CATEGORY: **BIOINFORMATICS & GENOMICS - BG06**

POSTER **P4190** | CONTACT NAME **Shuji Suzuki: suzuki@bi.cs.titech.ac.jp**

GPU TECHNOLOGY CONFERENCE

Accelerating identification of frequent k-mers in DNA sequences with GPU

Shuji Suzuki[1], Masanori Kakuta[1], Takashi Ishida[1] and Yutaka Akiyama[1]

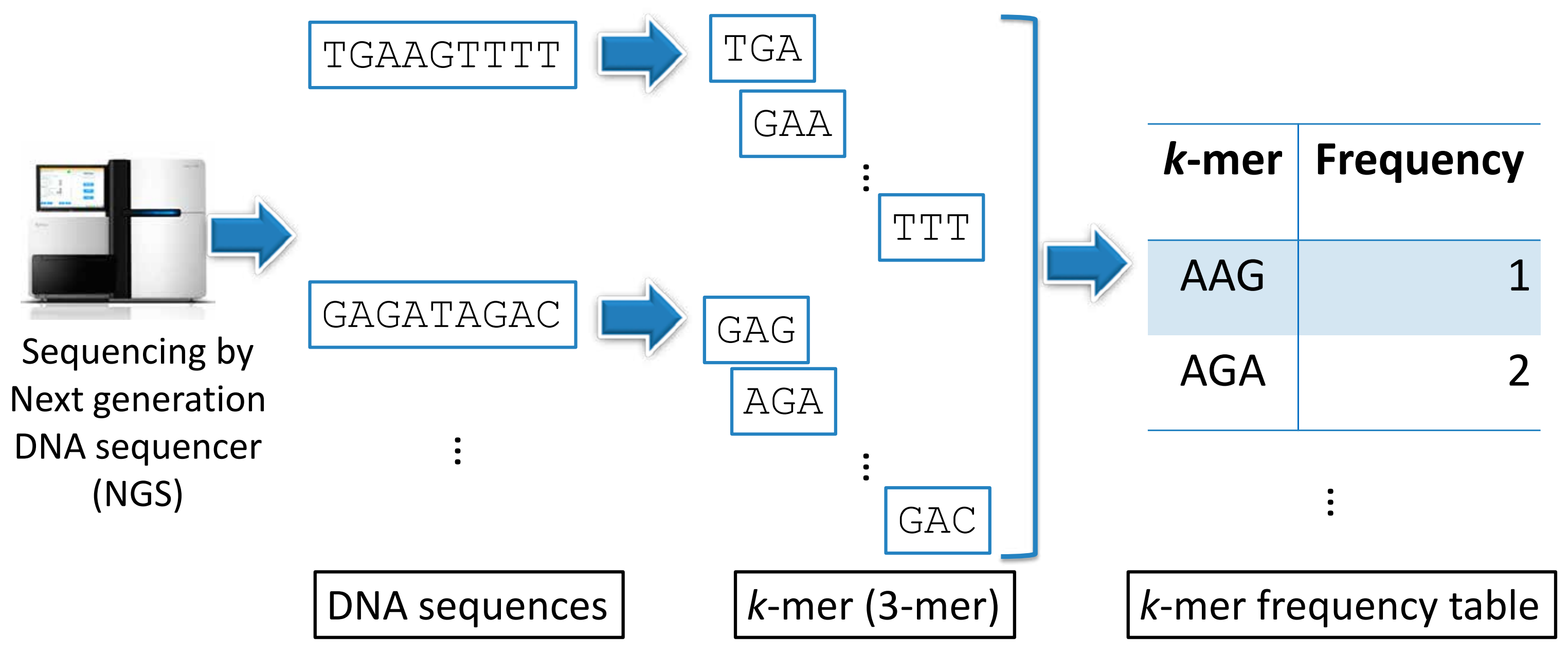
[1] Graduate School of Information Science and Engineering, Tokyo Institute of Technology, Japan

Abstract

Identifying the frequencies of k-mers (substring of length k) in strings is an important sub-problem in many bioinformatics applications. Especially, counting the number of k-mers takes the largest computing time in genome assembly, which is one of the most important analyses and often a first step after genome sequencing experiments. Moreover, DNA sequencing techniques have been rapidly improved. The throughputs of these sequencers are approximately 1,000 times higher than that of previous sequencers and have been increasing steadily. Therefore, a faster algorithm for identifying frequent k-mers is highly needed. Recently, an efficient algorithm, which uses sorting-based algorithm, was proposed. However, the algorithm, Turtle, is insufficient in the parallel performance and requires too large memory for GPU implementation. In this research, we propose a new k-mer counting algorithm suitable for GPUs calculations based on sorting algorithm. We implemented this algorithm on a CPU-GPU heterogeneous environment. The algorithm is similar to Turtle but we improved it by optimizing parallel k-mer counting algorithm and accelerating it by GPUs.
We implemented our algorithm onto GPU by using CUDA 5.0 and evaluated it by using real G. Gallus genome sequence data. As results, our algorithm with 12 CPU cores and 2 GPUs was 2.4 times faster than Turtle software on 12 CPU cores.

Introduction

"*k*-mer counting" is a problem to identify the frequencies of *k*-mers (substring of length *k*) in strings.



k-mer counting is an important sub-problem in many bioinformatics applications

- Genome assembly
- Error correction of sequencing reads
- Fast multiple sequence alignment
- Repeat detection

Current major *k*-mer counting tools

- Turtle [1]: sorting based algorithm
- Jellyfish [2]: algorithm with lock-free hash table optimized for counting k-mers

Problems :

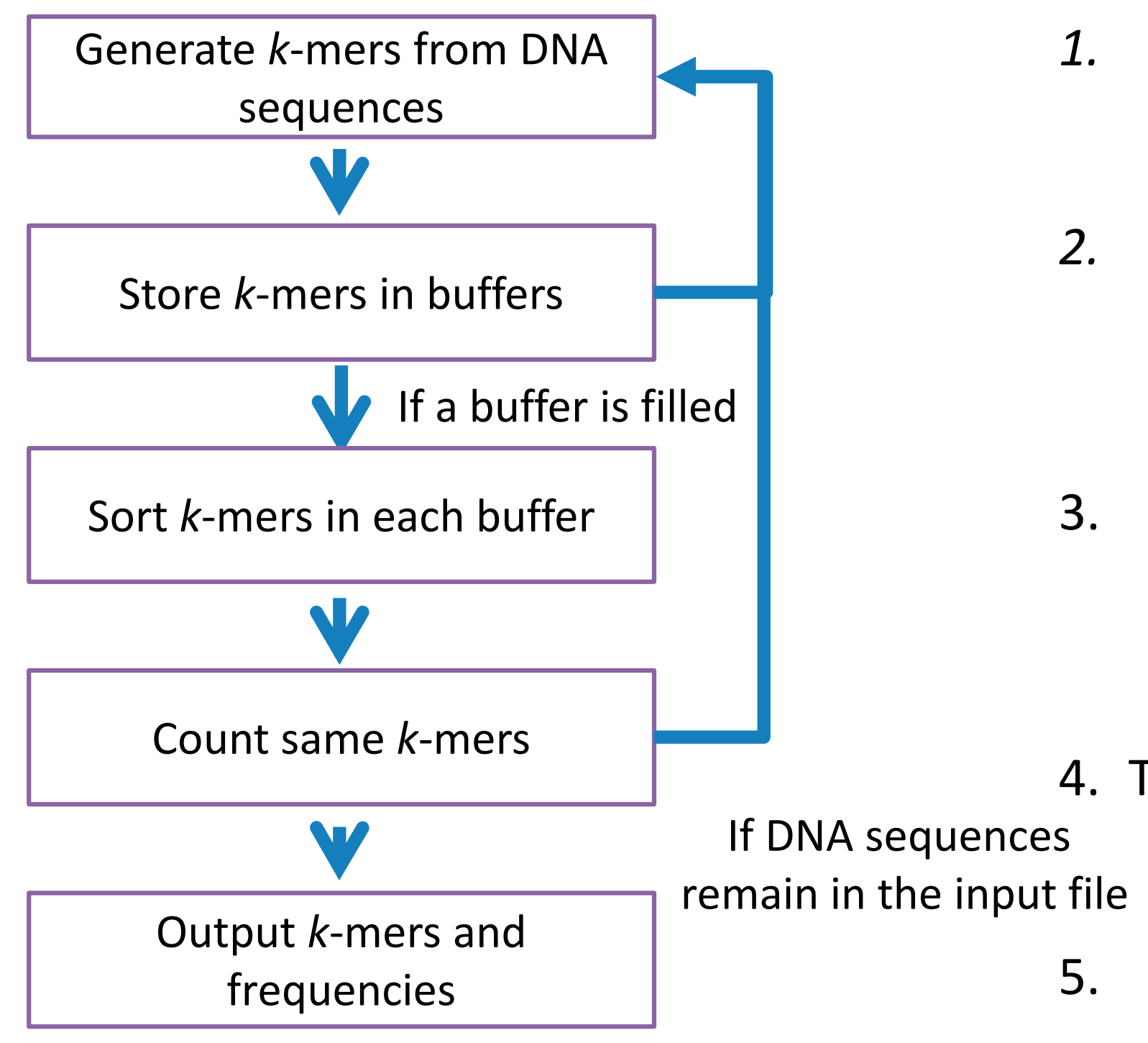
k-mer counting is useful but it requires a huge amount of computational time

1. Counting *k*-mers takes the most computing time of genome assembly.
2. A huge amount of genomic data
   The size of NGS (Hiseq2000) result is about 600 GB / run and about 10,000 times larger than that of the previous generation sequencers.

In this study, we propose a new k-mer counting algorithm suitable for GPU calculations based on sorting algorithm. We implemented this algorithm on a CPU-GPU heterogeneous environment.

Sorting Based *k*-mer Counting Algorithm [1]

The sorting based algorithm is currently the fastest algorithm for counting *k*-mers.


Generate k-mers from DNA sequences
Store k-mers in buffers
If a buffer is filled
Sort k-mers in each buffer
Count same k-mers
If DNA sequences remain in the input file
Output k-mers and frequencies

Overview of sorting based algorithm

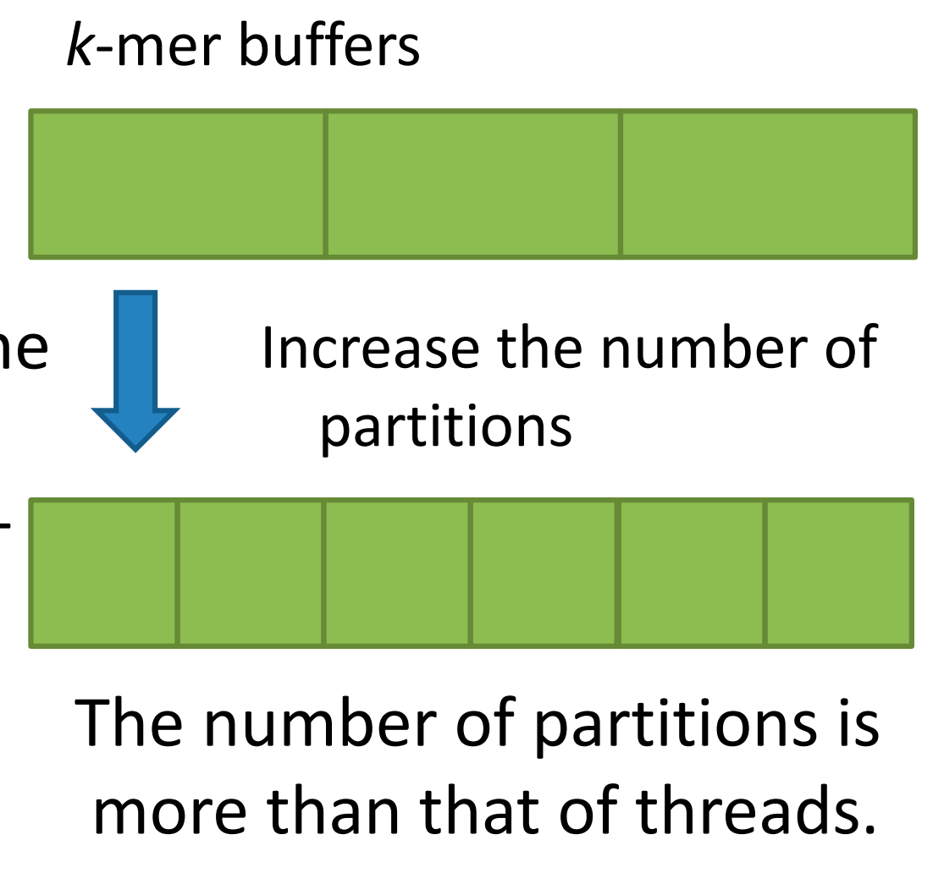
1. *k*-mers are generated from DNA sequences.
2. *k*-mers are stored in buffers.
   - The buffer to store the *k*-mer is decided from the hash value of the *k*-mer.
3. If a buffer is filled, *k*-mers are sorted.
   - Same *k*-mers are next to each other after sorting. Therefore, same *k*-mers are found by sequential search.
4. The same *k*-mers are counted
5. If DNA sequences did not remain in the input file,
   - *k*-mers and frequencies are outputted.

Proposed Method: Novel Sorting Based *k*-mer Counting Algorithm

We propose a new k-mer counting algorithm suitable for GPU calculations based on sorting algorithm. We implemented this algorithm on a CPU-GPU heterogeneous environment. Our contributions in this study are following.

Optimizing parallel *k*-mer counting

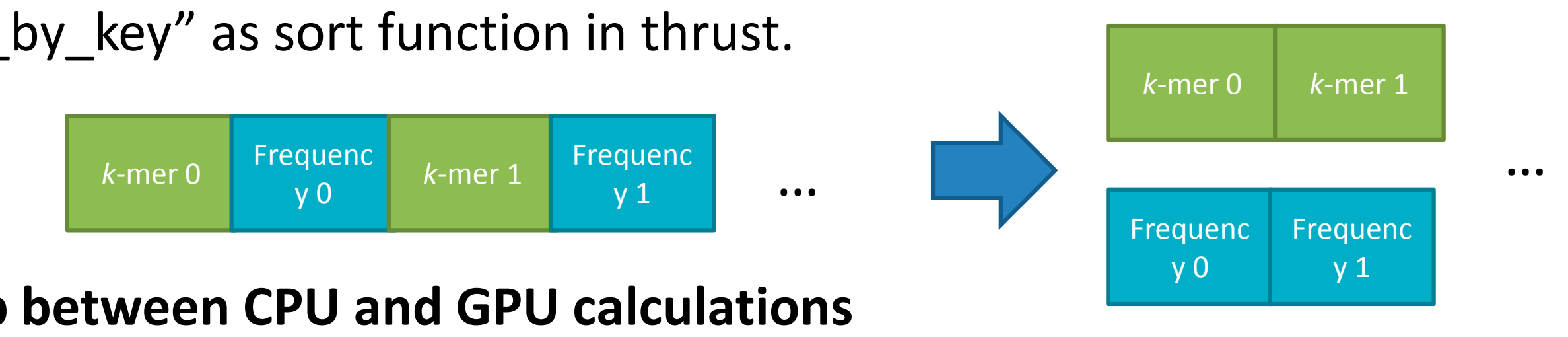
1. Reducing lock contentions
In parallel *k*-mer counting, threads often try to lock the same *k*-mer buffer in shared memory to store and sort *k*-mers.
⇒ Our algorithm increased the number of partitions of *k*-mers buffers to reduce lock contentions.


k-mer buffers
Increase the number of partitions
The number of partitions is more than that of threads.

Accelerating *k*-mer sort by GPUs

Our algorithm sorts *k*-mers on GPUs. Our approach uses both CPU and GPU.
1. Reducing working memory for sorting *k*-mers on GPUs
The memory size required for sorting *k*-mers is larger than the global memory in a GPU.
⇒ The working memory for sorting *k*-mers is reduced by dividing *k*-mers buffers.

2. Improving the data structure for *k*-mers and frequencies
Using the array of the structures of a *k*-mer and a frequency is a simple and good for sorting based algorithm on a CPU. However, the accesses to the array of the structures of a *k*-mer and a frequency are slow.
⇒ Our algorithm uses the structure of the arrays of *k*-mers and a frequencies and "sort_by_key" as sort function in thrust.



3. Overlap between CPU and GPU calculations
The calculations on CPU requires computing time.
⇒ Our algorithm reduces total computing time to overlap CPU and GPU calculations.
- the calculations on CPU and GPU can be calculated independently.

Results

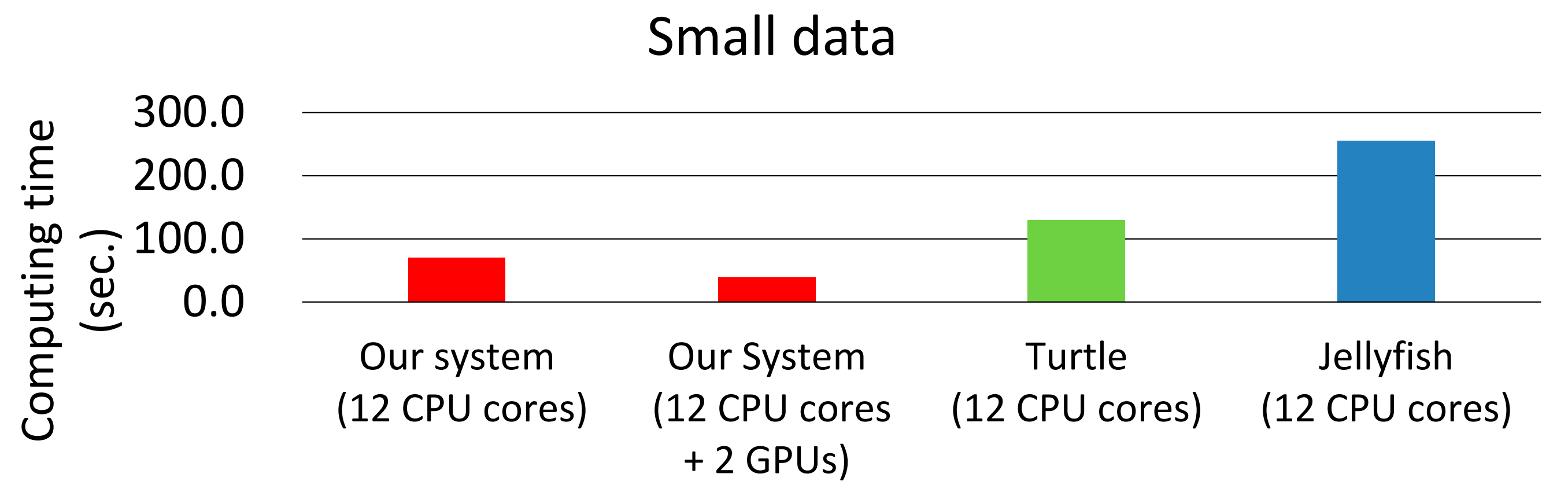
We performed *k*-mer counting to evaluate the performance of our system . We used the DNA sequence data obtained by using a next-generation sequencer. We compared our system with Turtle and Jellyfish. We used *k*=31, which is commonly used for the typical genome analyses.

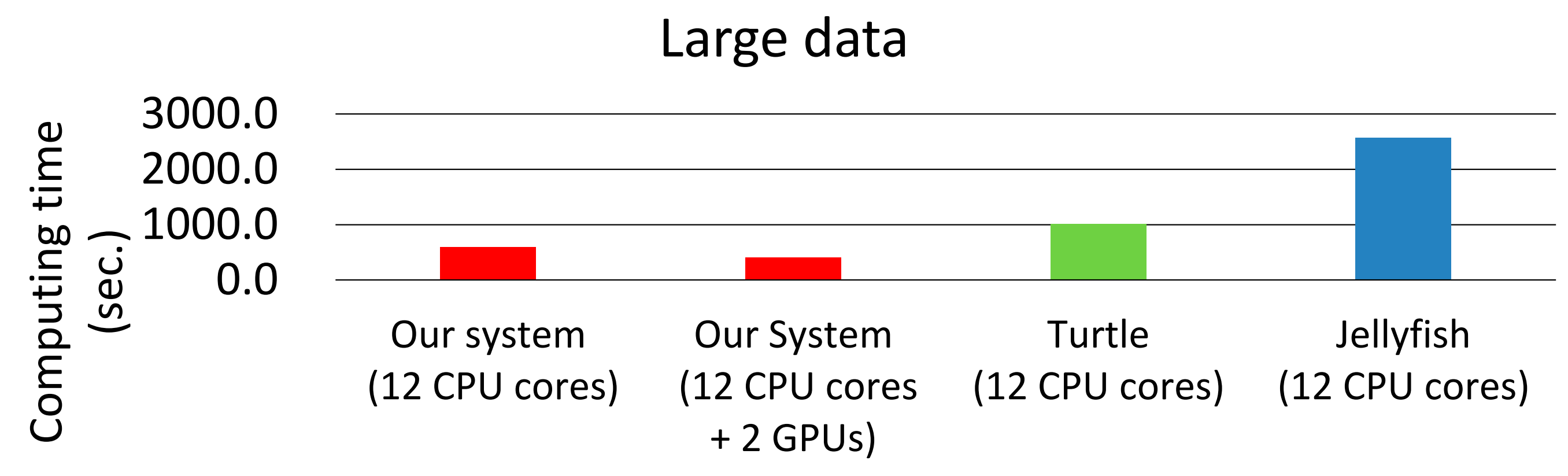
Computational environment:
TSUBAME2.5
(1 Thin node)

CPU: Intel Xeon 2.93 GHz (6 cores) x 2
GPU: NVIDIA Tesla K20X x 3
Memory: 96GB

Data set:
Small data:
D. Melanogaster dataset (SRX040485)
Sequencer: Illumina Genome Analyzer II
Read length: 77bp
Total size: 14GB

Large data:
G. Gallus dataset (SRX043656)
Sequencer: Illumina Genome Analyzer II
Read length: 100bp
Total size: 108GB





For large data, our system (12 CPU cores + 2 GPUs) was approximately
- **2.4 times** faster than Turtle (12 CPU cores)
- **6.3 times** faster than Jellyfish (12 CPU cores)

Our system with GPUs was approximately
1.5 times faster than without GPUs for large data.

References

[1] Roy RS, Bhattacharya D, Schliep A: Turtle: Identifying frequent k-mers with cache-efficient algorithms. arXiv:1305.1861 , 2013.
[2] Marçais G, Kingsford C: A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics 2011, 27:764–70.

Acknowledgments

This research was supported by a Grant-in-Aid for JSPS Fellows (248766) and HPCI Strategic Program Computational Life Science, Application in Drug Discovery and Medical Development by MEXT of Japan, Cancer Research Development funding by National Cancer Center, Japan and the CUDA COE Program by NVIDIA.