SOFTWARE

GMSA: An optimized system for multiple sequence alignment of multiple users

Na Bai, Xi Chen, Chen Wang, Shanjiang Tang*, Ce Yu and Hao Fu

Abstract

Background: The multiple sequence alignment (MSA) is a classical and powerful technique for sequence analysis in bioinformatics. With the rapid growth in biological datasets, With the rapid growth in biological datasets, it becomes difficult to get a result within a acceptable time by utilizing MSA. Hence, many works has been done to accelerate a single MSA job on GPUs. However, the optimization of different MSA jobs, in which datasets required are often similar, from users has been largely overlooked. In addition, prior studies only consider the MSA acceleration on GPUs only, which results in the underutilization of host CPUs. Co-run computation can be made use of to increase the utilization of computing resources on both CPUs and GPUs by dispatching workloads onto them simultaneously.

Results: In this paper, we propose GMSA, a robust and efficient MSA system on shared heterogeneous CPU/GPU platforms. To accelerate the computation of jobs from multiple users, data sharing and co-run optimization techniques are adopted in GMSA. Furthermore, we propose a scheduling strategy based on the similarity in datasets or tasks between MSA jobs, and exploit the co-run computation model to fully utilize both CPUs and GPUs. Experiments results show that GMSA can achieve a speedup of up to 32X.

Conclusion: GMSA is a system designed for accelerating MSA jobs with shared input datasets on heterogeneous CPU/GPU platforms. In this system, a strategy was proposed and implemented to find the common datasets among jobs submitted, and a scheduling algorithm is presented based on it. Results showed that it can speed up the total computation of jobs efficiently. The source code of GMSA is available at https://github.com/niefeng2015/GMSA.

Keywords: Heterogeneous; GPU; Multiple sequence alignment(MSA); Data sharing

Introduction

Multiple sequence alignment (MSA) refers to the problem of aligning three or more sequences with or without inserting gaps between the symbols [1]. It is a fundamental tool for sequence analysis in bioinformatics. However, the amount of genomic data is growing exponentially, due to the advances in technology and the evolution in the so-called Next-Generation Sequencing (NGS), including the latest second and third generation equipment. Therefore, MSA software needs to be efficient and scalable to handle large-scale datasets, which may contain hundreds of thousands of sequences.

Many users have urgent jobs that have to be completed in the shorter time. And the MSA jobs often need a large of time as the large datasets. Therefore, it is extremely significant for the MSA system users to get the results quickly. These MSA jobs might reach the system with a requirement for huge input data and produce another huge output data. The input data for these jobs might have shared data (i.e. several jobs ask for same sequences in different input sequence datasets or immediate data computed in). Thus, getting the right strategy that minimizes the amount of computing data and maximizes the global performance is a major issue. Moreover, Data sharing is defined as the case when several related jobs have a common file as an input requirement [2]. This aspect occurs when multiple jobs arrive at the system and might share their input and intermediate data especially if they belong to the same project. Instead of repeated computing, sharing the data might reduce the total execution time by reducing the time needed for a stage-in process that leads to a reduction in the job waiting time.

*Correspondence: tashj@tju.edu.cn
School of Computer Science and Technology, Tianjin University, Yaguan Road , Tianjin, China
Full list of author information is available at the end of the article

Recently, Graphics Processing Unit (GPU) with the Compute Unified Device Architecture (CUDA) programming model is widely used as additional accelerators for time-consuming computations. Moreover, heterogeneous CPU and GPU platform is a desirable way to overlap the computation of the CPU and GPU to fully exploit the computation capability and shorten the runtime [3]. However, in the multiple sequence alignment area, few parallel implementations exist that can address large-scale datasets and produce good acceleration for the situation of multiple users.

In this paper, we present GMSA, a robust and efficient MSA system for multiple users on the shared heterogeneous CPU/GPU platform. GMSA is based on the ClustalW algorithm and mainly tackles the problem of having the share of data and task among jobs in heterogeneous CPU/GPU paradigm and propose a scheduling strategy that handles this issue. The proposed strategy addresses the shared data under the multiple jobs and schedules the jobs to the CPU and GPU that reduce the replication computation that reduces the total execution time. It can perform and optimize multiple sequence alignment automatically for multiple users simultaneously. We propose a strategy based on the sharing given that different MSA users often perform similar datasets. Second, it adopts the co-run computation model that leverages both CPU and GPU for sequence alignment. So it could maximize the entire system utilization. A pre-computation mechanism is developed in GMSA to estimate the computing capacity of CPU and GPU in advance. GMSA then distributes the workloads for CPU and GPU based on this estimation to achieve a better load-balance.

We also compare the GMSA with the strategy that ignoring the sharing data. The results show that GMSA is much faster than the later and achieves an up to 32× speedup.

In summary, the main contributions of this paper are:

a) We take a data sharing approach to optimize the MSA performance when multi-users submitted MSA jobs. Instead of processing data repeatedly, sharing data and tasks will reduce the total executed time by reducing the time needed for a stage-in process that leads to a reduction in the job waiting time.

b) Our GMSA extends ClustalW algorithm for heterogeneous CPU/GPU platform by adopting the co-run computation model for maximizing the overall performance.

Background

In this section, we briefly describe the progressive multiple sequences alignment and the data sharing existing in MSA jobs and then is the heterogeneous CPU/GPU architecture.

Progressive Multiple Sequence Alignment

Multiple sequence alignment is an NP-Complete problem [4]. Therefore, it is often solved by heuristic techniques. Progressive multiple sequence alignment is one of the most popular multiple sequence alignment techniques, in which the pair-wise symbol matching scores can be derived from any scoring scheme or obtained from a substitution scoring matrix such as PAM [5] and BLOSUM [6]. In general, progressive multiple sequence alignment algorithm follows three steps.

1. **Distance matrix:** Perform all pair-wise alignments of the input sequences. Besides, a distance value between each pair of sequences is computed using Dynamic Programming (DP) algorithm. These values are stored in a so-called distance matrix.
2. **Guided tree:** Generate a dendrogram from the distance matrix obtained from the first step, either UPGMA [7] or Neighbor-Joining (NJ) [8] hierarchical clustering is used. The leaves of the tree represent the various sequences. The topology of the tree is totally dependent upon the sequences that are taken, i.e. closely related sequences are placed together and share a common branch in the guided tree and divergent sequences are widely spaced in the guided tree.
3. **Progressive Alignment:** Pair-wise align two sequences (or two pre-aligned groups of sequences) following the guided tree starting from the leaves to the root.

Table 1 The time complexity of ClustalW

Stage	$O(Time)$
Distance matrix	$O(n^2l^2)$
Guided Tree	$O(n^3)$
Progressive Alignment (per iteration)	$O(nl+l^2)$
Progressive Alignment (total)	$O(nl^2+n^2l)$
Total	$O(n^2l^2+n^3)$

ClustalW [9] is a classic implementation of the progressive multiple sequence alignment. Table 1 presents the complexities of different stages of ClustalW (version 2.1) , in which n is the number of sequences and l is the length of each sequence in the dataset needed to be aligned. From the complexities shown in Table 1, we can make the following conclusions about the general runtime behavior of ClustalW:

a) Stage 1 always has a longer runtime than Stage 3;

b) When $n < l^2$, Stage 1 has a longer runtime than Stage 2.

We tested the execution time of ClustalW algorithm with different sequence length and different numbers of sequences. Figure 1 shows The execution time of each stage of ClustalW algorithm for aligning 100 sequences with different length and Figure 2 shows the

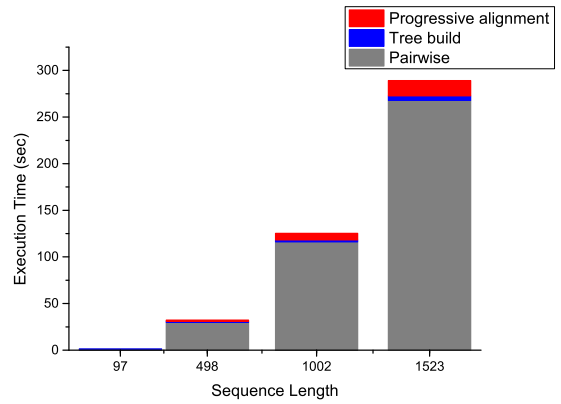


Figure 1 The execution time of each stage of ClustalW algorithm for aligning 100 sequences with different length.

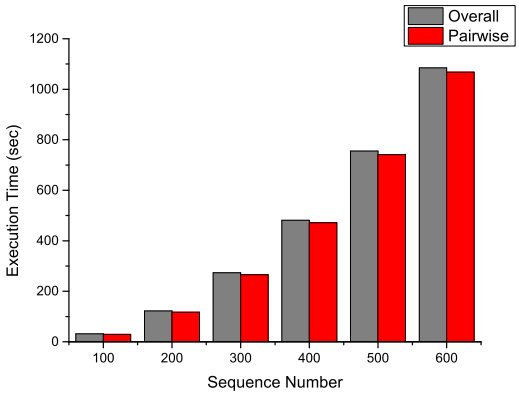


Figure 2 The execution time of pairwise stage and the overall runtime of ClustalW algorithm for aligning different number of sequences with length 498.

execution time of pairwise stage and the overall runtime of ClustalW algorithm for aligning different number of sequences with length 498. It is clear that the distance matrix generating(pairwise) stage is the most time-consuming step. In this paper, we focus on the first stage for performing all pairwise alignments to generate the distance matrix.

Pairwise Sequence Distance Computation

Given a set of n sequences $S = \{S_1, S_2, ..., S_n\}$. For any two sequences S_p, S_q belonging to S of length l_p and l_q, their distance $d(S_p, S_q)$ is defined as follows:

$$d(S_p, S_q) = 1 - \frac{nid(S_p, S_q)}{min\{l_p, l_q\}} \quad (1)$$

where $nid(S_p, S_q)$ denotes the number of exact matches in the optimal local alignment of S_p and S_q.

The value $nid(S_p, S_q)$ can be computed in linear space using three passes: a forward score-only pass using Smith-Waterman (SW) algorithm [10] [11], a reverse score-only pass using SW algorithm and a traceback computation pass using Myers-Miller algorithm [12].

We use the following notations for the SW algorithm: a substitution table sbt, a gap opening penalty ρ, and a gap extension penalty σ. the following recurrences for $1 \le i \le l_p, 1 \le j \le l_q$:

$$\begin{aligned} E(i,j) =& max\{E(i-1,j) - \sigma, H(i-1,j) - \rho - \sigma\} \\ F(i,j) =& max\{F(i,j-1) - \sigma, H(i,j-1) - \rho - \sigma\} \\ H(i,j) =& max\{0, E(i,j), F(i,j), H(i-1,j-1) + \\ & sbt\,(S_p[i], S_q[j])\} \end{aligned} \quad (2)$$

The recurrences are initialized as $H(i,0) = H(0,j) = E(0,j) = F(i,0) = 0$ for $0 \le i \le l_p$ and $0 \le j \le l_q$. The maximum local alignment score max-score is defined as the maximal value in the score matrix H.

The three arrows in Figure 3 show the data dependencies in the alignment score matrix: each cell depends on its left, upper, and upper-left neighbors. This dependency implies that all cells on the same minor diagonal in the alignment score matrix are independent of each other and can be computed in parallel. Thus, the alignment can be computed in minor-diagonal order from the top-left corner to the bottom-right corner in the alignment score matrix. Note that, in order to calculate minor diagonal i only the results of the middle diagonal i-1 and i-2 are necessary and therefore max-score can be found in linear space [13].

Data sharing in Multiple MSA Jobs

In recent years, some enterprises provide a cloud computing platform of which the hardware facilities are shared for multiple users.

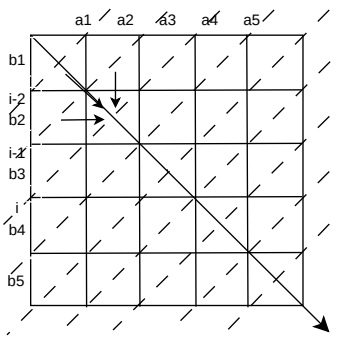


Figure 3 Parallel strategy of dynamic programming matrix

In the field of biological, many algorithms developed would provide a web page version of service to multiple users. And the jobs submitted by these users will share the hardware resources provided by the server. ClustalW, a classic progressive multiple sequence alignment, provides a web service (http://www.genome.jp/tools-bin/clustalw) to users to submit MSA jobs. In this case, when different users submit jobs to the server with similar data, there will be an opportunity to share data in the pairwise stage.

Heterogeneous CPU/GPU Architecture

In recent years, GPU is widely used to accelerate time-consuming tasks. GPU contains a scalable array of multi-threaded processing units known as streaming multi-processors (SM). Although GPU is originally designed to render graphics, general-purpose GPU (GPGPU) [14] breaks this limit and Compute Unified Device Architecture (CUDA) [15] is proposed as a general-purpose programming model for writing highly parallel programs. This model has been proven to be quite successful in programming a broad range of scientific applications.

A heterogeneous CPU/GPU platform is proposed to achieve the best performance. Figure 4 depicts this architecture. CPU and GPU are connected by PCI-E and both of them have their own memory space. There are two main methods for heterogeneous CPU/GPU programming as follows:

(i) Consider CPU as a master and GPU as a worker. CPU handles the work assignment, data distribution, etc. GPU is responsible for the whole computation.

(ii) CPU still plays the role of a master. At the same time, it takes a portion of GPU's computations.

The former method has a clear work division between CPU and GPU. However, it wastes the computing resource of CPU regrettably. The latter method has a better performance. But, it also brings in some tricky issues such as load balance and extra communications between CPU and GPU.

Additionally, there are several different parallel programming approaches on multi-core systems:

(i) Low-level multi-tasking or multi-threading such as POSIX Thread (pThread) library [16].

(ii) High-level libraries, such as Intel Threading Building Blocks [17], which provides certain abstractions and features attempting to simplify concurrent software development.

(iii) Programming languages or language extensions developed specifically for concurrency, such as OpenMP [18].

(iiii) A general-purpose programming model, CUDA(Compute Unified Device Architecture) [15] is proposed for writing highly parallel programs on the GPU end.

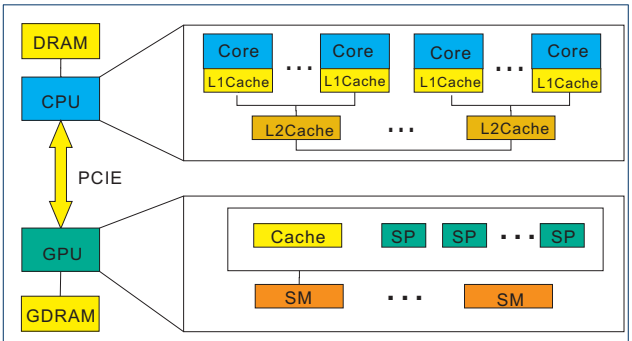


Figure 4 The heterogeneous CPU/GPU architecture

GMSA system

In this section, we introduce the design and implementation of GMSA. We start by first showing the challenges and our corresponding strategies. Then, we formulate the problem of data sharing for multi-users. Finally, we describe the design and implementation details of GMSA in the next section.

Challenges and Approaches

There are several key issues that we need to address for MSA in practice. In the following, we highlight these challenges and then give our corresponding solutions.

Redundant work of Multiple MSA jobs in the shared platform. In practice, it is most likely that different users often run MSA jobs with similar datasets in the shared platform. But most MSA systems only deal with the multiple jobs separately and ignore the relationship between these jobs which leads to a lot of redundant work and then degrades performance.

Data sharing across multiple MSA jobs. Taking into account that different MSA users often perform similar datasets, there are some opportunities for data sharing. Besides, we have analyzed that the first stage of ClustalW, distance matrix computing, is the most time-consuming part of a straightforward implementation of ClustalW. Therefore, we propose a data sharing based scheduling strategy for multiple MSA jobs, as we discussed in subsection Data Sharing.

Low utilization problem of the heterogeneous CPU/GPU platform. For further improving the performance of GMSA, parallelization is an efficient method. However, most GPU based MSA systems perform all computing tasks on GPU only. It's necessary to exploit the computing power of CPU and GPU at the same time.

Co-run computation model. To fully utilize all available computing capacities of the heterogeneous CPU/GPU platform, it is crucial to enable CPU and GPU to work concurrently for workload computations

(i.e., co-run computation), which means that CPU also takes a portion of computation instead of waiting for GPU to do all the computing work. We designed a pre-computation mechanism to decide how to distribute workload between CPU and GPU.

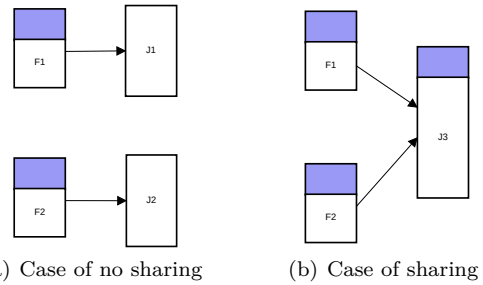
Data Sharing

The huge growth of data gives rise to problems of data staging and data sharing [2]. If a set of MSA jobs submitted to the system have a percentage of shared data and work among them, then how to deal with the common data with each job is a problem. However, the previous methods schedule the common task with common data tautologically, which decreases the system utilization and performance. Therefore, handling the shared data and tasks among MSA jobs is an important issue.

To illustrate this problem clearly, Figure 5 depicts a simple scenario of data sharing. The figure has two MSA jobs that perform the same subtask. Job $J1$ asks for file $F1$ while job $J2$ asks for file $F2$. Now, assume job $J1$ and job $J2$ comes together and the file $F1$ and file $F2$ have a part of same data(the blue part). In this scenario, the same data should be computed twice as described in Figure 5(a). To improve performance for urgent jobs, Figure 5(b) shows a possible policy that the scheduler merges the common computation of the same data by combining the two jobs to single job $J3$. This method reduces the time needed to re-compute the same data and then leads to a decrease of the total execution time.

Typcially, we define the data sharing problem formally as follows:

Given a set of submitted jobs J, which has n jobs. Each job $j_i \in J$ has an input data set d_i, where $1 \leq i \leq n$. Now, let $D = \{d_1, d_2, ..., d_n\}$, while d_i is a dataset, then, the data sharing exist if: $|d_1 \cup d_2 \cup ... \cup d_n| < |d_1| + |d_2| + ... + |d_n|$.



(a) Case of no sharing (b) Case of sharing

Figure 5 The figure of sharing. Two jobs ask for two files. The blue block represents the same data or the merged task

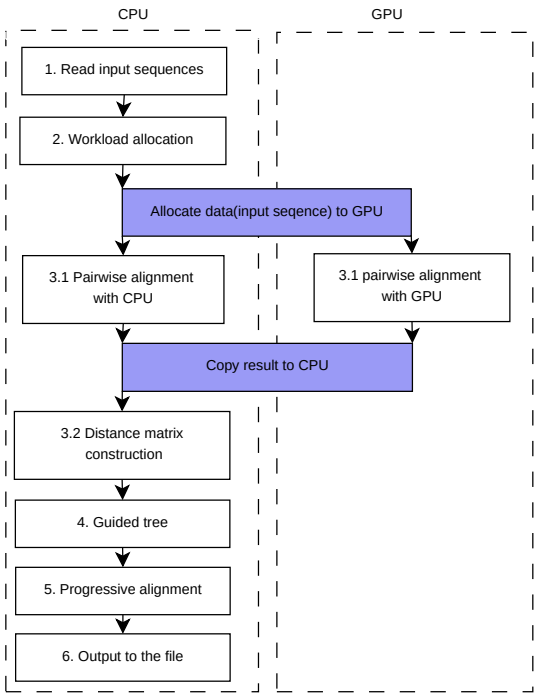


Figure 6 The overall flow of GMSA

Implementation

In this section, we first explain the execution overflow of GMSA. Then we discuss our strategy for MSA of multi-users. At last, we describe the implementation details of GMSA on the heterogeneous CPU/GPU platform.

Execution Overflow

GMSA is a heterogeneous CPU/GPU system, using CUDA and OpenMP for parallelization. To reduce the total execution time, CPU also carries out a part of alignment tasks rather than waiting for GPU to deal with the whole computing work. The execution flow of GMSA is shown in Figure 6. It contains the following steps:

1 **Data Reading:** GMSA reads all sequences submitted from all users into the host (CPU) memory. According to the scheduling strategy based on data sharing, if the same sequences occur in the different users then these sequences will be copied only once. After the pre-computation process, the part of data that would be handled by GPU will be sent to the device (GPU) memory.

2 **Workload Distribution:** GMSA performs a pre-computation process to decide how to distribute workload for CPU and GPU. In this process, a small number of sequences are aligned in advance to evaluate the computing capacity

of CPU and GPU. The detailed information of workload allocation will be described in Subsection *Workload Distribution*.

3 **Distance Matrix Computation:** This stage computes the distance of every pairwise alignment, then constructs the distance matrix. This part will be executed concurrently on the CPU and GPU devices. The detailed method will be discussed in Subsection *Data Sharing across Multi-users*.

4 **Guided Tree Generating:** This step uses the distance matrix obtained from the previous step and forms a guided-tree for every user.

5 **Progressive Alignment:** According to the guided tree, the most relevant sequences or group of sequences are aligned first, and at the end, the most divergent sequences are aligned to get the final result.

6 **Output to the file:** When both CPU and GPU finish their job, GMSA gathers the result from GPU and CPU, then merges the inserted gaps to generate the final result.

Data Sharing across Multi-users

The problem of data sharing exists if a set of jobs have a ratio of overlapping in their input data. This problem arises due to the huge growth in data sizes and its influence on replicated calculation, which affects the total execution time. To minimize the amount of overhead needed to manage the data, this work assumes that the sharing of data and tasks exists in a group of jobs that belong to one project and arrive simultaneously to the system.

The proposed strategy is called Data Sharing Multi-users (DSMU). It merges the same subtasks from multiple jobs to one task and computes them only once. To specify, DSMU has been done by finding the same sequences among the datasets submitted by the multiple users and execute the relevant tasks only once. The main steps for handling the sharing of data and tasks, as shown in Algorithm 1, are listed below:

1 Managing the user queue. First, if the dataset submitted is similar to the pre-stored datasets, the corresponding task is first invoked and reuse the pre-stored distance matrix to reduce the computation time. Otherwise, the corresponding task waits in the user queue.

2 Searching the datasets submitted and estimating the sharing opportunity. First, estimating the similarity among several datasets submitted by the users in the user queue. If such of them have a high degree of similarity, the corresponding tasks are invoked and executed together as a group.

3 Copying the same input sequences submitted by the users in the same group to memory only once.

Algorithm 1 DSMU

```
for numOfUser > 0 do
  for all users do
    Examine if the dataset of the user is similar to the pre-stored datasets;
    if true then
      Invoke the relevant task;
    else
      The relevant task waits in the user quene;
    end if
  end for
  for all users do
    Find the data sharing opportunity between datasets submitted by different user;
    if some datasets have high similarity then
      The revelant tasks are invoked and executed together as a group;
      Save the distance matrix required for multiple users;
    end if
  end for
  if there are not new jobs submitted then
    Invoke the earliest submitted task;
  else
    continue;
  end if
end for
```

4 Saving the part of reusable matrixes and classifying them into different groups according to the gene types. Execute the task with the same data once then save the results(distance matrix) required for multiple users.

Workload Distribution

One of the key issues of a heterogeneous system is load-balance. Since CPU and GPU differ greatly in computing capability, a heterogeneous system needs a way to estimate this differential to achieve the load-balance. Suppose the execution time of CPU and GPU are T_1 and T_2, then the total time of the pairwise alignment is the maximum value of T_1 and T_2. So the best performance is achieved when the computations of CPU and GPU are completely overlapped, which means $T_1 = T_2$.

In GMSA, a pre-computation process is performed to decide how to distribute the workload to CPU and GPU as shown in Algorithm 2. In this process, both CPU and GPU compute the same number of sequences

Algorithm 2 Pre-computation for Workload Distribution

Input: Datasets of sequence.

1: Extract a small portion of input sequence;
2: Execute distance matrix computation on CPU and store executing time T_1;
3: Execute distance matrix computation on GPU and store executing time T_2;
4: $R = \frac{t_1}{t_2}$;
5: workload on CPU: $\frac{w}{R+1}$;
6: workload on GPU: $\frac{w*R}{R+1}$;

(a small portion of input sequences). GMSA compares the execution time of CPU and GPU (denoted as T_1 and T_2) to calculate a ratio of computing capability R, $R = \frac{T_1}{T_2}$. According to this ratio, GMSA then assigns $\frac{w}{R+1}$ pairwise alignments to CPU and the rest $\frac{w*R}{R+1}$ pairwise alignments to GPU, where $w = \frac{n*(n-1)}{2}$ is the number of pairwise alignments of input sequences and n is the number of input sequences.

Parallel optimization

The multiple alignments of n sequences require completing the distance matrix, performing $w = \frac{n*(n-1)}{2}$ pairwise alignment operations between each pair of sequences. There are no dependencies between them so that these alignment operations can be executed in parallel. Besides, each pairwise alignment is partially distributed among several executing threads, thus obtaining two levels of parallelism.

In the CPU end, OpenMP is used to accelerate the pairwise alignment in a coarse-grained manner. The computation of the DP matrix and the backtracking of score matrices are mapped onto different threads. In other words, each thread is responsible for aligning a pair of sequences. Threads are working independently, and each thread handles its own memory space including allocating and releasing the resources. The number of threads is usually set to the number of cores in the CPU processor.

Typical GPU processor consists of multiple identical instances of computation units called Stream Multiprocessors (SM), which consists of a number of Stream Processor(SP). SM is the unit of computation, in which a group of threads, belong to a thread block, execute in parallel. Each thread will be launched to an SP included in SM. A CUDA program consists of several *kernels*. When a kernel is launched, a two-level thread structure is generated. The top level is called *grid*, which consists of one or more *blocks*, denoted as B. Each block consists of the same number of threads, denoted as T. The whole block will be assigned to one SM.

Like the implementation on CPU, each thread in a kernel aligns one sequence with the center sequence, which means each kernel computes $B \times T$ sequences. As we discussed early, GPU handles $\frac{Rw}{R+1}$ pairwise alignments totally. On the GPU end, $\frac{Rw}{(R+1)(BT)}$ kernels are executed. Since each kernel computes the same number of pairwise alignments and the DP matrices computed by each kernel are not used in the next kernel, we could recycle these memory resources. Before the first kernel is invoked, GMSA allocates the memory required for storing the DP matrices in one kernel. And when the last kernel finishes, the memory will be released. A DP matrix is stored in a one-dimensional way in the global memory of GPU. For example, Tesla K80 card has 12GB global memory. So in theory, each kernel can simultaneously compute 150000 pairwise alignment in terms that the length of sequences is 200 and each element in the DP matrix is a *short* type which are two bytes each.

Theoretically, the parallelization approach of stage 1 can also be applied to the computation of similarity matrices in Stage 3 of the ClustalW algorithm.

GMSA uses constant memory to store the read-only parameters and the substitution table is loaded into shared memory.

Results and Discussion

The performance of GMSA with data sharing is shown in this section. We evaluate GMSA on a heterogeneous CPU/GPU workstation.

Experimental Setup

Experimental Platform: The experiments are carried out on a heterogeneous CPU/GPU platform, which has two Intel Xeon E5-2680 2.4 GHz processors and four NVIDIA Tesla K80 graphic cards. Linux 4.4.0 is deployed and CUDA Toolkit 8.0 is used to compile the program. Each CPU consists of 14 cores. The detailed specification of Tesla K80 is shown in Table 2.

Table 2 GPU hardware specifications

Tesla K80	
CUDA compute capability	3.7
CUDA cores	2496
GPU clock rate(MHz)	560
Total amount of global memory(GB)	12
Memory bus width	384-bit
Shared memory size per block(bytes)	49152
The number of registers available per block	65536

Datasets: Four protein datasets were used with length 97, 498, 1002 and 1523. They were downloaded from the NCBI website(http://www.ncbi,nlm.nih.gov/), and each of them can be classified into several test sets according to the number of sequences. Besides, we redesigned the datasets according to the similarity $S \in [0, 1]$. For example, the datasets contain the 90% same sequences, i.e. having the similarity of 0.9.

Metrics: In the experimental tests, a program named "bali_score" that was downloaded from the Balibase benchmark [19](http://www.lbgi.fr/balibase/) was used to compare the alignment results of GMSA with those calculated by ClustalW v2.1. In "bali_score" program, the reference alignment is set to the alignment results of ClustalW v2.1, and then the alignment results of GMSA is the test alignment. Two scores, Sum-of-Pairs score (SP) and the Total-Column score (TC), were used to estimate the accuracy of a

test alignment comparing to the reference alignment. The maximal values of these two scores both are 1, means that the test alignment is identical to the reference alignment in aligning.

Baselines: To show the efficiency and accuracy of GMSA, we compare GMSA, ClustalW [20] and CUDA-ClustalW [21]. ClustalW is one of the popular MSA algorithms. CUDA-ClustalW proposed a GPU version of ClustalW v2.1 without checking on data sharing. It can achieve a considerable acceleration of the overall execution time compared with ClustalW v2.1.

Table 3 Overall execution time by ClustalW v2.1 and G-MSA (CPU only) for two users with 300 sequences and different similarity.

L=97	S=0.5	S=0.7	S=0.9	S=1
ClustalW	25.19s	25.32s	25.20s	24.49s
GMSA(CPU)	26.39s	22.92s	20.074s	18.594s
L=498	S=0.5	S=0.7	S=0.9	S=1
ClustalW	9m56.28s	9m54.11s	9m55.14s	7m29.61s
GMSA(CPU)	10m37.78s	9m9.37s	8m2.03s	5m3.11s
L=1002	S=0.5	S=0.7	S=0.9	S=1
ClustalW	39m1.02s	39m1.28s	39m1.20s	39m1.28s
GMSA(CPU)	41m47.40s	39m37.64s	35m16.98s	29m28.17s
L=1523	S=0.5	S=0.7	S=0.9	S=1
ClustalW	92m1.12s	91m22.56s	91m43.39s	91m55.12s
GMSA(CPU)	93m17.03s	90m17.24	75m10.69s	69m04.00s

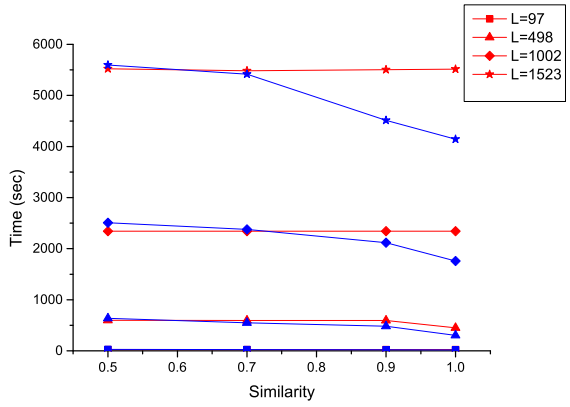


Figure 7 Overall execution time of ClustalW v2.1 and GMSA (CPU only) for two users with 300 sequences to be aligned in terms of different similarity. The red lines represent the ClustalW v2.1 execution time and the blue lines represent the GMSA(CPU only) execution time. additionally, the different symbols represent the different length of sequences as shown in this figure.

Table 4 Overall execution time by ClustalW v2.1 and GMSA for a single user alignment with 300 sequences of different length.

Sequence Length	97	498	1002	1523
ClustalW	13.20s	4m56.54s	18m56.93s	46m57.90s
CUDA-ClustalW	11.12s	19.27s	48.57s	102.54s
GMSA	10.8s	22.10s	47.92s	96.49s

Table 5 Overall execution time by ClustalW v2.1 and GMSA for a single user alignment with 1000 sequences of different length.

Sequence Length	97	498	1002	1523
ClustalW	100.23s	1040s	-	-
CUDA-ClustalW	40.33s	140.45s	370.89s	780.45s
GMSA	22.20s	111.10s	349.22s	718.23s

Shared-Based Algorithm for Distance Matrix

We defined the similarity(S) as the quotient of the number of identical sequences among the input datasets and the total sequences number of the input datasets.

Table 3 shows the overall execution time of ClustalW v2.1 and GMSA(CPU only) for two users that each has 300 sequences with different length(97, 498, 1002 and 1523) and similarity(0.5, 0.7, 0.9 and 1). From Table 3 and Figure 7, it is clear that the higher the similarity is, the shorter execution time is required. GMSA shows the better performance in the multi-users environment especially when datasets submitted by multi-users have high similarity.

Co-run Parallel Optimization

Table 4 and 5 show the overall execution time by ClustalW v2.1 and GMSA for a single user on 300 and 1000 sequences, respectively. From Table 4, there is less performance gain by GMSA with the length of 97 as the data size is too small. For the other test sets, execution time by GMSA is all shorter than that of ClustalW v2.1 and CUDA-ClustalW. From Table 5, all the execution time of GMSA are shorter than others. Moreover, the performance by GMSA is better than datasets with less number of sequences. Table 5 shows that the GMSA has better performance than CUDA-ClustalW when employing the co-run computation model to maximize the utilization of both CPU and GPU on heterogeneous CPU/GPU platform. But because of the pre-computation to distribute the workload to CPU and GPU and the communication between them, the acceleration of CUDA-ClustalW is not significant.

Efficiency and Scalability

We estimate the accuracy of a test alignment by comparing it to the reference alignment. The maximal values of these two scores SP(Sum-of-Pairs score) and

Table 6 SP and TC scores by ClustalW v2.1 and GMSA on 300 and 1000 sequences.

Number of sequence	Sequence length	SP score	TC score
300	97	0.99	0.99
	498	1	1
	1002	1	1
	1523	1	1
1000	97	0.99	0.99
	498	1	1
	1002	1	1
	1523	1	1

Table 7 Overall execution time by the different number of users on similarity 0.9. The length is 498 and number of sequences submitted by each user is 300.

	2	4	6	8	10	12
ClustalW(s)	595.07	1186.34	1734.49	2197.37	2783.68	3361.34
CUDA-Clustal(s)	36.05	73.38	111.51	148.47	156.68	191.23
GMSA(s)	26.68	41.65	56.08	73.69	88.96	104.73

TC(the Total-Column score) are both 1, meaning that the test alignment is identical to the reference alignment. Table 6 shows the SP and TC scores by GMSA on these eight datasets. From Table 7, we can observe that most of the alignment results of GMSA are identical to those of ClustalW v2.1. The alignment results of minor cases are not identical, but they are highly similar.

As an indication of how GMSA scales with datasets sizes, Table 7 shows the overall execution time with different number of users on similarity 0.9. The length of submitted sequences is 498 and the number of sequences submitted by each user is 300. Table 7 shows the overall execution time of GMSA with different number of users. It is clear that the larger the number of users is, the more time it would cost, as shown in Figure 8. Moreover, in all tests, the running time goes up linearly as the number of sequences increases, which demonstrates a great scalability of GMSA. Figure 9 shows the time speedup of the same experiments. The best speedup is not achieved in the first case, in which the runtime of the pre-computation and initialization makes up a considerable proportion since there are a low number of sequences. But with the increase of the number of sequences, the real computation would dominate most of the running time, which in turn reports a better time speedup.

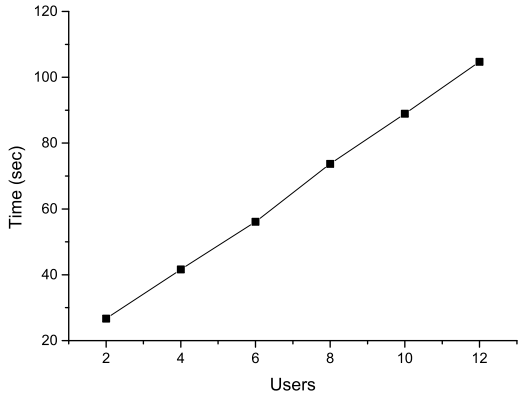


Figure 8 The overall execution time with different number of users on similarity 0.9.

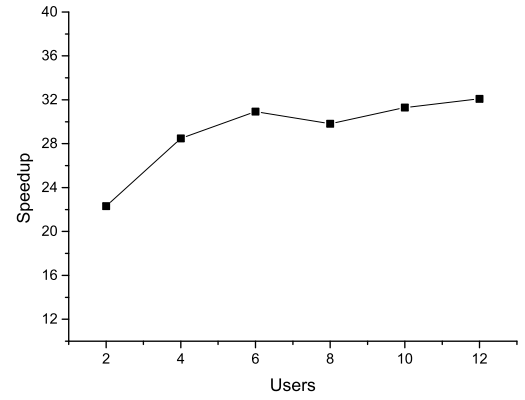


Figure 9 the time speedup of GMSA with different number of users on similarity 0.9.

Related Work

There are many parallel techniques which proposed to accelerate Clustal algorithm. Besides, it is necessary to optimize the performance of the multiple jobs. In this section, we review them from the following three aspects.

Multiple sequence alignment tools. The Clustal series [22] are the most widely used tools for global multiple sequence alignment. The first Clustal program combined the progressive alignment strategy with dynamic programming using a guided tree. Then ClustalW incorporated a number of improvements to the alignment algorithm, including sequence weighting, position-specific gap penalties and the automatic choice of a suitable residue comparison matrix [23]. SGI parallel Clustal [24] was the first attempt to accelerate ClustalW by parallelizing all three stages on a shared memory SGI Origin machine using OpenMP. It shows up to 10 folds speedup when running on 16 CPUs. ClustalW-MPI [25] was implemented on workstation clusters with distributed memory architecture. It uses MPI with dynamic scheduling to parallelize

the searching of sequences having the highest divergence from all other sequences. It achieved an overall speedup of 4.3× using 16 processors. A multi-threaded algorithm was proposed in [26] and was incorporated in ClustalW-MPI to make better use of CPU cycles and memory. pCLUSTAL [27] is a parallel version of ClustalW using MPI. It distributes the pairwise alignment computation on available processors achieving up to 10× speedup on a 64 node PC cluster.

Furthermore, there are several other MSA tools such as T-coffee [28] and MUSCLE [29], that are also based on the distance matrix computation in the pairwise phase.

Data sharing across multiple jobs. Taking into account that different jobs often perform similar work, there are many opportunities for sharing data and tasks.

Among the MapReduce systems, Hive [30] supports user-defined scan-sharing. Given two jobs reading from the same file, Hive adds a new, preprocessing MapReduce job. MRshare [31] focus on the sharing across multiple queries in MapReduce. MRshare transforms a batch of queries into a new batch that will be executed more efficiently, by merging jobs into groups and evaluating each group as a single query.

An
MPI applicationc [32] introduces intra-parallelization, a solution that avoids replicating all computation by introducing work-sharing between replicas. This paper proposes a new fault-tolerant technique for MPI HPC applications by introducing collaboration between the replicas of a logical process to execute computational intensive kernels more efficiently.

A cloud computing application [33] tackles the problem of scheduling data-intensive jobs that might have shared input files among them. This paper proposes an algorithm that gives a ready time for each file to be used later by the scheduler if another job asks about it. The results show a big impact of shared handling the shared files on the total transfer time and the throughput.

Coupled CPU-GPU architecture. There are a number of studies on task scheduling in heterogeneous computing. SMK(Simultaneous Multikernel) [34] proposed a fine-grained fair sharing scheduler, which can increase resource utilization while maintaining the resource fairness among kernels by scheduling kernels from different applications dynamically. Zhang et al. [35] had a performance study of scheduling tasks of an application to both CPU and GPU simultaneously by developing a benchmark suite called Rodinia. Tang et al. [36] proposed EMRF(Elastic Multi-Resource Fairness) for balancing fairness and efficiency in coupled CPU-GPU architectures.

However, there is little research about the specific application with data sharing on heterogeneous CPU/GPU architectures.

Conclusion

This paper presents the GMSA system which tackles the problem of scheduling MSA jobs that might have shared input datasets among them. GMSA takes a strategy that finding the common pairs of sequences and then computing them only once by the scheduler, which can reduce redundant work and then decrease the computation time. To utilize the overall resource of both CPU and GPU, GMSA is designed on the heterogeneous CPU/GPU platform employing the co-run computation model. The results show a big impact of sharing datasets on the total computing time and GMSA achieves an up to 32× speedup and outperforms the system without the data sharing policy.

Besides, GMSA focus on the stage of pairwise distance computation, which is also the basis of several other MSA tools such as T-coffee [28] and MUSCLE [29]. Hence, these tools could see a similar speedup from the accelerator presented in this paper.

Abbreviations

CPU: Central processing unit
GPU: Graphics processing unit
CUDA: Compute unified device architecture
pthread: POSIX thread
OpenMP: Open multiprocessing
MPI: Message passing interface
MSA: Multiple sequence alignment
NCBI: National center for biotechnology information
PCIe: Peripheral component interconnect express

Declarations

Availability of data and material

- Project name :GMSA
- Project home page: https://github.com/niefeng2015/GMSA
- Operating system(s): Linux 64-bit
- Programming language: C++, CUDA, OpenMP
- Other requirements: CUDA-capable GPU
- license: GUN GPL
- Any restrictions to use by non-academics: None

Author's contributions

NB conceptualized the study, carried out the design and implementation of the algorithm, analyzed the results and drafted the manuscript; XC and CW participated the parallel implementation of the system. SJT, CY and HF participated analysis of the results and contributed to revise the manuscript. All authors read and approved the final manuscript.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K80 GPU used for this research and the support of datasets by NCBI.

Competing interests

The authors declare that they have no competing interests.

Ethics approval and consent to participate

Not applicable.

Consent for publication
Not applicable.

Funding
This work is supported by the Joint Research Fund in Astronomy (U1531111) under cooperative agreement between the National Natural Science Foundation of China (NSFC) and Chinese Academy of Sciences (CAS), the National Natural Science Foundation of China (11573019, 61602336).

References

1. Karadimitriou, K., Kraft, D.H.: Genetic algorithms and the multiple sequence alignment problem in biology. In: Proceedings of the Second Annual Molecular Biology and Biotechnology Conference, pp. 1–7 (1996)
2. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared. In: 2008 Grid Computing Environments Workshop, pp. 1–10 (2008)
3. Schmidt, B.: Bioinformatics: High Performance Parallel Computer Architectures. CRC Press, Florida (2010)
4. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. Journal of computational biology **1**(4), 337–348 (1994)
5. Dayhoff, M.O., Schwartz, R.M.: A model of evolutionary change in proteins. In: In Atlas of Protein Sequence and Structure (1978). Citeseer
6. Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks. Proceedings of the National Academy of Sciences **89**(22), 10915–10919 (1992)
7. Sneath, P.H., Sokal, R.R.: Numerical taxonomy. the principles and practices of numerical classification. WF Freeman and Co., San Francisco **573** (1973)
8. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular biology and evolution **4**(4), 406–425 (1987)
9. Thompson, J.D., Higgins, D.G., Gibson, T.J.: Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic acids research **22**(22), 4673–4680 (1994)
10. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of molecular biology **147**(1), 195–197 (1981)
11. Gotoh, O.: An improved algorithm for matching biological sequences. Journal of molecular biology **162**(3), 705–708 (1982)
12. Myers, E.W., Miller, W.: Optimal alignments in linear space. Computer applications in the biosciences: CABIOS **4**(1), 11–17 (1988)
13. Liu, Y., Schmidt, B., Maskell, D.L.: Msa-cuda: multiple sequence alignment on graphics processing units with cuda. In: 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, pp. 121–128 (2009)
14. GPGPU. http://gpgpu.org/
15. Nvidia: Cuda programming guide version 1.1 (2007)
16. Garcia, F., Fernandez, J.: Posix thread libraries. Linux Journal **2000**(70es), 36 (2000)
17. Reinders, J.: Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism. " O'Reilly Media, Inc.", sebastopol (2007)
18. Dagum, L., Menon, R.: Openmp: an industry standard api for shared-memory programming. IEEE computational science and engineering **5**(1), 46–55 (1998)
19. Thompson, J.D., Koehl, P., Ripp, R., Poch, O.: Balibase 3.0: latest developments of the multiple sequence alignment benchmark. Proteins: Structure, Function, and Bioinformatics **61**(1), 127–136 (2005)
20. Thompson, J.D., Higgins, D.G., Gibson, T.J.: Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic acids research **22**(22), 4673–4680 (1994)
21. Hung, C.-L., Lin, Y.-S., Lin, C.-Y., Chung, Y.-C., Chung, Y.-F.: Cuda clustalw: An efficient parallel algorithm for progressive multiple sequence alignment on multi-gpus. Computational biology and chemistry **58**, 62–68 (2015)
22. Thompson, J.D.: The clustal series of programs for multiple sequence alignment. The Proteomics Protocols Handbook, 493–502 (2005)
23. GHALEB, F.F., REDA, N.M., AL-NEAMA, M.W.: An overview of multiple sequence alignment parallel tools. CSCCA (2013)
24. Mikhailov, D., Cofer, H., Gomperts, R.: Performance optimization of clustal w: Parallel clustal w, ht clustal, and multiclustal. SGI ChemBio (2001)
25. Li, K.-B.: Clustalw-mpi: Clustalw analysis using distributed and parallel computing. Bioinformatics **19**(12), 1585–1586 (2003)
26. Marucci, E.A., Zafalon, G.F., Momente, J.C., Pinto, A.R., Amazonas, J.R., Shiyou, Y., Sato, L.M., Machado, J.M.: Using threads to overcome synchronization delays in parallel multiple progressive alignment algorithms. American Journal of Bioinformatics, 50–63 (2012)
27. Cheetham, J., Dehne, F., Pitre, S., Rau-Chaplin, A., Taillon, P.J.: Parallel clustal w for pc clusters. In: International Conference on Computational Science and Its Applications, pp. 300–309 (2003). Springer
28. Notredame, C., Higgins, D.G., Heringa, J.: T-coffee: A novel method for fast and accurate multiple sequence alignment. Journal of molecular biology **302**(1), 205–217 (2000)
29. Edgar, R.C.: Muscle: multiple sequence alignment with high accuracy and high throughput. Nucleic acids research **32**(5), 1792–1797 (2004)
30. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment **2**(2), 1626–1629 (2009)
31. Nykiel, T., Potamias, M., Mishra, C., Kollios, G., Koudas, N.: Mrshare: sharing across multiple queries in mapreduce. Proceedings of the VLDB Endowment **3**(1-2), 494–505 (2010)
32. Ropars, T., Lefray, A., Kim, D., Schiper, A.: Efficient process replication for mpi applications: Sharing work between replicas. In: Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International, pp. 645–654 (2015)
33. Mehdi, N.A., Holmes, B., Mamat, A., Subramaniam, S.K.: Sharing-aware intercloud scheduler for data-intensive jobs. In: Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference On, pp. 22–26 (2012)
34. Wang, Z., Yang, J., Melhem, R., Childers, B., Zhang, Y., Guo, M.: Simultaneous multikernel: Fine-grained sharing of gpgpus. IEEE Computer Architecture Letters **15**(2), 113–116 (2017)
35. Zhang, F., Zhai, J., Chen, W., He, B., Zhang, S.: To co-run, or not to co-run: A performance study on integrated architectures, 89–92 (2015)
36. Tang, S., He, B.S., Zhang, S., Niu, Z.: Elastic multi-resource fairness: Balancing fairness and efficiency in coupled cpu-gpu architectures. In: High Performance Computing, Networking, Storage and Analysis, SC16: International Conference For, p. 75 (2017)