Dominoes: Exploratory Data Analysis of Software Repositories Through GPU Processing

Jose Ricardo
Esteban Clua
Leonardo Murta
Anita Sarma
Introduction

- Identifying expertise is important:

- Normally, expertise is identified through informal process:
  - Social network
  - Implicit knowledge of work dependencies
- Even more challenging in globally distributed development
Introduction

- Software development leaves behind the *activity* logs for mining relationships
  - *Commits* in a version system
  - Tasks in a *issue* tracker
  - Communication
- Finding them is *not a trivial* task
  - There is an extensive amount of data to be analyzed
  - Data is typically stored across *different* repositories
- *Scalability* problems depending on the project size
  - Processing the history of *large* repositories at *fine grain* for exploratory analysis at interactive rate
Related Work

- EEL scope the analysis to 1,000 project elements
  - Restrict the history to small chunk of data

- Cataldo analyze data at coarse-grain
  - Developer is expert of the whole artifact

- Boa allows fine-grain analysis by using a CPU cluster
  - Normally require a time slice for using the cluster
  - Require data submission for processing
Solving the Problem

Performance

Implement used operations for data analysis in GPU

+ 

Usability

Deeper Research

User interface

Happy user / researcher
Solving the Problem

- Efficient large-scale repository analysis
- Enable users to explore relationships across different levels of granularity
- No requirement for a specialized infrastructure
Dominoes

• Infrastructure that enables interactive exploratory data analysis at varying levels of granularity using GPU

• Organizes data from software repositories into multiple matrices
  • Each matrix is treated as Dominoes tile
  • Tiles can be combined through operations to generate derived tiles
    • Transposition, multiplication, addition, …
Dominoes UI

- Dominoes’ tiles resemble a **Dominoes game**, where the user can play with to build new relationships.
Basic Building Tiles

[developer|commit]

D   C

[class|method]

Cl   M

[package|file]

P   F

[issue|commit]

I   C

[commit|method]

C   M

[commit|file]

C   F

[file|class]

F   Cl
Examples of Derived Building Tiles

- \([\text{method}|\text{method}] \ (MM = \text{CM}^T \times \text{CM})\): represents method dependencies

- \([\text{class}|\text{class}] \ (\text{ClCl} = \text{CIM} \times \text{MM} \times \text{CIM}^T)\): represents class dependencies

- \([\text{issue}|\text{method}] \ (IM = IC \times \text{CM})\): represents the methods that were changed to implement/fix an issue
• **Extractor module** gather information from repository and save to database

• **Basic block builder** is responsible to generate building blocks relationship from database

• Operations are performed in GPU using a **Java Native Interface** call

• Derived and basic building block **still in memory** for future use
Data Structure

Java

…
long pointer m1, m2, res;
createObj(res);
multiplication(m1, m2, res)
…

C / CUDA

void multiplication(JNIEnv *env, jclass obj,
        jlong m1, jlong m2, res )
{
    Matrix * m1 = (Matrix*) m1;
    Matrix * m2 = (Matrix*) m2;
    Matrix * res = (Matrix*) m2;
    GPUMul(m1, m2, _res);
}

- Matrix are very sparse for some relationships
  - Developer x Commit

- The java side maintain a pointer to the sparse matrix allocated in C side
  - The matrix are stored in CRS format

- Matrix operations performed in C using a JNI interface
Operations in GPU

**Linear Transformation**
- Addition
- Multiplication
- Transposition

**Data Mining**
- Confidence
- Lift
- Support

**Statistics**
- Mean
- Standard Deviation
- Z-Score
Linear Transforms

- Allows connecting pieces in the Dominoes by changing its edge

- Allows extracting further relationships in the data by combining the different types of data

- Uses cusp library for performing linear transforms
Reduction

- Normally used for calculating the **amount** of relationship
  - Total of classes modified by a developer
  - How many bugs a developer have inserted in a method Y

- Uses the **Thrust** library for calculating it in GPU
Confidence

- Used to detect the relationship direction

- Each GPU thread is responsible for processing the confidence for each element

\[ M^{\text{conf}}[i,j] = \frac{M^{\text{SUP}}[i,j]}{M^{\text{SUP}}[i,i]} \]
Confidence

- Due to the fact that the row and column must be known, they are computed and stored in a vector.
- Given a sparse $M \times M$ with $t$ non zero values:

\[
\begin{array}{cccccc}
\text{Value} & V_1 & V_2 & V_3 & \ldots & V_t \\
\text{Row} & R_1 & R_2 & R_3 & \ldots & R_t \\
\text{Col} & C_1 & C_2 & C_3 & \ldots & C_t \\
\text{Diagonal} & D_1 & D_2 & \ldots & D_M
\end{array}
\]

For each $t$ GPU thread:

\[
\begin{align*}
\text{diagIdx} &= \text{row}[\text{idx}]; \\
\text{conf}[\text{idx}] &= \text{value}[\text{idx}] / \text{diagonal}[\text{diagIdx}]
\end{align*}
\]
Z-Score

• Responsible to convert an absolute value to a score above the mean

  $z = \frac{(x - \mu)}{\sigma}$

  $x = \text{absolute score}$
  $\mu = \text{mean}$
  $\sigma = \text{standard deviation}$

• Require a set of steps
  • Calculating the mean / column
  • Calculating the standard deviation
  • Finally calculating the z-score
Z-Score

• Calculating the mean / column
  • Given a Matrix $M \times N$, containing $t$ non zero values, the GPU is responsible to sum up all values for a column, producing a vector sized $N$ for the mean.

Value

| $V_1$ | $V_2$ | $V_3$ | ... | $V_t$ |

Col

| $C_1$ | $C_2$ | $C_3$ | ... | $C_t$ |

Kernel 1

For each $t$ GPU thread

```cpp
colIdx = col[idx]
atomicAdd(value[idx], sum[idx])
atomicAdd(1, count[idx])
```

Kernel 2

For each $N$ GPU thread

```cpp
mean[idx] = sum[idx] / count[idx]
```
Z-Score

- Calculating the standard deviation / column
- Given a Matrix $M \times N$, the GPU is responsible to sum up all values for a column, producing a vector sized $N$ for the standard deviation

\[ V_1, V_2, V_3, ..., V_t \]
\[ C_1, C_2, C_3, ..., C_t \]
\[ M_1, M_2, M_3, ..., M_N \]

**Kernel 1**

For each $t$ GPU thread:
- `colIdx = col[idx]`
- `colMean = mean[colIdx]`
- `deviate = value[idx] – colMean`
- `deviatePower2 = deviate * deviate`
- `atomicAdd(deviatePower2, variance[colIdx])`

**Kernel 2**

For each $N$ GPU thread:
- `colVariance = variance[idx]`
- `colVarianceSqrt = sqrt(colVariance / M)`
- `deviation[idx] = colVarianceSqrt`
- `M_1, M_2, M_3, ..., M_N`
# Z-Score

- Calculating the standard score

Given a Matrix $M \times N$ with $t$ non-zero elements, the GPU is responsible to produce the z-score

<table>
<thead>
<tr>
<th>Value</th>
<th>Col</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>$C_1$</td>
<td>$M_1$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$V_2$</td>
<td>$C_2$</td>
<td>$M_2$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>$V_3$</td>
<td>$C_3$</td>
<td>$M_3$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$V_t$</td>
<td>$C_t$</td>
<td>$M_N$</td>
<td>$S_N$</td>
</tr>
</tbody>
</table>

For each $t$ GPU thread

```
colIdx = col[idx]
colMean = mean[colIdx]
standardDev = sd[colIdx]
z = (value[idx] - colMean) / standardDev
zscore[idx] = z
```
Applicability

Dependency Identification

Expertise Identification

Expertise breadth identification
Results

- Evaluation time (support and confidence).
  - [file|commit] (34,335 x 7,578)
    - CPU: 696 minutes | GPU: 0.7 minutes | Speed up: 994
  - [method|commit] (305,551 x 7,578)
    - CPU: N/A | GPU: 5 minutes | Speed up: -

* Intel Core 2 Quad Q6600 2.40GHz PC with 4GB RAM and a nVidia GeForce GTX580 graphics card was used.
# Results

- **[Developer|File|Time]**: 114 layers of 36 x 3400 (13,953,600 elements)
- **[Developer|Method|Time]**: 114 layers of 36 x 43,788 (179,705,952 elements)

<table>
<thead>
<tr>
<th>EBD when Considering Files (seconds)</th>
<th>EBD when Considering Methods (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean &amp; SD</strong></td>
<td><strong>Z-Score</strong></td>
</tr>
<tr>
<td>CPU</td>
<td>2.19</td>
</tr>
<tr>
<td>GPU</td>
<td>0.10</td>
</tr>
<tr>
<td>Speed Up</td>
<td>21.90</td>
</tr>
</tbody>
</table>

*EBD = Expertise Breadth of a Developer.*
Results


Conclusions

• The main contribution is using GPU for solving Software Engineering problems

• Employment of GPU allows seamless relationship manipulations at interactive rates
  • Uses matrices underneath to represents building blocks

• Dominoes opens a new realm of exploratory software analysis, as endless combinations of Dominoes’ pieces can be experimented in an exploratory fashion

• Thanks to the use of GPU, the user can do its analysis on its own machine
Questions

jricardo@ic.uff.br

http://www.josericardojunior.com

https://twitter.com/jricardojunior

https://br.linkedin.com/in/jose-ricardo-da-silva-junior-7299987