2012 IEEE 12th International Conference on Data Mining

Rapid and Robust Denoising of Pyrosequenced Amplicons for Metagenomics

Byunghan Lee†, Joonhong Park♮, and Sungroh Yoon†*
†*Electrical Engineering and Computer Science, Seoul National University, Seoul 151-744, Korea*
♮*Civil and Environmental Engineering, Yonsei University, Seoul 120-749, Korea*
**Correspondence: sryoon@snu.ac.kr*

***Abstract*—Metagenomic sequencing has become a crucial tool for obtaining a gene catalogue of operational taxonomic units (OTUs) in a microbial community. High-throughput pyrosequencing is a next-generation sequencing technique very popular in microbial community analysis due to its longer read length compared to alternative methods. Computational tools are inevitable to process raw data from pyrosequencers, and in particular, noise removal is a critical data-mining step to obtain robust sequence reads. However, the slow rate of existing denoisers has bottlenecked the whole pyrosequencing process, let alone hindering efforts to improve robustness. To address these, we propose a new approach that can accelerate the denoising process substantially. By using our approach, it now takes only about 2 hours to denoise 62,873 pyrosequenced amplicons from a mixture of 91 full-length 16S rRNA clones. It would otherwise take nearly 2.5 days if existing software tools were used. Furthermore, our approach can effectively reduce overestimating the number of OTUs, producing 6.7 times fewer species-level OTUs on average than a state-of-the-art alternative under the same condition. Leveraged by our approach, we hope that metagenomic sequencing will become an even more appealing tool for microbial community analysis.**

***Keywords*-pyrosequencing; amplicons; cluster analysis; GPU; biomedical informatics; metagenomics**

I. Introduction

Metagenomics refers to analyzing the functions and sequences of the collective microbial genomes contained in an environmental sample [1]. In our body, the microbes collectively make up to 100 trillion cells, tenfold the number of human cells, and mostly reside in the intestine [2]. Since there exists substantial diversity of the gut microbiome between individuals [3], understanding the diversity by metagenomics may lead to breakthroughs in personalized medicine and related industry [4]. Metagenomic studies typically produce a huge amount of sequence data (*e.g.*, [2] generated 576.7 gigabases in total) due to the microbial diversity. Without data mining, it would be nearly impossible to analyze such large-scale data.

A new generation of high-throughput, low-cost sequencing technologies, referred to as *next-generation sequencing* (NGS) [5], is playing a key role in establishing a genomic catalog of microbiomes. In particular, pyrosequencing [6] as implemented by Roche's 454 has been revolutionary in microbial community analysis ranging from continents to within an individual's body [7]. Compared to the automated Sanger sequencing, NGS methods produce much shorter reads, posing new computational challenges for genome assembly and annotation [8]. The role of computational methods for robust and efficient processing of NGS data has been crucial in metagenomic sequencing [9].

In many applications, a homologous DNA region such as well-conserved 16S rRNA genes from diverse microbiomes is first amplified by polymerase chain reaction (PCR) before sequenced. Pyrosequencing has allowed much larger read numbers from PCR amplicons than ever before [10]. Even so, pyrosequenced reads are never free from error, and a robust data-mining method is required to remove such error. Otherwise, the noise introduced during PCR and pyrosequencing may lead to overestimating the number of operational taxonomic units (OTUs) by orders of magnitude [11]. Although there exist denoising methods [7], [10], [12]–[14], their time demand is high, typically taking days to process only part of the amplicons collected. This slow rate clearly has bottlenecked the whole analysis pipeline, let alone hindering efforts to improve robustness.

To address this computational challenge, we propose a denoising scheme that progressively eliminates errors induced during pyrosequencing. The proposed method exploits data-level parallelism underlying the noise-removal process and accelerates the most time-consuming steps by running them on graphic processing units (GPUs). According to our experiment with 62,873 amplicons generated by pyrosequencing a mixture of 91 full-length 16S rRNA clones, our approach gave about 26 times speed-up over 8-core runs of AmpliconNoise [10], a widely used amplicon denoiser, reducing the analysis time from nearly 2.5 days to some 2 hours. As for accuracy, our approach significantly reduced overestimating the number of OTUs, producing 6.7 times fewer species-level OTUs on average than AmpliconNoise under the same experimental condition.

II. Background

A. Flowgrams Bear Sequence Information

Given a DNA fragment, pyrosequencing returns its nucleotide sequence as a series of light intensity values. This series is called the *flowgram* of the fragment. Let $F = (f_1, f_2, \ldots, f_m)$ denote a flowgram consisting of m intensity values, where $f_i \in \mathbb{R}_+$. A flowgram is arranged in such a way that every first, second, third and fourth intensity

1550-4786/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDM.2012.68

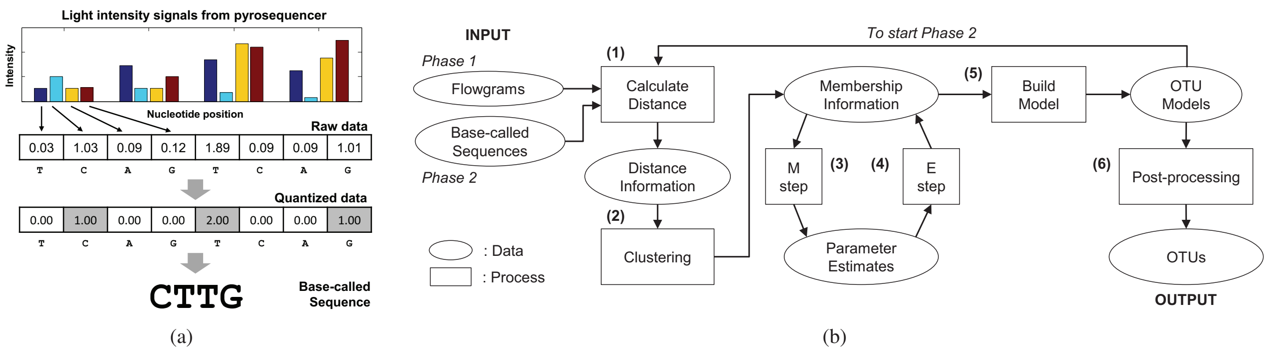


Figure 1. (a) Flowgrams and base-calling procedure. (b) Overview of removing noise in pyrosequenced amplicons.

values bear information on T, C, A, and G, respectively (Fig. 1(a)). The process of translating a flowgram into the corresponding nucleotide sequence is referred to as *base calling*. A simple base-calling method is to let $\lfloor f_i + 0.5 \rfloor$ indicate the number of nucleotides represented by f_i; 1.0 means a single nucleotide and a larger value a homopolymer (*e.g.*, 2.0 for AA and 3.0 for TTT). The *read* of a fragment means its sequence resulting from base calling.

Given two flowgrams $F = (f_1, f_2, \ldots, f_m)$ and $G = (g_1, g_2, \ldots, g_m)$, their distance is defined by

$$d(F,G) = \frac{1}{m} \sum_{i=1}^{m} \delta(f_i, g_i) \qquad (1)$$

where δ is an empirically determined bivariate function representing the distance between two intensity values [10]. The probability that F and G originate from the same sequence decreases exponentially as their distance increases. We denote this probability by $P(F = G)$, which is defined over $[0, +\infty]$ as follows [14]:

$$P(F = G) = \frac{1}{\sigma} \exp\left[-\frac{d(F,G)}{\sigma}\right] \qquad (2)$$

where σ is a user-specified parameter.

B. Noise Sources in Pyrosequenced Amplicons

Three major sources of noise exist [10]: base-calling error during pyrosequencing, PCR single nucleotide substitutions and PCR chimeric sequences. This paper focuses on removing the first two types. They are more general than the last, which requires knowledge specific to the given data set and is difficult to generalize. A confusion may arise when calling bases in a homopolymer, because observed light intensities do not perfectly match the homopolymer lengths. The *base-calling error* refers to determining the length of a homopolymer incorrectly. As stated in Section I, a DNA region of interest is normally amplified by PCR before sequenced. During PCR, it is possible that a nucleotide is erroneously replaced by another type (*e.g.*, base G becomes C). We call this type of noise *PCR single nucleotide substitutions*.

Table I
COMPARISON OF RELATED NOISE-REMOVAL METHODS

Method	Sequence of steps[†]	Target noise[‡]	Ref.
CRiSPy-CUDA	$1, 2, 5, 6$	P	[12]
CUDA-EC	$1, 2, 5, 6$	P	[13]
PyroNoise	$pc^{§}, 1, 2, (3, 4)^{*}, 5, 6$	B	[14]
AmpliconNoise	$pc^{§}, (1, 2, (3, 4)^{*}, 5)^{2}, 6$	B, P	[10]
Reeder and Knight	Greedy algorithm	P	[7]

[†] As labeled in Fig. 1(b); §: preclustering; $(\cdot)^2$: repeat twice; $(\cdot)^*$: repeat until convergence.
[‡] B: base-calling error; P: PCR single nucleotide substitutions.

C. Related Work

Table I lists existing denoisers, their procedures and target noise types. Most of these methods have many internal procedures in common, as annotated in the table. Existing tools can handle either the base-calling error or the PCR single nucleotide substitutions, except for AmpliconNoise. It can handle both error types by repeating the main error-correction loop as our approach. However, AmpliconNoise requires a preclustering step, which divides input data into nonoverlapping partitions. Each partition is processed separately, thus making it impossible to find global patterns appearing in multiple partitions. The proposed method can handle both types of pyrosequencing errors, and does not require the preclustering step. Leveraged by the use of GPU-based acceleration techniques, our approach also outperforms AmpliconNoise by large margin in terms of running time. Furthermore, the proposed method can estimate the number of OTUs more accurately under the same condition.

III. Methods

A. Overview

Fig. 1(b) depicts the overview of removing base-calling error and PCR single nucleotide substitutions. The input consists of the flowgrams of pyrosequenced amplicons and their base-called sequences, and the output is a set of OTUs (in other words, organisms) represented by the pyrosequenced amplicons. The whole denoising pipeline

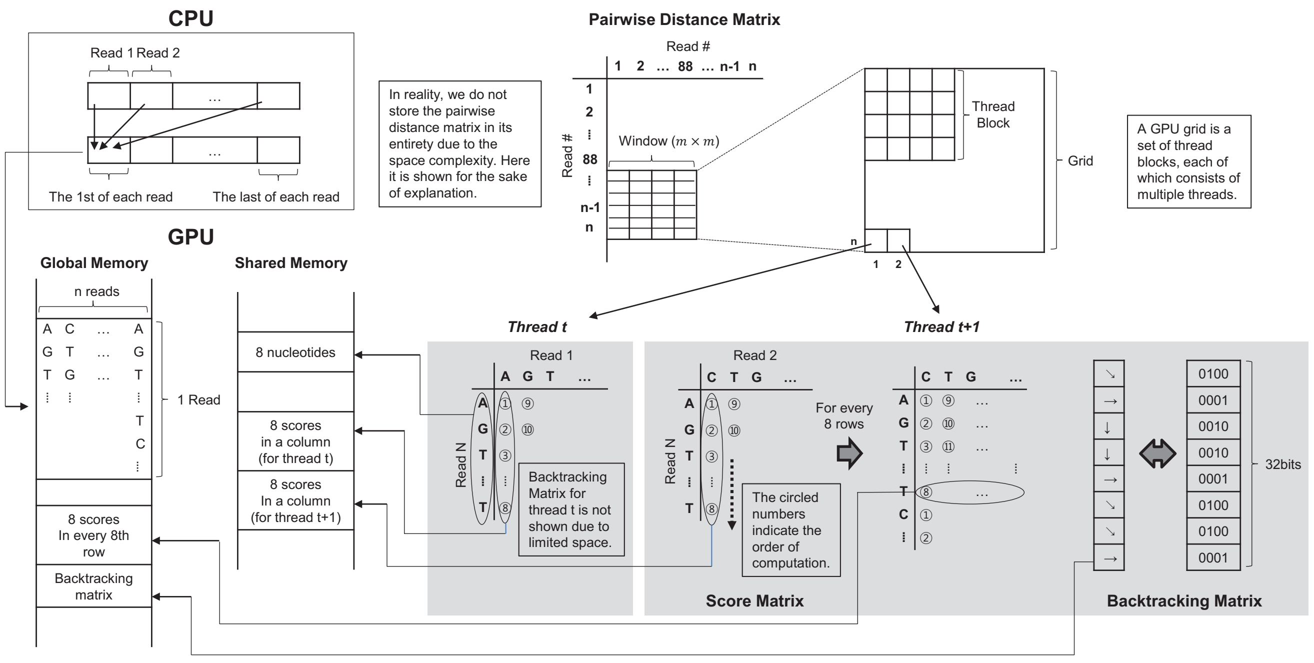

CPU
Read 1 Read 2
The 1st of each read
The last of each read
GPU
Global Memory
n reads
1 Read
8 scores In every 8th row
Backtracking matrix
Shared Memory
8 nucleotides
8 scores in a column (for thread t)
8 scores In a column (for thread t+1)
In reality, we do not store the pairwise distance matrix in its entirety due to the space complexity. Here it is shown for the sake of explanation.
Pairwise Distance Matrix
Read #
Window ($m \times m$)
Thread Block
Grid
A GPU grid is a set of thread blocks, each of which consists of multiple threads.
Thread t
Thread t+1
Read 1
Read 2
Read N
Backtracking Matrix for thread t is not shown due to limited space.
The circled numbers indicate the order of computation.
For every 8 rows
Score Matrix
Backtracking Matrix
32bits

Figure 2. Overview of the proposed GPU-based scheme for parallelizing the denoise process for high-throughput pyrosequenced amplicons.

consists of six steps as labeled in the diagram. As suggested in the literature [10], [14], we handle the two types of pyrosequencing error separately in two phases: In phase 1, we remove the base-calling error based on flowgrams in steps 1–5. In phase 2, we revisit these steps to eliminate the PCR substitution error based on base-called sequences.

The first step of phase 1 is to calculate the pairwise distance between flowgrams. Based on the resulting distance information, we perform clustering of flowgrams in step 2. This is to estimate the number of OTUs in the input data under the assumption that each cluster corresponds to an OTU. We can then regard the flowgrams in a cluster as the instances of the corresponding OTU. In steps 3 and 4, we find the best representative flowgram for each cluster by the expectation-maximization algorithm [15]. It allows us to estimate the membership of every flowgram in such a way that the likelihood of observing all the flowgrams in the input data is maximized. In step 5, we declare each of these representative flowgrams as the model for an OTU, and feed their base-called sequences to step 1, starting the second phase. In step 1 of the second phase, we calculate pairwise distance between based-called sequences. This step tends to be time-consuming, and we accelerate it by GPU-based parallelization. The rest of the second phase proceeds similarly to the first. In step 6, we finalize the denoising process by assigning each denoised sequence to one of the OTU models according to its similarity to a model.

For n reads of length L, the worst-case time complexity of the sequential algorithm is $O(L^2n^2)$, which can be significantly reduced by parallelization.

B. GPU-Based Computation of Distance between Sequences

We define the distance between two sequences based on their alignment returned by the Needleman-Wunsch algorithm [16]. It has quadratic worst-case time complexity, and the pairwise distance computation usually takes significant time for large data (Fig. 4(a)).

Assume that there are n reads in total, each of which has length L. Each of the $\binom{n}{2}$ computations is independent, and GPU is a good fit for handling this type of parallelism. GPU has a large number of cores with different types of memory arranged in hierarchy. To unleash the full potential of GPU, we should (1) maximize the utilization of cores and (2) exploit the memory hierarchy wisely also minimizing host-device memory transfer needs [17].

The basic idea is to let GPU threads simultaneously calculate pairwise distance in parallel, but there are many considerations to take for fully exploiting GPU. Two major issues are as follows. First, GPU has fewer threads than $\binom{n}{2}$ unless n is very small. We need a divide-and-conquer approach. Second, the Needleman-Wunsch algorithm computes distance between two reads by dynamic programming, which progressively completes two $L \times L$ matrices, one for scoring and the other for backtracking. A naïve implementation would then require $O(2L^2n^2)$ space, which typical GPU memory cannot store even for modest values of n and L. We need a scheme for efficient utilization of GPU memory.

Fig. 2 shows the overview of our GPU-based parallelization. The distance matrix splits into multiple submatrices

($m \times m$ 'windows'), and one GPU grid (a set of thread blocks) computes all the entries within one window simultaneously, by assigning one thread to computing one distance. More details are as follows:

- A GPU thread progressively fills up the two matrices by considering 8 rows as a unit of computation. Given an $L \times L$ matrix, the first $8 \times L$ submatrix is computed followed by the second, and so on.
- Each $8 \times L$ submatrix is filled up columnwise. That is, the first column of the submatrix is computed, followed by the second column, and so forth.
- The scores in the above 8×1 submatrix are stored in shared memory and are used for the next column.
- When the thread has completed each $8 \times L$ submatrix, its last row is stored in global memory so that the thread can utilize it for the next $8 \times L$ submatrix.
- The sequence of 8 nucleotides matching the rows of each $8 \times L$ submatrix is stored in shared memory.
- Each backtracking direction is encoded into 4 bits ('Backtracking Matrix' in Fig. 2), and 8 direction values are collectively stored in a 32-bit variable in global memory.
- For coalesced global memory access, the k-th nucleotides of all reads are stored followed by the $(k+1)$-th nucleotides, and so on (see the inlet labeled *CPU* in Fig. 2).

The GPU memory utilization explained above was inspired by prior work on the GPU-based parallelization of dynamic programming [18]–[20]. We improved existing approaches by reflecting domain knowledge on pyrosequenced amplicons and by devising memory coalescing techniques.

C. Constructing OTU Models

The fundamental idea is that, for each OTU, we find from flowgrams the one that best represents the OTU. Assume that there are n flowgrams $F_1, F_2, \dots, F_n$ in the input data and let $\mathbf{F}$ denote them collectively. In a typical metagenomics study, we do not generally know the number of OTUs in advance. Let k be the number of OTUs existing in a sample and let $\mathbf{O}$ collectively denote the k OTUs $O_1, O_2, \dots, O_k$. We estimate k by clustering flowgrams and then to select k flowgrams, each of which best represents a cluster. (Note that a cluster corresponds to an OTU.)

In the selection process, we assume that each flowgram can belong to multiple clusters. Let $z_{ij} \in [0,1]$ indicate the membership of flowgram i in cluster j. As the selection criterion, we use the likelihood of the n flowgrams generated by the k OTUs, according to [14]:

$$L(\mathbf{F}, \mathbf{Z}|W, \mathbf{O}) = \prod_{i=1}^{n} \prod_{j=1}^{k} [w_j P(F_i = O_j)]^{z_{ij}} \tag{3}$$

where w_j corresponds to the weight of cluster j and $\mathbf{Z}$ and W collectively represent z_{ij} and w_j, respectively.

1) Determining the Number of OTUs: This corresponds to the problem of figuring out the correct number of clusters existing in data, an important issue in unsupervised clustering [21]. One approach is to run k-means clustering for different values of k and select k that maximizes the median silhouette coefficient [22] of a clustering result. Another technique is to run hierarchical clustering and cut the resulting dendrogram at a certain threshold to produce k flat clusters. In either case, we calculate the pairwise distance between flowgrams using Eq. 1.

2) Selecting Model Flowgrams: The expectation-maximization (EM) algorithm [15] can find the best W and $\mathbf{O}$ that maximizes Eq. 3. To this end, the following two steps are repeated until convergence:

E-step: The expected membership is computed based on the outcome of the M-step.

$$\hat{z}_{ij}^{(t)} = \frac{\hat{w}_j^{(t)} P(F_i = \hat{O}_j^{(t)})}{\sum_{j=1}^{k} \hat{w}_j^{(t)} P(F_i = \hat{O}_j^{(t)})} \tag{4}$$

where t denotes the iteration count for EM. The initial estimate $\hat{z}_{ij}^{(0)}$ is set based on the clustering result: if flowgram i belongs to cluster j, $\hat{z}_{ij}^{(0)} = 1$; otherwise $\hat{z}_{ij}^{(0)} = 0$.

M-step: The estimates are updated using the membership estimate from the E-step.

$$\hat{w}_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} \hat{z}_{ij}^{(t)} \tag{5}$$

$$\hat{O}_j^{(t+1)} = \arg\min_{F \in \mathbf{F}} \left[\sum_{i=1}^{n} \hat{z}_{ij}^{(t)} d(F_i, F) \right] \tag{6}$$

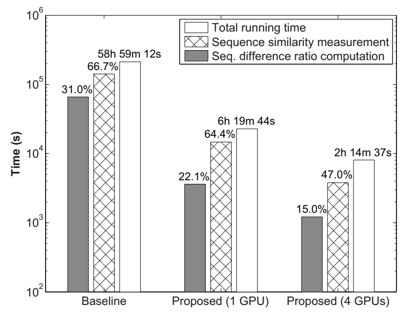
IV. Results and Discussion

A. Experiment Setup

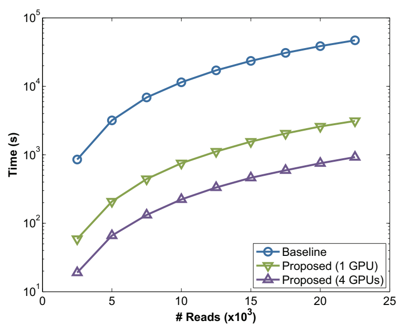
We tested the proposed approach with the data from pyrosequencing a mixture of 91 full length 16S rRNA clones from an Arctic soil sample [10]. This data set has 62,873 reads, each of which is 720 bases long. We used CUDA Toolkit 4.0.17 under an Ubuntu Server 10.04.3 LTS environment running on a server with two 64-bit quad-core Intel Xeon X5650 CPUs (2.67 GHz), 64-GB main memory and four NVIDIA GeForce GTX 580 cards (3-GB GDDR5 each). The baseline for comparison was set to running AmpliconNoise [10] on the same Linux box as above with 8-node MPI parallelization. For tests in cloud computing, we used Amazon Web Services (AWS) on which we created a 40-node (Xeon E5645 2.40 GHz, 7.5 GB) cluster.

B. Accuracy Comparison

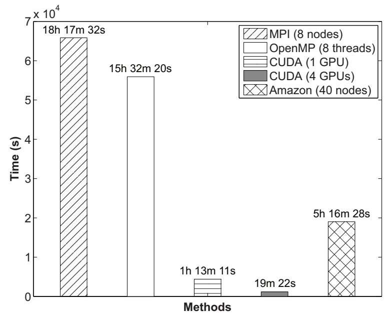
A key objective in a metagenomics study is to determine the number of OTUs in a sample accurately. Even for the same sample, the estimated number of OTUs varies depending on the strictness of defining an OTU in terms of the sequence difference ratio. Given two sequences, this ratio indicates the number of positions having different



(a)



(b)



(c)

Figure 4. Running time comparison (window size $m = 88$ for GPU runs). (a) Total running time with breakdowns. (b) Effect of read length variations. (c) Different parallelization options.

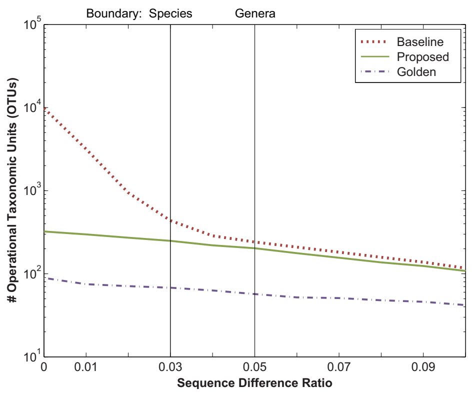


Figure 3. Comparing accuracy in terms of estimated number of OTUs.

nucleotides. 3% and 5% ratios are widely used as the boundaries for defining species and genera, respectively [14]. We used this sequence difference ratio as the threshold to cut the dendrogram in the process of determining the number of OTUs in Section III-C. Note that using different ratios (*i.e.*, cutting the dendrogram at different heights) produce different k values (*i.e.*different numbers of OTUs).

Fig. 3 compares the number of OTUs returned by the proposed and baseline methods under the same setup (data, time and machines) over various sequence difference ratios. Over the entire range of difference ratio, the proposed method outperformed the baseline, producing more accurate estimates to the golden reference. In particular, within the species boundary, the proposed method estimated 6.7 times fewer OTUs on average than the alternative. As the ratio approaches zero, the performance advantage of our method becomes more salient.

C. Running Time Comparison

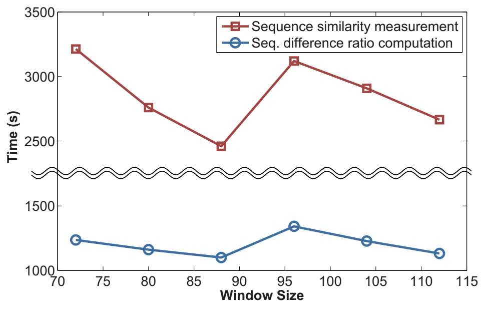
Fig. 4(a) depicts the running time of the proposed and baseline methods, along with the percentage of the two most time-consuming steps. They are both related to measuring pairwise distance between sequences by the Needleman-Wunsch algorithm, and this observation justifies the parallelization efforts explained in Section III-B. The 4-GPU and 1-GPU versions ran 26.29 and 9.34 times faster than the baseline, respectively. The speed-up by using additional cards was not linear due to non-parallelized code, but the speed-up of the two time-consuming steps was close to linear, as will be shown in Fig. 5(b). Fig. 4(b) compares the time taken to denoise reads of various lengths. The proposed method run on 4-GPU configuration is the fastest, and the proposed method are capable of denoising longer sequences given the same time.

There exist various parallelization options, and we wanted to see which is the best option for the denoising problem. Fig. 4(c) presents the running time of different parallelization approaches taken to complete the step labeled 'Seq. difference ratio computation' in Fig. 1(b). Evidently, the proposed GPU-based approach outperformed the alternatives by large margin. The execution of MPI version on Amazon cloud took more than expected, probably due to the fact that there is limited latency consideration in AWS. To get the same running time as the 1-GPU run of our approach, we would need 105 nodes on AWS.

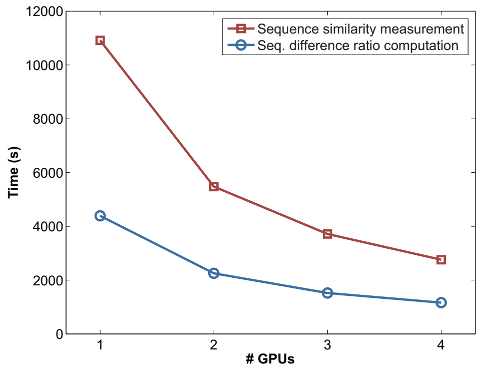
D. Additional Results

Fig. 5(a) shows how the computation window size (Section III-B) affects the running time of the two most time-consuming steps mentioned in Fig. 4(a). As we increase the window size, the running time decreases until a certain point (88 in this case). This is because more GPU threads are used in parallel. However, if we increase the window size further, then the running time starts to increase, since idling threads appear. If we further increase the window size, then the runtime decreases as there are enough threads to form a new block that can start working.

Fig. 5(b) shows the effect of using different amounts of GPU cards on the running time of the two most time-consuming steps. The speed-up was nearly proportional to the number of GPU cards used. Hence, using more GPU



(a) Effect of window size m (4 GPUs used)



(b) Effect of the number of GPUs

Figure 5. Additional results.

cards will increase the maximum speed-up we report in this paper (26.29 times over the baseline) even more.

V. Conclusion

According to our experimental results, the proposed technique was effective both in speed and accuracy. For speed, our approach reduced the time demand for denoising 62,873 amplicons from almost 2.5 days to only about 2 hours. We also found out that GPUs could be better for parallelizing denoisers than alternatives. For accuracy, the proposed method reduced overestimating the number of OTUs, producing 6.7 times fewer species-level OTUs on average than a state-of-the-art existing tool under the same experimental condition. Leveraged by our approach, we hope that metagenomic sequencing will become an even more appealing tool for microbial community analysis.

Acknowledgements

This work was supported by the National Research Foundation (NRF) of Korea funded by the Ministry of Education, Science and Technology (MEST) (No. 2011-0009963) and by the World Class University Program of NRF funded by MEST (R33-10076).

References

[1] C. Riesenfeld, P. Schloss, and J. Handelsman, “Metagenomics: genomic analysis of microbial communities,” *Annu. Rev. Genet.*, vol. 38, pp. 525–552, 2004.

[2] J. Qin *et al.*, “A human gut microbial gene catalogue established by metagenomic sequencing.” *Nature*, vol. 464, no. 7285, pp. 59–65, Mar. 2010.

[3] P. Eckburg *et al.*, “Diversity of the human intestinal microbial flora,” *Science*, vol. 308, no. 5728, pp. 1635–1638, 2005.

[4] P. Lorenz and J. Eck, “Metagenomics and industrial applications,” *Nature Reviews Microbiology*, vol. 3, no. 6, pp. 510–516, 2005.

[5] M. Metzker, “Sequencing technologies—the next generation,” *Nature Reviews Genetics*, vol. 11, pp. 31–46, 2009.

[6] M. Margulies *et al.*, “Genome sequencing in microfabricated high-density picolitre reactors.” *Nature*, vol. 437, no. 7057, pp. 376–80, Sep. 2005.

[7] J. Reeder and R. Knight, “Rapidly denoising pyrosequencing amplicon reads by exploiting rank-abundance distributions.” *Nature methods*, vol. 7, no. 9, pp. 668–9, Sep. 2010.

[8] M. Pop and S. L. Salzberg, “Bioinformatics challenges of new sequencing technology,” *Trends in Genetics*, vol. 24, no. 3, pp. 142–149, 2008.

[9] J. G. Caporaso *et al.*, “QIIME allows analysis of high-throughput community sequencing data Intensity normalization improves color calling in SOLiD sequencing,” *Nature methods*, vol. 7, no. 5, pp. 335–336, 2010.

[10] C. Quince *et al.*, “Removing noise from pyrosequenced amplicons.” *BMC bioinformatics*, vol. 12, p. 38, Jan. 2011.

[11] V. Kunin *et al.*, “Wrinkles in the rare biosphere: pyrosequencing errors can lead to artificial inflation of diversity estimates.” *Environmental microbiology*, vol. 12, no. 1, pp. 118–23, Jan. 2010.

[12] Z. Zheng *et al.*, “CRiSPy-CUDA: Computing species richness in 16S rRNA pyrosequencing datasets with CUDA,” *Pattern Recognition in Bioinformatics*, pp. 37–49, 2011.

[13] H. Shi *et al.*, “A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware,” *Journal of Computational Biology*, vol. 17, no. 4, pp. 603–615, 2010.

[14] C. Quince *et al.*, “Accurate determination of microbial diversity from 454 pyrosequencing data.” *Nature methods*, vol. 6, no. 9, pp. 639–41, Sep. 2009.

[15] A. Dempster, N. Laird, and D. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society*, pp. 1–38, 1977.

[16] S. Needleman and C. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[17] R. Farber, *CUDA application design and development*. Morgan Kaufmann Pub, 2011.

[18] M. C. Schatz *et al.*, “High-throughput sequence alignment using Graphics Processing Units,” *BMC Bioinformatics*, vol. 8, p. 474, 2007.

[19] S. A. Manavski and G. Valle, “CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment.” *BMC bioinformatics*, vol. 9 Suppl 2, p. S10, Jan. 2008.

[20] J. Blazewicz *et al.*, “Protein alignment algorithms with an efficient backtracking routine on multiple GPUs,” *BMC bioinformatics*, vol. 12, p. 181, 2011.

[21] C. Fraley and A. Raftery, “How many clusters? Which clustering method? Answers via model-based cluster analysis,” *The computer journal*, vol. 41, no. 8, pp. 578–588, 1998.

[22] P. N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Boston, MA: Pearson Addison Wesley, 2006.