Future Generation Computer Systems 29 (2013) 1736–1741

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# GPU enhanced parallel computing for large scale data clustering

Xiaohui Cui [a,c,*], Jesse St. Charles [b], Thomas Potok [a]

[a] Oak Ridge National Laboratory, Oak Ridge, TN 37831, United States

[b] Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, United States

[c] New York Institute of Technology, Manhattan, NY 10023, United States

ARTICLE INFO

*Article history:*
Received 17 June 2011
Received in revised form
4 June 2012
Accepted 20 July 2012
Available online 7 September 2012

*Keywords:*
GPU
Swarm intelligence
Data clustering
CUDA

ABSTRACT

Analyzing and clustering large scale data set is a complex problem. One explored method of solving this problem borrows from nature, imitating the flocking behavior of birds. One limitation of this method of data clustering is its complexity $O(n^2)$. As the number of data and feature dimensions grows, it becomes increasingly difficult to generate results in a reasonable amount of time. In the last few years, the graphics processing unit (GPU) has received attention for its ability to solve highly-parallel and semi-parallel problems much faster than the traditional sequential processor. In this paper, we have conducted research to exploit this architecture and apply its strengths to the flocking based high dimension data clustering problem. Using the CUDA platform from NVIDIA, we developed a Multiple Species Data Flocking implementation to be run on the NVIDIA GPU. Performance gains ranged from 30 to 60 times improvement of the GPU over the 3GHz CPU implementation.



## 1. Problem statement and background

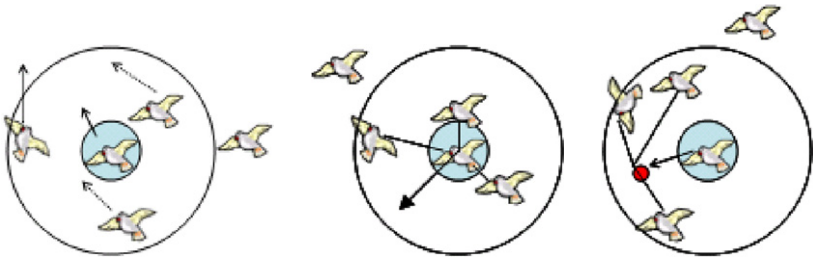
Cluster analysis is a descriptive data mining task, which involves dividing a set of data objects into a number of groups, called clusters. The motivation behind clustering a set of data is to find its inherent structure and expose that structure as a set of groups [1]. The data objects within each group should exhibit a large degree of similarity while the similarity among different clusters should be minimal [2]. The need for fast, efficient data analysis has driven the research community to continually develop and improve data clustering methods.

One method, multiple species flocking clustering [3], a nature-inspired computational model for simulating the dynamics of flocks of entities, is used for high dimensional unstructured data clustering. This method takes an agent-based approach and relies on emergent organization to effectively cluster data. The effectiveness of this approach relies on the organization that arises through a group of agents interacting through simple rules. In the case of data clustering, similar data sets flock together, loosely organizing themselves according to subject. This method has met with success in clustering high dimensional datasets better than traditional methods such as $K$-means [3]. Unfortunately the method is highly computational intensive and requires hours of computational time to generate acceptable results when analyzing more than a few thousands high dimensional datasets at a time. Our research investigates the possibility of implementing this algorithm on GPU enhanced machines, thereby reducing the computational time and bringing the flocking based data clustering capability to the data analyst's desktop. The unstructured text (document) is considering the most complex high dimensional unstructured data set for information analysis. In this chapter, we use document clustering as an example for explaining our implementation of GPU enhanced flocking data clustering.

Document clustering provides a structure for efficiently browsing and searching text. It is a fundamental operation used in unsupervised document organization, automatic topic extraction, and information retrieval. There are two major clustering techniques: partitioning and hierarchical [2]. Many document clustering algorithms can be classified into these two groups. Hierarchical algorithms break data into relational trees. This method has high computational requirements for large data sets. In recent years, it has been recognized that the partitioning techniques are well suited for clustering large document datasets due to their relatively low computational requirements [4]. The best-known partitioning algorithm is the $K$-means algorithm and its variants [5]. This algorithm is simple, straightforward and based on the firm foundation of analysis of variances. In 2008, Farivar and his colleagues [6] implemented $K$-means on an NVIDIA 8600 GT graphics card using CUDA and got a 13x performance improvement compared to a baseline 3 GHz Intel Pentium(R) based PC running the same algorithm. One drawback of the $K$-means algorithm is that the clustering result is sensitive to the selection of the initial cluster centroids and may converge to local optima, instead of global ones.

* Corresponding author at: Oak Ridge National Laboratory, Oak Ridge, TN 37831, United States.
*E-mail address:* cuixhui@gmail.com (X. Cui).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.07.009



(a) Alignment. (b) Separation. (c) Cohesion.

**Fig. 1.** The three basic rules in the boid.

Another limitation of the $K$-means algorithm is that it requires a prior knowledge of the approximate number of clusters for document collection.

In this research project, we implemented the flocking based data clustering algorithm on an NVIDIA 8800 GTX and compared the runtime performance of CPU and GPU versions of the algorithm for document clustering. Using an NVIDIA GPU platform we saw a dramatic 30–60 times improvement over the sequential CPU implementation. Ultimately, we are working toward illustrating a low-cost, high-capacity GPU computational platform and programming model suitable for most naturally inspired applications.

## 2. Core technology and algorithms

### 2.1. Flocking Behavior

Social animals or insects in nature often exhibit a form of emergent collective behavior known as "flocking". The flocking model is a biologically inspired computational model for simulating the animation of a flock of entities. It represents group movement as seen in flocks of birds and schools of fish. In this model each individual makes movement decisions without any communication with others. Instead, it acts according to a small number of simple rules, dependent only upon neighboring members in the flock and environmental obstacles. These simple local rules generate a complex global behavior of the entire flock. The basic flocking model was first proposed by Craig Reynolds [7], in which he referred to each individual as a "boid", and consists of three simple steering rules that each boid needs to execute at each instance over time: separation (steering to avoid collision with neighbors); alignment (steering toward the average heading and matching the velocity of neighbors); cohesion (steering toward the average position of neighbors). These rules describe how a boid reacts to other boids' movement in its local neighborhood.

As shown in Fig. 1, in the circled area of Fig. 1(a)–(c), the boid's (located in the center of the small circle with gray background) behavior shows how a boid reacts to other boids' movement in its local neighborhood. The degree of locality is determined by the range of the boid's sensor. The boid does not react to the flock mates outside its sensor range. These rules of Reynolds' boid flocking behavior are sufficient to reproduce flocking behaviors on the computer.

It has been shown, however, that these rules alone are not sufficient to simulate flocking behavior in nature. Our early experiments [3] indicate these three rules in Reynolds's flocking model will eventually result in all boids in the simulation forming a single flock. It cannot reproduce the real phenomena in nature: birds or other herd animals not only keep themselves within a flock that is composed of the same species or the same colony creatures, but also keep two or multiple different species or colony flocks separated. To simulate this natural phenomenon, we proposed a new Multiple Species Flocking (MSF) model [3] to model multiple species bird flock behaviors. In the MSF model, in addition to these three basic action rules in the Flocking model, a fourth rule, the feature similarity rule, is added into the basic action rules of each boid to influence the motion of the boids. Based on this rule, the flock boid tries to stay close to these boids that have similar features and stays away from other boids that have dissimilar features. The strength of the attracting force for similar boids and the repulsion force for dissimilar boids is inversely proportional to the distance between the boids and the similarity value between the boids' features. The addition of this rule allows the use of flocking behavior to organize groups of heterogeneous boids into homogeneous subgroups.

### 2.2. MSF document clustering algorithm

The document clustering algorithm that we used in our research was originally described in [3]. This approach treats documents as boids and uses the MSF model to cluster based on a similarity comparison between documents. In the MSF clustering algorithm, each document is summarized by a feature vector. A similarity matrix is then built for reference through calculation of the cosine distance between each document and all other documents. Once the matrix is calculated all documents are given a random position and velocity in a two-dimensional plane. The boids that share similar document vector features (same as the bird species and colony in nature) will automatically group together and became a boid flock. Other boids that have dissimilar document vector features will stay away from this flock. After several iterations, the simple local rules followed by each boid result in generating complex global behaviors of the entire document flock. Eventually a document clustering result emerges.

In the MSF model implementation, we use the following mathematical equations to illustrate the four action rules for each boid:

Alignment Rule:

$$d(P_x, P_b) \leq d_1 \cap d(P_x, P_b) \geq d_2 \Longrightarrow \overrightarrow{v_{ar}} = \frac{1}{n}\sum_{x}^{n} \overrightarrow{v_x} \tag{1}$$

where $v_{ar}$ is velocity driven by alignment rule, $d(P_x, P_b)$ is the distance between boid $B$ and its neighbor $X$, $n$ is the total number of boid $B$'s local neighbors, $v_x$ is the velocity of boid $X$, $d_1$ and $d_2$ are pre-defined distance values and $d_1 > d_2$.

Separation Rule:

$$d(P_x, P_b) \leq d_2 \Longrightarrow \overrightarrow{v_{sr}} = \sum_{x}^{n} \frac{\overrightarrow{v_x} + \overrightarrow{v_b}}{d(P_x, P_b)} \tag{2}$$

where $v_{sr}$ is velocity driven by the separation rule, $d_2$ is a pre-defined distance, $v_b$ and $v_x$ are the velocities of boid $B$ and $X$.

Cohesion Rule:

$$d(P_x, P_b) \leq d_1 \cap d(P_x, P_b) \geq d_2 \Longrightarrow$$
$$\overrightarrow{v_{cr}} = \sum_{x}^{n} \overrightarrow{(P_x - P_b)} \tag{3}$$

where $v_{cr}$ is velocity driven by cohesion rule, $d_1$ and $d_2$ are predefined distances and $\overrightarrow{(P_x - P_b)}$ calculates a directional vector point.

Feature Similarity Rule: The flock boid tries to stay close to other boids that have similar features. For the document clustering algorithm, the boid's feature is represented by the document vector. The strength of the attracting force is proportional to the distance between the boids and the similarity between the boids' feature values.

$$v_{ds} = \sum_{x}^{n} (S(B, X) \times d(P_x, P_b)) \tag{4}$$

where $v_{ds}$ is velocity driven by feature similarity, $S(B, X)$ is the similarity value between the feature of boid $B$ and $X$. The flock boid tries to stay away from other boids that have dissimilar features. The strength of the repulsion force is inversely proportional to the distance between the boids and the similarity value between the boids' features.

$$v_{dd} = \sum_{x}^{n} \frac{1}{S(B, X) \times d(P_x, P_b)} \tag{5}$$

where $v_{dd}$ is velocity driven by feature similarity.

In this research, rather than directly using the feature similarity rule, we nullified the alignment and cohesion rules when $S(B, X) < T$. $S(B, X)$ is the similarity value between the features of boid $B$ and $X$ and $T$ is the similarity threshold. Thus, for dissimilar boids, separation is the only active rule, causing them to repel one another.

As indicated in the MSF model, at the initial stage, each document vector is projected as a boid and randomly deployed in a 2D virtual space. All boids move at a constant speed throughout the simulation but each boid's direction changes at each step according to the flocking algorithm. Our earlier research indicated that the MSF clustering algorithm can provide more accurate document clustering results than the $K$-means and Ant colony Optimization algorithms [3].

To adapt the document flocking algorithm in a GPU SPMD environment, we implement the algorithm in two kernels (see Fig. 2). The first kernel creates a thread for each document boid pair ($n^2$ threads in total) and compares their locations in the 2D virtual space to determine if the distance between them is within the neighborhood threshold. If the distance is small enough, a document comparison is initiated. This comparison entails a reference to the cosine similarity matrix in global memory. If the distance value retrieved between the two documents is small enough, the documents are deemed similar and treat each other as flock mates. In every simulation step, each boid will determine its new velocity according to the rules listed in the MSF algorithm. All other boids that are considered in this boids' neighborhood will contribute to the modification of this boids velocity. Similar documents contribute to the final velocity of each using the separation, cohesion, and alignment rules discussed earlier. Dissimilar documents contribute to the final velocity of each using only the separation rule. Once each document's influence on the rest of the population is calculated, the second kernel is run. This kernel spawns $n$ threads, each updating the final velocity and position of a single document. Here we added some randomness to the simulation; fifteen percent of all final movement calculations are random. Adding a random element to the system ensures that documents suitably explore the solution space in search of other flock mates. From our observations, we can also see that without this element the system of birds will degrade to a few clusters of boids flying in circles. Each new velocity calculation (or random calculation) is normalized to have a standard magnitude, keeping all boids moving with constant speed. Additionally, limitations are in place in this kernel to prevent velocity direction from changing drastically in each generation. This forces each document to make gradual turns, exposing it to a larger number of neighbors and more accurately simulating the behavior of birds. When this kernel is finished executing, a generation is finished and the cycle begins again.

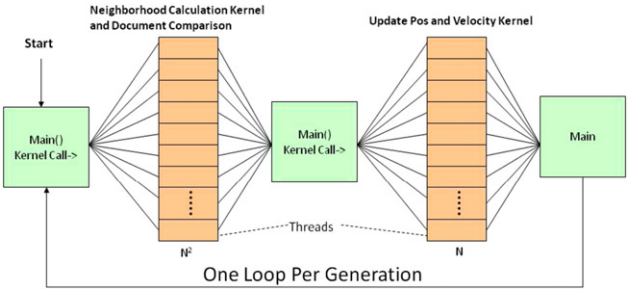



**Fig. 2.** Document flocking implementation in CUDA.

## 3. Implementations and evaluations

### 3.1. Experimental environment and data

In setting up our research we made an attempt to use low cost, commercially available equipment to help underline the cost and performance benefits of our approach. All tests that we performed were run on a single desktop workstation. This machine houses 4 GB of RAM and a single 3.0 GHz Intel core duo. We added an NIVIDA Geforce 8800 GTX graphics card to the workstation to enable the use of CUDA. The 8800 GTX contains 16 SIMD processors and has 768 MB of device memory. All experiments were run under Red Hat Enterprise Linux.

We compiled the documents used for clustering in our experiments from the TREC TIPSTER-2 data set. The TREC data set contains Associated Press news articles from 1988. We initially processed the documents by stripping out XML tags, stop words, numbers, and punctuation. We then stemmed the document content using a Porter Stemming algorithm [8]. Finally, we generated a term frequency list using TF-ICF [9], and normalized these frequencies for direct document comparison. Once document vectors were produced they were subjected to a dimensionality reduction; giving all document vectors a constant two-hundred dimensions. These reduced-dimension vectors were then used to build a cosine similarity matrix.

### 3.2. Challenges and solutions

One fundamental challenge of programming in CUDA is adapting to the Single Program Multiple Data (SPMD) paradigm, which differs from traditional parallel paradigms in that multiple instances of a single program act on a body of data. Each instance of this program uses unique offsets to manipulate pieces of that data. Data parallelism fits well in this paradigm while operational parallelism does not. Once the programming paradigm is understood, there are additional difficulties in using the CUDA language. Since each warp is executed on a single SIMD processor, divergent threads in that warp can severely impact performance. To take advantage of all eight processing elements in the multiprocessor, a single instruction is used to process data from each thread. However, if one thread needs to execute different instructions due to a conditional divergence, all other threads

```
texture<float2> tex_posArray;
cudaArray *array = NULL;
void setupTexture(int x, int y) {
    tex_posArray.filterMode = cudaFilterModePoint;
    tex_posArray.normalized = false;
    cudaChannelFormatDesc channelDesc = cudaCreateChannelDesc(sizeof(float)*8,
    sizeof(float)*8, 0, 0, cudaChannelFormatKindFloat);
    cudaMallocArray(&array, &channelDesc, x,y);
}
```

**Fig. 3.** Document positions were stored in texture memory.

must effectively wait until the divergent thread rejoins them. Thus, divergence forces sequential thread execution, negating a large benefit provided by SIMD processing. Another limitation in CUDA is the lack of communication and consequently the lack of synchronization between blocks. This creates possible problems of data consistency, typical of parallel modification of singular values.

Currently, standard (or otherwise) *C* libraries are not available/compatible for use on the GPU. NVIDIA does however package some basic FFT and Linear Algebra libraries with the CUDA toolkit. In the future, more libraries could be written for CUDA (by users) as device functions to help streamline the development process. Debugging can be difficult in CUDA. A debug mode is available in the CUDA compiler which forces sequential execution on the CPU by emulating the GPU architecture. Although this mode is useful for most general types of debugging, some errors are not exposed. The emulator cannot detect any concurrency problems as its execution is sequential. Write and read hazard behavior is undefined during thread execution on the GPU, therefore the programmer must be cautious to avoid these errors. Read and write memory hazards occur when data are being written or accessed in an order which is not intended or defined. While running a kernel on the GPU, no access is provided to the standard output. This effectively turns the GPU into a black box when it comes to runtime behavior.

The largest constraint for us in our work was the shortage of fast, local shared memory on GPU. Due to the large size of document information and our initial method of document comparison we were forced to make frequent reads from GPU global device memory [10]. Although the GPU global device memory is much faster than computer RAM, this global device memory is not cached and has a delay of hundreds of clock cycles per read associated with it compare to GPU local memory [11]. We tried to reduce the impact of this problem by caching some document terms in shared memory for fast access. After our initial efforts [10] implementing a GPU document flocking algorithm, we decided to revise our approach by creating a document similarity matrix and merely reading that value from device memory for each document comparison. In this reported experiment, the similarity matrix is pre-calculated on the CPU. This approach proved to be vastly superior to our initial approach demonstrating that our instinct to perform as much calculation inside the kernel was not suitable for our data. Another problem we encountered in our research was the requirement of thread divergence in the implementation. Certain conditional statements could not be avoided. This seemed to have some effect on the performance, but not a significant one when compared with the performance degradation of global memory reads. In an effort to improve the speed of position retrieval and distance calculation, all document positions were stored in texture memory. The texture memory is initialed by code shown in Fig. 3. This design decision did improve the performance of our implementation on the GPU, but it put a hard limit on the number of documents that could be compared.

## 4. Final evaluation and validation of results, total benefits, limitations

The MSF document clustering algorithm was implemented in CUDA and was run on the GPU of our test workstation. For performance comparison purposes, a similar but sequential implementation was written in *C* and run on the CPU of the same machine. The CPU version MSF clustering implementation only used a single core for computing. We conducted testing on differently sized sets of documents. Document set sizes ranged from 200–3600 documents in increments of 200 documents. We tested each set 30 times and then averaged the runtime of each. We used randomly generated values for the initial position and the movement direction of each document for each test to prevent accidental initial seeding optimization (seeding was based on CPU clock time). Each test ran the flocking simulation for four-hundred generations. This means that documents updated their positions and directions four-hundred times based on other documents present in their neighborhood. Based on our observations, four-hundred generations was an adequate number to allow the documents to converge into stable clusters of similar documents (Fig. 4).

The flock parameters of each simulation were identical. The "flying" space of the documents was $300 \times 300$ square units. This size space was selected to allow adequate room for each document to move. Each document had a static neighborhood radius of 30 units and a constant speed of two units per generation. These parameters were selected based on the flying space size and the observed behavior of the flocks. Each document had a maximum limit of a 0.35 radian deviation from its previous moving direction. The methods for selecting these parameters is described in [3]. We gave each rule a weight that encouraged system behavior typical of flocking birds. We assigned a weight of 0.33 to the alignment rule, 0.66 to the separation rule, and 0.33 to the cohesion rule. The document feature vector similarity threshold $T$ was 1.20. This value was selected based on experimental observation. It was small enough to clearly differentiate groups in the flock while not being so small that it prevented flocking altogether. As mentioned before,

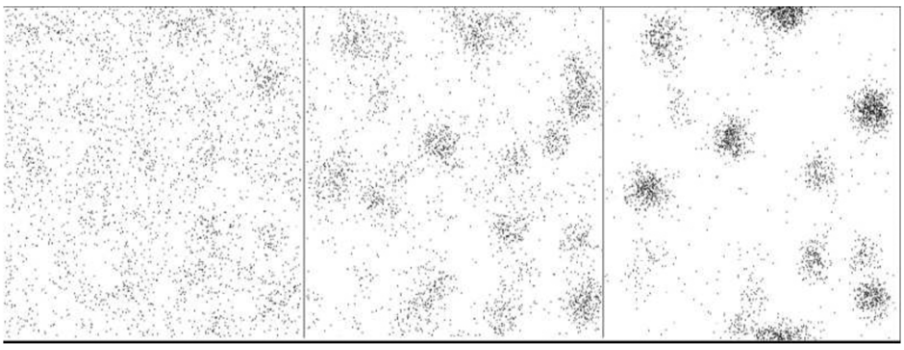


**Fig. 4.** Three thousand documents flocking at 44, 200, and 400 generations.

a random element was introduced at every generation. Through observation, we felt that fifteen percent random movement was high enough to keep the system from stagnating, while low enough to ensure flocking behavior persisted.

Through our experiments we observed that document flocking on the GPU has a runtime that is significantly smaller than its CPU counterpart (see Figs. 5 and 6). From Fig. 5, we can see the computing time running on GPU and CPU for clustering document data below 500 documents are not significantly different. For document numbers ranging from 500 to 3500 documents, the computing time on the GPU remains below 20 s, which is acceptable for most human analysts. However, the computing time on a CPU are exponentially increased to more than 900 s, which is 15 min. The equal scaling of the *Y* axle in Fig. 5 makes it difficult to compare the changing trend of computing time on GPU and CPU for different kinds of data size. In Fig. 6, we used variable scaling in the running time axle to show the same experiment results. As shown in Fig. 6, both GPU and CPU computing time change trends have a similar change pattern when increasing the clustering document number. It indicates the time complexity of flocking algorithm does not change after we implement the flocking algorithm on a high parallel gnu platform. From GPU speedup chart in Fig. 7, we observed that with 200 documents the GPU implementation is roughly 36 times faster than the CPU version. Initially, as we increased the number of documents in our test set, the improvement increased. For 1000 documents, we saw an improvement of nearly sixty times over the CPU. We hypothesize that the lower performance that occurs with less than 1000 documents is caused by not all processing elements on the GPU being utilized. From 1200 to 3400 documents however, the improvement degrades. As is clear in Fig. 7, after 1000 documents, performance degrades nearly linearly. While we have done no direct testing, we think this could be due to an increase in global memory reading requirements as the population size grows. One thing we need point out is, in this research, with respect to the performance comparison between GPU and CPU implementations, the CPU implementation is a single core version instead of utilizing the multi-core capability of the CPU. To utilize the multi-core capability, the CPU code needs to be parallelized and the parallelizing overhead will vary based on how optimized the parallel CPU codes are. To avoid reader's suspicions about our experiment trying to "Compare heavily optimized GPU code to unoptimized CPU code" [12], we intentionally only use a single core CPU implementation. We believe, the best scenario for optimized multi-core CPU code will be no parallelizing overhead at all. In that case, even for a six-cored CPU, the GPU can still achieve 5–10 times speedup compare to the CPU. Although this paper mainly discussed the single GPU data clustering solution, however, because of the distribution character of the flocking based algorithm, there will be no difficulty for interested readers in implementing such a solution on a multi-GPU platform.

## 5. Future directions

In future work, larger data sets could be investigated. Clustering hundreds of thousands or even millions of documents on a workstation quickly is currently an unsolved research problem but the GPU may provide hope. This paper mainly discussed the single GPU data clustering solution. Our next step research will mainly focus on distributing the document flocking algorithm across many GPUs to substantially improve the number of documents that can be handled during a simulation, possibly allowing multi-millions of documents to be clustered quickly. The currently released NVIDIA Tesla GPU architecture has many times the amount of device memory as the 8800 GTX we used here. These additional capabilities can greatly enhance the already high performance

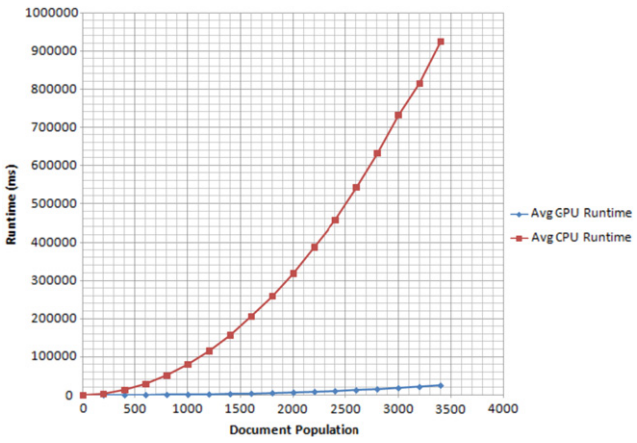



**Fig. 5.** Document flocking runtime, CPU vs. GPU.

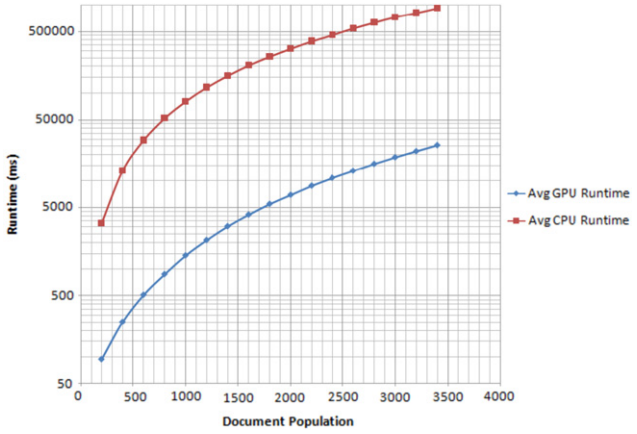



**Fig. 6.** Document flocking runtime, CPU vs. GPU.

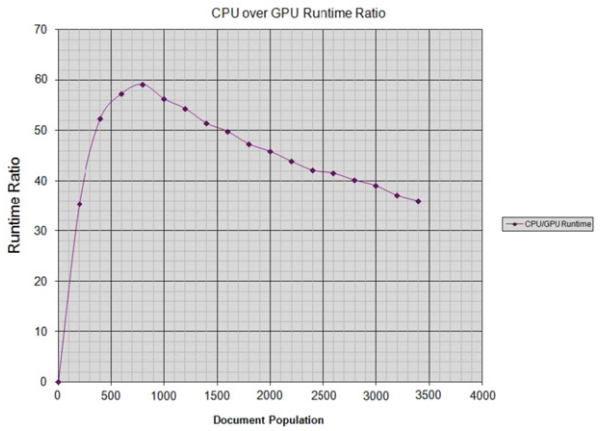



**Fig. 7.** GPU over CPU runtime ratio.

we saw in our tests. The GPU CUDA we used in this experiment is still in its early version. The recently released CUDA 4.0 and Nsight development environment have significantly reduced the difficulties in programming with CUDA. We believe in the near future, more applications will adopt the CPU+GPU computing model to reduce the system computing time. We hope that through this research we can provide some guidance, insight, and inspiration to other researchers who deal regularly with complex data parallel algorithms.

## Acknowledgments

This work was funded in part by the Lockheed Martin Corporation Shared Vision program and Oak Ridge National Laboratory LDRD Seed Money fund. The views and conclusions contained in this document are those of the authors. This manuscript has been authored by UT-Battelle, LLC, under contract DEAC05-00OR22725 with the US Department of Energy.

## References

[1] M.R. Anderberg, Cluster Analysis for Applications, Academic Press, Inc., New York, 1973.
[2] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Computing Surveys 31 (1999) 264–323.
[3] X. Cui, J. Gao, T. Potok, A flocking based algorithm for document clustering analysis, Journal of Systems Architecture (2006).
[4] M. Steinbach, G. Karypis, V.A. Kumar, Comparison of document clustering techniques, in: KDD Workshop on Text Mining, 2000.
[5] S.Z. Selim, M.A. Ismail, *K*-means-type algorithms: a generalized convergence theorem and characterization of local optimality, IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-6 (1984) 81–87.
[6] R. Farivar, D. Rebolledo, E. Chan, R. Campbell, A parallel implementation of *K*-means clustering on GPUs, in: WorldComp 2008, Las Vegas, Nevada, July 2008.
[7] C.W. Reynolds, Flocks, herds, and schools: a distributed behavioral model, Computer Graphics (ACM) 21 (1987) 25–34.
[8] M.F. Porter, An algorithm for suffix stripping, in: Readings in Information Retrieval, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1980, pp. 313–316.
[9] J. Reed, et al. TF-ICF: a new term weighting scheme for clustering dynamic data streams, in: Proc. Machine Learning and Applications, ICMLA'06, 2006, pp. 258–263.
[10] J.S. Charles, T.E. Potok, R. Patton, X. Cui, Flocking-based document clustering on the graphics processing unit, DOE Office of Science Journal of Undergraduate Research VIII (2008).
[11] NIVIDA, CUDA: compute unified device architecture, NIVIDA, Version 1.1, 2007. http://developer.NIVIDA.com/cuda.
[12] S. Pakin, Ten ways to fool the masses when giving performance results on GPUs, in: HPCwire, December, 2011.

Dr. Xiaohui Cui is the faculty of the NYIT computer science department and the staff scientist of Oak Ridge National Laboratory of Department of Energy. His research interests include swarm intelligence, GPU computing, agent based modeling and simulation, cyber security, GIS and transportation, emergent behavior, complex system, high performance computing, social computing, and information retrieval. His research programs have been supported by Office of Navy Research, Department of Homeland Security, Defense Threat Reduction Agency, Department of Energy and Lockheed Martin Company. He and his researches have been reported by MSNBC, New Scientist magazine etc. In 2008 and 2009, he received the Department of Energy Outstanding Mentor Award andthe Significant Event Award.