Integrating GPU-Accelerated Sequence Alignment and SNP Detection for Genome Resequencing Analysis

Mian Lu, Yuwei Tan, Jiuxin Zhao, Ge Bai, and Qiong Luo

Hong Kong University of Science and Technology
{lumian,ytan,zhaojx,gbai,luo}@cse.ust.hk

Abstract. DNA sequence alignment and single-nucleotide polymorphism (SNP) detection are two important tasks in genomics research. A common genome resequencing analysis workflow is to first perform sequence alignment and then detect SNPs among the aligned sequences. In practice, the performance bottleneck in this workflow is usually the intermediate result I/O due to the separation of the two components, especially when the in-memory computation has been accelerated, e.g., by graphics processors. To address this bottleneck, we propose to integrate the two tasks tightly so as to eliminate the I/O of intermediate results in the workflow. Specifically, we make the following three changes for the tight integration: (1) we adopt a partition-based approach so that the external sorting of alignment results, which was required for SNP detection, is eliminated; (2) we perform customized compression on alignment results to reduce memory footprint; and (3) we move the computation of a global matrix from SNP detection to sequence alignment to save a file scan. We have developed a GPU-accelerated system that tightly integrates sequence alignment and SNP detection. Our results with human genome data sets show that our GPU-acceleration of individual components in the traditional workflow improves the overall performance by 18 times and that the tight integration further improves the performance of the GPU-accelerated system by 2.3 times.

Keywords: data management for e-science, GPGPU, genomic data analytics.

1 Introduction

The second-generation DNA sequencing devices have been widely used for the past few years. They produce short DNA fragments, or short *reads*, at an ultra-high throughput. For today's genomics research based on short reads, two fundamental data analysis tasks are sequence alignment and single-nucleotide polymorphism (SNP) detection. Sequence alignment matches input reads to a reference sequence. SNP detection takes the output of alignment as input, and finds genetic variation information. In practice, these two tasks are typically performed in sequence as a basic workflow for genome resequencing analysis. Furthermore,

A. Ailamaki and S. Bowers (Eds.): SSDBM 2012, LNCS 7338, pp. 124–140, 2012.

the output of this workflow is usually adopted as input for a number of higher level applications, such as minor allele frequency (MAF) computation [4][20].

Traditionally, such a genome resequencing analysis workflow is implemented using multiple software packages, each performing a task in isolation. For example, 2BWT [6], SOAP2 [11], or Bowtie [9] can be used for the sequence alignment, and SOAPsnp [10] is used to detect SNPs. Moreover, as required by the SNP detection software, additional data processing tools are adopted between the alignment and SNP detection to sort the alignment results. Due to the large amount of data to process, this workflow may take an extremely long running time, e.g., around a week for the human genome. To improve the performance, previous studies have adopted graphics processors (GPUs) to speed up individual tasks, such as SOAP3 [13] and GSNP [14] for alignment and SNP detection, respectively. With the GPU acceleration, the evaluation based on operational genomics data sets have shown that the speedup is significant, e.g., up to 50 times. However, considering the alignment and SNP detection as a workflow, few previous studies further optimize it systematically.

We observe that, with the state-of-art GPU-accelerated tools, the overall performance of the workflow is dominated by the disk I/O, especially that incurred in the intermediate data processing between the alignment and SNP detection. Therefore, it is imperative to address the intermediate data processing in order to improve the overall performance of the workflow. In this work, we develop a GPU-accelerated genome resequencing analysis system tightly integrating the alignment and SNP detection for a higher overall performance. Our focus in this paper is on the integration techniques; details about GPU-acceleration for individual tasks can be found in previous studies [14][13].

We propose three techniques for the integration of alignment and SNP detection. Note that, although our system is based on the GPU for high efficiency, these techniques are applicable to both CPU- and GPU-based implementations.

1. To avoid the external sorting for SNP detection, we propose a partition-based approach. The partitioning is integrated in the alignment component. As a result, the intermediate result processing between the alignment and SNP detection is eliminated.
2. We move the computation of a global matrix that is originally calculated in the SNP detection to the component of sequence alignment. This move saves one scan of the alignment results in the workflow.
3. To further reduce I/O, we develop customized data compression techniques for alignment results.

With these techniques, our system is optimized for the genome resequencing analysis. Compared with a traditional workflow with individual components accelerated by the GPU, our integrated, GPU-accelerated workflow achieves a speedup of 2.3X. As a result, the new system improves the overall performance of a traditional CPU-based workflow by around 43 times.

The remainder of the paper is organized as follows. We introduce the background and related work in Section 2. We describe our integration techniques in detail in Section 3. We evaluate our system in Section 4 and conclude in Section 5.

2 Background and Related Work

In this section, we first briefly introduce the sequence alignment and SNP detection. Then we present the genome resequencing analysis workflow including these two functionalities.

2.1 Sequence Alignment

The second DNA generation sequencing devices can generate short DNA fragments, or *reads*, at an ultra-high throughput. The typical length of short reads is up to around one hundred base-pair (*bp*). For a given reference sequence and a large number of short reads, sequence alignment is to match each read against the reference. Mismatches are allowed for the alignment, e.g., typically two mismatches. The output file of sequence alignment contains multiple lines, and each line has a few attributes, such as the DNA base, the aligned position on the reference, the number of mismatches, and so on. We call such a line an *alignment*. Note that, one input read may have multiple alignments, as it may be matched to multiple positions on the reference. For the whole human genome, there are typically tens of billions of input short reads, and can generate an even larger number of alignments for the SNP detection.

Short read alignment algorithms can be categorized as hashing-based and Burrows-Wheeler transform (BWT) based. A hashing-based algorithm, such as WHAM [12], constructs a hash index containing the positions of all subsequences of the reference. In comparison, a BWT index is constructed with all suffixes of the reference and stored in suffix arrays. Alignment tools employing BWT index are Bowtie [9], 2BWT [6], SOAP2 [11], and SOAP3 [13]. Overall, a hashing based algorithm is efficient when the number of alignments is small, and the disadvantage is that the memory consumption is high. Therefore, in practice, the majority of sequence alignment tasks are done through the BWT index.

To improve the performance of sequence alignment, the GPU has been studied as a hardware accelerator, such as SOAP3 [13], and shown successful to speed up the processing significantly. When building this genome resequencing analysis system, we adopt our home-made GPU-accelerated sequence alignment tool for a tight integration. Our tool adopts a BWT-based sequence alignment algorithm. Additionally, compared with SOAP3, our tool adopts GPU-CPU coprocessing and customized data compression techniques. The measured performance of our sequence alignment component is slightly better than SOAP3.

2.2 SNP Detection

SNP detection is to find DNA variations for a single nucleotide between different members of a species. It calculates the likelihood and other information to

indicate whether a *site* (the position holding a base) is a SNP. For example, if the corresponding DNA fragments from two persons are ATCGGC and ACCGGC, respectively, then the second position is probably a SNP site.

A widely used SNP detection tool based on short reads is SOAPsnp [10] employing a Bayesian-based method. Due to the large size of input data, SOAPsnp reads and processes data window by window. A window is defined as a fixed number of consecutive sites on the reference. For a window of sites, the software loads the data related to the window (the corresponding alignments) from disk to memory to perform the computation and outputs SNP results.

To speed up the process of SNP detection, our previous work GSNP [14] implements the same functionality as SOAPsnp but adopts the GPU acceleration. With various optimization techniques, GSNP can achieve a speedup of around 50X over the CPU-based single-threaded SOAPsnp. This resequencing analysis system adopts GSNP as the component of SNP detection, with modifications for a tight integration.

2.3 The Workflow of Genome Resequencing Analysis

Overall, the genome resequencing analysis consists of the sequence alignment and SNP detection. Although the alignment result is the input of SNP detection, in practice, an additional data processing step is required between the alignment and SNP detection tools. This step is to sort the alignment result as well as data format conversion for the SNP detection tool. Therefore, traditionally, three separate software tools are used in the workflow, such as SOAP2 [11], *msort* [2][15], and SOAPsnp [10] for the alignment, sorting, and SNP detection, respectively. Figure 1 shows the overview of such a workflow. We describe the input and output of each software component in detail. Note that, as they are separate software packages, the input and output data are both stored on disk.

Alignment. The input for sequence alignment are the reference sequence and a large number of short reads, which are stored in plain text files. In practice, as the data size may not fit into the memory, multiple passes are performed for the alignment. The alignment result file can be very large, e.g., tens of gigabytes.

Sorting. The alignments should be sorted according to their matched positions on the reference before performing SNP detection. The purpose of sorting is to make the SNP detection tool process sites window-by-window. Figure 2 illustrates the sorting and the process of window-based SNP detection. As the data size is very large, this step is implemented using external sorting algorithms, which are expensive. The GNU *msort* [15] can be used to sort alignments. There are also other more efficient implementations, such as a dedicated alignment sorting tool [2]. By default, in this paper, the sorting program refers to this improved alignment sorting tool rather than GNU *msort*, unless otherwise specified.

SNP Detection. Overall, there are two steps in this task. The first step is to calculate a global matrix, which requires to access all alignment results. Based on the global matrix, the second step calculates likelihood for each site, which

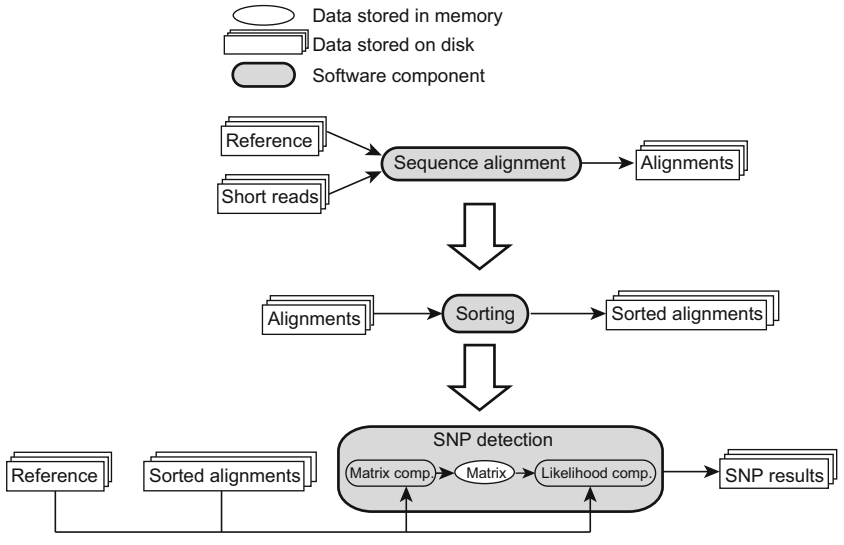


Fig. 1. The traditional workflow of genome resequencing analysis, which consists of sequence alignment, sorting, and SNP detection. These components adopt separate software packages.

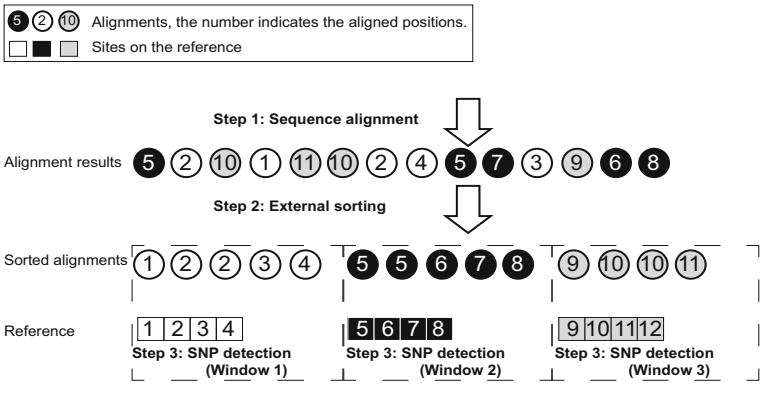


Fig. 2. Sorting and window-based SNP detection. Suppose that there are three windows, each of which contains four sites. A circle represents an alignment at a given aligned position on the reference. The color of a circle indicates which window an alignment or a site belongs to.

accesses all alignments again. Note that, in the second step, the computation for each site is independent. Therefore, with the sorted alignment results, the likelihood computation can adopt a window-based approach. This way, each window of sites and its related data that can fit into memory are loaded from disk to the main memory for the computation.

In summary, a traditional workflow employing three separate software packages contains an expensive external sorting step as well as redundant I/O accesses. Particularly, with the GPU acceleration for the in-memory computation, the I/O dominates the overall performance. We analyze the I/O cost in detail in Section 3.1. In our system, we address these issues through a tighter integration for the sequence alignment and SNP detection components. Moreover, we eliminate the external sorting through an inexpensive partition-based approach.

2.4 Related Work

Existing work on sequence alignment and SNP detection rarely considers integration techniques for a workflow. One exception is work by Wegrzyn et al. [19], which proposes a sequence alignment and SNP detection pipeline that utilizes machine learning algorithms to improve the speed and accuracy. However, their work is not based on short reads and practical algorithms used today. To the best of our knowledge, our study is the first to propose effective optimizations to tightly integrate state-of-the-art alignment and SNP detection algorithms to improve the overall performance systematically.

In addition to a single-machine solution, cloud computing solutions are investigated to improve the performance and scalability of sequence analysis. CloudBurst [16] is a parallel short sequence alignment program developed using Hadoop [1], whose running time scales near linearly with the number of nodes. Myrna [7] targets at gene expression calculation from large-scale RNA data sets, which combines Bowtie [9] and Bioconductor [3]. *Crossbow* [8] is a system that is built on Bowtie [9] and SOAPsnp [10] to perform the genome resequencing analysis in cloud computing using Hadoop. It first performs sequence alignment in the map phase on each node, then sorts the alignment result across all nodes, and finally detects SNPs on each node.

Compared with Crossbow as well as other cloud computing based systems, in addition to the GPU acceleration adopted in our system, we further consider optimizations for a tight integration on a single node. These single node optimizations can be applied on each node in the cloud computing environment. Furthermore, with our optimizations applied, the sorting phase in the MapReduce framework can be avoided.

Finally, there are a few studies for the GPU-accelerated sequence alignment, such as GPU-BLAST [18] and MUMmerGPU [17], which are designed for long reads. For the short read alignment, both SOAP3 [13] and BarraCUDA [5] implement the BWT-index based sequence alignment algorithm.

3 System Implementation

In this section, we describe the details of our integration techniques. Note that, as our system is built on the components with the GPU acceleration, by default, the sequence alignment and SNP detection components referred to in this paper are the GPU-based implementations, unless specified otherwise. Specifically, the sequence alignment is our home-made implementation, and the SNP detection is based on our previous work GSNP [14]. However, our integration techniques are applicable to both GPU- and CPU-based systems.

3.1 Analysis on the Traditional Workflow

As described in Section 2.3, the traditional workflow consists of three separate software packages to perform the sequence alignment, sorting, and SNP detection. Table 1 lists the time breakdown of a traditional workflow (with GPU acceleration) using the hardware and data sets described in Section 4.1. The three I/O intensive components, namely Output, Sorting, and Matrix Computation, take a total of 60% of the overall time. The Output component contains disk I/O operations only. In Matrix Computation, disk I/O takes around 92% of elapsed time. Since the source code of the sorting program is unavailable, we estimate the I/O in Sorting to be half of the elapsed time, assuming one read and one write for each alignment in sorting. There are two major issues through our further analysis. First, there are redundant I/O accesses. Each alignment record is accessed multiple times across the three software packages. Second, there is an expensive external sorting step. In our system, our target is to address these two issues for efficiency through a tight integration.

Table 1. Elapsed time of the traditional workflow (with GPU acceleration)

	Sequence alignment			Sorting	SNP detection	
	Input	Computation	Output		Matrix comput.	Likelihood comput.
Time (sec)	35	213	104	550	155	298
Percentage (%)	2.3	15.9	7.8	41	10.8	22.2

For the first issue, Figure 3 illustrates the multiple data reads and writes on alignment results. Note that, the sorting requires at least one read and one write for each alignment. It may incur more I/O depending on the buffer size. Additionally, as described in Section 2.3, there are two steps in the SNP detection (global matrix and likelihood computation), and each requires a full scan on the alignment results. These two scans on the same alignment results cannot be merged, as the likelihood computation relies on the result of the global matrix computation. In summary, for each alignment, there are at least five disk accesses: two reads and three writes (as shown in Figure 3). In our system, we optimize these multiple data accesses to only two necessary accesses: one write when generating the alignment, and one read when performing the SNP

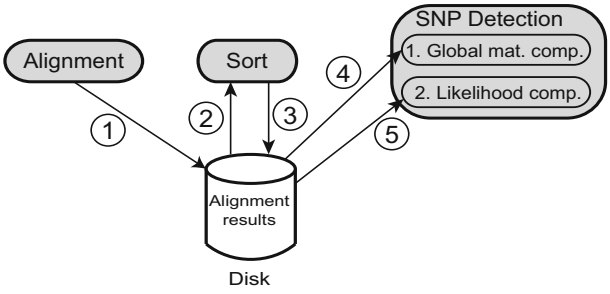


Fig. 3. Five accesses for each alignment in the traditional workflow. (1) Result output after the sequence alignment. (2) Input for the sorting. (3) Sorted result output. (4) Data input for the global matrix computation. (5) Data input for the likelihood computation.

detection. Note that, as the size of alignment file is usually large, e.g., tens of gigabytes, and some other tools may use the alignment results, we consider it necessary to store the alignment results as the intermediate data on disk. The reference sequence is also accessed twice in the workflow. However, the reference size (up to around 750 MB for the whole human genome) is much smaller than the alignment result, and it is straightforward to eliminate the second access by keeping it in memory.

For the second issue, we have presented the purpose of sorting in Section 2.3. Essentially, the sorting is used to arrange the alignments in the same SNP processing window consecutively on disk. Within a window, the order of alignments is not important for the SNP detection program, as there is a counting step to extract summary information for each alignment. Based on this observation, we propose to use range partitioning to achieve the same purpose, but with a low time cost.

3.2 System Overview

Overall, our system consists of the sequence alignment and SNP detection components, and works as follows. First, input reads are processed window-by-window for the sequence alignment. Within each window, when an alignment is produced, it is used immediately to update the global matrix that is used for detecting SNPs later. Then the partitioning function is applied to that alignment, and the alignment is stored in an in-memory buffer. When the buffer is full, its alignments will be compressed and written to the disk. After the sequence alignment for all input reads is done, we start the SNP detection component. The SNP detection component is also executed window-by-window. The window size depends on the partitioning function. Note that, there is no dependence between the window sizes of sequence alignment and SNP detection. Figure 4 illustrates the software components and the workflow in our system.

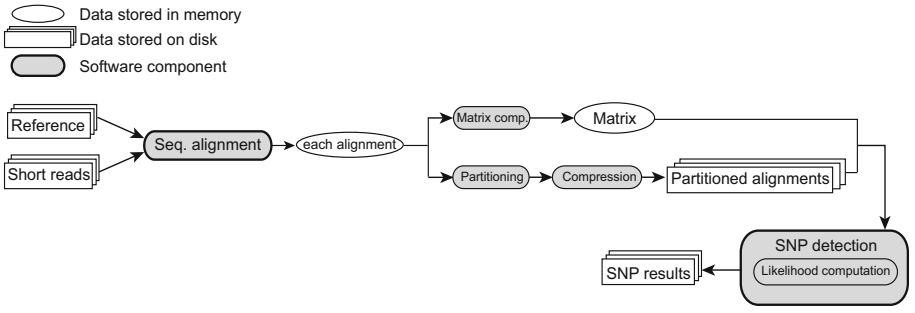


Fig. 4. Components and workflow in our genome resequencing analysis system

3.3 Range Partitioning

For range partitioning, we keep a number of buffers in memory. When an alignment is produced, it is sent into one buffer according to its aligned position in the reference. When a buffer is full, the alignments stored in that buffer are written to the disk (we call the data of each buffer a *block*). In order to facilitate the window-based processing in the SNP detection, the number of buffers and SNP processing windows are the same. Specifically, we maintain B buffers in memory, and each can hold up to m alignments. For the alignment that is matched to site i in the reference, it will be stored in the $\lfloor \frac{i}{m} \rfloor th$ buffer. If the buffer is full, m alignments in that buffer (as a data block) are written to the disk. As one window in the SNP detection may contain alignments from multiple data blocks, there is an additional data structure maintaining the block IDs for each window. Figure 5 illustrates an example of the partitioning corresponding to the sorting example (Figure 2).

The computation complexity of such a partition-based approach is $O(n)$, where n is the number of alignments. Moreover, when an alignment is generated, we can apply partitioning immediately without storing these original alignments on disk. The memory space cost of partitioning is on the in-memory alignment buffers and the block ID list. Suppose we have B buffers (or B SNP processing windows), and each can hold m alignments. Suppose each alignment requires b bytes, the total memory consumption is $(B \times m \times b)$ bytes. As the SNP detection is much more expensive than the partitioning, we mainly consider the performance of SNP detection to tune these parameters. The typical window size in the SNP detection is 256,000 sites [14] with around 1.5 GB GPU memory and 1 GB main memory consumed. A larger window size has little performance impact on SNP detection but significantly increases the memory consumption. Therefore, we set the SNP processing window size in our system as 256,000 sites by default. This way, the number of buffers is up to 11,719 ($B = 11,719$), when evaluating the whole human genome consisting of three billion sites. If the size of each buffer is 512 KB ($m \times b = 512$ KB), which saturates disk bandwidth, then each buffer can store around 2,000 alignments for 100-bp reads. As a result, the total memory consumption for the whole human genome is around 6 GB for

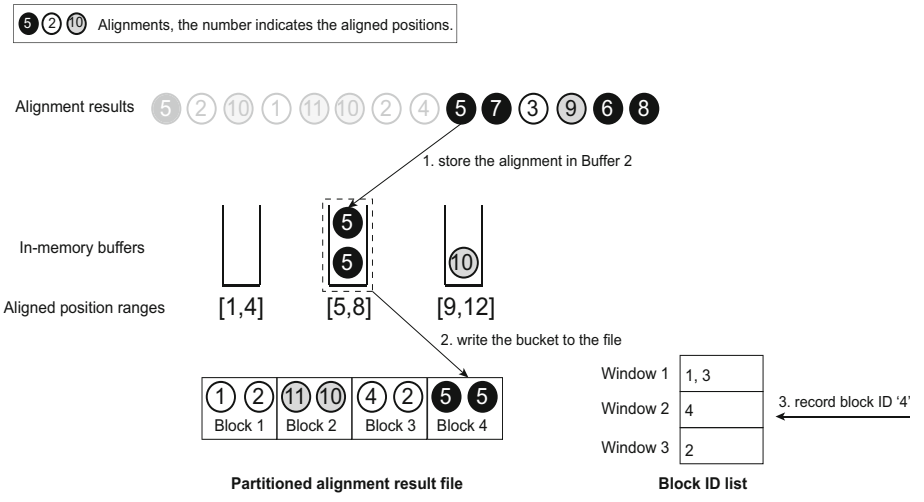


Fig. 5. An example of partitioning when handling the 9-*th* alignment with the aligned position 5. There are three buffers, and each can hold up to two alignments. The first eight alignments have been finished for the partitioning. **Step 1**: the alignment is stored in the second buffer for aligned positions from 5 to 8. **Step 2**: the second buffer is full, thus alignments are written to the disk as Block 4. **Step 3:** the block ID *4* is recorded in the list for Window 2.

in-memory buffers. This is affordable on our server. Additionally, the number of entries in the block ID list can be estimated as $\frac{n}{m}$, thus the memory consumption is around $(4 \times \frac{n}{m})$ bytes. Based on estimation, the memory consumption of the block ID list is around tens of megabytes for the whole human genome. Note that, in practice, the SNP detection is usually performed for a given individual human chromosome rather than the whole human genome, which requires less memory, allows larger buffers for efficiency.

3.4 Alignment Result Compression

Through partitioning and moving matrix computation forward, we have eliminated redundant I/O accesses, however, the size of the alignment result may still be very large. To further improve the performance of the workflow, we develop customized data compression techniques. Note that, we do not adopt general data compression algorithms or tools, such as *gzip*, as they introduce expensive computation cost for both compression and decompression. Additionally, the compression is applied before writing the alignment to the disk, and then the SNP detection component can decompress these compressed result in-memory directly, without additional disk I/O.

Recall that alignment tools output the result as multiple lines, and each line corresponds to an alignment with multiple attributes. Although various alignment tools have slightly different output formats, almost all alignment tools

contain these attributes required by the SNP detection tool: the read bases, the quality scores, the number of alignments for the given read, the read length, the reference name, the aligned position, the number of mismatches, and the mismatch information (including the positions of mismatches occurred and their substitution). Specifically, the read bases and quality scores together take more than 90% of the total size. This is because each base has one quality score, and each alignment represents multiple bases. Suppose the read length is l, then the read bases and quality scores attributes both have l values for an alignment. In comparison, for other attributes, each of them only has one value for an alignment.

For the read bases, we do not store them in the alignment result file. The basic idea is that the read can be reconstructed based on the aligned position, mismatch information, and the reference. Figure 6 shows an example of such an approach. The cost of such an approach is that we need to store the reference in memory and have computation overhead when reconstructing the read bases. However, the SNP detection requires to access the reference anyway, and the reference size is much smaller than the alignment result. Additionally, the computation cost is negligible compared with the saved I/O cost.

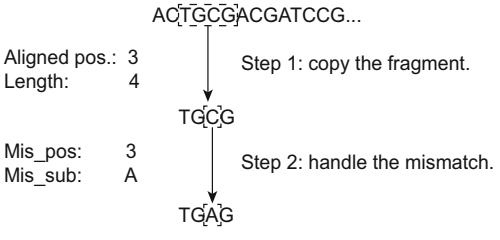


Fig. 6. An example of extracting the read bases based on the aligned position and mismatch information. Step 1: according to the aligned position and read length, we copy the fragment from the reference. Step 2: according to the position of mismatch occurred (mis_pos) and its substitution (mis_sub), we perform the mismatch on the copied fragment.

On the compression of the quality scores, the observation is that one read usually has multiple alignments. These alignments have the same quality score string. Therefore, we keep a table of all unique score strings, and append an additional ID attribute to each alignment for fetching the correct quality score string for a given alignment. To further compress this table, we apply dictionary encoding and run-length encoding.

4 Evaluation

In this section, we first study the performance impact of our integration techniques. Then we compare the end-to-end performance of our system with the original workflow, including the GPU- and CPU-based implementations.

4.1 Experimental Setup

Hardware Setup. We conduct the experiments on a server equipped with an NVIDIA Tesla C2070 and two Intel Xeon E5520, 2.27 GHz quad cores (8 cores, 16 threads in total). C2070 consists of 448 cores and 6 GB GPU memory. The server has 32 GB main memory.

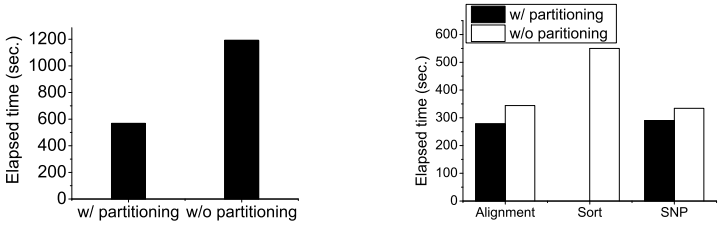
Implementation Details. Our system is built using the GPU acceleration. Specifically, the alignment component is implemented by ourselves based on the Bi-BWT algorithm [6]. The alignment results are reported with up to two mismatches. The SNP detection component is modified based on our previous work GSNP [14]. By default, the window size of the alignment and SNP detection are fixed to 1,000,000,000 reads and 256,000 sites, respectively, and the buffer size in the range partitioning is 1024 KB, unless otherwise specified. Additionally, the time of loading the alignment index from the disk to the main memory is excluded from measurement, as the index can reside in memory. We compare our system with the traditional GPU-based and CPU-based workflows. Recall that the traditional workflow consists of three separate software packages. The traditional GPU-based workflow adopts our home-made GPU-accelerated alignment tool, *msort* developed by BGI-Shenzhen [2] (denoted as *msort*), and GSNP [14]. The traditional CPU-based workflow adopts 2BWT [6], *msort* [2], and SOAPsnp [10]. We use 2BWT rather than other alignment software as it outperforms other tools in our evaluation. Additionally, all CPU-based implementations are single-threaded.

Data Sets. We use a data set for human chromosome 1 (Ch. 1), which is provided by our collaborator, BGI-Shenzhen. This data set contains 15 million short reads in total, and each is 100 *bp* long. The file of short reads is around 3.3 GB. The number of alignments for this data set is 38,584,511. Without compression, the alignment result file is around 9.4 GB. The reference contains around 247 million base pairs. The final SNP detection result file is around 800 MB.

4.2 Performance Impact of Integration Techniques

We first study the performance impact of three integration techniques. For each group of experiments, we compare the implementation without a specific technique with the optimized implementation. Note that, in each group, only the investigated technique will be removed and other optimizations are still employed. Overall, for performance comparison, we divide our system into two components: sequence alignment, and SNP detection. Note that, for our system, the alignment component performs alignment, global matrix computation, partitioning, and compression, and the SNP detection component contains the likelihood calculation step of the SNP detection (as shown in Figure 4).

Range Partitioning. If we do not use partitioning, the system works as follows: we first perform alignment and store all alignments on disk; then we perform



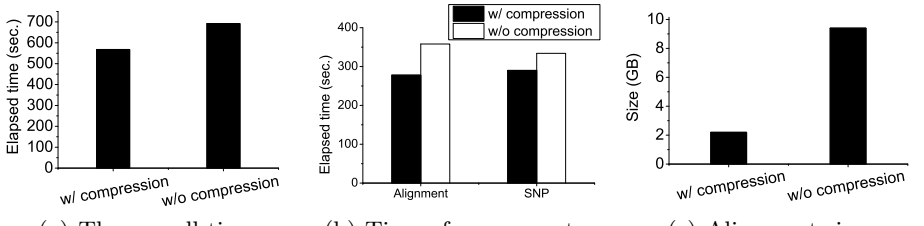
(a) The overall time. (b) Time of components.

Fig. 7. Performance comparison between using partitioning and using sorting

the external sorting; finally the SNP detection is used based on the sorted alignments stored on disk. Note that, if we use sorting, the compression becomes inapplicable, as the sorting tool can only be used to sort the file stored in specific formats. Figure 7(a) shows that with partitioning, the system is around 2X faster than that using the sorting. Figure 7(b) compares the elapsed time of three components when the system adopts the partitioning and sorting. As the data compression cannot be used when sorting is adopted, the sizes of the alignment output and SNP input both become larger, which slows down the performance of both components.

Data Compression for Alignment Results. Without the compression, the alignment result that is stored on disk as intermediate data will be larger. Figure 8(a) compares the overall elapsed time of the system with and without the data compression techniques for alignment results. The figure shows that with the compression, the overall performance is improved by around 20%. Figure 8(b) shows that, due to the reduced size of the alignment result file, the alignment and SNP detection components are around 22% and 13% faster than their counterparts without the compression, respectively. Figure 8(c) shows that the size of the compressed alignment result is only around 23.4% of that without the compression.

Move of Matrix Computation. Recall that, compared with the original workflow, our system eliminates a data scan on the alignment result when detecting



(a) The overall time. (b) Time of components. (c) Alignment size.

Fig. 8. Performance comparison between with and without alignment result compression

Table 2. End-to-end performance (seconds) comparison. OLD-CPU and OLD-GPU indicate the traditional workflow using the CPU- and GPU-based software, respectively.

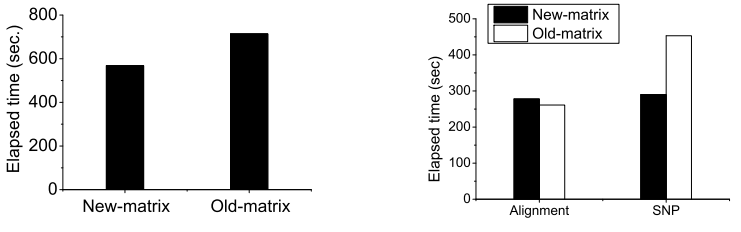
	Alignment	Sort	SNP detection	**Overall**
OLD-CPU	1562	550	22321	24433
OLD-GPU	330	550	453	1333
Our system	278	–	290	568

SNPs due to the change of matrix computation. Figure 9(a) shows the overall elapsed time of the system with and without the move of the global matrix computation. With this technique, the overall performance is improved by around 18%. This improvement is from the elimination of additional disk I/O on the alignment result when detecting SNPs. Figure 9(b) further shows that, as the global matrix is calculated in the alignment component in our system, this component is slightly slowed down. However, the overall elapsed time is reduced due to the significant performance improvement from the SNP detection component.

In summary, the range partitioning is the most significant optimization, which can eliminate the expensive external sorting. With the range partitioning, the system is around 2X faster than that without the optimization. The data compression technique and move of matrix computation can further improve the performance by around 20% and 18%, respectively.

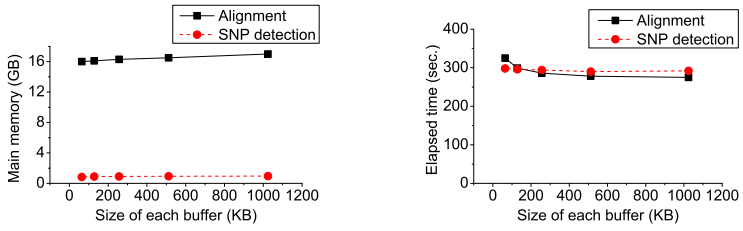
4.3 End-to-End Performance Comparison

We show the end-to-end performance comparison in Table 2. Note that, in our system, the partitioning, compression, and matrix computation are all included in the alignment component. This table shows that, the traditional workflow with the GPU acceleration for individual components outperforms its CPU counterpart by around 18 times. Furthermore, with the integration techniques, our system further improves the performance by around 2.3 times. This improvement is from all three components. First, for the alignment, the compression reduces the alignment size to save the I/O time. Second, the original expensive sorting is



(a) The overall time. (b) Time of components.

Fig. 9. Performance comparison between the systems with the move of global matrix computation (new-matrix) and the original global matrix computation (old-matrix)



(a) Memory consumption. (b) Elapsed time.

Fig. 10. The memory consumption and elapsed time with the buffer size in the range partitioning varied

eliminated in our system. Third, for the SNP detection, one data scan on alignment results is also eliminated in our system. Compared with the traditional CPU-based workflow (without the tight integration techniques), our system is around 43X faster.

Finally, we investigate the performance and memory consumption impact with the partitioning buffer size varied. The windows sizes of alignment and SNP detection are set according to the performance of alignment and SNP detection components. We only show the main memory consumption as the buffer size does not affect the GPU processing. Figure 10(a) shows that the memory consumption slightly increases when the buffer becomes larger for the alignment. This is because another data structure (a suffix array) dominates the overall memory usage for the alignment, which consumes around 12 GB. For the SNP detection, a larger partitioning buffer results in a smaller block ID list, which is insignificant in the overall memory consumption. Figure 10(b) shows that the alignment can benefit from a larger buffer, since the disk I/O throughput is higher when writing a larger data block. This parameter is less significant for the performance of SNP detection.

5 Conclusion

We have developed a GPU-accelerated system with a tightly integrated workflow optimized for genome resequencing analysis: the sequence alignment is used first for short reads, and then the SNPs are detected based on the alignment result. To reduce the I/O overhead in the traditional workflow, we propose three techniques for a tight integration of the alignment and SNP detection. We first use range partitioning to avoid the external sorting for alignment results. We also develop customized data compression techniques to further reduce the size of the alignment result. Finally, we calculate the global matrix computation when generating alignments, which is originally performed in the SNP detection component. As a result, compared with the traditional GPU- and CPU-based workflow consisting three separate software packages, our system can achieve a speedup of around 2.3X and 43X, respectively.

Acknowledgment. This work was supported by grants 617509 from the Hong Kong Research Grants Council and MRA11EG01 from Microsoft SQL Server China R&D. We thank our collaborator BGI Shenzhen for providing us application requirements and data sets.

References

1. Apache Hadoop, `http://hadoop.apache.org/`
2. Short Oligonucleotide Analysis Package, BGI-Shenzhen, China, `http://soap.genomics.org.cn`
3. Gentleman, R., Carey, V., Bates, D., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., Hornik, K., Hothorn, T., Huber, W., Iacus, S., Irizarry, R., Leisch, F., Li, C., Maechler, M., Rossini, A., Sawitzki, G., Smith, C., Smyth, G., Tierney, L., Yang, J., Zhang, J.: Bioconductor: open software development for computational biology and bioinformatics. Genome Biology 5(10) (2004)
4. Kim, S.Y., Lohmueller, K.E., Albrechtsen, A., Li, Y., Korneliussen, T., Tian, G., Grarup, N., Jiang, T., Andersen, G., Witte, D., Jorgensen, T., Hansen, T., Pedersen, O., Wang, J., Nielsen, R.: Estimation of allele frequency and association mapping using next-generation sequencing data. BMC Bioinformatics 12, 231 (2011)
5. Klus, P., Lam, S., Lyberg, D., Cheung, M.S., Pullan, G., McFarlane, I., Yeo, G., Lam, B.: BarraCUDA - a fast short read sequence aligner using graphics processing units. BMC Research Notes 5(1) (2012)
6. Lam, T.W., Li, R., Tam, A., Wong, S., Wu, E., Yiu, S.M.: High throughput short read alignment via bi-directional bwt. In: IEEE International Conference on Bioinformatics and Biomedicine, pp. 31–36 (2009)
7. Langmead, B., Hansen, K., Leek, J.: Cloud-scale RNA-sequencing differential expression analysis with myrna. Genome Biology 11(8) (2010)
8. Langmead, B., Schatz, M., Lin, J., Pop, M., Salzberg, S.: Searching for SNPs with cloud computing. Genome Biology 10(11) (2009)
9. Langmead, B., Trapnell, C., Pop, M., Salzberg, S.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology 10(3) (2009)
10. Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., Wang, J.: SNP detection for massively parallel whole-genome resequencing. Genome Research 19(6), 1124–1132 (2009)
11. Li, R., Yu, C., Li, Y., Lam, T.-W.W., Yiu, S.-M.M., Kristiansen, K., Wang, J.: SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics 25(15), 1966–1967 (2009)
12. Li, Y., Terrell, A., Patel, J.: Wham: A high-throughput sequence alignment method. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (2011)
13. Liu, C.-M., Lam, T.-W., Wong, T., Wu, E., Yiu, S.-M., Li, Z., Luo, R., Wang, B., Yu, C., Chu, X., Zhao, K., Li, R.: SOAP3: GPU-based Compressed Indexing and Ultra-fast Parallel Alignment of Short Reads. In: Third Workshop on Massive Data Algorithmics (2011)
14. Lu, M., Zhao, J., Luo, Q., Wang, B., Fu, S., Lin, Z.: GSNP: A DNA Single-Nucleotide Polymorphism Detection System with GPU Acceleration. In: International Conference on Parallel Processing, ICPP (2011)

15. Poser, W.: GNU msort, http://billposer.org/Software/msort.html
16. Schatz, M.C.: CloudBurst: highly sensitive read mapping with MapReduce. Bioinformatics 25(11), 1363–1369 (2009)
17. Trapnell, C., Schatz, M.C.: Optimizing data intensive gpgpu computations for dna sequence alignment. Parallel Computing 35, 429–440 (2009)
18. Vouzis, P.D., Sahinidis, N.V.: GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics 27(2), 182–188 (2011)
19. Wegrzyn, J.L., Lee, J.M., Liechty, J., Neale, D.B.: PineSAPsequence alignment and SNP identification pipeline. Bioinformatics 25(19), 2609–2610 (2009)
20. Yi, X., Liang, Y., et al.: Sequencing of 50 Human Exomes Reveals Adaptation to High Altitude. Science 329(5987), 75–78 (2010)