Parallelizing the Individual Haplotying Assembly Problem

Robert J. clucas

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

Abstract:

Key words: Brach, Bound, GPU, Haplotyping, Simplex

1. INTRODUCTION

It is commonly accepted that all humans share $\sim$99% of the same DNA, however, small variations cause human beings to have different physical traits. Single nucleotide polymorphisms (SNPs), which are variations of a single DNA base from one individual to another, are believed to be able to address genetic differences. For diploid organisms, which have pairs of chromosomes, a *haplotype* is a sequence of SNPs in each copy of a pair of chromosomes. A *genotype* describes the conflated data of the haplotypes on a pair of chromosomes. Haplotypes are believed to contain more generic information than genotypes [1], however, obtaining haplotypes correctly is a difficult problem, which is broken into two subdomains: haplotype assembly and haplotype inference.
Haplotype inference uses the genotype of a set of individuals. The genotype data tells the status of each allele at a position, but does not distinguish which copy of the chromosome the allele came from. This negative aspects of this approach are that it cannot distinguish rare and novel SNPs [2], and there is no way of knowing if the inferred haplotype is completely correct.
Individual haplotype assembly uses fragments of sequences generated by sequencing technology to determine haplotypes. The fragments of a sequence come from the two copies of an individual's chromosome, the goal of the individual haplotyping problem is to correctly determine two haplotypes, where each haplotype corresponds to one of the two copies of the chromosome.
The haplotype assembly problem was proven to be NP-Hard [3]. The algorithms used to solve the problem are thus computationally complex and until recently, there was no practical exact algorithm to solve the problem using minimum error correction (MEC) [4], However, recently an exact solution was proposed by [5] which is capable of solving the MEC problem exactly, and can thus correctly infer all haplotypes from the fragment sequences. Due the NP-Hardness of the problem, the algorithm results in long run times - in the range of days for chromosomes with high errors rates. Using a parallel implementation of any of the proposed solutions could reduce the long run times, allowing useful haplotype information to be quickly inferred from the available datasets, having positive effects in fields such as drug discovery, prediction of diseases, and variations in gene expressions, to name a few.
Come back to introduction ...
Parallel programming makes use of devices which have many simple cores, but which can execute the same instructions on each of the cores at the same time. The effectiveness of parallel programming is dependant on the nature of the problem, as per Amdahl's law. The first attempts at parallel programming came from Graphics Processing Units (GPUs) which were used to render many pixels simultaneously. More recently, General-Purpose GPU (GPGPU) programming has become prominent with API's like CUDA and OpenCL, allowing access to GPUs from C and C++ programs.

2. HAPLOTYPE ASSEMBLY PROBLEM

This section will provide a brief overview of the haplotype assembly (HA) problem, and define the notation used through the rest of the paper. The input to the problem is a set of reads from a given genome sequence, where each read contains fragments from each of the two chromosomes which make up the genome sequence. These characters of a read consist of elements from a *ternary string*, where a ternary string has characters from the set {0, 1, -}. A value of 0 refers to the major allele at a site, a value of 1 to the minor allele, and a value of - to the lack of a read at the site, and is referred to as a *gap*. These reads are then combined to form a matrix, M, where each row of the matrix corresponds to a read.
Each column of the matrix is known as an SNP site. At each site, the data could be accurate, missing, or have error. The goal of the haplotype assembly problem is to determine a haplotype, H = {*h*,*h'*} from the matrix. The following terminology will be used to refer to properties of the matrix and the fragments.
For the input matrix M, the number of fragments is denoted by m, which is the number of rows in M. The number of SNP sites is denoted by n, which is the number of columns in M, while the jth site of the ith fragment is given by f_{ij}. Furthermore, two fragments are said to conflict if the following conditions are true:

- $f_{ik} \neq f_{jk}$ **and** $f_{ik} \neq$ '-' **and** $f_{jk} \neq$ '-'

Essentially this means that for two fragments i and j, if at an SNP site k, the reads do not have error and are not gaps, the reads have different values (fragment i has a value 0 at site k, while fragment j has a value 1 at site k, or vice versa).

Following the notion of a conflict, the *distance* between two fragments (or ternary strings) is denoted by d(f_i, f_j) is the total number of positions for which the two fragments f_i and f_j conflict. Furthermore, to understand some of the problem formulations from the fragment data, it is useful to define a *conflict graph* [6] G = { V, E }, where V corresponds to a fragment, and E corresponds to and edge between two fragments if they conflict. If the matrix M contains no errors, then none of the fragments from the same chromosome will conflict and G will be bipartate. Fragments from the same chromosome may conflict. However, if there are errors (as is usually the case) in M, G will not be bipartate. The haplotype assembly problem then requires the correction of G from a non-bipartate graph to a bipartate graph. There are numerous methods for solving the problem:

- **Minimum Fragment Removal (MFR) :** This invloves removing the least number of fragments from the input data such that the resultant graph G is bipartate. It is shown in [6] that this can be solved in polynomial time.
- **Weighted Minimum Edge Removal (WMER) [7] :** This method is a recent method, which requires determining the minimum number of edges to remove such that removal of the edges results in G being bipartate.
- **Longest Haplotype Reconstruction (LHR) :** This requires finding a set of fragments which, when they are removed from M, result in G being bipartate and the length of the resultant haplotypes being maximized [8].
- **Minimum Error Correction (MEC) :** This method involves correcting the minimum number of elements (sites for all fragments) in the matrix M which allow the graph G to be bipartate. Although being the most complex method, it is the most widely used method as it provides the highest accuracy rates. Only recently has an exact algorith for the MEC problem formulation been proposed which can provide an exact solution for all cases.

Due to the accuracy and more extensive previous work, the MEC method for solving the problem was decided upon as the method for which a parallel implementation will be proposed.

2.1 *Minimum Error Correction Implementations*

Much research has been done on solving the MEC formulation of the HA problem. The first exact algorithm for solving the problem was a branch and bound algorithm proposed by [9]. The algorithm creates a tree which covers the search space of all possible corrections. The tree is then traversed to find the best solution. They use an upper bound for when branching that allows branches to be abandoned when a better solution than the current best cannot result from further exploration of the branch, thus improving performance. However, this exact method has time complexity of $O(2^m)$, where m is the number of fragments in the input data, and hence cannot produce solutions for large problem sizes. They also provide a heuristic *genetic algorithm* (GA) to improve the computational time, which only requires run times up to three orders of magnitude faster than the branch and bound implementation. While the GA implementation gives very similar results to the branch and bound implementation, it is slightly worse.

A dynamic programming solution was proposed by [10] which addresses the run time problem for large input sizes. The algorithm has time complexity of $O(mk3^k + mlogm + mk)$, where k is the maximum number of SNP sites a fragment covers. In practice k is usually small, and results were shown small k (less than 100). The proposed dynamic programming solution had significant run time improvements over the solution proposed by [9]. However, for larger k values, the algorithm cannot solve the MEC case of the HA problem in a feasible amount of time.

More recently, [5] proposed an exact algorithm for solving the MEC problem. The proposed algorithm is the currently the only algorithm which can solve the HA problem for both the homozygous (the alleles at a site are identical) and hetrozygous (the alleles at an SNP site are different) case. Most other work assumes that the input fragment data is hetrozygous, which, while true for most of the SNP sites in the input matrix, is normally false for a small number of the SNP sites. The exact solution removes unnecessary data from the matrix, and partitions the matrix into smaller sub-matrices. The problem is then formulated as an *integer linear programming* (ILP) problem and solved as an optimization problem. The implementation was run using an Intel i7-3960X CPU, and the HuRef dataset required 15 days to solve for the general case, and 24h for the all-hetrozygous case. Heuristics methods are also proposed which speed up computation time by up to 15 times. However, the results are only shown for smaller input sizes, and the heuristic methods do not always determine the optimal solution. Furthermore, the solutions for the general case show lower MEC scores, meaning that the all-hetrozygous assumption is not always valid.

2.2 *Parallelization*

Of the MEC implementations discussed in Section 2.1, the branch and bound solution proposed by [9], and the ILP formulation of the problem proposed by [5] have the most potential for a parallel implementation. While there are other methods for solving ILP problems [11] the most common methods for solving ILP problems are:

- **Branch and bound :** The search space is divided into sub spaces, each of which is explored for the optimal solution. Bounds are enforced to ensure that sections of the sub spaces for which an optimal solution will not be found, will not be searched.
- **Branch and cut :** The proble'm is first solved without the integer constraints to find the optimal solution. Cutting planes are then used to divide the search space and then branch and bound is applied.

Thus both the MEC methods mentioned above can be solved using a branch and bound algorithm. The branch and cut algorithm involves using a branch and bound method as well as the simplex method, and is thus more complex, hence the choice was made to use the branch and bound algorithm.
Recently, the increased interest in general purpose GPU (GPGPU) programming has resulted in attempts being made at parallel implementations of the branch and bound algorithm for certain problems.
A hetrogeneous CPU-GPU implementation was proposed by [12] and was applied to the knapsack problem [13], which is known to be NP-Hard. Their results show that the GPU implementation is only good when the tree has a large number of nodes (> 5000). Furthermore, the tree is built using a *depth first* strategy to favour the parallel nature of the GPUs. The GPUs are used for the brancing and bounding computations, and returns a list of nodes to search for each iteration. This iterative procedure requires a lot of communication between the CPU and GPU, which is undesirable. However, a speedup of up to 9.27 times was achieved for larger problem sizes, validating the feasibility of the branch and bound algorithm for parallel implementation.

REFERENCES

[1] J. Stephens. “Haplotype Variation and Linkage Disequilibrium in 313 Human Genes.” *Science*, vol. 293, no. 5529, pp. 489–493, Jul. 2001. [Online]. Available at `http://dx.doi.org/10.1126/science.1059431` [Accessed 27 June 2015].

[2] D. He, A. Choi, K. Pipatsrisawat, A. Darwiche, and E. Eskin. “Optimal algorithms for haplotype assembly from whole-genome sequence data.” vol. 26, no. 12, pp. i183–i190, 2010.

[3] R. Lippert, R. Schwartz, G. Lancia, and S. Istrail. “Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem.” vol. 3, no. 1, pp. 23–31, 2002.

[4] P. Bonizzoni, G. Della Vedova, R. Dondi, and J. Li. “The Haplotyping problem: An overview of computational models and solutions.” *Journal of Computer Science and Technology*, vol. 18, no. 6, pp. 675–688, 2003. [Online]. Available at `http://dx.doi.org/10.1007/BF02945456` [Accessed 27 June 2015].

[5] Z.-Z. Chen, F. Deng, and L. Wang. “Exact algorithms for haplotype assembly from whole-genome sequence data.” vol. 29, no. 16, pp. 1938–1945, 2013.

[6] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz. “SNPs Problems, Complexity, and Algorithms.” In F. auf der Heide, editor, *Algorithms ESA 2001*, vol. 2161 of *Lecture Notes in Computer Science*, pp. 182–193. Springer Berlin Heidelberg, 2001. [Online] Available at `http://dx.doi.org/10.1007/3-540-44676-1_15` [Accessed 16 July 2015].

[7] D. Aguiar and S. Istrail. “HAPCOMPASS: A fast cycle basis algorithm for accurate haplotype assembly of sequence data.” *Journal of Computational Biology*, vol. 19, no. 6, pp. 577–590, June 2012. [Online] Available at `http://www.brown.edu/Research/Istrail_Lab/papers/hapcompass_jcb_draft.pdf` [Accessed 16 July 2015].

[8] R. Schwartz. “Theory and Algorithms for the Haplotype Assembly Problem.” *Commun. Inf. Syst.*, vol. 10, no. 1, pp. 23–38, 2010. [Online] Available at `http://projecteuclid.org/euclid.cis/1268143371` [Accessed 16 July 2015].

[9] R.-S. Wang, L.-Y. Wu, Z.-P. Li, and X.-S. Zhang. “Haplotype reconstruction from SNP fragments by minimum error correction.” vol. 21, no. 10, pp. 2456–2462, 2005.

[10] M. Xie, J. Wang, and J. Chen. “A Practical Exact Algorithm for the Individual Haplotyping Problem MEC.” In *BioMedical Engineering and Informatics, 2008. BMEI 2008. International Conference on*, vol. 1, pp. 72–76. May 2008.

[11] L. Galli. “Algorithms for Integer Programming.”, December 2014. [Online] Available at `http://www.di.unipi.it/optimize/Courses/RO2IG/aa1415/IP_algorithms.pdf` [Accessed 27 June 2015].

[12] A. Boukedjar, M. Lalami, and D. El-Baz. “Parallel Branch and Bound on a CPU-GPU System.” In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pp. 392–398. Feb.

[13] M. Kedia. “Dynamic Programming.”, October 2005. [Online] Available at `http://www.cs.cmu.edu/afs/cs/academic/class/15854-f05/www/scribe/lec10.pdf` [Accessed 27 June 2015].