Speeding up tree/graph queries using GPUs

Lila Shnaiderman
(joint work with Prof. Oded Shmueli)

Computer Science Faculty, Technion
November 4, 2015
Technion
Outline

• Background

• XML Stream Storage

• GPU-Twig algorithm

• GPU-Twig experimental results

• GGQ (GPU Graph database Querying)

• GGQ experimental results
Background

- **XPath:**
  - The most popular query language for XML.
  - **Twig Pattern:** a tree pattern of selection predicates on a data tree or data graph.

- **Answer to a query:**
  - All occurrences of twig pattern in an XML tree.
  - Semantics different than that of XPath.
  - Variant: just answer nodes, similar to XPath.

- **Aim**
  - Reduce the processing time of a single twig query.

- **Approach:**
  - *Speed XPath processing using different parallel methods!*
XML DB model (1)

- **Database:**
  - forest of rooted, ordered, labeled trees.
  - **Node:** element or value.
  - **Edge:** element – sub element, element-value relationship.

- **Twig pattern:** An ordered labeled tree.
  - **Node:** element tags or string values.
  - **Edge:** parent-child (single line) or ancestor-descendant (double line).

- **Match of Q in D:**
  Given a twig pattern Q (query) and XML data D, mapping from Q to D such that:
  - each Q node maps to a distinct node in D.
  - the structural relationships between Q nodes are satisfied by the corresponding DB nodes.
Answer to $Q$ having $n$ nodes:

- Full answer: $n$-ary relation. Each tuple $(d_1, \ldots, d_n)$ consists of a distinct match of $Q$ in $D$.
- Answer nodes: Images of rightmost leaf node of $Q$ in matches of $Q$ in $D$, or images of some specific other $Q$ node (or nodes).

Twig Pattern Matching problem:

- Given a query twig pattern $Q$ and an XML file $D$ (represented via streams), compute the answer (all matches, or answer nodes) to $Q$ in $D$. 
Outline

• Background

• XML Stream Storage
  • GPU-Twig algorithm
  • GPU-Twig experimental results
  • GGQ (GPU Graph database Query) algorithm
  • GGQ experimental results
Streams Storage Scheme

- Stream $T_q$:
  - Extends the inverted index data structure used in DBs.
  - Separate stream for each element label.
  - Stream ‘a’ contains positional representations of data nodes whose labels are ‘a’:
    (DocId,LeftPos:RightPos, LevelNum)
Outline

• Background
• XML Stream Storage
• GPU-Twig algorithm
• GPU-Twig experimental results
• GGQ (GPU Graph database Query) algorithm
• GGQ experimental results
What Accounts For the Difference presented in graphs? Need to understand how CPUs and GPUs differ

- Latency Intolerance versus Latency Tolerance.
- Task Parallelism versus Data Parallelism.
- Multi-threaded Cores versus SIMT (Single Instruction Multiple Thread) Cores.
- 10s of Threads versus 10,000s of Threads.

**CPUs are low latency low throughput processors**

**GPUs are high latency high throughput processors**
Highly Parallel, Multithreaded, Manycore Processor

- GPUs have more ALUs for the same sized chip and therefore run many more threads of computation

Modern GPUs run 10,000s of threads concurrently!!!
GPU overview (3)

• **How?**
  - GPU avoids synchronization issues between so many threads?
  - GPU is patches, schedules, caches, and context switches 10,000s of threads?
  - GPU program 10,000s of threads?

• **Special Design!!!**
  - Threads are independent of each other – no synchronization issues
  - SIMD (Single Instruction Multiple Data) threads – minimize thread management
  - Reduce hardware overhead for scheduling, caching etc.
  - Program blocks of threads

• **What problems GPU can solve?**
  - Data Parallel Problems: Luckily ☺️, Plenty of problems fall into this category
    • Graphics, image & video processing, physics, scientific computing, ...
    • In fact the more the data, the more efficient GPUs become at these algorithms
Parallelism in CPUs v. GPUs:

- CPUs use task parallelism
- Multiple tasks map to multiple threads
- Tasks run different instructions
- 10s of relatively heavyweight threads run on 10s of cores
- Each thread managed and scheduled explicitly
- Each thread has to be individually programmed

- GPUs use data parallelism
- SIMD model (Single Instruction Multiple Data)
- Same instruction on different data
- 10,000s of lightweight threads on 100s of cores
- Threads are managed and scheduled by hardware
- Programming done for batches of threads.
GPU-Twig (first phase)

- Copy query relevant data streams from CPU memory to GPU global memory.
- While there are unprocessed nodes in the query tree (qTree)
  - Choose node $q$ from qTree
    - (such that all $q$’s child nodes were already processed)
  - Derive structural information of all nodes $n$ in the stream with $q$’s label name.
    - Each node is processed by a different thread!
    - Mark if $n$ is “an answer” of the sub query rooted at $q$. 

GPU-Twig (first phase)

The content: is current node has descendant that matches node with the given index in qTree?

qArray: Table indexes are the qTree indexes.

18:21,4 = the open index: the close index, the depth in the document

12345 00010
GPU-Twig (second phase)

- **Goal:** Find the answer nodes using the $qArray$ structures (holding the derived info).

- **$qPath$:** the path between the answer node $qAnsN$ and the $qTree$ root node.

- **AnsN is an answer:**
  - if there is at least one match of $qPath$ in $dTree$ such that
    - $AnsN$ node maps to $qAnsN$
    - *And* in all the path nodes $qArray$ is filled according to the requirements of the relevant $q$ node.
GPU-Twig (second phase)

Each node in qAnsN stream corresponds to different thread!
Outline

• Background
• XML Stream Storage
• GPU-Twig algorithm
• GPU-Twig experimental results
• GGQ (GPU Graph database Query) algorithm
• GGQ experimental results
Experiments

**Time: Twig1**

- **CPU_12_threads**
- **GPU_ALL**
- **GPU_WITH_RES**
- **GPU_ALG_ONLY**

**Time: Twig5**

- **CPU_12_threads**
- **GPU_ALL**
- **GPU_WITH_RES**
- **GPU_ALG_ONLY**

**XMark(1GB)**

- **CPU_12_threads**
- **GPU_ALL**
- **GPU_WITH_RES**
- **GPU_ALG_ONLY**

**XMark(2GB)**

- **CPU_12_threads**
- **GPU_ALL**
- **GPU_WITH_RES**
- **GPU_ALG_ONLY**

22/55
Experiments

![XMark Speedup Graph]

- GPU_ALL
- GPU_WITH_RES
- GPU_ALG_ONLY

Time (sec)

- twig 1
- twig 2
- twig 3
- twig 4
- twig 5

Graph showing speedup comparison for different twig types and GPU configurations.
Outline

• Background
• XML Stream Storage
• GPU-Twig algorithm
• GPU-Twig experimental results
• GGQ (GPU Graph database Query) algorithm
• GGQ experimental results
GraphDB

- Any database with a graph structure.
- Emerging format.
- Example (json format):

```
{
    "vertices": [
        {
            "name": "Ateret", "age": 2, "_id": 0, "_type": "vertex"},
        {
            "name": "Hallel", "age": 4, "_id": 1, "_type": "vertex"},
        {
            "name": "Jonathan", "lang": "computer", "_id": 2, "_type": "vertex"},
        {
            "name": "Lila", "age": 32, "_id": 3, "_type": "vertex"},
        {
            "name": "Baby", "lang": "babyish", "_id": 4, "_type": "vertex"},
        {
            "name": "Arie", "age": 42, "_id": 5, "_type": "vertex"
        }
    ],
    "edges": [
        {
            "_id": 6, "_type": "edge", "_outV": 0, "_inV": 1, "_label": "knows"},
        {
            "_id": 7, "_type": "edge", "_outV": 0, "_inV": 3, "_label": "knows"},
        {
            "_id": 8, "_type": "edge", "_outV": 0, "_inV": 2, "_label": "knows"},
        {
            "_id": 9, "_type": "edge", "_outV": 3, "_inV": 4, "_label": "created"},
        {
            "_id": 10, "_type": "edge", "_outV": 3, "_inV": 2, "_label": "created"},
        {
            "_id": 11, "_type": "edge", "_outV": 5, "_inV": 2, "_label": "created"},
        {
            "_id": 12, "_type": "edge", "_outV": 3, "_inV": 5, "_label": "knows"
        }
    ]
}
```
The problem

• **Context:** basic *parallel query processing* in graph structured documents.

• **Navigation:** Any graph DB query language.

• **Solution:** A new method called *GGQ*
  – matches between a graph DB and tree pattern queries.
  – uses *General-Purpose GPUs*.
  – allows massive parallelization of the above task and hence quick response.

• **We designed a** *New graph DB Storage Scheme for GPUs* that fits GGQ:
  – stores graph structured documents using arrays.
  – all graph characteristics are encoded within the arrays.
  – allows fast, concurrent and easy navigation over the data graph for many threads.
Current solutions (1)

• Graph DB query processing methods today:
  – Converting a graph DB to a relational DB, and using regular relational DB solutions.
  – Proposing new sequential query processors (GREMLIN, etc.).
  – Running many queries in parallel (each query is run sequentially).
  – Few works addressing parallel graph DB query processors (parallelGDB).
**GGQ Description**

**Algorithm Input:**
- A graph $G$.
- A query tree pattern $Q$.
- A set of nodes $V_q \subseteq V$ containing all nodes which are legal matches for the root node of $Q$.
  - In our case $V_q = \{0\}$

**Algorithm Goal:** Find all possible matches between $Q$ and $G$.

**Basic Idea:**
- Use the ID of the current thread to determine the portion of data with which to match.
- Execute a simple thread-based pattern matching attempt to the above portion.
GPU-graphDBq (2)

- Data graph
- qTree

- Name: Jonathan
  Language: computer
- Name: Ateret
  Age: 2
  Knows:
  - Name: Hallel
    Age: 4
    Created:
  - Name: Baby
    Language: babyish
    Created:
  - Name: Lila
    Age: 32
    Created:
  - Name: Arie
    Age: 42
    Created:

Number of bits in thread ID: <b3,b2,b1,b0>
- Thread ID=0: <0,0,0,0>
- Thread ID=2: <0,0,1,0>
- Thread ID=1: <0,0,0,1>
- Thread ID=5: <0,1,0,1>

\[ tNum_{\text{created}} = 2 \text{ (outgoing edges)} \Rightarrow \text{need 1 bit to represent} \]
\[ tNum_{\text{knows}} = 4 \text{ (outgoing edges)} \Rightarrow \text{need 2 bit to represent} \]
Algorithmic complications (2)

Second complication:

• Any node $v$ can have no more than $tNum$ outgoing edges labelled $lbl$ for any $lbl$.
• For Query $Q$ \[ Q\text{NumBits} = \log_2 \sum_{lbl \subseteq \text{queryEdges}} tNum(lbl) \] is number of bits needed to represent the current query.
• $Q\text{numBits}$ can be $\geq$ than available bits for thread ID (maxBitsThrdID).
• So, it is impossible to represent the query pattern with maxBitsThrdID.

Solution: Break the query into few queries, solve the problem in parts.

$Q$ \hspace{1cm} $Q_{\text{pref}}$ \hspace{1cm} $Q_{\text{remain}}$
Algorithmic complications (3)

The solution formally:

- Break the query into 2 (or more) parts:
  - Remove some subtrees from the full query pattern $Q$
  - the remaining new pattern $Q_{\text{pref}}$ is a “prefix” of $Q$
  - The idea: there is sufficiently many bits to explore $Q_{\text{pref}}$
- Run QQG on $Q_{\text{pref}}$, and save its answers to PrefSet.
- For each answer $ans$ in PrefixAnsSet:
  - Run $Q_{\text{remain}}$ query based on the matching presented by $ans$ in data graph $G$.
  - The answers are added to the FinalAnsSet.
Outline

• Background
• XML Stream Storage
• GPU-Twig algorithm
• GPU-Twig experimental results
• GGQ (GPU Graph database Query) algorithm
• GGQ experimental results
GGQ results (1)

The right two columns contain the speedup of GGQ run in comparison to a Gremlin run.

<table>
<thead>
<tr>
<th></th>
<th>Path Q1</th>
<th>Path Q2</th>
<th>Speedup</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gremlin</td>
<td>114</td>
<td>84</td>
<td>Q1 148</td>
<td>Q2 11</td>
</tr>
<tr>
<td>GPU – full</td>
<td>0.8</td>
<td>7.4</td>
<td>1267</td>
<td>1050</td>
</tr>
<tr>
<td>GPU – ans</td>
<td>0.09</td>
<td>0.08</td>
<td>1425</td>
<td>1200</td>
</tr>
<tr>
<td>GPU – alg</td>
<td>0.085</td>
<td>0.075</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

On a document with size 125MB, for path queries with 5 nodes.

On a document with size 600MB, for a path query with 5 nodes and a tree query with 6 nodes.
GGQ results (2)

The right two columns contain the speedup of GGQ run in comparison to a Gremlin run.

<table>
<thead>
<tr>
<th></th>
<th>Tree Q1</th>
<th>Tree Q2</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gremlin</td>
<td>187</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>GPU – full</td>
<td>49</td>
<td>5.4</td>
<td>3.8</td>
</tr>
<tr>
<td>GPU – ans</td>
<td>48</td>
<td>2.6</td>
<td>3.9</td>
</tr>
</tbody>
</table>

On a document with different sizes, for two different tree queries with 11 nodes, while stopping right after finding the first answer (GPU-full).

<table>
<thead>
<tr>
<th></th>
<th>Tree Q1</th>
<th>Tree Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gremlin</td>
<td>118</td>
<td>123</td>
</tr>
<tr>
<td>GPU – full</td>
<td>7.5</td>
<td>1.48</td>
</tr>
<tr>
<td>GPU – ans</td>
<td>7.4</td>
<td>1.09</td>
</tr>
</tbody>
</table>

On small documents, for two different tree queries with 11 nodes.
Thank You!