# PL/CUDA – In-database massive parallel analytics

KaiGai Kohei <kaigai@kaigai.gr.jp>
The PG-Strom Development Team

## Abstract

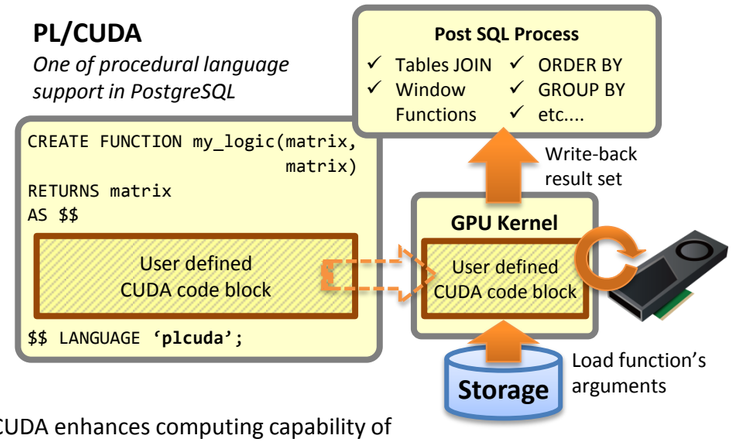
As literal, DBMS (database management software) is a software to manage databases, not a software to calculate advanced analytic algorithms rapidly. So, people usually export data-set from DBMS then run standalone applications for their researches, but it is often uncomfortable.
We hacked SQL-function's programming language infrastructure of PostgreSQL[*1] as a part of PG-Strom's [*2] new feature. It allows to put raw CUDA code block inside of SQL-function, then this code block is executed on GPU device with many processing cores. Its function arguments and result are transferred between host and device by the infrastructure, so researcher can focus on their problem.
We had an evaluation of this feature on drug-discovery workloads; to find out similar chemical compounds using k-NN similarity search algorithm towards 10million chemical compounds database.
Our query returned the results in 20sec, but CPU version of the equivalent logic took about 50minutes (x150 times faster!). It also means GPU has a power to changes researcher's daily job from batch-process to try&error manner.

## Concept of PL/CUDA functions

We call the infrastructure for advanced analytics in-database PL/CUDA (*Procedural Language for CUDA*).
It allows users to define a CUDA code block as body of SQL function.
Once PL/CUDA function is invoked in SQL-queries, its infrastructure set up a GPU kernel and load the function arguments to GPU RAM, then execute the GPU kernel.
Its results are also written back to the host RAM, with a form we can reference using regular SQL operations.
This feature gives users two major benefits:

**PL/CUDA**
*One of procedural language support in PostgreSQL*

```
CREATE FUNCTION my_logic(matrix,
                         matrix)
RETURNS matrix
AS $$
  User defined
  CUDA code block
$$ LANGUAGE 'plcuda';
```




① **Calculations close to the data**
Data migration is always expensive. PL/CUDA enhances computing capability of DBMS which manages the database. It is exactly location of the data.

② **SQL flexibility for data manipulation on pre-/post-process of the algorithm core**
SQL has rich functionality for data manipulation also. It is much productive than adjustment of stand-alone program or write-back processed data once exported into the database again.

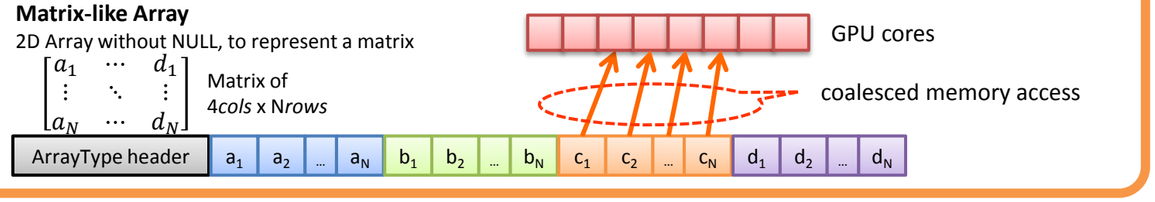
## Background

We tried to generate GPU kernel from SQL-statement automatically [*3] in the past. However, it was a tough way because SQL behavior requires many overhead like NULL-checks or overflow-checks for each arithmetic operations, and row-oriented data format of PostgreSQL database could not pull-out full memory access throughput.
So, once we left from the original idea, then designed an architecture to run advanced analytic algorithms; which expects at least $O(N^2)$ grade calculation amount.
The first re-design was manual optimization from automatic optimization. Nobody wants write-up analytic algorithms using SQL, and makes no sense on SQL compatibility. The other re-design is data format to improve data density using columnar structure by support of data format exchange function. (`Matrix` is a synonym of 2D-array but without NULL, which has been supported in PostgreSQL.)

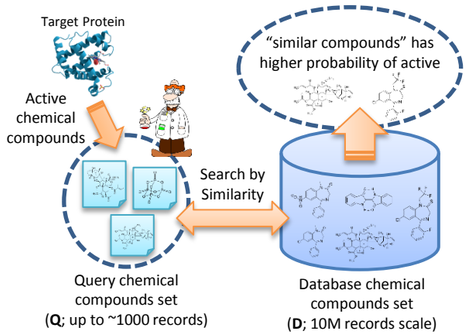
**Matrix-like Array**
2D Array without NULL, to represent a matrix

$$\begin{bmatrix} a_1 & \cdots & d_1 \\ \vdots & \ddots & \vdots \\ a_N & \cdots & d_N \end{bmatrix}$$

Matrix of 4*cols* x N*rows*




## Case Study: Drug-Discovery In-database




Drug is a chemical compound which affects to a particular protein working bad for human health, and must be harmless.
One approach to discover a new drug candidate is; looking for similar one to the existing/known chemical compounds.
In academic society, people daily reports drug-candidates that are "active" to the target protein. So, we can collect a set of known chemical compounds from the papers and so on.
Search space of the entire drug candidates is almost 10M entries nowadays. We implement a PL/CUDA function to calculate similarity from the set of query chemical compounds for each chemical compound in the database.
Researcher considers "similar" chemical compounds are promising ones than others, and valuable for investigations.

| ID | NAME | Fingerprint (1024bit) |
|---|---|---|
| 1 | CHEMBL153534 | 0000000000010000010000000000000010000000000000100... |
| 2 | CHEMBL405398 | 0000000000000001001000000000000000000000000000010... |
| 3 | CHEMBL503634 | 0000010000000000000000000010000001000000000000000... |
| : | : | : |

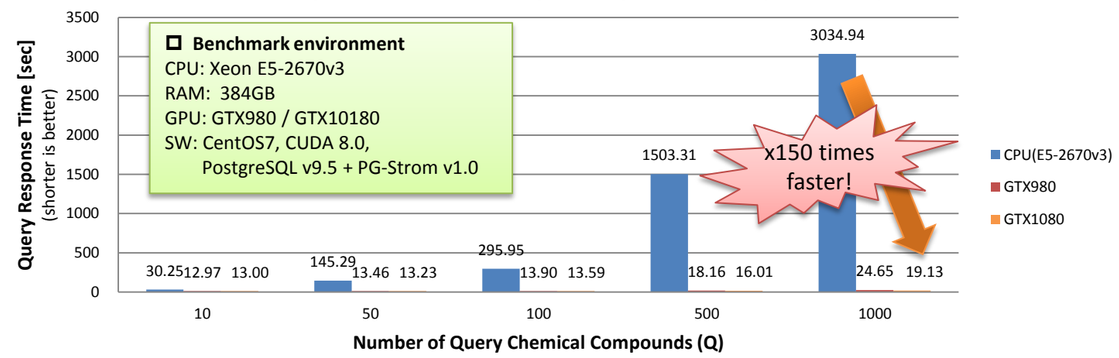
Similarity definition by Jaccard index:
$Similarity(A,B) = |A_{FP} \cap B_{FP}| / |A_{FP} \cup B_{FP}|$

Our PL/CUDA function calculates the similarity based on the k-NN method. This algorithm calculates distance for each combination of the query- and database chemical compounds, then picks up the nearest ***k*** distance for each database chemical compound. Average of them are considered as representative distance of the database chemical compounds.
It is highly calculation intensive. If Q-set has 1000 items, we have to calculate distance for 10-billion combinations and sort 10-million partitions.
GPU in-database shows destructive performance improvement. It shortened query execution time from 50min to 19sec. This performance leap allows interactive TRY&ERROR on researcher's jobs, rather than batch processing.

**PL/CUDA function invocation for k-NN similarity search**

```
SELECT knn_gpu_similarity(3,Q.matrix, D.matrix))
  FROM (SELECT cbind(array_matrix(id),
                     array_matrix(fingerprint)) matrix
          FROM finger_print_query) Q,
       (SELECT matrix
          FROM finger_print_10m_matrix) D
```

**Similarity search of chemical compounds by k-NN method (k=3, D=10M)**




## Conclusion&Future

DBMS is always the location "closest to data". Once DBMS gets integration of GPU computing power, it eliminates the necessity of data export to use external computing resources, and also allows to utilize SQL's flexibility for data manipulation for pre-/post-process of the core of advanced algorithms.
Right now, user has to write up algorithms from the scratch, but reinvention of the wheel. We plan to provide a collection of well known analytic algorithms, like MADLib, in-database and on-GPU.

[*1] PostgreSQL – Well used open source RDBMS, https://www.postgresql.org/
[*2] PG-Strom – Extension of PostgreSQL for GPU acceleration, http://strom.kaigai.gr.jp/
[*3] http://on-demand.gputechconf.com/gtc/2016/presentation/s6118-kohei-postgresql.pdf