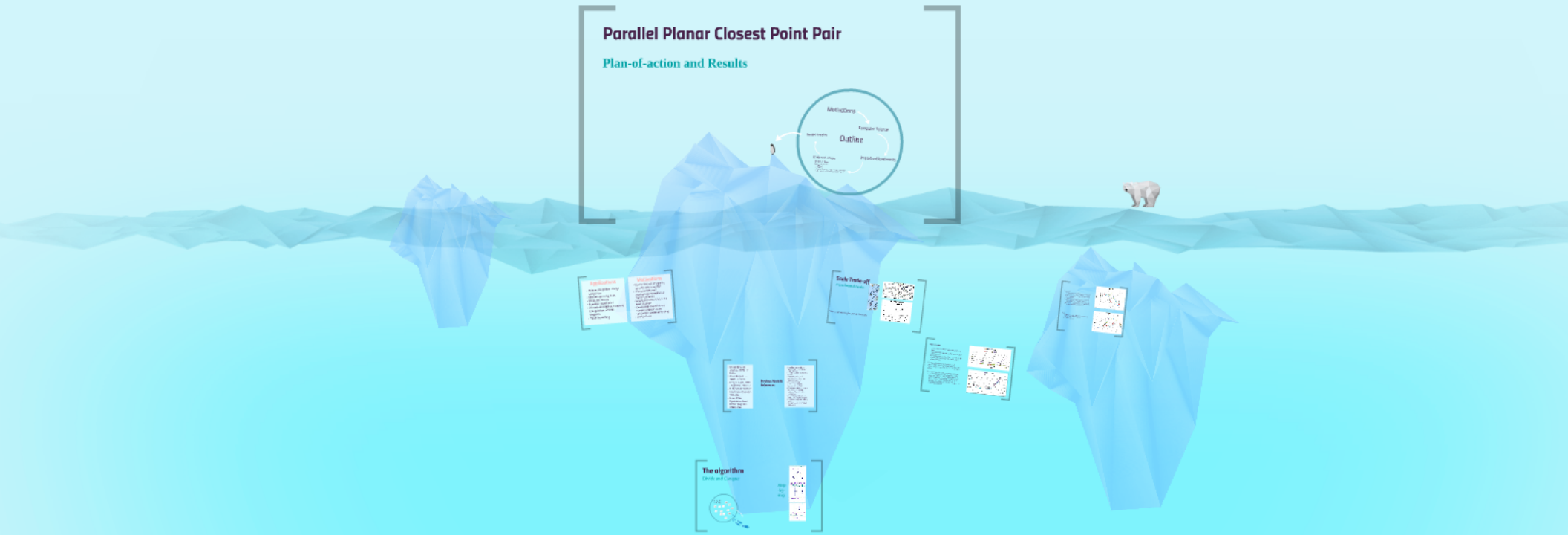
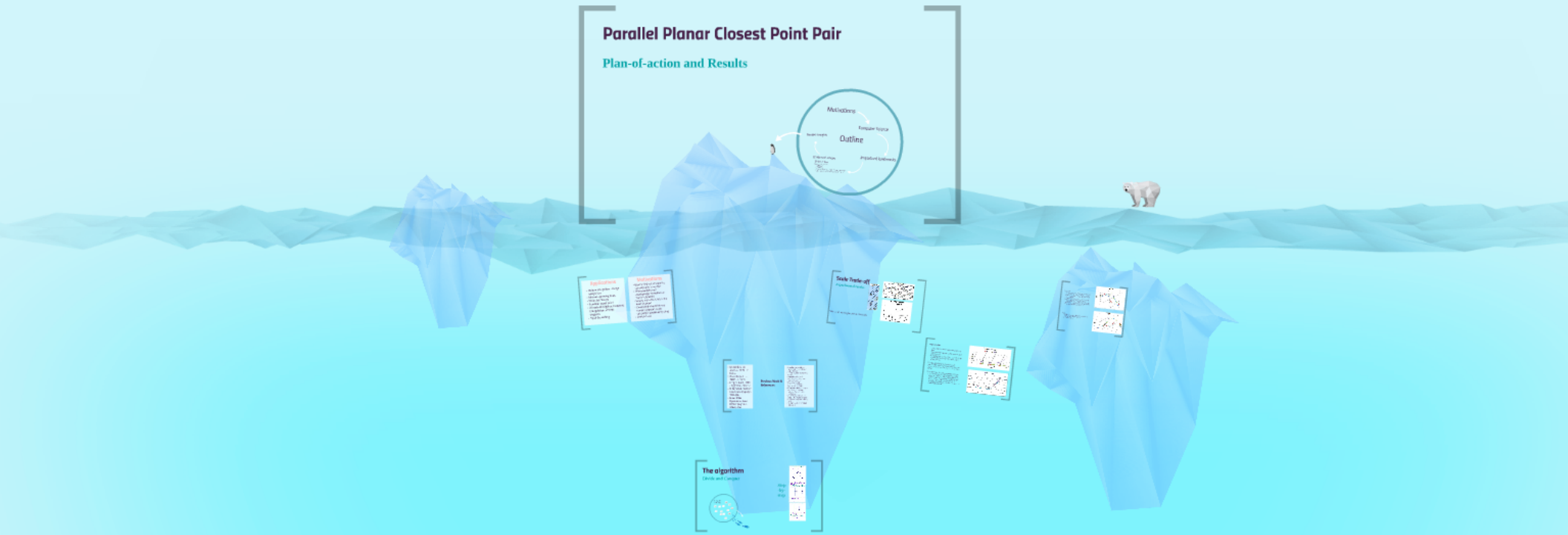


P - 4 | CSE633 Spring 2014

Parallel Planar Closest Point Pair

Angad Gadre & Piyush Sankla
Email: angadgad@buffalo.edu



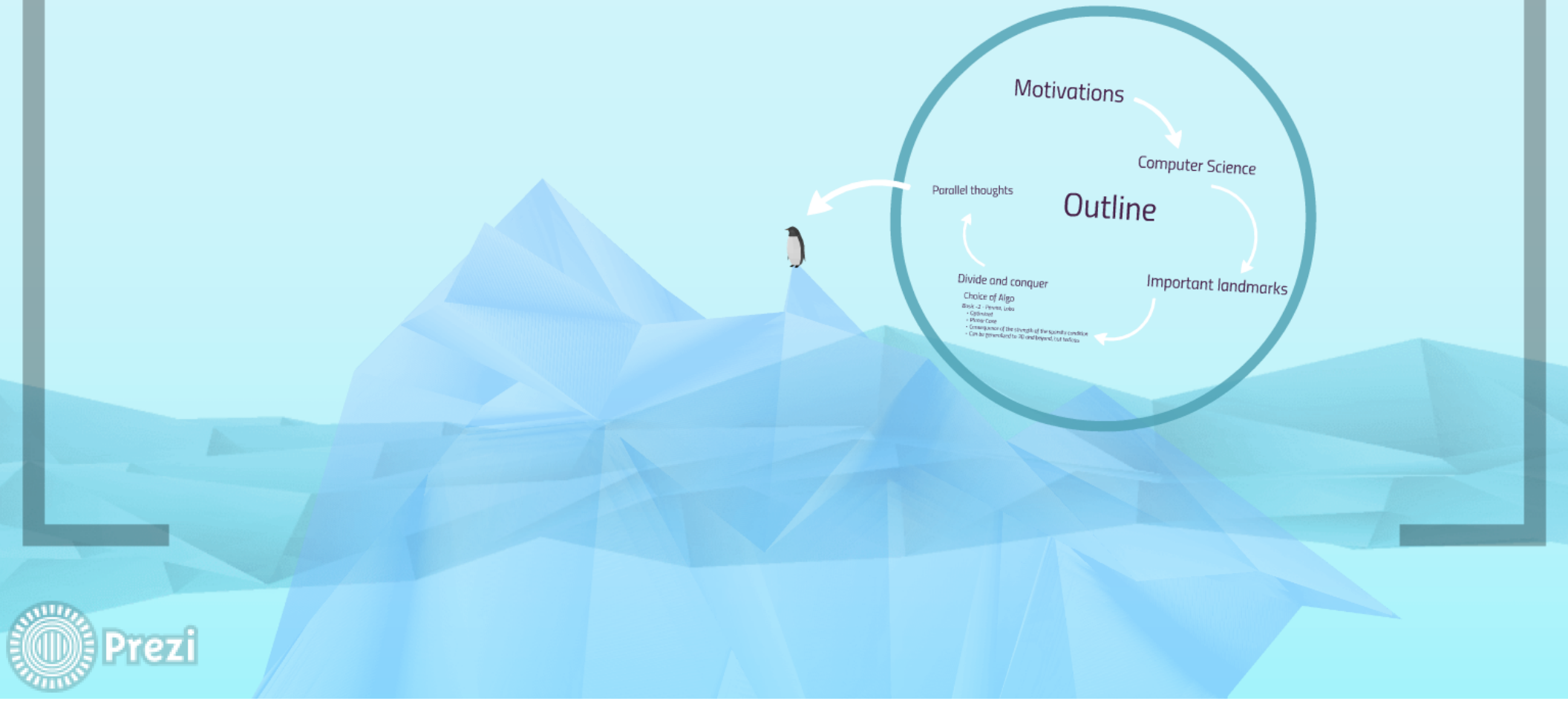
P - 4 | CSE633 Spring 2014

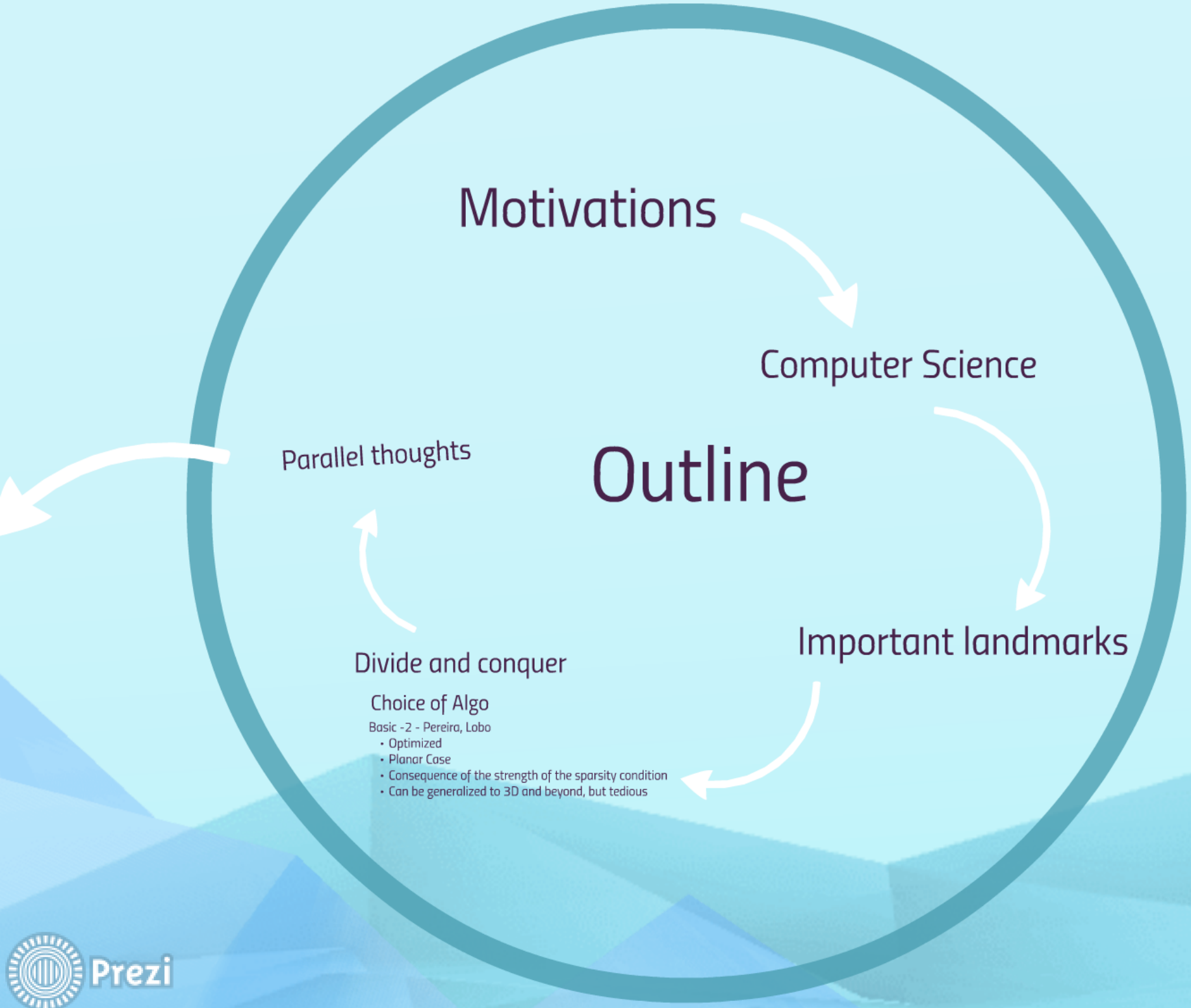
Parallel Planar Closest Point Pair

Angad Gadre & Piyush Sankla
Email: angadgad@buffalo.edu

Parallel Planar Closest Point Pair

Plan-of-action and Results




Motivations
Computer Science
Important landmarks
Divide and conquer
Choice of Algo
Basic -2 - Pereira, Lobo
• Optimized
• Planar Case
• Consequence of the strength of the sparsity condition
• Can be generalized to 3D and beyond, but tedious
Parallel thoughts
Outline

Applications

- Pattern recognition - Image recognition
- Minimal spanning trees
- Molecular biology
- Iterative closest point
- All nearest neighbor, Delaunay triangulation, Voronoi diagrams
- Tip of the iceberg

Motivations

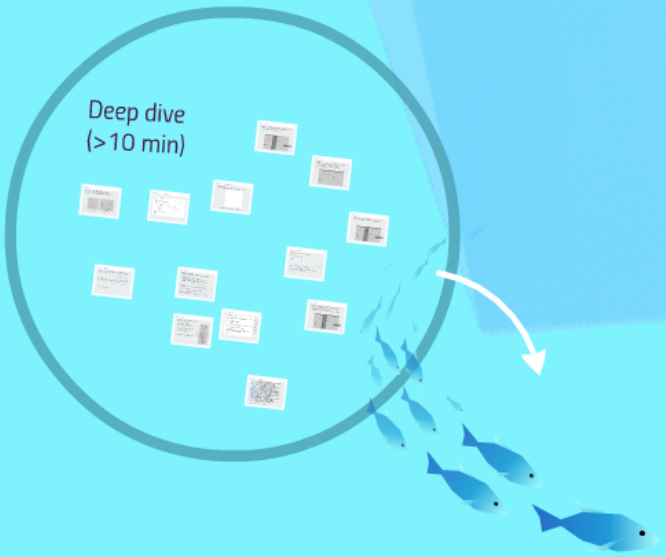
- Want to find out who are the closest people in my life?
- Who resembles me? Doppleganger or multiverse cousin? (Dynamic)
- How to reach Davis hall in the least distance?
- Understand why divide and conquer approaches are suitable for parallel computing
- Amdahl's law

- Jon Bentley, Ian Shamos, 1976 - 7 Points
- Zhou, Xiong, Zhu, 1998 - 4 points
- Jiang, Gillespie, 2007 - log factor - Basic 2
- Miller, Stout, Mesh of Processors O(sqrt(n)), 1984,86
- Boxer, Miller - Dynamic versions, PRAM O(log^2n). Widely cited

Previou
Referen

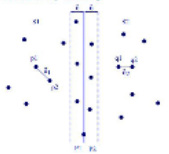
The algorithm

Divide and Conquer



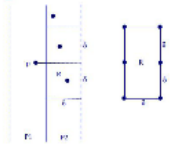
Step -by- step

- We partition S into S_1, S_2 by vertical line ℓ defined by median x-coordinate in S.
- Recursively compute closest pair distances δ_1 and δ_2. Set $\delta = \min(\delta_1, \delta_2)$.
- Now compute the closest pair with one point each in S_1 and S_2.



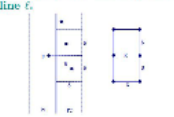
- In each candidate pair (p, q), where $p \in S_1$ and $q \in S_2$, the points p, q must both lie within δ of ℓ.

- Consider a point $p \in S_1$. All points of S_2 within distance δ of p must lie in a $\delta \times 2\delta$ rectangle R.



- How many points can be inside R if each pair is at least δ apart?
- In 2D, this number is at most 6!
- So, we only need to perform $6 \times n/2$ distance comparisons!

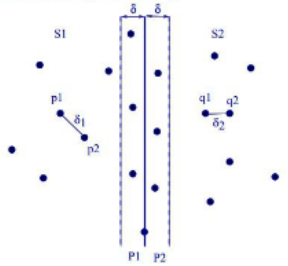
- In order to determine at most 6 potential mates of p_i, project p and all points of P'_2 onto line ℓ.



- Pick out points whose projection is within δ of p_i: at most six.
- We can do this for all p_i by walking sorted lists of P_1 and P_2, in total $O(n)$ time.
- The sorted lists for P_1, P_2 can be obtained from pre-sorting of S_1, S_2.
- Final recurrence is $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

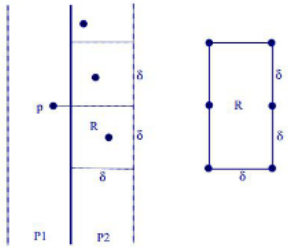
Step -by- step

- We partition S into S_1, S_2 by vertical line ℓ defined by median x-coordinate in S.
- Recursively compute closest pair distances δ_1 and δ_2. Set $\delta = \min(\delta_1, \delta_2)$.
- Now compute the closest pair with one point each in S_1 and S_2.



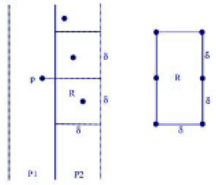
- In each candidate pair (p, q), where $p \in S_1$ and $q \in S_2$, the points p, q must both lie within δ of ℓ.

- Consider a point $p \in S_1$. All points of S_2 within distance δ of p must lie in a $\delta \times 2\delta$ rectangle R.



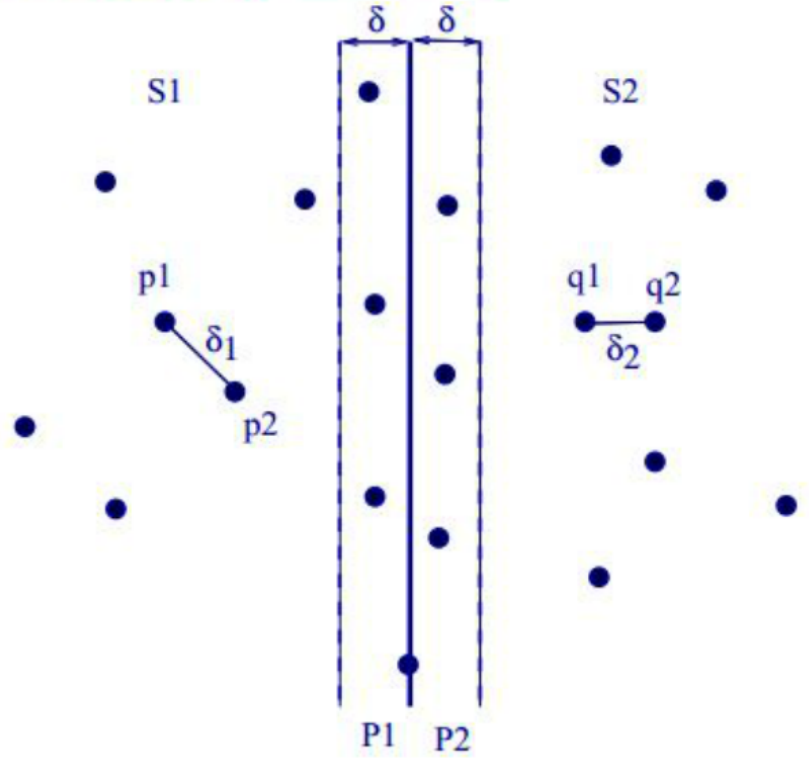
- How many points can be inside R if each pair is at least δ apart?
- In 2D, this number is at most 6!
- So, we only need to perform $6 \times n/2$ distance comparisons!

- In order to determine at most 6 potential mates of p, project p and all points of P_2 onto line ℓ.



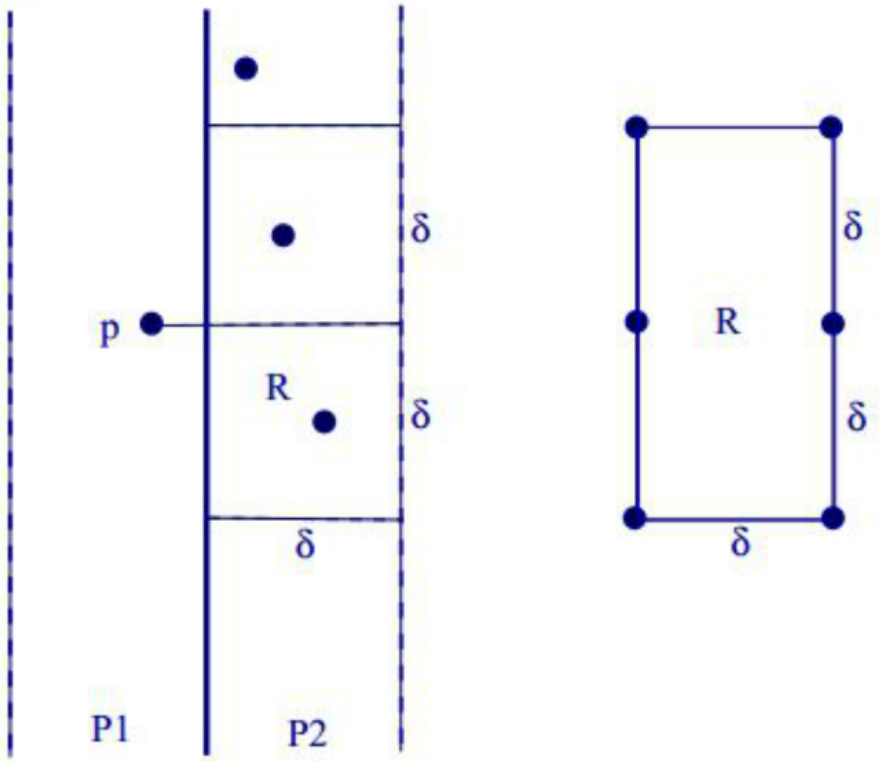
- Pick out points whose projection is within δ of p; at most six.
- We can do this for all p, by walking sorted lists of P_1 and P_2, in total $O(n)$ time.
- The sorted lists for P_1, P_2 can be obtained from pre-sorting of S_1, S_2.
- Final recurrence is $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

- We partition S into S_1, S_2 by vertical line ℓ defined by median x-coordinate in S.
- Recursively compute closest pair distances δ_1 and δ_2. Set $\delta = \min(\delta_1, \delta_2)$.
- Now compute the closest pair with one point each in S_1 and S_2.



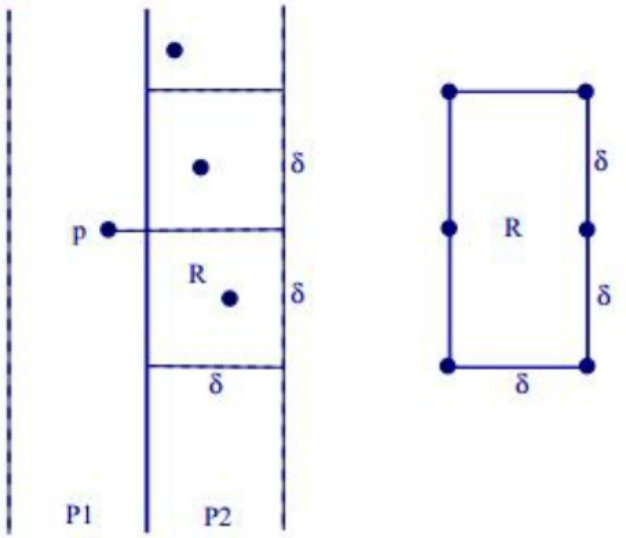
- In each candidate pair (p, q), where $p \in S_1$ and $q \in S_2$, the points p, q must both lie within δ of ℓ.

- Consider a point $p \in S_1$. All points of S_2 within distance δ of p must lie in a $\delta \times 2\delta$ rectangle R.



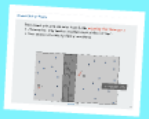
- How many points can be inside R if each pair is at least δ apart?
- In 2D, this number is at most 6!
- So, we only need to perform $6 \times n/2$ distance comparisons!

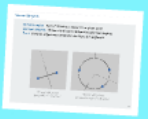
- In order to determine at most 6 potential mates of p, project p and all points of P_2 onto line ℓ.



- Pick out points whose projection is within δ of p; at most six.
- We can do this for all p, by walking sorted lists of P_1 and P_2, in total $O(n)$ time.
- The sorted lists for P_1, P_2 can be obtained from pre-sorting of S_1, S_2.
- Final recurrence is $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

Deep dive
(>10 min)











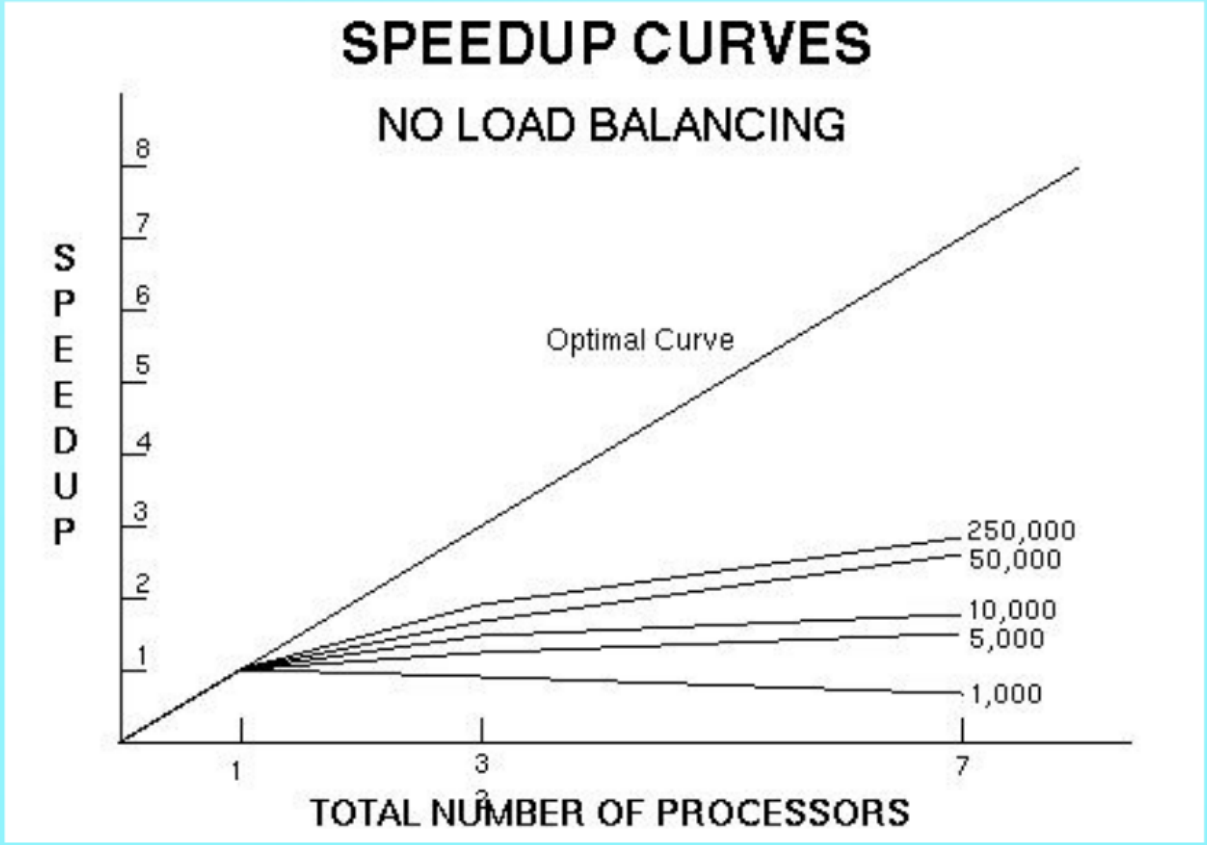


Brute Force	
ints	Time (sec)
100	0.000018
1000	0.001757
10000	0.164719
100000	16.323205
000000	Too long

de and Conquer	
ints	Time (sec)
100	0.000095
1000	0.000559
10000	0.006099
100000	0.081928
000000	1.110317
000000	2.550909
000000	5.810905
000000	7.703420
000000	12.714972
000000	14.095037
000000	16.361776
000000	19.348326

Timing Data for "Closest"

	Number of Points				
Machines	1000	5000	10000	50000	250000
1	*.82*	*3.63*	*8.15*	*54.83*	*470.87*
3	*.90*	*2.90*	*5.61*	*33.35*	*259.15*
7	*1.17*	*2.49*	*4.60*	*23.00*	*166.65*



Scale Trade-off

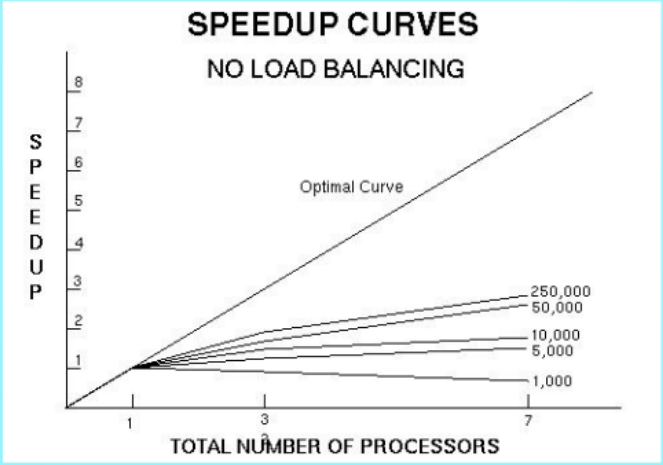
Experimental results

Brute Force	
#Points	Time (sec)
100	0.000018
1000	0.001757
10000	0.164719
100000	16.323205
1000000	Too long

Divide and Conquer	
#Points	Time (sec)
100	0.000095
1000	0.000559
10000	0.006099
100000	0.081928
1000000	1.110317
2000000	2.550909
4000000	5.810905
5000000	7.703420
6000000	12.714972
8000000	14.095037
9000000	16.361776
10000000	19.348326

Timing Data for "Closest"

	Number of Points				
Machines	1000	5000	10000	50000	250000
1	.82	3.63	8.15	54.83	470.87
3	.90	2.90	5.61	33.35	259.15
7	1.17	2.49	4.60	23.00	166.65



*John Thrall, Washington and Lee University

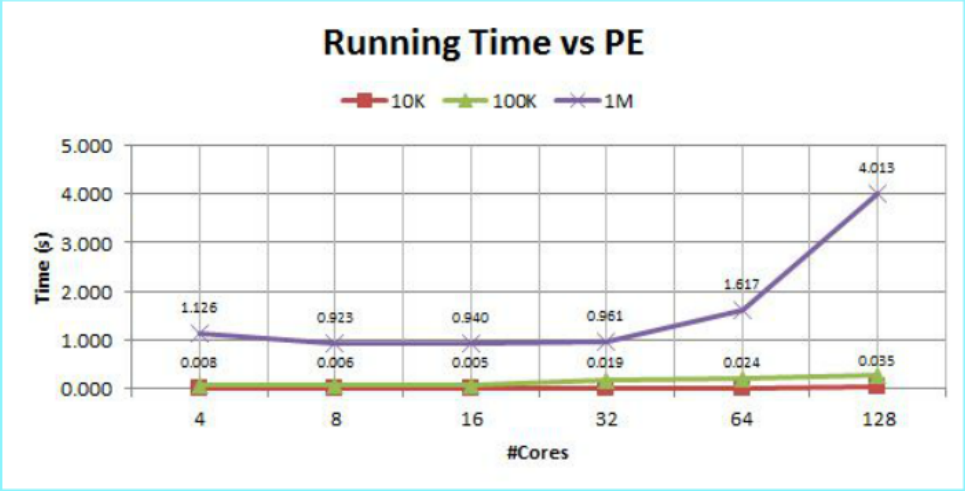
Approach for MPI

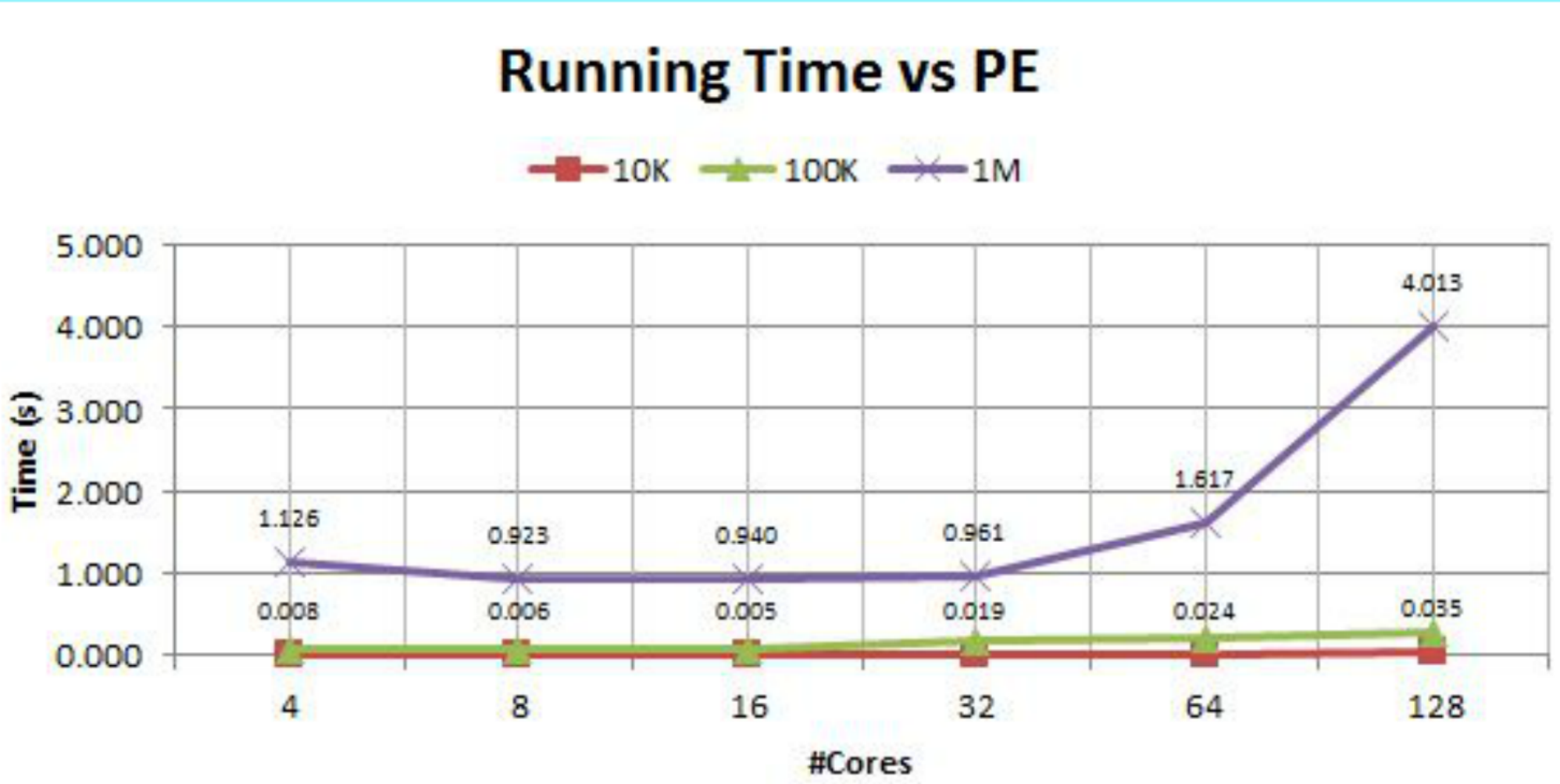
- Each process is given a section of the sequence of data.
- This is done by performing a scatter operation by the master process.
- Each process works on its subsequence individually using the P4 algorithm explained and sends its output for merge.

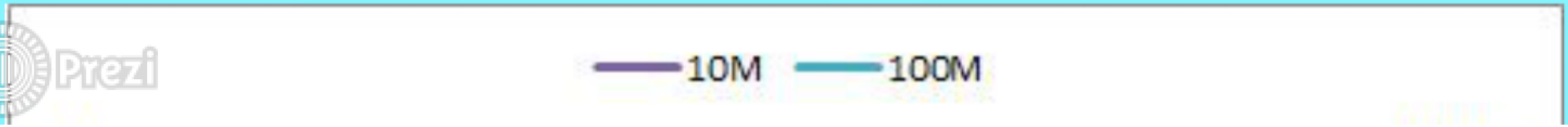
The major advantage of this technique is that communication time is kept low by sending only a sub-sequence to each process. But this is not an optimized solution.

The disadvantages of such a technique are as follows:

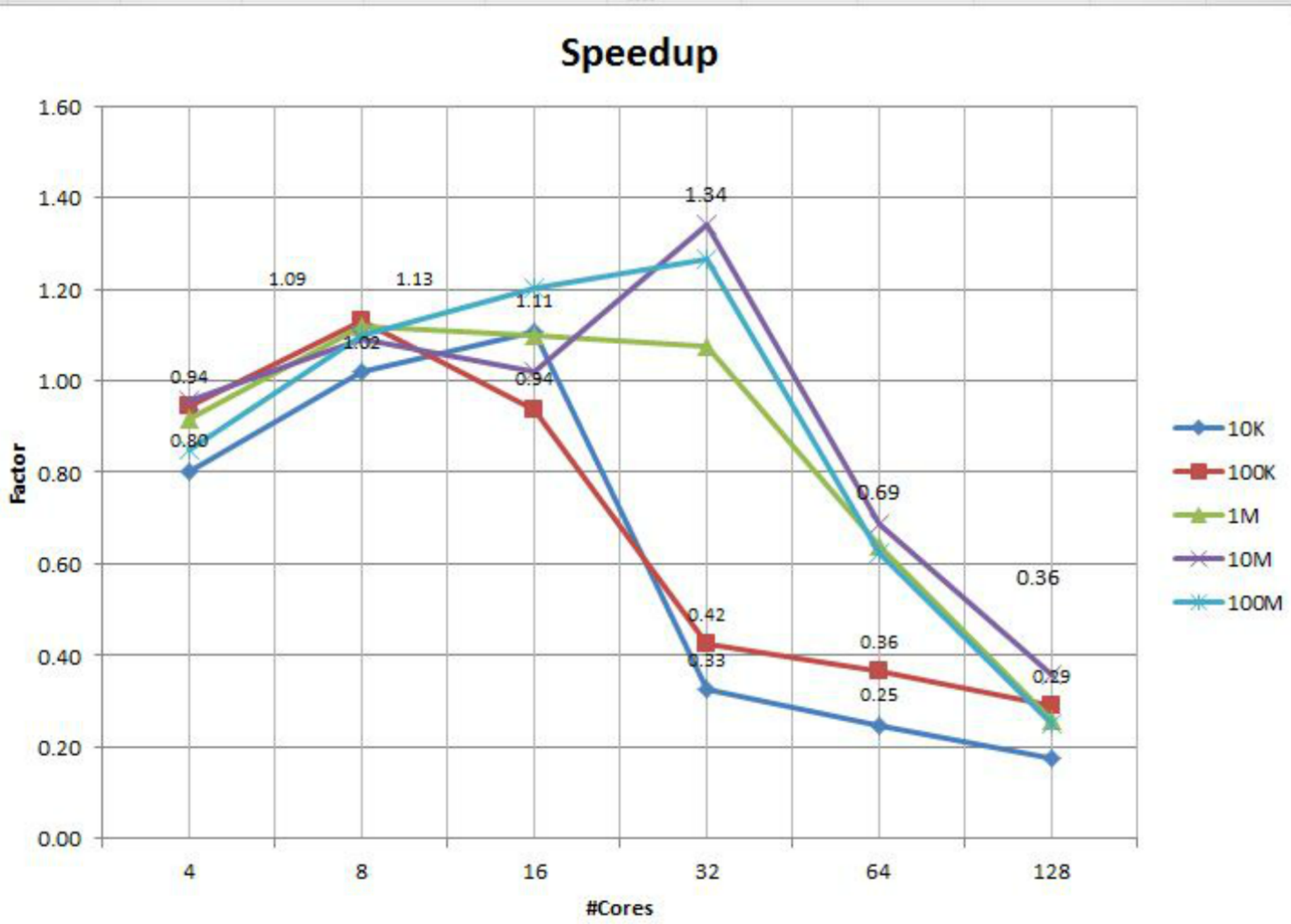
- Load balancing is not achieved since a particular process may finish its process and wait for merge operation while other processes are still busy. This leads to idle process time and is not efficient.
- Merge operations performed for each sub-sequence is also computationally.




Running Time vs PE
10K
100K
1M
5.000
4.000
3.000
2.000
1.000
0.000
Time (s)
1.126
0.923
0.940
0.961
1.617
4.013
0.008
0.006
0.005
0.019
0.024
0.035
4
8
16
32
64
128
#Cores


10M
100M

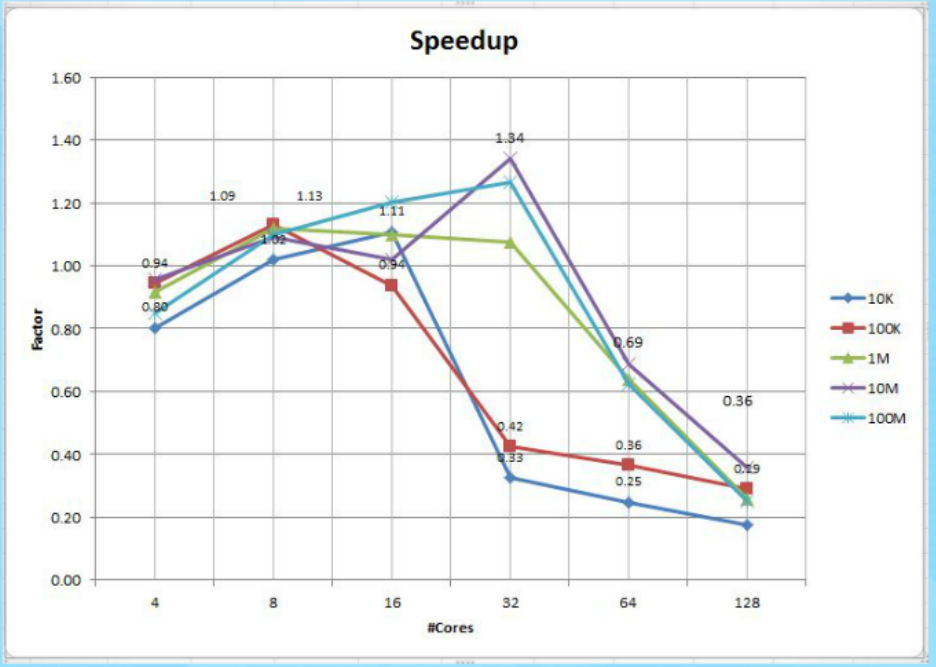

10M
100M
60.00
50.00
40.00
30.00
20.00
10.00
0.00
1200.00
1000.00
800.00
600.00
400.00
200.00
0.00
4
8
16
32
64
128
18.69
280.22
16.42
217.01
17.52
198.16
13.37
186.35
25.99
382.41
50.20
952.73


Speedup
1.60
1.40
1.20
1.00
0.80
0.60
0.40
0.20
0.00
Factor
4
8
16
32
64
128
#Cores
1.09
1.13
1.11
1.34
1.02
0.94
0.94
0.80
0.69
0.36
0.42
0.36
0.33
0.29
0.25
10K
100K
1M
10M
100M


Efficiency
Performance (%)
#Cores
50.00
45.00
40.00
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
4
8
16
32
64
128
45.64
44.09
42.85
36.99
23.31
17.74
15.90
15.00
13.56
10.40
9.06
8.72
8.14
3.78
3.21
10K
100K
1M
10M
100M

Conclusions

- In engineering contexts, efficiency is more often used for graphs than speedup, since all of the area in the graph is useful (whereas in a speedup curve 1/2 of the space is wasted)
- it is easy to see how well parallelization is working
- there is no need to plot a "perfect speedup" line
- In marketing contexts, speedup curves are more often used, largely because they go up and to the right and thus appear better to the less-informed.



Future

- Complete divide and conquer using hyperquicksort
- Implement pseudo EM like K-means
- Use Slurm effectively

s Work & nces

- Algorithms: Sequential and parallel, 2nd Ed; Russ Miller & Laurence Boxer
- Parallel algorithms in geometry, Goodrich
- Sequential and parallel algorithms for k closest pair problem; Lenhof, Smid
- A survey of Parallel computational geometry algorithms; Hehne, Sack
- An optimized divide and conquer algorithm for closest pair problem in the planar case; Pereira, Lobo
- Geometric Algorithms; Robert Sedgewick, Princeton University
- Closest Pair (PPT); Subhash Suri, UCSB
- Parallel Computing 101; Stout, Jablonowski