In-place computing on PostgreSQL
~SQL as a shortcut of GPGPU~

NEC Business Creation Division
The PG-Strom Project
KaiGai Kohei <kaigai@ak.jp.nec.com>
Subject of my talk

Where is better location to compute?

Local Computing

Knowledge

Location of the source data

DATA EXPORT

In-place Computing

Knowledge

Location of the source data

GPU Technology Conference 2016 - In-place computing on PostgreSQL
about In-place Computing

Advantage
- Less amount of data transfer
- Utilization of server grade hardware capability
- Analytics towards the latest dataset

Disadvantage
- Server needs to have sufficient computing resource.
  ➔ distributed system is also an option, like Hadoop
- Server software needs to be designed for data processing.

In-place Computing

Location of the source data
Who Am I?

name: KaiGai Kohei
mission: Development of GPU acceleration feature (PG-Strom) for PostgreSQL, and its business related stuff

company: NEC

background:
- about 10 years experiences in PostgreSQL developer community
- various contributions to the PostgreSQL core:
  - Security-Enhanced PostgreSQL (sepgsql), Security Barrier View, Writable FDW, Remote Join, Custom Scan/Join interface, ...etc...
- Joined to GPU/CUDA development since 2012
DBMS ≠ Database, as literal

DBMS = Database Management System

Software to “manage” database

- Optimized to store/reference fraction of the dataset scattered on the storage system (e.g. indexing, ...)
- Not designed for massive calculation as primary purpose
Dogma of data processing close to the data location

Database Management + Processing Capability

DBMS is the best software platform for data processing, if we could add reasonable computing capability.

→ **PG-Strom** is an extension of PostgreSQL, to adds data processing capability using GPU.
Overview of PG-Strom

Two core ideas
- GPU Native Code Generation on the fly
- Fully utilization of PostgreSQL infrastructure

Advantages
- CPU intensive SQL: OLAP, Reporting, Batch, Calculation, ...
- Continuity of application and user’s skill
- Inexpensive solution using OSS + GPU
Core ideas (1/2) – Native GPU code generation on the fly

QUERY: SELECT cat, count(*), avg(x) FROM t0
WHERE x between y and y + 20.0 GROUP BY cat;

E.g) Mathematical formula in SQL into CUDA code on the fly

```
// STATIC_FUNCTION(bool)
static inline bool gpupreagg_qual_eval(kern_context *kcxt,
                                     kern_data_store *kds,
                                     size_t kds_index)
{
    pg_float8_t KPARAM_1 = pg_float8_param(kcxt,1);
    pg_float8_t KVAR_3 = pg_float8_vref(kds,kcxt,2,kds_index);
    pg_float8_t KVAR_4 = pg_float8_vref(kds,kcxt,3,kds_index);

    return EVAL((pgfn_float8ge(kcxt, KVAR_3, KVAR_4) &&
                  pgfn_float8le(kcxt, KVAR_3, KVAR_4, KPARAM_1))));
}
```

Reference to input data
SQL expression in CUDA source code

Just-in-time Compile
Run-time Compiler (nvrtc)
Parallel Execution
Core ideas (2/2) – Fully Utilization of PostgreSQL Infrastructure

**No change of SQL syntax**
SQL parser break down the given query string into internal parse-tree structure. PG-Strom references only internal tree and don’t required any special syntax enhancement.

**No patch of PostgreSQL**
Custom plan interface allows to inject alternative query execution paths, then executes them if estimated cost is reasonable then built-in implementation. PG-Strom can be installed on the PostgreSQL now we’re operating.

**No change of Data Schema**
PG-Strom never requires special storage format more than what PostgreSQL currently has. It eliminates necessity of application modification and extra administrations.
Supported Features

① **GpuProjection**: Query with calculations

SELECT c.category,
max((x.p1 - y.p1)^2 + ... + (x.p10 - y.p10)^2) dist
FROM sample_data x
JOIN sample_data y ON x.id < y.id
JOIN category c ON x.cat_id = c.cat_id
WHERE x.date BETWEEN '2010-01-01'::date AND NOW()::date
AND x.quality > 0.8
GROUP BY c.category;

② **GpuJoin**: N-way parallel Join

③ **GpuScan**: Scan with WHERE clause

④ **GpuPreAgg**: Aggregation/GROUP BY

PG-Strom suggest GPU accelerated plan if above workloads.
Optimizer will choose cheaper execution plan if any.
**QUERY:**

SELECT cat, AVG(x) FROM t0 NATURAL JOIN t1 [, ...] GROUP BY cat;
✓ t0: 100M rows, t1~t10: 100K rows for each, all the data was preloaded.

**Environment:**
- PostgreSQL v9.5b + PG-Strom (4-Feb), CUDA 7.5 + CentOS 7(x86_64)
- CPU: Xeon E5-2670v3, RAM: 384GB, GPU: NVIDIA GTX980 (2048cores)
about DBT-3 benchmark:
- Simulate various reporting queries on large retail industry.

Note:
- Same test environment with last page
- Light-blue-colored bar involves no GPU execution due to query optimization
- Q21 in PostgreSQL didn’t finish 2Hr, so I tuned the parameter by hand
### Workloads characteristics

<table>
<thead>
<tr>
<th></th>
<th><strong>OLTP</strong></th>
<th><strong>OLAP</strong></th>
<th><strong>In-place Computing</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I/O</strong></td>
<td>Large (read+write hybrid)</td>
<td>Small (mostly read)</td>
<td>Tiny (mostly read)</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Small, sequential</td>
<td>Large, parallelizable</td>
<td>Large, parallelizable</td>
</tr>
<tr>
<td><strong>Typical SQL</strong></td>
<td>index accesses, UPDATE, INSERT,...</td>
<td>JOIN, GROUP BY, SORTING, ...</td>
<td>Numerical Calculations, ...</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>Massive</td>
<td>Small</td>
<td>Tiny</td>
</tr>
<tr>
<td><strong>KPI</strong></td>
<td>Latency</td>
<td>Throughput</td>
<td>Latency + Throughput</td>
</tr>
<tr>
<td><strong>For PG-Strom</strong></td>
<td>Not help</td>
<td>Originally Designed for</td>
<td>New Challenge</td>
</tr>
</tbody>
</table>
Test Scenario of In-place computing

**Local Computing**

**PostgreSQL**

**PostgreSQL + PG-Strom**

**Result**

**DATA EXPORT**

Hotspot: NW+CPU

Hotspot: Database

R as user interface
First Challenge – F-Test

F-Test

- A method for analysis of variance
- Checks the ratio of sample variance of two groups
- If same variance, its ratio shall follow F-distribution at statistically reasonable area.

→ Sample variances, # of items are needed.

Data Set

- Heterogeneity Activity Recognition Data Set
  - http://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition
- records of accelerometer and gyroscope of mobile device
- includes anonymized user-id, model of device, user’s act
- 43.9M rows, 3.3GB in total
# Obtain dataset from database
conn <- dbConnect(PostgreSQL())
sql <- "SELECT model, x, y, z
        FROM phones_accelerometer"
r <- dbGetQuery(conn, sql)
dbDisconnect(conn)
# Pickup ‘s3mini’ items
cond <- r[['model']] == 's3mini'
s3mini <- r[cond,]
# Pickup ‘nexus4’ items
cond <- r[['model']] == 'nexus4'
nexus4 <- r[cond,]
# Run F-test on two variances
result <- var.test(s3mini[['x']]+s3mini[['y']]+s3mini[['z']],
                   nexus4[['x']]+nexus4[['y']]+nexus4[['z']])

# Send query to get variances and number of items
conn <- dbConnect(PostgreSQL())
sql <- "SELECT count(*), variance(x+y+z) var
        FROM phones_accelerometer
        GROUP BY model
        HAVING model = 'nexus4' OR model = 's3mini'"
r <- dbGetQuery(conn, sql)
dbDisconnect(conn)
# Fetch the result
lbound <- qf(0.025, v[1,'count']-1, v[2,'count']-1)
ubound <- qf(0.975, v[1,'count']-1, v[2,'count']-1)
fv <- v[1, 'var'] / v[2, 'var']
# Set result
result <- c(lbound, fv, ubound)
Test results

Difference of accelerometer between ‘nexus4’ and ‘s3mini’

> `pgsql_ftest(0.05)`

[1] 0.9978336 1.0018144 1.0021688

\[ F_{lower} = 0.9978336, \quad F = 1.0018144, \quad F_{upper} = 1.0021688 \]

\[ F_{lower} \leq F \leq F_{upper} \Rightarrow \text{These two group have likely same variances.} \]

Time to run F-test for each method

<table>
<thead>
<tr>
<th>Method</th>
<th>Time to run F-test [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>by R-script</td>
<td>1.666</td>
</tr>
<tr>
<td>by PostgreSQL</td>
<td>4.09</td>
</tr>
<tr>
<td>by PG-Strom</td>
<td>1.75</td>
</tr>
</tbody>
</table>

Most of time consumption is data download from PostgreSQL database.
Adjustment of the test hypothesis

No difference in models, How about people’s act?

```sql
SELECT gt, count(*), variance(x+y+z) var
    FROM phones_accelerometer
    GROUP BY gt
    HAVING gt = 'walk' OR gt = 'bike'

> psql_ftest(0.05)
[1] 0.9979628 2.1744231 1.0020435

\( F_{lower} = 0.9979628, \quad F = 2.1744231, \quad F_{upper} = 1.0020435 \)

\( F_{upper} \ll F \) These two groups likely have different variances.
mobile=# EXPLAIN SELECT gt, count(*), variance(x+y+z) var
    FROM phones_accelerometer
    GROUP BY gt
    HAVING gt = 'walk' or gt = 'bike';

QUERY PLAN
-------------------------------------------------------------------
HashAggregate (cost=255006.21..255006.25 rows=3 width=30)
    Group Key: gt
    -> Custom Scan (GpuPreAgg) on phones_accelerometer
        (cost=10063.55..215505.32 rows=255 width=56)
        Reduction: Local + Global
        GPU Projection: index, arrival_time, creation_time, x, y, z, user_id, model, device, gt
        GPU Filter: ((gt = 'walk '::text) OR (gt = 'bike '::text))
        Kernel Source: /opt/..<snip>../pgsql_tmp_strom_118438.5.gpu
(6 rows)
% cat /opt/pgsql/kaigai/base/pgsql_tmp/pgsql_tmp_strom_118438.5.gpu

: 

STATIC_FUNCTION(bool)
gpupreagg_qual_eval(kern_context *kcxt,
kern_data_store *kds,
size_t kds_index)
{
    pg_text_t KPARAM_1 = pg_text_param(kcxt,1);
    pg_text_t KPARAM_2 = pg_text_param(kcxt,2);
    pg_text_t KVAR_10 = pg_text_vref(kds,kcxt,9,kds_index);

    return EVAL((pgfn_texteq(kcxt, KVAR_10, KPARAM_1) ||
    pgfn_texteq(kcxt, KVAR_10, KPARAM_2)));
}

: 

Equivalent to the condition in SQL:
gt = 'walk' or gt = 'bike'

constant values: ‘walk’ and ‘bike’
column ‘gt’ in this record
Next Challenge

Information Technology × Drug Discovery
Background (1/2) – Diversified Chemical Structures Library

To be avoided...

- Taking experiment on multiple but similar chemical components

Requirement

- Pick up small number of items from the database
- Each chemical structure are “different” as possible as we can

GPU Technology Conference 2016 - In-place computing on PostgreSQL
Background (2/2) – Chemical structures representation

**Molecular Quantum Numbers (MQN)**

A vector of 42 descriptor which represents characteristics of chemical structures

**List of the descriptors**

1. c (carbon) 16. csb (cyclic single bonds)
2. f (fluorine) 17. cdb (cyclic double bonds)
3. cl (chlorine) 18. ctb (cyclic triple bonds)
4. br (bromine) 19. rbc (rotatable bonds)
5. i (iodine) 20. hbam (H-bond acceptor sites)
6. s (sulfur) 21. hba (H-bond acceptor atoms)
7. p (phosphorous) 22. hbdm (H-bond donor sites)
8. an (acyclic nitrogen) 23. negc (negative charges)
9. cn (cyclic nitrogen) 24. posc (positive charges)
10. ao (acyclic oxygen) 25. asv (acyclic single valent nodes)
11. co (cyclic oxygen) 26. adv (acyclic divalent nodes)
12. hac (heavy atoms) 27. atv (acyclic trivalent nodes)
13. asb (acyclic single bonds) 28. aqv (acyclic tetravalent nodes)
14. adb (acyclic double bonds) 29. cdv (cyclic divalent nodes)
15. atb (acyclic triple bonds) 30. ctv (cyclic trivalent nodes)
31. ctv (cyclic trivalent nodes)
32. cqv (cyclic tetravalent nodes)
33. r3 (3-membered rings)
34. r4 (4-membered rings)
35. r5 (5-membered rings)
36. r6 (6-membered rings)
37. r7 (7-membered rings)
38. r8 (8-membered rings)
39. r9 (9-membered rings)
40. rg10 (10-membered rings)
41. afr (nodes shared by 2 rings)
42. bfrc (edges shared by 2 rings)

**A vector of 42 descriptors**

\[(20, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0, 0, 12, 2, 0, 6, 6, 0, 9, 9, 5, 1, 1, 0, 0, 5, 5, 3, 0, 8, 4, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0)
\]


© Institute for Theoretical Medicine, Inc, 2016
Workload – MAX-MIN Method

Pick up 2\textsuperscript{nd} item; the largest distance from the 1\textsuperscript{st}

Random selection of the first item
Workload – MAX-MIN Method

- Pick up 3rd item; the largest minimum distance
- Minimum distance from the selected items
Workload – MAX-MIN Method

Pick up 4\textsuperscript{th} item;
the largest minimum distance

minimum distance from the selected items
WITH next_item AS (  
    INSERT INTO pg_temp.subset_table (  
        SELECT r.*  
        FROM mqn r,  
            (SELECT id FROM pg_temp.dist_table  
                ORDER BY dist DESC LIMIT 1) d  
        WHERE r.id = d.id)  
    RETURNING *  
)

SELECT r.id, LEAST(d.dist, sqrt((r.c1 - n.c1)^2 +  
    (r.c2 - n.c2)^2 +  
        :  
    (r.c41 - n.c41)^2 +  
    (r.c42 - n.c42)^2)) dist  
INTO pg_temp.dist_table_new  
FROM pg_temp.dist_table d,  
    next_item n, mqn r  
WHERE r.id = d.id

Hot point of the SQL query
**Summary**

- 10M rows, 211MB in total
- Workload characteristics: $W_{I/O} \ll W_{CPU}$
- If no GPU support, calculation in RDBMS cannot be an option.
- PG-Strom recorded similar performance with download + R-script.
Conclusion

Advantages

- New usage of RDBMS, by integration of GPU capability.
- No more CSV dump to process statistical data. (also, analytics towards the latest data)
- SQL can be a way to alternate GPU programming for parallel-data processing.
- Inexpensive solution due to OSS + GPU

Future challenges

- Packaging for R, to run major analytic algorithm on database.
- More performance, than similar to local R-script.
  - Ideas) CPU+GPU hybrid parallel, SSD to GPU direct, columnar support
- Procedural Language support
Project Roadmap

**PostgreSQL 9.5**
Custom-Plan Interface

**PostgreSQL 9.6**
CPU Parallel Query Execution
Combined Aggregation
Upper Path Optimization

**PostgreSQL 9.7**
Native Columnar Storage
Native Partitioning Support

---

**PG-Strom 1.0**
The first release that support primitive data types and workloads.

**PG-Strom 1.1**
Multi-backend concurrency improvement
Extra operators & functions support

**PG-Strom 2.0**
CPU+GPU Hybrid Parallel
SSD-to-GPU Direct Support
More graceful query optimization
Enterprise grade SW quality

**PG-Strom 3.0(?)**
Columnar storage support
3rd party extensions support
Self defined CUDA function
Resources

Source Code
- https://github.com/pg-strom/devel

Documentations

Questions
- Issue tracker:
  - https://github.com/pg-strom/devel/issues
- Direct contact:
  - kaigai@ak.jp.nec.com
THANK YOU!
The PG-Strom Development Team
Challenge (1) – Cost to access datum in record

- **Symptom**: row-format is worst data structure for GPU
- **Solution**: columnar-format; planned for PostgreSQL v9.7

<table>
<thead>
<tr>
<th>HeapTuple Header</th>
<th>NULL bitmap (if has NULL)</th>
<th>OID of row (if exists)</th>
<th>1st column</th>
<th>3rd column</th>
<th>4th column</th>
</tr>
</thead>
</table>

Format of PostgreSQL Record

- No datum if NULL
- May be variable length

Challenge (2) – Less Concurrent Multiprocessing Gain

- **Symptom**: Each iteration is blocked by the client script
- **Solution**: Implement entire iteration with SQL/SQL function

**GPU**

- GPU Chunk1
- GPU Chunk2
- GPU Idle until Next Query

**CPU**

- Scan Chunk1
- Scan Chunk2
- Response to Client
- Parse & Optimize Next Query
- Scan Chunk1
- Scan Chunk2
- Response to Client
Backup: SSD-to-GPU under SQL execution

Ultimate “Near-Data Computing” than CPU/RAM

1. SSD ➞ GPU direct DMA
2. Execution of SQL on GPU (Select, Projection, Join)
3. Make result set on GPU
4. GPU ➞ RAM Data Transfer

New Data Flow

Existing Data Flow

Table

PG-Strom Preview (Feb-2016)
Orchestrating a brighter world

NEC

NEC brings together and integrates technology and expertise to create the ICT-enabled society of tomorrow.

We collaborate closely with partners and customers around the world, orchestrating each project to ensure all its parts are fine-tuned to local needs.

Every day, our innovative solutions for society contribute to greater safety, security, efficiency and equality, and enable people to live brighter lives.