A Parallel GPU Solution to the Maximal Clique Enumeration Problem for CBIR

Dr. Christopher Henry, P. Eng.
Dept. of Applied Computer Science
University of Winnipeg
515 Portage Ave.
Winnipeg, MB, Canada R3B 2E9

GTC 2014
Introduction

• Goal: Quantify the nearness (or apartness) of sets of objects based on their descriptions
• Tolerance near set theory provides a framework for this assessment
• Approach is dependent on finding all the tolerance classes on a set of objects
• The problem of finding tolerance classes has recently been mapped to performing Maximal Clique Enumeration (MCE)
• Focus: An efficient method for finding all tolerance classes on a set of objects
• Application: Content-Based Image Retrieval
Tolerance Near Sets

- Goal: Determine the perceptual nearness of disjoint sets of objects
- Near sets provide:
  - Framework
  - A systematic method to formally answer the question “Are these sets similar?”
    - To what extent
- Traces its origins contributions of Webner, Fechner, Poincaré, Zeeman, Sossinsky, Reiz, Pawlak, Orłowska, and Peters
Tolerance Relation and Tolerance Classes

• Tolerance near sets are defined by a description-based tolerance relation
• Tolerance relations provide a view of the world without transitivity

\[ \equiv_{B, \varepsilon} = \{(x, y) \in O \times O : \| \phi(x) - \phi(y) \|_2 \leq \varepsilon \} \]
Nearness Measure

- The nearness measure was created out of a need to determine the degree that near sets resemble each other

\[
tN M_{\Sigma_{B, \epsilon}}(X, Y) = 1 - \left( \sum_{C \in H_{\Sigma_{B, \epsilon}}(Z)} |C| \right)^{-1} \cdot \sum_{C \in H_{\Sigma_{B, \epsilon}}(Z)} |C| \frac{\min(|C \cap X|, |C \cap Y|)}{\max(|C \cap X|, |C \cap Y|)}
\]
Application: Content-Based Image Retrieval

- Goal: Retrieve images based on content of an image
  - Rather than on semantic string or keyword associated with the image
Perceptual Image Analysis

- Subimages are the perceptual objects used in this work
- Aim: find tolerance classes (sets) contained in the union of subimages from two images
- MCE is the approach used to find these tolerance classes
Image Dataset

- Results are based on images from the SIMPLcity image dataset
Maximal Clique Enumeration

• Observation: The problem of finding classes can be mapped to the Maximal Clique Enumeration Problem
• Classes can be found using an algorithm with reduced complexity
  • Based in graph theory
• MCE problem consists of finding all the maximal cliques among an undirected graph
  • Let $G=(V,E)$ denote an undirected graph
    • $V$: set of vertices
    • $E$: set of edges that connect pairs of distinct vertices from $V$
  • A clique is a set of vertices where each pair of vertices in the clique is connected by an edge in $E$
  • A maximal clique in $G$ is a clique whose vertices are not all contained in some larger clique
Bron-Kerbosh MCE Algorithm

• First serial algorithm for MCE was developed by Harary and Ross
• Since then, two main approaches have been established to solve the MCE problem
  • Greedy approach by Bron-Kerbosh
    • Concurrently discovered by Akkoyunlu
  • Output-sensitive approaches
• General idea: find maximal cliques through a depth-first search
  • Branches are formed based on candidate cliques
  • Backtracking occurs once a maximal clique has been discovered
• Algorithm essentially marks new nodes and processes them
Maximal Clique Enumeration

Algorithm 1: The BK algorithm

1. **Input**: A graph $G$ with vertex set $V$ and edge set $E$
2. $comp = \emptyset$
3. $comp' = V$
4. $\text{meta} = \emptyset$
5. Call CliqueEnumerate($comp$, $comp'$, $\text{meta}$)

(a) General BK algorithm

(b) The recursive clique enumeration function

(c) Example graph

(d) BK search tree
Visualization of Tree Structure
GPU Algorithm

- Each block of threads processes one pair of images
  - Performs comparison between query image and candidate image
- Each thread executes MCE algorithm on 1 node in the tree
  - *i.e.* each thread finds all child nodes for a given node
- One level of the tree is processed per iteration
- Threads and blocks are arranged in one dimension
GPU Algorithm
GPU Algorithm

• Each block is assigned memory space for processing a different pair of images

• Stopping condition
Algorithm Iteration

Block-level Sorting

```c
//
/* MCE GPU Kernel. */
__global__ void mceKernel()
{
    do {
        mceIteration();
        cubRadixSort();
        copyOutToIn();
    } while (moreNodes);
}
```

Device-level Sorting

```c
//
/* MCE GPU Kernel. */
void mceKernel()
{
    do {
        dim3 dimBlock(BLK_SIZE,1);
        dim3 dimGrid(GRID_SIZE,1);
        mceIteration<<<dimGrid, dimBlock, 0, stream>>>();
        cubRadixSort();
        copyOutToIn<<<dimGrid, dimBlock, 0, stream>>>();
    } while (moreNodes);
}
Node Data Structure

- Recall, each node contains three data structures
  1. clique: A list of vertices in the current search path
  2. cands: A list of candidates vertices that are not in clique
     - But, connected to every vertex in clique
  3. not: A list of vertices that are connected to every vertex in clique
     - But, would form a redundant path if combined with vertices in clique

- Each of these are stored as a bit set
  - Contained in an array of type unsigned char

- Bit set representation allows for efficient intersection and counting operations

- These nodes are stored in global memory
  - Also too large to store in shared memory
  - CGMA is 1 since node is only used by 1 thread
Global Memory

• The input and output of each kernel iteration are nodes
  • Each thread can generate multiple nodes during each kernel iteration
    • The number of output nodes are not known \textit{a priori}
• After each iteration a radix sort is used to move the output to contiguous input locations
  • Sort is performed on offsets using CUB library
• Each thread is also updating its portion of the nearness measure after each iteration
  • Can be calculated as a weighted average
Texture Memory

- Each node requires the adjacency matrix during MCE
- For this CBIR application the adjacency matrix is ~26KB
  - Too large for shared and constant memory
- Also not possible to know adjacency matrix access patterns a priori
- Thus, adjacency matrix is stored in texture memory
  - One for each block (each pair of images)
## Timing Results

<table>
<thead>
<tr>
<th>$\varepsilon$</th>
<th>CPU MCE (sec) i7-930</th>
<th>GPU MCE (sec) GeForce GTX 460</th>
<th>GPU MCE (sec) K20</th>
<th>K20 Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.85</td>
<td>0.1</td>
<td>0.009</td>
<td>585</td>
</tr>
<tr>
<td>0.2</td>
<td>0.84</td>
<td>0.06</td>
<td>0.023</td>
<td>724</td>
</tr>
<tr>
<td>0.3</td>
<td>2.45</td>
<td>1.42</td>
<td>0.035</td>
<td>862</td>
</tr>
<tr>
<td>0.4</td>
<td>28.71</td>
<td>3.39</td>
<td>0.078</td>
<td>1277</td>
</tr>
</tbody>
</table>
CBIR Results

Avg. Cat. 0

Avg. Cat. 1

Avg. Cat. 2

Avg. Cat. 3

Avg. Cat. 5

Avg. Cat. 6

Avg. Cat. 7

Avg. Cat. 8

Avg. Cat. 9
CBIR Results
CBIR Results
CBIR Results
Conclusion

- The algorithm presented here is tailored to a specific dataset
- Approach can be adapted for use on larger graphs
- The key restraint is the upper limit on the amount of nodes each thread can generate
  - Enough storage must be available for these nodes
Acknowledgements

• This research has been supported by:
  • The Natural Sciences and Engineering Research Council of Canada (NSERC) grant 418413
  • NVIDIA Corporation

• Special thanks to:
  • Keith Massey
  • Tariq Alusaifeer
  • Sheela Ramanna
  • James F. Peters
References

References

Thank You