ANALYSING AND OPTIMIZING SNPHAP USING RADIX-2 COMPUTATION AND OPENMP

Ratthaslip Ranokphanuwat, Apisit Rattanatranurak, Surin Kittitornkun
Department of Computer Engineering, Faculty of Engineering
King Mongkut’s Institute of Technology Ladkrabang
Bangkok 10520, Thailand
email: s100635@kmitl.ac.th, s4611301@kmitl.ac.th, kksurin@kmitl.ac.th

Sissades Tongsima
Genome Institute, National Center for Genetic Engineering and Biotechnology
113 Thailand Science Park, Paholyothin Road, Klong 1, Klong Luang
Pathumtani 12120, Thailand
email: sissades@biotec.or.th

ABSTRACT
In this paper, the run time complexity of SNPHAP, which is a haplotype inference tool, is extensively examined. The analysis is based on our previous work in terms of profiling and run-time complexity function. To reduce the run time complexity and enhance its performance, a Radix-2 computation and OpenMP multithreading are applied. The optimized results are compared with both original and compiler optimized versions on an AMD A6-3650 Linux machine. Due to the Radix-2 technique, the complexity is drastically reduced. In addition, the theoretical Speedup is consistent with the experimental one. Furthermore, up to 1,303% Speedup is achievable as a result of OpenMP multithreading.

KEY WORDS
SNPHAP, Radix-2, OpenMP, Optimization

1 Introduction

There are processes of nature found in many populations, for example, mutation, selection, recombination, gene-conversion have combined alleles at many loci in the large variety of genotypes. The problem is to discover the genotypes that have significant and biologically important associations with vital characteristics of interest. A solution and computational section of this problem is to infer haplotype information from genotype information [1].

The procedure for inferring haplotypes from the unphased genotype data is called haplotype inference. Regrettably, this procedure utilizes long computation time and uses intense resources, since the computational space or number of possible haplotypes grows exponentially [2]. Then, if the procedure can be extended into parallel optimization algorithms, the throughput can be increased.

There are several bioinformatics programs used for haplotype inferring include GERBIL, PL-EM, fastPhase, Phase, HaploRec, and SNPHAP [3]. We chose to optimize SNPHAP, which is an EM-based haplotype inference tool developed by Prof. David Clayton due to its popularity and availability of the program source code. Recently, Eroonen et al. [4], compared running time of those haplotyping tools while varying the number of loci as shown in Figure 1. From this report [4], the SNPHAP is the fastest tool among others. Although the SNPHAP has the strong point of advantage runtime, it is still the sequential (single thread) version. The runtime of SNPHAP can be further reduced with optimizing and multithreading technique. So, it is more challenging and profitable to accelerate the SNPHAP with optimization and parallelization.

Nowadays, manufacturers usually integrate the processor cores onto a single die. This can produce powerful computers simply by linking many existing smaller ones called a Multi-core processor that resolves the power problem in single-core processor with higher clock frequency. To achieve higher performance, programmers must parallelize (multithread) a single program running on multiple cores simultaneously [5].

In our previous work [6], the original SNPHAP was optimized with Radix Comparison algorithm and then multithreaded with OpenCL to reduce the computation time. However, it is not trivial in terms of Speedup.

The contributions of this study are as follows. The run time complexity of SNPHAP is thoroughly examined and expressed in Big-O. To reduce its Big-O, a new optimization method called Radix-2 Computation algorithm (Radix-2 Comparison + Radix-2 Qsort) is developed based on Radix-2 Comparison algorithm and then parallelized (multithreaded) with OpenMP 3.0 [7] running on Multi-core processor. The Radix-2 Computation utilizes binary shift operations for coding the haplotypes so that the Radix Comparison becomes a simple number comparison.

The rest of this paper is organized as follows. Section 2 presents an overview of SNPHAP, OpenMP and related works. Then, the SNPHAP is explored and profiled how to optimize with Radix-2 Computation, and Section 3 shows developing the OMP Radix Computation SNPHAP. Section 4 demonstrates and discusses the results of our exper-
Figure 1. Run time per genotype in seconds vs. Number of genotypes of each program (Loci=30) [3].

iments. Finally, Section 5 concludes and shows new ideas for future work.

2 Literature Review

2.1 SNPHAP:Haplotype Inference Tool

SNPHAP [3] is an application software for approximating frequencies of haplotypes of large numbers of diallelic markers from unphased genotypes from unassociated subjects. This application software provides for missing data in genotypes. The SNPHAP can use arbitrary starting points to explore for various results.

SNPHAP uses EM (Expectation Maximization) algorithm to compute maximum likelihood estimations (MLEs) for haplotype frequencies from given genotypes. The EM algorithm executes repeatedly until the parameter estimation is stable. There are two steps in EM algorithm, M step computes the next set of approximates of haplotype probabilities. The E step gives haplotype probability estimations and supposes Hardy-Weinberg equilibrium, computes phased probabilities and complete genotype assignment for each instance. For SNPHAP, the M step and the E step are placed in hap\_prior() and hap\_posterior() as shown in Figure 2, respectively.

Firstly, SNPHAP begins with an initial number of phases \( n_{phase} = 0 \). Secondly, the possible haplotypes are expanded by hap\_expand() and then sorted by qsort(). Next, the hap\_prior() and hap\_posterior() are executed until the haplotype frequencies are stable. Note that the input of hap\_prior() is the sorted haplotypes. After that, the haplotype instances which Posterior\_prob is less than min\_prior() will be removed. Finally, this algorithm will be executed repeatedly until the \( n_{phase} \) is correspond to \( L \) as shown in Figure 2.

Figure 2. Original SNPHAP and Optimized SNPHAP flowchart(in dotted box) [3].

2.2 OpenMP

OpenMP [7] is a set of compiler directives for thread-based parallel programming on shared memory multiprocessors. It is rapid growing the de-facto standard for parallelizing applications. OpenMP is designed for Fortran, C and C++ programming languages. Furthermore, OpenMP involves an environment variables and runtime libraries. These are normally used to inspect and revise the execution parameters.

The execution model for OpenMP is a fork/join model. An OpenMP program starts with a master thread. When the master thread run into a parallel construct then a group of worker threads is forked. The workers threads are joined at the end of parallel construct. It can specify the number of parallel constructs in a single program.
The OpenMP allows programmers to develop parallel constructs in nested.

OpenMP has the advantages such as ability to parallelize small section of a source code at a time, practicality of scaling the tasks to an intensive number of processors, implementation of simple parallel algorithms is fast and not difficult, and minor impact on code quantity depending size of changes required for parallel scalability. Therefore, OpenMP is a good alternative for implementing the program in parallel.

2.3 Related Works

There have been research to implement the SNP HAP in parallel. Udom Ranok, et al [8] found that cmp_hap(), hap_prior() and hap_posterior() are the hotspot functions. Furthermore, the quick sort algorithm utilizes outstanding run time after parallelization of cmp_hap(). Therefore, they parallelized cmp_hap() and qsort() with OpenMP 3.0 and ran it in parallel on an 8-core Intel Xeon E5405 and Intel Xeon E5520. Their work is remarkable in terms of Speedup at up to 316% and 410%, respectively with 151 genotyping loci with 10,000 populations samples.

Based on our previous work [6], the original SNP HAP have been optimized and parallelized to Sequential Radix and Parallel Radix SNP HAP on an AMD A6-3650 with OpenCL. This work proposes only Radix Comparison concept that uses in haplotype comparison method. It is not trivial in terms of Speedup at up to 260% and 271%, respectively with 10,000 populations samples with 151 loci.

3 Methods

3.1 SNP HAP Algorithm Analysis

The program performance depends on the algorithm, the language, the compiler, the architecture, and the actual hardware. The algorithm determines the instructions of program executed and affects the run time. Therefore, we can analyse and develop the algorithm to improve the program performance.

The hap_prior() and hap_posterior() in SNP HAP repeatedly compare a set of possible haplotypes. We have used GNU GProf Profiler [9] to profile SNP HAP at 10,000 genotypes with 151 loci. The results show that cmp_hap() is the hotspot function in both functions.

The worst cases of run time complexity can be now written in terms of Big-O notation as illustrated in Equation 1, and Equation 2, respectively.

\[ T_{prior}(L, M, G, n) = O(L^2 MG(2^n)) \]  
\[ T_{posterior}(L, M, G, n) = O(L^2 MG(2^n)) \]  
\[ T_{qsort}(L, G, n) = O(L^2 G(2^n) \times \log(L^2 G(2^n))) \]  

where \( G \) is the number of genotypes, \( M \) is the maximum iteration in EM algorithm, \( L \) is the number of loci in each genotype, and \( n \) is the number of heterozygous loci of each genotype and proportional with \( L \).

As described in related work [8], the quick sort algorithm uses significant execution time after parallelization of cmp_hap(). The run time complexity in terms of Big-O notation of qsort() is shown in Equation 3.

Since the Big-O notations which are demonstrated, the run time of SNP HAP depends on the haplotyping loci, the number of heterozygous loci of each genotype, the maximum iteration in EM algorithm, or the number of genotypes. These equations can be used to compare the developed or modified SNP HAP with algorithm.

3.2 Radix-2 Comparison

The original algorithm of SNP HAP has made a considerable run time of SNP HAP. The run time is conditional on the number of haplotyping loci and the number of heterozygous loci of each genotype that correspond to haplotyping loci. Thus, we should reduce the run time complexity in the number of haplotyping loci factor which is \( L^2 \).

To enhance the performance, we have developed a new haplotype comparison algorithm called Radix-2 Comparison by encoding each haplotype as an integer number while original SNP HAP encoding each haplotype as a string. Note that haplotyping 101 loci and 151 loci must be encoded by using two 64 bit integer.

There are two members of the haplotype element set, 1 stands for wild type and 2 stands for mutation type as shown in Equation 4.

\[ hap[i] \in \{1, 2\}; i = 0, 1, 2, 3, ..., L - 1 \]  

Thus, we can encode the haplotypes using the radix-2 or base-2 number for the haplotype computing as demonstrated in Equation 5.

\[ radix\_value = \sum_{i=0}^{L-1} ((hap[i] \geq 1) \ll i) \]  

To encode the haplotypes, we use shift right operation to encode the haplotype. Its performance is greater than the arithmetic operations. Note that our previous work [6] used the arithmetic operations .

There are few steps to encode the haplotypes to radix_value. Initially, shifting the haplotype elements right by one bit (\( hap[i] \geq 1 \)) to be 0, 1. After that, shifting them left by \( i \) bits (\( \ll i \)). Finally, storing the results to radix_value of each haplotype. This algorithm is hap_radix_value() that is located before qsort().

The \( L \) factor in Big-O notation of haplotype comparison are decreased. They are reduced from \( L^2 \) to \( L \) as illustrated in Equation 6 and Equation 7, respectively. Therefore, the run time of Radix-2 Comparison SNP HAP is much faster than the original SNP HAP.

\[ T_{radix\_prior}(L, M, G, n) = O(L MG(2^n)) \]  

422
\[ T_{\text{radix\_posterior}}(L, M, G, n) = O(LMG(2^n)) \]  
\[ T_{\text{radix\_qsort}}(L, G, n) = O(LG(2^n) \times \log(LG(2^n))) \]

where \( G \) is the number of genotypes, \( M \) is the maximum iteration in EM algorithm, \( L \) is the number of haplotyping loci and \( n \) is the number of heterozygous loci of each genotype and proportional with \( L \).

### 3.3 Radix-2 Qsort

There is \( \text{cmp\_hap()} \) that is haplotype comparison in the \( qsort() \). Consequently, we can use \( \text{radix\_value} \) to compare two haplotypes in this function. To improve the performance, we have developed a new \( qsort() \) that similar to the Radix-2 Comparison, and called it \( \text{radix\_qsort()} \).

The Radix-2 \( qsort() \) just simply compares two \( \text{radix\_value} \) in Equation 5. Thus, the \( L \) factor in Big-O notation of comparison is eliminated. It is reduced from \( L^2 \) to \( L \) as shown in Equation 8 similar to Radix-2 Comparison. However, this improvement is not significant when compared with Radix-2 Comparison.

### 3.4 Radix-2 Computation

The performance of original SNPHAP can be improved using both Radix-2 Comparison and Radix-2 Qsort. To integrate two algorithms, we have located \( \text{hap\_radix\_value}() \) after \( \text{hap\_expand}() \) to compute the \( \text{radix\_value} \) of each haplotype. Furthermore, the \( qsort() \) is replaced by the \( \text{radix\_qsort}() \) to sort the haplotypes using \( \text{radix\_value} \). Moreover, we have removed \( \text{cmp\_hap}() \) from \( \text{hap\_prior}() \) and \( \text{hap\_posterior}() \) and then placed \( \text{hap\_radix\_cmp}() \) after the \( qsort() \) to compare the \( \text{radix\_value} \).

There are few steps of Radix-2 Computation SNPHAP. Firstly, SNPHAP begins with an initial number of phases \( n_{\text{phase}} = 0 \). Secondly, the \( \text{hap\_expand}() \) expands the possible haplotypes. Thirdly, the \( \text{hap\_radix\_value}() \) is executed to compute the \( \text{radix\_value} \) of each haplotype. Next, the possible haplotypes are sorted by \( \text{radix\_qsort}() \). After that, \( \text{hap\_prior}() \) and \( \text{hap\_posterior}() \) that are EM algorithm are executed. Note that the input of \( \text{hap\_prior}() \) and \( \text{hap\_posterior}() \) are sorted and unsorted haplotypes, respectively. Then, the haplotype instances whose \( \text{Posterior\_prob} \) is smaller than \( \text{min\_prior}() \) will be eliminated. Finally, this algorithm will be executed repeatedly until \( n_{\text{phase}} \) is equal to \( L \). Therefore, this solution called \( \text{SEQ Radix SNPHAP} \) in short can boost the performance of SNPHAP.

### 3.5 OMP Radix Computation

The SEQ Radix SNPHAP have been parallelized (multithreaded) to OMP Radix SNPHAP using OpenMP 3.0. OpenMP provides the parallel directive so that a loop can be executed in parallel. Therefore, we have put parallel for directive in the following functions \( \text{hap\_radix\_value}(), \text{hap\_prior}() \) and \( \text{hap\_posterior}() \).

To parallelize \( \text{hap\_radix\_value}() \), some loops in this function are independent. Workloads can be divided and assigned equally. Note that each thread executes \( \text{hap\_radix\_value}() \) and passing the possible haplotype to this function. Figure 3 illustrates parallelization of \( \text{hap\_radix\_value}() \).

To modified \( \text{hap\_prior}() \), the objective of this function is to calculate the total probability and categorize the haplotypes. we can use reduction directive in OpenMP to parallel the summation and separate the works using groups of haplotypes that their pairs are the same as shown in Figure 4.

On the other hand, \( \text{hap\_posterior}() \) classifies the haplotypes using their genotypes. It can be separated the tasks using their genotypes and we can calculate log-likelihood using reduction directive. Figure 5 illustrates parallelization of \( \text{hap\_posterior}() \).

This solution called \( \text{OMP Radix SNPHAP} \) in short can reduce the run time of SNPHAP.

### 4 Results and Discussions

#### 4.1 Experiments Setup

All experiments are conducted on a 4-core AMD A6-3650 2.6-GHz with 16 GB DDR3-1333 MHz memory. The operating system is Ubuntu 10.04 LTS kernel 2.6.32-33-generic. The profiler is GNU GProf version 2.20.1 [9], the OpenMP Profiler is omp2 profiler tool [10] and the Compiler is GNU GCC version 4.4.3.

To show the effectiveness of our methods, the original, GCC-O3 Optimized SNPHAP, SEQ Radix SNPHAP are compared with OMP Radix SNPHAP in Table 1. We use SNAP [11] to generate all datasets and transform them as the input of SNPHAP.
4.2 Run time

The graphs of run time and haplotyping loci(L) of SNPHAP are plotted to test the performance of SNPHAP. Figure 6 (a), (b), (c), (d), (e) and (f) demonstrate the run time (in Seconds) of the original, GCC-O3, SEQ Radix SNPHAP and OMP Radix SNPHAP 4, 8 and 16 (T)threads, respectively. It can be noticed that the run time grows exponentially when dataset size of genotypes and loci increase.

Normally, both GCC-O3 and SEQ Radix SNPHAP can reduce the run time by an order of immensity compared with the original one. Note that GCC-O3 is slower than SEQ Radix SNPHAP in most instances. It is based on our methods alone and NOT further optimized by GCC. Faster run time of SEQ Radix SNPHAP proves that the complexity is reduced by $L^2$ to $L$ in Equation 1 and 2 to 6 and 7, respectively.

The results of OMP Radix SNPHAP 4T, 8T and 16T are similar. The 4T one is the fastest. However, they are faster than original, GCC-O3 and SEQ Radix SNPHAP.

Run time of OMP Radix SNPHAP is reduced by 2.2 times when compared with that of SEQ Radix SNPHAP. The reason is the tasks (threads) in OMP Radix SNPHAP are distributed to all four CPU cores. Nevertheless, Speedup of OMP Radix SNPHAP over SEQ Radix SNPHAP cannot achieve 4x because of the workload scheduling overhead and sequential code regions in this software.

4.3 Complexity Analysis

The original and SEQ Radix SNPHAP have been profiled using GNU GProf tool. Table 2 compares Big-O and time percentage of hotspot functions of original and SEQ Radix SNPHAP.

It can be observed that both run time complexity and
run time in Seconds are drastically reduced because 98% of the original run time is consumed by both \textit{hap.prior()} and \textit{hap.posterior()}. As a result of Big-O optimization using Radix-2 Computation, both functions spend about 85.8% of the total run time.

### 4.4 Speedup: Experimental vs. Theoretical

Speedup is the ratio between the original SNPHAP run time and the modified SNPHAP run time. The Speedups of GCC-O3 SNPHAP, SEQ Radix SNPHAP and OMP Radix SNPHAP 4, 8 and 16 T(threads) are plotted to compare the workload effects as shown in Figure 7 (a), (b) and (c) for \(L = 51, 101\) and 151 loci, respectively.

#### 4.4.1 SEQ Radix SNPHAP Speedup

It can be observed that the Speedups of all \(G = 500\) genotypes are fairly high compared to those of other datasets. Nonetheless, the maximum Speedup of SEQ Radix SNPHAP is 649\% at \(G = 500\) genotypes, \(L = 151\) loci.
loki as shown in Figure 7(c). It can be due to small data size compared with the size of cache resulting in fewer cache misses.

At \( L=101 \) loci, the Speedups of SEQ Radix SNPHAP (Red-Dashed line) are remarkably higher than \( L=51 \) loci but slightly lower than of \( L=151 \) loci on average in Figure 7(a), (b) and (c), respectively. On the theoretical view point, \( \text{Speedup}_{\text{radix}} \) of the SEQ Radix SNPHAP can be approximated as a ratio between Equation 1 and Equation 6 as

\[
\text{Speedup}_{\text{radix}} = \frac{O_o(L, M, G, n)}{O_{\text{radix}}(L, M, G, n)} \propto L \quad (9)
\]

where \( \text{radix} \) is the SEQ Radix SNPHAP, \( o \) is the original SNPHAP, \( G \) is the number of genotypes, \( M \) is the maximum iteration in EM algorithm, \( L \) is the number of haplotyping loci and \( n \) is the number of heterozygous loci of each genotype and proportional with \( L \). It can be observed in Equation 9 and the experimental results that \( \text{Speedup}_{\text{radix}} \) is proportional to \( L \).

### 4.4.2 OMP Radix SNPHAP Speedup

It can be noticed that Speedups of OMP Radix SNPHAP at 4T grow significantly at \( G = 500, 1,000, \) and \( 2,000 \) genotypes and increase slightly from \( G = 5,000 \) to \( 40,000 \) genotypes. OMP Radix SNPHAP at 4T is the highest speedups of \( L = 51, 101 \) and 151 loci. At 8T, the Speedups of OMP Radix SNPHAP rise remarkably from \( G = 500 \) to \( 5,000 \) genotypes and improve moderately at \( G = 10,000, 20,000 \) and \( 40,000 \) genotypes. The highest speedup of \( L = 101 \) loci at \( G = 40,000 \) genotypes reaches to 1,116%. Finally, the Speedups of OMP Radix SNPHAP 16T increase significantly for all \( G \) genotypes. The highest speedup of OMP Radix SNPHAP shoots up to 1,303% at \( G = 40,000 \) genotypes with \( L=151 \) loci.

In general, the Speedups of most of 4T OMP Radix SNPHAP are higher than those of 8 and 16T. It can be due to thread management overhead in OpenMP. Figures 8(a), (b) and (c) illustrate the \%Overhead of OMP Radix SNPHAP 4T, 8T and 16T of \( L = 51, 101 \) and 151 loci, respectively. The \%Overhead in OMP Radix SNPHAP of all threads shows similar trend. Those of OMP Radix SNPHAP 16T are the highest followed by those of 8T and 4T. The lower \%Overhead allows the higher Speedup. As a result, the appropriate number of threads should be determined accordingly to avoid the extra overhead.

### 5 Conclusion

In this paper, one of the fastest but exponentially complex bioinformatics tools named SNPHAP has been analysed in Big-O notations. Based on our average-case Big-O analysis, the Radix-2 Computation is applied resulting in the so-called SEQ Radix SNPHAP. Our OMP Radix SNPHAP is a parallel version of SEQ Radix SNPHAP using OpenMP 3.0 library. All experiments are conducted on a 2.6-GHz AMD A6-3650, 16GB-RAM Ubuntu Linux machine. The results show that OMP Radix SNPHAP can be 1,303% and 226% faster than the original and SEQ Radix SNPHAP, respectively. In addition, the theoretically derived Speedup of SEQ Radix versus the original SNPHAP is consistent with the experimental results.

For future work, an idea to utilize on-chip GPU can extend this framework further with OpenCL. This idea shall be investigated very soon.

### References


Figure 7. Speedups of SNPHAP on AMD A6-3650, $L = $ (a) 51 loci (b) 101 loci (c) 151 loci.

Figure 8. %Overheads of OMP Radix SNPHAP on AMD A6-3650, $L = $ (a) 51 loci (b) 101 loci (c) 151 loci.