Parallel \textit{d}-D Delaunay Triangulations in Shared and Distributed Memory
Daniel Funke and Peter Sanders

Jared Di Carlo

March 14, 2019
Triangulation in Arbitrary Dimensions

Definition

A $k$-simplex is a $k$-dimensional generalization of a triangle.
3D Triangulation
Definition
In $\mathbb{R}^n$, given a set $S$ of points, the Delaunay Triangulation of $S$ is a triangulation of $S$ such that the circumsphere of any simplex in $\mathcal{D}(S)$ does not contain any other point in $S$. 
In $\mathbb{R}^2$, Delaunay maximizes the minimum angle

In $\mathbb{R}^3$, it doesn’t

In $\mathbb{R}^n$, it minimizes the maximum containment sphere

Number of tetrahedra in 3D Delaunay tetrahedralization

- $O(n)$: uniform volume
- $O(n \log n)$: uniform surface
- $O(n^2)$: worst case (two lines)
Sequential 3D Delaunay

- Randomized insertion
- Flipping
- Watson Method
Watson Method
Strategy

(a) partitioning

(b) partial DTs

(c) border DT

(d) final DT
Part 1: Split

1: if $n < N \lor r = \log P$ then
2: \hspace{1em} return sequentialDelaunay($P$)
3: $k \leftarrow \text{splittingDimension}(P)$
4: $(P_1 \; P_2) = (p_1 \; \cdots \; p_s \; | \; p_{s+1} \; \cdots \; p_n) \leftarrow \text{divide}(P, k)$
5: $T = (T_1 \; T_2) \leftarrow (\text{Delaunay}(P_1, r + 1) \; \text{Delaunay}(P_2, r + 1))$

- Constant
- Alternating
- Largest
Part 2: Finding Border Triangles

6: $B \leftarrow \emptyset$; $Q \leftarrow \text{convexHull}(T_1) \cup \text{convexHull}(T_2)$ \hspace{1cm} \triangleright \text{initialize}

7: \textbf{parfor} $s_{i,x} \in Q$ \textbf{do} \hspace{1cm} \triangleright \text{si}

8: \hspace{1cm} \text{mark}(s_{i,x})

9: \hspace{1cm} \textbf{if} \circumsphere(s_{i,x}) \cap \text{boundingBox}(T_j) \neq \emptyset, \text{ with } i \neq j \textbf{ then}

10: \hspace{2cm} B \cup= \{s_{i,x}\} \hspace{1cm} \triangleright \text{circumsphere intersects other}

11: \hspace{2cm} \textbf{for} s_{i,y} \in \text{neighbors}(s_{i,x}) \land \neg \text{marked}(s_{i,y}) \textbf{ do}

12: \hspace{2cm} Q \cup= s_{i,y}; \hspace{0.5cm} \text{mark}(s_{i,y})

13: T_B \leftarrow \text{Delaunay}(\text{vertices}(B), r + 1)
Part 3: Merge

14: $T \leftarrow (T_1 \cup T_2) \setminus B$; $Q \leftarrow \emptyset$
15: parfor $s_b \in T_B$ do
16: if vertices($s_b$) $\not\subseteq P_1 \land$ vertices($s_b$) $\not\subseteq P_2$ then
17: $T \cup= \{s_b\}$; $Q \cup= \{s_b\}$
18: else
19: if $\exists s \in B : \text{vertices}(s) = \text{vertices}(s_b)$ then
20: $T \cup= \{s_b\}$; $Q \cup= \{s_b\}$
Part 4: Update Data Structures

Neighborhood update:

21: parfor \(s_x \in Q\) do
22:     for \(d \in \{1, \ldots, D + 1\}\) do
23:         if \(\text{neighbors}_d(s_x) \not\in T\) then
24:             \(C \leftarrow \{s_c : f_d(s_x) = f_d(s_c)\}\)
25:         for \(s_c \in C\) do
26:             if \(|\text{vertices}(s_x) \cap \text{vertices}(s_c)| = D\) then
27:                 \(\text{neighbors}_d(s_x) \leftarrow s_c;\) \(Q \cup= s_c\)
28:     return \(T\)
Data Structures

- Check for duplicates
  - Hash table of simplices
- Update Neighbors
  - Face hash table
  - Build during Part 2
- Merge Large Blocks
  - Binary Search Tree
  - $O(\log k)$ access after $k$ merges
Results

Relative Speedup

Distributions
- bubble
- ellipsoid
- lines
- normal
- uniform

$t_{tccal}/t_{tacc}$ vs cores
Results

Throughput and Memory Consumption

- 1 thread
- 4 threads
- 2 threads
- 8 threads

(points/s · 10^4)

(mem/mem_core)

1  2  4  8  16  32  64  128  256  512  1024  2048
Comparison Of Results

- Tested on 1 million uniformly distributed points (same dataset)
- Different output size depending on # of threads?
- Volume is wrong
- Intel E5-2699V3 (2014, 2.30 GHz, 18 cores), 32 GB RAM (nobody used this much)
- Very hard to find program (author has deleted published GitHub page)

<table>
<thead>
<tr>
<th></th>
<th>1 Core</th>
<th>16 Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>This Paper</td>
<td>233 s</td>
<td>42 s</td>
</tr>
<tr>
<td>CGAL</td>
<td>8 s</td>
<td>n/a</td>
</tr>
<tr>
<td>Tetgen</td>
<td>5 s</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Questions

- Results?
  - Paper: 50,000,000 points in 64 seconds on 32 cores (24k pts/(sec cpu))
  - My result: 1,000,000 points in 42 seconds on 16 cores (1.4k pts/(sec cpu))
- Do we actually need Delaunay?
  - Delaunay algorithms are $O(n \log n)$
  - Can get a “good” tetredralization in $O(n)$ time...