Toward High Utilization of Heterogeneous Computing Resources in SNP Detection

Myungeun Lim, Minho Kim, Ho-Youl Jung, Dae-Hee Kim, Jae-Hun Choi, Wan Choi, and Kyu-Chul Lee

As the amount of re-sequencing genome data grows, minimizing the execution time of an analysis is required. For this purpose, recent computing systems have been adopting both high-performance coprocessors and host processors. However, there are few applications that efficiently utilize these heterogeneous computing resources. This problem equally refers to the work of single nucleotide polymorphism (SNP) detection, which is one of the bottlenecks in genome data processing. In this paper, we propose a method for speeding up an SNP detection by enhancing the utilization of heterogeneous computing resources often used in recent high-performance computing systems. Through the measurement of workload in the detection procedure, we divide the SNP detection into several task groups suitable for each computing resource. These task groups are scheduled using a window overlapping method. As a result, we improved upon the speedup achieved by previous open source applications by a magnitude of 10.

Keywords: SNP detection, overlapped window, heterogeneous computing resources, CPU-GPU overlapping, multithreading.

Manuscript received Aug. 25, 2014; revised Jan. 26, 2015; accepted Feb. 3, 2015.

This work was supported by the IT R&D program of MKE/KEIT, Rep. of Korea [10038768, the Development of Supercomputing System for the Genome Analysis].

Myungeun Lim (melim@etri.re.kr), Minho Kim (kimmh@etri.re.kr), Ho-Youl Jung (hoyoul.jung@etri.re.kr), Dae-Hee Kim (dhkim98@etri.re.kr), Jae-Hun Choi (jhchoi@etri.re.kr), and Wan Choi (wchoi@etri.re.kr) are with the IT Convergence Technology Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Kyu-Chul Lee (corresponding author, kclee@cnu.ac.kr) is with the Department of Computer Engineering, Chungnam National University, Daejeon, Rep. of Korea.

I. Introduction

With the development of next-generation sequencing (NGS) technology [1], it has become possible to obtain human genome data at a relatively low cost. As the volume of genomic data is increasing, genomic researches using NGS data, such as variation discovery, de novo assembly, and genome-wide profiling, are being actively explored. The 1000 Genomes Project [2], which focuses on searching for rare variants that have less than 1% minor allele frequency of a normal genome, and the Cancer Genome Atlas [3], which is dedicated to the search for single-nucleotide variants and structural variants related to major cancers, are representative NGS applications.

Data obtained from NGS technology are sets of genome data fragments. Variation discovery, often called variation detection, is a procedure used to identify variant sites such as single-nucleotide or structural variants from the re-sequenced data. Single nucleotide polymorphism (SNP) refers specifically to an amino acid that appears differently from a reference genome within a locus of a given genome sequence. As demonstrated by the above representative researches, SNP has become primitive information in genome-wide association or disease-related studies.

In the early stages of SNP detection research, a cut-off was applied to the allele count and quality score when determining the SNP [4]–[5], but there are problems with a low accuracy of SNP detection when the sequencing depth is low. To minimize this imprecision, a system using statistical methods such as the Bayesian probability was developed to enhance the accuracy. Samtools' mpileup [6], SOAPsnp [7], and Genome Analysis ToolKit's (GATK) UnifiedGenotyper [8]–[9] are the most prominent systems for SNP detection based on a Bayesian probability model.

The amount of NGS data is huge, especially in the case of a whole genome. Moreover, as the cost of generating NGS data decreases, the amount of genome sequencing data is increasing even faster. The amount of raw data of a whole genome for about 30-fold coverage is over 200 GB, and the mapping results to the reference genome total about 90 GB. Thus, a significant amount of time is required in SNP detection. Various efforts have been made to expedite the analysis pipeline, including parallel computing using coprocessors such as a graphics processing unit (GPU) and distributed processing techniques such as Hadoop's map-reduce method [10]. The many-core architecture of a GPU makes it possible to execute an operation in parallel and manipulate massive data rapidly. In the past, GPUs were mainly used in graphics applications [11], but more recently, their usage has been extended to the area of accelerated computing [12]. In particular, lately, GPUs have been used in many applications for genome analysis requiring high-performance computing capability. CUDA-SW++ [13] and SOAP3 [14] are GPU-based sequence aligners, and Amber-GPU [15] is a well-known simulation tool used in molecular dynamics. It is also used for genomic data compression [16]. However, these are still insufficient when we consider that many other software (SW) are required in a genome analysis. Moreover, since many techniques using a GPU [13]–[16] have used GPU-centric approaches, the aspect of efficiently utilizing whole computing resources has been overlooked.

This paper proposes a novel method for reducing SNP detection time by enhancing the usability of heterogeneous processors in high-performance computing systems. Through a workload analysis of the tasks in SNP detection, we classify the tasks into CPU- and GPU-oriented tasks. We then propose a new task scheduling method to run these heterogeneous tasks in parallel and implement an SNP detection system using these methods. The proposed system supports the standard I/O format BAM and variant list (VCF) to provide the flexibility of combining other open-source analysis SW in building a genome analysis pipeline

The contents of this paper are as follows. Section II discusses previous researches related to an SNP analysis, and Section III describes the structures of the proposed system. Section IV details the algorithm used by the suggested method, and Section V discusses the experimental results of the system. Finally, we provide some concluding remarks in Section VI.

II. Related Works

Researches on SNP detection can be subdivided into cutoff-based and statistical model–based methods. In earlier SNP detection methods, the cutoff of each site's allele count and quality score was applied to determine the SNP or genotype. For example, if the allele count and quality score satisfy a certain ratio and the reference at a particular locus appears differently from the allele in a sample data, then this allele is classified as a heterozygous genotype. This type of method works well with a sufficiently large sequencing depth. However, if this requirement is not met, owing to low depth data, then a heterozygous allele can be under-called. Hedges and others attempted to relieve this problem through the use of an empirical threshold [17].

To improve the accuracy of the cutoff-based method, statistical methods have been developed. Using a Bayesian probability formula, the calculation for the genotype likelihood at a given site is conducted based on the prior probability and likelihood from a given read sequence set. As mentioned before, SAMtools' mpileup, SOAPsnp, and GATK's UnifiedGenotyper are well-known SNP detection tools that use the Bayesian probability model. In SOAPsnp [7], the genotype likelihood is calculated with several attributes to improve the detection accuracy. In addition to the allele type, three attributes — the quality score, the allele coordinates in the read, and the genotype occurrence — are utilized when calculating the genotype likelihood to reduce errors in the sequencing or mapping stage and achieve a higher accuracy. The genotype is then determined through the calculation of likelihood with the highest posterior. SOAPsnp also used the unique prior value reflecting the features acquired from interpreting the dbSNP [18] variant dataset.

Another line of research aims at improving the runtime performance of genome data analysis tools, including SNP detection tools. GSNP [19], a GPU version of SOAPsnp, proposes a method of utilizing a GPU to improve its runtime performance. The flow of SOAPsnp consists of mapped data reading, probability matrix build, genotype count, likelihood calculation, posterior calculation, and output consensus. While a single-thread CPU can only sequentially process according to each site's reference one at a time, GSNP utilizes the parallelization to process multiple sites simultaneously by assigning each GPU thread to a site. To minimize the bottleneck of GPU memory copying and reduce unnecessary calculations of empty data, the representation of the mapping information is revised to a sparse structure that consumes less GPU memory.

Crossbow [20] is a genome analysis pipeline in a Hadoop-based cloud computing environment that uses Bowtie [21] and SOAPsnp. When the user uploads the sequence data to the file system, the read alignment is processed using Bowtie at the map stage concurrently. The aligned reads from the map stage are rearranged by a genome partition in the sort step, and the SNP of each partition gets called at the reduce stage, which

then becomes merged and saved as the output. Because of its operation in a cluster environment, Crossbow has the advantage of having the ability to expand the system for faster data processing.

While Crossbow attempts to fully utilize the available resources in a distributed environment, our method in this paper does so in a single computing environment. Our approach is different from GSNPs in that we fully exploit both the CPU and the GPU in computing SNP, but GSNP has a biased usage of the GPU.

III. Overview of Genome Analysis System

1. SNP Analysis Pipeline

Generally, a variation discovery goes through a series of analysis steps: read mapping, SAM/BAM format conversion, sorting of the mapped results, merging of the sorted results, and SNP detection, as shown in Fig. 1. Using NGS technology, a whole genome with a length of about three-billion bases is broken into small fragments with a length of 35 to 250 base pairs. The fragments are called read fragments or read sequences. The read fragments in the files are mapped to the reference genome sequence. This process is called read mapping or read alignment. For read mapping, various tools can be used; for example, BWA [22], SOAP3 [14], Bowtie [21], and so on. The mapping results are written in SAM format [6], which is a generic format for storing large nucleotide sequence alignments and is widely used in genome analysis tools. Because the mapping results are unsorted and are often in multiple files, sorting and merging is required

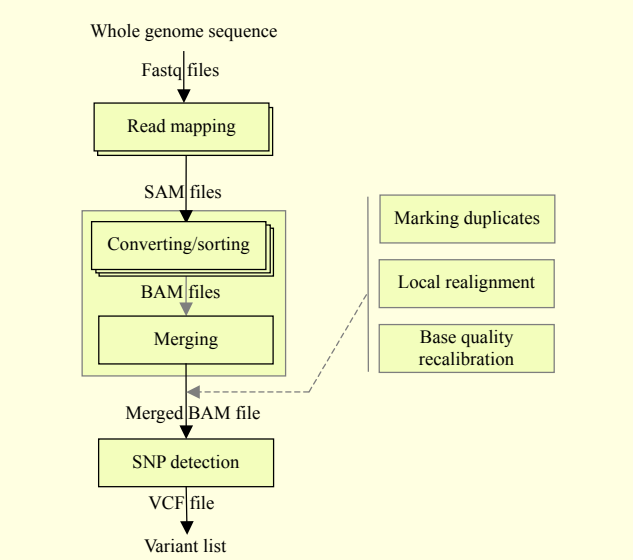

Fig. 1. SNP analysis pipeline.

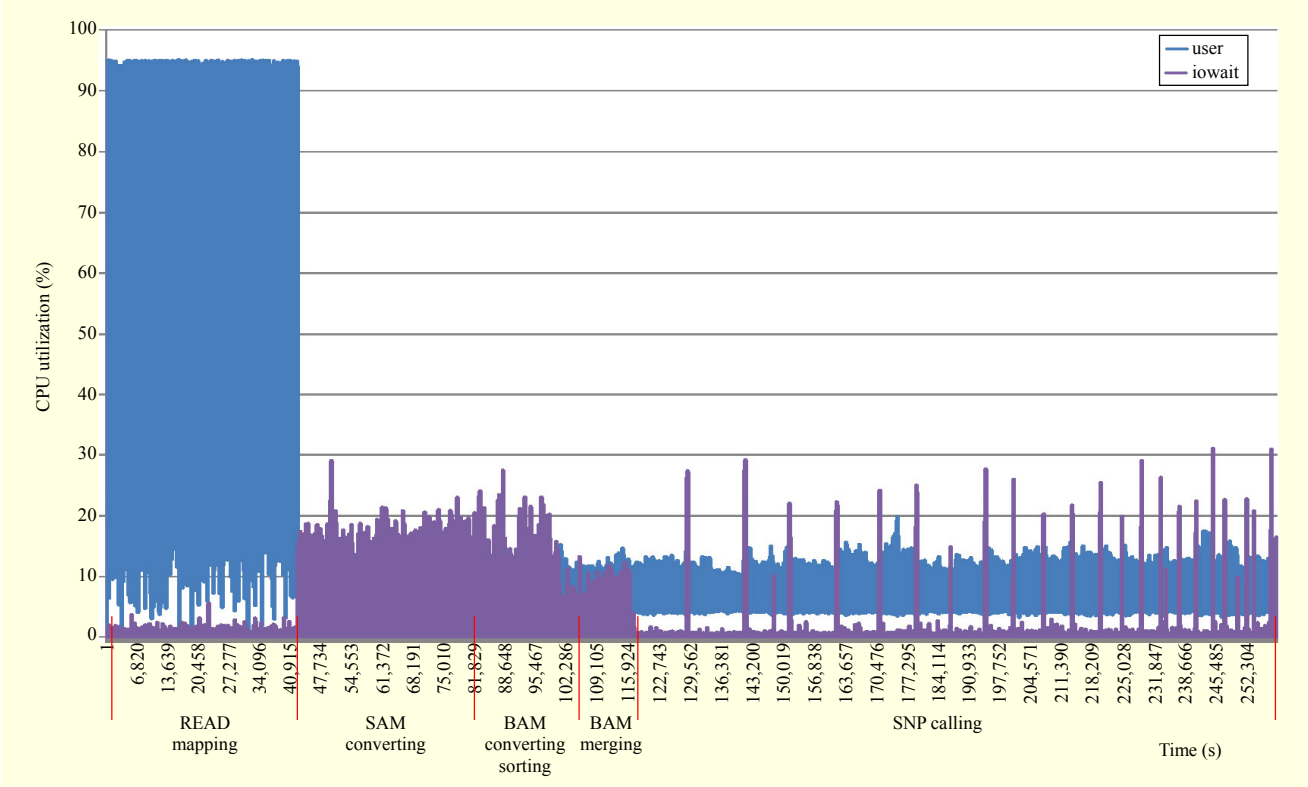

Fig. 2. Processor utilization patterns in variation analysis.

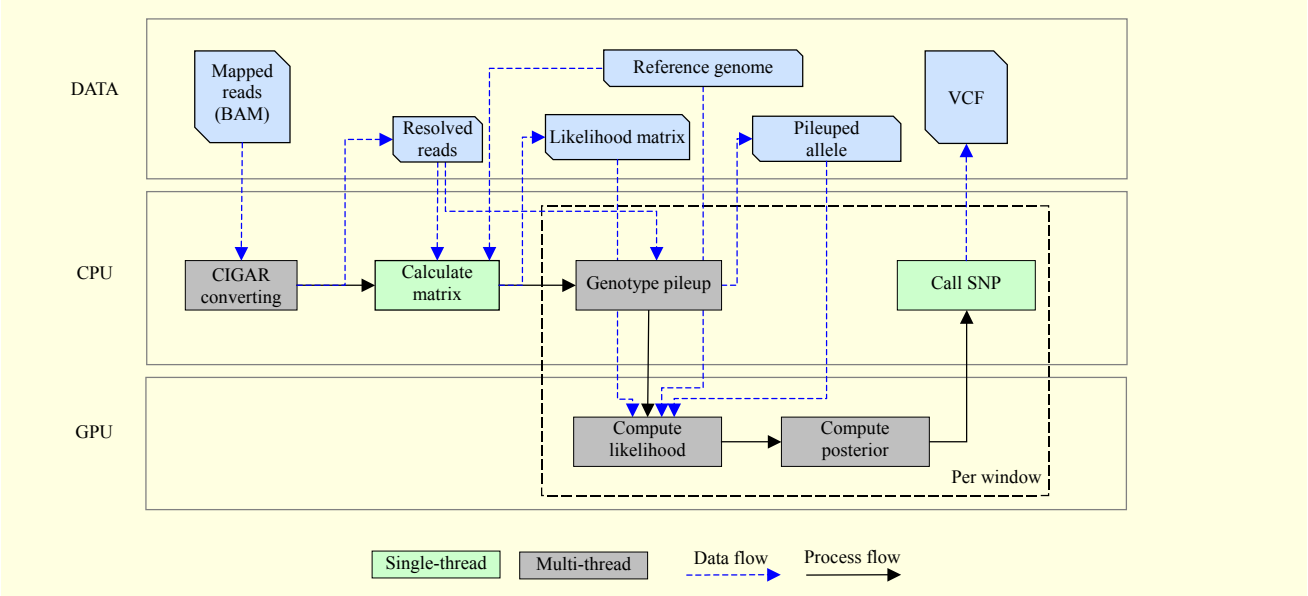

Fig. 3. Workflow of SNP detection.

after alignment. Sorting is based on the mapped position of each read fragment in the reference genome. Moreover, for efficiency, the SAM format is converted into the BAM format before sorting. SAMtools or Picard [23] can be used for the conversion, sorting, and merging processes. Finally, variants (that is, SNPs) are called in the SNP detection step (SNP detection is also called SNP calling). Additional steps such as realignment or quality recalibration can be positioned before the detection step to improve the accuracy of SNP calling.

Figure 2 shows the processor utilization pattern of the variant analysis pipeline. Here, the *x*-axis is the timeline and the *y*-axis is the percentage of CPU utilization. From Fig. 2, we can find several issues of the analysis pipeline. First, tasks in the pipeline have their own characteristics regarding the job intensities. For example, read mapping is computing-intensive because it spends a lot of time in finding the mapping position to the reference. However, format conversion is I/O intensive, as it spends a significant amount of time in the file I/O; that is, processes other than read mapping suffer from low processor utilization. Second, there is an I/O bottleneck between each step or within certain steps. Since the result of the previous step, or intermediate result, is delivered to the next process as a file, a massive file I/O is unavoidable. Thus, both issues need to be considered to improve the performance of the pipeline. In terms of intensity, SNP detection is computing-intensive rather than I/O intensive. However, previous systems mentioned in Section II show low processor utilization.

The issues regarding an I/O bottleneck in the conversion, sorting, and merging steps can be dealt with by removing temporal-sorted result files generated after or within the sorting step. Details of the modified sorting are beyond the scope of this paper. In this paper, we tackle the issue of low processor utilization in SNP detection using a window overlapping strategy. Details are given in the following section.

2. SNP Detection

Figure 3 illustrates the procedure of SNP detection. First, when the mapped data in the BAM files are loaded, the read is resolved by referencing the Compact Idiosyncratic Gapped Alignment Report (CIGAR) information. As shown in Fig. 4(a), all mapped data contains a read sequence (SEQ), aligned position (POS), mapping quality (MAPQ), CIGAR string (CIGAR), and base quality (QUAL). CIGAR describes the detail mapping state of bases in a read sequence. In Fig. 4(b), for example, if the sequence TTAGATAAAGGATACTG has CIGAR string 8M2I4M1D3M, then it denotes that "A" and "G" are newly inserted bases and "T" is a deleted base from the reference sequence, and the other bases are matched/mismatched bases. Based on the CIGAR string, the

QNAME	FLAG	RNAME	POS	MAPQ	CIGAR	RNEXT	PNEXT	TLEN	SEQ	QUAL
r001	163	ref	7	30	8M2I4M1D3M	=	37	39	TTAGATAAAGGATACTG	*

(a)

Reference: TTAGATAA GATAT CTG
Original read: TTAGATAA**AG**GATA CTG
Resolved read: TTAGATAA**I I**GATA**D**CTG

(b)

Fig. 4. CIGAR resolving example: (a) example of mapping information in SAM/BAM and (b) example of CIGAR resolving.

Table 1. CIGAR resolving rules.

CIGAR symbol	Action
I	Delete the base from the read
D	Substitute the base with 'D' (skipped in pileup stage)
S	Adjust position if it is the start of the read, otherwise skip
M (=, X)	Keep the base
Others	Skip (no action)

resolved read sequence is built during the CIGAR conversion step. The resolving rule of each CIGAR symbol is defined in Table 1.

Second, resolved reads are used to calculate the base likelihood matrix. We adopted the posterior probability model of SOAPsnp [7], which makes use of the base likelihood matrix. Each cell of the matrix contains a probability that represents the possibility of a base occurrence at a certain position in the read. The matrix has to be calculated on all read sequences before calculating the posterior probabilities of each genome site.

Third, the resolved read sequences are piled up. Here, a pileup means aligning the mapped read sequences in a memory space according to their mapped position in the reference sequence, as shown in Fig. 5. This makes it possible for the read bases in each position to be dealt with independently. In other words, the purpose of the pileup is to build a data structure that enables the parallel processing of the independent positions.

Fourth, for each genome site, we compute the likelihoods and then the posterior probabilities of the possible genotypes. Genotypes are generated from the base characters (that is, A, C, G, and T) by pairing two of them. As shown in the Bayesian probability model of SOAPsnp, the genotype likelihood of T_i in observed data D is represented as

$$P(T_i|D) = \frac{P(T_i)P(D|T_i)}{\sum_{x=1}^{S} P(T_x)P(D|T_x)}, \quad (1)$$

where S is the total number of genotypes. For example, $S = 4$ in the case of the human haploid genotype {A, C, G, T}. In (1), if the observed allele count at a certain locus is n, then $P(D|T)$ can be obtained from

$$P(D|T) = \prod_{k=1}^{n} P(d_k|T). \quad (2)$$

As the real DNA is diploid, the genotype probability $P(d_k|T)$ is calculated practically by the haploid probability; that is,

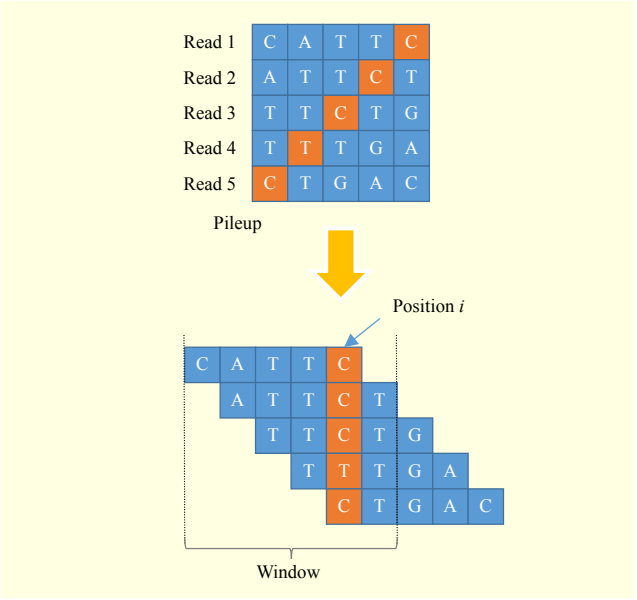

Fig. 5. Pileup example.

$$P(d_k|T) = \frac{P(d_k|H_m) + P(d_k|H_n)}{2}. \quad (3)$$

Using the base likelihood matrix, $P(d_k|H)$ is calculated as

$$P(d_k|H) = P((o_k, q_k, c_k)|H). \quad (4)$$

Finally, we decide whether a site is an SNP by comparing the genotype having the highest posterior probability with the corresponding base in the reference sequence. The detected SNP is written to a variant list file in VCF format.

As all positions in the reference genome cannot be processed at a time due to memory limitation, tasks in the dotted box of Fig. 3 are processed per window, which means the subsequence in the reference. The size of the window is influenced by the size of GPU memory, which is smaller than CPU. In the experiment, we set the size of the window as 32,768.

IV. Proposed Methods

As mentioned earlier, our purpose is to minimize the runtime of SNP detection by highly utilizing the computing resources. This pertains to parallel processing. Several tasks exist in an SNP detection that can be processed in parallel. That is, a bunch of genome sites, called a *window*, as shown in Fig. 5, can be dealt with in a batch. The tasks in gray rectangles in Fig. 3 are the candidates of a parallel execution. Naive parallelization does not always indicate a high utilization of the computing resources. To achieve our purpose, we analyzed the

ETRI Journal, Volume 37, Number 2, April 2015
http://dx.doi.org/10.4218/etrij.15.2314.0144

workloads and thereby presented new methods of utilizing heterogeneous computing resources.

1. Building Pileup and Calculating Posterior Probability

A. Workload Analysis

The CPU architecture provides relatively smaller numbers of cores than a GPU; thus it provides a low degree of parallelism. However, the dependency between threads is weak; therefore, threads with various types of jobs can be run separately. On the other hand, a GPU provides hundreds of cores and can run many threads in one clock cycle. Since a GPU can execute the same operation for all the threads, the performance is limited when the task has many divergences. To assign a proper job to the processors, the characteristic of the task has to be interpreted in detail.

As mentioned in Section II, the current GPU-based SNP detection utilizes only a GPU in a pileup and probability computation; thus, more CPU utilization is required to enhance the detection performance. To determine the appropriate jobs on a CPU and GPU, we analyzed the workload of the computation-related subtasks during the SNP detection procedure, as shown in the dotted box in Fig. 3. First, we measured the runtime of the *GPU_Pileup* method, whose subtasks, except for fetching the read data, are executed on a GPU. We then changed the pileup task into a CPU-runnable task and measured the runtime to investigate the possibility of a parallel execution with other tasks. In the *CPU_Pileup* method, pileup and fetch tasks are executed on the CPU, while the other tasks are executed on the GPU. Table 2 shows the measured runtimes of the detection subtasks on human chromosome 1. We ignored the runtime of the "call SNP" task in Fig. 3 since its runtime is trivial. The *MemCpy* column in Table 2 is the time used in copying data between the CPU and GPU memory. Except for the I/O related tasks, a *Pileup* task takes a remarkably large amount of time compared to *Likelihood* or *Posterior* tasks in GPU_Pileup. It is noticeable that the difference in runtime between the Pileup and Likelihood is reduced in the CPU_Pileup. From the analysis, it can be conjectured that if we partition the tasks into two groups (that is, Fetch and Pileup as one group and the remaining tasks in Table 2 as the other group), then the workloads of the two groups can then be balanced.

Figure 6 illustrates the occupancy percentage of runtime of three subgroups; that is, Fetch + Pileup, MemCpy + Likelihood, and Posterior. As shown in the figure, the runtime of Fetch + Pileup is similar to the sum of those of MemCpy + Likelihood and Posterior. In other words, the occupancy ratios are near 50% over the whole of the window cycles.

Table 2. Measured runtimes of subtasks in SNP detection on human chromosome 1.

	Fetch	Pileup	MemCpy	Likelihood	Posterior	Total
GPU_Pileup	155.6	88.9	100.7	9.97	32.95	388.12
CPU_Pileup	163.2	52.1	97.1	63.3	30.75	406.45

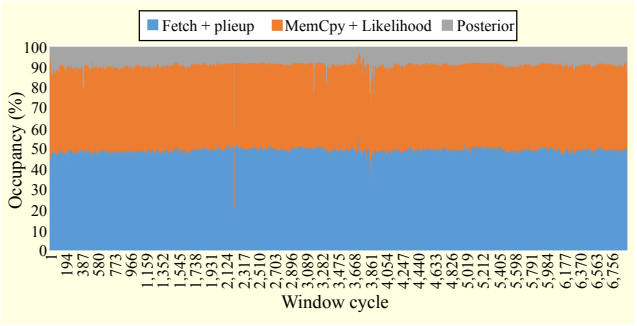

Fig. 6. Occupancy graph of subgroups of SNP detection tasks.

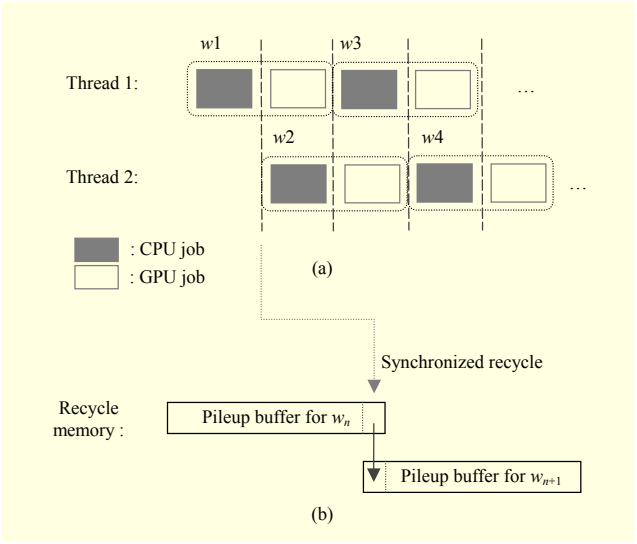

Fig. 7. Execution flow of overlapped window.

B. Overlapped Window Scheduling

Based on the observation, it is inferred that if we run the task of Fetch + Pileup and those of MemCpy + Likelihood and Posterior on different devices simultaneously (that is, on the CPU and GPU, respectively), we can obtain a high utilization of computing resources and thereby improve the runtime performance.

The orders of tasks in windows and execution orders of windows have to be maintained during executions. Let the CPU job be C and GPU job be G, and the workflow is then executed with the order of C_1, $<G_1, C_2>$, $<G_2, C_3>$, ... , $<G_{n-1}, C_n>$, G_n, which is also illustrated in Fig. 7(a). By following this

Algorithm. Window overlapped variation calling.

```
omp_set_num_thread(2);    // manage 2 overlapped window thread
call genotype()
{
    …
    init_shared_buffer();   // initialize shared variable buffer
    #pragma omp parallel  // invoke threads
    {
      omp_set_lock(lock_cpu);                          // lock CPU job
      for all reads in resolved_read_set,
      {
          fetch read from resolved_read_set;
          if read.position > window,
          {
              pileup(fetched_read_set);
              copy variables to shared buffer;
              omp_unset_lock(lock_cpu);   // unlock CPU job
              omp_set_lock(lock_gpu);      // lock GPU job

              copy variables from shared buffer;
              copy to device memory;
              likelihood();
              posterior();
              call_n_write();
              omp_unset_lock(lock_cpu);   // unlock GPU job
              omp_set_lock(lock_cpu);      // lock CPU job
          }
      }
    }
}
```

CPU job

GPU job

Fig. 8. Thread lock control for overlapped window.

scheduling scheme, all tasks and windows are kept in order. To make it possible to run threads concurrently, the data buffer of each thread has to be managed by a double-sized CPU buffer.

As shown in Fig. 7(b), a pileup fragment exists in a window. This is partial pileup data just after the end of window w_i. However, it actually belongs to window w_{i+1}. We have to deliver it from window w_i to window w_{i+1}. While it is delivered at the end of each window cycle in a non-overlapped method, the data are ready to be used by a thread for window w_{i+1} immediately after the pileup is ended by the other thread for window w_i.

The algorithm of the lock control to manage overlapped windows is described in Fig. 8. When the CPU job is locked for a thread, the other thread has to wait for the lock to be released. We use the same lock control for the GPU jobs. Before a thread changes the computing mode, all related data and variables have to be copied into shared buffers. The other thread loads the data before the execution of a new job.

2. CIGAR Conversion

A. Workload Analysis

A task with low processor utilization still exists in the SNP detection routine; that is, the CIGAR conversion. As the resolving is processed for all bases in each read sequence, a CIGAR conversion requites a lot of time. In the analysis of a CIGAR conversion for genome sequencing data (specifically, human chromosome 1), we found that CIGAR resolving occupies 29% of the total CIGAR conversion time, as shown in Table 3. As other tasks are regarding the file I/O, the resolving task has to be improved to reduce the runtime of the CIGAR conversion.

Table 3. CIGAR conversion time.

Steps	Load	Resolve	Write	Others	Total
Time (s)	157.3	82.3	43.1	3.9	286.6
Rate (%)	55	29	15	1	100

B. Multithreading in CIGAR Conversion

We implement a buffered multithreading in the CIGAR conversion. BAM-read sequences are pooled to a buffer and their CIGARs are then resolved simultaneously using multithreads. Because they are related to the file I/O, it is better to run them on the CPU rather than on the GPU.

V. Experiments and Results

1. Experimental Environments

Whole human genome data from the Personal Genome Institute, Republic of Korea [24] were used to evaluate the system performance. The sequence set is composed of 90 lengths of paired-end reads with a 32-fold depth coverage generated using an Illumina Solexa sequencing system. The total amount of data is 219 GB, which is divided into 14 files with the fastq [25] format. UCSC hg19 (NCBI version GRCHh37) is used as a reference sequence. The genetic variant data used for verifying the system are dbSNP 132.

SOAP3 aligner is used to map the 14 sequence files, and SAMtools is used to sort and merge the aligned results into one BAM file. The amount of the sorted BAM is 87 GB.

The configured system for testing has two 3.33 GHz Intel Xeon E5680 processors with a main memory of 24 GB. The GPU used in the system is an NVidia Tesla C2075, which has 1.15 GHz 448 cores and 6 GB of memory. E5680 provides six cores; thus, a maximum of 24 CPU threads can be utilized in the system by hyperthreading. OpenMP 3.0 is used to implement the CIGAR resolving and CPU_Pileup modules, and CUDA 4.0 is used to implement the GPU modules.

2. Detection Verification

The quality of the variant detection system can be evaluated using various metrics. In an evaluation by Isaac [26], several metrics were used, such as the call rate, transition-to-transversion ratio (Ts/Tv), heterozygous-to-homozygous

Table 4. Estimated verification metrics.

Filtering condition	Calling rate	Het/Hom	Ts/Tv	Novelty rate
q: 40, *d*: 10, *D*: 80	95.05	1.76	1.90	6.92

variant ratio (Het/Hom), and percent of called SNPs not found in dbSNP (novelty rate). These are metrics acquired by analyzing real biological data statistically. We measured these metrics to evaluate the quality of our variant detection system.

Before evaluating the metrics, we added filtering options regarding the quality and depth to filter the variant results to the system. Here, *q* is the minimum quality score, *d* is the minimum depth, and *D* is the maximum depth. The depth indicates the number of pileup bases in a site. By allowing the filtering option to be set, the user can control the output result for the purpose of their experiment.

From the whole genome data and dbSNP under the condition of "*q*:40, *d*:10, and *D*:80," we obtained the results for each metric, as shown in Table 4. The detection rate, Het/Hom, and Ts/Tv of the system show consistent values compared to those of Isaac's experiment. However, the novelty rate of the system is slightly higher than that of Isaac's, which means that the system called more variants as novel SNPs. This situation can occur owing to the different data used in the experiment. Except for the novelty rate, it can be interpreted that the measured results are within an acceptable range.

3. Runtime Evaluation

To determine the optimal numbers of threads in the CIGAR resolving, we measured the elapsed resolving time of chromosome 1 data while changing the number of threads. The resolving graph in Fig. 9 shows the elapsed resolving time of chromosome 1 data. The BAM size of chromosome 1 is about 7 GB. Compared to the runtime by a single thread, a runtime by 22 threads is decreased with a ratio of 9%. These 22 threads are all the available threads since one core (corresponding to two threads) needs to be assigned to the operating system. It can be observed in the graph that the speedup is slowed down when the number of threads exceeds 10.

Similarly, the runtime of CPU_Pileup shows the best result when all available threads are used in the calculation, and the performance efficiency is reasonable when the number of threads is between 10 and 15. From the result, it is assumed that by using these numbers of threads an efficient performance in utilizing the CPU threads can be acquired. Thus, we selected 15 as the number of threads for the resolving and CPU_Pileup.

To evaluate the performance of the proposed overlapped window method, we measured the runtimes of three different

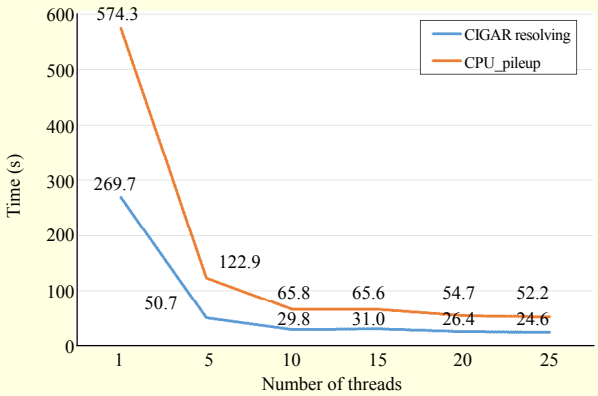

Fig. 9. Estimated times with changes in thread size.

Table 5. Comparison of SNP detection time.

Time (sec)	GPU_Pileup	CPU_Pileup	CPU + GPU
CIGAR converting (resolving)	256.13 (22.26)	261.29 (24.57)	257.3 (23.81)
Fetch	155.64	163.3	149.07*
Pileup	88.96	52.16	56.01*
MemCpy	128.51	116.06	119.55^{+}
Likelihood	9.97	63.35	65.21^{+}
Posterior	32.95	30.75	31.6^{+}
Others	158.15	229.04	—
Total	830.31	915.95	801.47

Note: * is run on GPU and $^{+}$ is run on CPU

Table 6. Comparison with other SNP detection SW.

	GPU + CPU	Samtools	GATK
Time (min)	118	1,125	662

methods, GPU_Pileup, CPU_Pileup, and CPU + GPU, on human chromosome 1 data. Table 5 shows the results of each method. Compared to CPU_Pileup, the runtime of CPU + GPU using an overlapped window is reduced to about 114.48 s, and the improvement is 28.84 s when compared to GPU_Pileup.

It can be seen that the CPU tasks take 215.45 s and the GPU tasks take 222.02 s in the overlapped window method, which means that the workload balance is reasonable during the execution. As the CIGAR conversion requires about 32% of the total time, the expected improvement by the overlapped window method can have a limited effect.

It can be seen in Table 5 that the portion of CIGAR resolving time in the CIGAR conversion is lowered to less than 10%,

which was 29% before applying the multithreading in Table 3. The optimization of the CIGAR conversion is proved to be effective.

We also measured other open source SNP detection applications to compare the performance of our system. For a comparison in a real field situation, we used whole human genome sequencing data. Table 6 shows the runtime of the three types of SW. The performance of our system is evaluated to be 9.5-times faster than SAMtools mpileup and 5.6-times faster than GATK's UnifiedGenotyper.

VI. Conclusion

In this paper, methods for enhancing the utilization of heterogeneous computing resources were presented to speed up the SNP detection process. From the analysis of workloads in the detection procedure, we divided the tasks into task groups suitable for different computing resources. By scheduling these task groups based on a window overlapping method, it became possible to run the task groups concurrently. In the experiment, we found that the presented methods achieved up to 9.5-times the speedup as compared to previous open source applications and a 12.5% speedup compared to a non-overlapped method. It is expected that the proposed system can be utilized to analyze huge amounts of sequencing data, such as whole human genome sequencing data.

Through the parallelization of the analysis components, we tried to relieve the bottleneck occurring in the analysis pipeline. However, the I/O bottleneck caused by temporary files between each analysis step still remains a problem. Further approaches have to be tackled to reduce this problem. One suggestion is to build a pipeline with multiple parallelizable streams of analysis tools, each of which processes such a small dataset that the result can be delivered in memory, without file writing. This might sacrifice the flexibility of the pipeline organization, but the performance will be improved.

References

[1] M. Metzker, "Sequencing Technologies — the Next Generation," *Nature Rev. Genetics*, vol. 11, Jan. 2010, pp. 31–46.

[2] R.M. Durbin et al., "A Map of Human Genome Variation from Population-Scale Sequencing," *Nature 467*, Oct. 2010, pp. 1061–1073.

[3] F.S. Collins and A.D. Barker, "Mapping the Cancer Genome," *Sci. American 296*, Mar. 2007, pp. 50–57.

[4] O. Harismendy et al., "Evaluation of Next-Generation Sequencing Platforms for Population Targeted Sequencing Studies," *Genome Biol.*, vol. 10, Mar. 2009, pp. R32–R32.13.

[5] J. Wang et al., "The Diploid Genome Sequence of an Asian Individual," *Nature 456*, Nov. 6, 2009, pp. 60–65.

[6] H. Li et al., "The Sequence Alignment/Map (SAM) Format and SAMtools," *Bioinformat.*, vol. 25, no. 16, 2009, pp. 2078–2079.

[7] R. Li et al., "SNP Detection for Massively Parallel Whole-Genome Resequencing," *Genome Res.*, vol. 19, May 2009, pp. 1124–1132.

[8] A. McKenna et al., "The Genome Analysis Toolkit: A Mapreduce Framework for Analyzing Next-Generation DNA Sequencing Data," *Genome Res.*, vol. 20, July 2010, pp. 1297–1303.

[9] M.A. DePristo et al., "A Framework for Variation Discovery and Genotyping Using Next-Generation DNA Sequencing Data," *Nature Genetics*, vol. 10, Apr. 2011, pp. 491–498.

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, no. 51, no. 1, Jan. 2008, pp. 107–113.

[11] D.-H. Ko et al., "Construction and Rendering of Trimmed Blending Surfaces with Sharp Features on a GPU," *ETRI J.*, vol. 33, no. 1, Feb. 2011, pp. 89–98.

[12] S. Kim, M.-H. Kyung, and J.-H. Lee, "Relighting 3D Scenes with a Continuously Moving Camera," *ETRI J.*, vol. 31, no. 4, Aug. 2009, pp. 429–437.

[13] C. Angermüller, A. Biegert, and J. Soding, "Discriminative Modelling of Context-Specific Amino Acid Substitution Probabilities," *Bioinformat.*, vol. 28, Oct. 2012, pp. 3240–3247.

[14] C.-M. Liu et al., "SOAP3: Ultra-Fast GPU-Based Parallel Alignment Tool for Short Reads," *Bioinformat.*, vol. 28, no. 6, Jan. 2012, pp. 878–879.

[15] A.W. Goetz et al., "Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs - Part I: Generalized Born," *J. Chem. Theory Comput.*, vol. 8, no. 5, Mar. 2012, pp. 1542–1555.

[16] G. Guo et al., "GPU-Accelerated Adaptive Compression Framework for Genomics Data," *IEEE Int. Conf. Big Data*, Silicon Valley, CA, USA, Oct. 6–9, 2013, pp. 181–186.

[17] D.J. Hedges et al., "Exome Sequencing of a Multigenerational Human Pedigree," *PLoS ONE*, vol. 4, no. 12, Dec. 2009, e8232.

[18] S.T. Sherry et al., "dbSNP: The NCBI Database of Genetic Variation," *Nucleic Acid Res.*, vol. 29, no. 1, 2001, pp. 308–311.

[19] M. Lu et al., "GSNP: A DNA Single-Nucleotide Polymorphism Detection System with GPU Acceleration," *Int. Conf. Parallel Process.*, Taipei, Taiwan, Sept. 13–16, 2011, pp. 592–601.

[20] B. Langmead et al., "Searching for SNPs with Cloud Computing," *Genome Biol.*, vol. 10, Nov. 2009, R134.

[21] B. Langmead et al., "Ultrafast and Memory-Efficient Alignment of Short DNA Sequences to the Human Genome," *Genome Biol.*, vol. 10, Mar. 2009, R25.

[22] H. Li and R. Durbin, "Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform," *Bioinformat.*, vol. 25, no. 14, May 2009, pp. 1754–1760.

[23] *Picard Project.* Accessed June 16, 2014.

http://hpicard.sourceforge. net
[24] *Personal Genome Institute*. Accessed July 4, 2014. http://pgi.re.kr
[25] P.J.A. Cock et al., "The Sanger FASTQ File Format for Sequences with Quality Scores, and the Solexa/Illumina FASTQ Variants," *Nucletic Acids Res*., vol. 38, no. 6, 2010, pp. 1767–1771.
[26] *Fast, Accurate and Easy Alignment and Variant Calling with Isaac Genome Alignment Software and Isaac Variant Caller*, Illumina Inc. Accessed July 4, 2014. http://res.illumina.com/documents/products/hitepapers/whitepaper_iassc_workflow.pdf

Myungeun Lim received his BE and MS degrees in computer engineering from Dongguk University, Seoul, Rep. of Korea, in 1999 and 2001, respectively. He is currently a senior researcher at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His research interests include bioinformatics, data mining, and high-performance computing.

Minho Kim received his BS degree in electronics engineering from Korea University, Seoul, Rep. of Korea, in 1997. He received his MS and PhD degrees in information and communications engineering from Gwangju Institute of Science and Technology, Rep. of Korea, in 1999 and 2006, respectively. He is currently a senior researcher at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His research interests include bioinformatics, algorithms, and high-performance computing.

Ho-Youl Jung received his BS, MS, and PhD degrees in computer science from Pusan National University, Rep. of Korea, in 1997, 1999, and 2002, respectively. From 2002 to 2004, he worked as a principal researcher for the National Genome Research Institute of Korea Centers for Disease Control and Prevention, Seoul, Rep. of Korea. He is currently a principal researcher at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His research interests include bioinformatics, graph theory, and computational geometry.

Dae-Hee Kim received his BS and MS degrees in electrical engineering from Inha University, Incheon, Rep. of Korea, in 1996 and 1998, respectively. Since he joined the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, in 2001, he has been a senior member of the engineering staff. His research interests include image processing, health care, human–computer interaction, and bioinformatics.

Jae-Hun Choi received his BS, MS, and PhD degrees in computer science from Chonbuk National University, Jeonju, Rep. of Korea, in 1994, 1996, and 2000, respectively. He is currently a principal researcher and a section manager at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His research interests include big data, data mining, ontology, and bio-medical informatics.

Wan Choi received his BS degree in electronic engineering from Kyungpook National University, Daegu, Rep. of Korea, in 1991. He received his MS degree in computer science from the Korea Advanced Institute of Science and Technology, Daejeon, Rep. of Korea, in 1985. He joined the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea, in 1985 and is currently both a principal researcher and a department manager. His research interests include cloud computing, video-service platforms, and high-performance computing.

Kyu-Chul Lee received his BE, ME, and PhD degrees in computer engineering from Seoul National University, Rep. of Korea, in 1984, 1986, and 1996, respectively. In 1994, he worked as a visiting researcher at the IBM Almaden Research Center, San Jose, CA, USA. From 1995 to 1996, he worked as a visiting professor at the CASE Center at Syracuse University, NY, USA. He is currently a professor with the Department of Computer Engineering, Chungnam National University, Daejeon, Rep. of Korea. His current areas of research interest include multimedia database systems, hypermedia systems, object-oriented systems, and digital libraries. He has authored over 100 technical articles published in various journals and conferences. He is a member of ACM, the IEEE Computer Society, and the Korea Information Science Society.