Towards Fast SQL Query Processing in DB2 BLU Using GPUs

A Technology Demonstration

Sina Meraji sinamera@ca.ibm.com

Please Note

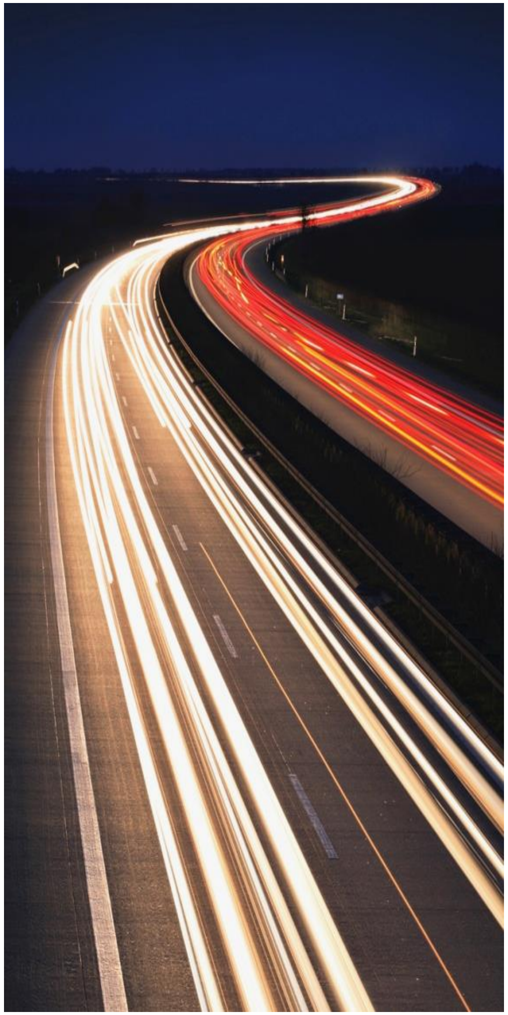
- IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
- The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Outline

- DB2 BLU Acceleration
- Hardware Acceleration
- Nvidia GPU
- Key Analytic Database Operators
- Our Acceleration Design
- Live Technology Demonstration

DB2 with BLU Acceleration



Next generation database

- Super Fast (query performance)
- Super Simple (load-and-go)
- Super Small (storage savings)

Seamlessly integrated

- Built seamlessly into DB2
- Consistent SQL, language interfaces, administration
- Dramatic simplification

Hardware optimized

- Memory optimized
- CPU-optimized
- I/O optimized

OpenPOWER™

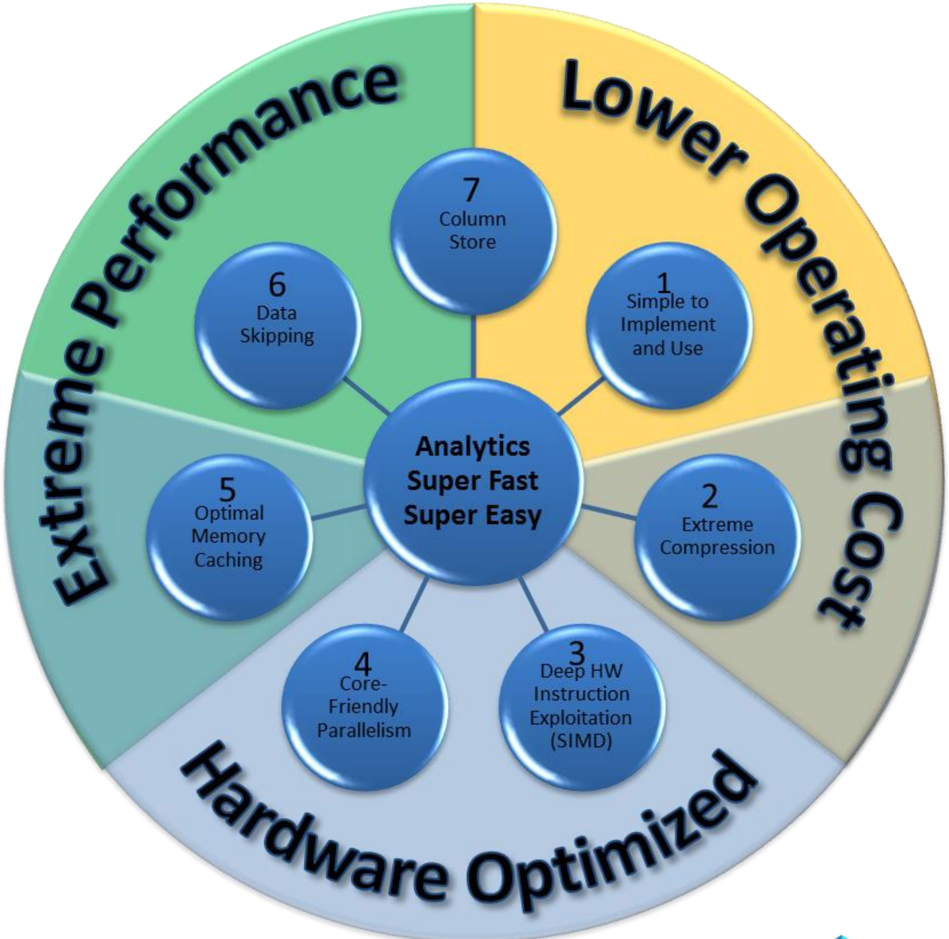
Risk system injects 1/2 TB per night from 25 different source systems.
"Impressive load times."

Some queries achieved an almost 100x speed up with literally no tuning.

6 hours!
Installing BLU to query results

One of the world's most profitable and secure rated banks.

DB2 with BLU Acceleration: The 7 Big Ideas



OpenPOWER™

Hardware Acceleration

- Use specific hardware to execute software functions faster
- Popular accelerator technology
  - SIMD
    - Present in every CPU
  - GPUs
    - Easy to program
  - FPGA
    - Hard to program

Nvidia GPU

- NVIDIA Tesla K40
  - Kepler technology
  - Peak double precession performance: 1.66 TFLOPs
  - Peak single precession performance: 5 TFLOPs
  - High Memory Bandwidth: up to 288 GB/Sec
  - Memory Size: 12GB
  - Number of cores: 2880

Key Analytic Database Operators

- ## GROUP BY / Aggregation
  - ```
    SELECT column_name, aggregate_function(column_name)
    FROM table_name
    WHERE column_name operator value
    GROUP BY column_name;
    ```
- ## Join
  - ```
    SELECT column_name(s)
    FROM table1
    JOIN table2
    ON table1.column_name=table2.column_name;
    ```
- ## Sort
  - ```
    SELECT column_name
    FROM table_name
    ORDER BY column_name;
    ```

OpenPOWER™
    ```

Hardware Configuration

- POWER8 S824L
  - 2 sockets, 12 cores per socket, SMT-8, 512GB
  - Ubuntu LE 14.04.02 LTS

- GPU:
  - 2 NVIDIA Tesla K40

Infrastructure

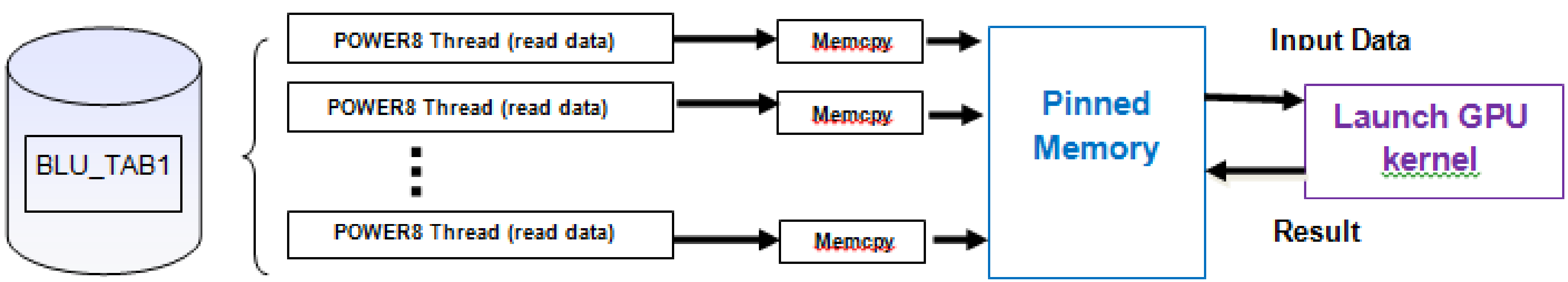
- Adding the support for Nvidia GPU
  - CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by NVIDIA
- Memory Management
  - Pin/Unpin memory
  - To run on GPU, threads asks for pinned memory
  - This is for fast transfers to/from GPU
    - PCI-E Gen3
  - Will be improved in 2016 with Nvlink on POWER

GPU Scheduler

- Each CPU thread can submit tasks to GPU scheduler
  - Should submit memory requirement on GPU
- The scheduler checks all the GPUs on the system
  - Reserve the memory on the GPU
  - Create a stream
  - Return to the CPU thread with GPU number and stream Id

Our Acceleration Design

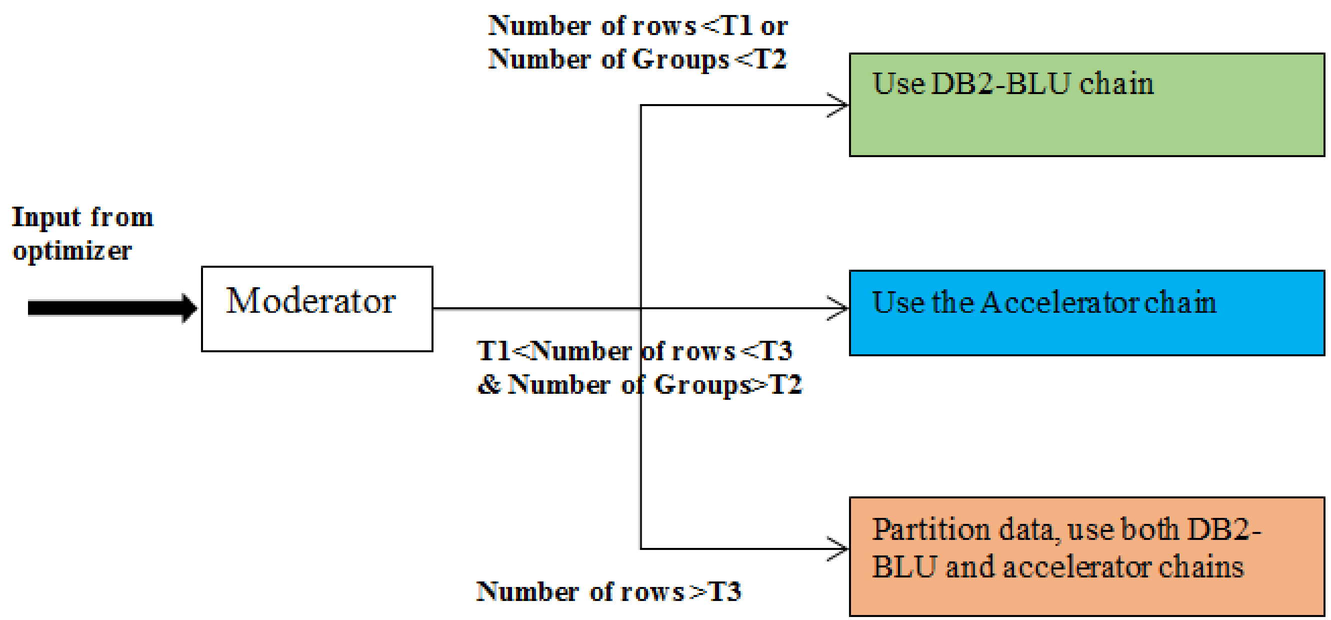
- Use parallel POWER8 threads for reading/pre-processing data
- Transfer data to GPU
- Have the GPU to process the query
- Transfer the result back to host machine



OpenPOWER™

Hybrid Design: Use Both POWER8 and GPU for Query Processing

- Decide where to execute the query dynamically at runtime
  - Use GPU only
  - Use CPU only
  - Use both



GPU Kernels

- Design and develop our own GPU runtime
- Developed fast kernels
  - e.g. GROUP BY, aggregation
- Use Nvidia CUDA calls
  - e.g. Atomic operations
- Use Nvidia fast kernels
  - e.g. sort

Group By/Aggregation

- What does it do?
- SELECT C1, SUM(C2) FROM simple_table GROUP BY C1

Simple_Table

C1	C2
9	98
2	21
9	92
3	38
9	90
2	22

C1	SUM(C2)
9	280
3	38
2	43

OpenPOWER™

Group by/Aggregate Operation in GPU

- Hash based algorithm
- Use grouping keys and a hash function to insert keys to a hash table
- Aggregation
  - Use Nvidia Atomic CUDA calls for some data types (INT32, INT64,etc)
  - Use Locks for other data types (Double, FixedString, etc)
- Three main steps
  - Initialize the hash table
  - Grouping/Aggregation in a global hash table
  - Scanning the global hash bale to retrieve groups

Initialization kernel

- Create/initialize the hash table in device memory
- Data needs to be aligned → May need Padding
  - Grouping key can be anywhere in the hash table based on alignment requirements
- Initialization happens in parallel using parallel GPU threads
- Select SUM(C1), MIN(C2), MAX(C3) from table1 Group by(C1)
  - Int 64: C1, C2
  - Int 32: C3

C1(64bit)	SUM(C1) (64bit)	MAX(C2)(64bit)	MIN(C3)(64bit)	Padding(32 bit)
FFFFFFFFFFFFFFFF	0	-9223372036854775808	2147483647	0
FFFFFFFFFFFFFFFF	0	-9223372036854775808	2147483647	0
…	…			
FFFFFFFFFFFFFFFF	0	-9223372036854775808	2147483647	0

Hash based Group by/Aggregate

- Group by:
  - Parallel threads read keys/payloads from table and insert keys to HT
  - Use a hash function to hash keys
    - Murmur hashing: Wide keys(larger than 64bit)
      - http://en.wikipedia.org/wiki/MurmurHash
    - Mod hashing: short keys(smaller than 64bit)
  - If collision happens, we check the next empty slot in hash table
- Aggregation:
  - If thread key matches an entry in HT we need to perform the Agg function

Thread i

Key	Payload 1	Payload2
ABFGH	13	21.2

HT(before Aggregation)

Key	Sum	Min
.....	.....	.....
ABFGH	8	1.2
.....	.....	.....

HT(After Aggregation)

Key	Agg1	AGG2
.....	.....	.....
ABFGH	21	1.2
.....	.....	.....

OpenPOWER™

Aggregation

- CUDA atomic operations for
  - Implemented in hardware(very fast)
  - Use for both global and shared memory
  - Specific data types(INT32, INT64, etc)

- Use AtomicCAS for specific data types e.g. Double
  - Specific Agg functions/data types

```
__device__  double atomicAdd(double* address, double val) {
    unsigned long long int* address_as_ull = (unsigned long long int*)address;
    unsigned long long int old = *address_as_ull, assumed;
    do {
        assumed = old;
        old = atomicCAS(address_as_ull, assumed, __double_as_longlong(val + _longlong_as_double(assumed)));
    }while(assumed != old);

    return __longlong_as_double(old);
}
```

Check Nvidia docs for more details: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#atomic-functions

Aggregation(Continued)

- Locks
  - For datatypes that are larger than 64 bits
    - Decimal, FixedString
  - Each thread needs to perform following
    - Acquire a lock which is associated with the corresponding entry in hash table
    - Apply the AGG function
    - Release the lock
  - Costly operation

Hash-Based Group By/Aggregate

- SELECT C1, SUM(C2) FROM Simple_Table GROUP BY C1

Simple_Table

KEY	Value
93	5
23	2
93	1
23	5
93	0
93	1000

Parallel HT Creation

Hash Table

Key	Aggregated Value
23	7
93	1006

Parallel Probe

Result

23	7
93	1006

OpenPOWER™

Supported Data Types & AGG functions

SQL--	MAX	MIN	SUM	COUNT
SINT8	Cast to SINT32	Cast to SINT32	Cast to SINT 32	AtomicCount
SINT16	Cast to SINT32	Cast to SINT32	Cast to SINT32	AtomicCount
SINT32	AtomicMax	AtomicMin	AtomicAdd	AtomicCount
SINT64	AtomicMax	AtomicMin	AtomicAdd	AtomicCount
REAL	Use AtomicCAS	Use AtomicCAS	AtomicAdd	AtomicCount
DOUBLE	Use AtomicCAS	Use AtomicCAS	Use AtomicCAS	AtomicCount
DECIMAL	Lock	Lock	Use AtomicADD(2-3 steps)	AtomicCount
DATE	CAST to SINT32	CAST to SINT32	N/A	AtomicCount
TIME	CAST to SINT32	CAST to SINT32	N/A	AtomicCount
TIMESTAMP(64bit)	AtomicMax	AtomicMIn	N/A	AtomicCount
FixedString	Lock	Lock	N/A	AtomicCount

OpenPOWER™

GPU SORT

- Reduced the amount of data transferred between host and GPU device
  - Use Nvidia Fast sort kernel
  - Copy key and data to GPU memory, use 4-byte key and 4-byte payload
  - Skip the copying back of the sorted keys
  - Skip the copying of payload data into GPU memory on subsequent sorts to resolve duplicates.
  - Use the same data format between DB2 and GPU sort routines

- Handling multiple small sort jobs concurrently in the GPU
  - Handle multiple small sort jobs in the GPU
  - Each thread works on sort data range
  - there are more sort key bytes to process

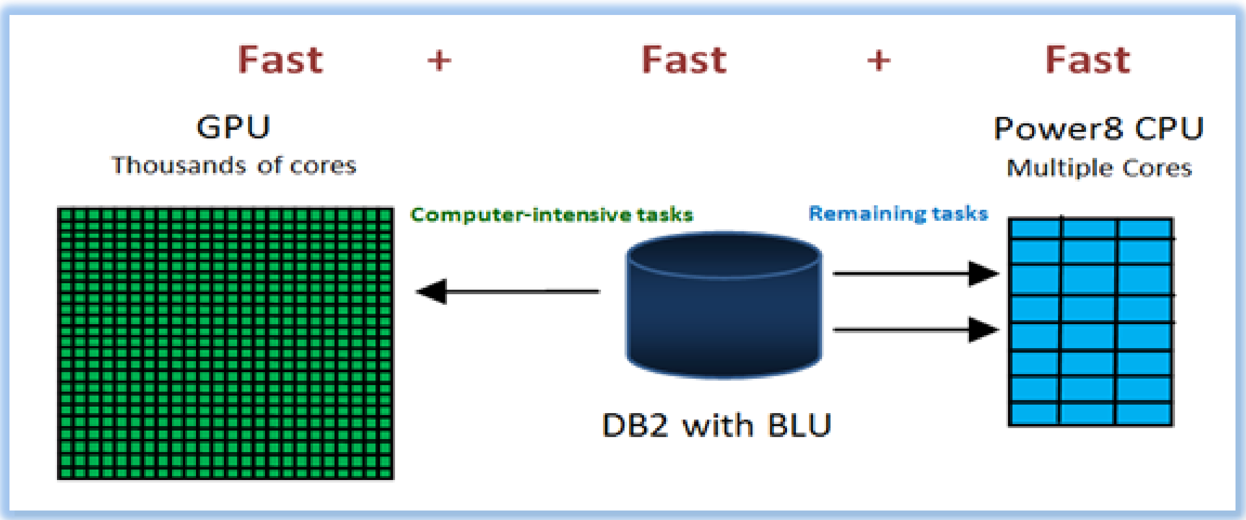
GPU SORT

- Where GPU performs BEST:
  - Up to 750M rows when all sort data fit within GPU device memory
  - Sort on single integer column of size 4-byte or less.
    i.e. only one trip to the GPU is required

Acceleration Demonstration

- Accelerating DB2 BLU Query Processing with Nvidia GPUs on POWER8 Servers
  - A Hardware/Software Innovation Preview

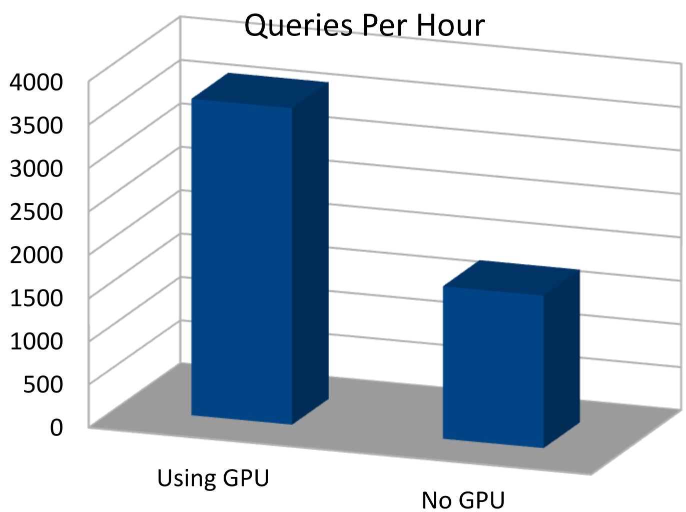
- Compare query acceleration of DB2 BLU with GPU vs. non- GPU baseline
- Show CPU offload by demonstrating increased multi-user throughput with DB2 BLU with GPU

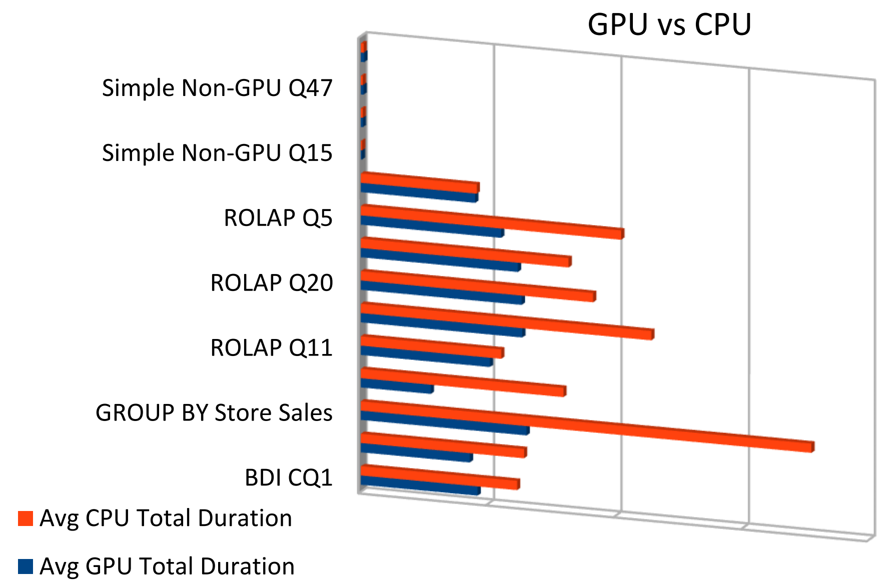


BLU Analytic Workload

- A set of Queries from existing BLU Analytic workloads
  - TPC-DS database schema
    - Based on a retail database with in-store, on-line, and catalog sales of merchandise
  - 15% of queries use GPU heavily
  - 50% of queries use GPU moderately
  - 35% of queries do not use GPU at all
- Benchmark Configuration
  - 100 GB (raw) Data set
  - 10 concurrent users

Performance Result



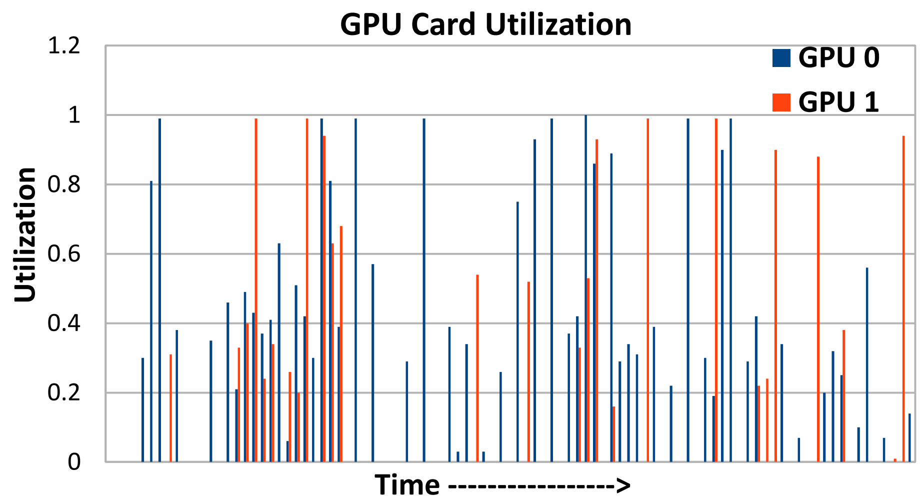


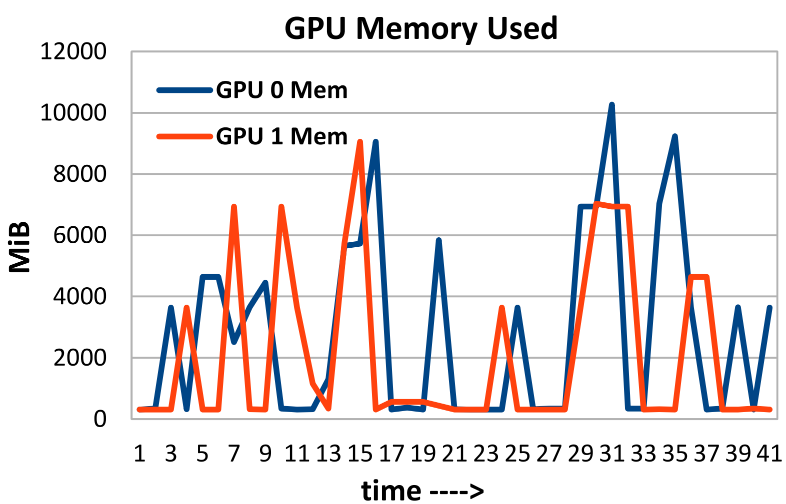
- ~2x improvement in workload throughput
- CPU Offload + improved query runtimes are the main factors

- Most individual queries improve in end-to-end run time

GPU Utilization

The DB2 BLU GPU demo technology will attempt to balance GPU operations across the available GPU devices





These measurements are taken from the Demo Workload running in continuous mode.

OpenPOWER™

Summary

- Hardware/Software Innovation Preview demonstrated GPU Acceleration
- Improved DB2 BLU query throughput
  - Use both POWER8 processor and Nvidia GPUs
  - Design and develop fast GPU kernels
  - Use Nvidia kernels, function calls, etc
- Hardware Acceleration shows potential for
  - Faster execution time
  - CPU off-loading