Applying GPUs for Smith-Waterman Sequence Alignment Acceleration

Phong H.Pham
Tan N. Duong
Ngoc M.Ta
High Performance Computing Center
Hanoi University of Science and Technology
Hanoi, Vietnam
Email: {phongph.hut, dn.nhattan,
ngoctm255}@gmail.com

Duc H.Nguyen
Thuy T.Nguyen
Hung D.Le
Department of Information Systems
Hanoi University of Science and Technology
Hanoi, Vietnam
Email: {ducnh, thuynt}@soict.hut.edu.vn

Cuong Q.Tran
Ministry of Police, Hanoi, Vietnam

Abstract—**The Smith-Waterman algorithm is a common local sequence alignment method which gives a high accuracy. However, it needs a high capacity of computation and a large amount of storage memory, so implementations based on common computing systems are impractical. Here, we present our implementation of the Smith-Waterman algorithm on a cluster including graphics cards (GPU cluster) – swGPUCluster. The algorithm implementation is tested on a cluster of two nodes: a node is equipped with two dual graphics cards NVIDIA GeForce GTX 295, the other node includes a dual graphics cards NVIDIA GeForce 295 and a Tesla C1060 card. Depending on the length of query sequences, the swGPUCluster performance increases from 37.33 GCUPS to 46.71 GCUPS. This result demonstrates the great computing power of GPUs and their high applicability in the bioinformatics field.**

Keywords: local sequence alignment; smith-waterman; cuda; gpu cluster.

I. INTRODUCTION

Proteins and DNAs that have a significant biological relationship to one another often share only isolated regions of sequence similarity. For identifying relationships of this nature, the ability to find local regions of optimal similarity is advantageous over global alignments that optimize the overall alignment of two entire sequences. Sequence alignment is one of the typical sequence data analysis problems in bioinformatics. It is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Figure 1 is an example of the result of a two sequences alignment where some gaps are inserted into the first sequence to achieve the biggest region of similarity between them.

```
Global   FTFTALILLAVAV
         F--TAL-LLA-AV

Local    FTFTALILL-AVAV
         --FTAL-LLAAV--
```

Figure 1. Example of an alignment for two sequences

Very short or very similar sequences can be aligned by hand. However, most interesting problems require the alignment of lengthy, highly variable or extremely numerous sequences that cannot be aligned solely by human effort. Instead, human knowledge is applied in constructing algorithms to produce high-quality sequence alignments, and occasionally in adjusting the final results to reflect patterns that are difficult to represent algorithmically (especially in the case of nucleotide sequences). Computational approaches to sequence alignment generally fall into two categories: global alignments and local alignments. Calculating a global alignment is a form of global optimization that "forces" the alignment to span the entire length of all query sequences. By contrast, local alignments identify regions of similarity within long sequences that are often widely divergent overall. Local alignments are often preferable, but can be more difficult to calculate because of the additional challenge of identifying the regions of similarity. A variety of computational algorithms have been applied to the sequence alignment problem, including slow but formally optimizing methods like dynamic programming, and efficient, but not as thorough heuristic algorithms or probabilistic methods designed for large-scale database search.

.Currently, there are many researches of resolving the sequence alignment problem, mostly focus on three main branches: methods using point matrixes, dynamic programming methods and the BLAST method. For the global sequence alignment problem, the most developed algorithm is the Needleman-Wunsch [1]. This is a global sequence alignment method based on dynamic programming, to calculate points for the alignment process, using the substitution matrix PAM250 or BLOSUM62 for protein sequences. This method ensures, from the mathematics side of view that finding an optimal answer with a specific mechanism is possible, but it has a large amount of calculations. On the other hand, the BLAST algorithm [2] allows searching subsequences (a sequences database) which are similar to the given sequence (the query sequence). BLAST uses the heuristic approach so the speed

DOI: 10.5176_2010-2283_1.2.56

is remarkably fast when performing for gene banks. This has made BLAST the most popular tool in bioinformatics. Although the speed is lower than that of the BLAST algorithm, but with a higher accuracy, the Smith – Waterman algorithm is considered as one of the most popular algorithms of solving the local sequence alignment problem. Since the execution of the Smith – Waterman algorithm requires a large amount of calculation and storage memory, due to huge biological data together with the dynamic programming algorithm, the implementation process is unacceptable for common computing systems. Another tendency is being developed for the next generation computers: multi-core structures, which help solving a lot of problems requiring large computing power, including bioinformatics. In this paper, we perform parallelism for the Smith Waterman algorithm on a multi-core cluster equipped with NVIDIA's graphics cards (called a GPU cluster). Results of this experiment have shown that the speed of implementing the algorithm has increased significantly compared to executions on other common computing environments. This has proved the extremely high computing power of graphics cards and their applicability in bioinformatics.

II. The Smith-Waterman Algorithm and Related Works

A. *The Smith-Waterman Algorithm*

In the Smith – Waterman algorithm, the alignment process is executed by aligning every pairs of characters in the two sequences. The point for each pair depends on the followings: two characters are a match, two characters are a mismatch and points for adding or removing gaps (or penalties). The result of local alignment is that we can find out segments having the highest similarity between two sequences. The algorithm is based on the dynamic programming method to calculate the point of the alignment process. The Smith – Waterman algorithm is developed to identify the optimal local alignment answer of two biological sequences by grading the similarity using the dynamic programming method. Suppose that two sequences S_a and S_b have the following lengths: l_a and l_b, the Smith – Waterman algorithm calculates the match points of two sequences, using the matrix $H(i,j)$ of two subsequences S_{ai}, S_{bj} (sequences end at points i,j of S_a, S_b ; $0<i<l_a$; $0<j<l_b$). $H(i,j)$ is calculated by the following recursive formula:

$$E(i,j) = \max\{E(i,j-1) - g,\ H(i,j-1) - g - e\}$$
$$F(i,j) = \max\{F(i-1,j) - g,\ H(i-1,j) - g - e\}$$
$$H(i,j) = \max\{0, E(i,j), F(i,j), H(i-1,j-1) + matScore[S_a(i), S_b(j)]\}$$
$$H(i,0) = H(0,j) = E(i,0) = F(0,j) = 0;\ 0<i<l_a,\ 0<j<l_b$$

Figure 2. Calculation of the matrix H

Where *matScore* is the scoring matrix, g is the penalty for an opening gap, e is the penalty for lengthen gaps. The maximum point of the local alignment is the maximum value in the matrix H. Figure 3 describes the process of calculating the matrix H. In this figure, we can see that each cell in the matrix is calculated based on values of three other cells. If we number sub-diagonals of the matrix H, then each cell on the sub-diagonal i^{th} depends on cells of the sub-diagonals $(i-1)^{th}$, $(i-2)^{th}$. Therefore, cells on the same sub-diagonal do not depend on each other and they can be calculated in parallel.

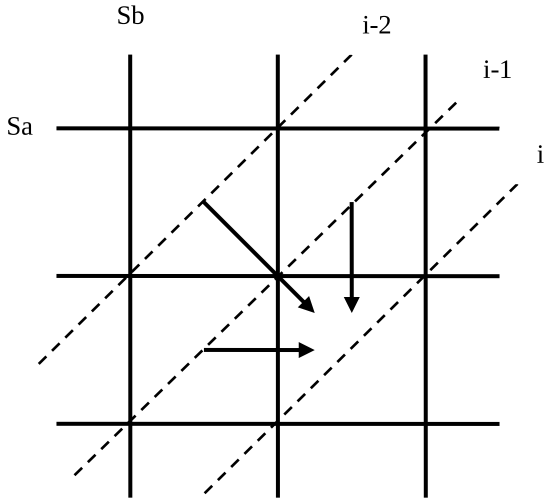

Figure 3. Illustration of calculating one cell on the sub-diagonal i^{th}

B. *Related Works*

It can be observed that the Smith – Waterman algorithm requires three matrixes of the size mxn where m, n are lengths of two sequences needed to be compared. The dynamic programming algorithm also requires all values of these three matrixes to be completely filled. With such required amount of calculation and memory storage, the algorithm becomes less practical in common cases when we need to align a query sequence and a large database of correlative sequences with high lengths.

To achieve high results, most implementations of this algorithm use parallel processing computer architectures. In [4], [5], authors have paralleled the Smith – Waterman algorithm on general purpose processors according to the SIMD architecture. Their result has shown that the implementation speed has increased by 6 times. Manavski SA and Valle G, in [6], have paralleled the algorithm on a system equipped with 2 graphics cards GeForce 8800 GTX of NVIDIA and obtained the result of 3.5 GCUPS (the number of billions of cells updated per second). In comparison with the previous best implementations, on one single GPU and on the SIMD architecture, the implementation speed has increased about from 2 to 30 times. This implementation has definitely marked the importance of the GPU multi-core architecture in bioinformatics problems. Inheriting this method of Farrar [5] and fully exploiting the computing power of processing cores, in [7], authors have asserted that SWPS3 is the fastest vectorized installation of the Smith – Waterman algorithm on Cell/BE and x86/SSE architectures, with a computer system using Quad core Pentium, it can reach 15.7 GCUPS. In that implementation, the algorithm was installed based on the multi-core architecture and it was paralleled by the multi-thread method. In some situations, the alignment speed of SWPS3 is calculated to be the same as that of BLAST algorithm, the fastest heuristic algorithm at the moment.

As mentioned above, another direction which is intensively considered in the installation of Smith – Waterman algorithm is the application of the great parallelism computing power of GPU in solving problems which require a huge amount of calculation. The paper [8] presents about CUDASW++, another installation of the Smith – Waterman algorithm on graphics cards of NVIDIA. The version running on one single GPU reaches the speed of about 10 GCUPS with NVIDIA GeForce GTX 280 graphics cards and the one running on multi-GPUs can reach 16 GCUPS. These results have shown a much greater performance compared to SWPS3 or NVBI-BLAST and demonstrated a high applicability of the GPU multi-core architecture in solving the sequence alignment problem.

C. *Our Approach*

With the exponential development of biological sequences database, the necessity of high performance computing methods is considered to solve the bioinformatics problems, especially the sequence alignment problem. Recent results using GPU to implement the Smith – Waterman algorithm have shown an outstanding performance compared to other methods. However, that execution can only be installed on one single GPU or on one computer equipped with multi-GPUs, and there is not any installation executed on a cluster, where nodes equipped with multi-GPUs. In this paper, we implemented the Smith–Waterman algorithm on a cluster including multi-GPUs - GPUCluster. These results which have been compared to the previous effective implementations show a remarkable improvement of performance and demonstrate the great computing power, a high applicability of GPUCluster in bioinformatics problems.

III. GPU CLUSTER

A. *Gpgpu and Cuda*

Graphics Processing Units (GPUs), which commonly accompany standard Central Processing Units (CPUs) in consumer PCs, are special purpose processors designed to efficiently perform the calculations necessary to generate visual output from program data. CUDA [9] is a software which supports to develop applications for multi cores graphics processing unit of NVIDIA, including the device controller, the application development tool (the programming language on GPUs and the compiler). CUDA is an extension of the language C. A CUDA program includes one or a few special piece of code, called parallel *kernels*. These kernels can be executed in parallel on the large number of *threads* on GPUs. Since threads execute the same code of the kernel, only be identified by the Ids of threads; so to increase the application performance, programmers need to apply data parallel techniques (data is divided into small parts and they are assigned to threads for implementing).

Programmers can determine the number of threads when activating the parallel kernel. To ensure that the program do not depend the hardware (the number of streaming multiProcessor - SM on GPUs), threads are divided into small groups which are executed on the same SM, called *thread blocks*, these blocks are also designed to a *grid*. When a grid is executed, the CUDA scheduler, based on the hardware, will identify the number of thread blocks which are concurrently executed. If this number is smaller than the number of SMs of GPUs, the whole grid will be executed concurrently; on the contrary, the thread blocks will be automatically divided into different parallel session. Beside the work of dividing data into different threads and organizing the grid and thread blocks, accessing data is also an important problem when programming on GPUs. And memory is hierarchically organized for effective usage.

With the ability to perform data parallelism on such a lot of threads, GPU is an appropriate choice to implement the Smith-Waterman algorithm, where each thread can calculate one cell on sub-diagonals of the matrix H.

B. *Gpucluster*

In fact, we cannot put many graphics cards into a PC, only a few GPUs. Practical problems require very high capacity of computing power, due to the very quick increase of the size of input data. So, to combine the computing power of many GPUs for a problem, we have to put multi-GPUs on different nodes. A GPU cluster will solve this. It is a cluster of nodes in which each node is equipped with one or more GPUs. Such a system includes hardware components such as CPU, GPU and to connect nodes we need to add a network connection such as Gigabit Ethernet. Required software installed on a GPU node includes: operating system, GPU driver in each node and parallel programming interfaces such as MPI.

In recent decades, there have been a lot of GPU cluster systems deployed as installation of GraphStream [10], but they are only virtual systems. A number of projects deployed GPU computing nodes such as: GPU Cluster "DQ" 160 nodes at the LANL- [11], and "QP" 16 nodes at NCSA [12], both based on NVIDIA's QuardroPlex technology. These installations mainly provide experimental results in the production field using high performance.

To harness the great computing capability of GPU cluster, we use the idea of plunging the CUDA framework in a message passing interface environment – ***MPI***. Each of nodes inside the cluster has its own task of sending data and intensive parallel tasks to GPU, making CPU free for performing network communication between nodes. To compile and run MPI programs, it is not difficult if using NVIDIA's compiler - *nvcc* to compile the entire mixed CUDA and MPI code together. Here, we mix MPI code into CUDA source files, since CUDA is an extension of C language and the compiler *nvcc* wraps the compiler *mpicc*.

The next section will present our implementation of the Smith-Waterman algorithm on GPUCluster system and some experimental results.

IV. Implementing the Smith-waterman Algorithm on Gpu Cluster

A. Strategy

With the description of the SW algorithm above, along with analysis of factors which can be paralleled in the process of calculating the matrix, we employ two approaches on GPU cluster with two data parallel levels, proposed by Yongchao Liu, Douglas L Maskell and Bertil Schmidt in [8]. In the first level, the algorithm can perform alignment with different input sequences in parallel. In second one, the calculation of the matrix H can be simultaneously executed for values of cells on sub-diagonals. Suppose that alignment of two sequences is a single task, the first level can be considered as *inter-task parallelism*, the second one is considered as *intra-task parallelism* (splitting a task into several sub-tasks).

Inter-task parallelism: each task is assigned to an execution thread. In one block of threads, tasks are simultaneously performed by different threads.

Intra-task parallelism: each task is assigned to a block. Each thread inside a block will calculate one cell on the i^{th} sub-diagonal based on values of cells on two sub-diagonals $(i-1)^{th}$ and $(i-2)^{th}$. After finishing the calculation, values of sub-diagonal vectors are swapped to calculate values of the next diagonal.

Inter-task parallelism requires much more memory, but it gives a better performance, so it is suitable for the alignment of sequences with short lengths. In contrast, intra-task parallelism does not require much memory and it has a lower performance, it is suitable for longer sequences. To separate these two methods, we use a threshold value of the sequence length to decide which method is used.

B. Paralleling the Algorithm on Gpucluster

Based on the method of parallelization with two levels as mentioned above, we distribute data on GPU cluster system for implementing on each GPU of each node. The cluster system uses a shared data directory which is synchronized between nodes. This directory stores biological sequences data. Suppose that the GPU cluster system includes *n_client* nodes, each node is equipped with *n_device* GPUs. First, the data sequence database is divided into *n_client* parts based on the size of the data. To avoid conflicts when accessing files containing sequence data, we use a data locker *db_lock* and a data-status-record *db_stat* for each access of nodes. The data locker and the data-status-record are unique and transferred between nodes. When a node finds out that data is not locked, it will read the data-status-record to determine the previous position to continue loading data into memory. When reading is complete, the node will update the data-status-record and unlock the data locker.

```
Initialize MPI, get the number of nodes;
db_block_size = totalDBSize / n_client;
WHILE data is still locked
      Wait;
ENDWHILE
Create data locker;
Read record of data status to determine data
position;
temp_size = 0;
WHILE temp_size < db_block_size &&dbIsNotEmpty
       Get (Seq, SeqName, SeqLen) in DataFile;
       temp_size += SeqLen;
ENDWHILE
Save the data status record;
Unlock data;
```

Figure 4. Pseudo code of distributing data into nodes

Before data is divided into GPUs memory, it is arranged in ascending order of sequence lengths. This arrangement aims at two purposes. The first one is to separate data into two blocks which are handled in two ways (as described above). This process is controlled by a runtime parameter *threshold* (if the sequence length < *threshold*, it is assigned to the first block, otherwise it belongs to the second block). These blocks continue being divided into *n_device* parts corresponding to GPUs, thus each GPU will only have to handle similar blocks of data in both ways. The second purpose is that threads of the same block will align sequences of approximately equal lengths, so their runtime is approximately similar; this increases the performance of the implementation.

The program consists of two *kernels* performing two parallel algorithms with two levels on the GPU cluster (one kernel corresponding to inter-task parallelism and the other corresponding to intra-task parallelism). After loading sequence data into memory of each GPU, the program will call kernels to align sequences. Then results are saved in the memory of nodes and each node will write the results into temporary files. At node 0, the program will collect results performed on all nodes. Below is pseudo code for the implementation of the kernels on the GPU Cluster:

```
/********************ASSUMPTIONS****************
nThread: the number of threads executed in
parallel;
nSM: the number of multi-processors per GPU;
InterSeqNo: the number of sequences whose lengths
are less than the value threshold per GPU
IntraSeqNo: the number of sequences whose lengths
are more than the value threshold per GPU
*********************************************/
n_batch = InterSeqNo/threads;
WHILE n_batch > 0
   n = min(n_batch, nSM);
   DECLARE grid of n blocks;
   DECLARE each blocks of nThread threads;
   CALL  inter_kernel  WITH  interSeqs,  interSeqNo
    RETURN d_result;
   DECREASE n_batch BY n;
ENDWHILE
IF intraSeqNo > 0
   maxSeqOnePass = 256;
   n_batch = intraSeqNo;
   WHILE n_batch > 0
     n = min(n_batch, maxSeqOnePass);
     DECLARE grid of n blocks;
     DECLARE each blocks of nThread threads;
     CALLintra_kernel  WITH  intraSeqs,  intraSeqNo
      RETURN d_result;
     DECREASE n_batch BY n;
   ENDWHILE
ENDIF
transferToHostResult(d_result);
```

Figure 5. Pseudo code of implementing the kernels on GPU Cluster

V. EXPERIMENTAL RESULTS

To remove the dependency on the query sequences and the databases used for the different test, *cell updates per second* (CUPS) is a commonly used performance measure in bioinformatics. CUPS presents the number of cells of the matrix H calculated per second, including the calculation of intermediate values of the matrix E, F). The formula (1) calculates CUPS values of one sequence alignment answer:

$$cups = qLen * dbLen/t \quad (1)$$

Where *qLen* is the length of a query sequence, *dbLen* is the length of a subject sequence; *t* is the runtime of the program. The value *t* includes the time of loading data from main memory to device memory, the time of calculation on GPUs and the time of transferring results to CPU.

In our test, we used a set of query sequences of lengths which are from 100 to 5000, the biological sequences database *UniProt* release 2010_05 - Apr 20, 2010 which includes 516,080 sequences and 181,676,505 amino acids. With this database and the value *threshold* is 3072, there are up to 515,472 sequences which are aligned by intra-task parallelism and 608 others are aligned by inter-task parallelism. Experimentation of the swGPUCluster is tested on two nodes Node0 and Node1, using multi-GPUs (three dual cards GTX295 – 6 GPUs, one card Tesla C1060 – 1 GPU). With our GPU cluster system, the maximum performance is achieved when the block size *threads* = 256 and the grid size *blocks* = 30 (the number of streaming multiprocessors of GPU). The performance of the swGPUCluster increases according to lengths of query sequences, from the minimum value 37.328 GCUPS to the maximum value 46.706 GCUPS. This result is described in the table 1.

We have compared the results of the swGPUCluster to other solutions implementing the SW algorithm such as: cudaSW++ or swps3. cudaSW++ was tested on one GTX295 GPU. Its result shows that the minimum performance is 8.387 GCUPS and the maximum is 9.232 GCUPS. In comparison to the cudaSW++ on one single GPU, the speed of implementing the swGPUCluster is about 4.4 to 5 times faster than the cudaSW++.

Another comparison of performance is performed with the swps3 implementation. The swps3 was tested on x86/sse2 platform including one node equipped with a processor Core 2 Quad Q8400 2.66 Ghz (4 cores), 8GB RAM with one thread or four threads. The performance of the swGPUCluster is about 13.8 to 22.8 times faster than the swps3 x86/sse2-single-core, and it is approximately 3.4 to 11.4 times faster than swps3 x86/sse2 multi-cores, as shown in figure 6.

TABLE I.
RESULTS OF THE IMPLEMENTATION OF THE SMITH-WATERMAN ALGORITHM ON GPU CLUSTER

Query	Length	Time(s)	GCUPS
P02232	144	0.779393	37.328
P01111	189	0.944164	39.202
P14942	222	1.078783	40.795
P07327	375	1.721378	42.945
P25705	553	2.532260	43.419
P21177	729	3.224185	44.907
P27895	1000	4.337310	45.849
P07756	1500	6.456398	46.065
P04775	2005	8.579664	46.355
P19096	2504	10.681426	46.530
P0C6B8	3564	15.174918	46.612
P08519	4548	19.348871	46.653
P33450	5147	21.894938	46.690
Q9UKN1	5478	23.294546	46.706

VI. CONCLUSION

In this paper, we present the swGPUCluster – an implementation of the Smith-Waterman sequence alignment algorithm on a GPU cluster system consisting of two nodes equipped with multi-GPUs (3 dual cards GTX295 - 6GPUs and one card Tesla C1060 - 1GPU). With the test input which is biological sequences database *UniProt* version 2010_05 - Apr 20, together with the optimal configuration set, the performance of the swGPUCluster increases with the length of query sequences from the minimum value of 37,328 GCUPS to the maximum value of 46,706 GCUPS. The swGPUCluster gives a significantly better performance than the implementation previously installed on GPU or on multi-core architectures such as swps3 or cudaSW++. Our results show a high applicability of GPUs to speed up the implementation of algorithms in bioinformatics, if we well exploit characteristics of computing hardware. The outstanding performance also shows that the performance of GPUs increases much faster than the performance of multi-core CPUs.

REFERENCES

[1] www2.cs.uh.edu/~zhenzhao/Review/alignment.htm

[2] http://blast.ncbi.nlm.nih.gov/Blast.cgi.

[3] http://en.wikipedia.org/wiki/SmithWaterman_algorithm.

[4] Rognes T, Seeberg E: "Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors" . *Bioinformatics* 2000 , 16(8)**:**699-706

[5] Farrar M: "Striped Smith-Waterman speeds database searches six times over other SIMD implementations" **.** *Bioinformatics* 2007 , 23(2)**:**156-161

[6] Manavski SA, Valle G: "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment"

[7] Szalkowski A, Ledergerber C, Krahenbuhl P and Dessimoz C: "SWPS3 – fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and ×86/SSE2" . *BMC Research Notes* 2008, 1:107.

[8] Yongchao Liu, Douglas L Maskell and Bertil Schmidt: "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units". *BMC Research Notes* 2009, 2**:**73.

[9] NVIDIA.http://www.nvidia.com/object/cuda_home_new.html

[10] (2009) GraphStream, Inc. website. [Online]. Available: http://www.graphstream.com/.

[11] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. McCormick, S. Buijssen, M. Grajewski, and S. Tureka, "Exploring weak scalability for FEM calculations on a GPU-enhanced cluster," Parallel Computing, vol. 33, pp. 685-699, Nov 2007.

[12] M. Showerman, J. Enos, A. Pant, V. Kindratenko, C. Steffen, R. Pennington, W. Hwu, "QP: A Heterogeneous Multi-Accelerator Cluster," in Proc. 10th LCI International Conference on High-Performance Clustered Computing, 2009. [Online]. Available:http://www.ncsa.illinois.edu/~kindr/papers/lci09_paper.pdf.

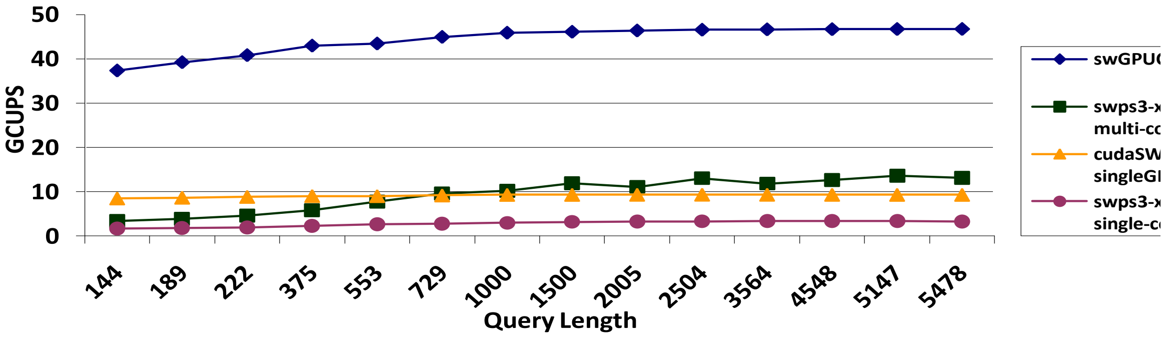

Figure 6. Comparison of performance of swGPUCluster with cudaSW++ and swsp3-x86/sse2.

Nguyen Thanh Thuy, Professor. Professor Thuy received the B.E. (1982), D.E. (1987) degrees in Computer Science from Hanoi University of Science and Technology. He is a professor, School of Information and Communication Technology. His current interests include reasoning, machine learning, artificial intelligence and HPC.

Nguyen Huu Duc, Ph D.
Dr Nguyen received his Ph.D. in computer science from the Japan Advanced Institute of Science and Technology in 2006. His primary interest is in the area of programming languages, specifically design and implementation of parallel programming languages for multicore/manicore architectures.

Pham Hong Phong
Phong received B.E degrees in computer science from Hanoi University of Science and Technology in 2009. His current interests include artificial intelligence, high performance computing and parallel computing on GPUs.

Duong Nhat Tan
Tan received B.E degrees in computer science from Hanoi University of Science and Technology in 2010. Currently, Tan's interests are high performance computing, parallel computing on GPUs and could computing.