Parallel Algorithms for the Distance Transformation

Hugo Embrechts * *and Dirk Roose*

Katholieke Universiteit Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract. The distance transformation (DT) is a basic operation in image analysis where it is used for object recognition. A DT converts a binary image consisting of foreground pixels and background pixels, into an image where all background pixels have a value equal to the distance to the nearest foreground pixel.

We present several approaches for the parallel calculation of the distance transform based on the "divide-and-conquer" principle. The algorithms and their performance on an iPSC®/2 are discussed for the city block (CB) distance that is an approximation for the Euclidean Distance.

1 Introduction

A DT converts a binary image consisting of foreground and background pixels, into an image where all background pixels have a value equal to the distance to the nearest foreground pixel.

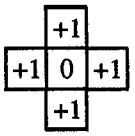
Computing the Euclidean distance from a pixel to a set of foreground pixels is essentially a global operation and therefore needs a complicated and time-consuming algorithm. However, reasonable approximations to the Euclidean distance measure exist that allow algorithms to consider only a small neighbourhood at a time. They are based on the idea that the global distances are approximated by propagating local distances, i.e. distances between neighbouring pixels. Two of the distance measures proposed in [1, 2] are the city block distance and the chamfer 3-4 distance. They are defined by the masks of Fig. 1. The DT applied to an image with one foreground pixel centered at the middle of the image is shown in Fig. 2. For the CB distance we present parallel algorithms.

The DT is a basic operation in image analysis where it is used for object recognition. It can be used for computing skeletons in a non-iterative way. Further applications are merging and segmentation, clustering and matching [1].

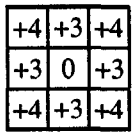
2 The Sequential Algorithm

The sequential algorithm is a known algorithm [1] consisting of two passes during which the image is traversed, once from top to bottom and from left to right, and the second time in reverse order. When a pixel is processed, its distance value (infinity if not yet determined) is compared to the distance value of a number of neighbours augmented by their relative distance and is replaced by the smallest resulting value. This causes the distance values to propagate from the object boundaries in the direction of the scan and yields, after the second pass, the correct DT-values.

* The following text presents research results of the Belgian Incentive Program "Information Technology" – Computer Science of the future, initiated by the Belgian State – Prime Minister's Service – Science Policy Office. The scientific responsibility is assumed by its authors.

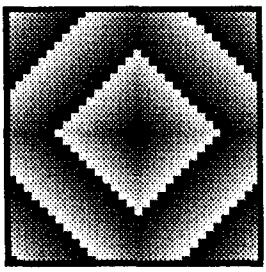


City Block distance

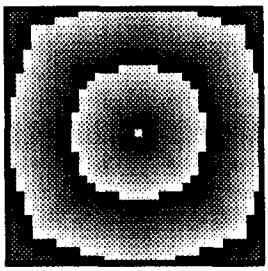


Chamfer 3-4 distance

Fig. 1. These masks show for the indicated distance measures the distance between the central pixel and the neighbouring pixels. The distance between two image points a and b is defined as the sum of the distances between neighbouring pixels in the path connecting a and b, that minimizes this sum.



City Block



Chamfer 3-4

Fig. 2. The DT of an image with one foreground pixel centered in the middle of the image for the City Block and Chamfer 3-4 distances. Growing distance is represented by a greytone repeatedly varying from black to white (to accentuate the contours of the DT).

3 Introduction to the Parallel Approach

Parallelism is introduced by the 'divide-and-conquer' principle. This means that the image is subdivided into as many subregions as there are processors available ; the operation to be parallelized, in our case the DT, is computed on each subregion separately and these local DTs have to be used to compute the global DT on the image. Let LDT (local DT) denote the DT applied to a subregion or, where indicated, a union of neighbouring subregions and let GDT (global DT) denote the DT applied to the whole image.

The algorithm consists of the next three steps :

I. On each subregion the LDT is computed *for the boundary pixels* of that subregion.
II. The GDT values *for the boundary pixels* are computed out of the LDT values.
III. On each subregion the GDT values for the internal pixels are determined out of the GDT values for the boundary pixels and the local image information. We call this part IDT (internal DT).

The first step could be done by executing the sequential DT algorithm on each subregion and retaining the boundary values. However, in [3] we present a shorter one pass algorithm which traverses each pixel at most once.

For step II we consider two possible solutions. In the first solution (*hierarchical algorithm*) we consider a sequence of gradually becoming coarser partitions p_l ($l = 1, 2, \ldots, L = \log_2 p$) of the image, with the finest partition p_1 being the chosen partition of the image containing as many subregions as there are processors available. Each of the other partitions p_l ($l > 1$) consists of subregions that are the union of two subregions of p_{l-1}. The coarsest partition p_L contains as only subregion the image itself. The *LDT on partition* p_l is defined as the result of the DT on each of the subregions of p_l separately. In this approach we calculate from the LDT on p_l *for the boundary pixels* of its subregions the corresponding values on p_{l+1} for $l = 1, 2, \ldots, L-1$. The values of the LDT on partition p_L are by definition the GDT values. Then the GDT values for the boundary pixels of the subregions of p_l are computed for decreasing l. This approach is similar to the hierarchical approach we used for component labelling [4].

These computations can be implemented in two ways. In the first approach (*agglomerated computation*), on a particular recursion level l each subregion of p_l is processed by one processor. This means that processors become idle on higher recursion levels. In an alternative implementation (*distributed computation*), pixel values of a subregion are not agglomerated into one processor, but are distributed in a way that each processor contains a part of the boundary of one subregion.

The second solution (*directional algorithm*) for step II consists of an inter-subregion propagation in successive directions. The feasibility of this approach, however, and the complexity of the resulting algorithm depend on the distance measure used.

The step III of the parallel algorithm is done by executing the sequential algorithm on each subregion starting from the original image and the GDT values obtained in step 2.

We refer to [3] for a full description and correctness proof of the algorithms.

4 Asymptotical Complexity

The calculation of the LDT-values of the boundary pixels of a subregion, as well as the IDT, is *local* and can be performed in an amount of time asymptotically proportional to the number of pixels of the image.

The calculation of GDT-values out of LDT-values for the border pixels of the subregions is *global* and consists of computation and communication. The latter can be divided into the initiation and the actual transfer of messages. A summary of the complexity figures for the global operations, derived in this section, is shown in table 1. We assume an image of $n \times n$ pixels and p processors.

	hierarchical alg.		direct. alg.
	agglom.	distrib.	
t_{start_up}	$O(\log p)$	$O(\log^2 p)$	$O(\log p)$
$t_{transfer}$	$O(n)$	$O(\frac{n}{\sqrt{p}})$	$O(\frac{n}{\sqrt{p}})$
t_{comp}	$O(n)$	$O(\frac{n}{\sqrt{p}})$	$O(\frac{n}{\sqrt{p}})$

Table 1. A summary of the complexity analysis of the global computations of the presented DT algorithms for the CB distance.

The Hierarchical Algorithm.

Agglomerated Computation. Since the number of messages sent on each recursion level is constant and since initiating a message takes constant time, the total start up time is proportional to the number of recursion levels $L = \log_2 p$.

The transfer time is proportional to the amount of data sent. The amount of data sent on recursion level l is proportional to the size of a subregion of p_l being

$$S_l = \mathrm{O}(\frac{n}{2^{(L-l)/2}}). \tag{1}$$

Therefore the total transfer time is $t_{transfer} = \mathrm{O}(\sum_{l=1}^{L} S_l) = \mathrm{O}(n)$. The computational complexity is also $\mathrm{O}(n)$ as the data are processed in linear time.

Distributed Computation. On recursion level l processors cooperate in groups of 2^l processors to compute the LDT on p_{l+1} on the borders of the subregions of p_{l+1}. If the CB distance measure is used, the operations to be done on recursion level l can be done in $\mathrm{O}(l)$ steps. In each of these steps an amount of data proportional to the boundary length of the subregions of p_l divided by the number of processors 2^l is transferred and processed :

$$D_l = \mathrm{O}(\frac{n}{\sqrt{p}} \frac{2^{l/2}}{2^l}). \tag{2}$$

The total start up time is therefore $t_{start\_up} = \mathrm{O}(\sum_{l=1}^{L} l) = \mathrm{O}(\log^2 p)$ and the total amount of execution and transfer time $t_{transfer} = t_{comp} = \mathrm{O}(\sum_{l=1}^{L} D_l l) = \mathrm{O}(\frac{n}{\sqrt{p}})$.

The Directional Algorithm. The directional algorithm consists of calculating a number of partial minima that can be done in $\mathrm{O}(\log p)$ communication steps requiring in total $\mathrm{O}(\frac{n}{\sqrt{p}})$ transfer and processing time. See [3].

5 Timing and Efficiency Results

We used as test images a number of realistic images and a few artificial images, among which the one of Fig. 2. The execution time of the sequential DT algorithm on one node of the iPSC/2 is proportional to the number of pixels of the image and is typically about 800 ms for a 256×256 image. For images of this size the LDT is typically 100 ms.

The parallel efficiency, as a function of the size of the image, is shown in Fig. 3 for a sample image. From the asymptotical complexity figures of section 4 we learn that for large image sizes the execution time of the global computations is negligible with respect to the the execution time of the IDT and the LDT parts of the algorithm. The ratio of the latter two mainly determines the parallel efficiency. For smaller images the LDT part gets more important with respect to the IDT part. The image size for which the two parts take an equal amount of time is typically 32 pixels for both distance measures. For smaller images also the global computations get more important.

A factor that influences the efficiency too, is the load imbalance of the algorithm. It occurs, when a part of the algorithm takes more time on one processor than on the others and the processors have to wait for one another. A measure for the load imbalance of a part of the algorithm is

$$I = \frac{t^{max} - t^{av}}{t^{av}} \tag{3}$$

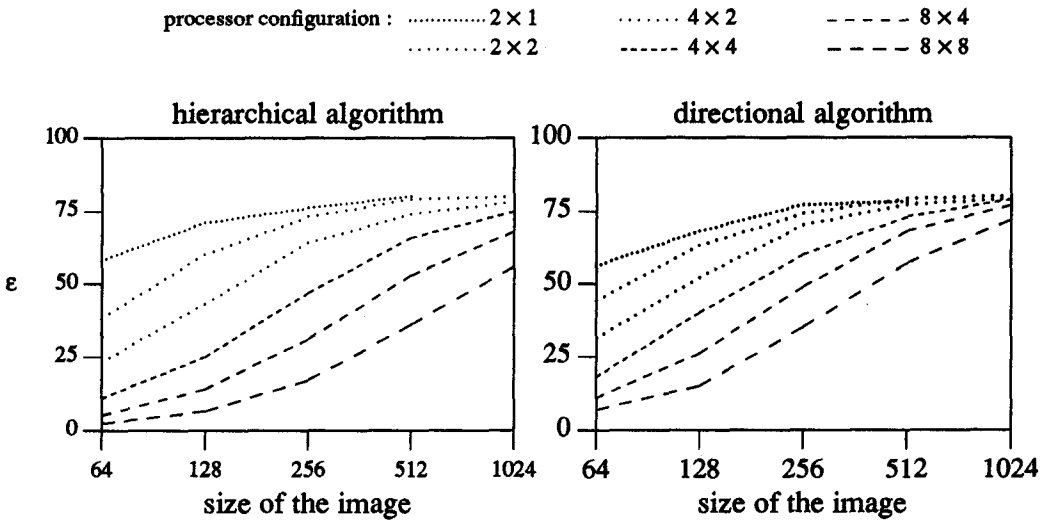


Fig. 3. The parallel efficiency, as a function of the image size, for the image of Fig. 2, when the hierarchical algorithm with agglomerated calculation or the directional algorithm is used.

with t^{max} and t^{av} the maximal and average execution times of the part of the algorithm under investigation. We can distinguish two sources of load imbalance.

A first source of load imbalance is caused by the data dependence of the LDT part of the algorithm. This is practically unavoidable, because for most images at least one subregion contains a considerable amount of background pixels and determines the execution time of the LDT part of the algorithm.

A second source of load imbalance is the data dependence of the IDT algorithm. This part of the load imbalance I grows with the number of subregions. However, we can find a hard upper limit for the possible load imbalance similar to the analysis in [4].

Acknowledgements

We wish to thank Oak Ridge National Laboratory for letting us use their iPSC/2 machine.

References

1. Borgefors, G.: Distance transformations in arbitrary dimensions. Computer Vision, Graphics and Image Processing **27(3)** (1984) 321-345
2. Borgefors, G.: Distance transformations in digital images. Computer Vision, Graphics and Image Processing **34(3)** (1986) 344-371
3. Embrechts, H., Roose, D.: Parallel algorithms for the distance transformation. Technical Report **TW151** (1991) Katholieke Universiteit Leuven
4. Embrechts, H., Roose, D., Wambacq, P.: Component labelling on an mimd multiprocessor. Computer Vision, Graphics and Image Processing : Image Understanding, to appear

This article was processed using the LaTeX macro package with ECCV92 style