Parallel Algorithm for Natural Neighbor Interpolation

Alexander Tsidaev

[1] Bulashevich Institute of Geophysics, Yekaterinburg, Russia
[2] Ural Federal University, Yekaterinburg, Russia
pdc@tsidaev.ru

Abstract. This paper describes parallelization technique for Natural Neighbor interpolation algorithm. It is based on Green-Sibson Voronoi tesselation method and works without use of Delaunay triangulation. Obtained results show that algorithm is efficient enough to be used for sequential interpolation of multilayer data.

Keywords: interpolation · natural neighbor · parallel algorithm · geophysics · CUDA · OpenMP

1 Introduction

Interpolation is very important part of many scientific processes, which rely on measured data. Obtaining of such a data is always complicated with different factors: usually it is lack of resources and physical impossibility to perform measurements in some point. Also, most of the interpretation methods work with regular grids while real data is usually measured as a set with irregular structure.

This is why the interpolation is frequently needed to convert measured sets to some convenient initial models, which will be then interpreted.

Particularly, author experiences the need of fast interpolation schemes during conversion of seismic profile data to 3D model on regular grids [1]. Example is presented on Fig. 1.

Empirically it has been found, that the best results are obtained using Natural Neighbor algorithm [2]. But on relatively large grids (1M points and more) this algorithm, which requires Voronoi diagram calculation for each point, works with insufficient efficacy. I.e. in 3D layered models the interpolation should be performed independently in a big set of layers (e.g. Fig. 1 consists of 801 layers: 80 km depth per 100 m). Even algorithms implemented in commercial software (such as Golden Software Surfer) took hours and even days to complete interpolation for all layers. So the efficient parallel natural neighbor algorithm is required.

Current paper presents the needed theory for the problem (Sections 2 and 3), algorithm itself (Section 4) and the result of the test in comparison with non-optimized implementation (Section 5).

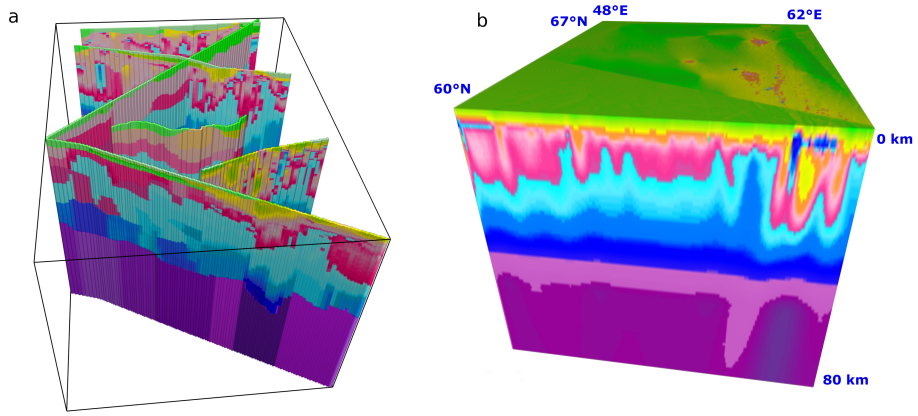

Fig. 1. Example of interpolation. Spare space between seismic profiles (a) is filled with interpolated values (b)

2 Voronoi Diagrams

Let X be the 2-dimensional plane and $\{P_j\} \in X$ be the set of the points (called seeds, sites or generators). Voronoi tesselation [3] of the X is a set of regions (polygons), each containing one seed and all points closer to that seed than to any other (Fig. 2):

$$R_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j)\} \tag{1}$$

Here R_k - element of Voronoi diagram; $d(x, a)$ is a distance between points x and a and $d(x, A) = \inf\{d(x, a) \mid a \in A\}$.

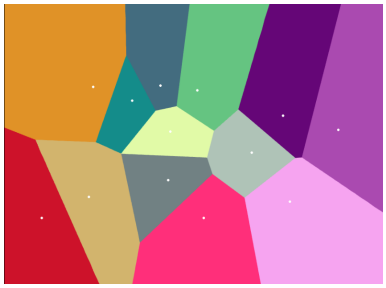

Fig. 2. Voronoi diagram for 13 points (in white). Each element is presented in a different color

It is easy to notice that all segments that form the boundaries between regions are the perpendiculars bisectors to the segments, which connect points of the initial set.

3 Natural Neighbor Algorithm

Natural Neighbor algorithm had been developed in 1981 by Robin Sibson [2]. Input data for the algorithm is a set of points $\{(x_i, y_i)\}_{i=0}^{N}$ given with some function f values, specified in these points: $\{f(x_i, y_i)\}_{i=0}^{N}$. Algorithm idea is to calculate Voronoi diagram for all initial points (x_i, y_i), and then to add each interpolated point (x, y) into the tessellation with sequential diagram recalculation. The value $G(x, y)$, which is attributed to the interpolated point, depends on how much of the area of initial diagram elements was "stolen" by the region of new inserted point (Fig. 3).

$$G(x, y) = \sum_{i=1}^{N} w_i f(x_i, y_i) \tag{2}$$

here $f(x_i, y_i)$ is the measured value in point (x_i, y_i), $w_i = \frac{Q_k}{R_k}$ is ratio of "stolen" area (see Fig. 3). R_k is the area of the initial Voronoi diagram element for point P_k; Q_k is the intersection area of R_k and newly constructed element for the point (x, y).

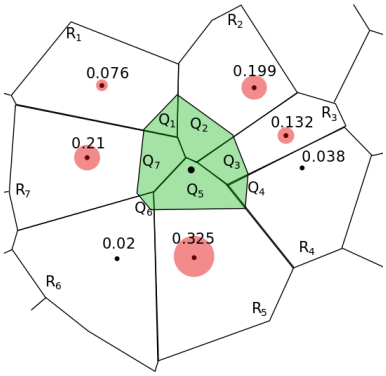

Fig. 3. Natural neighbor interpolation. The coloured circles, which represent the interpolating weights, w_i, are generated using the ratio of the shaded area to the cell area of the surrounding points. The shaded area is new Voronoi element for the point to be interpolated (based on image by Markluffel, distributed under CC BY-SA 3.0)

So the natural neighbor algorithm is basically the algorithm to insert an additional point into existing Voronoi diagram. And it is well known that Voronoi diagram is a dual graph of Delaunay triangulation. This is why most of methods of the iterative Voronoi diagram construction are based on triangulation (e.g. [4], [5]). This approach has many advantages in case when many incremental steps should be performed sequentially. But in our case we always need to insert only one new point and to calculate resulting weights w_i then. After this the obtained modified Voronoi diagram is no longer needed and we construct next

one for the next point. The weight w_i calculation depends on geometrical features of Voronoi diagram and could not be performed on its dual graph. This means that natural neighbor method requires reconstruction of Voronoi from Delaunay on each iteration step if implemented using the triangulation. In this research we use an algorithm for incremental Voronoi diagram construction by P.J. Green and R. Sibson [6] with natural neighbor weights calculation added. It does not require costly conversion of Voronoi graph to Delaunay graph and back on any step.

4 Iterative Algorithm for Voronoi Diagram Construction

As described in previous section, the natural neighbor algorithm requires insertion of new point and recalculation of Voronoi diagram. To speed up the process and make easier understanding of how the existing polygons shapes were modified, it is better to implement incremental point insertion algorithm instead of whole diagram recalculation.

Let P_i be the initial points set with values $f(P_i)$. Let R_i be the Voronoi polygon for site P_i. We need to add new site P into the Voronoi diagram for P_i. Let $d(P_a, P_b)$ be the distance between points P_a and P_b. Then the algorithm can be written as follows.

Algorithm 1 Incremental Voronoi diagram construction step of Natural Neighbor interpolation method

1: **Select** ▷ Get the closest to P point

$$P_n : d(P, P_n) = \min_{\forall i}\{d(P, P_i)\}$$

2: **Calculate perpendicular bisector to segment** PP_n
3: **Find two edges of** R_n**, which cross the bisector:** E_1 **and** E_2 ▷ Assume for definiteness that rotation direction of minimum angle between E_1 and E_2 with respect to point P is counter-clockwise.
4: **Construct first edge of new Voronoi diagram element** R **for point** P **by connecting together points of intersection with bisector**
5: **Calculate coefficient** w_n **as ratio of areas as described in previous section**
▷ Now constructing remaining edges of region R in counter-clockwise direction
6: *finalEdge* $\leftarrow E_1$
7: **while** $E_2 \neq$ *finalEdge* **do** ▷ While rotation is not finished and we have not returned to the R_n
8: **Find region** R_k**, which shares edge** E_2 **with** R_n
9: $(E_1, E_2) \leftarrow$ findIntersectedEdges(P, R_k) ▷ Same as steps 2-3 of the algorithm
10: **Construct next edge of new region** ▷ See step 4
11: **Calculate** w_k ▷ See step 5

12: **end while**
13: **return** $\sum_{i=1}^{N} w_i f(x_i, y_i)$ ▷ All region edges were constructed and result value can be calculated

5 Parallel Implementation and Test Results

Parallelization idea is quite simple and follows from the non-obstructive nature of Algorithm 1. Since the value in each interpolation point is calculated independently and none of the initial data is modified during process, then any number of point values can be calculated at the same time.

Algorithm 1 was implemented in C++ for two parallel platforms: OpenMP and CUDA. Initial interpolation point set consisted of 644 points. Interpolation grid was a square of 1024x1024 points. All tests were performed on "Uran" supercomputer (Krasovskii Institute of Mathematics and Mechanics, Yekaterinburg, Russia).

Table 1. Interpolation time for a single layer (of 801)

Hardware	**Calculation time, s**	**Speed up factor**
1 CPU core (Intel Xeon E5520)	187	1x
8 CPU cores (Intel Xeon E5520 OpenMP)	119	1.57x
1 GPU (Tesla M2050)	29.916	6.25x
2 GPUs (Tesla M2050)	17.096	10.94x
4 GPUs (Tesla M2050)	14.643	12.77x

As it is seen from the Table 1, OpenMP parallelization showed unsatisfactory results. This may be connected with the fact that no attempts to perform manual optimizations were implemented and only the main loop was parallelized. Also it is important to note that other Intel-based optimization techniques, such as vectorization, are not applicable for current algorithm at all (because of branching and different length of loops for different regions) - and so there is no chance to get additional speed up by introducing SIMD instructions.

Acceleration obtained with GPU usage is quite acceptable, speed up factor is around one order of magnitude for current grid size and 4 GPUs. Our previous research showed that calculation time can have almost linear dependency versus number of GPU [7]. But in current case increase of involved GPUs does not entail corresponding linear increase of acceleration factor. This may be caused by complex program structure, which cannot be efficiently parallelized, and overhead connected with need to copy all the data to all GPUs. Program complexity caused the allocation of maximum allowed number of GPU registers, and this significantly reduced number of blocks that can be executed in parallel.

6 Conclusion

Parallel algorithm for natural neighbor interpolation was presented in this paper. Despite the obtained acceleration is enough for the current task of geophysical modeling, it seems perspective to implement algorithm for different parallelization platforms, such as MPI. Test example showed that even such complex algorithms, which cannot be vectorized well, can still be efficiently parallelized. And even MPI seems to be the most appropriate solution for parallel execution of inhomogeneous blocks, GPU computations could be used as a much cheaper alternative.

References

1. Ladovsky I.V., Martyshko P.S., Druzhinin V.S., Byzov D.D., Tsidaev A.G., Kolmogorova V.V.: Methods and results of crust and upper mantle volume density modeling for deep structure of the Middle Urals region. Ural Geophysical Messenger. 2(22). 31–45 (2013)
2. Sibson, R.: A brief description of natural neighbor interpolation (Chapter 2). In V. Barnett. Interpreting Multivariate Data. Chichester: John Wiley. 21–36 (1981)
3. Voronoi, G.: Nouvelles applications des paramtres continus la thorie des formes quadratiques. Journal fr die Reine und Angewandte Mathematik. 133 (133): 97–178 (1908)
4. Fan, Q., Efrat, A., Koltun, V., Krishnan, S., Venkatasubramanian, S.: Hardware-assisted natural neighbor interpolation. In C. Demetrescu, R. Sedgewick, R. Tamassia (Eds.), Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithms and Combinatorics. 111–120 (2005)
5. Guibas, L.J., Knuth, D.E., Sharir, M.: Randomized Incremental Construction of Delaunay and Voronoi Diagrams. Algorithmica 7. 381–413 (1992)
6. Green, P. J., Sibson, R.: Computing Dirichlet Tessellations in the Plane. The Computer Journal 21 (2): 168–173 (1978)
7. Tsidaev, A.: CUDA Parallel Algorithms for Forward and Inverse Structural Gravity Problems. Proceedings of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists. 50–56 (2015)