis, when column *i* is colored after column *i-k* and column *i+k* are computed. If a site is the closest one to some pixels of column *i* and its *x* coordinate is bigger than *i+k*, it must be in P_{i+k}, because it must have related perpendicular bisectors intersecting column *i+k* and column *i* reasonably. Similar deduction can be done for the sites of $S^b{}_i$ that have their *x* coordinates smaller than *i-k*, when considered with column *i-k*. As a result, $S^b{}_i$ is the union of the following three sets of sites, where if there are more than one sites in a row, only the one nearest to column *i* remains while the others are excluded.

- Part1: the sites in P_{i+k} with their *y* coordinates $\in [j_1, j_2]$.
- Part2: the sites in P_{i-k} with their *y* coordinates $\in [j_1, j_2]$.
- Part3: the Sites in S_i with their *y* coordinates $\in [j_1, j_2]$ and their *x* coordinates $\in [i-k, i+k]$.

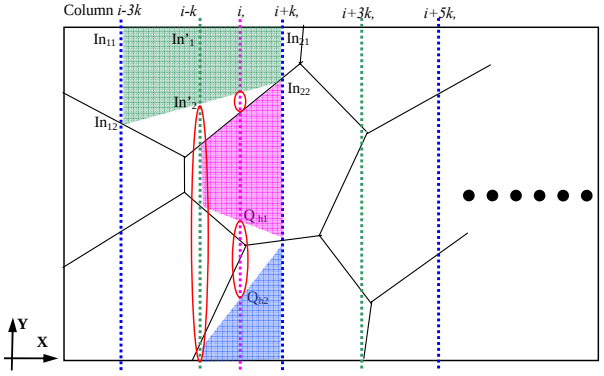

Fig. 2. Illustration to get the closest sites for the pixels in the coherence regions, as represented by colored quadrangles.

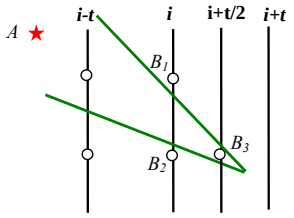

Fig.3. The special case that a voronoi cell intersect a column in between its two neighbouring pixels.

However, the above measures may miss a special case. As illustrated in Fig.3, the Voronoi cell for site *A* intersects column *i* in between its two neighbouring pixels B_1 and B_2, due to the fact that pixels are discrete. In this case, when we treat column *i* by the information of columns *i-t* and *i+t*, site *A* will not exist in the intersection set of P_{i-t} and P_{i+t}. If pixels B_1 and B_2 are colored by coherence information of columns *i-t* and *i+t*, the segment for the Voronoi cell of site *A* to intersect column *i* will not be handled. This has no problem to color column *i*, but may causes errors in the followed processing, e.g., coloring pixel B_3 with another site instead of site *A* when column *i+t/2* is treated by the information of columns *i* and *i+t*, and S_i^b is produced by

the measures in the above paragraph. Considering this, we put forward to produce a new S'_i instead of S_i to color column i. S'_i is the union of P_{i-t}, P_{i+t} and $S_i^{b1} \cup S_i^{b2} \cup \ldots$, where $b1$, $b2$, … refer to the uncolored blocks in column i. By this, site A exists in P_i and pixel B_3 can be colored correctly. When computing P_i from S'_i, it is clear that the sites of $P_{i-t} \cap P_{i+t}$, must be in P_i and so not necessary to be checked, by which much computation can be further saved.

3.2 Fast Getting the Closest Sites

After P_i is computed, it is used to find the closest site for each pixel in column i. As shown in Fig.1, when the sites of P_i are ordered by their y coordinates, the intersections between column i and the perpendicular bisectors are also ordered in accordance with their related sites. Thus, we can use the intersections to fast get the closest sites for pixels. For example, for pixels between intersections In_j and In_{j-1}, they must share the closest site $S_{i,j-1}$. In this way, many distance computations and comparisons can be saved, in comparison with the corresponding measures in the DR algorithm [8].

4. Parallelization Implementation

In the DR algorithm [8], dimensions are treated one by one. In treating a dimension, the rows or the columns are computed respectively. Thus, with every row handled by a thread, the rows can be computed in parallel. This is also true for the columns. In this way, parallelization implementation of this algorithm can be realized.

Similarly, our progressive algorithm can be parallelized too. It is in fact to separate columns to handle in groups, and the columns of a group can be computed in parallel, though the groups selected with different intervals have to be treated sequentially.

Here, there is a balancing problem of using parallelism or coherence for acceleration. If the initial interval is too big, the level of parallelism is reduced as fewer columns are computed in parallel, and the coherence between nearby columns is also not high. On the contrary, if it is too small, more columns can be computed in parallel, but the efficiency of using coherence would be reduced. Thus, the initial interval should not be too big or too small, which should be related to the size of corresponding Voronoi cells.

To efficiently use the coherence of differently sized Voronoi cells, our scheme is to reduce the intervals progressively to select columns to treat. In principle, more columns between two computed columns can be taken to treat at a same time with intervals reduced more quickly in coherent regions. But this is not beneficial to the smaller coherent regions, which can be derived by similar reasoning in the above paragraph. Thus, we choose to reduce the intervals half by half in our current implementation. As for the optimal speed to reduce intervals, it is dependent on the sizes of the Voronoi cells, which would be further investigated in the future.

By the above discussion, the level of parallelism of ours is not high as that of the BP algorithm [8], even lower than that of the parallelization implementation of the DR algorithm. Due to this, ours has a lower requirement on the number of processors, and so helpful to treat large images on ordinary computers. This is because ordinary computers may fail in providing enough threads for full parallel computation in treating large images. As a result, ours can be more helpful for high precise applications, because in this case grids are always large and more coherences may occur.

5. Results and Discussion

We implemented our algorithm and the DR algorithm [8] using MS Visual Studio 2010, and downloaded the programs of the PB algorithm [1], and then made tests to compare them, where the parameters of the PB algorithm were assigned optimized values as suggested in [1]. To test the efficiecny of using coherence and treating high precise applications, we always produced a set of randomly sampled sites and let the related images in various pixel resolutions. We first made tests on CPU to compare our algorithm and the DR algorithm on their sequential executions and parallel executions respectively, then compared our algorithm, the DR algorithm and the PB algorithm on their parallel executions with GPU.

On the CPU. The notebook computer for our tests is installed with an Intel Core I7 CPU, 4G RAM and the operation system MS Windows 7. The CPU has 4 cores and can provide 4 threads for parallel computation.

The statistics data for their sequential executions are listed in Tables 1 & 2 for 5000 and 10000 sites respectively. Clearly, our algorithm can be much faster than the DR algorithm by 49.2%~181.9%. By checking the acceleration ratios along each column, it is known that we can obtain more acceleration with pixel resolutions increasing. Generally, with the initial intervals being bigger, more acceleration can be achieved, and especially evident for the bigger images.

The statistics data for their parallel executions are listed in Tables 3 & 4. As our algorithm has a lower level of parallelism than the DR algorithm, the obtained acceleration ratios are not higher than those for their sequential executions. But ours can still run faster than the DR algorithm, and with the images and initial intervals being bigger, acceleration ratios are higher and higher, and even up to 137.1%.

By the above discussions, it is clear that our algorithm can efficiently use coherence to speed up EDT computation, very helpful to high precise applications.

Table 1. Statistics data for sequential executions with 5000 sites (Time: ms)

Methods	Ours												DR
Initial interval	2		4		8		16		32		64		
Size: n	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time
512	63	49.2	63	49.2	62	51.6	62	51.6	62	51.6	62	51.6	94
1024	234	53.4	203	76.9	203	76.8	202	77.7	202	77.7	202	77.7	359
2048	764	73. 6	640	107.2	593	123.6	593	123.6	593	123.6	593	123.6	1326
4096	2449	76.6	1986	117.8	1763	145.4	1690	156.0	1690	156.0	1654	161.6	4326
8192	7207	76.2	5819	118.2	5164	145.9	4914	158.4	4852	161.7	4505	181.9	12698

Acce.#: the acceleration ratio computed as $(a - b)/b*100\%$, where a and b refer to the time cost by DR and ours.

Table 2. Statistics data for sequential executions with 10000 sites (Time: ms)

Methods	Ours												DR
Initial interval	2		4		8		16		32		64		
Size: n	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time
512	63	49.2	63	49.2	62	51.6	62	51.6	62	51.6	62	51.6	94
1024	234	53.6	219	64.4	219	64.4	218	65.1	218	65.1	218	65.1	360
2048	890	63.0	765	89.7	733	98.0	733	98.0	733	98.0	733	98.0	1451
4096	3089	75.2	2543	112.8	2293	136.0	2184	148.0	2183	148.0	2183	147.9	5411
8192	9953	76.2	7928	121.1	6911	153.7	6536	168.2	6428	172.7	6396	174.1	17532

Table 3. Statistics data for parallel executions with 5000 sites (Time: ms)

Methods	Ours												DR
Initial interval	2		4		8		16		32		64		
Size: n	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time
512	47	0.0	47	0.0	46	2.2	46	2.2	46	2.2	46	2.2	47
1024	109	29.4	109	29.4	94	50.0	94	50.0	93	51.6	93	51.6	141
2048	281	55.5	249	75.5	249	75.5	234	86.8	234	86.8	234	86.8	437
4096	790	59.9	671	88.2	656	92.5	639	97.7	640	97.3	640	97.3	1263
8192	2189	59.6	1857	88.2	1701	105.4	1654	111.3	1638	113.3	1638	113.3	3494

Table 4. Statistics data for parallel executions with 10000 sites (Time: ms)

Methods	Ours												DR
Initial interval	2		4		8		16		32		64		
Size: n	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time	Acce.#	time
512	47	0.0	47	0.0	47	0.0	47	0.0	46	2.2	46	2.2	47
1024	109	33.0	94	54.3	94	54.3	93	55.9	93	55.9	93	55.9	145
2048	312	47.8	281	64.1	265	74.0	250	84.4	250	84.4	250	84.4	461
4096	936	65.0	811	90.4	780	98.0	734	110.4	733	110.6	733	110.6	1544
8192	2823	66.4	2309	103.4	2075	126.3	2014	133.2	1981	137.0	1981	137.1	4696

On the GPU. The GPU is famous for its high performance on numerical computation by a SIMD manner,

but not beneficial to logic computation, due to its lower efficiency on branching. Since our algorithm needs a lot of logic operations to take advantage of coherences for acceleration, it is not very suitable for execution on the GPU. However, it can still use parallel computation for acceleration. We made tests to compare our algorithm, the DR algorithm and the PB algorithm on a personal computer installed with an Intel Core2 CPU, 2G RAM, and a NVIDIA GeForce 8600 GTS GPU with a buffer of 256M. Its operation system is MS Windows XP. As ours is not very suitable for the architecture of the GPU, its performance is lower than others when our initial intervals are bigger than 2. When the initial interval is 2, our algorithm can save many logic computations and achieve its highest level of parallelism. By this, ours may perform better than others at sometimes, as shown in Table 5. From the statistics data in Table 5, it is known that the DR algorithm can be always faster than the PB algorithm when the processors are fewer and the images are not very big. As for our algorithm, when the processors are fewer, it can be faster than the others in a lot of cases, especially when the images are bigger, as marked by red and blue stars in the table. This is because in these cases more coherences can be used, and threads are not too many. When there are more processors, ours can win when the treated images have fewer sites. By checking the data of the last two columns in the table, it is clear that with the images being bigger, our algorithm can gradually catch up with the others and even exceed them, as marked by stars. This means our algorithm is very suitable for high precise applications, even with the GPU.

Table 5. Statistics data for tests on the GPU with the initial interval of ours being 2 (Time: μs)

Processors	Size:n	Num. of sites	PB	DR	Ours	Acce.[1]	Acce.[2]
16	512	200	2387.712	1860.448	1936.224	* 23.32	-3.91
		800	2903.744	2594.878	2791.036	* 4.04	-7.03
		3000	3424.836	3203.074	3615.52	-5.27	-11.41
		5000	3617.7	3370.404	3885.09	-6.88	-13.25
		10000	3873.8	3556.26	4269.796	-9.27	-16.71
	1024	200	6682.884	5610.148	4883.874	* 36.84	* 14.87
		800	8220.008	7775.576	6283.164	* 30.83	* 23.75
		3000	10218.024	10988.09	8624.512	* 18.48	* 27.41
		5000	10661.4	11967.61	9609.278	* 10.95	* 24.54
		10000	11150.28	13348.41	11182.622	-0.29	* 19.37
	2048	200	24015.262	16125.99	14876.66	* 61.43	* 8.40
		800	27632.862	20712.74	18226.08	* 51.61	* 13.64
		3000	34923.018	32253.25	27138.76	* 28.68	* 18.85
		5000	37526.788	37104.52	31663.2	* 18.52	* 17.18
		10000	39756.452	42954.71	37751.508	* 5.31	* 13.78
32	512	200	1347.04	1678.336	1787.072	-24.62	-6.08
		800	1630.08	2347.127	2631.036	-38.04	-10.79
		3000	1870.08	2889.148	3420.158	-45.32	-15.53
		5000	1957.728	3030.656	3672.864	-46.70	-17.49
		10000	2084.8	3207.356	4860.106	-57.10	-34.01
	1024	200	3690.24	3222.236	3277.856	* 12.58	-1.70
		800	4532.636	4316.728	4454.016	* 1.77	-3.08
		3000	5623.88	5890.724	6240.348	-9.88	-5.60
		5000	5886.744	6384.392	6888.802	-14.55	-7.32
		10000	6157.026	7055.39	7885.798	-21.92	-10.53
	2048	200	12683.288	10367.03	9141.24	* 38.75	* 13.41
		800	14491.928	13543.72	11278.28	* 28.49	* 20.09
		3000	18206.43	22260.11	17650.37	* 3.15	* 26.12
		5000	19423.42	26790.53	21367.22	-9.10	* 25.38
		10000	20671.284	31164.89	25149.31	-17.81	* 23.92

Acce.[1]: the acceleration ratio computed as $(a - c)/ c*100\%$, where a and c refer to the time cost by PB and ours.
Acce.[2]: the acceleration ratio computed as $(b - c)/ c*100\%$, where b and c refer to the time cost by DR and ours.

6. Conclusions

This paper presents a new algorithm for exact EDT computation by progressively selecting columns to treat, based on the strategy of the DR algorithm to handle the dimensions one by one. In this way, many coherences between neighbouring columns can be used to save a lot of computation, and results show that ours can be much faster than the DR algorithm on the CPU, no matter whether they are executed sequentially or in parallel. When compared with the PB algorithm on the GPU, the fastest algorithm for exact EDT to date, ours

is inferior to it when the density of sites in the image is high, because ours has a lower parallelism level due to the progressive manner. However, in high precise applications, the density of sites would be reduced and so more coherences can be used for acceleration. As a result, ours may be faster than the PB algorithm on the GPU in these cases. Another advantage is that ours has not many parameters to tune as the PB algorithm for good performance, and so it is convenient to use in applications. Our algorithm can be also extended to treat spherical distance tranform, as done in Ref. [2] to extend the the DR algorithm [8] to treat spherical distance transform.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Program No.: 60773026, 60873182, 60833007). The programs for the banding parallel algorithm [1] are downloaded from *http://www.comp.nus.edu.sg/~tants/pba.html.*

References

[1] T. Cao, K. Tang, A. Mohamed, T. Tan, Parallel banding algorithm to compute exact distance transform with the GPU, in *Proc. of ACM SIGGRAPH symposium on Interactive 3D Graphics and Games (I3d)* (2010), pp.83~90.

[2] W. Chen, R. Wang, M. Pan, W. Hua, GPU-based fast spherical distance transform, *Chinese Journal of Computers* 34, 3 (2011), pp. 499~507.

[3] N. Cuntz, A. Kolb, Fast hierarchical 3D distance tranforms on the GPU, in *Proc. of EUROGRAPHICS* (2007), short paper.

[4] R. Fabbri, L. Costa, J. Torelli, O. Bruno, 2D Euclidean distance transform algorithms: A comparative survey, *ACM Computing Survey* 40, 1 (2008), pp. 1~44.

[5] M. Jones, J. Baerentzen, M. Sramek, 3D distance fields: A survey of techniques and applications, *IEEE Transaction on Visualization and Computer Graphics* 12, 4 (2006), pp. 581~599.

[6] T. Hayashi, , K. Nakano, S. Olariu, Optimal parallel algorithms for finding proximate points, with applications, *IEEE Transaction on Parallel and Distributed Systems* 9, 12(1998), pp. 1153~1166.

[7] Y. Lee, S. Horng, J. Seitzer, Parallel computation of the Euclidean distance transform on a three-dimensional image array, *IEEE Transaction on Parallel and Distributed Systems* 14, 3 (2003), pp. 203~212.

[8] JR. Maurer, R. Qi, V. Raghavan, A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 2 (2003), pp. 265~270.

[9] J. Schneider, M. Kraus, R. Westermann, GPU-based real-time discrete Euclidean distance transforms with precise error bounds, in *Proc. of International Conference on Computer Vision Theory and Applications (VISAPP)* (2009), pp. 435–442.

[10] Y. Wang, S. Horng, Y. Lee, P. Lee, Optimal parallel algorithms for the 3D Euclidean distance transform on the CRCW and EREW PRAM models, in *Proc. of the 19th Workshop on Comb. Math. and Comp. Theory* (2001), pp. 209~218.