GPU-BLAST: using graphics processors to accelerate protein sequence alignment

Panagiotis D. Vouzis[1] and Nikolaos V. Sahinidis[1,2,*]

[1]Department of Chemical Engineering and [2]Lane Center for Computational Biology, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Associate Editor: Burkhard Rost

ABSTRACT

Motivation: The Basic Local Alignment Search Tool (BLAST) is one of the most widely used bioinformatics tools. The widespread impact of BLAST is reflected in over 53 000 citations that this software has received in the past two decades, and the use of the word 'blast' as a verb referring to biological sequence comparison. Any improvement in the execution speed of BLAST would be of great importance in the practice of bioinformatics, and facilitate coping with ever increasing sizes of biomolecular databases.

Results: Using a general-purpose graphics processing unit (GPU), we have developed GPU-BLAST, an accelerated version of the popular NCBI-BLAST. The implementation is based on the source code of NCBI-BLAST, thus maintaining the same input and output interface while producing identical results. In comparison to the sequential NCBI-BLAST, the speedups achieved by GPU-BLAST range mostly between 3 and 4.

Availability: The source code of GPU-BLAST is freely available at http://archimedes.cheme.cmu.edu/biosoftware.html.

Contact: sahinidis@cmu.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 INTRODUCTION

BLAST was introduced as a sequence alignment heuristic that was an order of magnitude faster than earlier approaches for analyzing biological information. Very quickly, this software became a landmark enabling technique for bioinformatics. According to the *Web of Science*, the paper that describes the first version of ungapped BLAST (Altschul *et al.*, 1990) has been cited more than 28 000 times. In addition, the paper that describes the gapped version of the algorithm and a technique to speed up the earlier version by a factor of three (Altschul *et al.*, 1997) has been cited more than 25 000 times. The level of usage of BLAST suggests that any improvement in its execution speed will result in significant impact in bioinformatics. Research efforts in this direction have been substantial and have relied mainly on custom-designed hardware (Sotiriades and Dollas, 2007) and parallel supercomputing (Lin *et al.*, 2008). Even though these efforts have resulted in impressive speedups of up to three orders of magnitude, neither custom hardware nor supercomputers are easily accessible by the majority of BLAST users.

With the advent of multicore processors, there have been several efforts to parallelize BLAST and speedup its execution on commodity hardware. The National Center for Biotechnology Information (NCBI) has developed a version of BLAST that exploits multicore processors for the first phase of the algorithm (Camacho *et al.*, 2009). Another parallel version of BLAST (Nguyen and Lavenier, 2009) exploits two features of modern microprocessors– SSE instructions and multithreading–and achieves speedups of up to 5.6 times compared with NCBI-BLAST. However, the resulting protein alignments are up to 5.9% different than those produced by NCBI-BLAST (Table 4 in Nguyen and Lavenier, 2009).

Recently, Graphics Processing Units (GPUs) became available as a general purpose processing platforms. We were drawn to GPUs because of their exceptionally high performance-to-cost ratio. For around $1500, it is possible to combine a personal computer with a GPU and achieve trillions of peak floating point operations per second (FLOPS) performance. GPU technology brings supercomputing power to the desktop, thus facilitating the widespread use of parallel algorithms by bioinformaticians. However, algorithms that perform well on a CPU may not perform as well on a GPU (c.f. Elble *et al.*, 2010). Algorithm developers must develop new algorithms in order to harvest the GPU's massive parallel nature.

GPUs were designed to accelerate graphics processing and quickly outperformed CPUs by over an order of a magnitude in terms of FLOPS and memory bandwidth performance. This potential was initially difficult to harness in applications beyond graphics. The situation changed in 2007 with the introduction of NVIDIA's Compute Unified Device Architecture (CUDA), a software and hardware environment that facilitates the adoption of GPUs in general purpose computing (Nickolls, 2007). Since then, the use of GPUs has proved advantageous in a number of computationally intensive bioinformatics problems, including the Smith–Waterman alignment algorithm (Manavski and Valle, 2008), molecular docking (Sukhwani and Herbordt, 2009), the protein-folding problem (Beberg *et al.*, 2009; Shirts and Pande, 2000), DNA sequencing (Schatz *et al.*, 2007), computational proteomics (Hussong *et al.*, 2009), statistical phylogenetics (Suchard and Rambaut, 2009), biological systems simulation (Dematte and Prandi, 2010) and cellular-level simulation (Richmond *et al.*, 2010). Several GPU-based bioinformatics software can be found at http://www.nvidia.com/object/tesla_bio_workbench.html.

A GPU-based BLAST was recently developed by Ling and Benkrid (2010), leading to speedups between 1.7 and 2.7 in comparison to NCBI-BLAST. Its authors report that this

---

*To whom correspondence should be addressed.

implementation is not guaranteed to give results identical to those from NCBI-BLAST. Liu's GPU-based BLAST (www.nvidia.com/object/blastp_on_tesla.html) achieves speedups of six. Using default options for Liu's code and NCBI-BLAST 2.2.24, we obtained different alignments for all 51 sequences provided in the 'queries' directory of the installation of GPU-BLAST. However, most users of bioinformatics software are reluctant to use implementations of BLAST that may produce alignments that are not identical to those obtained from NCBI-BLAST.

We built GPU-BLAST directly on top of the NCBI-BLAST code. As a result, GPU-BLAST has a familiar interface to the user and, most importantly, produces identical search results with NCBI-BLAST. We took advantage of the algorithm's parallel aspects by mapping it on the GPU multithreaded processing environment, while also allowing the concurrent utilization of multiple CPU threads in parallel. Although GPU-BLAST shares many data structures with NCBI-BLAST, we made necessary modifications to exploit the GPU without compromising the accuracy of the produced alignments. The current version of GPU-BLAST can perform protein alignments up to 4 times faster than the single-threaded NCBI-BLAST. Even compared to a six-threaded NCBI-BLAST, the GPU-BLAST is nearly twice as fast. These attributes are likely to facilitate the adoption of GPU-BLAST by the bioinformatics community.

The remainder of the article is organized as follows. In Section 2, we present an overview of the BLAST algorithm. In Section 3, we describe the most important architectural features related to GPU-BLAST, and in Section 4 we present the implementation of the algorithm on the GPU. In Section 5, we present the quantitative results of the implementation, followed by conclusions in Section 6.

2   ALGORITHM

The sequence alignment problem calls for searching a sequence database for matches with a query sequence. The first proposed method to solve this problem was the Smith–Waterman algorithm (Smith and Waterman, 1981). Although this algorithm produces an optimal alignment between two sequences and runs in time polynomial in the length of the two sequences, it is computationally expensive for long sequences, and, in many cases, overlooks alignments that are suboptimal but may provide useful biological information. These shortcomings became increasingly pronounced with the increasing size of biological databases.

BLAST addresses these problems with a heuristic that is fast and biologically relevant. The approach consists of three main steps: seeding, extension, and evaluation. The seeding step identifies short words common between the query and a database sequence and uses them as seeds in the extension step. The word length is user defined and affects the accuracy and speed of the algorithm; longer words result in fewer seeds, and, consequently, shorter execution times.

The second step investigates whether the seeds belong to longer, common subsequences. This step discards the false positive seeds that occur by chance and keeps the seeds that occur because they are part of a longer common subsequence. This is achieved by extending the alignment to the left and right of the seed. The unconditional left and right extension in the initial version of BLAST (Altschul *et al.*, 1990) is called the one-hit method and typically consumes over 90% of BLAST's total execution time (Peters and Sikorski, 1997). In order to improve this computationally intensive part of the algorithm, the two-hit extension was introduced in 1997

(Altschul *et al.*, 1997). In this method, an extension is invoked only for seeds that are within a user-defined distance from non-overlapping seeds, thus reducing the computational cost by half. Initially, the seeds are extended from both the left and right without inserting any gaps. During this process, the quality of the ungapped alignment is gauged by the score of each pair of aligned amino acids using a scoring matrix, such as the popular BLOSUM62 (Henikoff and Henikoff, 1992). If the ungapped score is above a user-defined threshold, the seed can be used to produce a gapped alignment based on a Smith–Waterman type algorithm (Smith and Waterman, 1981).

The evaluation step relies on the score produced by the ungapped or the gapped extension step, the query and database sequence lengths, the substitution matrix and the sequence statistics. With this information, the alignment is accepted as statistically significant if the probability of finding such an alignment by chance is lower than a user-defined value (Karlin and Altschul, 1990).

A profiling study of NCBI-BLAST for protein alignment is depicted in Figure 1. The time spent in each step of the algorithm can vary substantially with different queries. However, as Figure 1 reveals, the seeding and the ungapped extension are the most computationally intensive parts. For ungapped alignments, these two steps consume over 95% of the total execution time. For gapped alignments, the seeding and the ungapped extension steps consume 75% of the time, while 20% of the time is spent on the gapped extension. Based on these observations, we decided to focus our parallelization efforts on the seeding and ungapped extension steps.

3   SYSTEM AND METHODS

A GPU is a massively parallel computer designed to accelerate computationally intensive applications by operating in a single-instruction multiple-thread (SIMT) mode. The same instructions are executed in parallel by multiple threads that run on identical cores and can operate on different data. Figure 2 presents a block diagram of an NVIDIA GPU as it is executing GPU-BLAST. The schematic shows that there are N GPU multiprocessors, each containing M processors.

The executing threads are organized into so-called blocks, and the blocks are organized into a so-called grid of blocks. The number of threads and blocks is user defined, and the GPU scheduling mechanism assigns the execution of each thread block to a specific GPU multiprocessor. Since each GPU multiprocessor has N processors, there is a maximum number of threads that can physically execute in parallel. This thread group is called a *warp*. A thread block may contain more than one warp, and the GPU scheduler decides in which order and when to execute each warp. This gives the capability to the scheduler to increase the overall utilization of the GPU multiprocessor by putting a warp on hold, e.g. when waiting for data, and allow another one to execute.

Since each GPU multiprocessor has a single-instruction unit, there is one instruction dispatched at any given time. Hence, parallelization is maximized when all threads of a warp agree on their execution path. If threads of a warp diverge via a data-dependent conditional branch, the warp serially executes each branch path taken, disabling threads that are not on that path. When all paths complete, the threads converge back to the same execution path. Extensive thread divergence can have a detrimental effect on performance.

The schematic in Figure 2 illustrates that there are available different types of memory, each with different functionality, size and speed. Depending on the amount of data and anticipated data access pattern, the programmer must organize and store the data in the most appropriate memory, in order to achieve the best possible utilization of the available memory bandwidth. The global memory is the largest in size and the slowest. It can be read and written by the CPU and the GPU threads, thus allowing the CPU to send data to the GPU and vise versa. The global memory access pattern by the threads
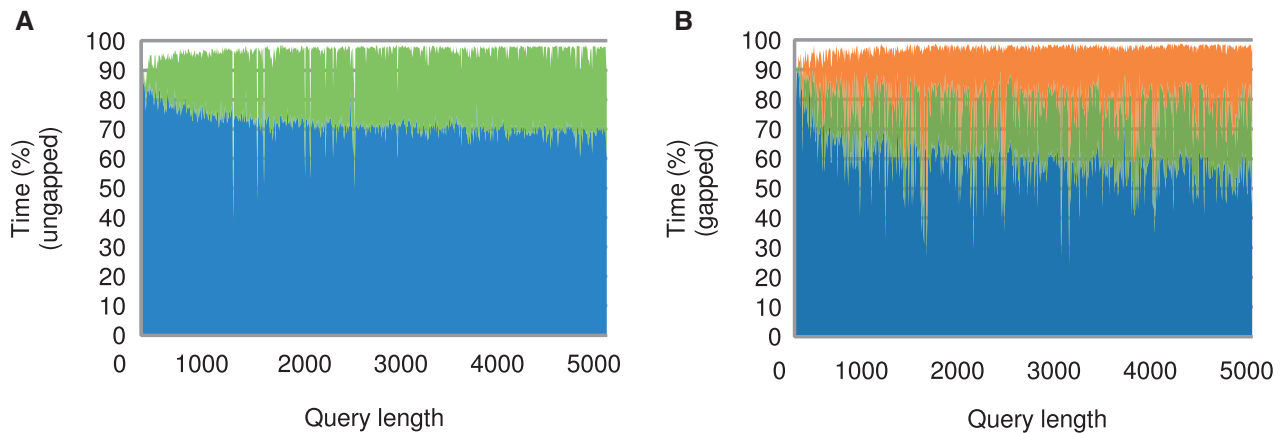
Fig. 1. Profiling of the NCBI-BLAST code for queries of length 2 to 4998 for two-hit extensions. (**A**) For ungapped alignments, on average, the seed identification and extension steps, respectively, consume 75% of the total time (blue) and 20% of the total time (green). (**B**) For gapped alignment, on average, the seed identification, two-hit ungapped extension and gapped extension steps, respectively, consume 55% of the total time (blue), 20% of the total time (green) and 20% of the total time (orange).
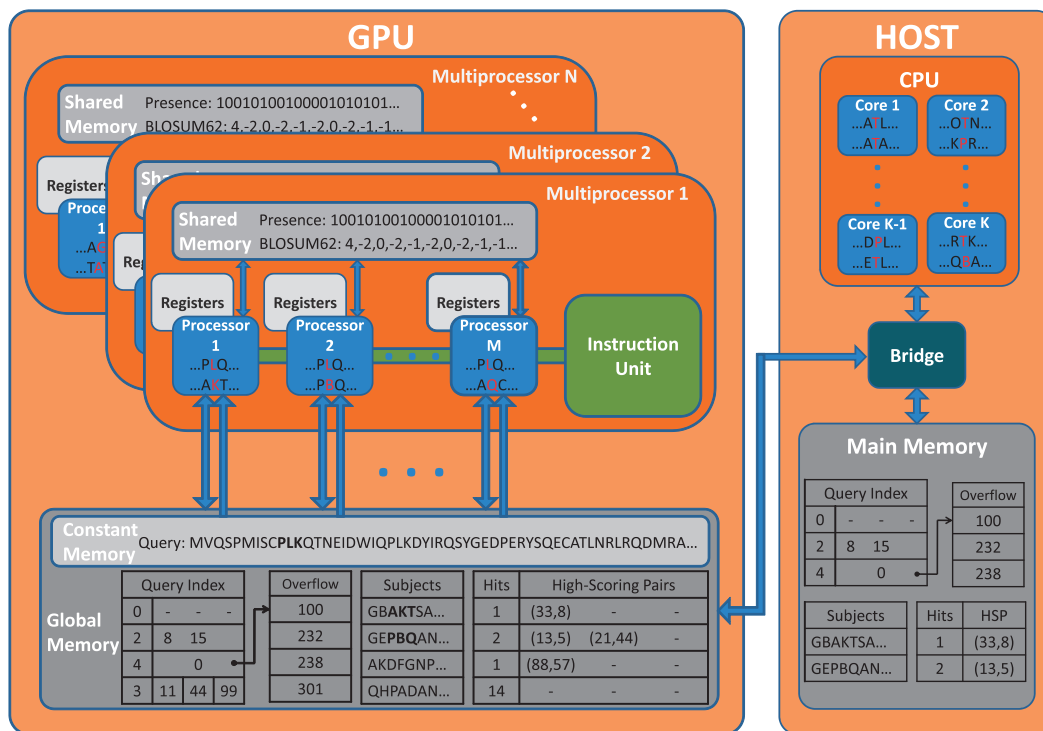

Fig. 2. The architecture and the memory organization of an NVIDIA GPU as it is executing GPU-BLAST. The data structures used by GPU BLAST are stored in the appropriate memory type, according to their size and access pattern.

can affect substantially the data transfer bandwidth; the more coalesced the memory accessing within a half warp, the higher the achieved bandwidth.

The constant memory is part of the global memory and is read-write for the CPU and read-only for the GPU threads. Constant memory can offer higher bandwidth than the global memory when all threads of a half warp access the same input data. The shared memory is the smallest and the fastest and is shared by all processors of a GPU multiprocessor. It is a read-write memory by the GPU threads only, and it can be used to communicate data between threads that belong to the same block. Shared memory bandwidth

can be affected by the thread-access pattern, but to a lesser extent than global memory accessing.

The registers of a GPU multiprocessor are shared between its processors, and each thread uses an exclusive set of registers. The programmer does not have explicit control on the registers, as the latter are used for the execution of a program in the same way as on a general purpose CPU. GPUs also contain local and texture memory, which were not found useful in the context of GPU-BLAST and are not depicted in Figure 2. Local memory, in particular, is used by the compiler automatically to store variables

if needed, but was not used for GPU-BLAST. Texture memory, on the other hand, is controlled by the programmer and can benefit applications with spatial locality where global memory access is the bottleneck. As discussed later, however, thread divergence instead of global memory access is GPU-BLAST's bottleneck.

4 IMPLEMENTATION

The most important component of the implementation is the design of the data structures, which affect the efficiency of the parallelization and overall implementation. Since GPU-BLAST is embedded in the NCBI-BLAST code, the two implementations share data structures. As shown in Figure 2, the most important data structures used by GPU-BLAST consist of a table holding the substitution matrix, a presence bit vector holding information on whether a specific amino acid word is present in the query, a query-index table and an overflow table holding the positions of the words of the query, a table holding the database subjects and an index table holding the resulting ungapped alignments between the query and each of the database subjects. The location of each data structure in memory was carefully selected, depending on data size and how often each structure is accessed during execution. In particular, we have stored frequently accessed structures in the fastest possible memories that could accommodate their size.

The query-index table is created in a preprocessing step of the algorithm. For each word, this table stores how many times the word appears in the query, and the location of each appearance. Theoretically, if the word length is w and the query length is l, a word can appear up to $l-w+1$ times (corresponding to the case when all the query amino acids are identical). In practice, however, each word appears only a few times in a query. For this reason, the locations of words that appear up to three times are stored in the query index table. For all other cases, instead of locations, the index table contains a pointer to an overflow vector that holds the locations of the words in the query. The index table and the overflow vector cannot fit in the shared memory and are stored in the global memory, as shown in Figure 2. Each bit of the presence vector corresponds to a word and is set only if that word appears in the query. Since its size is only a fraction of the query index table, this vector can be stored in the smaller but faster shared memory. The query is uniformly accessed by all GPU threads. For this reason, it is stored in the constant memory, which is the most suitable for this access pattern. The protein database that the query is compared against is also stored in the global memory due to its size.

Parallelization in the execution step of GPU-BLAST involves assigning the database subjects to different GPU threads. In order to balance the load between the threads and avoid having threads of the same warp work on database sequences with substantial length differences, the sequences are first sorted according to the number of amino acids they contain. Sorting is embedded in the formatting of a FASTA database, which is required by NCBI-BLAST. This operation is done once per database before this database is used and does not affect the alignment obtained. Thus, this operation does not add any overhead to the execution time of the algorithm for NCBI- or GPU-BLAST. Not having the database sorted would result in cases where threads of the same warp have to compare the query with sequences that differ significantly in length, thus causing excessive thread divergence. By sorting, this thread divergence overhead is reduced considerably.

Each thread scans consecutive words of a different subject and checks, via the presence vector, whether these words exist in the query or not. The presence vector is not necessary for the implementation of the algorithm since the query index table stores the word locations. Yet, the presence vector is small enough to fit in the shared memory of a GPU multiprocessor. Thanks to the information provided by this vector, a processor can identify word matches through information readily available in the fast shared memory, without having to access the slow global memory. Only when matches exist, the processor accesses the query index table in the global memory to retrieve information on the number and locations of the seeds. The substitution matrix is stored in the shared memory because it is used very frequently during the alignment score calculations.

In the next step, each seed is extended left and right according to the two-hit method. Each extension that achieves a score above the user-defined threshold is characterized as a high scoring pair and its coordinates are stored in the output table. Since it is not known in advance how many high scoring pairs per database subject will be discovered, for their storage we follow a similar technique used for the query index table. In practice, for each subject, there are only a few high scoring pairs discovered. Thus, the coordinates of only up to two pairs are stored in the output table. Database subjects that have more pairs are processed by the CPU after completion of the GPU execution.

The CPU has a copy of the database and the data structures, and, instead of waiting idle for the GPU to parse the entire database, carries out part of the alignment task, thus reducing the total execution time. The database is split in two parts that are processed separately, and when both processors finish execution, the CPU merges the results, and, if desired, carries out a gapped alignment.

In BLAST, comparison of a query with any sequence in the database can be carried out independently from comparisons with other database sequences. While this observation makes parallelization of this algorithm appear an obvious task, the challenge here is to develop a mechanism capable of distributing comparisons to different processors so that processors are fully utilized and complete their assigned tasks at the same time. The processing of short and long database sequences must be done in a way that minimizes idle times for processors. The data structures used in GPU-BLAST result in a carefully orchestrated parallel execution of comparisons of short and long sequences, thus utilizing the GPU as much as possible.

Since GPU-BLAST is built on top of NCBI-BLAST, both share a common user interface. GPU-BLAST has the following additional options:

```
*** GPU options
-gpu <Boolean>
  Use GPU for blastp
  Default = 'F'
-gpu_threads <Integer, 1...1024>
  Number of GPU threads per block
  Default = '64'
-gpu_blocks <Integer, 1...65536>
  Number of GPU block per grid
  Default = '512'
-method <Integer, 1...2>
```

```
Method to be used
   1 = for GPU-based sequence alignment
   2 = for GPU database creation
Default = '1'
 * Incompatible with:  num_threads
```

'-gpu <Boolean>' determines whether the GPU is used or not. With '-gpu_blocks' and '-gpu_threads', the user can define the number of blocks and threads per block to be used by the GPU. When '-gpu T' GPU-BLAST carries out the sequence alignment using the aforementioned options. When '-method 2', and provided that '-gpu T', GPU-BLAST converts the input database into the format required by GPU-BLAST, stores the produced database into a separate file and produces a second file which includes information about the GPU database. The conversion has to be done only once for each database, and all subsequent executions of GPU-BLAST read this database from the disk. Hence, this time is amortized over thousands or millions of future queries. For '-gpu F', all the previous options are ignored and GPU-BLAST executes according to NCBI-BLAST. The current version of GPU-BLAST works only for protein alignments and can utilize more than one CPU threads in parallel with the GPU by using the NCBI-BLAST option '-num_threads <Integer>=1>'. The option '-method' is incompatible with the option '-num_threads' because the creation of the GPU database does not support multiple threads.

The execution of GPU-BLAST consists of three basic components: (i) initialization of the GPU and data transfer from the CPU to the GPU, (ii) concurrent GPU-CPU algorithm execution and (iii) transfer of results from the GPU to the CPU. These basic steps are depicted in the flow chart of Figure 3. We can see BLAST's basic steps and their execution sequence depending on whether the user chooses to use the GPU or not. If the GPU is used ('-gpu T'), the CPU reads the GPU-BLAST database and sends all the necessary data to the GPU. For the seeding and extensions steps, the CPU and GPU work concurrently; the GPU by deploying multiple parallel threads on the sequences, as defined by '-gpu_blocks' and '-gpu_threads', and the CPU on the remaining sequences by using one or more threads as defined by '-num_threads'. After both platforms finish, the high scoring pairs are transferred from the GPU to the CPU, and the CPU merges them with its own high scoring pairs. From that point, the algorithm follows the NCBI-BLAST execution path without any modifications. GPU-BLAST follows the NCBI-BLAST execution path when '-gpu F'.

GPU-BLAST was implemented on an NVIDIA Fermi C2050 GPU with 448 processors at 1.15 GHz, 64 KB of shared memory per GPU multiprocessor, 64 KB of constant memory and 3 GB of global memory. The implementation was built using CUDA, which offers better performance than OpenCL on NVIDIA GPUs (Weber *et al.*, 2010). While this software choice limits GPU-BLAST to NVIDIA cards, future versions will provide support for OpenCL in order to extend applicability to other GPU hardware.

The GPU was combined with a six-core Intel Xeon host CPU at 2.67 GHz with 12 GB of memory. Since GPU-BLAST uses the GPU and the CPU concurrently, the workload has to be properly distributed between the two in order to maximize the utilization of the CPU–GPU combination. The ideal load balancing is achieved when the execution times on the CPU and the GPU are equal. GPU-BLAST assigns predetermined fractions of the
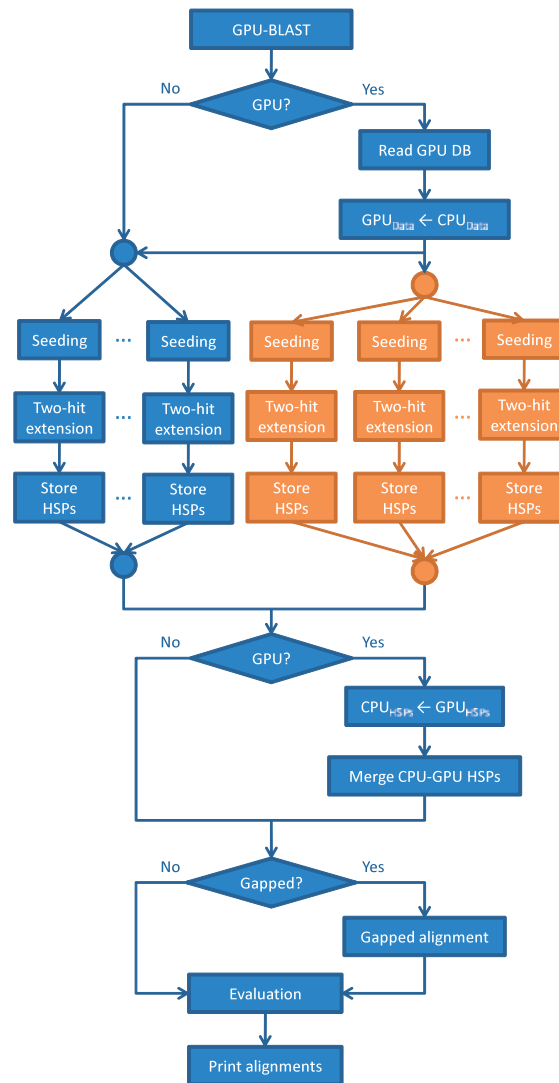

Fig. 3. Execution flow of GPU-BLAST. The blue blocks are executed on the CPU and orange ones on the GPU (HSP: high scoring pairs, DB: database).

database between the CPU and the GPU, based on the number of available CPU threads. We determined these ratios after extensive experimentation with different databases and number of available CPU threads.

5 RESULTS

The database used for computations was the latest releases of the env_nr (ftp://ftp.ncbi.nlm.nih.gov/blast/db/) protein database, which contains 6 031 291 sequences and its size is 1.3 GB (October 2010). The queries were 51 mouse sequences with lengths from 2 to 4998. These sequences were obtained from the UniProt database (http://www.uniprot.org/) and are provided in the 'queries' directory of the GPU-BLAST distribution.

Figure 4A depicts the speedups achieved by the ungapped and gapped versions of GPU-BLAST, in comparison to one-threaded and six-threaded NCBI-BLAST for the env_nr database. These speedups depend on the query length. The speedups increase for query lengths
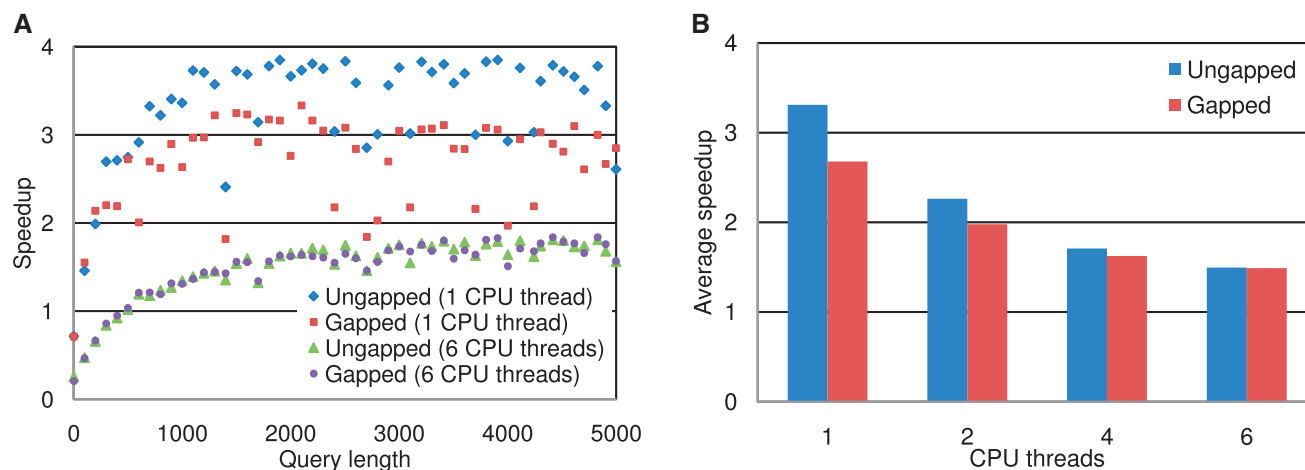
Fig. 4. GPU-BLAST speedups relative to the CPU as a function of sequence length (**A**), and average speedups as a function of CPU threads working in parallel with GPU-BLAST (**B**). Speedups were calculated based on start-to-finish wall-clock times.

of approximately up to 1000 amino acids for the one-threaded and 2000 for the six-threaded implementations, after which the speedup remains essentially constant. For shorter queries, the speedup is slightly lower because the seed identification and the extension steps consume a smaller percentage of the total execution time, as seen in Figure 1. The scattering of the speedups in Figure 4A can be attributed to several factors, including the number of seeds identified, the extension length around each seed and the number of ungapped and/or gapped extensions, which affect the thread divergence on the GPU and consequently its performance.

The GPU's theoretical peak performance is 1030 GFLOPS in single precision and 515 GFLOPS in double precision. The corresponding numbers for the CPU are 128 GFLOPS and 64 GFLOPS. Although the GPU's peak performance in GFLOPS is about eight times higher than the CPU's, the speedups achieved by GPU-BLAST are currently around four. The reason for this difference is that the SIMT architecture of the GPU executes concurrently multiple threads that operate on different data and follow the same execution path in each warp. Whenever the execution paths within a warp diverge, the threads are serialized and overall performance is reduced.

The one-threaded GPU-BLAST is faster for ungapped than gapped alignments because it is possible to transfer 95% of the computations to the GPU in the ungapped case, compared with only 75% in the gapped case as shown in Figure 1. For the six-threaded GPU-BLAST, the total speedup is smaller and the difference between the ungapped and gapped version diminished because the CPU can handle a bigger workload leaving a smaller margin to the GPU to speedup the total running time. For the one-threaded GPU-BLAST the speedup is always bigger than one, except for the first sequence which has length two. The six-threaded GPU-BLAST offers speedup for sequences longer than 500 amino acids.

In Figure 4B, we present average GPU-BLAST speedups when using up to six CPU threads in parallel with the GPU. The times used to calculate each speedup are elapsed times to carry out a sequence alignment, which start from the beginning of GPU-BLAST's execution and finish with the writing of the output

alignments to a file. We can see that GPU-BLAST achieves the largest speedups compared with single-threaded NCBI-BLAST, and the speedups decrease as the number of CPU threads increase.

Finally, in Figure 5, we present speedups relative to a single-threaded CPU. Both multi-threaded CPU and CPU/GPU combinations are considered as a function of the number of available CPU threads. In all cases, speedups were calculated based on the total time to align the entire set of queries. As this figure shows, the multithreaded NCBI-BLAST itself does not scale linearly. For instance, with six CPU threads, the NCBI-BLAST speedup is less than four. GPU-BLAST inherits some of these limitations as it is built on top of NCBI-BLAST in order to guarantee the same output results. Nonetheless, in all cases, the addition of the GPU considerably increases the observed speedups. For instance, the six-threaded GPU-BLAST achieves a speedup of nearly six for both gapped and ungapped alignments.

6 CONCLUSIONS

Using carefully orchestrated parallel execution of comparisons of short and long sequences on a GPU, this article has demonstrated that GPU-BLAST can speed up the popular NCBI-BLAST code by nearly four times while producing identical results. Moreover, our implementation is capable of using the GPU along with multiple CPU cores concurrently. Hence, the performance of GPU-BLAST will benefit from future hardware advances of both CPU and GPU technologies.

The present version of GPU-BLAST only works for BLASTP. Future work will extend the implementation to other BLAST methods, including PSI-BLAST which is more sensitive in detecting weak relationships between protein sequences (Altschul *et al.*, 1997). PSI-BLAST uses multiple iterations to scan the database, with each iteration constructing a position-specific score matrix that replaces the simple query. Although there are differences in the implementations of BLAST and PSI-BLAST, both algorithms share several subroutines. A profiling study of PSI-BLAST reveals that PSI-BLAST and gapped BLAST share the same subroutines that take most of their execution time. In particular, the profiling graph
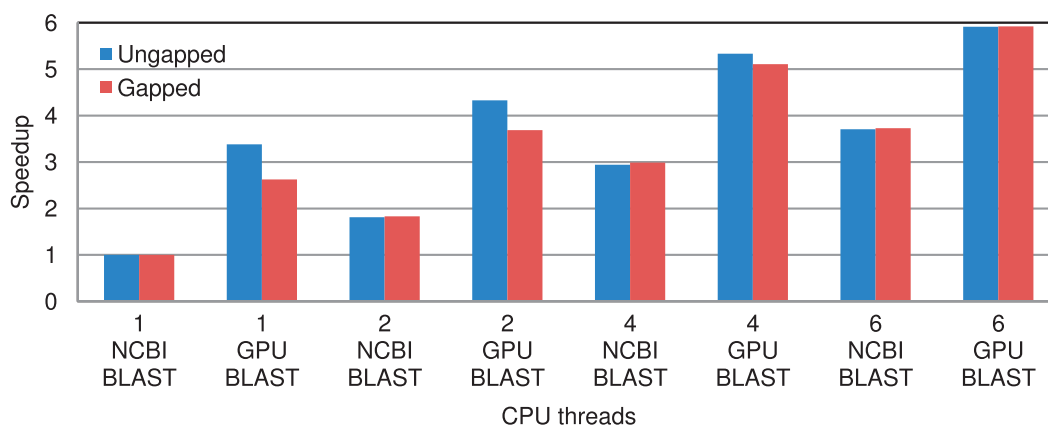
Fig. 5. Speedups relative to a single-threaded CPU as a function of CPU threads. Speedups were calculated based on start-to-finish wall-clock times to align the entire set of queries.

of PSI-BLAST is almost identical to Figure 1B. This suggests that PSI-BLAST can be implemented on the GPU in a similar fashion with GPU-BLAST and that similar speedups are likely.

Conflicts of Interest: none declared.

REFERENCES

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Altschul,S.F. *et al.* (1997) Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.

Beberg,A.L. *et al.* (2009) Folding@home: lessons from eight years of volunteer distributed computing. In *Proceedings of the 8th IEEE International Workshop on High Performance Computational Biology*. IEEE, Rome, Italy, pp. 1–8.

Camacho,C. *et al.* (2009) BLAST+: architecture and applications. *BMC Bioinformatics*, **10**, 421.

Dematte,L. and Prandi,D. (2010) GPU computing for systems biology. *Brief. Bioinform.*, **11**, 323–333.

Elble,J.M. *et al.* (2010) GPU computing with Kaczmarz's and other iterative algorithms for linear systems. *Parallel Comput.*, **36**, 215–231.

Henikoff,S. and Henikoff,J. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA*, **89**, 10915–10919.

Hussong,R. *et al.* (2009) Highly accelerated feature detection in proteomics data sets using modern graphics processing units. *Bioinformatics*, **25**, 1937–1943.

Karlin,S. and Altschul,S.F. (1990) Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl Acad. Sci. USA*, **87**, 2264–2268.

Lin,H. *et al.* (2008) Massively parallel genomic sequence search on the Blue Gene/P architecture. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Austin, TX, pp.1–11.

Ling,C. and Benkrid,K. (2010) Design and implementation of a CUDA-compatible GPU-based core for gapped BLAST algorithm. *Procedia Comput. Sci. USA*, **1**, 495–504.

Manavski,S. and Valle,G. (2008) CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, **9** (Suppl. 2), S10.

Nguyen,V.H. and Lavenier,D. (2009) PLAST: parallel local alignment search tool for database comparison. *BMC Bioinformatics*, **10**, 329.

Nickolls,J. (2007) Nvidia GPU parallel computing architecture. In *IEEE Hot Chips 19*, IEEE Technical Committee on Microprocessors and Microcomputers, Stanford, CA.

Peters,R. and Sikorski,R. (1997) BLAST off! *Science*, **278**, 510–502.

Richmond,P. *et al.* (2010) High performance cellular level agent-based simulation with FLAME for the GPU. *Brief. Bioinform.*, **11**, 334–347.

Schatz,M. *et al.* (2007) High-throughput sequence alignment using graphics processing units. *BMC Bioinformatics*, **8**, 474.

Shirts,M. and Pande,V.S. (2000) Screen savers of the world unite! *Science*, **290**, 5498.

Smith,T. and Waterman,M. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **137**, 195–197.

Sotiriades,E. and Dollas,A. (2007) A general reconfigurable architecture for the BLAST algorithm. *J. VLSI Signal Process.*, **48**, 189–200.

Suchard,M.A. and Rambaut,A. (2009) Many-core algorithms for statistical phylogenetics. *Bioinformatics*, **25**, 1370–1376.

Sukhwani,B. and Herbordt,M.C. (2009) GPU acceleration of a production molecular docking code. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, ACM, Washington, DC, pp.19–27.

Weber,R. *et al.* (2010). Comparing hardware accelerators in scientific applications: a case study. *IEEE Trans.Parallel and Distributed Systems*. IEEE computer Society Digital Library, IEEE Computer Society. Available at http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.125 (last accessed date June 2, 2010)