Parallel 3D Exact Signed Euclidean Distance Transform

Release 1.00

Robert Staubs[1], Andriy Fedorov, Leonidas Linardakis, Benjamin Dunton and Nikos Chrisochoides

September 16, 2006

Department of Computer Science, The College of William and Mary
[1]rdstau-at-wm.edu

Abstract

The computation speed for distance transforms becomes important in a wide variety of image processing applications. Current ITK library filters do not see any benefit from a multithreading environment. We introduce a three-dimensional signed parallel implementation of the exact Euclidean distance transform algorithm developed by Maurer *et al.*[1] with a theoretical complexity of $O(n/p)$ for n voxels and p threads. Through this parallelization and efficient use of data structures we obtain approximately 3 times mean speedup on standard tests on a 4-processor machine compared with the current ITK exact Euclidean distance transform filter[5].

1 Introduction

The Euclidean distance transform (EDT) of an N-dimensional binary image I is an N-dimensional image I_{EDT} such that for all indices i, the element $e_{EDT,i} \in I_{EDT}$ is the Euclidean distance from the element $e_{I,i}$ to the nearest foreground element in I. If we assume the typical convention of negative distances for the interior of a feature, we can further define the signed EDT (SEDT) as I_{SEDT} such that $\forall i, e_{SEDT,i} \in I_{SEDT} the |e_{SEDT,i}|$ is the Euclidean distance from $e_{I,i}$ to the nearest surface foreground element and $e_{SEDT,i}$ is negative if $e_{I,i}$ is internal to a feature, positive if external, and zero if on an edge.

There are three existing ITK filters for computing distance transforms:

1. `itk::SignedDanielssonDistanceMapImageFilter`
2. `itk::ApproximateSignedDistanceMapImageFilter`
3. `itk::SignedMaurerDistanceMapImageFilter`

The first of these computes an approximation of the SEDT with less than 1 pixel error in two dimensions[2]. The second computes an approximation using the Chamfer distance[3]. The third computes the exact SEDT and is an implementation of Maurer, *et al.*[1][5] All three can compute the SEDT to arbitrary dimensions and can handle anisotropic image dimensions.

In this paper we present a new ITK filter, SignedMaurerParallelDistanceMapImageFilter, for computing the signed exact Euclidean distance transform for 3D images in parallel. It is approximately 3 times faster than `itk::SignedMaurerDistanceMapImageFilter` on 4 simultaneous multithreading (SMT) processors with 2 threads each. We compare output and speeds of our parallel 3D implementation based on Maurer's algorithm with the Maurer and Danielsson filters and give implementation details on our filter.

2 Algorithm

In Maurer[1], the authors present a fast (linear time) algorithm for computing the exact distance transforms using the L_p distance metrics and, more generally, the weighted L_p distance metrics

$$\Delta(x,y) = \left(\sum_{i=1}^{k} w_i \left| x_i - y_i \right|^p \right)^{\frac{1}{p}}, w_i > 0, \tag{1}$$

the difference between weighted (anisotropic) and non-weighted (isotropic) calculations being that between 1 and $\neq 1$ values of the w_i term. We treat the specific application of this to the 3D Euclidean distance

$$\Delta(x,y) = \left(\sum_{i=1}^{3} w_i \left| x_i - y_i \right|^2 \right)^{\frac{1}{2}}, w_i > 0. \tag{2}$$

The base algorithm for SignedMaurerParallelDistanceMapImageFilter relies on dimensional reduction. First, the x dimension is considered and the closest feature voxel (CFV) is determined for every element in the image. Using the CFV calculations for x, the same can be done for y, and then z in turn. Our parallelization distributes contiguous groups of z planes to the threads. After all threads have finished their first and second dimension CFV calculation, groups of contiguous x planes are assigned to the threads for the final step. This *a priory* method maximizes cache hits and has a low run queue creation/access cost. Each thread calculates its processing region independent of others and this region is protected from access by other threads. Thus the region calculation cost is minimized and there are no synchronization costs directly associated with region access management.

A combination of this method and a run queue would also maximize cache hits but would suffer from queue-associated costs and, because of the highly symmetric nature of the image processing, would not generally provide significant benefits in load-balancing. The batch regions of slices would need to be determined statically and access to the shared queue would need to be controlled (i.e., by a mutex), both resulting in performance losses. The number of operations is largely the same across an image, varying somewhat by the number of features in a scan line (and thus, region). High load imbalances are therefore not especially likely and thus the cost due to dynamic load-balancing can be eliminated. Experimental comparisons with a load-balancing version of our filter also fail to show consistent benefits for any particular degree of load-balancing tested.

This analysis will be expanded further elsewhere.[4]

3 Filter Use and Implementation

This filter was designed to adhere to the expectations of users of the Danielsson and `itk::SignedMaurerDistanceMapImageFilter` filters and as such shares many of the same options. It takes as input an image of any type and can output either a squared or an actual distance transform.

Apart from slight differences in parameters the most noticeable divergence is that this is strictly a three-dimensional implementation. Two-dimensional images are easily handled secondarily through extraction. Our next implementation will be n-dimensional ($n = 2,3,4,\ldots$).

The parameters that differ between this filter and Danielsson or Maurer are as follows:

- SetForegroundValue allows the specification of the foreground element value (typically 1, which is the default). All other values are assumed to be background values.
- SetBinaryConvert specifies whether the input image is already a binary image of values 1 and 0. This situation corresponds to a value of false. The default value is true. This option is useful for skipping a processing step if more information about the input is known.
- SetNumberOfThreads sets the requested number of threads for the filter. The actual number of threads used may differ. It sets the requested number of threads for the filters used internally as well as to the filter's own processes. The default behavior is to request the maximum available threads on the system.

Our filter uses the following filters internally on at least one option:

- `itk::AddImageFilter` (for signed transform only)
- `itk::BinaryErodeImageFilter` (for signed transform only)
- `itk::BinaryThresholdImageFilter` (for non-preprocessed binary images)

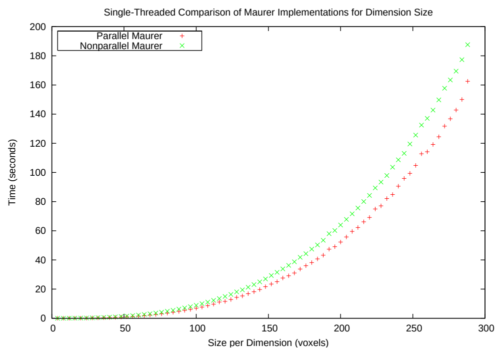
All three of these filters are of time $O(n/p)$ when used in parallel in the current release of ITK. The distance transform itself is $O(n/p)$ as stated by Maurer[1]. Therefore, these pre-transform operations result only in a higher constant factor in the run time and not an increase in order.

4 Comparative Results

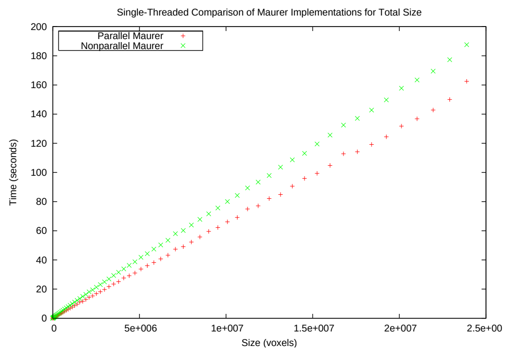
Our filter was compared with the `itk::SignedMaurerDistanceMapImageFilter` and Danielsson filters for accuracy and speed. The arithmetic difference between our filter and Maurer was found to be equal to zero in all cases using analogous settings, as expected from their use of the same base algorithm. The absolute difference between our filter and Danielsson was found to be less than 1, again as expected, and equalled the difference between the nonparallel Maurer filter and Danielsson. The speedup relative to Danielsson is in line with that demonstrated in Tustison, *et al.*[5]. A comparison with `itk::ApproximateSignedDistanceMapImageFilter` was also performed and our tests show a marked advantage in speed here as well.

Figures 1 and 2 display the various comparisons of runtimes for the three filters on a large number of image sizes for a standard test image (a cube of volume $\frac{1}{8}$ the total of the image). They clearly show the benefits of our filter in a multiple-thread environment. The advantage over the current ITK single-threaded implementation when running in a single-threaded environment can also be seen. The lower coefficient (as they are both still $O(n)$) is primarily due to the use of efficient memory-access and iterator use.

Table 1 shows the mean speedups for the filter. First is the unscaled speedup ratio of the processing time using 1 thread to the time for p processors, T_1/T_p, for the cube test data. Second is the mean projected scaled speedup using the processing time for image of size n with 1 thread and the time for image of size np with

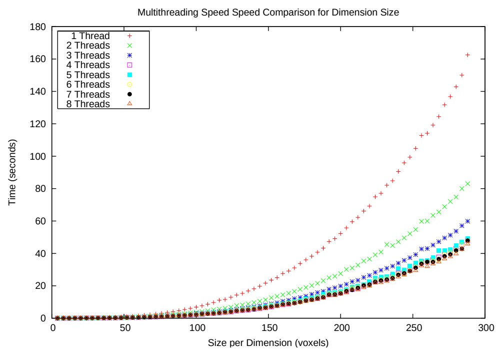


(a) For size of image in each dimension.

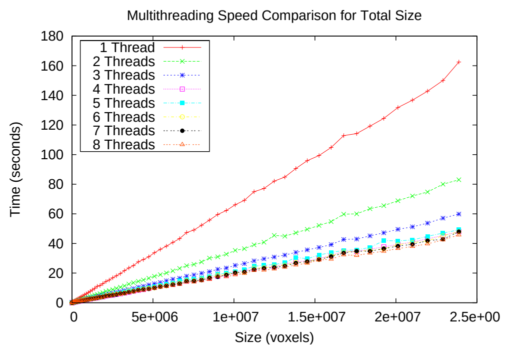


(b) For total size of image.

Figure 1: Comparison of run times of our parallel Maurer implementation running with a single thread and the existing ITK Maurer implementation.



(a) For size of image in each dimension.



(b) For total size of image.

Figure 2: Comparison of run times of our parallel Maurer distance map implementation for a varying number of requested threads.

p threads, $p \times T_1(n)/T_p(np)$. The scaled speedup thus represents a parallelization with constant region size per thread for each n. The values used for the scaled speedup are for the projected times for the sample cube data based on a linear regression of the data.

All tests were performed on a machine with 4 Intel Xeon 1.5 GHz processors with (2 SMT threads per processor) and 2 GB of physical memory running SuSE Linux 9.2 and using gcc 3.3.4 and ITK 2.8.1 (using revision 1.3 of `itk::SignedMaurerDistanceMapImageFilter`.

The shown test results are for the full number of multithreads on the tested system–8. However, a decrease in benefit of increased threads is clearly visible after 4 threads. When the filter actively uses both SMT threads on a processor these processors generate resource access costs that offset some of the parallelization benefits.[4]

Number of Threads	2	3	4	5	6	7	8
Mean Speedup	1.819	2.461	2.972	2.767	2.920	2.955	3.060
Mean Speedup	1.740	2.445	3.054	2.880	3.085	3.172	3.325

Table 1: Mean speedup T_1/T_p for cube data and mean projected scaled speedup $p \times T_1(n)/T_p(np)$ for cube data.

5 Conclusions

Our introduction of SignedMaurerParallelDistanceMapImageFilter, an ITK filter which calculates the signed Euclidean distance in parallel, will be of assistance in various parallel image processing applications. Its theoretical complexity of $O(n/p)$ is a substantial improvement over filters which calculate the estimated EDT or which calculate the exact EDT sequentially. The filter behaves similarly to the established filters and should thus be easily adapted to existing applications. Our planned extension from 3D to arbitrary dimensions will make this integration even easier.

References

[1] Calvin R. Maurer, Jr., Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):265–270, 2003. `http://dx.doi.org/10.1109/TPAMI.2003.1177156`. (document), 1, 2, 3

[2] P. E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980. 1

[3] Karl Krissian and Carl-Fredrik Westin. Fast and accurate redistancing for level set methods. In Roberto Moreno Diaz and Alexis Quesada Arencibia, editors, *Computer Aided Systems Theory (EUROCAST'03)*, pages 48–51, Las Palmas de Gran Canaria, Spain, February 24–28 2003. Universidad de Las Palmas de Gran Canaria. 1

[4] R. Staubs, A. Fedorov, L. Linardakis, and N. Chrisochoides. An ITK Filter for Parallel Computation of Euclidean Distance Transforms. *To be submitted to ACM Transactions on Mathematical Software*, 2006. 2, 4

[5] N. J. Tustison, M. Siqueira, and J. Gee. N-D linear time exact signed euclidean distance transform. *The Insight Journal*, January - June 2006. `http://insight-journal.org/dspace/handle/1926/171`. (document), 1, 4