Large-Scale Neo-Heterogeneous Programming and Optimization of SNP Detection on Tianhe-2

Yingbo Cui[1], Xiangke Liao[1], Shaoliang Peng[1](✉), Yutong Lu[1], Canqun Yang[1], Bingqiang Wang[2], and Chengkun Wu[1]

[1] School of Computer Science, National University of Defense Technology, Changsha, China
pengshaoliang@nudt.edu.cn

[2] National Supercomputing Center in Shenzhen, Shenzhen, China

Abstract. SNP detection is a fundamental procedure in genome analysis. A popular SNP detection tool SOAPsnp can take more than one week to analyze one human genome with a 20-fold coverage. To improve the efficiency, we developed mSNP, a parallel version of SOAPsnp. mSNP utilizes CPU cooperated with Intel® Xeon Phi™ for large-scale SNP detection. Firstly, we redesigned the key data structure of SOAPsnp, which significantly reduces the overhead of memory operations. Secondly, we devised a coordinated parallel framework, in which CPU collaborates with Xeon Phi for higher hardware utilization. Thirdly, we proposed a read-based window division strategy to improve throughput and parallel scale on multiple nodes. To the best of our knowledge, mSNP is the first SNP detection tool empowered by Xeon Phi. We achieved a 45x speedup on a single node of Tianhe-2, without any loss in precision. Moreover, mSNP showed promising scalability on 4,096 nodes on Tianhe-2.

Keywords: SNP detection · SOAPsnp · Parallelized algorithm · Xeon Phi · Many Integrated Core (MIC) Coprocessor · Tianhe-2

1 Introduction

For DNA sequence analysis, reads are segments of DNA sequence generated by sequencer. Just like string matching in computer science, reads will usually be mapped back to a reference DNA sequence to locate their positions in the reference, which are called aligned reads. SNP (Single Nucleotide Polymorphism) detection is a fundamental and essential process in whole genome analysis. It takes aligned reads, the reference sequence, and sometimes curated database like dbSNP [1] as input, detects the information of aligned reads and reference site by site, and generates a list of SNP sites. Constrained by the memory size, the reference is usually divided into multiple windows with even size. SNPs are detected window by window. However, the division may separate one read into two different windows, generating overlapped bases. As a result, the previous window has to share the information of overlapped bases with the next window when switching windows.

J.M. Kunkel and T. Ludwig (Eds.): ISC High Performance 2015, LNCS 9137, pp. 74–86, 2015.
DOI: 10.1007/978-3-319-20119-1_6

SOAPsnp [2] is a popular SNP detection tool developed by BGI as a member of its SOAP (Short Oligonucleotide Analysis Package) series analysis tools [3]. The software adopts a Bayesian model to call consensus genotype by carefully considering the data quality, alignment and recurring experimental errors. All these information is integrated into a single quality score for each base to measure the calling accuracy. SOAPsnp usually costs several days to analyze one human genome with sequencing depth of 20X, which may account for more than 50 % time at most of a commonly used genome analysis pipeline. The low efficiency calls for a performance boost by advanced computing technologies.

Intel Xeon Phi coprocessor [4] is becoming prevailing with a number of potential applications in accelerating various computations, such as sparse matrix-vector multiplication [5], 1D FFT computations [6], Linpack Benchmark calculation [7], molecular dynamics [8], computational biology [9], and so on.

We performed an in-depth dynamic test of SOAPsnp with gprof [10] and VTune [11], and located the limiting factors that deter its performance. One of those is that the core input data (aligned bases) is stored as a highly sparse matrix, which results in a large amount of redundant computation and huge overhead of switching windows. Moreover, the current version of SOAPsnp is a CPU-based single-threaded program although SNP detections between different DNA sites are independent. In this paper, we aim to improve the efficiency of SNP calling algorithm and develop a high performance version of SOAPsnp utilizing Xeon Phi. The ultimate goal is to apply the improved tool in large-scale SNP detection of human or other complex species genome.

To realize the above objectives, we proposed a series of optimization strategies:

(1) We proposed a space-efficient data structure to replace the original inefficient sparse matrix in SOAPsnp. The new structure can dramatically reduce memory overhead and improves operation efficiency.
(2) We transported the Bayesian model to Xeon Phi with offload mode and developed a coordinated parallel framework utilizing both CPU and Xeon Phi.
(3) For large scale parallelism, we proposed a read-based window division (RWD) strategy, which enables parallel file reading for different processes. RWD efficiently improves the throughput and parallel scale of mSNP.

mSNP is freely available from https://github.com/lemoncyb/mSNP under GPL license. We evaluated our work on the Tianhe-2 supercomputer [12], where each compute node (see Table 2) is equipped with two Xeon E5-2692 v2 2.2 GHz CPUs and three Xeon Phi 31S1P coprocessors. On one compute node of Tianhe-2, mSNP managed to finish the analysis of one 20X human genome within two hours, whereas the original CPU-based SOAPsnp used several days. The software maintains promising scalability on 4,096 nodes (98,304 CPU cores and 688,128 MIC cores). Our experiments demonstrated that mSNP is an efficient and scalable software for large-scale SNP detection of human genome. The details of evaluation are presented in Sect. 5.

The remaining of this paper is organized as follows. Section 2 presents related work. Section 3 presents the analysis of our work. Section 4 describes the architecture of mSNP. Performance evaluation is presented in Sect. 5. Section 6 concludes the paper.

2 Related Work

In this section, we survey most of the popular SNP detection tools and related optimization work. We also introduce the Intel Xeon Phi coprocessor, which has been deployed as the primary accelerator on Tianhe-2.

2.1 SNP Detection Tools

SNP detection tools take aligned reads, a reference sequence, in some cases dbSNP as input to detect SNPs. Web-based tools, such as HaploSNPer [13] and SNi-Play [14], were deployed on web servers that can be accessed from anywhere conveniently via a web page. However, the data security and uploading time prevents them from performing large-scale analysis. Therefore, stand-alone tools like QualitySNP [15], SAMtools [16], SOAPsnp [2], GATK [17] and Illuminas Isaac [18] etc. were developed.

SNP detection is time-consuming; as a result, some optimization efforts have been carried out to improve the performance. Crossbow [19] is a parallel solution using Hadoop [20] and accelerates detection with cloud computing. Rainbow [21] optimizes Crossbow for larger sequencing datasets. GSNP [22] accelerates SNP detection with GPU (graphics processing unit) to achieve better performance. Mucahid adopts cluster for computation and achieves a good load balance [23]. To the best of our knowledge, mSNP is the first SNP detection tool powered by Intel Xeon Phi.

2.2 Intel Xeon Phi Coprocessor

Intel announced its Xeon Phi coprocessor based on Many Integrated Core (MIC) architecture in November 2012 [4]. The coprocessor is equipped with 50+ cores clocked at about 1 GHz and 6 GB or more on-card memory. Each core supports 4 hardware threads. The double precision peak performance of each coprocessor is higher than 1 TFlops. The architecture of MIC is x86-compatible, which alleviates the efforts needed to transport applications to Xeon Phi compared to its counterpart GPU. Some simple applications can even run directly on Xeon Phi simply after re-compiling. There are two major modes to employ Xeon Phi in applications:

(1) native mode, where Xeon Phi has one copy of the application and runs the application natively like a compute node.
(2) offload mode, where the application runs as a master thread on CPU and offloads some selected work to Xeon Phi, treating Xeon Phi as a coprocessor [24].

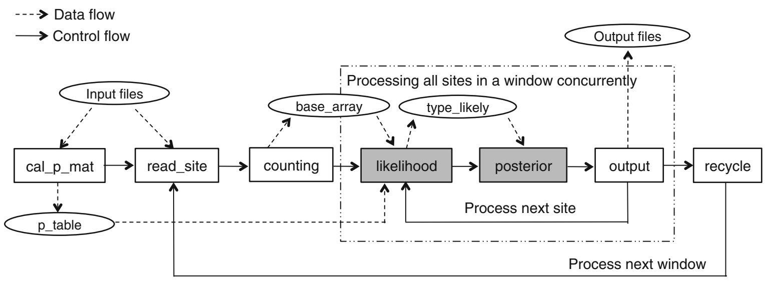


Fig. 1. Workflow of SOAPsnp. The dash lines illustrate data flow, and the real lines illustrate control flow.

As mentioned in Sect. 1, more and more applications are accelerated by Xeon Phi, from basic scientific computation to biology application [5–9]. Xeon Phi is showing great potential in parallel computing.

3 Performance Profiling Analysis of SOAPsnp

In this section, we will present our analysis of the workflow and bottleneck profiling of SOAPsnp.

3.1 Workflow of SOAPsnp

Figure 1 illustrates the workflow of SOAPsnp. SOAPsnp takes aligned reads, a reference genome, and in some cases dbSNP as input. The output is consensus genotype information. SOAPsnp mainly contains seven modules: ***cal_p_mat***, ***read_site***, ***counting***, ***likelihood***, ***posterior***, ***output***, and ***recycle*** (italics bold font represents function module, italics font represents data structure). The core data structures include *p_matrix*, *base_info*, and *type_likely* (the oval block of Fig. 1).

cal_p_mat module takes input reads and generates a calibration matrix *p_matrix* for ***likelihood*** computation. *p_matrix* is a four-dimensional matrix ($256 \times 256 \times 4 \times 4 = 1,048,576$) with a size of 8 MB. Constrained by the memory size, SOAPsnp divides the reference into multiple windows with even size. In each window, SNP calling is performed site by site. The ***read_site*** module loads a fixed number of sites (a window size) from input file. Then ***counting*** collects the information of aligned bases for each site and stores the information in *base_info*. ***likelihood*** takes *base_info* and *p_matrix* as inputs, calculates the likelihood and stores it in *type_likely*. After posterior calculation, calling results of one site are written to the output file. Then the next site of the current window will be processed from ***likelihood*** too. The ***recycle*** module switches windows by dealing with the overlapped bases and re-initializes buffers for new window.

base_info is a four dimensional matrix ($4 \times 64 \times 256 \times 2$, corresponding to $base \times score \times coord \times strand$), storing the information of bases aligned to each

Table 1. Time breakdown of SOAPsnp

Module	cal_p_matrix	read_site	couting	likelihood	posterior	recycle	output
Time/s	16.73	2.35	40.59	1478.36	23.47	210.19	752.98
%	0.66 %	0.09 %	1.59 %	57.93 %	0.92 %	29.50 %	8.42 %

DNA site. The dimensions stand for four aspects of an aligned base: the base type, the sequencing quality score, the coordinates on read and the strand of read.

3.2 Bottleneck Profiling

To identify the performance bottleneck of SOAPsnp, we analyzed the code of SOAPsnp and divided it into seven main modules, as described in Subsect. 3.1. Then we timed each module and obtained a time breakdown, as listed in Table 1. ***likelihood*** is the most time-consuming module, which takes about 58 % of the total processing time. The second is ***recycle*** taking 30 % percentage. ***output*** is ranked third with 8.4 %. Then we investigated further with Intel VTune [11] to detect the most time-consuming operations in ***likelihood*** and ***recycle***. Further code analysis shows large amounts of these operations are memory accesses, especially the accesses to *base_info* data structure storing information of aligned reads. As mentioned in Subsect. 3.1, ***likelihood*** traverses *base_info* to fetch aligned reads. ***recycle*** module copies the information of aligned reads across adjacent windows. The optimization strategy to *base_info* is described in Subsect. 4.1.

4 Design and Implementation of mSNP

In this section, we will describe the design and implementation of mSNP in detail.

4.1 Consolidating Sparse Matrix

SOAPsnp detects SNP site by site. For each site computation, every element of *base_info* will be accessed exactly once (including zero elements), which would generate a total number of 131,072($= 4 \times 64 \times 256 \times 2$) memory accesses. That is, 393 trillions memory accesses will be made for a whole human genome with about 3 billion sites.

One notable fact is that, *base_info* is a highly sparse matrix. Each element of *base_info* stands for a combination of four dimensions ($base \times score \times coord \times strand$) and is initialized with zero. The element value will increase by one if a base in a read matches the combination. However, sequencing depth is usually smaller than 100X, and the bases in one human genome are relatively fixed, so most elements in *base_info* are zero. We tested several human genomes and found that only less than 0.08 % of *base_info* elements are non-zero. This means that most memory accesses to the matrix are in vain.

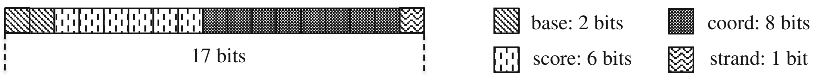


Fig. 2. Bit composition of *base_array*.

To reduce the amount of unnecessary memory accesses, we design a space-efficient data structure *base_array* to store the information of each DNA site. The *base_array* only stores the dimensions information of non-zero elements. As illustrated in Fig. 2, the four dimensions ($base \times score \times coord \times strand$) are integrated into 17 bits in one word (32 bits) by bit operations. For repeated bases, *base_array* stores multiple copies of the same coordinates. Thus, all information is maintained and the space complexity of *base_info* is significantly reduced, as the percentage of non-zero elements of *base_info* is only 0.08 %.

By analyzing the source code, we discovered that recycle module produces a large amount of memory copy operations when switching windows, especially the copy of *base_info* with a size of 13MB. With the introduction of *base_array*, the cost of windows switching is reduced by three orders of magnitude too.

4.2 Coordinated Parallelism Between CPU and Xeon Phi

As described in Subsect. 3.1 and illustrated in Fig. 1, SOAPsnp divides the SNP calling of one genome into multiple windows. In each window, SNP detection is performed site by site. Theres no dependency between sites. We parallelize the procedure by multi-threading on both CPU and Xeon Phi, where each thread handles one site.

Due to Xeon Phi's weak ability of file operation in native mode, mSNP adopts the offload mode of Xeon Phi. In the naive offload mode, CPU will have to pause and wait for the results to be returned from Xeon Phi, which results in a waste of the CPU computing power. In mSNP, we make the data transfer between CPU and Xeon Phi asynchronous, which allows CPU to take on other job immediately after launching the offload region, as illustrated in Fig. 3. When the offload region is finished, CPU retrieves results from Xeon Phi and resumes other operations.

4.3 Collaborated Parallel Window Division

As described in Subsect. 4.2, mSNP parallelizes the SNP detections of different sites in a window with multi-thread. While the computing power of one computing node is limited, to achieve higher performance, we have to parallelize SOAPsnp across nodes.

SOAPsnp performs SNP calling window by window. One straightforward strategy to parallelize SOAPsnp across nodes is to assign each node at least one window. Different nodes call SNPs of different windows simultaneously.

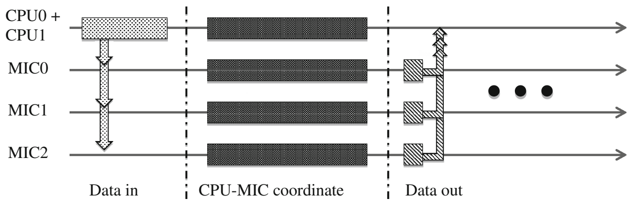


Fig. 3. Coordinated parallelism between CPU and Xeon Phi. Data in stands for CPU transfers input data to MICs. CPU-MIC coordinate means CPU and MIC perform computation simultaneously. Data out denotes MICs transfer results back to CPU.

SOAPsnp divides windows evenly according to the coordinates of base on reference sequence, denoted as coordinate-based window division (CWD) as illustrated in Fig. 4. The coordinate of base is stored in the aligned reads file, as attached information to each read. The coordinate is not known until the read sequence is loaded into memory. It's impossible to locate a read with given nonzero coordinate beforehand. That is to say, it's impossible for different processes to load reads from the start of any window simultaneously. To parallelize the SNP calling of different windows, we have to choose a master-slave mode. The master process loads reads into memory sequentially, prepares base information for each window, and sends window task to idle slave processes. For the master-slave mode, the throughput and parallel scale are limited by the master process, where all input data come from. Different processes cannot load reads simultaneously.

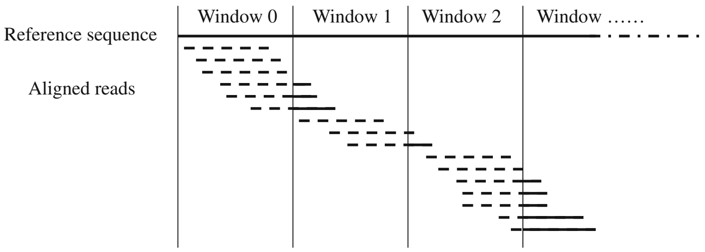


Fig. 4. Coordinate-based window division of SOAPsnp. The dash lines denote aligned reads. The real zones of dash lines represent the bases that belong to the next window, but are loaded by the previous window, that is overlapped bases.

To improve the parallel scale across nodes, we designed a read-based window division (RWD) strategy. As illustrated in Fig. 5, windows are divided by the number of reads, each window containing almost equal number of reads. As each aligned read occupies four lines in file, the RWD strategy actually divides windows by file lines. Different processes can load reads from different lines of input file simultaneously.

Another problem of RWD strategy is to deal with overlapped reads which belong to the next window, but are loaded by the previous window, responding to the real lines in Figs. 4 and 5. SOAPsnp detects SNP site by site. In order to maintain the completeness of each site, the information of overlapped bases has to be transferred to the next window before the next window launches. To realize this, two adjacent processes P_n and P_{n+1} have to send one message to each other. As illustrated in Fig. 5, P_{n+1} sends the position of its first site Pos_1 to P_n, P_n sends the information of sites after Pos_1 (overlapped bases) to P_{n+1}. Different processes accomplish loading step evenly at the same time, because the number of assigned reads is evenly equal. Then all processes communicate with each other at the same time. When communication finishes, all processes start SNP calling even simultaneously. Based on the above description, the throughput and parallel scale of mSNP improve efficiently compared with SOAPsnp. Moreover, for each process is assigned evenly equal number of reads, different processes can get a better load balance.

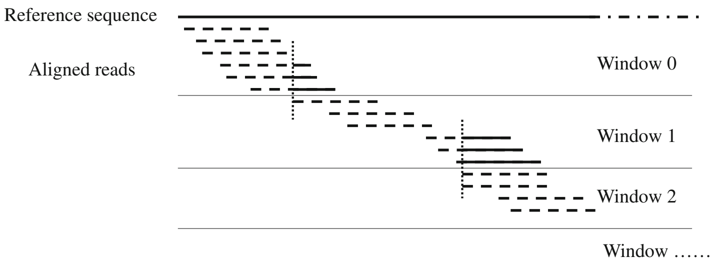


Fig. 5. Read-based window division (RWD) of mSNP. The dash lines illustrate aligned reads. The real zones of dash lines mean the bases that belong to the next window, but are read by the previous window, that is overlapped bases.

5 Evaluation

We evaluated the performance of mSNP from four aspects: effectiveness of space-efficient format *base_array*, CPU and Xeon Phi cooperation, RWD performance and scalability.

5.1 Experimental Setup

We evaluated our work on the Tianhe-2 supercomputer in the National Super Computer Center in Guangzhou (NSCC-GZ) [12]. The configuration of Tianhe-2 is described in Table 2. The whole system consists of 16,000 compute nodes.

The latest version of SOAPsnp is v1.03, which is available from it's website[1]. SOAPsnp v1.03 is a CPU-based single thread program. In consideration of

[1] SOAPsnp website: http://soap.genomics.org.cn/soapsnp.html.

equality, we chose different baselines for different evaluations, and the details are described in the subsections. We prepared three datasets for evaluation: 3.2 GB, 73 GB and 542 GB. The details are described in the subsections too.

Table 2. Configuration of Tianhe-2's compute node

	Xeon® E5-2692 v2 CPU	Xeon® Phi™ 31S1P
Sockets × *Cores* × *Threads*	2 × 12 × 2	3 × 57 × 4
Clock Frequency (GHz)	2.20	1.10
L1/L2/L3 Cache (KB)	32/256/30,720	32/512/-
Memory Size (GB)	64	6

5.2 Dimension Reduction of Sparse Matrix

We adopted SOAPsnp v1.03 as baseline in this evaluation and the optimized version is also CPU-based single thread. The size of test data is 3.2 GB. Figure 6 shows the time consumptions of ***likelihood*** and ***recycle*** before and after optimization on one node of Tianhe-2. Our space-efficient new representation format *base_array* outperforms *base_info* 32+ times in ***likelihood*** and 56+ times in ***recycle*** function. *base_array* stores only non-zero elements to avoid unnecessary memory accesses.

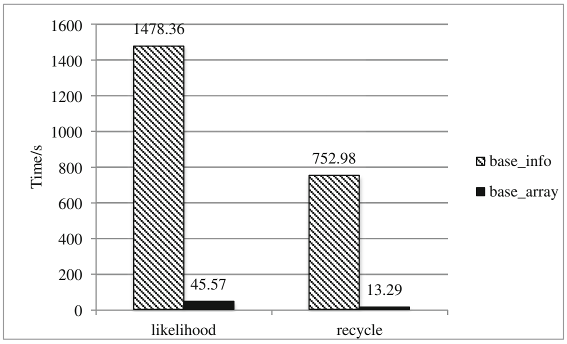
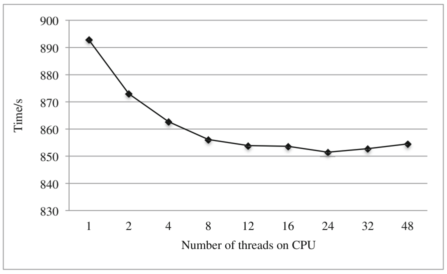


Fig. 6. Performance of ***likelihood*** and ***recycle*** before and after optmization

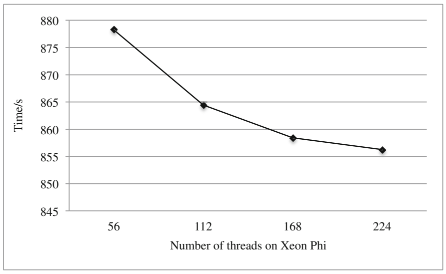
Table 3 shows the time breakdown of the CPU-based optimized single thread version of SOAPsnp. The main two modules optimized are ***likelihood*** and ***recycle***. After optimization, for a 3.2 GB dataset, the percentage of ***likelihood*** is 5.10 %, and that of ***recycle*** is 1.49 %. The two modules are no longer the bottlenecks. ***output*** becomes the most time-consuming module taking more than 84 % and turns into the big bottleneck. It's hard to parallelize output in multi-threads, while possible in multi-processes.

Table 3. Time breakdown of CPU-based optimized version of SOAPsnp

Module	cal_p_matrix	read_site	couting	likelihood	posterior	recycle	output
Time/s	17.05	2.13	41.21	45.57	22.64	13.29	750.88
%	1.91 %	0.24 %	4.62 %	5.10 %	2.54 %	1.49 %	84.11 %



(a)



(b)

Fig. 7. (a) Performance of mSNP on CPU. (b) Performance of mSNP on Xeon Phi.

5.3 CPU and Xeon Phi Cooperation

mSNP supports coordinated computation between CPUs and MICs. We tested mSNP under different number of CPUs and MICs on one node of Tianhe-2. To determine the proper number of threads launched in CPU and MIC, we evaluated the performance of mSNP with the number of threads varying first. We adopted 3.2 GB dataset in the tests. Figure 7(a) illustrates the performance of mSNP on CPU. There are two sockets 12-core CPUs in each compute node. The time decreases with the number of threads increasing, and the peak performance is obtain in 24 threads, 1 core assigned with 1 thread. After 24 threads, the performance drops gradually. Figure 7(b) shows the performance on Xeon Phi. The peak performance is obtained in 224 threads, 1 core assigned with 4 threads. Thus, we launched 24 threads on CPUs and 224 threads on one Xeon Phi for the later tests.

The smooth varieties in Fig. 8 are contributed by the big proportion of output in mSNP (Table 3). It's hard to parallelize output in multi-thread. To make the illustration for CPU-Phi cooperation distinct, we chose 73 GB dataset and presented the time of ***likelihood*** only, because other modules of mSNP are not parallelized by multi-thread. Figure 8(a) shows the performance of ***likelihood*** for CPUs cooperating with Phis. As illustrated, for SNP detection, the computing power of one Xeon Phi corresponds to that of two CPUs in Tianhe-2. The high accelerator speedup comes from the massively parallelism on Xeon Phi. There's no other communication, except for transferring input data to Phi and getting results from Phi to CPU. Thus, the performance increases nearly linearly as the number of Phi increases.

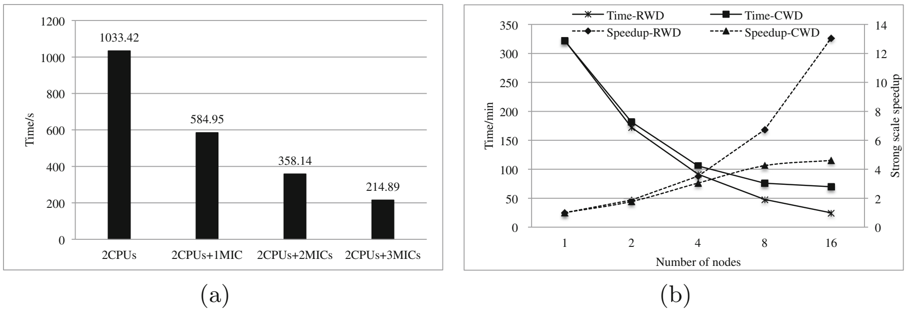


Fig. 8. (a) Performance of ***likelihood***. (b) Time and speedup of RWD vs. CWD.

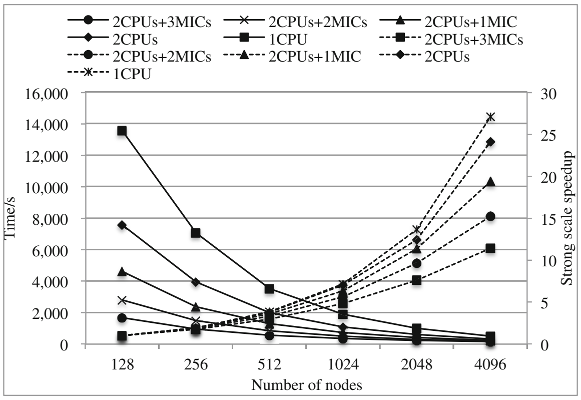


Fig. 9. Strong scale speedup of mSNP

5.4 RWD Performance

Figure 8(b) shows the performance and strong scale speedup of RWD vs. CWD from 1 to 16 nodes on Tianhe-2. All two CPUs with 24 threads and three MICs with 224 threads per Phi are used in each compute node. The size of dataset is 73 GB. RWD and CWD achieve evenly equal speedup before 4 nodes. After 4 nodes, the performance of RWD exceeds CWD more and more. The speedup of CWD increases slower and slower after 4 nodes. CWD only obtains about 4.2 folds speedup on 16 nodes, while RWD achieves about 13 folds speedup, which is over 3 folds faster. The good scalability is contributed by RWD's parallel reading capability.

5.5 Scalability

We tested the scalability of mSNP from 128 nodes to 4,096 nodes on Tianhe-2. Figure 9 presents the strong scale speedup of mSNP. The dash lines indicate speedup and the real lines indicate time. The size of test data is 542 GB.

For a better presentation, we used the performance achieved on 128 nodes as baseline. The lines in the figure stand for the number of processors (CPU, MIC) used in each compute node of Tianhe-2. We observed a speedup of about 27.5 from 128 nodes to 4,096 nodes with 2CPUs+3MICs, and a speedup of 24 with 2CPUs+2MICs, a speedup of about 19 with 2CPUs+1MIC, about 15 folds speedup with 2CPUs, about 11.5 folds speedup with 1CPU. These results demonstrate a promising result for strong scalability on the large scale CPU-MIC heterogeneous system, Tianhe-2.

6 Conclusion

In this paper, we presented mSNP, which a large-scale parallel SNP detection tool accelerated by Intel Xeon Phi. Firstly, we proposed a space-efficient representation format that can substantially reduces the amount of memory accesses and overhead of switching windows. Secondly, we developed a coordinated parallel framework using CPU and Xeon Phi, which optimized hardware utilization. Thirdly, we proposed a read-based window division strategy to improve data throughput and parallel scale across nodes. We evaluated our work on Tianhe-2 supercomputer. It achieves about 45x speedup on one node and exhibits strong scalability on 4,096 nodes. The algorithm optimization, parallelization on both CPU and Xeon Phi lead to a significant reduction of computing time.

Acknowledgments. We would like to thank Mr. Yingrui Li from BGI for providing the source code of SOAPsnp and Dr. Jun Wang from BGI for providing related test data. We would also like to thank Prof. Hans V. Westerhoff from University of Manchester for discussions of the human genome re-sequencing analysis problem and thus improving our own understanding. This work is supported by NSFC Grant 61272056, U1435222, 61133005, 61120106005, 91430218 and 61303191.

References

1. National Center for Biotechnology Information. http://www.ncbi.nlm.nih.gov/SNP/
2. Li, R., Li, Y., Fang, X.: SNP detection for massively parallel whole-genome resequencing. Genome Res. **19**(6), 1124–1132 (2009)
3. Short Oligonucleotide Analysis Package Sites. http://soap.genomics.org.cn/index.html
4. James, J., Reinders, J.: Intel Xeon Phi Coprocessor High Performance Programming. Morgan Kaufmann, Newnes (2013)
5. Liu, X., Smelyanskiy, M., Chow, E., Dubey, P.: Efficient sparse matrix-vector multiplication on x86-based many-core processors. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, pp. 273–282. ACM (2013)
6. Park, J., Bikshandi, G., Vaidyanathan, K., Tang, P.T.P., Dubey, P., Kim, D.: Tera-scale 1D FFT with low-communication algorithm and Intel® Xeon Phi™ coprocessors. In: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, p. 34. ACM (2013)

7. Heinecke, A., Vaidyanathan, K., Smelyanskiy, M., Kobotov, A., Dubtsov, R. et al.: Design and implementation of the linpack benchmark for single and multi-node systems based on intel® Xeon Phi coprocessor. In: 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), pp. 126–137. IEEE (2013)
8. Pennycook, S.J., Hughes, C.J., Smelyanskiy, M., Jarvis, S.A.: Exploring SIMD for molecular dynamics, using intel® Xeon® processors and intel® Xeon Phi coprocessors. In: 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1085–1097. IEEE (2013)
9. Misra, S., Pamnany, K., Aluru, S.: Parallel mutual information based construction of whole-genome networks on the intel® Xeon PhiTM coprocessor. In: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, pp. 241–250. IEEE (2014)
10. Graham, S.L., Kessler, P.B., McKusick, M.K.: Gprof: a call graph execution profiler. ACM SIGPLAN Not. **39**(4), 49–57 (2004)
11. Wikipedia Sites of VTune. http://en.wikipedia.org/wiki/VTune
12. TOP500 Supercomputer Sites. http://www.top500.org/system/177999
13. Tang, J., Leunissen, J.A.M., Voorrips, R.E.: HaploSNPer: a web-based allele and SNP de-tection tool. BMC Genet. **9**(1), 23 (2008)
14. Dereeper, A., Nicolas, S., Le Cunff, L.: SNiPlay: a web-based tool for detection, management and analysis of SNPs. Application to grapevine diversity projects. BMC Bioinform. **12**(1), 134 (2011)
15. Tang, J., Vosman, B., Voorrips, R.E.: QualitySNP: a pipeline for detecting single nucleotide polymorphisms and insertions/deletions in EST data from diploid and polyploid species. BMC Bioinform. **7**(1), 438 (2006)
16. Li, H., Handsaker, B., Wysoker, A.: The sequence alignment/map format and SAMtools. Bioinformatics **25**(16), 2078–2079 (2009)
17. DePristo, M.A., Banks, E., Poplin, R.: A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature Genet. **43**(5), 491–498 (2011)
18. Raczy, C., Petrovski, R., Saunders, C.T.: Isaac: ultra-fast whole-genome secondary analysis on Illumina sequencing platforms. Bioinformatics **29**, 2041–2043 (2013). btt314
19. Langmead, B., Schatz, M.C., Lin, J.: Searching for SNPs with cloud computing. Genome Biol. **10**(11), R134 (2009)
20. Shvachko, K., Kuang, H., Radia, S., The hadoop distributed file system. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10. IEEE (2010)
21. Zhao, S., Prenger, K., Smith, L.: Rainbow: a tool for large-scale whole-genome sequencing data analysis using cloud computing. BMC Genomics **14**(1), 425 (2013)
22. Lu, M., Zhao, J., Luo, Q.: GSNP: a DNA single-nucleotide polymorphism detection system with GPU acceleration. In: 2011 International Conference on Parallel Processing (ICPP), pp. 592–601. IEEE (2011)
23. Kutlu, M., Agrawal, G.: Cluster-based SNP calling on large-scale genome sequencing data. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid computing (CCGrid), pp. 455–464. IEEE (2013)
24. Cui, Y., Liao, X., Zhu, X.: mBWA: a massively parallel sequence reads aligner. In: Saez-Rodriguez, J., Rocha, M.P., Fdez-Riverola, F., De Paz Santana, J.F. (eds.) PACBB 2014. AISC, vol. 294, pp. 113–120. Springer, Heidelberg (2014)