Design and Implementation of a Practical Parallel Delaunay Algorithm

G. E. Blelloch, J. C. Hardwick, G. L. Miller, and D. Talmor

Abstract. This paper describes the design and implementation of a practical parallel algorithm for Delaunay triangulation that works well on general distributions. Although there have been many theoretical parallel algorithms for the problem, and some implementations based on bucketing that work well for uniform distributions, there has been little work on implementations for general distributions. We use the well known reduction of 2D Delaunay triangulation to find the 3D convex hull of points on a paraboloid. Based on this reduction we developed a variant of the Edelsbrunner and Shi 3D convex hull algorithm, specialized for the case when the point set lies on a paraboloid. This simplification reduces the work required by the algorithm (number of operations) from $O(n \log^2 n)$ to $O(n \log n)$. The depth (parallel time) is $O(\log^3 n)$ on a CREW PRAM. The algorithm is simpler than previous $O(n \log n)$ work parallel algorithms leading to smaller constants.

Initial experiments using a variety of distributions showed that our parallel algorithm was within a factor of 2 in work from the best sequential algorithm. Based on these promising results, the algorithm was implemented using C and an MPI-based toolkit. Compared with previous work, the resulting implementation achieves significantly better speedups over good sequential code, does not assume a uniform distribution of points, and is widely portable due to its use of MPI as a communication mechanism. Results are presented for the IBM SP2, Cray T3D, SGI Power Challenge, and DEC AlphaCluster.

Key Words. Delaunay triangulation, Parallel algorithms, Algorithm experimentation, Parallel implementation.

1. Introduction. A Delaunay triangulation in $\mathbb{R}^2$ is the triangulation of a set $S$ of points such that there are no elements of $S$ within the circumcircle of any triangle. Delaunay triangulation—along with its dual, the Voronoi Diagram—is an important problem in many domains, including pattern recognition, terrain modeling, and mesh generation for the solution of partial differential equations. In many of these domains the triangulation is a bottleneck in the overall computation, making it important to develop fast algorithms. As a consequence, there are many sequential algorithms available for Delaunay triangulation, along with efficient implementations. Su and Drysdale [1] present an excellent experimental comparison of several such algorithms. Since these algorithms are time and memory intensive, parallel implementations are important both for improved performance and to allow the solution of problems that are too large for sequential machines. However, although several parallel algorithms for Delaunay triangulation have been described [2]–[7], practical implementations have been slower to appear, and are
mostly specialized for uniform distributions [8]–[11]. One reason is that the dynamic
nature of the problem can result in significant interprocessor communication. This is
particularly problematic for nonuniform distributions. A second problem is that the par-
allel algorithms are typically much more complex than their sequential counterparts.
This added complexity results in low parallel efficiency; that is, the algorithms achieve
only a small fraction of the perfect speedup over efficient sequential code running on one
processor. Because of these problems no previous implementation that we know of has
achieved reasonable speedup over good sequential algorithms when used on nonuniform
distributions.

Our goal was to develop a parallel Delaunay algorithm that is efficient both in theory
and in practice, and works well for general distributions. In theory we wanted an algorithm
that for $n$ points runs in polylogarithmic depth (parallel time) and optimal $O(n \log n)$
work. We were not concerned with achieving optimal depth since no machines now or
in the foreseeable future will have enough processors to require such parallelism. In
practice we wanted an algorithm that performs well compared with the best sequential
algorithms over a variety of distributions, both uniform and nonuniform. We considered
two measures of efficiency. The first was to compare the total work done to that done by
the best sequential algorithm. We quantify the constants in the work required by a parallel
algorithm relative to the best sequential algorithm using the notion of $\alpha$ work-efficiency.
We say that algorithm A is $\alpha$ work-efficient compared with algorithm B if A performs
at most $1/\alpha$ times the number of operation of B. An ideal parallel algorithm is 100%
work-efficient relative to the best sequential algorithm. We use floating-point operations
as a measure of work—this has the desirable property that it is machine independent.
The second measure of efficiency is to measure actual speedup over the best sequential
algorithm on a range of machine architectures and sizes.

Based on these criteria we considered a variety of parallel Delaunay algorithms.
The one eventually chosen uses a divide-and-conquer projection-based approach, based
loosely on the Edelsbrunner and Shi [12] algorithm for 3D convex hulls. Our algorithm
does $O(n \log n)$ work and has $O(\log^3 n)$ depth on a CREW PRAM. From a practical
point of view it has considerably simpler subroutines for dividing and merging sub-
problems than previous techniques, and its performance has little dependence on data
distribution. Furthermore, it is well suited as a coarse-grained partitioner, which splits
up the points evenly into regions until there are as many region as processors, at which
point a sequential algorithm can be used. Our final implementation is based on this idea.

Our experiments were divided into two parts. A prototyping phase used the parallel
programming language NESL [13] to experiment with algorithm variants, and to measure
their work-efficiency. An optimized coarse-grained implementation of the final algorithm
was then written in C and a toolkit based on MPI [14], and was compared with the best
existing sequential implementation. For our measurements in both sets of experiments
we selected a set of four data distributions which are motivated by scientific domains
and include some highly nonuniform distributions. The four distributions we use are
discussed in Section 3.1 and pictured in Figure 8.

Our NESL experiments show that the algorithm is 45% work-efficient or better for
all four distributions and over a range of problem sizes when applied all the way to the
end. This is relative to Dwyer’s algorithm, which is the best of the sequential Delaunay
algorithms studied by Su and Drysdale [1]. Figure 1 shows a comparison of floating-point