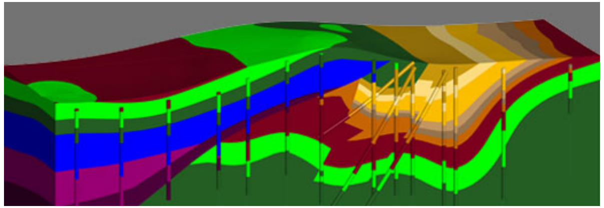
Large-scale 3D geospatial processing made possible

Lucas C. Villa Real, Bruno Silva, Dikran S Meliksetian, Kaique Sacchi
IBM Research

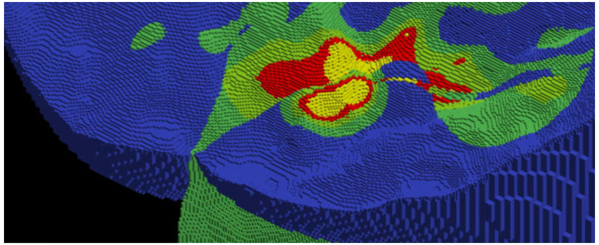
Spatial data in the mining domain

Drill holes (line geometries)
Minerals (Au, Cu, etc)
Lithology (granite, pyrite, etc)
Visible alteration
Geological structure
Gold grade
...

Geological models (shapes)



Resource models (blocks)

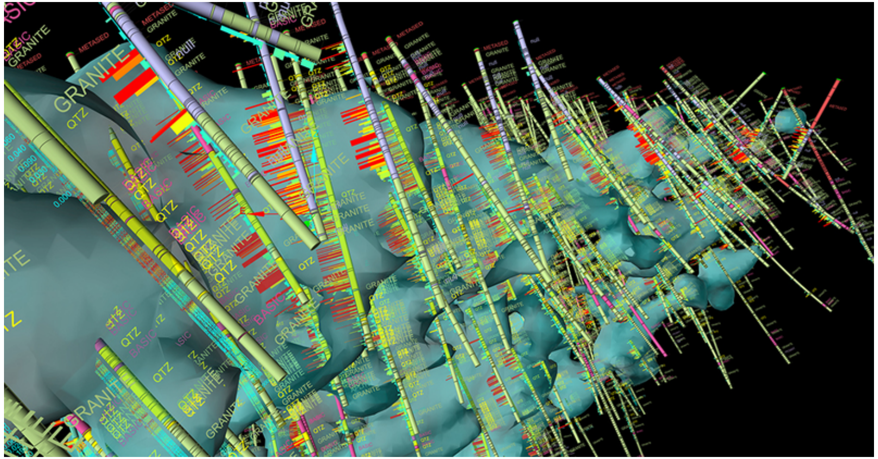


Large-scale data and spatial queries

Distance-based
"Get the average ore grade of all block models in a zone no farther than 300 meters from this given drill hole"

Intersection-based
"Get a list of all drill hole *segments* that intersect with ore shapes"

Volume-based
"Get a list of shapes associated with copper and whose volume is greater than *M*"



Numbers from real-world databases
- 16M blocks
- 30k drill holes
- 1.2M drill hole segments
- Shapes with > 100k triangles

3D Spatial Databases (SQL/MM and OGC)

Few options
- PostgreSQL (PostGIS)
- Oracle Database (Oracle Spatial)

Scalability issues
- Brute-force algorithms
- Poor support for parallel queries
- Queries may take SEVERAL DAYS to run
- Severe number of function calls

Tuning
- Execution cost of spatial functions
- Statistics: average geometry size / table
- Number of parallel workers
- Cache size

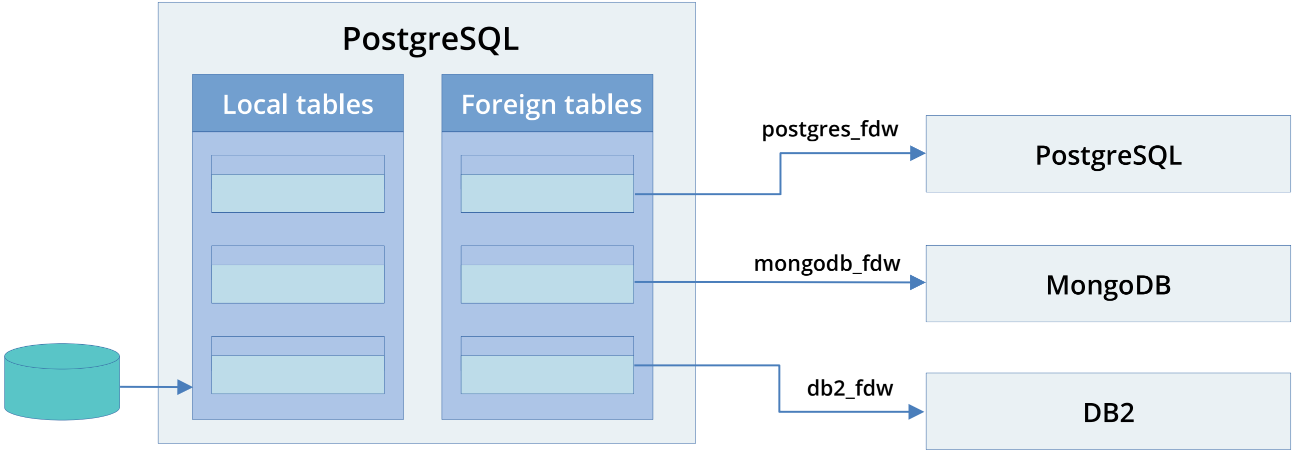
```
SELECT st_3ddistance(a.geom, b.geom) FROM lines a, shapes b
```

Input Geometries		
Line 1	Shape 1	st_3ddistance(Line 1, Shape 1)
Line 2	Shape 2	st_3ddistance(Line 2, Shape 2)
Line 3	Shape 3	st_3ddistance(Line 3, Shape 3)
Line 4	Shape 4	st_3ddistance(Line 4, Shape 4)
Line 5	Shape 5	st_3ddistance(Line 5, Shape 5)
…	…	…
Line N-1	Shape M-1	st_3ddistance(Line N-1, Shape M-1)
Line N	Shape M	st_3ddistance(Line N, Shape M)

SQL/MED: Foreign Data Wrappers

Federated server architecture
- Server receives and decomposes a query
- Query segments are dispatched to execute on remote servers
- Results are assembled by the server

Idea
What if we extract spatial elements from the query and dispatch them to a remote server that processes them on a GPU?



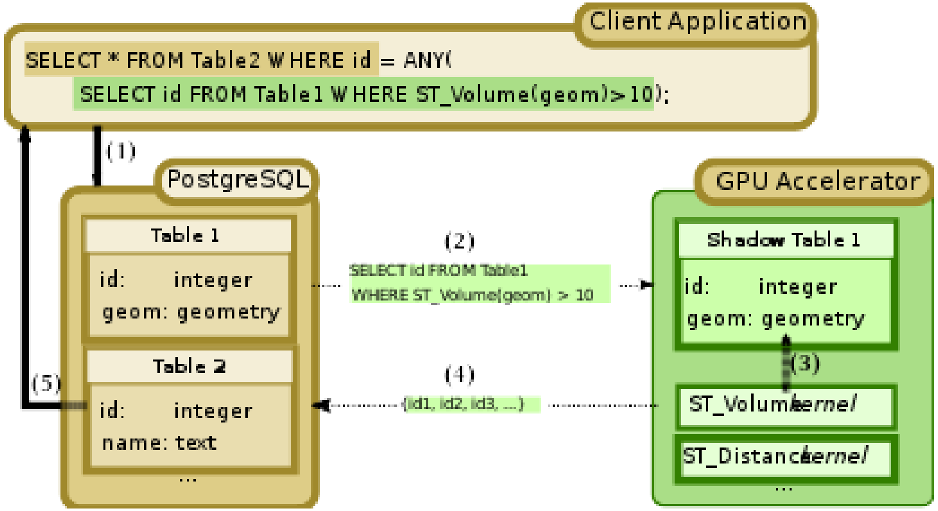
Presenting Lumic-GIS

Accelerated spatial queries using GPUs

- Lumic-GIS disguises as a PostgreSQL server

- PostgreSQL is configured to split and forward calls to PostGIS API to Lumic-GIS

- Spatial queries execute on the GPU



Details

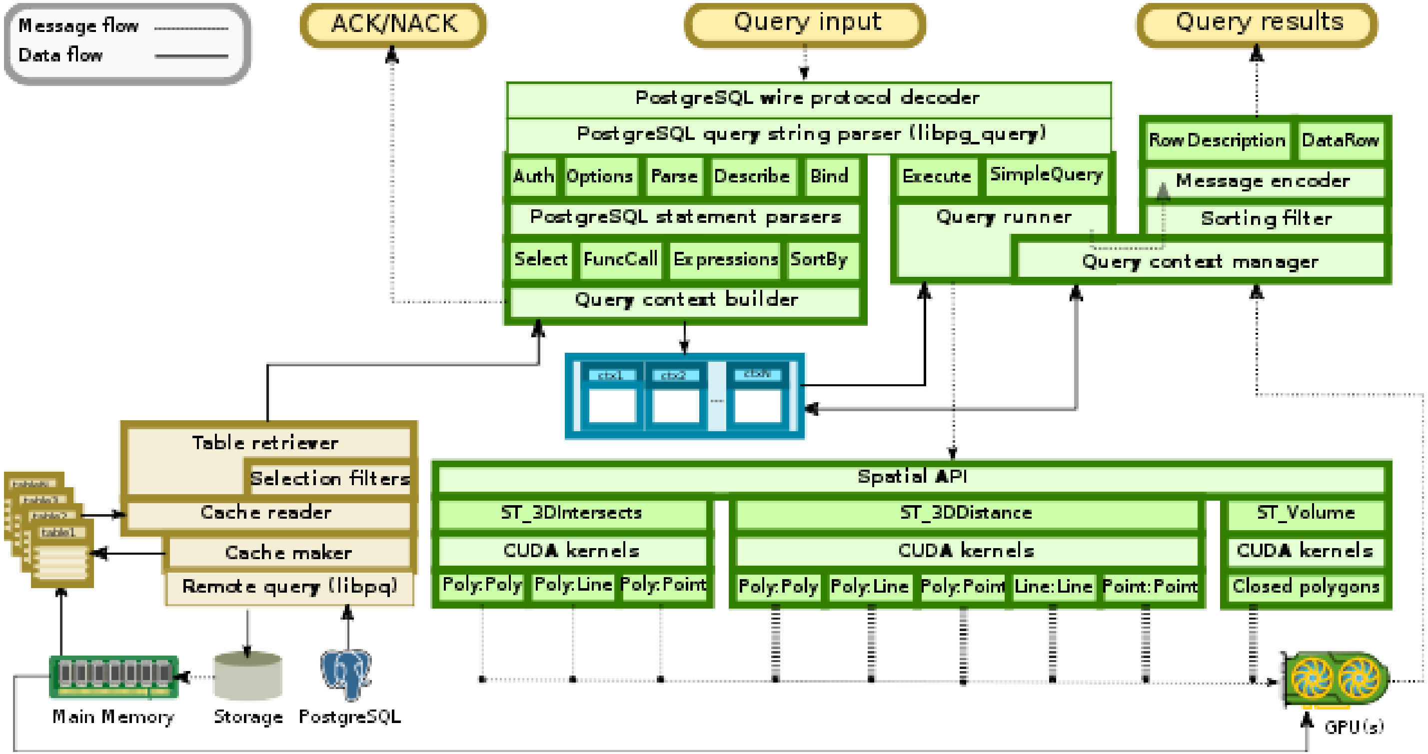
No modifications to PostgreSQL installation

- `postgres_fdw` ships by default with PostgreSQL

Geometry data is kept in memory

A single function call to the kernel is needed

The architecture of Lumic-GIS

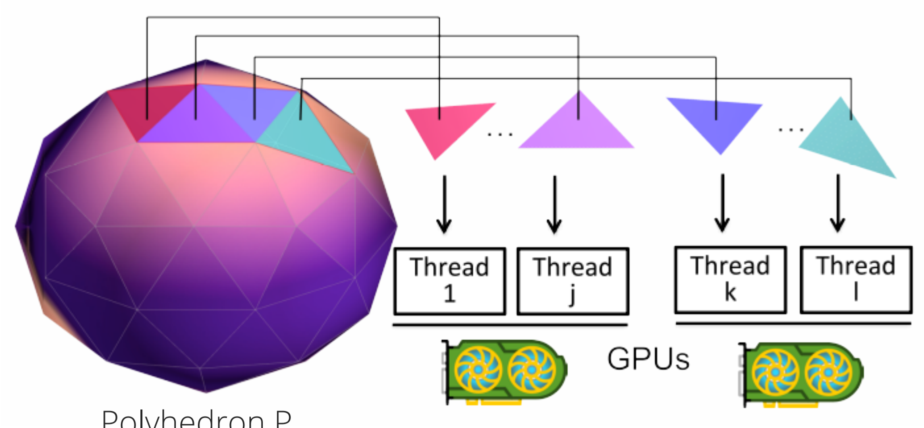


Parallelism
- CUDA streams
- OpenMP threads

Memory management
- CUDA unified memory
- Reuse of memory buffers

GPU allocation
- Given *P* processors and *M* GPUs, we allocate *S* = *P*/*M* CUDA streams per GPU

CUDA kernel: Volume



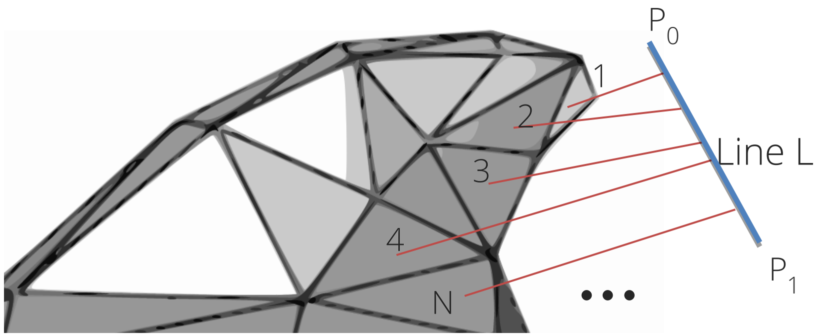
Polyhedron P

Based on the divergence theorem:

$$V = \int_P 1 = \frac{1}{6}\sum_{i=0}^{N-1} a_i \cdot \hat{n}_i$$

We evaluate the flux across each face to get the volume

CUDA kernel: 3D Distance



$$L(t) = P_0 + t\vec{d}, 0 \leq t \leq 1$$

$$T(u,v) = V_0 + u\vec{e_0} + v\vec{e_1}$$

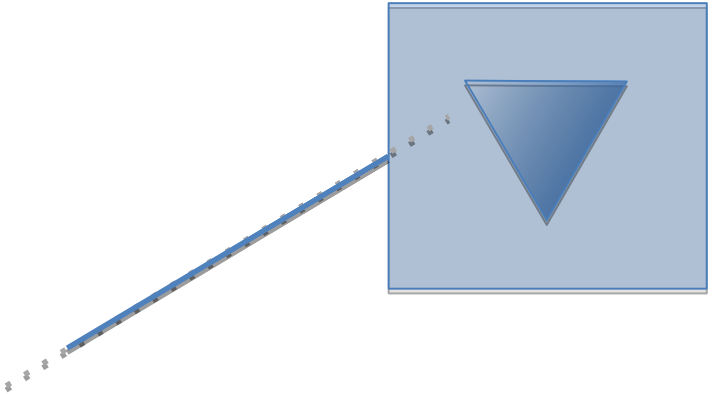
$$Q(u,v,t) = |T(u,v) - L(t)|^2$$

1. Define the triangle as a vector T
2. Define the line as a vector L
3. The minimum distance between T and L is given by the squared distance Q = $(T–L)^2$

Benefits

- No discretization of the line segment
- Embarassingly parallel

CUDA kernel: 3D Intersection



- Same parametric representations as before
- Same face decomposition approach
- We intersect the line segment with the **plane** containing the triangular face
- We pick the intersecting point and test if it is **within** the triangle

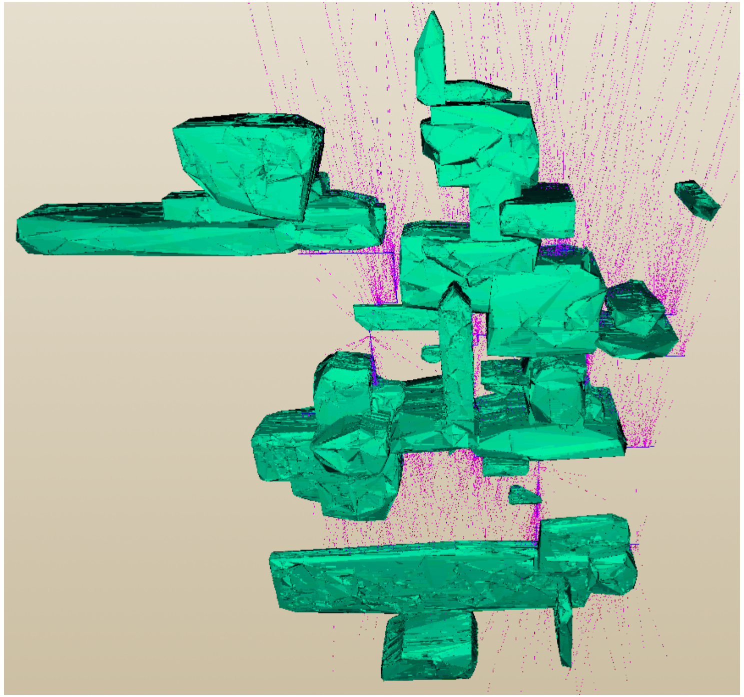
Performance evaluation

Queries
- **Distance** of drills to areas of interest
- **Intersection** of drill holes with geological shapes
- **Volume** of a geological shape

Synthetic dataset
- 7,846 drill holes
- 228,772 drill hole segments
- 71 geological shapes

Hardware
- Two NVIDIA Tesla V100 GPU cards
- Intel E5-2620 v4 / 32 cores, 768 GB of RAM, 1.2 TB of SSD

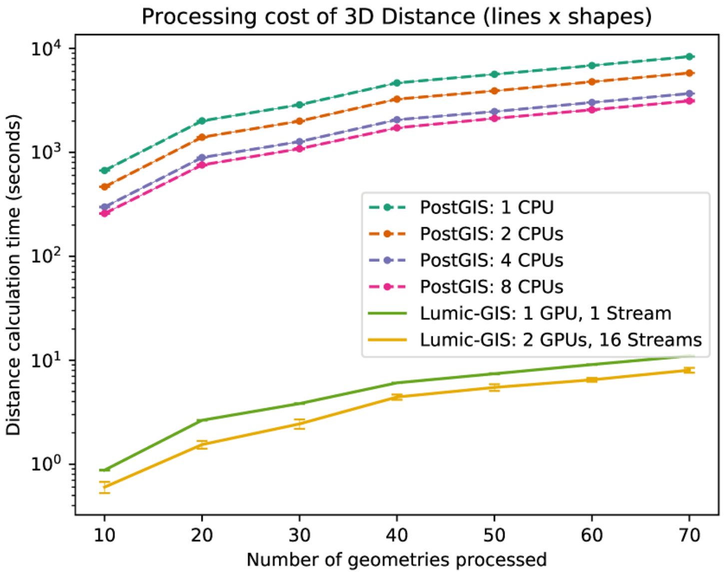
Software
- PostgreSQL 12 beta1, cache set to 50 GB
- PostGIS 3.0.0 alpha1
- Enforced use of parallel processing
- Modified PostGIS function cost estimates

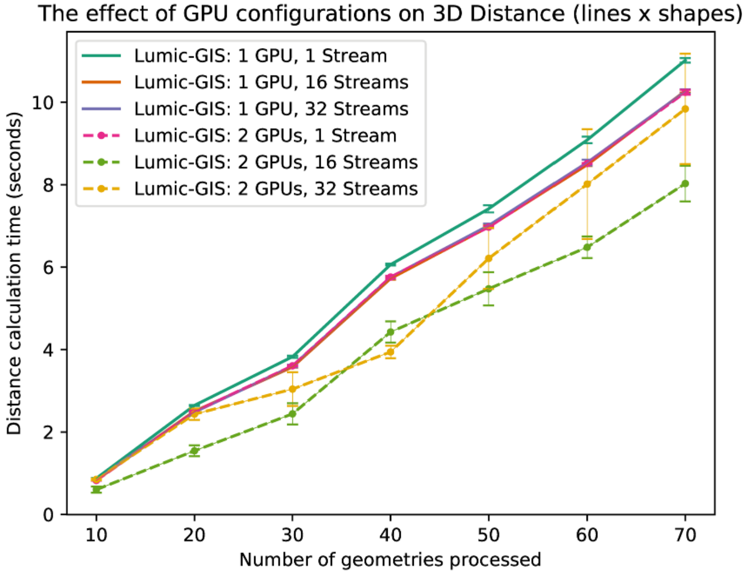


https://github.com/lucasvr/synthetic-mine-maker

3D Distance (7,846 lines x *N* shapes)

- Largest set computed: 557k pairs of geometries
- PostGIS: **2h20min** (worst), **51 min** (best)
- Lumic-GIS: **11 sec** (worst), **7 sec** (best)
- Lumic-GIS is **1300x faster** than PostGIS
- *Only 5 workers planned and launched by PostgreSQL*





3D Distance (7,846 lines x 7,846 lines)

Use case scenario

- Identify drill holes that are next to drilled segments rich in certain minerals

Notes

- PostgreSQL uses parallel workers to SCAN the tables...
- ... then joins the results with a single-processor call to `ST_3DDistance`
- Best recorded speedup of Lumic-GIS over PostGIS: **61x**

Configuration		Query time (sec)
PostGIS	8 CPUs	718.9 ± 0.2
	4 CPUs	721.2 ± 2.2
	2 CPUs	721.3 ± 1.5
	1 CPU	719.1 ± 0.4
Lumic-GIS	1 GPU, 1 Stream	13.9 ± 0.2
	1 GPU, 32 Streams	13.5 ± 0.4
	2 GPUs, 16 Streams	11.8 ± 0.2

3D Distance (1 shape x 1 shape)

Notes

- Hand-picked pairs of geometries with different number of triangles
- Single pair of geometries leads to a single PostgreSQL worker
- Lumic-GIS configuration: 1 GPU, 32 CUDA threads
- *It is unfeasible to compute the distance between larger shapes on PostGIS*

Number of triangles		Query time (sec)		
Shape 1	Shape 2	PostGIS	Lumic-GIS	Speedup
1,936	2,270	9.2	0.05	159x
10,447	10,656	232 ± 1	0.45	515x
18,596	21,785	852 ± 20	1.44 ± 0.01	591x
49,052	51,095	5,212 ± 82	8.80 ± 0.24	**592x**
150,571	230,681	**72,187** ± 451	**123** ± 0.24	582x

20 hours **2 minutes**

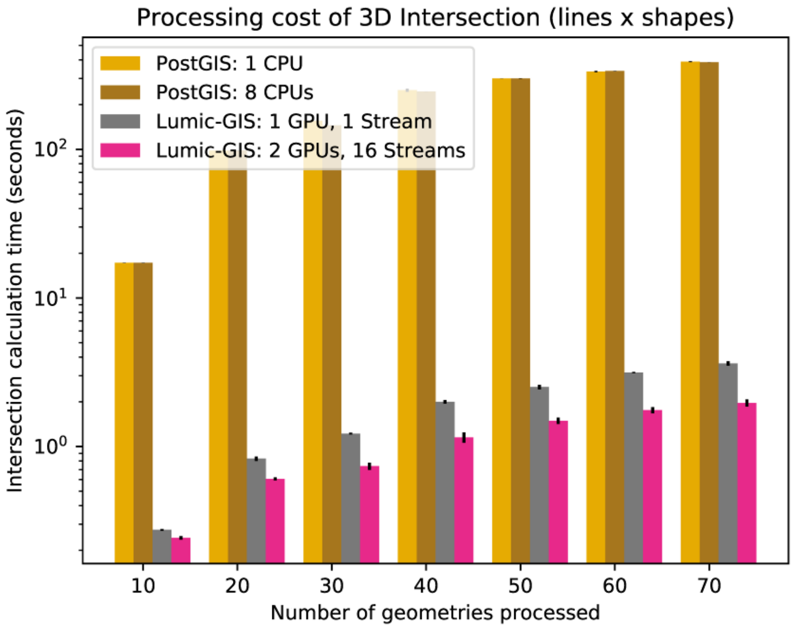
3D Intersection (7,846 lines x *N* shapes)

Notes

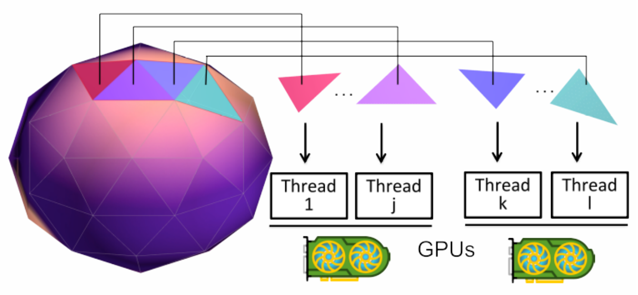
- Both implementations employ a two-phase algorithm

- *The query planner causes PostgreSQL to schedule a single worker thread*

- Lumic-GIS performs **250x faster** than PostGIS



Volume computation (from our previous paper)



Geometry
Closed shape with 500 faces

Results
PostGIS: **42 minutes** (**2530 ± 68** seconds)
Lumic-GIS: **under a second** (**0.91 ± 0.006** seconds)
Improvement of **2770x** over PostGIS

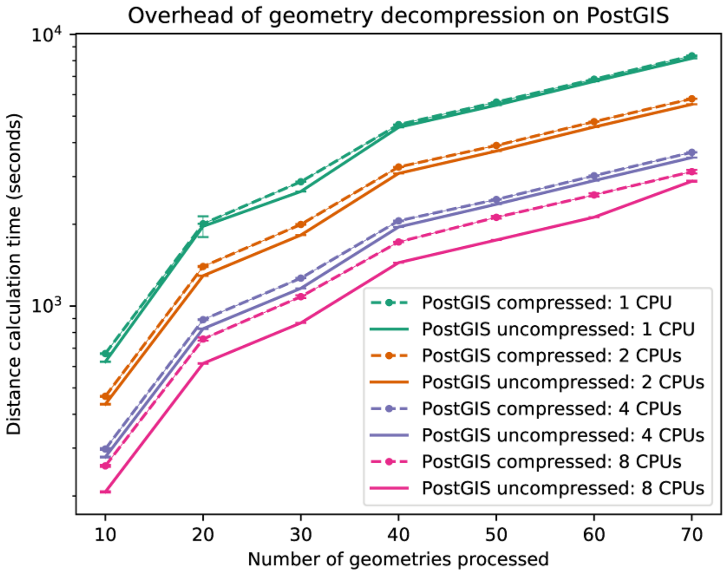
Notes
PostGIS does not split the geometry among multiple workers
Real-life databases have shapes with more than 100k faces

Overhead of geometry decompression on PostGIS

TOAST: The Oversized-Attribute Storage Technique
- PostgreSQL uses a fixed page size (usually 8kB)
- Large field values are compressed and/or broken into multiple physical rows

Queries over decompressed TOAST
- Test: 3D Distance (lines x shapes)
- Best case: Lumic-GIS **1270x faster** (N=20)
- Worst case: Lumic-GIS **690x faster** (N=30)

CPUs	Shapes	Compressed (sec)	Uncompressed (sec)	Overhead
1	10	668	624	7.1%
	20	2,009	1,966	2.1%
	30	2,869	2,642	**8.6%**
	40	4,656	4,549	2.3%
	50	5,635	5,490	2.6%
	60	6,839	6,718	1.8%
	70	8,339	8,164	2.1%
8	10	258	207	24.6%
	20	756	615	22.9%
	30	1,083	867	**24.9%**
	40	1,722	1,445	19.1%
	50	2,121	1,750	21.1%
	60	2,565	2,124	20.7%
	70	3,125	2,882	8.4%



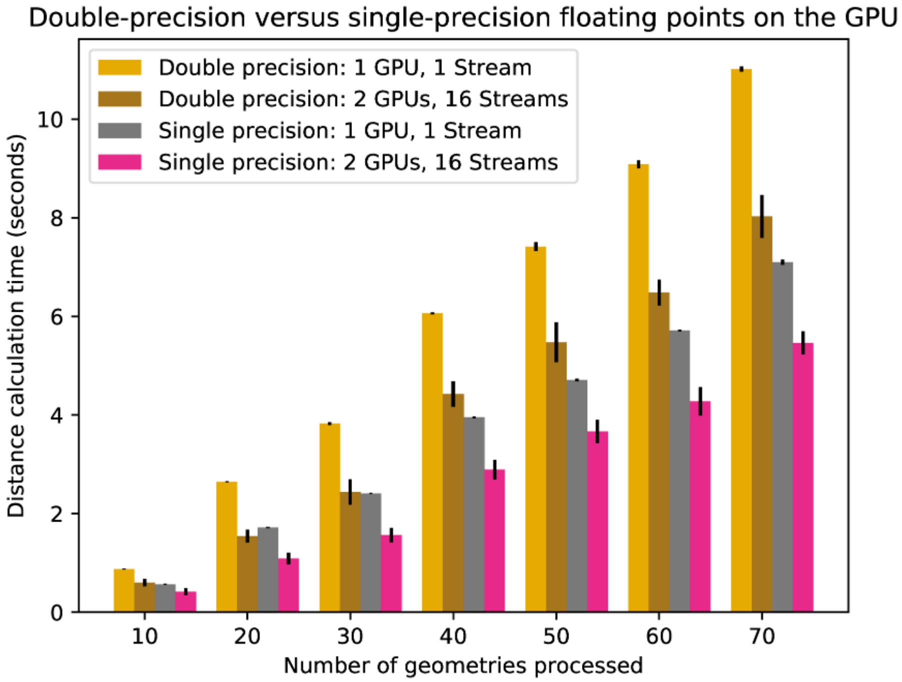
Double-precision versus Single-precision

Tradeoff between speed and accuracy is acceptable in many use cases

Lumic-GIS is built with single-precision and double-precision IEEE floats

Test: 3D Distance (lines x shapes)

We are **59% faster** with single-precision



Conclusions

- HPC can change business

  Transformation of decision making process: going from **hours** or **days** to seconds

- **Zero learning curve** for those already using PostGIS

  Seamless integration with PostgreSQL / foreign-data-wrappers

- Opportunity to improve other industries

Large-scale 3D geospatial processing made possible

Lucas C. Villa Real, Bruno Silva, Dikran S Meliksetian, Kaique Sacchi
IBM Research

IBM