Sequence analysis

H-BLAST: a fast protein sequence alignment toolkit on heterogeneous computers with GPUs

Weicai Ye¹, Ying Chen¹, Yongdong Zhang¹,* and Yuesheng Xu¹,²,*

¹School of Data and Computer Science, and Guangdong Province Key Laboratory of Computational Science, Sun Yat-sen University, Guangzhou 510275, People’s Republic of China and ²Professor Emeritus of Department of Mathematics, Syracuse University, Syracuse, NY 13244, USA

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on August 6, 2016; revised on November 7, 2016; editorial decision on November 29, 2016; accepted on December 12, 2016

Abstract

Motivation: The sequence alignment is a fundamental problem in bioinformatics. BLAST is a routinely used tool for this purpose with over 118 000 citations in the past two decades. As the size of bio-sequence databases grows exponentially, the computational speed of alignment softwares must be improved.

Results: We develop the heterogeneous BLAST (H-BLAST), a fast parallel search tool for a heterogeneous computer that couples CPUs and GPUs, to accelerate BLASTX and BLASTP—basic tools of NCBI-BLAST. H-BLAST employs a locally decoupled seed-extension algorithm for better performance on GPUs, and offers a performance tuning mechanism for better efficiency among various CPUs and GPUs combinations. H-BLAST produces identical alignment results as NCBI-BLAST and its computational speed is much faster than that of NCBI-BLAST. Speedups achieved by H-BLAST over sequential NCBI-BLASTP (resp. NCBI-BLASTX) range mostly from 4 to 10 (resp. 5 to 7.2). With 2 CPU threads and 2 GPUs, H-BLAST can be faster than 16-threaded NCBI-BLASTX. Furthermore, H-BLAST is 1.5–4 times faster than GPU-BLAST.

Availability and Implementation: https://github.com/Yeyke/H-BLAST.git

Contact: yux06@syr.edu

Supplementary information: Supplementary data are available at Bioinformatics online.

1 Introduction

In bioinformatics, the basic local alignment search tool (BLAST) (Altschul et al., 1990, 1997) is not only a daily used algorithm identifying regions of local similarity between biological sequences, but also ‘the principal means by which many other algorithms query large genomic datasets’ (Loh et al., 2012). Hence, there are fruitful applications of BLAST in inferring functional (Mackelprang et al., 2011) and evolutionary (Huang et al., 2014) relationships between the corresponding organisms. As the next generation sequencing (NGS) technique advances, the size of sequence databases has exponentially grown (Daniels et al., 2013) and the search speed of BLAST is insufficient. Therefore, sequence alignment with BLAST became a major bottleneck. To solve this problem, a number of tools were developed in the literature.

There are two categories of methods to accelerate BLAST searching against protein databases. Methods of category one change indexing targets from query as in BLAST to database, and modify the non-exact match seeding strategy with a reduced amino-acid alphabet and spaced seeds. Commonly used software tools of this category include BLAT (Kent, 2002), USEARCH (Edgar, 2010), RAPSearch2 (Zhao et al., 2012) and DIAMOND (Buchfink et al., 2015). Methods of another category are parallel implementation on various specific hardware, such as FPGAs (field-programmable gate arrays) (Fei et al., 2008; Herboldt et al., 2006; Wienbrandta et al., 2011), Cell Broadband Engines (Zhang et al., 2008), multi-core CPUs (Camacho et al., 2008), graphic processing units (GPUs) only (Cheng and Benkridb, 2010; Suzuki et al., 2012), heterogeneous computers with GPUs (Liu et al.; Liu et al., 2011; Vouzis and
The standard BLASTX, a basic tool from NCBI-BLAST distribution (Camacho et al., 2008), is the de facto standard tool for functional and taxonomical classification of DNA sequences in metagenomics (Bazinet and Cummings, 2012; Hauswedell et al., 2014). To identify protein coding genes against protein database, BLASTX imitated the algorithm from protein BLAST (BLASTP in brief) for finding homology between amino-acid sequences. Among the many solutions mentioned above, mpiBLAST (Darling et al., 2003), ScalaBLAST (Oehmen and Baxter, 2013, Oehmen and Nieplocha, 2006) and GPU-BLAST (Vouzis and Sahinidis, 2011) are dedicated to accelerating BLAST by parallel implementation and producing the same alignment results as NCBI-BLAST. MpiBLAST and ScalaBLAST support BLASTX and BLASTP, and GPU-BLAST supports BLASTP. Both mpiBLAST and GPU-BLAST utilize the independency of search procedures on different subjects. mpiBLAST schedules different searches on CPUs of a high performance cluster. Even though speedups of mpiBLAST can be up to thousands times, its use is heavily relied on the availability of huge supercomputers. Heterogenous computers offer a low cost solution to high performance. Employing CPUs and GPUs sharing the same host, heterogeneous computing not only gains high peak performance rivaling that of massive CPU cores, but also eases the memory limitation of the GPU-only computing. GPU-BLAST is an example. It gains a speedup of 3–4 times over sequential NCBI-BLAST.

We propose in this paper a fast BLAST algorithm (H-BLAST) and develop its software on heterogeneous computers to speed up BLASTX and BLASTP, preserving the same alignment results as NCBI-BLAST. Key challenges of accelerating BLAST on heterogeneous computers with GPUs are refining the BLAST algorithm to suite the lock step execution style of GPUs, and balancing workloads among CPU cores and GPUs. Therefore, H-BLAST employs a locally decoupled seed-extension algorithm for GPUs within a new fine-grained GPU kernel. A fine-grained BLAST GPU kernel is proposed in (Zhang et al., 2014) with promising speedup. H-BLAST offers a performance tuning mechanism for better efficiency among various CPUs and GPUs combinations, including manual and automatic tuning. GPU memory demand for H-BLAST can be as less as 1/8 of that for GPU-BLAST.

In this paper, we first review the algorithm of BLASTX and BLASTP, and describe main ideas of our H-BLAST algorithm, and then present details about its implementation, including the fine-grained GPU algorithm and the performance tuning mechanism within a CPU/GPU cooperation pipeline. We also address the usage of H-BLAST. We compare the performances of H-BLAST, NCBI-BLAST and GPU-BLAST by conducting experiments in different scenarios. Finally, we draw a conclusion with future work.

### 2 Materials and methods

Both BLASTX and BLASTP searching procedures include four stages: the setup stage, the preliminary search stage, the traceback stage and the output stage. Without loss of generality, we take the BLASTX searching procedure as an example. At the setup stage, BLASTX not only prepares searching options, query sequences and subject sequences from the given database, but also creates a lookup table indexing query sequences. At the preliminary search stage, it scans each subject, lists all matching words (called hits or seeds) by the lookup table, extends each hit linking another hit to a gapped-free alignment. Each gapped-free alignment exceeding a threshold score is saved as a high scoring pair (HSP). Then, an HSP triggers a gapped alignment. At the traceback stage, BLASTX tries to find more aggressive alignments and records the number and positions of insertions, deletions and matching letter information. Finally, at the last stage, it prints out all alignments by queries. The pseudo codes are listed in Algorithm 1.

Algorithms of BLASTX and BLASTP are very similar. In fact, those used in their seeding step and the ungapped extension step are exactly the same. Their major difference is the data type of queries: nucleotide and amino-acid for BLASTX and BLASTP, respectively. Compared to BLASTP, there are extra jobs in BLASTX: translating nucleotide query sequences into amino-acid sequences in all six reading frames in the setup stage, and considering insertion and deletion events of translated nucleotide query sequences among different frames in the gapped extension step.

**Algorithm 1.** The algorithm shared by BLASTX and BLASTP

```plaintext
for each query chunk
  1. Setup
  2. Preliminary search
     2.1 Seeding
     2.2 Ungapped extension
     2.3 Score only gapped extension
  3. Gapped extension with traceback
  4. Output
end for each
```

The most promising approach to accelerate BLAST is to speed up the preliminary search stage, which is the most computation intensive stage in BLAST reported in profiling studies (Vouzis and Sahinidis, 2011; Zhao and Chu, 2014). In our profiling study of NCBI-BLASTX, seeding and ungapped extension steps are still the most time consuming and they account for 80–90% of the total computing time.

Reasons why we use heterogeneous computing to speed up BLAST can be listed as follows. Heterogeneous computers with GPUs offer a low cost approach to high performance. GPUs are computing power sources. A latest TitanX GPU can delivers 11Tflops peak performance with 250 W power budget, while a latest Intel 18-core CPU presents about 1.3Tflops with 145 W power budget. A computer server can be equipped with 1–8 GPUs. However, one bottleneck of GPU-only computing is the memory limitation. The main stream memory size of a GPU is 2–4 GB, the largest is 24 GB. Facing a 58 GB NCBI nonredundant (NCBI-nr in brief) protein database (released in 2016) and massive alignment results, there is not enough memory space on a GPU. Using the host memory space can ease this problem.

The state-of-the-art accelerated BLAST algorithms on heterogeneous computers mainly focused on designs of fast GPU kernels and CPU/GPU cooperation pipelines. CUDA-BLASTP (Liu et al., 2011) not only used a compressed DPA for hit detection with a sorted database for better load balancing in GPUs, but also processed gapped step in GPUs. GPU-BLAST (Vouzis and Sahinidis, 2011) and G-BLASTN (Zhao and Chu, 2014) offered GPU kernels preserving the same alignment results as NCBI-BLAST. GPU-BLAST claimed a speedup of 3–4 times over sequential NCBI-BLASTP. G-BLASTN provided a CPU/GPU cooperation pipeline scheme to speed up
BLASTN, cuBlastp (Zhang et al., 2014) introduced a fine-grained GPU kernel, which replaced the embarrassing parallel approach and bundled execution tasks of the same diagonals into queues for parallel execution. It improved the memory accessing pattern of GPU kernel and claimed a speedup of 3.7 times over multi-threaded NCBI-BLASTP. However, seeding and extension branches were still mixed together that would harm the performance.

The major bottleneck of accelerating BLAST on heterogeneous computers is unable to fully utilize all available parallelism, especially on GPUs. The source of this bottleneck is the unpredictable execution paths of the BLAST heuristic (Zhang et al., 2014). Its many different coupled branches are not compatible to the lock step execution style of GPUs and increase difficulties on balancing workloads among CPU tasks and GPU tasks. That is a performance killer of GPU applications (Bakhoda et al., 2009). The heuristic with massive parallelism needs huge memory space to store on-the-fly results. For example, using algorithms of GPU-BLAST in BLASTX searching would consume more than 12GB memory, which cannot be easily satisfied by many GPUs.

The key ideas underlying H-BLAST is to solve this bottleneck. H-BLAST tries to decouple alignment jobs and bundle branches of the same type into queues for parallel execution. Our algorithms eliminate the interrelationship of finding seeds (short stretches of identities) and ungapped seed extension in a short subject region to preserve the same alignment results as those from NCBI-BLAST. Jobs in a queue share multiple GPU threads (known as fine-grained parallelism) in batch execution. It leads to much less divergence and better performance theoretically. H-BLAST also applies a linear workload prediction model for workload balancing in a CPU/GPU cooperation pipeline. Furthermore, it uses a lossless data compression strategy with a dynamic memory allocation approach in GPU algorithms for decreasing the memory need.

To speed up Algorithm 1, H-BLAST presents two approaches depending on cases. When only one GPU is used, the seeding and ungapped extension steps are accelerated by multi-core CPUs and a GPU; the score only gapped extension step is speeded up by multi-core CPUs; the rest part runs in serial. To gain a better total speedup, a cooperation step pipelining tasks on CPUs and tasks on the GPU is involved with extra cost. While analyzing massive query sequences with k GPUs, H-BLAST takes advantages of the independence of alignment results of different sequences and divides queries into k parts equally. As a result, stages 1, 3, 4 and the cooperation step can be accelerated by k CPU cores.

H-BLAST software consists of three modules: a GPU scanning module, a CPU scanning module and a cooperation module. The GPU scanning module is for accelerating the seeding and ungapped extension step on GPUs, its entire algorithms are the same as those in both BLASTX and BLASTP. The CPU scanning module processes not only the HSPs from the GPU module with gapped extension, but also some unprocessed database sequences with the whole BLAST algorithms, and merges the alignment results. The cooperation module is to parallelize the CPU and GPU tasks. The main framework of H-BLAST is illustrated in Figure 1.

![Fig. 1. The main framework of H-BLAST and the workflow on the protein sequence alignment with both H-BLAST and NCBI-BLAST modules (Color version of this figure is available at Bioinformatics online.)](image)

3 Implementation of the H-BLAST algorithm

We implement the H-BLAST algorithm with C language for CPU codes and CUDA C language for GPU codes. The H-BLAST software is based on NCBI-BLAST 2.2.28+ software package and integrates code segments from GPU-BLAST. We describe the implementation of the fine-grained GPU kernel design and the cooperation pipeline.

3.1 Accelerating the seeding and ungapped extension steps on GPUs

There are three major challenges in accelerating the BLAST algorithm on GPUs, due to the unpredictable execution path and the storage demand of the heuristic. Parallelizing the algorithm as well as balancing the workload and the storage demand are primary challenges in parallel applications. Minimizing divergence of the execution paths is a specified issue for GPU applications. Especially, divergence of the execution paths is the performance killer of GPU applications. H-BLAST offers a systematic approach to settle these issues.

The H-BLAST GPU kernel adopts a fine-grained parallel strategy, i.e. mapping the alignment task of a database subject to multiple GPU threads. To this end, H-BLAST takes advantages of hierarchy parallelism. Among different database subjects, the alignment tasks are independent, denoted as the external parallelism. For a given offset of one database subject, all seeding tasks among different query offsets and ungapped extension tasks can be parallelized, denoted as the internal parallelism. The execution of an H-BLAST GPU kernel is organized by groups of GPU threads with the external parallelism. The GPU threads within a group share all tasks with the internal parallelism. The design of the H-BLAST GPU kernel is shown in Algorithm 2.

To minimize divergence of the execution paths and balance the workload among different GPU threads, a locally decoupled seed-extension algorithm is proposed, depicted in Algorithm 2. In our algorithm, the execution of H-BLAST seeding and ungapped extension jobs are organized in batch by a dual-job-queue pipeline. The first stage of the pipeline is PSeeding, presented in Algorithm 3. A hit-queue used in Pseeding is formed by the contents (indexed by a u-word) from the lookup table. GPU threads searching the same subject sequence filter these hits and push all hits requesting extension into extension queues for lazy
processing. The second stage of the pipeline is PUngapped_extension, shown in Algorithm 4. Unlike seeding jobs executed every offset, ungapped extension jobs are triggered in batch every 20 offsets.

The group size of GPU threads sharing one subject’s sequence alignment jobs varies from query sequences in H-BLAST adaptively. The group size permits a tradeoff between the parallel efficiency and the computational capacity ratio (CCR in brief) between a CPU core and a GPU. The load balancer for H-BLAST is self-adaptive and hardware-aware, based on a workload prediction model. The model is related to the total length of the protein database sequences and the computational capacity ratio (CCR in brief) between a CPU core and a GPU. The former reflects that the workload obeys uniform distribution of lengths of sequences approximately. The latter is aware of hardware settings changing. The CCR is defined by the following formula:

\[ \text{CCR} := \frac{T_C}{T_G}, \]

where \( T_c \) (resp. \( T_G \)) denotes the execution time of seeding and ungapped extension steps running on one CPU core (resp. one GPU thread).
GPU). The load of GPUs is determined by the total length of the database and the amount of HSPs predicted by that of the last database volume, and the rest load is for CPUs.

To optimize the division of the work, we introduce a real-time greedy algorithm, which is based on minimizing the difference of the execution time between concurrent tasks on CPUs and that on GPUs. Let \( r_{i+1} \) be the unknown ratio of the database sequences used in GPUs with volume id \( n+1 \), which is satisfied the following equation:

\[
\frac{T_{g}(W_{g}(r_{i+1}), T_{c}(n+1))}{W_{g}} = \frac{T_{g}(W_{g}(r_{i}), T_{c}(n))}{W_{g}}, \quad n \geq 2, \tag{2}
\]

where \( T_{g} \) and \( T_{c} \) are execution time prediction functions of \( n_{c} \) CPU cores and \( n_{g} \) GPUs, respectively; \( W_{g} \) and \( W_{g} \) are predicted workloads of CPUs and GPUs, respectively; \( T_{c}(i) \) and \( T_{g}(i) \) present the actual execution time of searching database sequences with volume id \( i \) on CPUs and GPUs, respectively. To be more accurate, we use the real-time data \( T_{c} \) and \( T_{g} \) to correct our empirical time prediction model.

At the beginning of alignment, we give an initial load assignment ratio between CPUs and GPUs. After each database volume is searched, the load assignment rate is adjusted automatically by the difference between the execution time of the CPU task and that of the GPU task as well as a given CCR.

The load balancer for H-BLAST is user-tunable. There is an interface for setting options, such as CCR and the value of an initial load assignment ratio. The details about the tuning interface will be discussed in the next section.

4 Usage

The aligner of H-BLAST is ‘H-BLASTM’, which has the same command line interface as NCBI-BLAST with additional options.

- **module** &lt;**String**, Permissible values: ‘blastp’
- ‘blastx’

The module to be used.

Default = ‘blastp’

- **-gpu_cards** &lt;**Integer**, 1..8&gt;

Use the no. of GPU cards to accelerate blastp and blastx

Default = ‘1’

- **-screen_out** &lt;**String**&gt;

The file is to log all screen out, including stdout and stderr, for error checking.

Default = ‘outfile’

Use H-BLAST in two steps: making a sorted NCBI-BLAST format database and an H-BLAST format database, and doing the alignment job. Firstly, a sorted NCBI-BLAST format database is converted from a FASTA format database and divided into multiple volumes by ‘makeblastdbX’. A small size database volume not only fits the limited GPU memory size, but also is good for overlapping the workload among CPUs and GPUs. Secondly, an H-BLAST format database used on GPUs is built from preceding sorted database by ‘makeGPUB’. These two new databases are constructed once and can be used repeatedly. Finally, do an alignment job with ‘H-BLASTM’. As the result, the alignment search will be executed on both CPUs and GPUs simultaneously.

For advanced performance tuning, H-BLAST presents a configuration file, named ‘H-BLAST_runtime_options’. This file lists options to control the alignment workload of H-BLASTM among CPU threads and GPUs. A default one would be used if there does not exist a customized one. An example file is shown in the supplement document. In this file, one of the key options is the value of CCR defined in Formula (1). However, it is hard to determine the values of these options since they vary according to different machines and test cases. We offer some examples in the supplement document.

5 Results

To assert the execution time and scalability of H-BLAST, we ran H-BLAST, NCBI-BLAST and GPU-BLAST in different scenarios. We compared only the execution time with different numbers of CPU threads and GPUs, since the alignment results of the three aligners are the same. The version of NCBI-BLAST that we used is v2.2.28+, and GPU-BLAST is v1.1.

5.1 General setup and datasets

The experiments were all conducted on a server with two 8-core Intel Xeon E5-2670@2.60GHz CPUs, whose frequency boosts to 3.3GHz when only one CPU core is used. The peak performance of the 8-core CPU is about 0.33Tflops. Two NVIDIA K20x GPUs were installed, and each has 6GB RAM. The peak performance of the GPU is 3.95Tflops. NVIDIA K40m GPUs with 4.29Tflops peak performance were used since each has 12GB RAM. The server had RAM 64GB and ran a linux operating system with CUDA 5.5.

Database. The database used in our tests was an 8GB NCBI-nr protein database (consists of 14 324 397 sequences and 4 906 523 086 letters, released in 2011). The size of H-BLAST format database for CPUs was about 4.3 GB.

Queries. The amino acid queries used were Swiss-Prot sequences, and the nucleotide queries were Homo Sapience sequences. We extracted two query sets from the file swissprot. One including 250 sequences with length being 100–5000 was denoted as query set 0; the other having 6 sequence groups was named as set 3, total length of each group was around 9000, the length of queries ranged from 100 to 600 and was the same in each group. Two nucleotide query sets were extracted from the file Hs.seq.all (ftp://ftp.ncbi.nih.gov/repository/UniGene/), denoted as query set 1 and 2. Query set 1 contained 250 sequences with length being 100–5000. Query set 2 consisted of 4 sequence groups. The total length of each group was 45 000. The length of each sequence in query set 2 ranged from 100 to 400, and is the same in each group.

5.2 Test methods

We applied the single mode test method (Morgulis et al., 2008; Zhao and Chu, 2014) and the batch mode test method in our experiment. Each command line call of H-BLAST or NCBI-BLAST took one query as input in single mode tests. In batch mode tests, the input of each command line call included multiple queries. Before all experiments were conducted, a sorted multi-volume NCBI-BLAST format database and an H-BLAST format database had been built. On our computer, it took about 20 min. Since these two databases were constructed once only, the time spent on building them was not included in H-BLAST running time. The execution time is the wall clock time of the whole alignment procedure. All time records
were measured by the Linux system call ‘time’. We ran each test 5 times and recorded the average execution time.

5.3 Results of single mode tests

In this subsection, we compared the performance of H-BLAST with that of NCBI-BLAST on query sets 0 and 1 in single mode tests. Three procedures, ungapped BLASTP, gapped BLASTP and BLASTX were tested with two metrics: the length of queries and the number of CPU threads. All tests were conducted on a link to analyze the correlation with application performance. Ungapped BLASTP and gapped BLASTP tests were on query set 0, while BLASTX tests were on query set 1. GPU-BLAST and H-BLAST used only one K20x GPU in all tests.

In terms of queries’ length, the speedup results of H-BLAST over NCBI-BLAST and GPU-BLAST are depicted in Figures 2 and 3, respectively. These speedups are about 1-threaded and 4-threaded CPU cases. Shown on Figure 2, H-BLAST outperforms NCBI-BLAST in all tests beyond 100–300 cases. Compared to sequential NCBI-BLAST, speedups of ungapped BLASTP procedure range from 9.0–7.2, when the length of queries is greater than 2600; the major part of the gapped BLASTP’s 1-threaded speedups range mostly from 4.0–7.2; 1-threaded speedups of BLASTX are mostly within 5–7.2. As the length of query increases, a larger speedup is obtained in both the ungapped BLASTP and BLASTX procedures. When the length of queries is greater than 2000, 4-threaded speedups are greater than 2. Figure 3 shows the speedups over GPU-BLAST by sample mean and sample maximum of different length intervals. Compared to GPU-BLAST using 1 CPU thread and 1 GPU, H-BLAST is about 1.5–2.5 (resp. 2–4) times faster in gapped BLASTP (resp. ungapped BLASTP) procedure; using 4 CPU threads and 1 GPU, H-BLAST can be 2.2 times faster than GPU-BLAST, except in 100–300 cases. Details can be found in the supplement document.

Beside hardware settings, speedups mainly depend on the time share of seeding and ungapped extension step in the whole procedure. We denote this time share by $\phi$. Since the seeding and ungapped extension step is the only one task assigned to GPUs, the greater value $\phi$, the more speedup would gain. Values of $\phi$ in single mode tests of the ungapped BLASTP, the gapped BLASTP and the BLASTX range from 85–94%, 33–88% and 80–88%, respectively. As a result, speedups of ungapped BLASTP are always the best among the three procedures. Furthermore, if the value of $\phi$ is not stable, the speedups would be scattering, like gapped BLASTP.

In Figure 4, we present the relationship between speedups of H-BLAST and the number of CPU threads. Specifically, we show speedups over sequential NCBI-BLAST about multi-threaded NCBI-BLAST, GPU-BLAST and H-BLAST. Both multi-threaded CPU and...
CPU/GPU combinations are considered as a function of the number of available CPU threads. In all tests, H-BLAST gets the observed speedups and outperforms NCBI-BLAST and GPU-BLAST using the same number of CPU threads. As the number of CPU threads increases, speedups of the three procedures in H-BLAST become closer.

Unapped BLAST tests are used to estimate the speedup of the seeding and unapped extension steps, which accounts for 90-95% of the total computing time in the serial version. By 1-threaded results, the computing power contributed by one GPU for H-BLAST can be more than that of 9 CPU cores. The joint speedup of CPUs and a GPU monotone increases as the number of CPU threads increases. When the number of CPU threads is greater than 4, the time share of the cooperation and the serial part of Algorithm 1 increase to 10-30%. Hampered by the preceding cost, speedups are much less than the sum of the number of CPU threads and the speedup gained from the GPU.

5.4 Results of batch mode tests
For rapidly analyzing massive short queries, searching multiple queries concurrently is needed. Many existing solutions have gained promising speedup over NCBI-BLAST on single mode tests, like CUDA-BLASTP and CUDASW++ (Liu et al., 2013). However, they process massive queries one by one. Introduced by (Wang et al., 2003), NCBI-BLAST can align multiple queries concurrently, that would be the silver bullet to be faster than other solutions processing queries one by one in gapped BLASTP and BLASTX procedures. GPU-BLAST and H-BLAST inherited this favorite feature from NCBI-BLAST. The next two experiments focus on the solutions that produce the same alignment results as NCBI-BLAST.

The first experiment was to simulate analyzing metagenomic DNA sequences on a machine with 2 K-20x GPUs in the BLASTX procedure. We ran H-BLAST and NCBI-BLAST on the query set 2, which included 4 groups identified by the length of its short nucleotide reads, about 100-400. Since the BLASTX procedure searches queries chunk by chunk with chunk size 10 000, each query group were processed in 5 chunks sequentially. To measure the speedup archived by H-BLAST on one computer, we compared the performance with 8 and 16 threads NCBI-BLAST.

Results shown on Figures 5 and 6 indicate that H-BLAST is much faster than NCBI-BLAST in all BLASTX tests. Specifically, Figure 5 shows the execution time. Since the running time of H-BLAST and multi-threaded NCBI-BLAST is close to 0 in comparison with that of sequential NCBI-BLAST, the logarithm of the execution time is depicted. The execution time of H-BLAST with 1 CPU thread and 1 GPU is compatible with that of 8-threaded NCBI-BLAST. With 2 CPU threads plus 2 GPUs, H-BLAST is faster than 16-threaded NCBI-BLAST. In Figure 6, speedups over sequential NCBI-BLAST as a function of length of queries are presented. Using 16 CPU threads plus 2 GPUs, H-BLAST can offer a speedup of 21.1 times over sequential NCBI-BLAST. Furthermore, H-BLAST exhibits excellent parallel performance on increasing the number of GPUs. Using 2 H-BLAST instances with 1 CPU thread and 1 GPU for each instance, speedups can approximately double those of using 1 H-BLAST instances. That is, if we double GPUs, speedups double.

Speedups of H-BLAST are uniform on all four query groups in the batch mode BLASTX tests, unlike scattering in the single mode tests. There are two main reasons linked to this fact. One is that the variation of time share of the seeding and ungapped extension step is small, due to small diffusion of queries’ length. Another is using the adaptive load balancer of H-BLAST. Identical to single mode tests, the load balancer is activated when the number of CPU threads is greater than two.

The second experiment was to process massive short protein sequences on a machine with 2 K-40m GPUs in the gapped BLASTP procedure. Since GPU-BLAST required about 7GB RAM on a GPU when using the same computation grids as H-BLAST, K-40m GPUs replaced K-20x GPUs in the experiment. We tested H-BLAST,
 NCBI-BLAST and GPU-BLAST on the query set 3. There were 6 groups identified by the length of its protein reads, about 100–600. In all tests, we turned on the seg filter.

Results shown on Figure 7 of the batch mode gapped BLASTP tests are similar to those of the batch mode BLASTX tests. Speedups of H-BLAST over sequential NCBI-BLAST range from 20.7 to 27.6 with 16 GPU threads and two GPUs, which outperforms NCBI-BLAST and GPU-BLAST.

6 Conclusion and future work

We have proposed H-BLAST, a protein database search tool that accelerates BLASTX and BLASTP on heterogeneous computers with GPUs. H-BLAST speeds up BLASTX and BLASTP by a new fine-grained GPU algorithm, and integrates performance tuning interfaces supporting multi-CPU cores and GPUs. The experiments have demonstrated that H-BLAST can nicely accelerate NCBI-BLAST, and its scalability is excellent. We believe that biologists will benefit from the performance of H-BLAST on relatively inexpensive GPUs.

The next version of H-BLAST will be improved in two directions. Firstly, H-BLAST will take advantages of high performance clusters and a cloud computing environment for handling TB-size short-read query sequences, such as in (The Human Microbiome Project, 2012). Secondly, H-BLAST will support other tools of BLAST with new algorithms, such as (Chen et al., 2015) for BLASTN.

Acknowledgements

The authors of this paper are grateful to Yinghui Huang and Donglin Chen for their assistance in testing.

Funding

This research is supported in part by the Special Project on High-performance Computing under the National Key R&D Program (2016YFB0200602) and supported in part by the Natural Science Foundation of China under grants 11471013, 11571386, 61301180, 91530117.

Conflict of Interest: none declared.

References

Liu,Y. et al. (2013) CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling GPU and CPU SIMD instructions. BMC Bioinformatics, 14, 117.