GPU-Accelerated GATK HaplotypeCaller with Load-Balanced Multi-Process Optimization

Shanshan Ren, Koen Bertels, Zaid Al-Ars
Computer Engineering Lab
Delft University of Technology
2628CD Delft,The Netherlands
{s.ren, k.l.m.bertels, z.al-ars}@tudelft.nl

***Abstract*—Due to its high-throughput and low cost, Next Generation Sequencing (NGS) technology is becoming increasingly popular in many genomics research labs. However, handling the massive raw data generated by the NGS platforms poses a significant computational challenge to genomics analysis tools. This paper presents a GPU acceleration of the GATK HaplotypeCaller (GATK HC), a widely used DNA variant caller in the clinic. Moreover, this paper proposes a load-balanced multi-process optimization of GATK HaplotypeCaller to address its implementation limitation which forces the sequential execution of the program and prevents effective utilization of hardware acceleration. In single-threaded mode, the GPU-based GATK HC is 1.71x and 1.21x faster than the baseline HC implementation and the vectorized GATK HC implementation, respectively. Moreover, the GPU-based implementation achieves up to 2.04x and 1.40x speedup in load-balanced multi-process mode over the baseline implementation and the vectorized GATK HC implementation in non-load-balanced multi-process mode, respectively.**

***Index Terms*—GPU acceleration; GATK HaplotypeCaller; multi-process; pair-HMMs forward algorithm;**

I. Introduction

Next Generation Sequencing (NGS) [1] technology makes DNA sequencing more affordable and accessible than ever before. DNA sequencing is essential for a deep understanding of human genetics and is considered as an enabler on personalized medicine. Many applications for DNA sequence analysis have been developing at a fast rate in the last decade, such as sequence alignment, genome de-novo assembly, variant calling, haplotype phasing and so on.

Variant calling is a crucial step for DNA sequence analysis, which is used to find the positions where a given patient DNA sequence is different from a reference genome in order to detect DNA variants. These variants include SNVs (single nucleotide variations), small insertions/deletions (INDELs) and structural variations (SVs). Many tools (called variant callers) have been proposed to detect variants in order to be used in practice to diagnose genetic disease, for example.

Early variant callers, such as the GATK UnifiedGenotyper [2], SAMtools [3] and VarScan2 [4], detect variants at different positions in isolation. These tools are very effective in detecting SNVs, but are lacking when it comes to the accuracy of identifying INDELs and SVs.

More recent haplotype-based callers, such as the GATK HaplotypeCaller [5], Platypus [6] and freebayes [7], have improved the accuracy of detecting INDELs. INDELs are easily misaligned when mapping the patient DNA to a reference genome, which is ahead of variant calling. In order to correctly identify INDELs, haplotype-based variant callers add extra steps, such as local de-novo assembly of haplotypes [5][6] or direct detection of haplotypes [7]. Moreover, haplotype-based callers enhance the accuracy of identifying SNVs by making use of linkage disequilibrium between nearby variants. The GATK UnifiedGenotyper, for example, is less effective than the GATK HaplotypeCaller in detecting INDELs. However, this comes at the cost of higher execution time.

The GATK HaplotypeCaller (or GATK HC) is widely used in many large-scale sequencing projects. However, GATK HC suffers from long execution time, which would limit its feasibility in many situation. In this paper, we investigate and propose the first GPU-accelerated version GATK HC to improve its performance. Regarding to the optimization of GATK HC, Intel processors and IBM POWER processors both exploit vector instructions to speed up the pairwise alignment kernel [8][9], which is the most time consuming part of GATK HC. There are also a couple of publications on FPGA-based and GPU-based hardware acceleration of the pairwise alignment kernel of GATK HC, but they do not discuss the acceleration of the overall application [10][11].

In this paper, we present an efficient GPU-accelerated implementation of GATK HC and evaluate its effectiveness. An important component of the work is integrating the GPU acceleration into the Java-based GATK HC code and minimizing the incurred overhead. Compared with the baseline implementation, it achieved 1.71x speedup in single-threaded mode. Moreover, we found an important limitation in the GATK HaplotypeCaller implementation which forces the sequential execution of the program and prevents effective utilization of the accelerated part. A load-balanced multi-process optimization is proposed to overcome this limitation, which makes the GPU-based implementation up to 2.04x and 1.40x faster than the baseline implementation and the vectorized implementation, respectively.

The rest of this paper is organized as follows. Section II presents a brief overview of GATK HC. Section III presents the details of the GPU-accelerated implementation of GATK HC and the load-balanced multi-process optimization. Section IV presents the experimental results along with analyses. Section V concludes this paper.

II. Background

A. GATK HaplotypeCaller

GATK HC is a Java-based DNA variant caller that is widely used in practice. It is divided into the following four main steps [12].

(i) **Define active regions**—Active regions are determined based on the presence of significant evidence for variation. The following steps only operate on the active regions and ignore the inactive regions.

(ii) **Determine haplotypes**—For each active region, a de Bruijn-like graph is built to reassemble the active region and a list of haplotypes is determined based on the graph. Here, haplotype is a sequence covering the entire length of an active region. Then each haplotype is realigned against the reference sequence using the Smith-Waterman algorithm in order to identify potentially variant sites.

(iii) **Determine likelihoods of the haplotypes**—For each active region, a pairwise alignment of each read against each haplotype is performed using the pair-HMMs forward algorithm, which produces a matrix of likelihoods of haplotypes given the reads.

(iv) **Assign genotypes**—For each potential variant site, Bayes' rule is applied to calculate the likelihoods of each genotype using the likelihoods of haplotypes given the reads. The genotype with the largest likelihoods is selected.

GATK HC can run in single-threaded and multi-threaded mode, as shown in Figure 1. When GATK HC runs in single-threaded mode (Figure 1(a)), it first defines active regions (Step(i) in the list above). If there is an active region, it executes Step(ii), Step(iii) and Step(iv) in succession and jumps back to Step(i). If there are no more active regions in Step(i), GATK HC ends execution.

When GATK HC runs in multi-threaded mode (Figure 1(b)), each thread executes in the same way as in single-threaded mode. However, since the size of the active region is not known in advance, Step(i) has to first complete calculating the current active region before the next active region can be calculated. This leads to only one thread executing Step(i) at any time.

In order to investigate which of these steps is most time-consuming and which one is most suitable for GPU-based acceleration, we analyzed and profiled GATK HC (GATK version 3.7) with a typical workload (chromosome 10 of the whole human genome dataset G15512.HCC1954.1).

Firstly, GATK HC was executed in single-threaded mode in order to find which step is most time-consuming. The profiling results in single-threaded mode are shown in Table I. The relative execution time and type of processing are specified in the table as well. As shown in Table I, Step(iii) is most time-consuming, which consumes 48.5% of the total execution time. The main operation of Step(iii) is pairwise alignments implemented by the pair-HMMs forward algorithm, which is executed millions of times. Therefore, acceleration of the pair-HMMs forward algorithm is very important to improve the performance of GATK HC in single-threaded mode.

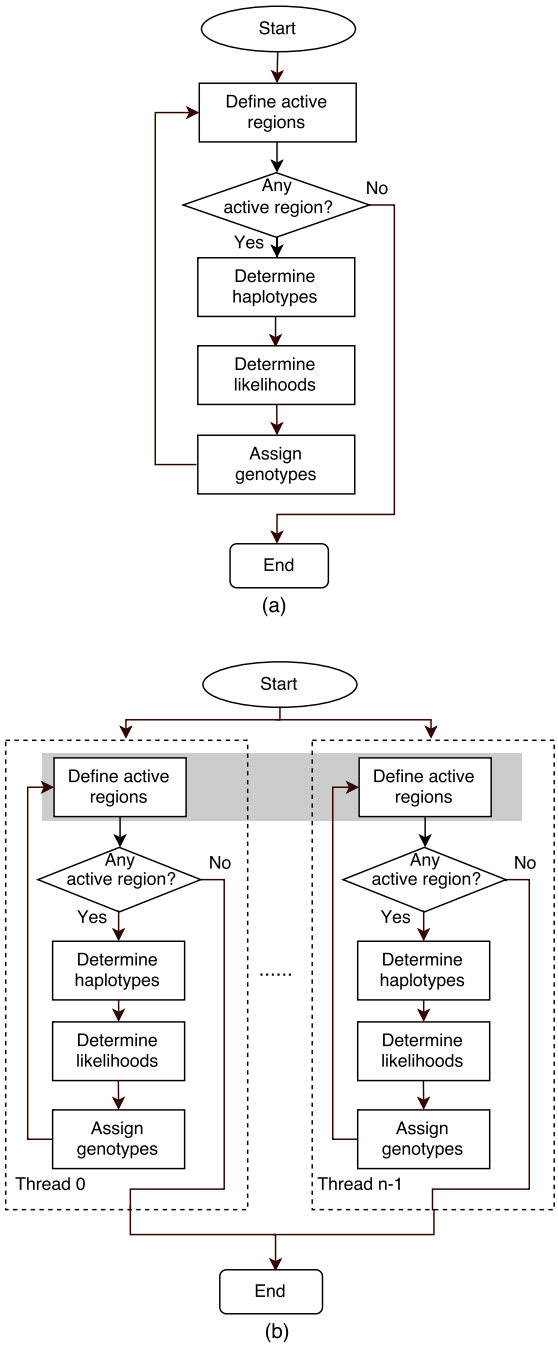

Fig. 1. GATK HC workflow in (a) single-threaded mode and (b) multi-threaded mode

GATK HC then was executed in multi-threaded mode. The total execution time with different number of cores was recorded, which is shown in Figure 2. Moreover, in order to find the influence of the data dependency of Step(i) across multiple threads, we modified GATK HC by disabling the execution of the other steps and made it only executed Step(i).

TABLE I
PROFILING RESULTS OF GATK HAPLOTYPECALLER

Steps	Time	Processing
Define active regions	15.5%	Sequential
Determine haplotypes	34.0%	Sequential
Determine likelihoods	48.5%	Parallel
Assign genotypes	1.3%	Parallel
Other	0.7%	

The total execution time of the modified GATK HC running on 20 cores is 1164 seconds, while the total execution time of original GATK HC running on 20 cores is 1249 seconds. This indicates that when the number of cores is big enough, GATK HC execution time becomes bottlenecked by Step(i) in multi-threaded mode.

In practice, GATK HC is usually executed in multi-process mode instead of in multi-threaded mode to overcome the data dependencies in Step(i). In multi-process mode, the input file is split into chunks based on genome regions and each chunk is processed by GATK HC independently. Although there may be some accuracy loss at the boundaries between consecutive chunks within the same chromosome, the accuracy loss can be negligible in many cases.

The simplest way to divide the input file is to split the genome regions into multiple intervals of equal length. [13][14] proposes to split the genome regions based on the number of reads mapped to each regions, taking the read coverage into consideration. However, both methods do not results in a balanced division of the input file in the case of GATK HC. In this paper, we present a load-balanced multi-process optimization to minimize the total execution time.

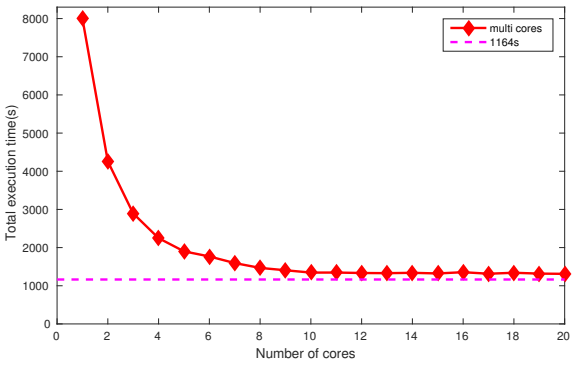

Fig. 2. Multi-threaded execution time of GATK HC

B. Pair-HMMs forward algorithm

In GATK HC, the pair-HMMs forward algorithm takes a read and a haplotype as the input and calculates the overall alignment probability. Previous research on increasing the speed of the pair-HMMs forward algorithm can be found in [8][9][10][11], most of which exploit the inherent parallelism of the algorithm. Intel and IBM researchers employ vector instructions on their respective processors [8][9] to reduce the execution time. These vectorization approaches have been implemented and can be integrated into GATK HC easily. The authors claim a dramatic improvement of the performance of single-threaded GATK HC.

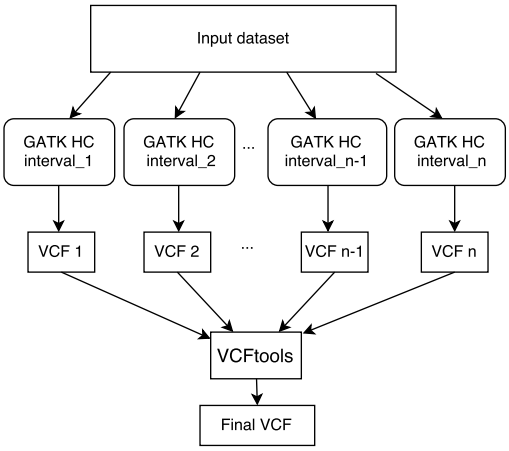

Fig. 3. Data flow of the multi-process GATK HC

On the other hand, [10][11][15] propose FPGA-based implementations of the pair-HMMs forward algorithm. [10] utilizes a systolic array to map the algorithm on FPGAs, while [11] proposes pipelined processing elements within a systolic array and [15] reduces the overhead in the systolic array. However, these implementations have not been integrated into GATK HC.

In addition, [16] proposes several GPU-based implementations of the pair-HMMs forward algorithm and compares these implementations using datasets with different number of read-haplotype pairs. It investigates two different acceleration approaches: the inter-task and intra-task parallelization. According to the paper, when the number of read-haplotype pairs is small, the intra-task GPU-based implementation outperforms all other investigated implementations.

III. METHODS

Our efforts to improve the performance of GATK HC can be divided into two aspects: (1) a load-balanced multi-process parallelization approach to reduce the sequential execution of Step(i) and (2) integration of GPU acceleration of the pair-HMMs forward algorithm into the multi-process GATK HC.

A. Load-balanced multi-process optimization

Figure 3 shows the data flow of the multi-process GATK HC. The GATK HC argument $-L$ is used to confine processing to a specific genome interval instead of actually dividing the input file into small parts. Each interval is processed individually by a GATK HC instance. The output of these GATK HC instances, represented by VCF files, are combined by VCFtools [17] into one VCF file.

As mentioned in Section II-A, Step(ii), Step(iii) and Step(iv) operate only on active regions. If each genome interval has the same number of active regions, this will most probably result in a load-balanced multi-process implementation. Since the active regions are determined by the presence of significant evidence of variation in Step(i), the number of variants in each genome interval is the key parameter to ensure load-balancing.

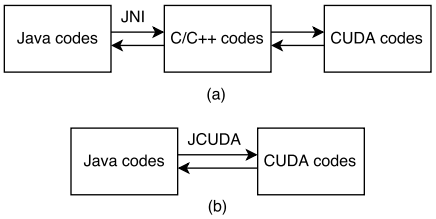

Fig. 4. Two methods of calling CUDA programming modules from Java code

When running GATK HC, the argument $-D$ and a dbsnp file is usually used in order to annotate variants found by GATK HC with the corresponding reference ID. Since the dbsnp file includes all known variants and the positions of these variants on the genome, we can use this file to divide the genome into regions based on the number of known variants. Although GATK HC might find novel variants not present in the dbsnp file, the number of these variants is relatively small and would have a negligible influence on the genome region division.

In order to divide the genome region using the dbsnp file, we modified BCFtools, which uses the HTSlib library to realize fast accesses of the dbsnp file. The modification of BCFtools is to add a function in the vcffilter.c file. The new function first calculates the total number of variants and then outputs the start and end positions of each genome interval, which has the same number of variants.

Besides the number of variants, the number of reads on each genome interval may also influence the load-balancing. One solution is to take these two factors together into consideration. However, it turns out that calculating the total number of reads in the input file is very time consuming. Moreover, for each new input file, the calculation of the total number of reads has to be done again. Thus, the genome region is divided only based on the variant numbers in the dbsnp file.

B. Application level GPU acceleration

In GATK HC, the number of read-haplotype pairs processed by the pair-HMMs forward algorithm depends on the number of the reads and haplotypes found in each active region. For example, the number of read-haplotype pairs ranges from 4 to 38912 for each active region in chromosome 10 of the whole human genome dataset G15512.HCCI954.1. According to [16], for this type of dataset, the intra-task GPU-based implementation of the pair-HMMs forward algorithm is the most effective GPU-based implementation that gives the highest performance.

The GPU acceleration of the pair-HMMs forward algorithm is implemented using CUDA. Although CUDA is able to support various programming languages, it does not support Java, which is the programming language of GATK HC. One method to call CUDA code from Java is using JNI (Java Native Interface), which enables Java code to call modules written in programming languages such as C and C++. As shown by Figure 4(a), this is done by first using JNI to call C++ code from Java, which in turn calls CUDA code.

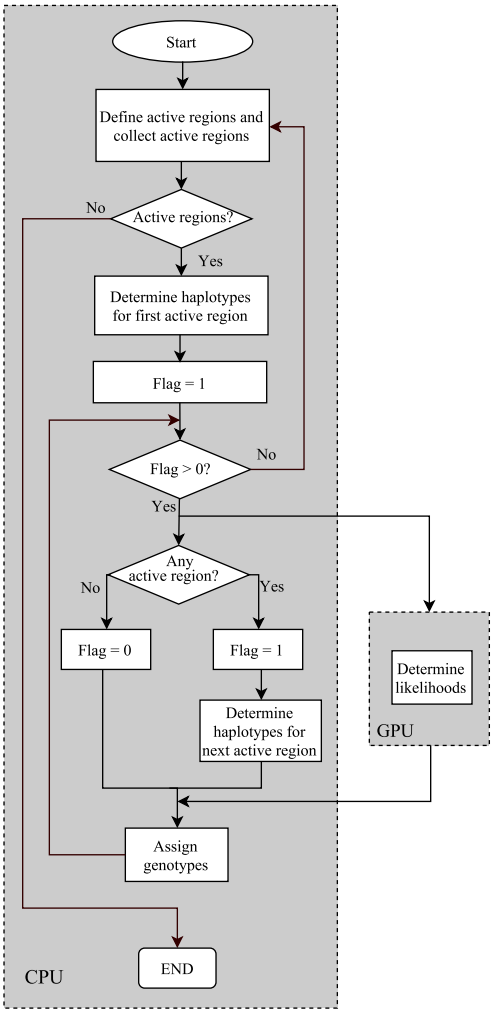

Fig. 5. GATK HC new workflow in single-threaded mode

Another method is to apply JCuda[18], which supplies direct accesses of the CUDA code from Java code. In essence, the design of JCuda also uses JNI to call C++ code and lets C++ code call CUDA programming modules. However, JCuda hides these details from the user. Thus, we can directly call CUDA programming modules from Java codes, as shown by Figure 4(b). For GATK HC, JCuda is employed to call CUDA code from Java.

As shown by Figure 1(a) and (b), the last three steps of the workflow (Determine haplotypes, Determine likelihoods and Assign genotypes) are executed sequentially for each active region. This prevents hiding the execution time of the GPU accelerated part. If the Java implementation of the pair-HMMs forward algorithm is replaced by the GPU implementation, the CPU would be idle and wait for the GPU results. In order to allow hiding GPU execution time, we need to modify the workflow in order to make GPU and CPU run in parallel.

Figure 5 is the new workflow of GATK HC in single-threaded mode. GATK HC first produces and collects multiple active regions and passes these active regions to subsequent steps. Then, Determine haplotypes is executed for the first

active region and the results are transferred to the GPU. Next, Determine likelihoods for the first active region is executed on GPU while Determine haplotypes for the second active region is executed on CPU simultaneously. The rest of the active regions are processed in the same manner. In this way, Determine likelihoods on GPU and Determine haplotypes on CPU are executed simultaneously.

With regard to multi-threaded mode, GATK HC first produces multiple active region, which is still sequentially executed. The other steps of multi-threaded mode are modified in the same way as the steps of single-threaded mode in Figure 5. For each thread, Determine haplotypes on CPU and Determine likelihoods on GPU are executed simultaneously.

IV. Results and Discussion

A. Experimental Setup

In this paper, we use an IBM Power System S824L (82478-42L) to perform experiments and measure performance results. This system includes two IBM Power8 processors (10 cores each) running at 3.42 GHz, 256 GB of DDR3 memory, and an NVIDIA Tesla K40 card. The NVIDIA Tesla K40 card has 2880 cores running at up to 745 MHz, with CUDA compute capability 3.5.

This paper uses GATK version 3.7 for the analysis, which is the latest version of GATK at the time of writing. The paper compare the performance of three GATK HC implementations: (1) the baseline GATK HC with the pair-HMMs forward algorithm implemented in Java, which is downloaded from the GATK website; (2) GATK HC with the pair-HMMs forward algorithm optimized using vector instructions, the library of which is implemented by IBM research [9]; (3) GATK HC with the pair-HMMs forward algorithm accelerated on GPU. The dataset used for the measurement is chromosome 10 of the whole human genome dataset G15512.HCC1954.1.

B. Single-threaded

Table II shows the results of the GATK HC implementations in single-threaded mode. As shown by Table II, the baseline implementation took the longest time (8034.05 seconds). The vectorized GATK HC is 1.42x faster than the baseline implementation. The GPU-based GATK HC is 1.71x and 1.21x faster than the baseline implementation and the vectorized GATK HC implementation, respectively.

TABLE II
Results of GATK HC implementations in single-threaded mode.

GATK HC	Total time [s]	Time pair-HMMs [s]	Speedup
Baseline	8034.05	3676.12	—
Vectorized	5655.96	1289.62	1.42x
GPU-based	4687.08	hidden	1.71x

C. Multi-threaded

Figure 6 shows results of the three GATK HC implementations in multi-threaded mode with thread number ranging from 2 to 20.

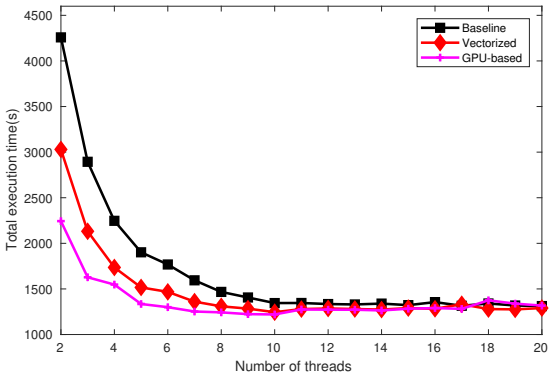

Fig. 6. Total execution time of GATK HC in multi-threaded mode

The execution time of all three implementations decreases while the thread number increases. This is the benefit of using multi-thread, which makes Step(ii) Determine haplotypes, Step(iii) Determine likelihoods and Step(iv) Assign genotypes be executed in parallel.

When the thread number is small, the execution time of the GPU-based implementation is the lowest. However, the execution time of the three implementations becomes very similar when the number of threads becomes bigger than 10. This means that the acceleration of the pair-HMMs forward algorithm does not have big effects on the total execution time when the number of threads is big enough.

D. Multi-process

The three implementations were executed with different number of processes from 2 to 20 both in the load-balanced and non-load-balanced multi-process mode. In the load-balanced multi-process mode, the input file is divided according to the method proposed in Section III-A; in the non-load-balanced multi-process mode, the input file is divided simply using equal number of bases in each segment of the genome.

In order to verify the accuracy of GATK HC in multi-process mode, VCFtools is used to merge the output files produced by the GATK HC instances into one file and compare this file with the output file produced by the baseline implementation in single-threaded mode. The comparison results show that there is no accuracy loss for all the multi-process GATK HC experiments with process number from 2 to 20. For other input file, there might be some accuracy loss.

Generally, the execution time of each GATK HC instance in multi-process mode is not the same. Therefore, the maximal execution time is measured and used as a metric for comparison. Actually, the maximal execution time presents the total time used to handle the input file in multi-process mode. Figure 7 shows the maximal execution time of the three implementations running with different number of processes.

As shown in Figure 7, the maximal execution time of the three implementations in the load-balanced multi-process mode is smaller than these in the non-load-balanced multi-process mode.

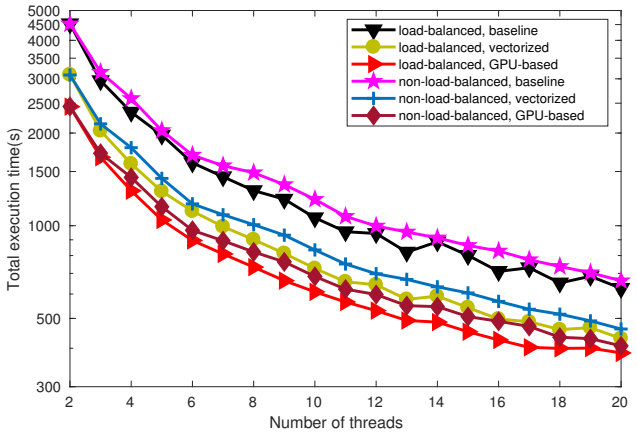

Fig. 7. Maximal execution time of GATK HC implementations in multi-process mode

Moreover, the GPU-based GATK HC implementation in the load-balanced multi-process mode is the fastest. Compared with the baseline implementation and the vectorized GATK HC implementation in the non-load-balanced multi-process mode, the GPU-based GATK HC implementation in the load-balanced multi-process mode achieves up to 2.04x and 1.40x speedup, respectively.

V. Conclusions

This paper presents a novel implementation of a GPU accelerated GATK HC to improve the overall performance of this computationally intensive application. The paper also proposes a load-balanced multi-process optimization that divides the genome into regions of different sizes to ensure a more equal distribution of computation load between different processes. In addition, the paper compares the GPU-based, vectorized and baseline GATK HC implementations in single-threaded, multi-threaded and multi-process modes.

In single-threaded mode, the GPU-based GATK HC is 1.71x faster than the baseline implementation and 1.21x faster than the vectorized GATK HC implementation. In multi-threaded mode, the GATK HC workflow limits the performance improvement achievable by accelerating the pair-HMMs kernel. In multi-process mode, the GPU-based GATK HC implementation is the fastest. In addition, the GPU-based implementation achieves up to 2.04x and 1.40x speedup in load-balanced multi-process mode over the baseline implementation and vectorized GATK HC implementation in non-load-balanced multi-process mode, respectively.

References

[1] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nat Biotech*, 26(10):1135–1145, 10 2008.

[2] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A DePristo. The genome analysis toolkit: A mapreduce framework for analyzing next-generation dna sequencing data. *Genome Research*, 20(9):1297–1303, 09 2010.

[3] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 08 2009.

[4] Daniel C. Koboldt, Qunyuan Zhang, David E. Larson, Dong Shen, Michael D. McLellan, Ling Lin, Christopher A. Miller, Elaine R. Mardis, Li Ding, and Richard K. Wilson. Varscan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Research*, 22(3):568–576, 03 2012.

[5] Geraldine A. Van der Auwera, Mauricio O. Carneiro, Chris Hartl, Ryan Poplin, Guillermo del Angel, Ami Levy-Moonshine, Tadeusz Jordan, Khalid Shakir, David Roazen, Joel Thibault, Eric Banks, Kiran V. Garimella, David Altshuler, Stacey Gabriel, and Mark A. DePristo. From fastq data to high confidence variant calls: the genome analysis toolkit best practices pipeline. *CURRENT PROTOCOLS IN BIOINFORMATICS*, 11(1110):11.10.1–11.10.33, 10 2013.

[6] Andy Rimmer, Hang Phan, Iain Mathieson, Zamin Iqbal, Stephen R.F. Twigg, WGS500 Consortium, Andrew O.M. Wilkie, Gil McVean, and Gerton Lunter. Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nat Genet*, 46(8):912–918, 08 2014.

[7] Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907 [q-bio.GN]*, 2012.

[8] Allison Proffitt. *Broad, Intel Announce Speed Improvements to GATK Powered by Intel Optimizations*. http://www.bio-itworld.com/2014/3/20/broad-intel-announce-speed-improvements-gatk-powered-by-intel-optimizations.html.

[9] Geraldine VdAuwera. *Speed up HaplotypeCaller on IBM POWER8 systems*. https://software.broadinstitute.org/gatk/blog?id=4833.

[10] Shanshan Ren, Vlad M. Sima, and Zaid Al-Ars. Fpga acceleration of the pair-hmms forward algorithm for dna sequence analysis. *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1465–1470, 2015.

[11] Megumi Ito and Moriyoshi Ohara. A power-efficient FPGA accelerator: Systolic array with cache-coherent interface for pair-HMM algorithm. In *2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX)*, pages 1–3. IEEE, 2016.

[12] *HaplotypeCaller Call germline SNPs and indels via local re-assembly of haplotypes*. https://software.broadinstitute.org/gatk/documentation/tooldocs/current/org_broadinstitute_gatk_tools_walkers_haplotypecaller_HaplotypeCaller.php/.

[13] Hamid Mushtaq and Zaid Al-Ars. Cluster-based apache spark implementation of the gatk dna analysis pipeline. In *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1471–1477, Nov 2015.

[14] Hamid Mushtaq, Frank Liu, Carlos Costa, Gang Liu, Peter Hofstee, and Zaid Al-Ars. Sparkga: A spark framework for cost effective, fast and accurate dna analysis at scale. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*, ACM-BCB '17, pages 148–157, New York, NY, USA, 2017. ACM.

[15] Johan Peltenburg, Shanshan Ren, and Zaid Al-Ars. Maximizing systolic array efficiency to accelerate the pairhmm forward algorithm. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 758–762, Dec 2016.

[16] Shanshan Ren, Koen Bertels, and Zaid Al-Ars. Exploration of alternative gpu implementations of the pair-hmms forward algorithm. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 902–909, Dec 2016.

[17] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and . The variant call format and vcftools. *Bioinformatics*, 27(15):2156, 2011.

[18] Yonghong Yan, Max Grossman, and Vivek Sarkar. Jcuda: A programmer-friendly interface for accelerating java programs with cuda. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 887–899, Berlin, Heidelberg, 2009. Springer-Verlag.