MGUPGMA: A Fast UPGMA Algorithm With Multiple Graphics Processing Units Using NCCL

Evolutionary Bioinformatics
Volume 13: 1–7
© The Author(s) 2017
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1176934317734220

Guan-Jie Hua[1], Che-Lun Hung[2], Chun-Yuan Lin[3], Fu-Che Wu[4], Yu-Wei Chan[5] and Chuan Yi Tang[1,6]

[1]Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. [2]Big Data Research Center, Department of Computer Science and Communication Engineering, Providence University, Taichung, Taiwan. [3]Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan, Taiwan. [4]Department of Computer Science and Communication Engineering, Providence University, Taichung, Taiwan. [5]College of Computing and Informatics, Providence University, Taichung, Taiwan. [6]Department of Computer Science and Information Engineering, Providence University, Taichung, Taiwan.

ABSTRACT: A phylogenetic tree is a visual diagram of the relationship between a set of biological species. The scientists usually use it to analyze many characteristics of the species. The distance-matrix methods, such as Unweighted Pair Group Method with Arithmetic Mean and Neighbor Joining, construct a phylogenetic tree by calculating pairwise genetic distances between taxa. These methods have the computational performance issue. Although several new methods with high-performance hardware and frameworks have been proposed, the issue still exists. In this work, a novel parallel Unweighted Pair Group Method with Arithmetic Mean approach on multiple Graphics Processing Units is proposed to construct a phylogenetic tree from extremely large set of sequences. The experimental results present that the proposed approach on a DGX-1 server with 8 NVIDIA P100 graphic cards achieves approximately 3-fold to 7-fold speedup over the implementation of Unweighted Pair Group Method with Arithmetic Mean on a modern CPU and a single GPU, respectively.

KEYWORDS: Phylogenetic tree, UPGMA, GPU, parallel computing, multiple GPUs

RECEIVED: May 8, 2017. **ACCEPTED:** September 6, 2017.

PEER REVIEW: Two peer reviewers contributed to the peer review report. Reviewers' reports totaled 599 words, excluding any confidential comments to the academic editor.

TYPE: Review

FUNDING: The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was partially supported by the Ministry of Science and Technology under grant MOST103-2632-E-126-001-MY3 and MOST106-3114-E-029-001.

DECLARATION OF CONFLICTING INTERESTS: The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

CORRESPONDING AUTHOR: Che-Lun Hung, Big Data Research Center, Department of Computer Science and Communication Engineering, Providence University, Taichung 43301, Taiwan. Email: clhung@pu.edu.tw

Introduction

In the past decades, biologists have used phylogenetic trees to observe the meaningful insight into biology. A phylogenetic tree is capable of showing the evolutionary relationships among a variety of organisms based on similarities between their physical or genetic residues. The computational algorithms to construct phylogenetic trees can be mainly classified into 2 categories: the distance-based ones and the character-based ones. The distance-based methods construct a phylogenetic tree by calculating pairwise genetic distances between taxa. Different from the distance-based methods, the concept of the character-based methods is to construct the tree by aligning sequences of nucleotide or amino acid residues. In both categories, the similarity of each pair individuals usually will be represented by a distance matrix. The clusters of organisms are grouped by similarity of multiple genome sequences called operational taxonomic units (OTUs).[1] Operational taxonomic units show different taxonomic levels of species and have been frequently used in microbial classification such as 16S or 18S ribosomal RNA sequence analysis.

Two commonly used distance-based approaches, namely, Unweighted Pair Group Method with Arithmetic Mean (UPGMA) and Neighbor Joining (NJ), have been used in many biological research works, and they are also integrated into many multiple sequence alignment tools. In general, the time complexity of original UPGMA algorithm is $O(n^3)$,[2] and the improved UPGMA is able to be optimally reduced to $O(n^2)$.[3] For original NJ algorithm, the time complexity is $O(n^4)$, and the complexity of improved algorithm, named FastNJ,[4] is able to reach $O(n^2)$ in the best cases.

Due to the time complexities of UPGMA and NJ, the number of sequences is critical to the computational performance. The execution time of the transitional UPGMA is more than 4000 seconds when the number of OTU is more than 1000 seconds.[5] Due to next-generation sequencing (NGS) technologies,[6] the sequencing data have been increasing faster than computers can keep up. The size of sequence data has increased recently from 1 GB to 1 TB in a single sequencing run. The computational performance of analyzing such huge amount data is unacceptable. The NGS has forced researchers to develop new software to enhance the performance. For constructing a phylogenetic tree from such huge amount of data set, the methodologies, such as caching and parallelization, have been proposed to satisfy this requirement. These methods are introduced to optimize throughput. Message Passing Interface (MPI)[7] is a specification of message-passing libraries which address the message-passing parallel programming model for parallel computation on cluster systems connected by networks. pNJtree[8] is a parallel program that implements NJ using MPI. It approximately takes 3000 seconds to construct a NJ tree from 10 000 sequences while executing on 32 processors within its application in ClustalW. An MPI-based multiple sequence alignment tool, ClustalW-MPI,[9] has been proposed to align multiple protein sequences by leveraging the computing power of cluster system. ClustalW-MPI simultaneously aligns multiple sequences according to the order of branches of a guide tree constructed by NJ algorithm. The experimental result in ClustalW-MPI shows that a tree from 10 000 sequences is constructed in 25 418 seconds using 32 processors.

Nowadays, most of the current systems contain multicore processor. A multicore processor has 2 or more processors which are used to process multiple tasks in parallel. Open Multi-Processing (OpenMP)[10] is an application programming interface that allows multicore CPU to launch multiple threads and supports shared memory model. FastTree[11] is an implement of sequential programming model to build phylogenetic trees by sequentially aligning nucleotide/protein sequences. FastTree is 100 to 1000 times faster than PhyML 3.0 or RAxML 7. FastTreeMP is an implement of parallel model of FastTree using OpenMP, and its computational performance on 3 CPUs is 1.5 to 1.7 times faster than sequential version of FastTree.

Recently, Graphics Processing Units (GPUs), which possess thousands of small but efficient cores, have become an important role to accelerate the computational applications in many scientific domains and achieve a better performance than original applications. CUDA (Compute Unified Device Architecture) is a programming model proposed by NVIDIA in 2006, which can be written in C, C++, and Fortran. CUDA adheres to the single instruction, multiple threads execution model, it exploits GPU to run many threads independently and simultaneously, and it even allows divergent instruction streams. Till today, there have already been many powerful graphic cards supporting CUDA, thus CUDA programs can be run on those GPUs. Liu et al[12] proposed a parallel algorithm to construct NJ tree executing on a single GPU. This superior algorithm is able to achieve 26-fold speedup over the original NJ algorithm on CPU to construct a tree from 10 000 sequences. The GPU-UPGMA[5] is a highly computation-efficient method to generate a phylogenetic tree based on GPU architecture. It can achieve 95 times faster than the sequential UPGMA algorithm executing on CPU.

People start to combine multiple GPUs to handle the rapid growth of data as GPU has such a powerful parallel computation ability and as it becomes cheaper than CPU. In the beginning, one computer can only be equipped at most with 2 graphic cards by some techniques such as SLI (Scalable Link Interface) of NVIDIA and CrossFire of ATI to get an enhancement of 1.4× to 1.6× speedup of performance. It was nevertheless unable to cope with intensive computation request with big data. Message Passing Interface has then become a solution which may connect multiple computers with 1 or 2 graphic cards installed, enabling them to send and receive messages to/from each other through Ethernet. Hung et al[13] proposed an MPI version implementation of UPGMA on multiple computers equipped with GPUs. However, the bottleneck of the MPI version implementation is the slow response over Ethernet. Today, one computer is able to accommodate 2 to 8 GPUs, and these GPUs can communicate with each other through PCIe bus. Thus, the response time is much less than Ethernet. Furthermore, a new communication technique, called NVLink, was proposed by NVIDIA to enable ultrafast communication between CPU and GPU or among GPUs. NVIDIA claimed that this technology can accelerate data transfer rate to 5 to 12 times faster than PCIe bus. It means that GPUs can access data from each other at the speed of accessing local data from themselves. But in the programming level, it is usually complicated to write a program involving the communication among multiple GPUs. Fortunately, there is a library called NCCL (NVIDIA Collective Communications Library), which provides a communication model that is very similar to MPI. This tool provides 3 major and simple data collective communication primitives, namely, All-Reduce, All-Gather, and Broadcast, to ease the management of data communication among multi-GPUs. Not only it is familiar to users who are used to the MPI but NCCL also optimizes the data transfer efficiency in a few ways, which we will discuss later in this article and take on in the "Method" section.

Nowadays, the cost of sequencing is decreasing drastically due to the development of NGS, being capable of producing huge amount of genome sequences for environmental sampling. To construct a phylogenetic tree by UPGMA with such data set, the computational performance of existing UPGMA algorithms will certainly be unsatisfied. Therefore, we propose a novel parallel UPGMA algorithm based on multiple GPU devices to accelerate the tree construction process with large-scale sequence data.

In this article, we design a parallel algorithm of UPGMA, which is suitable to run on multiple GPUs. NVIDIA Collective Communications Library is used to connect these multiples GPU devices together and control the communication among them. The input data, a distance matrix, are split and spread to all available GPUs. Each GPU works with parts of data in parallel, partial results are then synchronized between iterations. It is obvious by experiments' results that the proposed algorithm on multiple GPUs can effectively enhance the performance of previously existing UPGMA algorithms.

Method

Unweighted Pair Group Method with Arithmetic Mean

The UPGMA algorithm constructs a phylogenetic tree from the pairwise similarity matrix, also known as distance matrix, which describes similarities between all possible pairs of given OTUs. The UPGMA algorithm contains 3 steps.

Step 1: Initializing the distance matrix. In this step, the distance matrix is initialized by filling in its entries with distance values between corresponding sequence pairs; a smaller value means a closer relationship, thus more similar. Figure 1 shows an example of the distance matrix. Distance between the *i*th sequence *i* and the *j*th sequence *j* will be written into the entry of *i*th row and *j*th

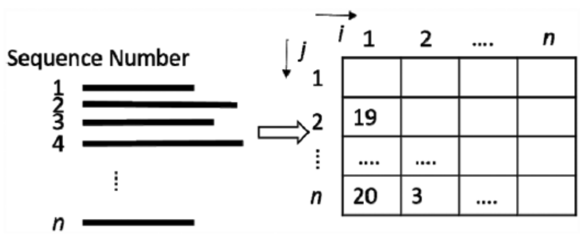

Figure 1. Pairwise distance matrix for each pair of sequences.

column of the matrix. Hence, each element in the matrix records, respectively, a distance value between 2 sequences.

Step 2: Grouping OTUs. In this step, the element currently recording the minimal distance value will be selected. If there is more than 1 element that records the minimum value, then a random one among them will be selected. The corresponding OTUs are then grouped together, and they form a new branch in the phylogenetic tree.

Step 3: Updating the distance matrix. The distance matrix must be updated after the pair of OTUs with minimum distance has been grouped into a new branch. Rows and columns of the grouped OTUs are removed from the distance matrix, but before that another row and column recording the new distances between the newly formed OTU group and the rest OTUs that are calculated based on the removed ones will be added.

Steps 2 and 3 are run consistently until all OTUs are merged into one group, and then a phylogenetic tree is completely constructed at the same time.

UPGMA based on GPU

It is noticed in literature[5] that most part of the computational time of UPGMA is dedicated to find the minimum and update the distance matrix. Therefore, these steps should be ported to GPU to leverage the computing power of GPU. The GPU-UPGMA is the GPU implementation of UPGMA on CUDA; it contains 3 stages to reduce the computational cost of manipulating the distance matrix. These stages are described as follows.

Stage 1. In this stage, the minimum distance value in the matrix will be found in 2 hierarchical steps. In the first one, the matrix is split into small slices, and then lots of thread blocks are dispatched simultaneously for these slices, one for each. Finally, a local minimum value in each slice is found by a parallel reduction process on GPU. All of these local minimum values are then transferred to the global memory of the GPU device. The second step, which is similar to the first one, is to find the global minimum value based on the set of local minimums by another reduction.

Stage 2. The distance matrix is updated in this stage. Another CUDA kernel function will be launched to calculate distances between the newly grouped OTU and the rest. Then, each thread is responsible for filling in an entry of the row and column to be added.

Stage 3. This stage is similar to the step 3 of UPGMA. A new branch is built according to the OTU pair found in the previous stages. As it is basically a sequential and simple task, this stage is executed on CPU.

Three stages run consistently in order until the tree is completely constructed.

UPGMA based on multiple GPUs through NCCL

"MPI is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation."[2] Its goals are high performance, scalability, and portability. Today, it remains the dominant model used in high-performance computing[3] to manage multiple computing nodes, and both point-to-point and collective communication are supported.

Although MPI has been proved to be very useful to manage the data communication and parallel tasks execution, it faces a critical bottleneck which may be due to external conditions but still a fundamental one: the speed of data communication through Ethernet.

Recently, it is possible to equip one server with a few GPUs to enhance computational performance. On that single server, those GPUs can communicate with each other through the PCIe bus, and it is much faster than Ethernet. However, it is also difficult to handle the communication among them. There are many issues that need to be considered such as bandwidth efficiency and time delay problem. For these reasons, we use NCCL not only to ease communications but also to address the issues inherently.

NVIDIA Collective Communications Library was proposed to enable and optimize multi-GPU collective communications. It is a library of optimized primitives implemented in C++ on CUDA libraries. The framework of NCCL is very similar to that of MPI. Therefore, many programming models already tested within MPI are ready to be applied in NCCL.

NVIDIA Collective Communications Library provides 3 major kinds of data collective communication primitives: All-Reduce, All-Gather, and Broadcast. The All-Reduce collective reduces data across multiple GPUs with a certain reduction operations. For example, suppose that there are many GPUs and each of them stores an array of numbers, furthermore, if the reduction operation is added, then All-Reduce enables every GPU to figure out an array of sums, which in fact is the sum over all elements of the same index in the arrays distributed on those GPUs. The "All" prefix before hyphen means that all GPUs get the same result in the end. The illustration of the elements of the same index painted by the same color is shown in Figure 2. Figure 3 shows how All-Gather collects data from each other GPUs and puts them together into each single GPU. The last collective method is Broadcast; it sends an array from one GPU to the rest; at last, every GPU will have a duplicate array stored in itself. The illustration of Broadcast is shown in Figure 4.

NVIDIA Collective Communications Library usually uses a function called GPUDirect for data transfer. This function is

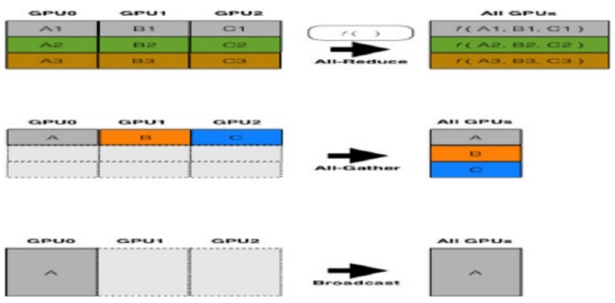

Figure 2. Illustration of NCCL All-Reduce collective function. GPUs indicate Graphics Processing Units; NCCL, NVIDIA Collective Communications Library.

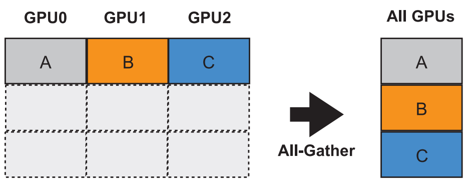

Figure 3. Illustration of NCCL All-Gather collective function.

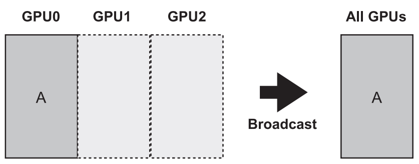

Figure 4. Illustration of NCCL Broadcast collective function. GPUs indicate Graphics Processing Units; NCCL, NVIDIA Collective Communications Library.

widely used in direct peer-to-peer connection between 2 GPUs. For some reasons, the device may not support GPUDirect; in that case, NCCL will reserve a main memory space as a buffer, through which GPUs can communicate.

Practically, data transfer between GPUs relies the PCIe topology, and a suitable choice of data transfer path that achieves high efficiency of communication. For example, let us say that GPU0 needs to broadcast 1GB data to all others; a PCIe topology is shown in Figure 5. Illustrations of 2 data transfer models on the PCIe topology are shown in Figure 6. In Figure 6A, the peer-to-peer data transfer from GPU0 to GPU1 is performed first, and then GPU0 transfer data to GPU3 and GPU1 transfer the data, obtained from GPU0, to GPU2 simultaneously. In this model, GPU0 and GPU1 will compete with each other in the bus between switch 1 and switch 2. Another transfer model is shown in Figure 6B. A data transfer from GPU0 to GPU2 is issued first, and then data transfer from GPU0 to GPU1 and from GPU2 to GPU3 is performed afterward. In this way, the PCIe bus traffic between 2 switches is used only for GPU0 to GPU2. Thus, the second model is better than the first one.

At the implementation level, a general strategy for a ring routing path is adopted and the algorithm just chooses the best outset to start. Furthermore, NCCL splits the whole data into many small chunks and transfer them one by one as streams through the ring routing path. By this way, it avoids that GPU devices wait a long time for data transferred from

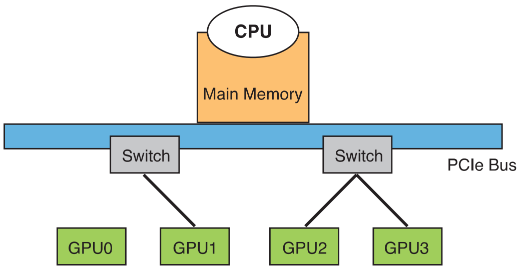

Figure 5. Illustration of PCIe bus topology. GPU indicates Graphics Processing Unit.

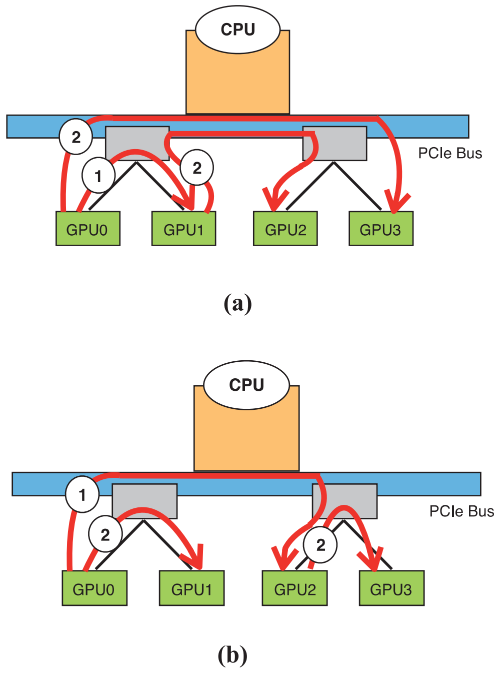

Figure 6. Two peer-to-peer transfer models between GPU devices across PCIe switches. GPU indicates Graphics Processing Unit. Figure 6(a) and 6(b) present the peer-to-peer transmission model between GPUs crossing PCIe switch twice and only once, respectively.

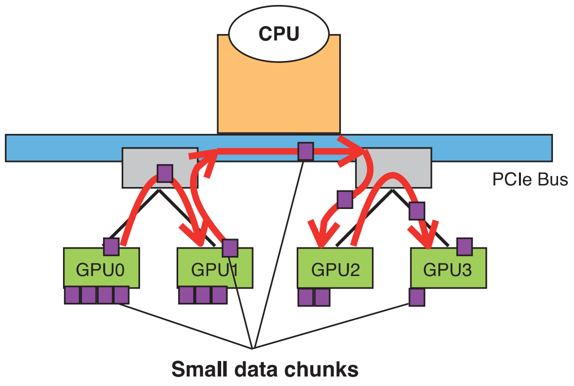

Figure 7. The ring-based transfer model between GPU devices in NCCL. GPU indicate Graphics Processing Unit; NCCL, NVIDIA Collective Communications Library.

other GPU devices before computation. An illustration is shown in Figure 7.

Eventually, to construct a phylogenetic tree from a larger amount of sequences using GPU-UPGMA, a single GPU

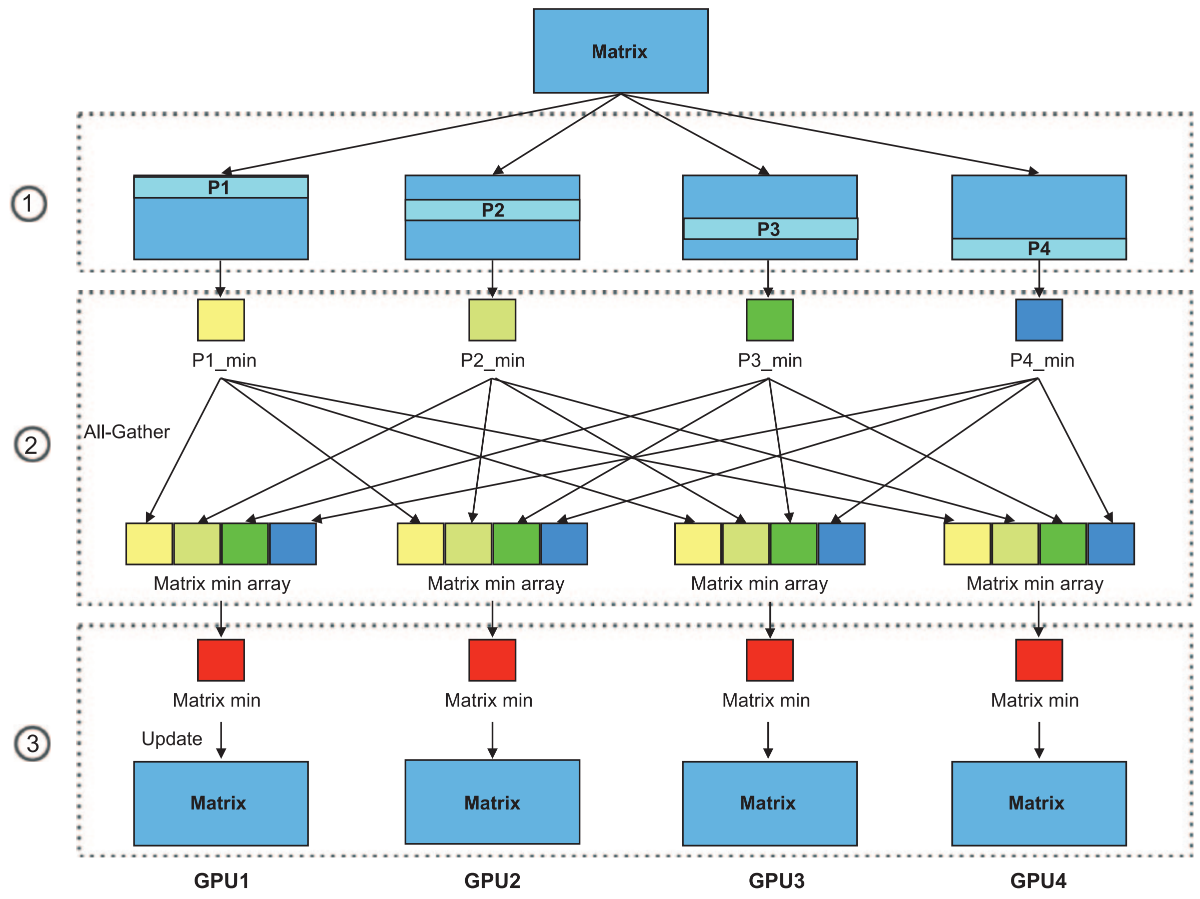

Figure 8. The flowchart of UPGMA (Unweighted Pair Group Method with Arithmetic Mean) on multiple GPU nodes. GPU indicates Graphics Processing Unit.

device is incapable to deal with them. Therefore, we proposed the parallel UPGMA algorithm to build a tree by multiple GPU devices within one single machine. All of the GPU devices have to collaborate together based on this idea. Therefore, NCCL is chosen to achieve this goal. The flowchart is shown in Figure 8. The steps of the proposed algorithm are listed as follows.

Step 1: Initialization. In the first step, all the available GPU devices have to been recognized by the NCCL. The input sequences are thus split into a number of sequence groups corresponding to the number of recognized GPU devices.

Step 2: Finding the minimum value. In this step, the local minimum value in a local distance matrix of each sequence group on a GPU will be found, and then All-Gather function is used to distribute all the local minimum values to each other.

Step 3: Updating the distance matrix. When All-Gather is finished, each GPU has a full set of local minimum values from itself and other GPUs, and then the global minimum value will be found and the distance matrix can be updated according to this value.

Steps 2 and 3 run consistently until the tree is completely constructed. The flowchart of these steps is shown in Figure 8.

Experiment

NVIDIA released an embedded GPU system, called NVIDIA DGX-1, which includes an Intel Dual 20-core E5-2698, 2.2 GHz CPU, 8 NVIDIA Tesla P100 GPU architecture with 3584 CUDA cores, and 512 GB DDR4 RAM. DGX-1 is capable of more than 170 TFLOPS (tera floating point operations per second), thanks to the NVIDIA's NVLink interconnect technique. DGX-1 also provides NCCL library for programmers. In this work, the proposed algorithm is implemented on a DGX-1 server with multiple P100 GPU devices. The underlying CUDA version is 6.5. The input data are the initial distance matrices which are generated randomly. The number of sequences varies from 1000 to 10 000. Consequently, dimension of the input distance matrix varies from 1000 × 1000 to 10 000 × 10 000.

Computations can be grouped into 3 parts: finding the local minimums, then the global minimum, and finally the matrix updates. Figure 9A shows the time evolution of finding the local minimum for different sizes of data with different numbers of GPUs. It is a computation-consuming step, but the figure shows that the computation work is scalable in parallel because it obtains more acceleration with more GPUs working together. Figure 9B shows the time of finding the global minimum. In this step, GPUs need to communicate with each other.

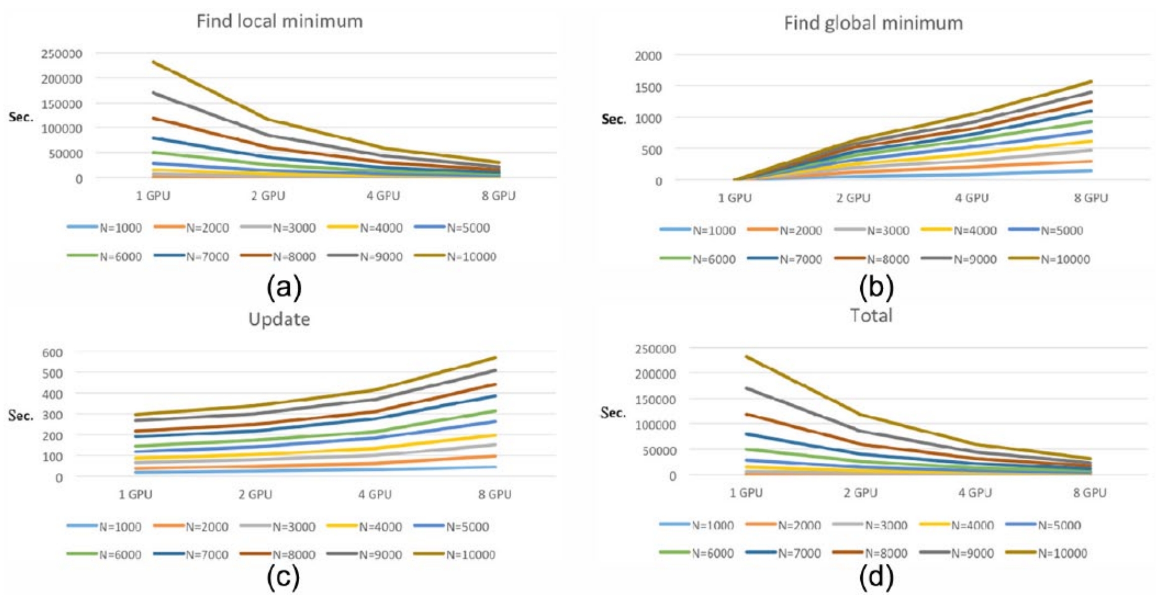

Figure 9. The comparison of time cost between the results produced by GPU-UPGMA (Unweighted Pair Group Method with Arithmetic Mean) using NCCL on one DGX-1 server and the proposed algorithm on 8 Tesla P100 GPU devices with 1000 to 10 000 sequences: (a) find local minimum, (b) find global minimum, (c) update, and (d) total execution. The *y*-axis represents the execution time, and Sec. denotes seconds. The *x*-axis represents the number of GPU devices. GPU indicate Graphics Processing Unit; NCCL, NVIDIA Collective Communications Library.

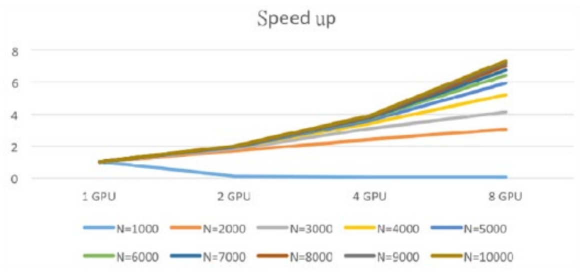

Figure 10. The comparison of speedup of total execution time between the results produced by GPU-UPGMA (Unweighted Pair Group Method with Arithmetic Mean) using NCCL on one DGX-1 server and the proposed algorithm on 8 Tesla P100 GPU devices with 1000 to 10 000 sequences. The *y*-axis represents the speedup, and the *x*-axis represents the number of GPU devices. GPU indicate Graphics Processing Unit; NCCL, NVIDIA Collective Communications Library.

The result shows that the computation time increases if more GPUs are involved in this step, and the probable reason lies in the increase in communication cost. Figure 9C shows the time needed for the matrix update, and it shows a similar behavior to that of Figure 9B because it involves communication too.

Although the time spent by steps 2 and 3 is increasing as shown in Figure 9B and C, they only contribute a very small part in total computation time. Figure 9D shows the total time evolution of a complete execution of our algorithm. Obviously, the proposed algorithm achieves much better performance when handling a huge amount of sequences. But when the amount of sequences is not big enough, the proposed algorithm may provide less speedup. The reason is that the communication time among GPUs will score a higher proportion in the total execution time. Figure 10 shows the evolution of speedup for input size varying from 1000 to 10 000 along the number of GPUs. In an extreme case, the proposed algorithm may even cut down performance when communication dominates computation, as shown in Figure 10, for data size of 1000. However, another interesting observation from Figure 10 is that the speedup is almost linear with the number of GPUs employed when treating a large amount of data, which suggests that our algorithm is highly scalable when the data size is big enough. Obviously, the proposed algorithm achieves the significant speedup over 2000 sequences. Therefore, it is suitable for building a UPGMA tree from huge amount of sequences.

Conclusions

In this work, we propose a novel UPGMA algorithm, based on GPU-UPGMA, with multiple GPUs using CUDA framework and NCCL to enhance the computational performance of constructing a phylogenetic tree from a huge amount of sequences. Experiments demonstrate that the proposed algorithm on multiple GPU devices is able to significantly improve the computational performance of GPU-UPGMA on a single GPU device when dealing with a large enough amount of data. And, in contrast to the MPI solution, the communication bottleneck is eliminated by the technique of NCCL and NVLink; hence, the proposed algorithm is also much faster than the MPI version of GPU-UPGMA.

Author Contributions

G-JH implemented the program, carried out the experiment, and wrote the manuscript with C-LH with support from CYT. C-LH provided the original idea and designed the algorithm with G-JH and C-YL. F-CW and Y-WC helped revise the manuscript.

REFERENCES

1. Caron DA, Countway PD, Savai P, et al. Defining DNA-based operational taxonomic units for microbial-eukaryote ecology. *Appl Environ Microbiol*. 2009;75:5797–5808.
2. Sokal RR, Michener CD. A statistical method for evaluating systematic relationships. *Univ Kansas Sci Bull*. 1958;38:1409–1437.
3. Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*. 2004;32:1792–1797.
4. Li JF. A fast neighbor joining method. *Genet Mol Res*. 2015;31:8733–8743.
5. Liu Y, Schmidt B, Maskell DL. Parallel reconstruction of neighbor-joining trees for large multiple sequence alignments using CUDA. Paper presented at: 2009 IEEE International Symposium on Parallel & Distributed Processing; May 23-29, 2009; Rome, Italy. http://ieeexplore.ieee.org/document/5160923/.
6. Metzker ML. Sequencing technologies—the next generation. *Nature Rev Genet*. 2010;11:31–46.
7. Barker B. Message passing interface (mpi). Paper presented at: Workshop: High Performance Computing on Stampede; January 14-15, 2015; Ithaca, NY. https://www.cac.cornell.edu/education/training/StampedeJan2015/Welcome.pdf.
8. Du Z, Lin F. pNJTree: a parallel program for reconstruction of neighbor-joining tree and its application in ClustalW. *Parallel Comput*. 2006;32:441–446.
9. Li KB. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*. 2003;19:1585–1586.
10. Dagum L, Menon R. OpenMP: an industry standard API for shared-memory programming. *IEEE Comput Sci Eng*. 1998;5:46–55.
11. Price MN, Dehal PS, Arkin AP. FastTree: computing large minimum evolution trees with profiles instead of a distance matrix. *Mol Biol Evol*. 2009;26:1641–1650.
12. Lin Y-S, Lin C-Y, Hung C-L, Chung Y-C, Lee K-Z. GPU-UPGMA: high-performance computing for UPGMA algorithm based on graphics processing units. *Concurr Comput Pract Exp*. 2015;27:3403–3414.
13. Hung C-L, Lin C-Y, Wu F-C, Chan Y-W. Efficient parallel UPGMA algorithm based on multiple GPUs. Paper presented at: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM); December 15-18, 2016; Shenzhen, China. http://ieeexplore.ieee.org/document/7822640/.