ABSTRACT
Document classification and clustering algorithm requires a lot of vector similarity calculation. The complexity of the algorithm is \(O(n^2)\) and its speed is seriously lower down as the number of documents increases. The performance of similarity calculation will be highly improved if we use a GPU parallelism. In this paper, we propose a method of improving the calculation speed by CUDA framework as GPU parallelism can accelerate vector operation. Most troubling part of using CUDA framework for GPU parallel processing is a memory loading time. We tried to find the best way to reduce memory loading time. The fastest method is to use texture-memory and the second fastest way is to use global-memory. Operation on the GPU was three times faster than CPU operation when texture-memory was used.

KEY WORDS
GPU parallelism, document classification, CUDA framework, memory loading time, texture memory, pinned memory

1. Introduction
As the social network services produce much document in a short time, one of the most widely pursued subjects these days is the mass document classification. People express themselves through a number of social network services such as Twitter and Facebook, so a technology to categorize large amount of documents is needed to analyze people’s opinions.

kNN(k-Nearest Neighbor algorithm) algorithm is a basic categorizing algorithm that classifies a target based on the \(k\) number of its nearest neighbours. It uses shared information from those \(k\) neighbours, with the number \(k\) traditionally being an odd number for the sake of efficiency and accuracy. If \(k\) is an even number, kNN could not judge in some cases [1].

2. Related Researches

2.1 k-Nearest Neighbor Algorithm
kNN algorithm is a basic categorizing algorithm that classifies a target based on the \(k\) number of its nearest neighbours. It uses shared information from those \(k\) neighbours, with the number \(k\) traditionally being an odd number for the sake of efficiency and accuracy. If \(k\) is an even number, kNN could not judge in some cases [1].

2.2 Vector Space Model
Vector space model is a method to convert documents into vector to perform mathematical operations. The keywords that compose a document are converted into vector by various methods such as VSM. Researchers analyze documents using mathematical operations of the vector. The cosine similarity value and the distance similarity value are often used [2].

2.2.1 Cosine Similarity
Cosine similarity is the intrinsic value of the vectors in VSM. It lies between 0 and 1 depending on the characteristics of internal value, indicating how similar the documents are. The closer the value is to 1, the similar the documents are. Therefore, if the similarity value is close to 1, the documents in question can be considered similar. The cosine similarity calculation formula is described in Figure 1.

\[
sim(d_i, d_j) = \frac{\vec{d}_i \cdot \vec{d}_j}{||\vec{d}_i|| \times ||\vec{d}_j||} = \frac{\sum_{w \in W} W_{w} \times W_{w}^j}{\sum_{w \in W} W_{w}^2 \times \sum_{w \in W} W_{w}^2}
\]

Figure 1. Cosine similarity calculation
2.2.2 Euclidean Distance Similarity
Euclidean distance similarity is the mathematical distance value of the two documents. Small distance indicates similarity between the two documents. Long distance means the two documents are not similar. Figure 2 is Euclidean distance calculation formula.

\[ \text{sim}(d_j, d) = \sqrt{\sum_{i=1}^{n}(w_y - w_{id})^2} \]

Figure 2. Euclidean distance calculation

2.2.3 TF-IDF (Term Frequency - Inverse Document Frequency)
TF-IDF is one of the famous ways to convert a document into vectors. It makes suitable vectors, whose values contain importance. Because vectors created by TF-IDF are creditable, researchers often use TF-IDF generated vector in VSM. TF is a frequency of keywords in document and IDF is the inverse frequency of other documents. TF-IDF value is the multiplication of TF and IDF values [9].

2.3 CUDA Framework
CUDA framework was created by NVIDIA company as a GPU development tool. Programmers perform parallel processing operations through CUDA framework. Many ALU on the GPU makes it suitable for parallel processing. Although ALU of the GPU is simpler than its counterpart of the CPU, ALU of the GPU is more suitable in a simple operation. In addition, CUDA framework provides the ability to use a variety of memories for users [6, 7, 10].

2.4 Vector Operations using CUDA Framework
Improving the calculation speed of the vector space model using CUDA framework has already been studied by Park(2012). According to previous studies, cosine similarity calculation speed on GPU parallel processing by CUDA framework is approximately 15 times faster than using the CPU. This study dealt only with the shared memory on GPU, and did not go into memory transition time [11].

3. Document Classification by GPU Programming

3.1 Document Classification by GPU
All documents had stemmed 10,000 keywords so they could be represented as vectors with 10,000 dimensions. The random generator is based on TF-IDF scheme, so the value of vector was under 1 and over 0. Because kNN algorithm needs the classified documents to classify new documents, existing documents were already classified. kNN algorithm used the code in Figure 3 to classify the documents.

```
1. Function similarity_operation
   (refer_doc_vec_table, input_doc_vec_table )
2. n = length( refer_doc_vec_table )
3. m = length( input_doc_vec_table )
4. similarity_list
5. // new array of n*m empty list
6. for i = 0 to m do
7. // the number of new docs
8. for j = 0 to n do
9. // the number of existing docs
10. refer_doc_vector = refer_doc_vec_table[ j ]
11. input_doc_vector = input_doc_vec_table[ i ]
12. similarity_list[ i*m+n ] =
13. similarity ( refer_doc_vec, input_doc_vec )
   // comparing part about GPU and CPU
14. return similarity_list
```

Figure 3. Document classification code using kNN

According to the code in Figure 3, the program found the nearest K by the similarity values. The cosine value and the Euclidean distance value were used as the similarity value. We compared operation time on the CPU and GPU parallel processing using CUDA framework. Various memories on GPU such as texture-memory, global-memory, and pinned-memory were used to improve the operation speed.

While CPU time has been calculated with the entire program that was executed on the CPU, GPU computation time has been measured using both CPU and GPU. Only part of the similarity calculation was executed on GPU using CUDA framework. The rest was executed on CPU.

Figure 4. Memory structure
3.2 Memory Overhead Solution

Operation on GPU needs to read data from CPU memory to GPU. It happens to be the most troubling part of using CUDA framework for GPU parallel processing. As our focus was to find the best method of reducing the memory loading time, we used various types of memories since each memory’s loading time is different [4,5].

Fig 4 is a simplified diagram that shows the structure of the GPU memories. In this paper, we used pinned-memory in CPU, texture-memory in GPU, and global-memory in GPU and compared the time when each memory was used.

There are characteristics of each memory. GPU parallel operation with pinned-memory uses the data bus several times because pinned-memory is located on the CPU. Texture-memory and global-memory are in the GPU, and their capacity is large. So, bulk transfer efficiency is highly.

When we compared the memories, shared memory was not considered. Although it is accessible and efficient, its capacity is the worst of memories because of its memory size. Thus, this experiment disregarded shared memory.

4. Experimentation

As mentioned in the related research, similarity calculation speed of CPU was compared with that of GPU with parallel processing. However, the research compared the operation speed of vectors only. Reading and writing to memory were not considered although one of the most important points in using the CPU parallel processing is the problem of loading memory.

In this paper, substantial kNN run-time was measured using a variety of GPU memory types. In addition, we studied which method and memory are the fastest at calculating the similarities of document classification algorithm [11].

4.1 Environments

The experiment was conducted in the environment mentioned in Table 1. Its target data is mentioned in Table 2. The experiment data created by the random function were documents.

Table 1. Experiment environment

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Core i7 980x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- 6 core (12threads)</td>
</tr>
<tr>
<td></td>
<td>- 3.33 GHz</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA GeForce GTX 580</td>
</tr>
<tr>
<td></td>
<td>- 512 CUDA Cores (32 Cores/SM x 16 SM)</td>
</tr>
<tr>
<td></td>
<td>- 2.0 CUDA Compute Capability</td>
</tr>
<tr>
<td></td>
<td>- 384Bit Memory Interface Width</td>
</tr>
<tr>
<td></td>
<td>- 192.4 GB/Sec Memory Bandwidth</td>
</tr>
</tbody>
</table>

Table 2. Experimental data

<table>
<thead>
<tr>
<th>Classified Docs</th>
<th>Unclassified Docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x10, 10x100, 10x1000, 100x10, 100x100, 100x1000, 1000x10, 1000x100, 1000x1000</td>
<td></td>
</tr>
</tbody>
</table>

4.2 Experimental Results and Evaluation

Experiment results with variety of memories are organized in table 3 and table 4. Table 3 shows the result of the kNN algorithm with cosine. Table 4 shows the result of the kNN algorithm with Euclidean Distance.

Table 3. Comparison of calculation time of kNN with Cosine

<table>
<thead>
<tr>
<th>Cosine</th>
<th>CPU</th>
<th>GPU pinned</th>
<th>GPU global</th>
<th>GPU texture</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x10</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10x100</td>
<td>16</td>
<td>160</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>10x1000</td>
<td>165</td>
<td>1778</td>
<td>109</td>
<td>93</td>
</tr>
<tr>
<td>100x10</td>
<td>16</td>
<td>48</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>100x100</td>
<td>156</td>
<td>1709</td>
<td>52</td>
<td>32</td>
</tr>
<tr>
<td>100x1000</td>
<td>1654</td>
<td>17890</td>
<td>525</td>
<td>385</td>
</tr>
<tr>
<td>1000x10</td>
<td>155</td>
<td>506</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>1000x100</td>
<td>1549</td>
<td>17595</td>
<td>417</td>
<td>357</td>
</tr>
<tr>
<td>1000x1000</td>
<td>16562</td>
<td>178609</td>
<td>4993</td>
<td>3373</td>
</tr>
</tbody>
</table>

Table 4. Comparison of calculation time of kNN with Euclidean Distance

<table>
<thead>
<tr>
<th>Euclid</th>
<th>CPU</th>
<th>GPU pinned</th>
<th>GPU global</th>
<th>GPU texture</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x10</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>10x100</td>
<td>9</td>
<td>161</td>
<td>24</td>
<td>15</td>
</tr>
<tr>
<td>10x1000</td>
<td>107</td>
<td>1779</td>
<td>105</td>
<td>89</td>
</tr>
<tr>
<td>100x10</td>
<td>9</td>
<td>51</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>100x100</td>
<td>88</td>
<td>1707</td>
<td>52</td>
<td>33</td>
</tr>
<tr>
<td>100x1000</td>
<td>1090</td>
<td>18039</td>
<td>569</td>
<td>399</td>
</tr>
<tr>
<td>1000x10</td>
<td>91</td>
<td>555</td>
<td>70</td>
<td>69</td>
</tr>
<tr>
<td>1000x100</td>
<td>892</td>
<td>17687</td>
<td>458</td>
<td>382</td>
</tr>
<tr>
<td>1000x1000</td>
<td>10847</td>
<td>180269</td>
<td>5422</td>
<td>3548</td>
</tr>
</tbody>
</table>

Above tables contain actual running time on the CPU and GPU when a variety of documents and memories were used. The numbers on the table rows indicate the number of documents in two parts: the number of pre-classified documents and the number of new documents. For example, 10×10 means 10 pre-classified documents and 10 unclassified documents. The 10×10 execution time has standard of 1. Above tables are full of measuring times.
varying in accordance to differences in memories and documents. The measuring value is the median value of 10 experiment times.

Table 3 has the measured values of the kNN algorithm with cosine similarity. It clearly shows how quantity of documents affects the speed of kNN algorithm. In case of \(1000 \times 1000\), the large amount of documents resulted in more time. Texture-memory use in kNN algorithm yielded the fastest result in \(1000 \times 1000\) cases. The resulting time was approximately 4.9 times faster than that of CPU use. Global-memory used computation time was also 3.3 times faster than using CPU.

Table 4 contains the measured values of the kNN algorithm with Euclidean Distance. The result of Table 4 and Table 3 are similar. On the whole, the fastest method is to use texture-memory and the second fastest way is to use global-memory. Operation on the GPU is three times faster than CPU operation when texture-memory was used. GPU computation time using global-memory was also twice as faster than using CPU.

The operation on the CPU was the fastest when kNN classified a small number of documents. However, in the case of classifying a large number of documents, the operation on the GPU with texture-memory was the fastest and the pinned memory use was the slowest method.

### 5. Conclusion

Document classification and clustering algorithm needs a lot of same operations of similarity calculation in which those operations can be done simultaneously. This kind of operation is of good shape to compute on the GPU platform, and the CUDA framework can speed up the whole computation time through GPU parallelism. Many algorithms that are running in huge number of servers like information service providers are also very good to operate on the GPU. However, there are some problems regarding the CUDA version of the parallel algorithm because it needs a bulk transfer of memories and memory size. So, people using the CUDA framework should be careful for memory loading and memory transfer time. The same goes for the document classification and document clustering algorithm.

The experiment shows that the best method of implementing text classification algorithm is to operate on CPU when the number of documents is relatively small. When the amount of document is large, using texture-memory or global-memory on GPU have been faster than using them on the CPU. The method of using pinned-memory was the worst because pinned-memory requires a data bus frequently. Even the method using pinned-memory operated on the GPU parallel processing was not efficient. As a conclusion, the best method to processing a huge number of documents such as social network data is to use a texture memory on the GPU programming.

### Acknowledgements

This research has been performed in a Prof. Eun-Jin Im’s graduate class of GPU programming in Kookmin university. We really appreciate Prof. Im’s good comments about GPU and CUDA programming and Jin-Woo Lee’s lots of help in implementing and testing the software as a member of project team.

### References