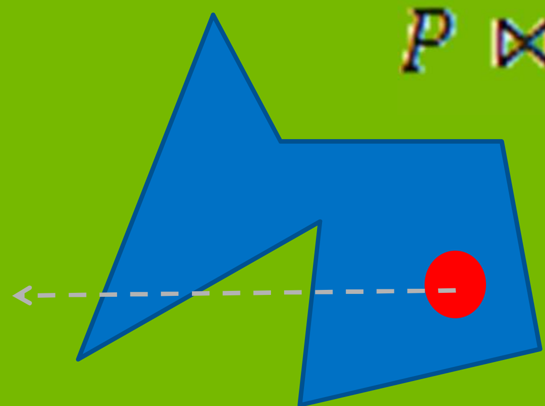
PIP (1)

Point-in-Polygon Test

- Answer whether a given point lies inside a polygon
- Fundamental operation in spatial databases and GIS
- Ray casting algorithm

$$P \bowtie_{PIP} Q$$



```
int pnpoly(int npol, float *xp, float *yp, float x, float y)
{
    int i, j, c = 0;
    for (i = 0, j = npol-1; i < npol; j = i++) {
     if ((((yp[i] <= y) && (y < yp[j])) ||
          ((yp[j] <= y) && (y < yp[i]))) &&
         (x < (xp[j] - xp[i]) * (y - yp[i]) / (yp[j] - yp[i]) + xp[i]))
       c = !c;
    }
    return c;
}
```

7 lines of C code due to W. Randolph Franklin@RPI

https://wrf.ecse.rpi.edu//Research/Short_Notes/pnpoly.html

The even-odd rule works for any closed set of polygons (polygons with multiple rings; polygons with holes…)

SELECT Point.ID, Polygon.ID
FROM Point, Polygon
WHERE ST_WITHIN(Point.geometry, Polygon.Geometry)

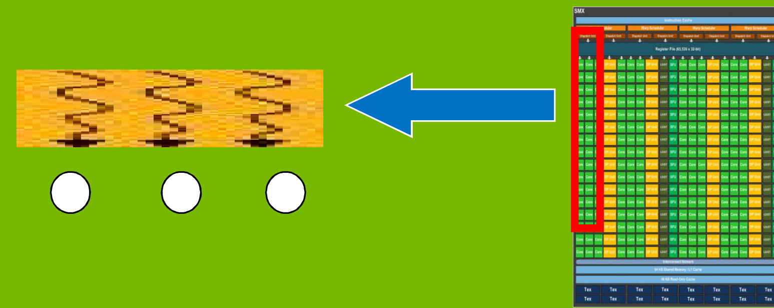
GDAL/OGR: OGRGeometry.**Contains**(OGRGeometry)

PIP (2)

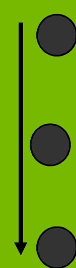
```
__global__ void gpu_pip_cuda(
int num_pnt,
const double *pnt_x, const double
*pnt_y,
int num_f, const uint *f_pos, uint*
r_pos,
double *poly_x, double *poly_y,
int* res)
```

Can be changed to
Templated type: T

Points



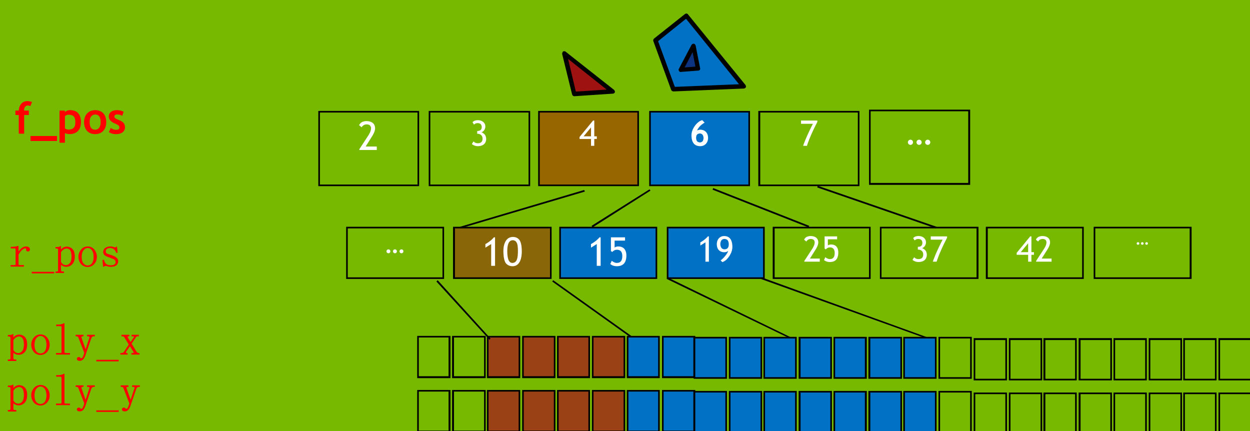
Polygon vertices



•Coalesced memory accesses

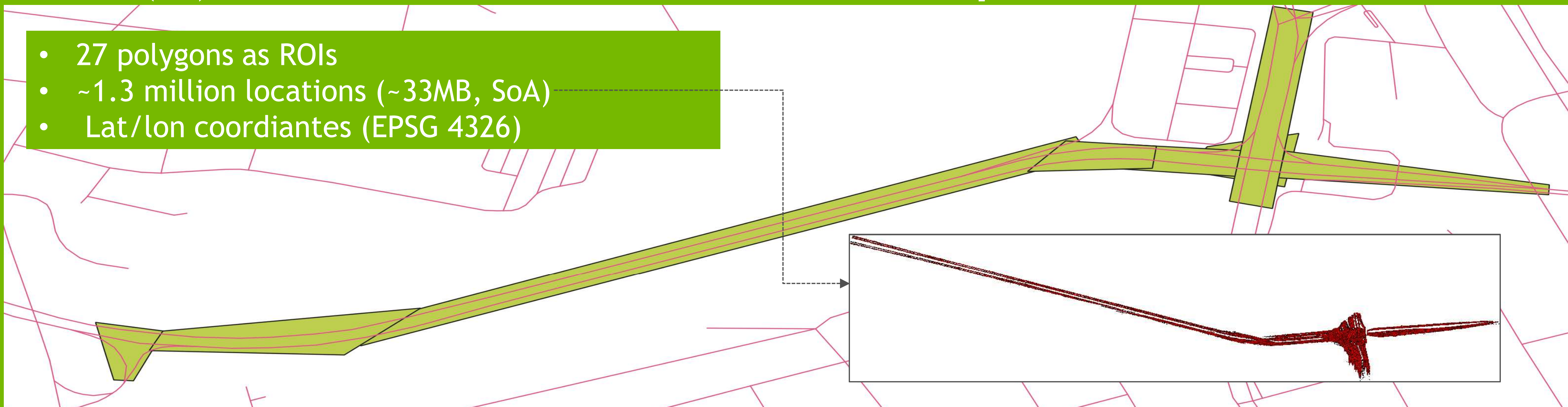
res:
- Array of polygon IDs
  - a point can only be in one polygon
- Array of bitvectors
  - a point can be in mutlple polygons

```
For num_f<8:  char*
For num_f<16: short *
For num_f<32: int *
…
```



PIP (3): Performance on IVA outputs

- 27 polygons as ROIs
- ~1.3 million locations (~33MB, SoA)
- Lat/lon coordiantes (EPSG 4326)



- PIP CUDA test code:
- 0.92 millisecond GPU kernel time
- ~3 milliseconds CPU->GPU data transfer time
- 334 milliseconds CPU time (single thread, serial)
- 72.2 milliseconds CPU time (12 threads, OpenMP)

4.4x

- GDAL C++ test code:
- 83167.6 milliseconds (single thread, serial)
- 31795.1 milliseconds (12 threads, OpenMP)

2.6x

CPU/GPU (with data transfer)=334/(0.92+3)=85X
CPU/GPU (without data transfer)=334/0.92=365X
GDAL/GPU(without data transfer)= 83167.6/0.92~10^5

Hardware:

- Intel i7-7800X CPU (6 core/12 threads)
- 32 GB CPU memory, PCI-E 3 bus
- Titan V GPU : 5120 CUDA core, 12 GB

OS/Software: Ubuntu 18.04, GCC 7.3.0, CUDA 10.0, CC 7.0

Basic spatial and temporal queries

- Spatial Window (SW) query
  - how many items (points) inside window (x1,y1,x2,y2)?
  - ./stquery hwy20 -1 "SW -90 -180 90 180“
- Temporal Point (TP) query:
  - how many items (points) **at** any of year, month, day, hour, minute, second, millisecond , or any of them combinations (OLAP type)
  - ./stquery hwy20 -1 "TP 18 3 11 11 -1 -1 -1“   -1 for “do not care”
- Temporal Window (TW) query:
  - how many items (points) between two timestamps (up to milliseconds) ?
  - ./stquery hwy20 -1 "TW 18 3 11 11 18 3 11 12"

Basic spatial and temporal queries

```
typedef thrust::pair<Time, Time> TBBox;
typedef thrust::pair<Location, Location> SBBox;

thrust::count_if(In, In + n, sw_query(SBBox box));
thrust::copy_if(In, In + n, Out, sw_query(SBBox box));

thrust::count_if(In, In + n, tp_query(Time ts));
thrust::copy_if(In, In + n, Out, tp_query(Time ts));

thrust::count_if(In, In + n, tw_query(TBBox box));
thrust::copy_if(In, In + n, Out, tw_query(TBBox box));
```

- New functors of compound queries can be developed
- Arrays for In would need be zipped together for Thrust primitives
- auto In=thrust::make_zip_iterator(thrust::make_tuple(…))

```
typedef struct Time
{
    uint y : 6;
    uint m : 4;
    uint d : 5;
    uint hh : 5;
    uint mm : 6;
    uint ss : 6;
    uint wd: 3;
    uint yd: 9;
    uint mili: 10;
    uint pid:10;
}Time;
```

- Allow millisecond
- 64 bits (well supported by both CPU and GPU – cast as int64_t)
- Allow up to 2^6=64 years with configurable starting year
- Allow look up directly without calculation (different from cuDF)

Basic spatial and temporal queries - Performance

~1.3 million locations (~33MB, SoA)

	Intel 7800X + Titan V	Intel i7-5820k + GT 730	Intel i7-5820k +GT 710
Host-to Device Bandwidth (GB/s)	11.98	3.10	1.59
Data Loading Time (hot cache) (ms)	13.54	17.23	15.71
Preparation (mem allocation and CPU-GPU data transfer) Time (ms)	5.78	13.96	26.22
Spatial Window Query (SW) ("SW -90 -180 90 180")	**0.49**	**2.43**	**4.97**
Temporal Point Query (TP) ("TP 18 3 11 11 -1 -1 -1")	**0.47**	**1.44**	**2.77**
Temporal Window Query (TW) ("TW 18 3 11 11 18 3 11 12")	**0.53**	**2.91**	**6.01**
Wrap-up (mem deallocation and potentially others) (ms)	0.42	1.70	1.75

Basic trajectory operations

Trajexplore_gpu.cu

- Read camera configuration file and populate a lookup table
- On-the-fly Lat/lon to x/y transformation

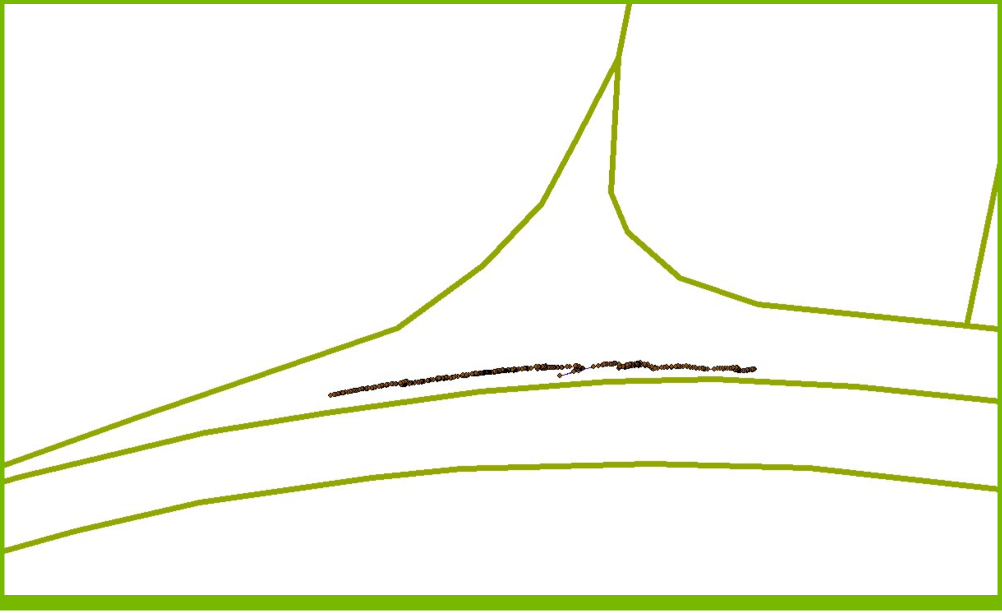
```
thrust::transform(Loc, Loc + num_rec,  coord, coord_transformation(camera_orig
```

```
struct coord_transformation : public thrust::unary_function<Location,Coord>{
      Coord operator()(Location pt){
          Coord c;
           c.x = ((origin.lon - pt.lon) * 40000.0 * cos((origin.lat + pt.lat) * M_PI / 360) / 360);
           c.y = (origin.lat - pt.lat) * 40000.0 / 360;
          return c;}
      }
```

- Generate trajectories
  - stable_sort_by_key: sort with objectid as the key and zip all fields as the value
  - reduce_by_key: count number of unique objectids (trajectories)+generating sbbox+tbbox
- Compute distance/speed
- Output as shapefile(s) to visualize in GIS (ArcGIS/QGIS)
  - GDAL APIs (CPU): create dataset/→create layer→create fields+create feature→add points

Basic trajectory operations

- Temporal resolution of IVA derived location data (~10fps) is much finer than GPS data (seconds to minutes per record) ; spatial resolution seems to be high
- More suitable for map-matching and lane-level clustering
- Potentially facilitate developing simper and GPU-friendly algorithms for map-matching and trajectory clustering



- Visualizing trajectory with objctid=13568 from schema_HWY_20_AND_LOCUST-filtered.json
- Aligns well with road segment
  - NN distance based map matching seems to work
  - average distance to other road segments are much larger)

Basic trajectory operations

- **1.3 million points→ 52862 trajectories ;** Three steps:
  - Sorting based on objectid+timestamp [in_place] (stable_sort_by_key)
  - Counting unique objectid →#traj (unique)
  - Location transformation : (lon/lat)→(x,y) (transform)
  - Group by objectid →counts+time range (reduce_by_key)

	GPU (Thrust)	CPU1(Thrust) (host - serial)	CPU2 (Thrust) (OpenMP-12 threads)	CPU2 (__gnu_paralle) (OpenMP-12 threads
initialization	CPU→GPU 6.453		CPU→Device 19.346	
Transform	0.014	19.301	2.242	17.185
Data Re-Org				4.187
Sort	9.210	253.035	172.401	41.660
Unique	2.771	2.174	2.305	46.163
Grouping	1.810	68.726	42.276	3.982
Tot	13.805	343.236	219.224	113.177

Basic trajectory operations: Compute Distance/Speed

```
__global__ void kernel_ds(int num_traj,const Coord * coord,const
           Time *time, const int *len,const int *pos, float* dis, float *sp)
{
           int pid=blockIdx.x*blockDim.x+threadIdx.x;
           int bp=(pid==0)?0:pos[pid-1], ep=pos[pid]-1;
           float td=(time[ep].yd-time[bp].yd)*86400;
           td+=(time[ep].hh-time[bp].hh)*3600;
           td+=(time[ep].mm-time[bp].mm)*60;
           td+=(time[ep].ss-time[bp].ss);
           td+=(time[ep].ms-time[bp].ms)/(float)1000;
           //handle invalid trajectories…
           float ds=0;
           for(int i=0;i<len[pid]-1;i++)
           {
                      float dt= coord[bp+i+1] and coord[bp+i]
                      ds+=sqrt(dt);

                      dis[pid]=ds*1000; // km to m
                      sp[pid]=ds*1000/td; // m/s
           }
}
```

Much more parallelization friendly and much more efficient than **difftime (time_t, time_t**) on CPUs

GPU kernel time: **0.859** ms
CPU Time: **4.172** *
CPU Time using **difftime: 119.397***

Speedup: 4.86X/139X

*OpenMP (with 12 threads) does not help on any of the two CPU implementations)