2017 IEEE International Parallel and Distributed Processing Symposium Workshops

Parallel NGS Assembly Using Distributed Assembly Graphs Enriched with Biological Knowledge

Julia D. Warnke-Sommer
Department of Pathology and Microbiology
University of Nebraska Medical Center
Omaha, NE 68198, USA
julia.warnke@unmc.edu

Hesham H. Ali
College of Information Science and Technology
University of Nebraska Omaha
Omaha, NE 68182, USA
hali@unomaha.edu

***Abstract*—High performance computing has become essential for many biomedical applications as the production of biological data continues to increase. Next Generation Sequencing (NGS) technologies are capable of producing millions to even billions of short DNA fragments called reads. These short reads are assembled into larger sequences called contigs by graph theoretic software tools called assemblers. High performance computing has been applied to reduce the computational burden of several steps of the NGS data assembly process. Several parallel de Bruijn graph assemblers rely on a distributed assembly graph. However, the majority of assemblers that utilize distributed assembly graphs do not take the input properties of the data set into consideration to improve the graph partitioning process. Furthermore, the graph theoretic foundation for the majority of these assemblers is a distributed de Bruijn graph. In this paper, we introduce a distributed overlap graph based model upon which our parallel assembler Focus is built. The contribution of this paper is three-fold. First, we demonstrate that the application of data specific knowledge regarding the inherent linearity of DNA sequences can be used to improve the partitioning processes for distributing the assembly graph. Second, we implement several parallel graph algorithms for assembly with greatly improved speedup. Finally, we demonstrate that for metagenomics datasets, the graph partitioning provides insights into the structure of the microbial community.**

Keywords—next generation sequencing; assembly graph; high performance computing; algorithms

I. Introduction

Current biomedical technologies are producing massive amounts of data at increasingly faster rates. In particular, next generation sequencing technologies are capable of producing gigabases and even terabases of genetic sequence data in a single run. Next generation sequencing is used to determine the order of nucleotides in a given strand of DNA. This is accomplished by a fragmenting sample DNA sequence into a library of short segments. These DNA segments are then sequenced to obtain the ordering of their nucleotide bases. The resulting sequences, ranging from $10^2 – 10^4$ base pairs (bps) in length depending on which sequencing technology was used, are called reads. Illumina technologies such as the HiSeq X series are able to output up to 1800 Gb of sequence data with 150 bp long reads [1]. PacBio systems are currently able to produce reads exceeding 10 kb in length with up to 1 Gb of data produced per run [2].

The relatively short lengths of the produced reads in comparison to the input sample DNA strand to be analyzed makes it difficult to extract any information from the reads individually. However, the reads are produced at a high coverage of the original DNA sequence, such that many of the reads overlap. These overlap relationships can be used to order and merge the reads into a representation of the original target sequence. Specialized software tools called assemblers are used to order and merge the reads into contiguous stretches of sequence called contigs. Most of these assembly tools rely on graph theoretical approaches to model the reads and their overlap relationships. Two primary approaches exist for modeling next generation reads. Typically, the overlap graph based assemblers [3] model each read as a node in an overlap graph and overlap relationships as edges in the overlap graph. Edges can be weighted by criteria such as the length of the overlap shared between the two reads represented by the endpoints of that edge. String graphs [4] are an extension of the basic overlap graph formed by removing transitive edges and other redundant information. The de Bruijn graph assemblers [3] split each read into its set of component k-mers. Each unique k-mer becomes an edge in the de Bruijn graph. The left and right k-1-mers of each unique k-mer becomes an edge in the de Bruijn graph. In both graph approaches an ordering of the reads is obtained by traversing the assembly graph.

Due the massive size of next generation sequencing data sets, the assembly graph can become extremely large and difficult to process. To address this issue, several parallel assemblers have been developed that distribute the assembly graph across processors. De Bruijn graph assemblers such as AbySS [5], Ray [6], PASHA [7], and the SWAP-Assembler [8] distribute the de Bruijn graph by assigning the generated k-mers to different processors. MPI communication is then used for parallel graph simplification and traversal of the distributed de Bruijn graph. The PCAP [9] assembler is an overlap-layout-consensus assembler that uses parallel processing to speed up read overlap detection and scaffold processing. HipMer [10] is a parallel version of the Meraculous assembler designed in particular for extreme scale analysis. Map reduce has been applied to process a string graph assembly model [11]. Other assemblers such as MEGAHIT [12] take advantage of GPU architecture for parallel assembly.

Due to the linearity of DNA, reads covering a contiguous region in the sequence will form a cluster within the assembly graph. The parallel assemblers discussed in the previous paragraph do not take this input characteristic of the read data

978-0-7695-6149-3/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.143

into consideration during the graph partitioning process. In many applications, high performance computing is primarily viewed as a means for speeding up computational solutions for the purpose of producing faster results for downstream analysis. However, we propose that high performance computing is not just a method for obtaining faster results but can be applied in intelligent ways to extract meaningful information from big data. It is very tempting to use naïve methods for high performance computing but by using domain knowledge one may get rewarded.

Previously, we introduced a novel hybrid graph model for assembly that integrates multiple levels of an iteratively coarsened overlap graph to capture the structural features of the input data set, such as repetitive or transposable sequence regions, in a concise yet feature rich approach. This model forms the foundation of an assembly and analysis tool called Focus [13]. Here we describe a distributed version of this graph model and associated parallel graph assembly methods.

The contributions of this paper are three-fold. 1) We show that the integration of previous knowledge of the structure of the input data set can benefit the graph partitioning algorithm's performance. Due to the linear nature of the DNA sequence, groups of nodes representing contiguous sequence regions can simply be assigned to the same graph partition with minimal processing. Results show that this dramatically improves runtime for all data sets and the edge cut of the partitioning in the majority of cases. 2) We develop several parallel graph algorithms applied on our distributed graph model for sequence assembly. Results demonstrate greatly improved speedup times for the distributed algorithms, which include graph cleaning and graph traversal methods. 3) Finally, we demonstrate that we are rewarded for integrating domain knowledge into our graph partitioning process. Insights into metagenomics community structure can be obtained from the resulting graph partitioning. For metagenomics read data sets, where the underlying DNA sample is obtained from a community of organisms, it is shown that related species tend to cluster within the same graph partition. This information may be useful to researchers trying to identify the composition of a given metagenomics data set. The concept of knowledge integration into naive algorithms may have important extensions to additional methods in high performance computing.

II. Focus Assembler Overview

The Focus assembly algorithm is composed of six component steps. These steps include read data preprocessing, read alignment, multilevel graph set generation, hybrid graph set generation, hybrid graph trimming, and hybrid graph traversal. These steps are discussed briefly in the following sections. Please see [13] for an in-depth discussion of the Focus algorithm.

A. Read Data Preprocessing

The Focus assembler accepts both fasta and fastq data as input. Each read is processed individually. First the 5' end and 3' end of a given read is trimmed by fixed lengths specified by the user to remove any tags or adaptors. Then each read is trimmed from its 3' end based on quality values. A sliding window w of length l is applied starting from the 3' end and moving to the 5' end with a user specified step size of k. The average quality value for the sliding window is calculated at each step. Once the average quality value becomes greater than a minimum threshold q specified by the user, the read is trimmed from the right end of the sliding window to the 3' end of the read. The reverse complement of the trimmed read is generated and added to the read dataset. Once the read trimming and reverse complement generation is complete, the read data set is split into a number of subsets specified by the user that can be processed in parallel by the read alignment module. For more information regarding parallel read alignment of Focus please see [13].

B. Parallel Read Alignment

Each pair of read subsets produced by the read preprocessor is analyzed for determining read overlap relationships between reads. For parallel read alignment, each pair of read subsets can be sent to a different processor for independent analysis. A reference read sub set R_r is indexed by a suffix array S_r [14]. Each read in the query read sub set R_q is visited successively. Let r_q be a read that is currently being visited in R_q. The read r_q is decomposed into its component k-mers, where k is specified by the user. Each k-mer is used to query the suffix array S_r to search for short k-mer matches in the reads of R_r. Let r_r be a read in R_r. If r_r contains number of k-mer hits greater than a specified threshold, then r_r will be extracted from R_r and aligned to r_q using banded Needleman-Wunsch alignment. If the alignment produces an overlap relationship, where the prefix of r_r is the suffix of r_q or vice versa or where one read is completely contained in the other, then the overlap relationship is evaluated against minimum user specified thresholds. If the overlap meets minimum thresholds for alignment length and alignment minimum identity, then the ids of the query and reference reads are recorded to file along with the alignment length and minimum alignment identity.

C. Multilevel Graph Set

The initial overlap relationships produced by the read alignment algorithm are loaded into an overlap graph G_0. Let $V(G_0)$ be the node set of G_0 and $E(G_0)$ be the edge set of G_0. $|V(G_0)|$ is the number of nodes in G_0 and $|E(G_0)|$ is the number of edges in G_0. In this overlap graph each node represents a sequencing read and edges represent the overlap relationships between reads. Each edge records both the alignment length and alignment identity. The weight of each edge in G_0 is set to its recorded alignment length. This graph is extremely large, making both assembly and any graph-based data analysis processes difficult. Graph coarsening [15] reduces the overlap graph G_0 by finding a matching M on G_0 and then merging pairs of nodes that are the endpoints of the edges in M to form G_1. Heavy edge matching [15] and node merging is applied on G_1 to form G_2. This process produces a multilevel graph set $G = \{G_0, G_1, \ldots G_n\}$, where $|V(G_0)| \leq |V(G_1)| \leq \ldots \leq |V(G_n)|$. An example of the multilevel graph set is shown in Fig. 1.A.

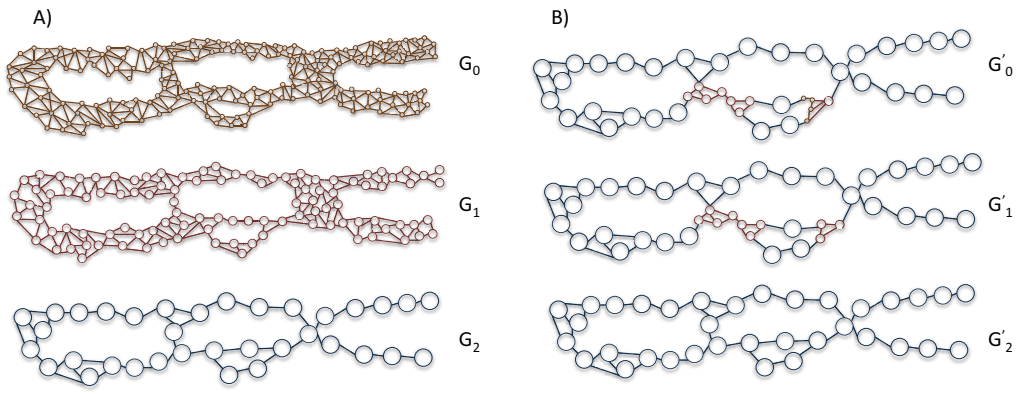

Fig. 1. Multilevel graph set and hybrid graph set. (A) Heavy edge matching and node merging is used to create a set of multilevel graphs G_0, G_1, G_2, where $|V(G_0)| \geq |V(G_1)| \geq |V(G_2)|$. (B) Nodes are integrated from different levels of the multilevel graph set to create the hybrid graph set G'_0, G'_1, G'_2. The graph G'_2 contains all of the best representatives from G'_2 as well as the non-representative nodes from G'_2. The graph G'_1 contains all of the best representative nodes from G_2, G_1 as well as the non-representative nodes from G_1. The graph G'_0 contains all of the best representative nodes selected from all graph levels and is called the hybrid graph.

D. Hybrid Graph Set

The result of the graph coarsening process is the multilevel graph set $G = \{G_0, G_1, \ldots G_n\}$. In this multilevel graph set for each graph G_i where $i \geq 1$, every node in G_i corresponds to a cluster of nodes in G_0 and therefore a cluster of reads. It would be optimal if each read cluster represented by a node in $G_1 \ldots G_n$ assembled into a contiguous contig. However, this is not likely, as certain genomic regions may not be best represented at all levels in the multilevel graph set. For example, a repetitive region might be over reduced in later graph levels. To address this issue, best representative nodes are integrated from multiple levels of the multilevel graph set to create a hybrid graph set $G' = \{G'_0, G'_1, \ldots G'_n\}$. A best representative node is defined to be a node that is selected from the most reduced graph as possible whose corresponding read cluster assembles into a contiguous contig. The graph G'_0 is denoted as the hybrid graph as it will contain all of the best representative nodes selected from G_n, G_{n-1}, … G_0. Fig. 1.B provides an example of a hybrid graph set obtained from a multilevel graph set.

E. Graph simplification and error correction

Focus follows the graph trimming algorithms introduced by previous assembly algorithms [16] to remove short dead end paths and small bubbles from the hybrid graph. We have adapted our graph-trimming module to be run in parallel on the distributed hybrid graph. More details regarding our graph trimming methods and their parallel implementations will be discussed under Sections V-A, V-B, and V-C of this paper.

F. Maximal Path Extraction and Contig Construction

For the purpose of contig construction, maximal paths are extracted from the hybrid graph. The contigs are constructed following the order of the read clusters represented by the nodes in the maximal path. Section V-D provides more details regarding the implementation of the maximal path algorithm and its parallel application on the distributed hybrid graph.

III. Graph Coarsening

The Focus algorithm follows the approach introduced by Karypis and Kumar [15] for partitioning large graphs. This multilevel approach first coarsens an input graph to form a multilevel graph set $G = \{G_0, G_1, \ldots G_n\}$, where $|V(G_n)| \leq |V(G_{n-1})| \leq \ldots \leq |V(G_0)|$. The most reduced graph G_n is then partitioned into two approximately equal halves while attempting to minimize the edge-cut between the partitions. The final step is un-coarsening. During this stage the partition on G_n is projected recursively through graph levels G_{n-1} to G_0. At each projection onto a given graph G_i in the spectrum, a refinement algorithm refines the projection.

The approach described above requires that the reduced graph be un-coarsened completely to obtain a partitioning on the original overlap graph G_0. Refinement algorithms can become fairly costly for finer graph levels that have a large number of edges and nodes. However, if the data that a given graph represents have intrinsic characteristics specific to the data domain, then that domain specific knowledge can be used to improve the partitioning process. Since the nodes of the overlap graph represent consecutive regions in the genome, it is likely that many groups of nodes representing the same genomic region can be safely assigned to the same partition without dramatically impacting the overall edge-cut between partitions.

The multilevel graph set represents complete un-coarsening. In contrast, the hybrid graph set represents a compromise between complete un-coarsening and incorporation of biological knowledge. Recall nodes in hybrid graph represent clusters of nodes in the original overlap graph. If there is biological knowledge regarding which nodes should cluster together then there is no need to completely un-coarsen the graph. The representative nodes in the hybrid graph represent clusters of nodes whose corresponding read clusters have already been determined to form a contiguous contig. It would be reasonable to assume that a cluster of nodes in the

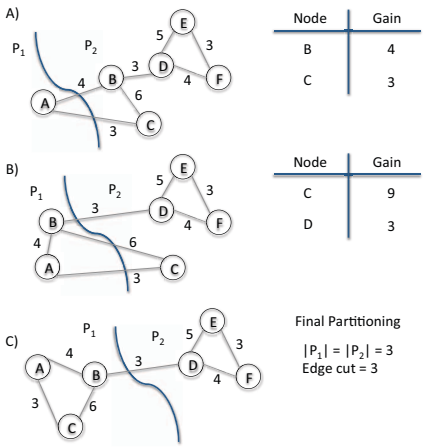

Fig. 2. Greedy graph growing. (A) Node *A* is placed into partition P_1. The gains of *A*'s adjacent neighbors are calculated. (B). The node *B* has the greatest gain and is placed into P_1. The gains are updated. (C). The final node *C* is added to P_1 according to its gain. The partitions are now evenly weighted so the greedy graph growing is terminated.

original overlap graph that maps to a single node in the hybrid graph would belong to the same partition.

This paper demonstrates that a partitioning on the hybrid graph will be as good as of a partitioning on the overlap graph in terms of edge cut and has a much faster running time. This partitioning found on the hybrid graph can then be simply mapped to the original overlap graph and multilevel graph set.

IV. Graph Partitioning

The graph partitioning process proceeds as following on a coarsened graph set to create *k* partitions, where $k = 2^i$ and *i* is an integer number as recursive bisectioning is used to partition the graph. A partitioning P on a graph G is defined as $P = P_1, P_2, \ldots P_k$, where $P_1, P_2, \ldots P_k$ are disjoint sets of nodes of in V(G) and $P_1 \cup P_2 \cup \ldots \cup P_k = V(G)$. Each $P_1, P_2, \ldots P_k$ is a partition of V(G). First, an initial partitioning with two equal halves is created on the most reduced graph, G_n, by a greedy graph growing algorithm. This initial partitioning is refined with the Kernighan-Lin [17] refinement algorithm. The refined partitioning is projected onto the next graph level G_{n-1}. This projected partitioning is then refined by the Kernighan-Lin algorithm. The projection and subsequent refinement of the partitioning is continued until the final graph G_0 is partitioned. Each generated partition is recursively partitioned, projected, and refined until G_0 is partitioned into the desired number of sections, *k*. After the *k* partitions on the multilevel graph set $G_0, G_1, \ldots G_n$ are obtained, each graph level is refined by a heuristic k-way Kernighan-Lin algorithm. The graph partitioning method is discussed in detail in the following sections.

A. Greedy Graph Growing

The approach by Karypis and Kumar [15] is followed with some modifications to customize the greedy graph growing algorithm to the data characteristics. This algorithm attempts to greedily add nodes to initial partitions P_1 and P_2 on the most reduced graph G_n. Define the gain g_{vz} of a node v_z in graph G_n by placing v_z into a partition P_i as:

$$g_{v_z,P_i} = \sum_{e(v_z,v)\epsilon E(G_n),v\epsilon P_i} w(v_z,v) - \sum_{e(v_z,v)\epsilon E(G_n),v\notin P_i} w(v_z,v)$$

where $w(v_z, v)$ is the weight of the edge $e(v_z, v)$. The greedy algorithm begins by choosing a random seed node v_s to insert into the partition P_1. The gains of the nodes adjacent to v_s are calculated and the nodes placed into a priority queue according to their gains. The nodes whose gains have been computed are the horizon of the currently growing partition. The node with the greatest gain v_g is removed from the priority queue and added to P_1. The gains of the neighbors of v_g are calculated and the nodes are added to the priority queue or, if they are already in the queue, their gains are updated.

The growing graph algorithm in this paper attempts to maintain balanced node weight and edge weight across the partitions. This is done so that the distribution of edges and nodes will be balanced across the final partitioning. The edge weight of a given partition P_i is defined as the following.

$$ew_{partition}(P_i) = \sum_{e(v_1,v_2)\,|\,v_1\epsilon P_i \wedge v_2 \epsilon P_i} w(v_1,v_2)$$

where $w(v_1, v_2)$ is the weight of the edge $e(v_1, v_2)$. If $ew_{partition}(P_1)$ becomes greater than $1.03ew_{partition}(P_2)$, then the partition growing of P_1 is terminated. The value of 1.03 places an upper bound of 3% on edge weight balance between the partitions. A new seed v_s is chosen and added to P_2. The partition P_2 is grown according to the method introduced previously. If $ew_{partition}(P_2)$ becomes greater than $1.03ew_{partition}(P_1)$, than the partition growing of P_2 is terminated and a new seed is chosen for P_1. This alternating partition growing is continued until $nw_{partition}(P_1) \geq 0.5nw_{graph}(G_n)$ or $nw_{partition}(P_2) \geq 0.5nw_{graph}(G_n)$; the remaining nodes are then placed into the smaller partition. Thus each partition will have approximately equal node weight. The terms $nw_{partition}$ and nw_{graph} are the node weight of a given partition P_i or graph G_i, respectively, and are defined as following.

$$nw_{partition}(P_i) = \sum_{v\epsilon P_i} nw(v)$$

$$nw_{graph}(G_i) = \sum_{v\epsilon N(G_i)} nw(v)$$

Any remaining nodes are added to the partition with the least node weight. Thus the greedy growing algorithm will produce an initial partitioning $P = P_1, P_2$, while attempting to balance the partitions according to both node and edge weight. An illustration of the greedy graph growing algorithm can be found in Fig. 2.

B. Kernighan-Lin Refinement Algorithm

Once the initial partitioning $P = P_1, P_2$ is created by the greedy graph growing algorithm, it is then refined by the Kernighan-Lin refinement algorithm. The Kernighan-Lin refinement algorithm relies on node swapping to improve the edge cut of a partitioning. First we provide definitions need for the Kernighan-Lin algorithm. Given a node v_z in a given partition P_i on G_n, the *external cost* E_{v_z} and the *internal cost* I_{v_z} of v_z is defined as follows.

$$E_{v_z} = \sum_{e(v_z,v)\epsilon E(G_n), v\notin P_i} w(v_z, v)$$

$$I_{v_z} = \sum_{e(v_z,v)\epsilon E(G_n), v\in P_i} w(v_z, v)$$

The D value of v_z, written as D_{v_z}, is defined as $D_{v_z} = E_{v_z} - I_{v_z}$. Let $v_z \in P_1$ and $v_y \in P_2$. The gain $g_{v_z v_y}$ of swapping v_z and v_y is defined as $D_{v_z} + D_{v_y} - 2w(v_z, v_y)$ if $e(v_z, v_y) \in E(G_n)$ and $D_{v_z} + D_{v_y}$ if $e(v_z, v_y) \notin E(G_n)$.

The Kernighan-Lin algorithm is iterative in nature. Given a partitioning $P = P_1, P_2$ such that $|P_1| \approx |P_2|$ on the graph G_n, all nodes in P_1 and P_2 are initially unlocked, meaning that they can be exchanged to a different partition. The algorithm then identifies the pair of nodes (v_z, v_y) that have the greatest gain $g_{v_z v_y}$. This pair of vertex is then swapped between partitions and v_z and v_y are locked. The D values of the remaining unlocked nodes are updated. The selection and locking of the pair of nodes with the greatest swapping gain and subsequent updating of the remaining node's D values is continued until there are no longer any unlocked node pairs remaining. For each pair $(v_{zk}, v_{yk})_k$ of nodes that was selected, the partial sum $S(v_{zk}, v_{yk})_k = \sum_{i=0}^{k} (g_{v_{zk} v_{yk}})_i$ of the total gain is computed. The pair of nodes such that the partial sum is maximized is identified. All node pair exchanges that occurred after this node pair are undone. This is a single pass of the Kernighan-Lin algorithm. Multiple passes are conducted until the maximal partial sum of gains is zero, meaning that no more improvement on the partition edge cut can be found. A straightforward implementation of this algorithm would have a complexity of $O(n^3)$, where $n = |P_1| + |P_2|$, as it would require $(\frac{n}{2})^2$ to find the best node pair to exchange and $\frac{n}{2}$ node pairs exchanged. This leads to a complexity of $O(\frac{n}{2}(\frac{n}{2})^2) = O(n^3)$.

Using appropriate data structures and strategies, the complexity of the Kernighan - Lin algorithm can be reduced to $O(n^2 \log(n))$. This is the approach taken in this paper and will be discussed briefly. The nodes assigned to each partition P_1 and P_2 are inserted into two priority queues Q_1 and Q_2, respectively, according to their D values, where nodes with greater D values have greater priority within the queue. It is reasonable that the node pair that has the greatest gain value will be near the top of the priority queues Q_1 and Q_2. Indeed, only a subset of the node pairs need to be explored from Q_1 and Q_2. Pairs of nodes (v_{zk}, v_{yk}) are evaluated in decreasing order of $Dv_{zk} + Dv_{yk}$ and the gain $g_{v_{zk} v_{yk}}$ is computed. The maximum $g_{v_{zk} v_{yk}}$ encountered thus far as the node pairs are being sequentially evaluated is recorded as the current g_{max}. Once a pair of nodes (v_{zk}, v_{yk}) is found such that $Dv_{zk} + Dv_{zy} \leq g_{max}$ the search through the nodes pairs is terminated. The node pair (v_{zk}, v_{yk}) such that $g_{v_{zk} v_{yk}}$ is the current g_{max}, is selected as the node pair to exchange. This approach requires $\frac{n}{2}\log\frac{n}{2}$ to sort the nodes in the priority queues. Since $\frac{n}{2}$ node exchanges are made, the total complexity is $O(\frac{n}{2}(\frac{n}{2}\log(\frac{n}{2}))) = O(n^2 \log(n))$. The diagonal scanning approach in [18] is used to determine the ordering of node pairs evaluated from Q_1 and Q_2. An additional strategy is also utilized from [15] to speed up run time. Let S_{max} be the maximal partial sum of gains that has been encountered thus far as the node pairs are being evaluated. If S_{max} does not increase for fifty node exchanges, then the node exchanges are terminated. As before, all node-pairs exchanges that occurred after S_{max} are undone. Figure 3 provides an example of the Kernighan-Lin algorithm.

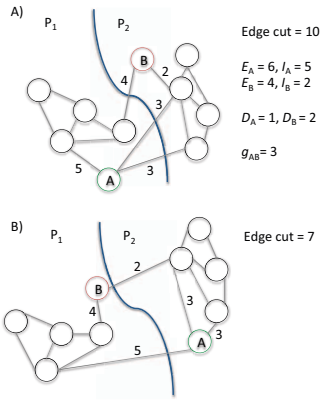

Fig. 3. Kernighan-Lin. (A) Let nodes *A* and *B* be the node pair with the greatest gain. The *E*, *I*, and *D* values are also shown for *A* and *B*. (B) The nodes *A* and *B* are swapped. The updated edge cut is shown.

C. Projection of the Partition

After the partitioning $P = P_1, P_2$ is found on G_n, it is projected iteratively onto G_{n-1}, G_{n-2} … G_0. If v_z is a node in $N(G_n)$ assigned to a partition P_i, then its component child nodes in $N(G_{n-1})$ will be assigned to P_i on G_{n-1}. Once a projected partitioning $P = P_1, P_2$ is established on G_{n-1} from G_n, it is refined by the Kernighan-Lin algorithm. The partitioning on G_{n-1} is then projected onto G_{n-2} and refined.

Recursive bisection is applied to create a partitioning $P = P_1, P_2, \ldots P_k$ in $\log_2(k)$ steps, using the greedy graph growing and Kernighan-Lin algorithms described earlier. First G_n is partitioned into two equal partitions that are projected and refined onto G_{n-1}, G_{n-2}, … G_0. These two partitions are then partitioned into four partitions, which are projected and refined. These partitions continue to be recursively bisected until k partitions are created. Observe that there is a natural

parallelism as the coarsened graph set is being recursively bisected. At each recursive bisection step $i = 0 \ldots \log_2(k)$-1 there are 2^i partitions that can be processed concurrently. The number of partitions that will need to be bisected in a given step i is 2^i. Thus if 2^i processors are assigned at each ith step, the partitioning $P = P_1, P_2, \ldots P_k$ on $G_n, G_{n-1}, \ldots G_0$ can be generated in $\log_2(k)$ steps in contrast to k-1 number of steps. Thus the upper bound on parallelism is $O(\frac{k-1}{log_2(k)})$.

D. Global K-Way Kernighan-Lin Refinement Algorithm

After the multilevel recursive bisection with the greedy graph growing and Kernighan-Lin algorithm is complete, there is a partitioning $P = P_1, P_2, \ldots P_k$ on $G_n, G_{n-1}, \ldots G_0$. This paper follows the global Kernighan-Lin heuristic approach in [19] to perform a k-way refinement on P for each graph level. Let G_i be the graph level currently begin refined and $P = P_1, P_2, \ldots P_k$ be the partitioning on G_i. First, the boundary nodes in P are identified. A given node v_z in G_i is a boundary node in P if $E_{v_z} \neq 0$. The boundary nodes in P are inserted into a priority queue Q according to their gains. Here the gain g_{v_z} of a node v_z is calculated as $E_{v_z} - I_{v_z}$. The nodes in Q are then evaluated in the order of descending gain. Let v_z be the current node being evaluated and P_i be the partition that v_z is currently assigned to. Let $E_{v_z P_j}$ be the *external cost* of moving v_z to a partition P_j given by $E_{v_z P_j} = \sum_{e(v_z,v)\epsilon E(G_i), v_z \notin P_j, v \in P_j} w(v_z, v)$. The node v_z is moved to a partition P_j such that $E_{v_z P_j}$ is the maximum *external cost* out of all of the *external costs* calculated for each partition. Partition balancing conditions must also be met; a node will not be moved to a partition P_j from a partition P_i if $|P_j| \geq 1.03|P_i|$. As before, let S_{max} be the maximal partial sum of gains that has been encountered thus far as nodes have been moved between partitions. The k-way refinement algorithm terminates after fifty moves have been made and no improvement to S_{max} occurs. All node moves that occurred after the S_{max} was identified are undone. This is a single pass of the global k-way Kernighan-Lin heuristic algorithm. Multiple passes are conducted until no more node move improvements can be made. Each graph level can be refined independently by the global k-way Kernighan-Lin algorithm with multiple processors.

V. Distributed Graph Algorithms

In this section, distributed graph algorithms for graph simplification, error correction, and graph traversal are introduced.

A. Transitive edge reduction

Let $P = \{P_1, P_2, \ldots P_k\}$ be a partitioning on the hybrid graph G'_0. Each partition P_i is assigned to a different worker processor. In parallel, each worker processor iterates through the nodes assigned to its partition sequentially. Following the approach described by [4], the edges of each node in $V(G'_0)$ are examined to identify transitive edges. Each transitive edge is recorded for removal from the hybrid graph. After each worker processor completes the transitive reduction of its partition, the master process removes the recorded transitive edges from the hybrid graph. If a transitive edge crosses a graph partition, then both of the partitions to which the endpoints of that edge are assigned will record that edge as being transitive. The master process will remove the recorded transitive edge from the hybrid graph.

B. Containment removal

The graph simplification process removes redundant information in the hybrid graph G'_0. This includes nodes that represent contigs whose sequences are contained within longer contigs represented by nodes in the hybrid graph. To remove these nodes, each worker processor again iterates through the nodes assigned to its partition sequentially. Let v_y be a node that is currently being evaluated. The contig c_y corresponding to node v_y will be aligned to the contigs represented by the neighboring nodes of v_y. If c_y is found to be contained within a neighboring contig sequence, then v_y will be recorded for removal from the hybrid graph. This alignment process is also used to detect false-positive edges within the hybrid graph. If the contigs corresponding to the endpoints of an edge in G'_0 have an overlap length less than 50 bps, then that edge will be recorded for removal from G'_0. After each worker process completes the evaluation of its assigned nodes, then the master process removes those nodes from the hybrid graph.

C. Error removal

Focus employs short dead end path trimming and bubble removal techniques utilized by many assembly tools and described in [16]. Each worker node evaluates its own partition and explores each node to see if it is part of a dead end path or bubble. Nodes that are a part of a short dead end path or bubble are recorded. These nodes are removed from the hybrid graph by the master process.

D. Graph traversal

Focus constructs contigs by recovering all maximal paths in the hybrid graph. Each worker processor evaluates the nodes in its partition sequentially. Let v_y be a node that is currently being evaluated and is not already in a path. The node v_z becomes the seed for a new path and is first extended by out-edges. If v_y has a single out-edge $e = (v_y, v_z)$ and e is the only in-edge of v_z, then v_z is added to the growing path. If v_z is not part of the same partition as v_y or the above criteria is not met, the path extension with out-edges is terminated. The path is then extended by out-edges from v_z by the same method used for v_y. Path extension by out-edges continues until no additional nodes can be added to the path. The path is then extended from v_y by in-edges following the same approach as the extension by out-edges. Once all of the worker processes have finished path extension, the master process joins the sub-paths produced by each worker process. Let p_1 and p_2 be two sub paths such that the right endpoint of p_1 has an out-edge incident to the left endpoint of p_2. If the left endpoint of p_2 has no other in-edges incident to it, then the master processor joins p_1 and p_2. If p_2 has other incident in-edges, then the two paths are not joined. The produced paths are used to construct contigs, which are output by the assembler.

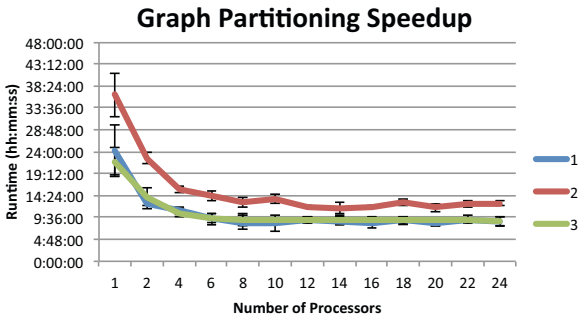

Fig. 4. Graph partitioning speedup. Speedup curve for graph partitioning on the hybrid graph sets for the three read data sets (blue, red, green). The number of graph partitions generated by each run was held constant at 16.

TABLE I. DATA SET CHARACTERISTICS

Data set	SRA ID	Data set size (GBases)	Read length (bps)
1	SRR513170	5.02 Gb	100 bp
2	SRR513441	4.93 Gb	100 bp
3	SRR061581	4.97 Gb	100 bp

VI. RESULTS

This results section is organized into five subsections. First the data sets and computational environment are described in detail. In the next section, an experiment is performed to demonstrate the natural parallelism that can be exploited during the graph partitioning process. In Section VI-C, it is shown that incorporating biological knowledge improves the partitioning process. Section VI-D describes the speed up curve for trimming and graph traversal algorithms implemented on the distributed hybrid graph. Finally, the results section is concluded by showing that the partitions on the hybrid graph are able to capture features of metagenomics community structure.

A. Data sets and computational environment

For the purpose of evaluating the graph partitioning and associated parallel processes for the Focus assembler algorithm, three Illumina read data sets were downloaded from the NCBI Sequence Read Archive (SRA) [20].

These Illumina read data sets were sequenced from the gut microbiome of healthy individuals. Details regarding these data sets can be found in Table 1. All experiments were run on the Crane high performance computing cluster at the Holland computing center [21]. This cluster has 452 nodes with two Intel Xeon E5-2670 2.60GHz processors/16 cores per node. Each node has 64 GB of memory.

Prior to graph partitioning, the Focus algorithm was first used to perform pairwise alignment of the reads with a minimum overlap length of 50bps and minimum alignment identity of 90%. Focus was then applied to produce both multilevel and hybrid graph sets for each read data set.

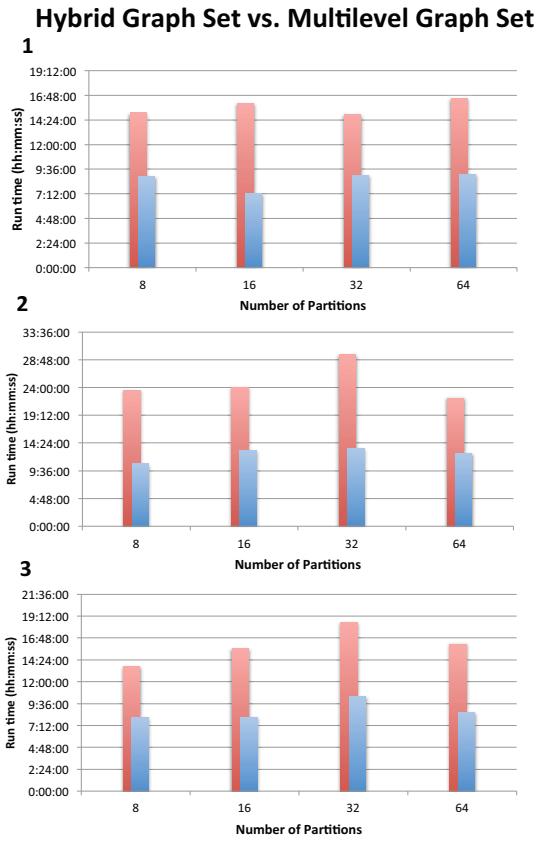

Fig. 5. Hybrid graph set vs. multilevel graph set. Each hybrid and multilevel graph set was partitioned four times with 8, 16, 32, and 64 final partitions. For each graph partitioning, $2^{log_2(k)-1}$ processors were assigned, where k is the final number of partitions. The runtime is shown for partitioning the hybrid graph set (blue) and multilevel graph set (red) for each read set (1, 2, and 3).

TABLE II. EDGE CUT FOR THE OVERLAP AND HYBRID GRAPHS

Part. Num	Data Set	Edge Cut (Hyb.)	Edge Cut (Ovl.)
8	1	18878760	17240330
	2	72303920	74859760
	3	26310110	30489660
16	1	21097730	21308940
	2	80966850	85452040
	3	31688290	35577120
32	1	26994530	25354840
	2	84722430	89415260
	3	34377760	41504860
64	1	31898070	32828080
	2	90542060	95242730
	3	38594120	47502740

B. Parallel graph partitioning

Recall that Section IV-C discussed the natural parallelism of the graph partitioning process. Let $G_0, G_1 \ldots G_n$ be a graph set created by graph coarsening, where $| N(G_n) | \leq | N(G_{n-1}) | \leq \ldots \leq | N(G_0) |$. Let $P = P_0, P_1, \ldots P_k$ be a partitioning that is

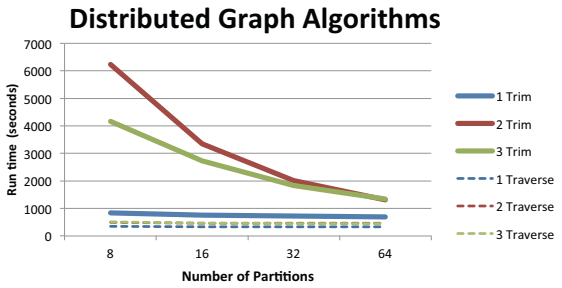

Fig. 6. Distributed graph algorithms. The trimming and traversal algorithms were applied to the different hybrid graph partitionings. The runtimes for the three read data sets are shown in blue, red, and green.

created in $\log_2(k)$ recursive bisection steps on G_0, G_1 … G_n. There are 2^i partitions that can be processed concurrently for each recursive bisection step $i = 0$ … $\log_2(k)$-1. Observe that the maximum number of processors needed to fully take advantage of this inherent parallelism is $2^{log_2(k)-1}$. Thus for any partitioning of size k for a given graph, the number of processors needed to achieve optimal speedup is $2^{log_2(k)-1}$. After recursive bisection of the graph set is complete, the multilevel Kernighan-Lin algorithm further refines each graph in G_0, G_1 … G_n. Each graph G_0, G_1, … G_n can be processed independently, thus the number of processors needed to achieve the best speedup overall will be max $(n, 2^{log_2(k)-1})$.

Here we present the results of applying an increasing number of processors to create a speedup curve for partitioning the hybrid graph sets of each read data set into 16 partitions. For each number of processors, the graph partitioning algorithm was run three times. The average of those runs and their standard deviation are shown in Fig. 4. Multiple runs were averaged together in this experiment due to time variation introduced by the random seed nodes selected by the greedy graph growing algorithm. As expected the speedup curve gains begin to level off at about eight to ten processors as $2^{\log(16)-1} = 8$ and there were ten graph levels for each of the multilevel and hybrid graph sets.

C. Incorporation of domain knowledge for graph partitioning

As stated previously, the multilevel graph set represents full graph uncoarsening to the original overlap graph G_0. The hybrid graph set represents a compromise between full graph uncoarsening and incorporation of biological knowledge. Each node in the hybrid graph G'_0 represents a cluster of nodes in G_0 that are likely to represent consecutive genomic regions within the sample DNA.

In this section, results from partitioning both the multilevel graph set and hybrid graph set are provided. For each of the read data sets, the multilevel graph sets and hybrid graph sets were partitioned into 8, 16, 32, and 64 partitions in four separate runs. The number of processors used to generate each partitioning was set to $2^{log_2(k)-1}$, where k is the number of partitions.

Results from the runs are shown in Fig. 5 and Table 2. Fig. 5 displays the runtimes for partitioning the hybrid and multilevel graph sets for each data set into 8, 16, 32, and 64 partitions. The runtime for partitioning the hybrid graph is shown in blue, while the red bars represent the runtime of the partitioning of the multilevel graph spectrum. Observe that the runtime for partitioning the hybrid graph sets was nearly half of the runtime needed to partition the multilevel graph sets. The edge cut for each partitioning was recorded in Table 2. None of the edge cuts were more than 0.43% of the total edge weight of the original overlap graphs.

TABLE III. ASSEMBLY STATISTICS

Data set	Part. Num.	N50 (bp)	Max Contig (bp)	Num. of Contigs
1	4	2 082	25 968	104 185
	16	2 083	25 968	104 219
	32	2076	25 968	104 470
	64	2076	25 968	104 470
2	4	1 513	10 486	151 411
	16	1 514	9 920	151 408
	32	1 520	9 920	151 210
	64	1 520	9 920	151 210
3	4	1 286	6 861	117 596
	16	1 285	6 861	117 629
	32	1 284	6 861	117 632
	64	1284	6861	117 632

For each data set and partition size, the edge cut for the hybrid graph G'_0 and overlap graph G_0 is shown. The lowest edge cut for either the hybrid graph or overlap graph is shaded. For all cases except for two, the partitioning of the hybrid graph set produced the lowest edge cut numbers. These results demonstrate the improvement that the partitioning algorithm was able to obtain by the inclusion of biological knowledge. To obtain a partitioning on the original multilevel graph set, the hybrid graph set partitioning can simply be projected onto the multilevel graph set.

D. Performance of Distributed Graph Algorithms

In this section, the performance of the distributed graph trimming and graph traversal algorithms is described. First the distributed graph-trimming algorithm, which includes transitive reduction, dead-end trimming, bubble popping, and containment removal, was applied to the distributed hybrid graphs for each of the three read data sets. The trimming algorithm was applied for the 8, 16, 32, and 64 partitionings generated for each hybrid graph. Following distributed graph trimming, distributed graph traversal was used to obtain an ordering of the hybrid graphs' nodes for contig construction. The runtimes for the distributed graph trimming and graph traversal algorithm can be found in Fig. 6.

For the distributed trimming algorithm, run time decreased steeply for data sets 2 and 3 as the hybrid graph was split into increasing numbers of partitions. For data 1, the run time also decreased, but not as sharply as data sets 2 and 3. This may be due to the underlying complexity of the input data sets as data set 1 also required much less run time than data set 2 or 3. Graph traversal of all of the hybrid graphs required very little run time, which remained static across the number of partitions of the hybrid graph.

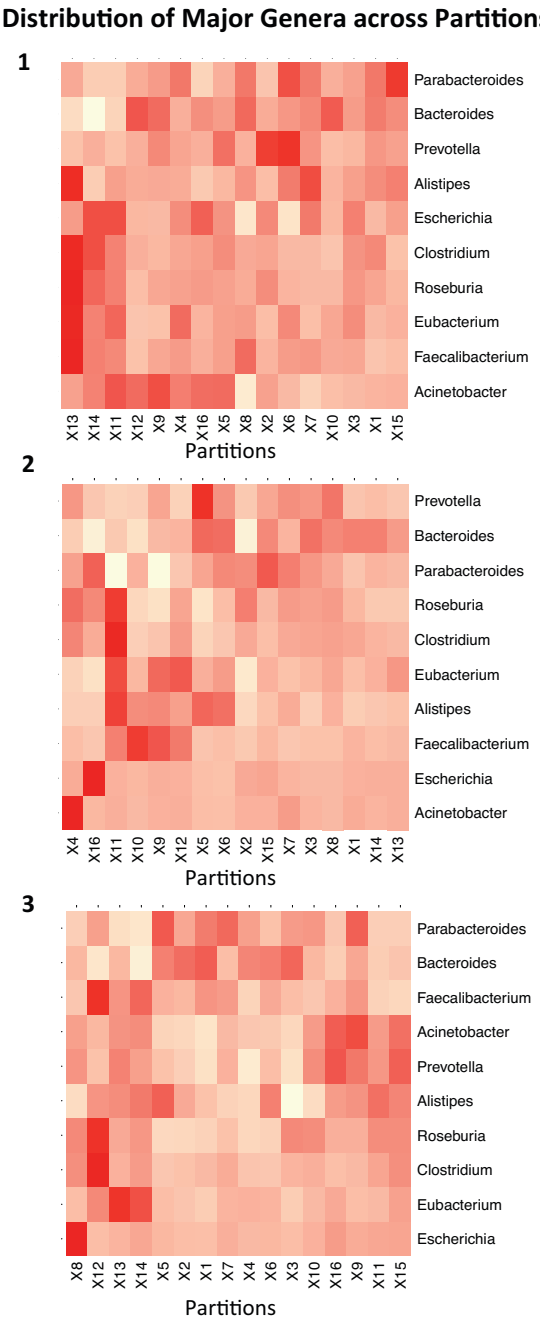

Fig. 7. Distribution of major genera across partitions. A partitioning of size 16 is represented by the columns of the heat map. Genera are represented by the rows. The colors of the heat map represents the fraction of reads from a genera which is found in a partition. Darker red colors represent a greater fraction of reads

Finally, to assess the consistency of the assembler performance across various partitionings of the hybrid graph, contigs were fully assembled and analyzed from the hybrid graph traversals. The results for each assembly of contigs can be found in Table 3.

Table 3 demonstrates the consistency of each assembly produced from runs on different partitionings of the hybrid graph. The N50 lengths are very consistent throughout different numbers of partitions. The number of contigs produced for the different partitionings is very similar, only varying by less than a couple hundred contigs for any of the data sets. The maximum length of contigs produced is stable across different partitioning configurations on the hybrid graph. Thus the assembly performance is consistent across different partition configurations of the hybrid graph.

The algorithms implemented to run on the hybrid graph are basic in nature. We plan to expand the number of analysis algorithms that can be applied to the distributed hybrid graph. For example, variant detection algorithms can be implemented to be run on the distributed hybrid graph.

E. Extracting Biological Knowledge from Graph Partitionings

In this section it is shown that the partitions generated for the hybrid graph built from the three gut metagenomics data sets can capture the community structure of those data sets. First, the gut microbiome reference sequence database for the human microbiome project was downloaded. BWA [22] was used to index this database and align the sequence reads to the reference gut microbiome sequences. The sequence reads were classified to a genus according to their best hits. If no hits were found for a given read, it remained unclassified.

After the alignment of the reads to the gut microbiome database was complete, the major genera for the three data sets were computed. The read classification counts for each of the three data sets were pooled together. The top ten genera that had the greatest pooled read counts were selected for further analysis (*Alistipes*, *Bacteroides*, *Clostridium, Escherichia, Eubacterium, Faecalibacterium, Prevotella, Parabacteroides , Roseburia)*. The distribution of these genera across the 16-partitioning for each of the three data sets was analyzed. The distribution of a given genera was calculated as the fraction of its classified reads that correspond to nodes in each partition.

The distribution of the major genera can be found in Fig. 7. Observe that the distribution of the different genera is not static across the partitions. Different genera cluster preferentially into different partitions. This is understandable as the nodes representing a contiguous genome are likely to be adjacent to one another in the hybrid graph. Nodes that are highly connected are more likely to be assigned to the same partition.

Also, it is notable that many genera that belong to the same phylum often tend to have greater read counts across the same partitions. For example, the genera *Roseburia* and *Clostridium* both have higher fractions of read counts in the same partitions. Both of these genera belong to the phylum Firmicutes. *Eubacterium* is found more frequently in the same partitions as *Roseburia* and *Clostridium* for data sets 1 and 2. The genus *Eubacterium* also belongs to the Phylum

Firmicutes. Sequences that are genetically related will have many similar regions of genome that will be represented by interconnected nodes in the hybrid graph. These connected nodes are more likely to be assigned to the same partition.

VII. Discussion

In this paper, the construction of a distributed assembly graph for next generation sequencing data was presented. Unlike most previous approaches that utilize a distributed de Bruijn graph model, this approach discussed methods for partitioning an extended overlap graph based model.

This paper covered three major objectives. First it was demonstrated that the integration of prior biological knowledge into a naïve graph partitioning algorithm could improve its performance. Partitioning the multilevel graph set represents the original naïve partitioning algorithm since the multilevel graph set is fully uncoarsened to the overlap graph during partitioning. Partitioning the hybrid graph set and projecting that partitioning onto the original multilevel graph set represents a compromise between full uncoarsening and incorporation of biological knowledge regarding the linearity of DNA. The runtime for partitioning the hybrid graph set was roughly half of the runtime for partitioning the full multilevel graph set. Finally, the edge cut was improved in most cases for the hybrid graph set in comparison to the multilevel graph set. Results demonstrate that the edge cut obtained from the partitioning was never more than 0.43 % of the original overlap graph edge weights for both the multilevel graph set and hybrid graph set.

The second objective was to successfully develop parallel algorithms for NGS assembly on the distributed graph model. Multilevel graph partitioning was applied to partition both the multilevel graph set and the hybrid graph set. Several distributed graph algorithms were then implemented on the distributed hybrid graph. Results demonstrated a substantial speedup for the graph trimming algorithm. The graph traversal algorithm did not show a great speedup; however, this algorithm had a very fast runtime. Finally, assembly results obtained from different graph partitionings were consistent, demonstrating that assembly quality is not affected by partitioning the hybrid graph.

The third objective was to demonstrate that biological knowledge could be obtained from the graph partitioning. It was shown that the distribution of genera was not equal across partitions, but that nodes representing reads from a given genus tended to be assigned to the same partition. Furthermore, phylogenetically related genera were also often found in the same partition.

This paper demonstrates high performance computing techniques for information extraction from big data. Typically, high performance computing focuses solely on faster runtimes. High performance computing is also a tool for extracting usable knowledge from big data that would have other been impossible with limited computing resources. Finally, this paper demonstrates the importance of taking input data characteristics into consideration when designing and applying algorithms. It was shown that integrating biological knowledge into the naïve partitioning process can greatly improve its results. We anticipate that this prior knowledge integration approach can improve numerous other naïve computational algorithms.

References

[1] "Illumina | Sequencing and array-based solutions for genetic research." [Online]. Available: http://www.illumina.com/. [Accessed: 07-Dec-2016].

[2] *Pacific Biosciences*. [Online]. Available: http://www.pacb.com/. [Accessed: 07-Dec-2016].

[3] J. R. Miller, S. Koren, and G. Sutton, "Assembly algorithms for next-generation sequencing data," *Genomics*, vol. 95, no. 6, pp. 315–327, Jun. 2010.

[4] E. W. Myers, "The fragment assembly string graph," *Bioinformatics*, vol. 21, no. suppl 2, pp. ii79–ii85, Jan. 2005.

[5] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. M. Jones, and İ. Birol, "ABySS: A parallel assembler for short read sequence data," *Genome Res.*, vol. 19, no. 6, pp. 1117–1123, Jun. 2009.

[6] S. Boisvert, F. Laviolette, and J. Corbeil, "Ray: Simultaneous Assembly of Reads from a Mix of High-Throughput Sequencing Technologies," *J. Comp. Biol.*, vol. 17, no. 11, pp. 1519–1533, Oct. 2010.

[7] Y. Liu, B. Schmidt, and D. L. Maskell, "Parallelized short read assembly of large genomes using de Bruijn graphs," *BMC bioinformatics*, vol. 12, no. 1, p. 354, 2011.

[8] J. Meng, B. Wang, Y. Wei, S. Feng, and P. Balaji, "SWAP-Assembler: scalable and efficient genome assembly towards thousands of cores," *BMC Bioinformatics*, vol. 15, no. Suppl 9, p. S2, Sep. 2014.

[9] X. Huang, J. Wang, S. Aluru, S.-P. Yang, and L. Hillier, "PCAP: A Whole-Genome Assembly Program," *Genome Res.*, vol. 13, no. 9, pp. 2164–2170, Sep. 2003.

[10] E. Georganas et al., "HipMer: An Extreme-scale De Novo Genome Assembler," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, New York, NY, USA, 2015, pp. 14:1–14:11.

[11] Y. Chang, et al. "A de novo next generation genomic sequence assembler based on string graph and MapReduce cloud computing framework." *BMC genomics* 13.7 (2012): 1.

[12] D. Li, C.-M. Liu, R. Luo, K. Sadakane, and T.-W. Lam, "MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph," *Bioinformatics*, vol. 31, no. 10, pp. 1674–1676, May 2015.

[13] J. Warnke and H. Ali, "Focus: A New Multilayer Graph Model for Short Read Analysis and Extraction of Biologically Relevant Features," in *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, New York, NY, USA, 2014, pp. 489–498.

[14] N. J. Larsson and K. Sadakane, "Faster Suffix Sorting," *Theor. Comput. Sci.*, vol. 387, no. 3, pp. 258–272, Nov. 2007.

[15] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Jan. 1998.

[16] D. R. Zerbino and E. Birney, "Velvet: Algorithms for de novo short read assembly using de Bruijn graphs," *Genome Res.*, vol. 18, no. 5, pp. 821–829, May 2008.

[17] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[18] S. Dutt, "New faster Kernighan-Lin-type graph-partitioning algorithms," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, 1993, pp. 370–377.

[19] G. Karypis and V. Kumar, "Multilevel k-way Partitioning Scheme for Irregular Graphs," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, Jan. 1998.

[20] "Home - SRA - NCBI." [Online]. Available: https://www.ncbi.nlm.nih.gov/sra. [Accessed: 07-Dec-2016].

[21] "Holland Computing Center | University of Nebraska–Lincoln." [Online]. Available: http://hcc.unl.edu/. [Accessed: 07-Dec-2016].

[22] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, July 2009.