Computing Distance Maps Efficiently Using An Optical Bus

Yi Pan[1], Yamin Li[2], Jie Li[3], Keqin Li[4], and Si-Qing Zheng[5]

[1] University of Dayton, Dayton, OH 45469-2160, USA
[2] The University of Aizu Aizu-Wakamatsu 965-80 Japan
[3] University of Tsukuba Tsukuba Science City, Ibaraki 305-8573, Japan
[4] State University of New York New Paltz, New York 12561-2499, USA
[5] University of Texas at Dallas Richardson, TX 75083-0688, USA

Abstract. This paper discusses an algorithm for finding a distance map for an image efficiently using an optical bus. The computational model considered is the arrays with a reconfigurable pipelined bus system (LARPBS), which is introduced recently based on current electronic and optical technologies. It is shown that the problem for an $n \times n$ image can be implemented in $O(\log n \log \log n)$ time deterministically or in $O(\log n)$ time with high probability on an LARPBS with n^2 processors. We also show that the problem can be solved in $O(\log \log n)$ time deterministically or in $O(1)$ time with high probability on an LARPBS with n^3 processors. The algorithm compares favorably to the best known parallel algorithms for the same problem in the literature.

1 Introduction

In the areas of image processing and computer vision, it is usually required to extract information about the shape and the position of the foreground pixels relative to each other for a digital image. There are many computational techniques which can be used for such an information retrieval. One technique to accomplish this task is the distance map (or distance transform). The distance transform is to convert a binary image to another image, such that each pixel has a value to represent the distance between it and its nearest black pixel. The new image is called the distance map of the old image. Note that this problem is different from the problem of finding nearest neighbors, where the search for nearest black neighbors is only performed for black pixels, not for all pixels in an image. Hence, finding the distance map for an image is more difficult.

Consider a black and white $n \times n$ binary image: i.e., a 2-dimensional array where $a_{ij} = 0$ or 1, for $i, j = 0, 1, 2, ..., n - 1$. The index i stands for the row and the index j for column. Pixel (0,0) is the upper left point in the image. There are many different distance transforms available using different distance metrics. The Euclidean distance transform (EDT) is to find for each point (i, j) its Euclidean distance from the set of all black pixels $B = \{(i, j) : a_{ij} = 1\}$. In other words, we compute the array

$$d_{ij} = \min_{(x,y) \in B} \{((i - x)^2 + (j - y)^2)^{1/2}\}, \quad 0 <= i, j <= n - 1.$$

The EDT is a basic operation in computer vision, pattern recognition, and robotics. For instance, if the black pixels represent obstacles, then d_{ij} tells us how far the point (i, j) is from these obstacles. This information is useful when one tries to move a robot in the free space (white pixels of the image) and to keep it away from the obstacles (black pixels).

Finding the distance transform with respect to the Euclidean metric is rather time consuming. Yamada [11] presented an EDT algorithm that runs in $O(n)$ time using n^2 processors on an 8-neighbor connected mesh. Kolountzakis and Kutulakos presented an $O(n^2 \log n)$ sequential algorithm for the EDT problem [5]. They also showed that the time complexity of their algorithm is $O((n^2 \log n)/r)$ on an EREW PRAM model with $r <= n$ processors. Chen and Chuang improved the algorithm in [2] by reducing the time complexity to $O(n^2/r + n \log r)$ on an EREW PRAM model. Bossomaier *et al.* studied the speedup and efficiency of Yamada's algorithm on the CM-2 Connection machine, and introduced new techniques to improve its performance [1]. More recently, Chen and Chuang proposed an algorithm for computing the EDT on a mesh-connected SIMD computer [3]. For an $n \times n$ image, their algorithm runs in $O(n)$ time on a 2-dimensional $n \times n$ torus-connected processor array. In this paper, we propose more efficient parallel algorithms for computing the Euclidean distance transform on the LARPBS model. For the same problem, one of our algorithm runs in $O(\log n \log \log n)$ time or in $O(\log n)$ time with high probability on an LARPBS with n^2 processors. We also show that the time complexity can be further reduced if n^3 processors are used. The results presented in this paper improve on all previous results in the literature.

The computational model used is reviewed in the next section. Our algorithm for computing the distance map is described in section 3. Section 4 discuss the possibility of further reducing the time complexity of the algorithm by using more processors. We conclude our paper in section 5.

2   The LARPBS Model

Massively parallel processing using optical interconnections poses new challenges. In general, it does not worth the effort to improve performance by simply replacing the metal interconnections of a parallel system with optical interconnections in a one-to-one fashion. This is due to the fact that in doing so the inherent large bandwidth and high parallelism of optical interconnects are underutilized. The characteristics of optical interconnects have significant implications. New system configurations need to be designed due to the changes in architectural freedom and constraints when optical interconnects are incorporated. To fully utilize the bandwidth offered by the optical interconnections, scheduling and data communication schemes based on new resource metrics need to be investigated. Algorithms for a wide variety of applications need to be developed under novel computation models that are based on optical interconnections. Computations under these new models, which can be drastically different from existing theoretical PRAMs and parallel systems with electrical interconnections, may

require new algorithm design techniques and performance measures.

A linear array with a reconfigurable pipelined bus system (LARPBS) consists of N processors P_1, P_2, ..., P_N connected by an optical bus. In addition to the tremendous communication capabilities, a LARPBS can also be partitioned into $k \geq 2$ independent subarrays $LARPBS_1$, $LARPBS_2$, ..., $LARPBS_k$, such that $LARPBS_j$ contains processors $P_{i_{j-1}+1}$, $P_{i_{j-1}+2}$, ..., P_{i_j}, where $0 = i_0 < i_1 < i_2 \cdots < i_k = N$. The subarrays can operate as regular linear arrays with pipelined optical bus systems, and all subarrays can be used independently for different computations without interference (see ([7, 8]) for an elaborated exposition, and ([9]) for similar reconfigurable pipelined optical bus architectures).

For ease of algorithm development and specification, a number of basic communication, data movement, and global operations on the LARPBS model implemented using the coincident pulse processor addressing technique ([4, 6]) have been developed ([7, 8, 9, 10]). They include one-to-one routing, multicast, broadcast, segmented broadcast, compression, minimum finding, binary sum and image transpose. All of them except minimum finding can be performed in a constant number of bus cycles [7, 10]. Minimum finding can be done either in $O(\log \log N)$ time deterministically or in $O(1)$ time with high probability [8]. These powerful primitives that support massive parallel communications, plus the reconfigurability of the LARPBS model, make the LARPBS very attractive in solving problems that are both computation and communication intensive, such as image processing. Our algorithms are developed using these operations as building blocks. For further implementation details on these operations, the reader is referred to [7, 8, 10].

3 Algorithm Using n^2 Processors

In our algorithm description, the following terms are used. We divide the image into two parts for each pixel. Pixels to the left of pixel (i, j) are referred to pixels in the left plane of pixel (i, j). Similarly, Pixels to the right of pixel (i, j) are referred to pixels in the right plane of pixel (i, j).

If we denote the nearest black pixel to pixel (i, j) as $M(i, j)$, then the distance from (i, j) to $M(i, j)$ is the EDT at (i, j). We can reduce the search time for $M(i, j)$ by dividing the image into sections and search the sections separately.

In order to compute (1), it is sufficient to give an algorithm for the computation of

$$l_{ij} = \min_{(x,y) \in B} \{((i - x)^2 + (j - y)^2)^{1/2}\}, 0 \leq i, j \leq n - 1.$$

Here, l_{ij} is the distance of the point (i, j) from the part of B that is to the left of (i, j). Computing the distance from the right and from the left and then taking the minimum of the two gives d_{ij}. So only the computation of l_{ij} is described here.

For a given j, we define W_j as the set of all black pixels to the left of column j. Denote the pixel in W_j nearest to (i, j) as $W(i, j)$. The following lemma is due to Kolountzakis and Kutulakos [5]:

Lemma 1 *Let $A = (a, j)$ and $B(b, j)$, $b < a$, be two pixels on the same column with A above B. Let $W(a, j) = (x, y)$ and $W(b, j) = (z, w)$. Then $z \leq x$; namely, $W(a, j)$ is above or on the same row as $W(b, j)$. Similarly, for two pixels on the same row, if $W(i, a) = (s, t)$ and $W(i, b) = (u, v)$ with $b < a$, then $v \leq t$; namely, $W(i, a)$ is left to or on the same column as $W(i, b)$.*

The search spaces for the nearest black pixels can be reduced by applying this lemma. Suppose it is known that $W(i, j) = (z, w)$. By lemma 1, we know $W(i, k)$ must be between columns 0 and $w - 1$ if $k < j$, and between columns w and $n - 1$ if $k > j$. Thus, we do not need to search all the columns.

Now, we describe the algorithm using the basic data movement operations described in the preceding sections. The basic idea of the algorithm is: 1) to find the nearest black pixel in each column for all pixels (including white pixels). 2) to search the nearest black pixel of a pixel in its left region followed by finding the nearest black pixel of a pixel in its right region. 3) we obtain the nearest black pixel for a pixel in the whole image by selecting the nearest black pixel in both regions.

Assume that initially each pixel a_{ij} is stored in the LARPBS with n^2 processors in row-major order. That is, $a(i * n + j) = a_{ij}$, for $0 \leq i, j < n$. Each processor has several local variables. In the following discussion, we use $D(i * n + j)$ and $D(i, j)$ interchangeably, both representing a variable D in processor $k = i * n + j$.

The algorithm consists of the following steps:

Step 1: In this step, we calculate for pixel (including white pixels) at (i, j) the row index of the closest black pixel in the j-th column on or above the i-th row of the image. Initially, if $a_{ij} = 1$, set $UPNEAREST(i * n + j) = i$. Otherwise, set $UPNEAREST(i * n + j) = \infty$. In order to do this, the image is first transposed into column-major order, and the array is segmented into n subsystems. Each subsystem contains a column of the image. Then, a right segmented broadcast is performed using $UPNEAREST(i * n + j)$ as the local data value and the pixel value a_{ij} as the local logical value. The results are stored in $UPNEAREST(i * n + j)$.

Step 2: Calculate for pixel at (i, j) the row index of the closest black pixel in the j-th column on or below the i-th row of the image. Initially, if $a_{ij} = 1$, $DOWNNEAREST(i * n + j) = i$. Otherwise, $DOWNNEAREST(i * n + j) = \infty$. At the beginning of this step, the image is already in column-major order, and the processor array is already segmented into n subsystems. A left segmented broadcast is performed using that $DOWNNEAREST(i * n + j)$ as the local data value and a_{ij} the local logical value. The final results are stored in $DOWNNEAREST(i * n + j)$.

Step 3: Calculate the EDT values in their left regions for all the points in the image. In order to do this, the image is first transposed back into row-major order, and the array is segmented into n subsystems. Denote these subsystems as $ARR(i)$, $0 \leq i \leq n - 1$. Each subsystem $ARR(i)$ contains the i-th row of the image. We use subarray $ARR(i)$ to calculates EDT values in their left regions of row i for $0 \leq i \leq n - 1$. The computation is carried out through bisectioning each row until all EDT values in

their left regions are calculated in a given row. Since the EDT values in their left regions for all rows are calculated similarly, in the following discussion, we concentrate on row i. First, we use subarray $ARR(i)$ to calculate the EDT value in the left region for point $(i, n/2,)$. This is done as follows. Assume that $DOWNNEAREST(i,j) = i_1$, $UPNEAREST(i,j) = i_2$, $PE(i,j)$ first calculates $DOWNDIST(i,j) = ((i_1 - i)^2 + (j - n/2))^{1/2}$ and $UPDIST(i,j) = ((i_2 - i)^2 + (j - n/2))^{1/2}$. These are the distances of the nearest black pixels in its column and the point $(i, n/2)$, whose EDT value is being computed. Then, $PE(i,j)$ compares $DOWNDIST(i,j)$ and $UPDIST(i,j)$, and put the smaller in $DIST(i,j)$. Finally, use the linear subarray $ARR(i, *)$ of n processors to find the minimum of the n distances $DIST(i,j)$, $0 \leq j \leq n - 1$. Clearly, this can be done in $O(\log \log n)$ time deterministically.

Suppose that $W(i, n/2) = (z, w)$. Column w divides row i into two subsections. Use lemma 1, to find the EDT value in the left reqion for another point, we need only to search the minimum values among the $DIST(i,j)$ values in its corresponding subsection depending on its position. Hence, we divide the subarray $ARR(i, *)$ into 2 subarrays using w as the partition column, and use the first subarray to calculate the EDT value in the left reqion at point $(i, n/4)$ and the second subarray to calculate the EDT value in the left reqion at point $(i, 3n/4)$. Notice that the sizes of the two subarrays may be different in a row and different rows have different partitions. The reconfiguration is done via disconnecting the bus at $PE(i, w)$ for $0 \leq i \leq n - 1$. This process is carried out for all rows in the system. To calculate the EDT value in the left reqion at point $(i, n/4)$, we only need to find the minimum of $DIST(i,j)$, for $0 \leq j < w$, in the first subarray. This is done as follows. Assume that $DOWNNEAREST(i,j) = i_1$, and $UPNEAREST(i,j) = i_2$. $PE(i,j)$ in the left subarray first calculates $DOWNDIST(i,j) = ((i_1 - i)^2 + (j - n/4))^{1/2}$ and $UPDIST(i,j) = ((i_2 - i)^2 + (j - n/4))^{1/2}$. Then, $PE(i,j)$ compares $DOWNDIST(i,j)$ and $UPDIST(i,j)$, and put the smaller in $DIST(i,j)$. Finally, use the first subarray in row i to find the minimum of the distances $DIST(i,j)$, $0 \leq j < w$. Since the size of the subarray is at most n, this can be done in $O(\log \log n)$ time using the deterministic minimum finding algorithm [8]. Similarly, to calculate the EDT value in the left reqion at point $(i, 3n/4)$ in the second subarray also takes at most $O(\log \log n)$ steps. Now, we can further partition a subsection into two parts and we reconfigure the subarray $ARR(i)$ into four subarrays according to the values $W(i, 3n/4)$, $W(i, n/2)$, and $W(n/4)$. We can find the EDT values in their left regions of the middle point in each of the four subsections using a similar procedure. Again, we only need to search each subarray to perform the minimum finding of the $DIST$ values base on lemma 1. Hence, at most $O(\log \log n)$ time is needed in this iteration since the minimum findings can be performed concurrently in these four subarrays. This process continues until the size of each subsection is 1.

Step 4: Once the distance of the point (i, j) the left region of (i, j) is computed,

a method similar to the one used in step 3 can be employed to calculate the distance of the point (i, j) in the right region.

Step 5: Comparing the distances to a pixel in its left region and right region and then taking the minimum of the two gives d_{ij}. This step involves only a local operation.

At the end of the algorithm, the EDT result is in $EDT(i, j)$ for $0 \leq i, j \leq n - 1$. The time can be calculated as follows. Step 1 calculates the indices of the closest black pixel in the k-th column on or above the i-th row of the image. Similarly, step 2 finds the index of the closest black pixel in the k-th column on or below the i-th row of the image. Since both a transpose operation and a segmented broadcast operation require $O(1)$ time, step 1 takes $O(1)$ time. Similarly, step 2 uses $O(1)$ time. In step 3, several operations such as transpose operation, segmented broadcast, and minimum finding, and several local operations are employed. All operations use $O(1)$ time, except the minimum finding operation which uses $O(\log \log n)$ time. Clearly, we need $\log n$ iterations to find all the EDT valuesv in row i. Since all the n rows are calculated concurrently and $\log n$ iterations are needed, the time taken in step 3 is $O(\log n \log \log n)$. Therefore, the total time used in the algorithm is $O(\log n \log \log n)$.

As pointed before, minimum finding can also be performed in $O(1)$ with high probability using a randomised algorithm [8]. Since each row is partitioned into many segments in each iteration, the probability of using $O(1)$ time in step 3 depends on the number of segments and the probability of the minimum finding algorithm used in each segment. In our algorithm, if the size of a subarray is smaller than 256, we can simply use the deterministic $O(\log \log n)$ time minimum-finding algorithm given above. For $n = 256$, $\log_2 \log_2 256 = 3$, and $O(\log \log 256) = O(1)$. If the size of a subarray is larger than 256, the above randomised minimum finding algorithm is used. Clearly, during the first iteration, there is only one segment. During the second iteration, we have two segments. In general, during the i-th iteration, we have 2^i segments, where $1 \leq i \leq \log n$. The total number of minimum findings in each row is less than n since each minimum finding corresponds to the calculation of the EDT value for one pixel. Hence, the total number of minimum findings in step 3 is $X = n^2/256$. Let P_i be the probability of $O(1)$ running time for a particular minimum finding i during the algorithm. The probability of the algorithm running in $O(1)$ time is the product of all X P_i's. The total number of randomized minimum finding operations is no larger than $X = n^2/256$, and each has a probability of at least $1 - 256^{-\alpha}$ (α is a function of C used in the minimum finding algorithm) of requiring $O(1)$ time. Thus, the probability of all the minimum finding operations running in $O(1)$ time during step 3 is $(1 - 256^{-\alpha})^{n^2/256}$. For images with practical sizes, this probability is very close to 1. To see this, consider $\alpha = 4$ and an LARPBS of 1,000,000 processors. The probability of all the minimum finding operations running in $O(1)$ time during step 3 is at least 0.9999991. Hence, step 3 uses $O(\log n)$ time with high probability. Similarly, step 4 requires $O(\log n)$ time with high probability. Step 5 performs some local comparisons and hence uses $O(1)$ time. Since each step in the algorithm requires either $O(1)$ time or

$O(\log n)$ time, the total time of the algorithm is $O(\log n)$.

In the above analysis for steps 3 and 4, we require α to be no less than 4. Fortunately, this can be easily achieved because a Monte Carlo algorithm which runs in $O(T(N))$ time with probability of success $1 - O(1/N^\alpha)$ for some constant $\alpha > 0$ can be turned into a Monte Carlo algorithm which runs in $O(T(N))$ time with probability of success $1 - O(1/N^\beta)$ for any large constant $\beta > 0$ by running the algorithm for $\lceil \beta/\alpha \rceil$ consecutive times and choosing one that succeeds without increasing the time complexity. Summarizing the above discussions, we obtain the following results:

Theorem 1 *The distance map problem defined on an $n \times n$ images can be solved using an LARPBS of n^2 processors in $O(\log n \log \log n)$ time deterministically or in $O(\log n)$ time with high probability for any practical size of n.*

4 Algorithm Using n^3 Processors

We can further reduce the time in the above algorithm through using more processors. In this section, we describe an algorithm which works on an LARPBS with n^3 processors. The n^3 algorithm is similar to the n^2 algorithm described in the preceding section. The only difference is that we use n^2 to find the EDT values in a row instead of using n processors, thus reducing the time used.

Initially, the n^2 pixels are stored in the first n^2 processors. Actually, all the steps except steps 3 and 4 use the first n^2 processors only, and hence have the same steps as in the n^2 algorithm described in the preceding section.

Now we describe step 3 in the new algorithm in detail. Notice that all values such as $DIST$'s have been computed and stored in local processors. We divide the LARPBS into n subsystems with each having n^2 processors. Denote these subsystems as LARPBS-i, $0 \leq i \leq n - 1$. Distribute the n rows of pixels along with the computed values such as $DIST$'s in the previous steps to the first n processors of the n subsystems. Thus, each subsystem is responsible for a row of pixels. There are n EDT values to be computed in a row and each subsystem has n^2 processors. Hence, we can let n processors calculate an EDT value and all the EDT values can be computed concurrently. An EDT value can be computed using the deterministic minimum finding algorithm or the randomized minimum finding algorithm on the $DIST$ values computed in step 2. Obviously, this step involves only broadcast, multicast, array reconfiguration, and minimum finding operations. All these operations takes $O(1)$ time except the minimum finding algorithm. Using a similar analysis described previously, it is easily obtained that step 3 can be computed in $O(\log \log n)$ time deterministically or in $O(1)$ time with high probability. Hence, we have the following results:

Theorem 2 *The distance map problem defined on an $n \times n$ images can be solved using an LARPBS of n^3 processors in $O(\log \log n)$ time deterministically or in $O(1)$ time with high probability for any practical size of n.*

5 Conclusions

Due to the high bandwidth of an optical bus and several efficiently implemented data movement operations, the distance map problem is solved efficiently on the LARPBS model. It should be noted that algorithms on plain a mesh or a reconfigurable mesh cannot reach the time bounds described in this paper.

References

1. T. Bossomaier, N. Isidoro, and A. Loeff, "Data parallel computation of Euclidean distance transforms," Parallel Processing Letters, vol. 2, no. 4, pp. 331-339, 1992.
2. L. Chen and H. Y. H. Chuang, "A fast algorithm for Euclidean distance maps of a 2-D binary image," Information Processing Letters, vol. 51, pp. 25-29, 1994.
3. L. Chen and H. Y. H. Chuang, "An efficient algorithm for complete Euclidean distance transform on mesh-connected SIMD," Parallel Computing, vol. 21, pp. 841-852, 1995.
4. D. Chiarulli, R. Melhem, and S. Levitan, "Using Coincident Optical Pulses for Parallel Memory Addressing," IEEE Computer, vol. 20, no. 12, pp. 48-58, 1987.
5. M.N. Kolountzakis and K.N. Kutulakos, "Fast computation of Euclidean distance maps for binary images," Information Processing Letters, vol. 43, pp. 181-184, 1992.
6. R. Melhem, D. Chiarulli, and S. Levitan, "Space Multiplexing of Waveguides in Optically Interconnected Multiprocessor Systems," The Computer Journal, vol. 32, no. 4, pp. 362-369, 1989.
7. Yi Pan and Keqin Li, "Linear array with a reconfigurable pipelined bus system: concepts and applications," Special Issue on "Parallel and Distributed Processing" of Information Sciences, vol. 106, no. 3/4, pp. 237- 258, May 1998. (Also appeared in International conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale, CA, August 9-11, 1996, 1431-1442)
8. Y. Pan, K. Li, and S.Q. Zheng, "Fast nearest neighbor algorithms on a linear array with a reconfigurable pipelined bus system," Parallel Algorithms and Applications, vol. 13, pp. 1-25, 1998.
9. S. Rajasekaran and S. Sahni, "Sorting, selection and routing on the arrays with reconfigurable optical buses," IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 11, pp. 1123-1132, Nov. 1997.
10. J. L. Trahan, A. G. Bourgeois, Y. Pan, and R. Vaidyanathan, "Optimally scaling permutation routing on reconfigurable linear arrays with optical buses," Proc. of the Second Merged IEEE Symposium IPPS/SPDP '99, San Juan, Puerto Rico, pp. 233-237, April 12-16, 1999.
11. H. Yamada, "Complete Euclidean distance transformation by parallel operation," Proc. 7th International Conference on Pattern Recognition, pp. 69-71, 1984.

This article was processed using the LaTeX macro package with LLNCS style