Improving performances of an embedded RDBMS with a hybrid CPU/GPU processing engine

Samuel Cremer¹,², Michel Bagein¹, Saïd Mahmoudi¹, Pierre Manneback¹

¹UMONS, University of Mons
Computer Science Department, University of Mons, Rue de Houdain 9, 7000, Mons, Belgium

²HEH, Haute Ecole en Hainaut
Computer Engineering Department, Haute Ecole en Hainaut, Av. Maistriau 8A, 7000, Mons, Belgium

samuel.cremer@heh.be
michel.bagein@umons.ac.be
said.mahmoudi@umons.ac.be
pierre.manneback@umons.ac.be
Context (1)

- **Exponential growth of data volumes**
  - Big Data and NoSQL
  - Not only data centers -> end-user applications

- **RDBMS -> still essential**

- **Embedded RDBMS (SQLite, MySQL embedded, SQL Server Compact)**
  - Targets: personal computers, embedded devices and servers
  - Used as: storage system
    - local cache system  -> In-Memory DB
  - Does not take advantage of current hardware specificities
Context (2)

• Idea:
  
  improving the performances of SQLite with a hybrid implementation over multicore CPU and GPU (CuDB)

• Benefits of faster Embedded RDBMS:
  – Better latencies
  – Better energy efficiency
  – Processing of larger data volumes
Why using a GPU?

- GPUs -> widely available

- GPUs are SIMT architectures (Single Instruction, Multiple Threads)
  - Fast for processing a same instruction on different data
  - SQL -> processing a same query on different rows

- Compared to CPUs, GPUs have overall better:
  - Number of cores: 2560 vs 8 (16 threads)
  - Computing power: ~9000 Gflops vs ~800 GFlops
  - Memory bandwidth: ~300 GB/s vs ~80 GB/s
  - Energy efficiency: 50 GFlops/W vs 6 GFlops/W

- GeForce GTX 1080 (~800€) vs Xeon E5-1660 v4 (~1000€)

- Offloads the CPU
Hybrid VM chooses to execute processing, either on CPU cores or GPU cores according to the data volume they have to process.

\[ x = \text{size of the biggest accessed table (threshold} = \sim 1000 \text{ records)} \]

GPU engine uses CUDA threads / CPU engine uses POSIX threads

Entire database is in GPU global memory: “In-GPUMem DB”
CuDB: Specificities

**SELECT queries are boosted by S(QP)MD paradigm** (Single Query Plan, Multiple Data)

Insertions are processed asynchronously by the CPU

Multiple storage engines:
- **Affinity** (row order and dynamic typing)
- **Boost** (column order and static typing) <- fastest engine
### Experimental results: Hardware

<table>
<thead>
<tr>
<th></th>
<th>Intel Core i7 2600K</th>
<th>GeForce GT740 GDDR5</th>
<th>GeForce GTX 770</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cores</strong></td>
<td>4 (8 Threads)</td>
<td>384</td>
<td>1536</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>3.4 – 3.8 GHz</td>
<td>~1 GHz</td>
<td>~1 GHz</td>
</tr>
<tr>
<td><strong>Memory Bandwidth</strong></td>
<td>21.4 GB/s</td>
<td>80 GB/s</td>
<td>224 GB/s</td>
</tr>
<tr>
<td><strong>Computing Power (SP)</strong></td>
<td>217 GFlops</td>
<td>762 GFlops</td>
<td>3.213 GFlops</td>
</tr>
<tr>
<td><strong>TDP</strong></td>
<td>95 W</td>
<td>64 W</td>
<td>230 W</td>
</tr>
</tbody>
</table>

- **Only focused on extraction queries** -> execution time of prepared statements

- **CuDB compared to:**
  - SQLite with an In-Memory database
  - MySQL 5.7 with MEMORY tables

- **Transfer times required to send query-plans and results were considered**
Experimental results: SELECT WHERE Queries

Average speedups with SELECT WHERE Queries

Peak speedup of 411x with: SELECT * WHERE col LIKE ‘%substring%’
Experimental results: SELECT JOIN Queries

Average speedups with SELECT JOIN Queries
SQLite and CuDB build transient indexes, MySQL does not
Peak speedup of 66x with self-join queries
Experimental results: Energy Efficiency

CPU (X)T = CPU engine of CuDB with (X) threads
Conclusion and Future Works

• Great speedups for full table scans

• Better energy efficiency

• We plan to:
  – overcome the limitations of the GPU memory capacity
  – add full indexation mechanisms
  – improve SQL support  ->  TPC-H and SSB
Thank You!

Any questions?