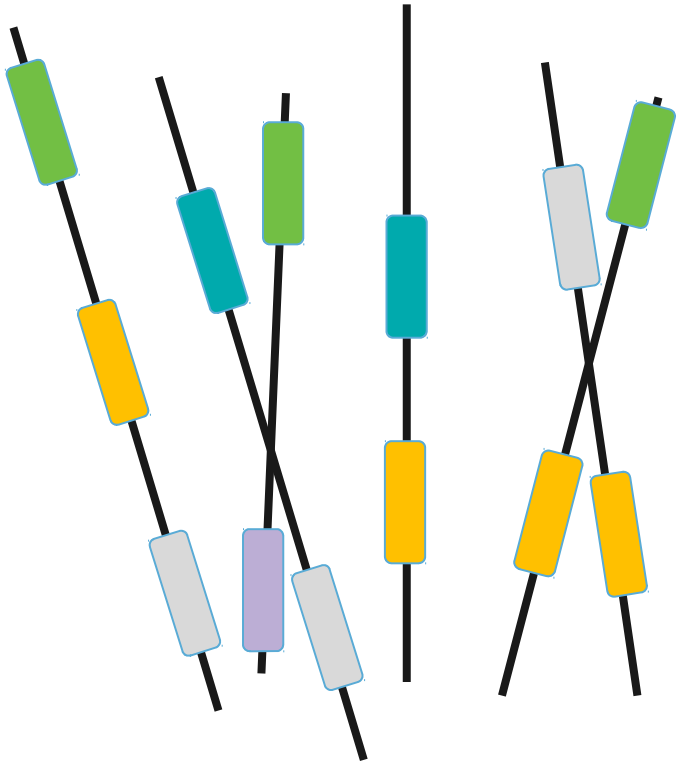
Full Speed Ahead

3D Spatial Database Acceleration with GPUs

Lucas C. Villa Real and Bruno Silva
IBM Research – Brazil

Where do we find 3D spatial data?



Case study: mining

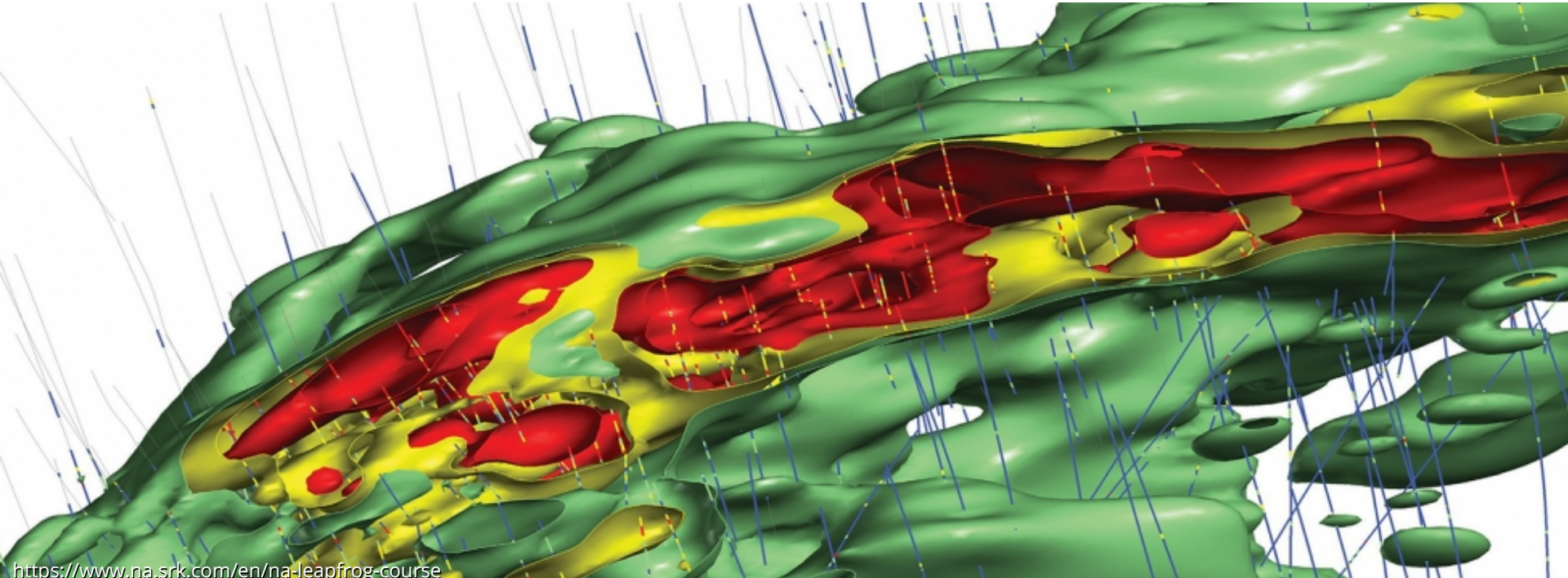
Drill hole data

1. Minerals (Au, Cu, etc)
2. Lithology (granite, pyrite, etc)
3. Visible alteration
4. Geological structure
5. Gold grade
6. ...

Geometries: 3D spatial objects
Attributes: regular data types

Common spatial operators

3D Distance between drill holes and ore bodies
3D Intersection between geological shapes and drill holes
Volume of solids



https://www.na.srk.com/en/na-leapfrog-course

Spatial data types in databases

SQL/MM and OGC (Open Geospatial Consortium) Simple Feature Access
Extends SQL to address simple 2D elements and 3D geometries (storage and access model)

- **PostGIS** (3D)
- Oracle Spatial (3D)
- IBM DB2
- Microsoft SQL Server
- MonetDB/GIS
- MySQL Spatial Extensions

...

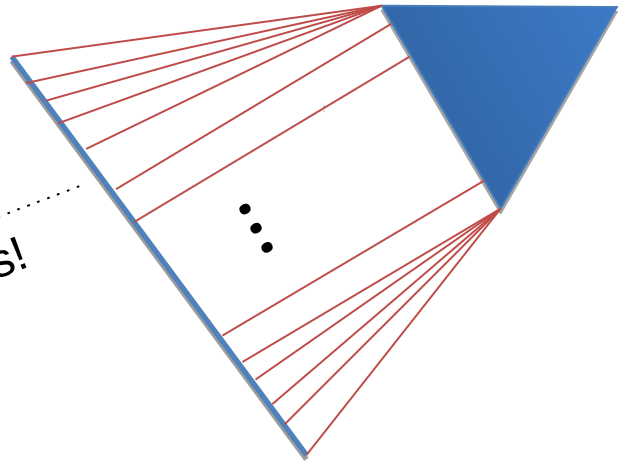
ST_Area(geom)
ST_Distance(geom1, geom2)
ST_Intersects(geom1, geom2)
...
ST_Volume(geom)
ST_3DDistance(geom1, geom2)
ST_3DIntersects(geom1, geom2)
...

Issue: complex geometries and large volumes of data slow down queries to a crawl, even in the presence of spatial indexes

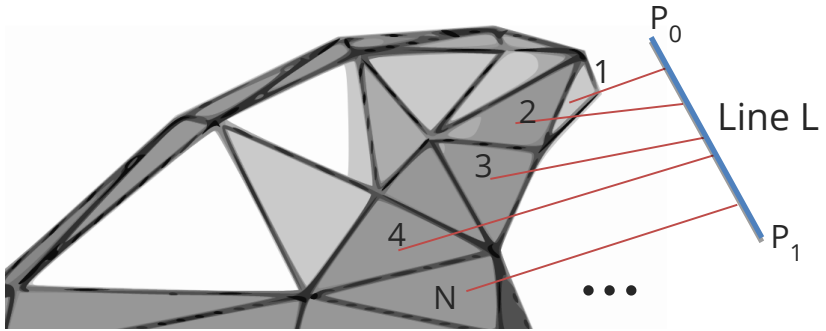
Why doesn't PostGIS scale? (ST_3DDistance)

```
/**

 * search all the segments of pointarray to see which one is closest to p
 * Returns distance between point and pointarray
 */
int
lw_dist3d_pt_ptarray(POINT3DZ *p, POINTARRAY *pa,DISTPTS3D *dl)
{
        uint32_t t;
        POINT3DZ        start, end;
        int twist = dl->twisted;

        LWDEBUG(2, "lw_dist3d_pt_ptarray is called");

        getPoint3dz_p(pa, 0, &start);

        for (t=1; t<pa->npoints; t++)
        {
                dl->twisted=twist;
                getPoint3dz_p(pa, t, &end);
                if (!lw_dist3d_pt_seg(p, &start, &end,dl)) return LW_FALSE;

                if (dl->distance<=dl->tolerance && dl->mode == DIST_MIN) return LW_TRUE; /*just a check if  the answer is already given*/
                start = end;
        }

        return LW_TRUE;
}
```

Line is discretized as points!



Implementing spatial operators on the GPU



3D Distance:

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:

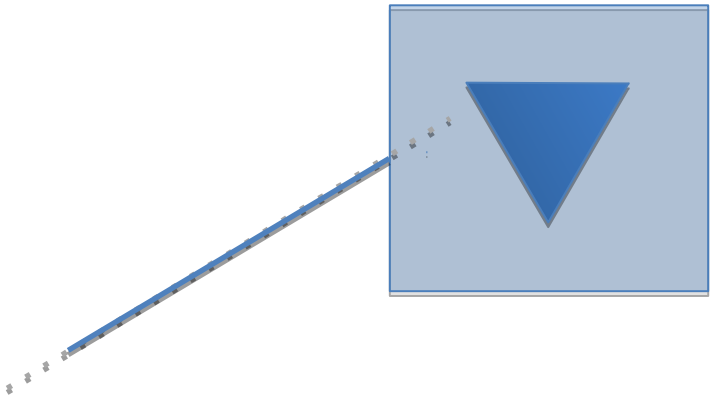
- No discretization of the line segment
- Embarassingly parallel

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

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

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

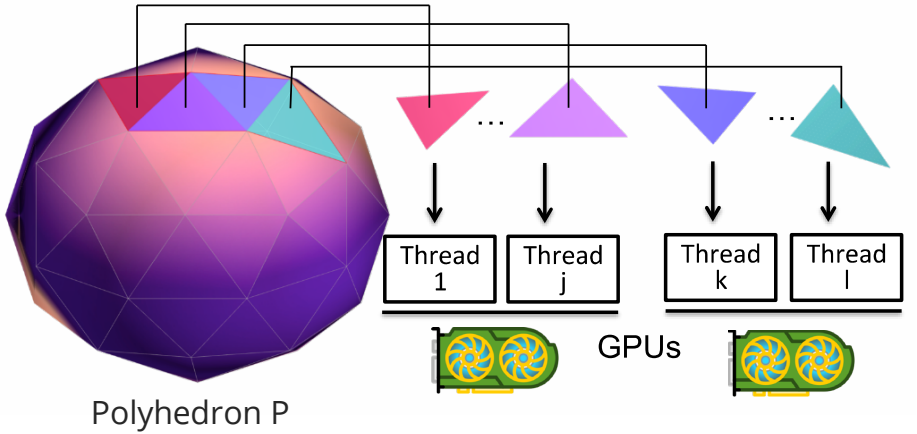
Implementing spatial operators on the GPU



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

Implementing spatial operators on the GPU



Polyhedron P

Volume:

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

Extending PostgreSQL + PostGIS

SQL/MED: Management of External Data
- Syntax extensions to SQL
- Enables access to data that lives outside the database
- SQL server can **decompose the query** and dispatch its fragments to foreign servers

PostgreSQL's Foreign Data Wrappers
- Features hundreds of extensions (orthogonal to the FUSE filesystem framework)
- Ships with the *postgres_fdw* extension to talk to foreign PostgreSQL servers

postgres_fdw:
- Sends the relevant *WHERE* clauses to the remote server
- Does not retrieve columns not needed for the current query
- Can invoke functions provided by other extensions

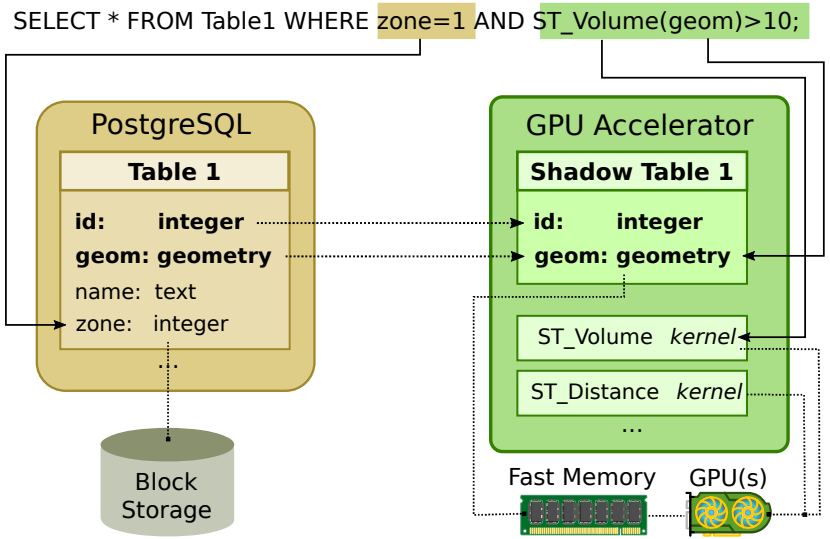
3D Spatial Acceleration Platform's Architecture

GPU Accelerator

Disguises as a PostgreSQL server:
- Takes **sub-queries** from *postgres_fdw*
- Can be accessed from popular frontends (psql)

Implements spatial operators as **GPU kernels**:
- *ST_Volume*
- *ST_3DDistance*
- *ST_3DIntersects*

Holds in-memory **shadow tables**



Performance evaluation

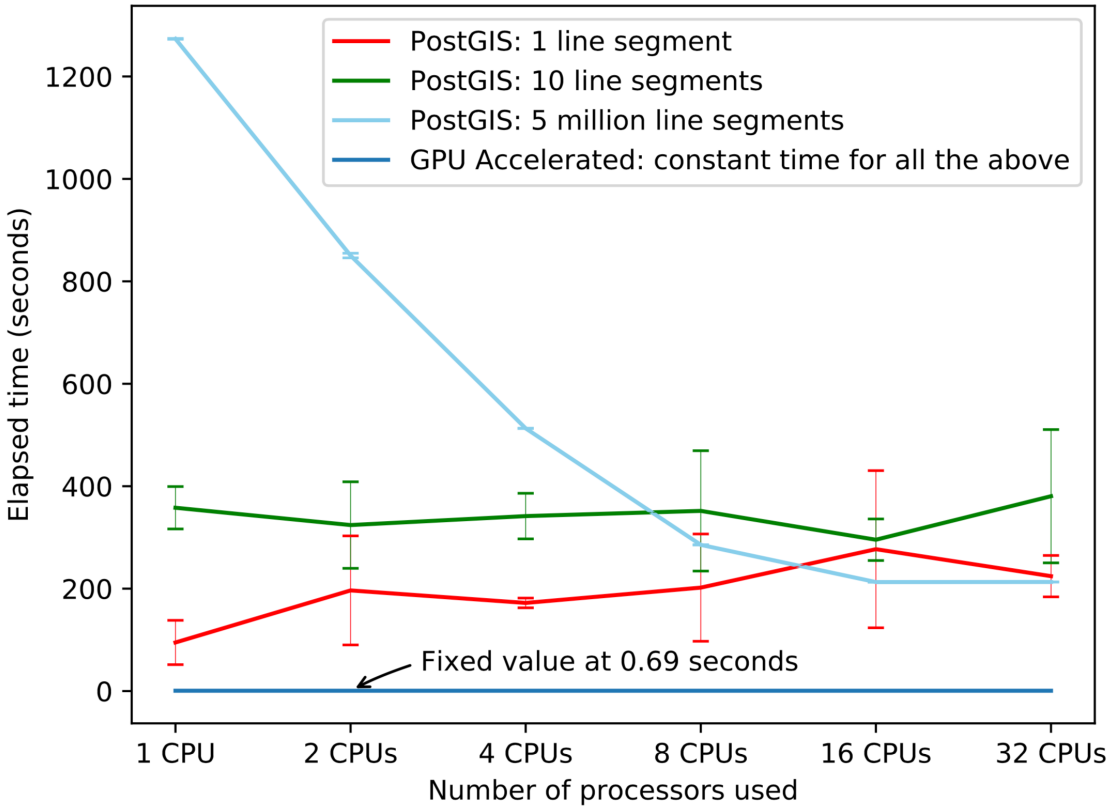
Use case: spatial operations performed by geologists on a daily basis
- Computing volume of geological shapes
- Filtering drill holes based on their distance to profitable areas of a mine
- Retrieving drill holes that intersect with certain rock types

Data set:
- A geological shape with 500 faces
- 5 million drill holes

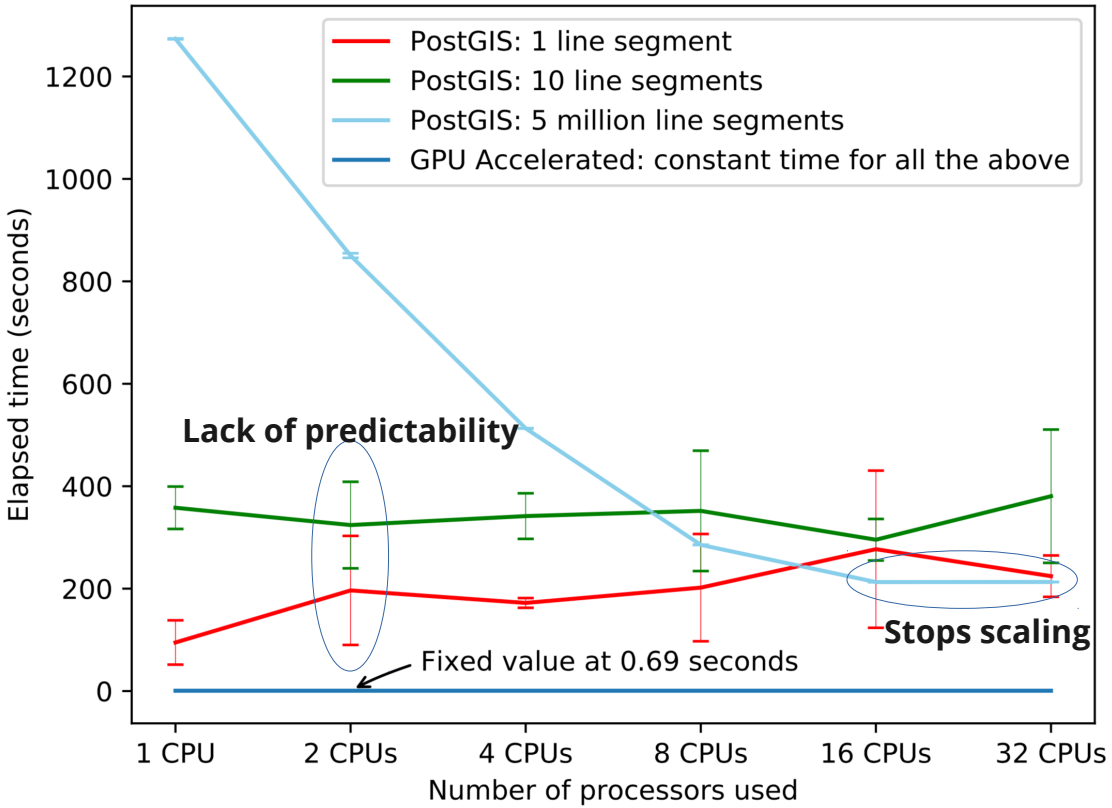
Hardware stack:
- Intel E4-2620 v4 with 16 cores, 256 GB of memory, 800 GB of SSD, one NVIDIA Tesla V100

Software stack:
- PostgreSQL 10.4, PostGIS 2.4.4, SFCGAL 1.3.2, Cuda 9.1.85
- PostgreSQL cache set to 50 GB
- Enforced use of parallel processing
- Modified the cost estimates of PostGIS functions to enable them to execute in parallel

Performance evaluation: 3D Distance



Performance evaluation: 3D Distance



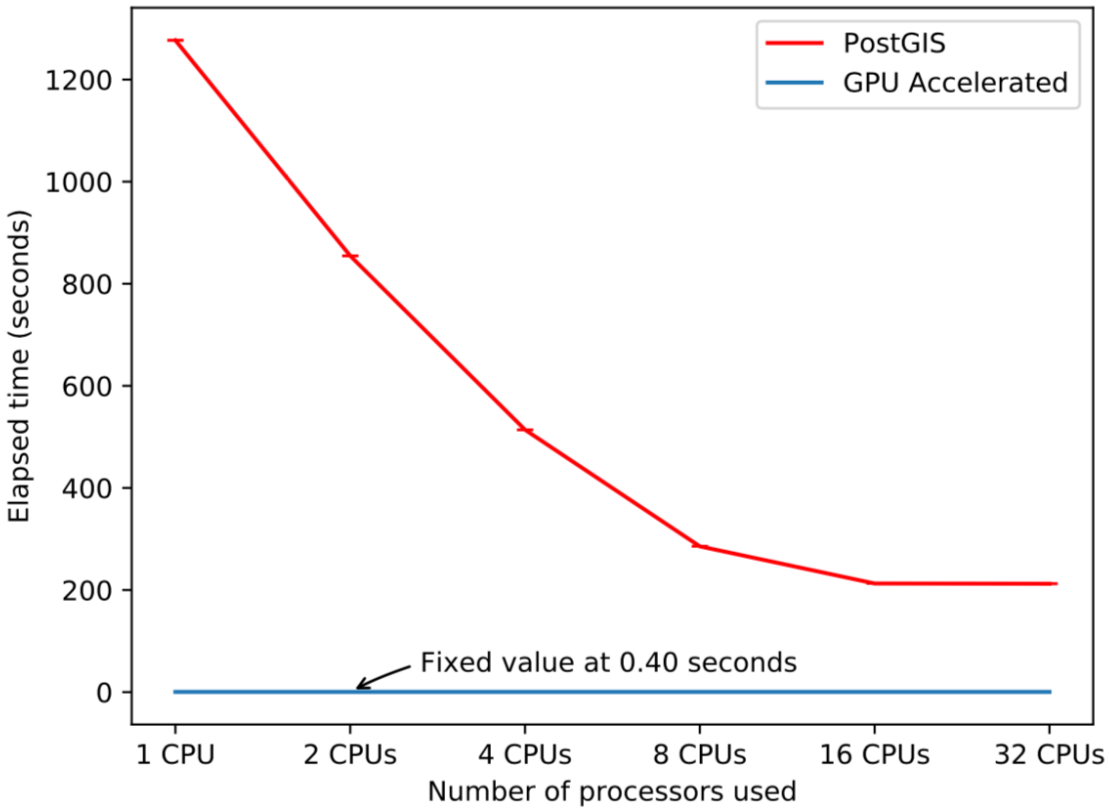
Notes:

- PostgreSQL query planner used at most 11 workers: 5 cores were left sitting idle!

- Performance gains of **6x** with PostGIS' CPU-based parallelization

- The GPU accelerator improves **1860x** over PostGIS' sequential run

Performance evaluation: 3D Intersection



Notes:

- Results show the 5 million line segments alone

- Same performance with PostGIS' CPU-based parallelization as before

- The GPU accelerator improves **3230x** over PostGIS' sequential run

Performance evaluation: Volume

- PostGIS does not split the geometry among multiple workers

- PostGIS computes the volume in **42 minutes** (**2530 ± 68** seconds)

- The GPU accelerator computes it in **0.91 ± 0.006 seconds**, improving **2770x** over PostGIS

Conclusions

1. Accelerating spatial database systems with foreign services powered by GPUs is feasible

2. Speedups as observed can change the way industries conduct their business

3. There are several research opportunities in this area, such as:

- Geometry prefetching algorithms
- Geometry caching strategies
- GPU-assisted geometry compression/decompression
- Cooperation between concurrent GPU kernels

Full Speed Ahead

3D Spatial Database Acceleration with GPUs

Lucas C. Villa Real and Bruno Silva
IBM Research – Brazil