*Bioinformatics*, 2016, 0–0

doi: 10.1093/bioinformatics/xxxxx

Advance Access Publication Date: DD Month YYYY

Manuscript Category

|  |
| --- |
| Genome Analysis  Omega 3: First Parallel OLC Metagenome Assembler  Corresponding Author1,\*, Co-author2 and Co-Author2  1Department of XXXXXXX, Address XXXX etc., 2Department of XXXXXXX, Address XXXX etc.  \*To whom correspondence should be addressed.  Associate Editor: XXXXXXX  Received on XXXXX; revised on XXXXX; accepted on XXXXX  Abstract  **Motivation:** An important step in metagenomics analysis is the assembly of multiple genomes from mixed sequence reads of multiple species in a microbial community. Most conventional pipelines use a single-genome assembler with carefully optimized parameters. A limitation of a single-genome assembler for *de novo* metagenome assembly is that sequences of highly abundant species are likely misidentified as repeats in a single genome, resulting in a number of small fragmented scaffolds. We extended a single-genome assembler for short reads, known as ‘Velvet’, to metagenome assembly, which we called ‘MetaVelvet’, for mixed short reads of multiple species. Our fundamental concept was to first decompose a de Bruijn graph constructed from mixed short reads into individual sub-graphs, and second, to build scaffolds based on each decomposed de Bruijn sub-graph as an isolate species genome. We made use of two features, the coverage (abundance) difference and graph connectivity, for the decomposition of the de Bruijn graph. For simulated datasets, MetaVelvet succeeded in generating significantly higher N50 scores than any single-genome assemblers. MetaVelvet also reconstructed relatively low-coverage genome sequences as scaffolds. On real datasets of human gut microbial read data, MetaVelvet produced longer scaffolds and increased the number of predicted genes.  **Results:** We conducted a comprehensive assessment on state-of-the-art de novo assemblers and revealed that the performance of each assembler depends critically on the sequencing depth. To address this problem, we developed a pipeline named InteMAP to integrate three assemblers, ABySS, IDBA-UD and CABOG, which were found to complement each other in assembling metagenomic sequences. Making a decision of which assembling approaches to use according to the sequencing coverage estimation algorithm for each short read, the pipeline presents an automatic platform suitable to assemble real metagenomic NGS data with uneven coverage distribution of sequencing depth. By comparing the performance of InteMAP with current assemblers on both synthetic and real NGS metagenomic data, we demonstrated that Omega3 achieves better performance with a longer total contig length and higher contiguity, and contains fewer misassemblies than others.  **Availability:** https://github.com/abiswas-odu/Omega3  **Contact:** panc@ornl.gov  **Supplementary information:** Supplementary data are available at *Bioinformatics* online. |

# Introduction

While voluminous datasets from high-throughput sequencing experiments have allowed new biological questions to emerge, the technology's speed and scalability are not yet matched by available analysis techniques and the gap between them has been steadily growing [[3](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR3), [4](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR4)]. The de Bruijn graph is a structure for storing DNA words - or k-mers - that occur in sequence datasets [[5](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR5), [6](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR6)]. Recent work showed that adding colors to a de Bruijn graph can allow variants to be called even in the absence of a complete genome reference [[7](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR7)].

The field of metagenomics is concerned with the analysis of communities by sampling the DNA of all species in a given microbial community. The assembly of metagenomes poses greater and more complex challenges than single-genome assembly as the relative abundances of the species in a microbiome are not uniform [[8](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR8)]. A compounding factor is the genetic diversity represented by polymorphisms and homologies between strains, which increases the difficulty of the problem for assemblers [[8](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR8)]. Moreover, the underlying diversity of the sample increases its complexity and adds to the difficulties of assembly. Last but not least, DNA repeats can produce misassemblies [[9](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR9)] in the absence of fine-tuned, accurate computational tools [[10](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR10)].

The microbial diversity in microbiomes contains the promise of finding new genes with novel and interesting biological functions [[11](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR11)]. While the throughput in metagenomics is increasing fast, bottlenecks in the analyses are becoming more apparent [[12](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR12)], indicating that only equally parallel - and perhaps highly distributed - analysis systems can help bridge the scalability gap. Parallel sequencing requires parallel processing for bioprospecting and for making sense of otherwise largely unknown sequences.

Environmental microbiomes have been the subject of several large-scale investigations. Viral genome assemblies have been obtained from samples taken from hot springs [[13](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR13)]. Metabolic profiling of microbial communities from Antarctica [[14](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR14)] and the Arctic [[15](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR15)] provided novel insights into the ecology of these communities. Furthermore, a new Archaea lineage was discovered in a hypersaline environment by means of metagenomic assembly [[16](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR16)]. The metabolic capabilities of terrestrial and marine microbial communities have been compared [[17](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR17)]. The structure of communities in the environment has been reconstructed [[18](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR18)]. All these studies show that environmental microbiomes are reservoirs of genetic novelty [[19](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR19)], which bioprospecting aims at discovering.

Through metagenomic analysis, the interplay between host and commensal microbial metabolic activity can be studied, promising to shed light on its role in maintaining human health. Furthermore, precisely profiling the human microbial and viral flora at different taxonomic levels as well as functional profiling may hint at improved new therapeutic options [[20](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR20)]. To that end, the human distal gut microbiome of two healthy adults was analyzed by DNA sequencing [[21](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR21)], and subsequently the human gut microbiome of 124 European individuals was analyzed by DNA sequencing from fecal samples by the MetaHIT consortium [[22](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR22)]. Another study proposed that there are three stable, location-independent, gut microbiome enterotypes [[23](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR23)]. Finally, the structure, function and diversity of the healthy human microbiome were investigated by the Human Microbiome Project Consortium [[24](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR24)].

With 16S rRNA gene sequencing, species representation can be extracted by taxonomic profiling [[25](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR25)]. However, using more than one marker gene produces better taxonomic profiles [[26](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR26), [27](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR27)]. Furthermore, a taxonomy based on phylogenetic analyses helps in the process of taxonomic profiling [[28](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR28)]. While taxonomic profiles are informative, functional profiling is also required to understand the biology of a system. To that end, gene ontology [[29](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR29)] can assign normalized functions to data.

Although not designed for metagenomes, distributed software for single genomes, such as ABySS [[30](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR30)] and Ray [[31](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR31)], illustrate how leveraging high-performance and parallel computing could greatly speed up the analysis of the large amount of data generated by metagenome projects. Notably, sophisticated parallel tools are easily deployed on cloud computing infrastructures [[32](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR32)] or on national computing infrastructures through their use of a cross-platform, scalable method called the message-passing interface.

Taxonomic profiling methods utilize alignments [[26](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR26), [27](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR27), [33](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR33), [34](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR34), [35](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR35), [36](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR36)]or hidden Markov models [[37](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR37)] or both[[38](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR38)]. Few methods are available for metagenome *de novo* assembly (MetaVelvet [[39](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR39)], Meta-IDBA [[40](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR40)] and Genovo [[41](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR41)]), none couples taxonomic and ontology profiling with *de novo* assembly, and none is distributed to provide scalability. Furthermore, none of the existing methods for *de novo* metagenome assembly distributes memory utilization over more than one compute machine. This additional difficulty plagues current metagenome assembly approaches.

The field of metagenomic urgently needs distributed and scalable processing methods to tackle efficiently the size of samples and the assembly and profiling challenges that this poses. Herein we show that Ray Meta, a distributed processing application, is suited for metagenomics. We present results obtained by *de novo*metagenome assembly with coupled profiling. With Ray Meta, we show that the method scales for two metagenomes simulated to incorporate sequencing errors: a 100-genome metagenome assembled from 400 × 106 101-nucleotide reads and a 1,000-genome metagenome assembled from 3 × 109 100-nucleotide reads. Ray Communities utilizes bacterial genomes to color the assembled de Bruijn graph. The Greengenes taxonomy [[28](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2012-13-12-r122#CR28)] was utilized to obtain the profiles from colored k-mers. Other taxonomies, such as the NCBI taxonomy, can be substituted readily. We also present results obtained by *de novo* metagenome assembly and taxonomic and functional profiling of 124 gut microbiomes. We compared Ray Meta to MetaVelvet and validated Ray Communities with MetaPhlAn taxonomic profiles.

Metagenome assemblers attempt to reconstruct genomes of microorganisms in a community from its metagenomic sequencing data. In recent years, many isolate genome assemblers have been developed for Illumina sequencing data using de Bruijn graphs [e.g. ABySS ([Simpson *et al.*, 2009](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-17)), IDBA ([Peng *et al.*, 2010](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-11)), ALLPATH ([Butler *et al.*, 2008](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-1)), Velvet ([Zerbino and Birney, 2008](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-19)) and SOAPdenovo ([Li *et al.*, 2010](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-7))] and overlap graphs [e.g. SGA ([Simpson and Durbin, 2012](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-16)) and PEGASUS ([Haider, 2012](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-3))]. However, they cannot be directly applied to metagenome assembly for the following reasons. First, isolate genome assemblers typically assume a uniform coverage depth across a genome. This assumption is used for identifying repeat regions in a genome and estimating the size of a genome. In metagenome assembly, however, genomes may have vastly different coverage depths depending on their relative abundances in a community. Second, isolate genome assembly only needs to resolve repeat regions within a single genome, while metagenome assembly also has to handle repeat regions between multiple genomes. Third, sequencing errors significantly convolute the assembly by introducing false overlaps between reads and disrupting true overlaps. Error correction can be performed for isolate genome assembly using consensus sequences. However, it is difficult to separate sequencing errors from single nucleotide polymorphisms (SNPs) in metagenome assembly. To address these challenges, some of the de Bruijn graph assemblers have been upgraded for Illumina metagenomic sequencing data, including MetaVelvet ([Namiki *et al.*, 2012](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-10)) and IDBA-UD ([Peng *et al.*, 2012](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-12)).

In this study, the Omega (overlap-graph metagenome assembler) algorithm was developed specifically for metagenome assembly. Omega followed the general overlap graph (string graph) approach described in BOA ([Myers, 2005](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-9)) and PEGASUS ([Haider, 2012](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-3)). Here, the overlap graph approach was adapted to metagenome assembly by addressing its differences from isolate genome assembly described above. The assembly performance of Omega was compared with SOAPdenovo, IDBA-UD and MetaVelvet on Illumina HiSeq 100-bp data and MiSeq 300-bp data. SOAPdenovo was selected because it was used for metagenome assembly in the human microbiome project ([Pop, 2011](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-13)) and many Joint Genome Institute studies. IDBA-UD and MetaVelvet were designed specifically for metagenome assembly. A widely used overlap-layout-consensus assembler, Celera ([Myers *et al.*, 2000](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-8)), was also compared using the MiSeq 300-bp data.

The pioneering work in metagenomics analysis was done by Venter *et al.*([1](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-1)) for the Sargasso Sea environmental genome analysis. The bulk extraction of diverse microbial genomes from the environment without prior laboratory cultivation is one of the most fascinating features of metagenomics. There have been several analyses of various kinds of environmental genomes, such as ([2](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-2)). Recent progress in next-generation sequencing technology offers more opportunities for metagenome analyses and permits deep sequencing (especially the Illumina Genome Analyzer) for highly diverse microbial populations. However, while a number of metagenomes have been sequenced using next-generation sequencers and deposited into public genome databases, only a few studies have reported their assembly results ([3](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-3),[4](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-4)). This is mainly because of the short length of sequence reads from next-generation sequencers. Furthermore, there is also a fundamental difficulty of metagenomics analysis compared with isolated genome analysis. In a microbial community, the number of strains is initially unknown, and their relative abundance is also unknown and potentially skewed ([5](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-5)).

There are currently a few ‘*de novo*’ assemblers specifically devoted to metagenome assembly from mixed sequence reads of multiple species ([6](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-6),[7](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-7)). In contrast, there are two alternative approaches to ‘*de novo*’ analysis for mixtures of sequence reads from environmental genomes: (i) applying a single-genome assembler to metagenome sequence reads ([3](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-3),[4](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-4),[8](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-8),[9](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-9)) and (ii) binning (clustering) a set of sequence reads into different clusters ([10](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-10),[11](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-11),[12](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-12)). However, single-genome assemblers were not designed to assemble multiple genomes from a mixture of sequence reads with nonuniform sequence coverages. On the other hand, the unsupervised binning of sequence reads also has the limitation of clustering the input reads based only on *k*-mer frequencies in the ‘short’ reads without assembly. A third approach (not *de novo*) is comparative genome analysis mapping reads or aligning contigs to reference genomes ([5](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-5),[13](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-13),[14](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-14)). Unfortunately, the comparative approach cannot cover any microbial species whose reference genomes or closely related genomes have not been assembled.

In spite of such difficulties, our primary goal was ‘to reconstruct the whole genomes of multiple species in a microbial community, particularly from very short sequence reads generated by a next-generation sequencer’. To accomplish such *de novo* metagenome assembly, we extended a single-genome assembly program (assembler), named ‘Velvet’, using a de Bruijn graph ([15](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-15),[16](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-16)), to a metagenome assembly program for mixed short reads of multiple species. A de Bruijn graph is a data structure for genome assembly programs that compactly represents an overlap between short reads. The de Bruijn graph-based assembly program identifies the overlaps between reads using a de Bruijn graph and merges the reads to reconstruct longer sequences. Note that the de Bruijn graph representation was first used for sequencing by hybridization by ([17](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-17)).

Our fundamental strategy for metagenome assembly was to consider that a de Bruijn graph constructed from mixed sequence reads of multiple species is equivalent to the mixture of multiple de Bruijn subgraphs, each of which is constructed from sequence reads of individual species and to decompose the mixed de Bruijn graph into individual subgraphs and build scaffolds based on each decomposed subgraph ([Figure 1](http://nar.oxfordjournals.org/content/40/20/e155.long#F1)).

# Methods

In this section we describe several novel optimizations developed to improve the computational performance and memory utilization of overlap graph construction and simplification algorithms. We have developed 3 version of Omega for three distinct usage patterns. Briefly they are:

1. OpenMP version: This version uses only openmp multithreading to speedup computation and runs on a single machine with multiple CPUs. The RAM usage of the version is approximately equal to the disk space size of the uncompressed reads. This version is best suited for small to medium size metagenomes as the memory requirement is the major limiting factor.
2. MPI HighMem version: This version uses multiprocess and multithreaded parallelism to improve execution time of the OpenMP version. Traditional point-to-point MPI communication is used to synchronize graph construction. The RAM usage of each MPI process is approximately equal to the disk space size of the uncompressed reads. This version is also best suited for very fast assembly of small to medium size metagenomes as the memory requirement is the major limiting factor.
3. MPI RMA version: This version uses multiprocess parallelism to reduce the memory usage of the individual processes. The read data storage is distributed equally across all the processes and random memory access feature of MPI-3 is used to for querying data from other processes. Traditional point-to-point MPI communication is still used to synchronize graph construction. The RAM usage of each MPI process can be significantly reduced by almost 70% provided enough compute nodes are available. This version is also best suited for very assembly of large metagenomes as the memory requirement is generally not a limiting factor.

We designed a distributed hash table structure to efficiently store and access data for overlap computation. We combined MPI remote memory operations and point-to-point communication for synchronizing graph construction and simplification. In the following section we will describe the design of out MPI RMA version and point out major differences from the other two versions as appropriate.

## Omega3 Design Consideration

The parallel and computational efficiency refactoring of Omega was carefully designed to assemble very large sequence volumes (~1Billion reads) and very complex metagenomes (~1000 organisms) within reasonable memory and time constraints. The major design goals were:

1. Low memory utilization
2. Distributed memory capability
3. Accurate use of paired reads

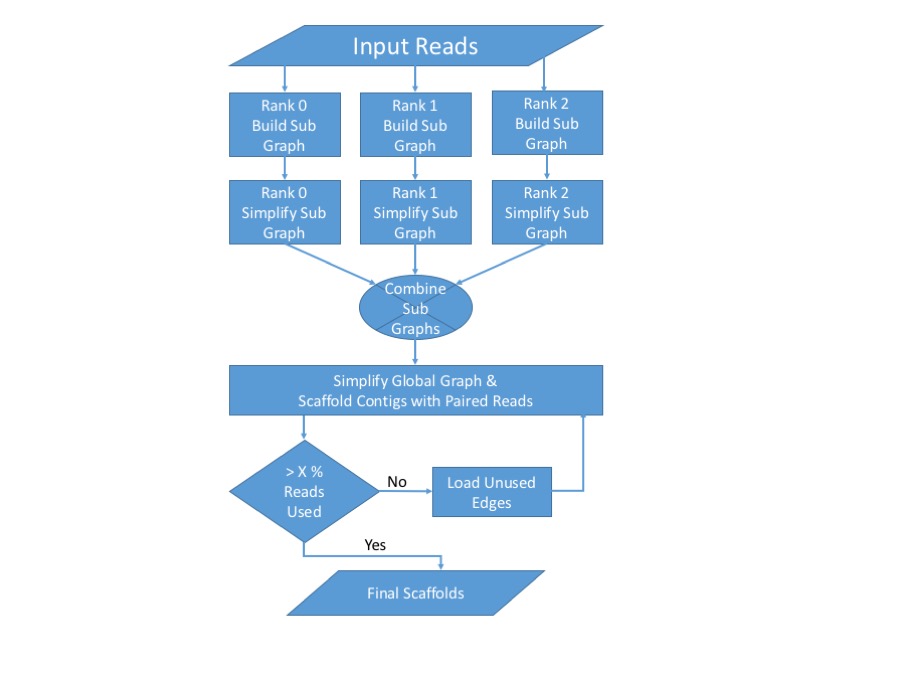
Compared with other assemblers, several distinct features of MAP algorithm should be pointed out. First, MAP does not refer to any other information such as genome length or sequencing coverage that is often used in the assemblers targeting the isolated genomes, because such information is clearly not applicable to the situation of metagenomic assembly. What is more important is that MAP employs mate-paired information different from other assemblers did. For example, the Celera Assembler (Myers et al., 2000) used mate-paired information in the scaffold constructing. The Celera Assembler later developed a new pipeline CABOG, which finds the best overlap graph in the unitigger module (Miller et al., 2008).

In this algorithm, mate pairs are used to correct the misassemblies by breaking the unitigs which are found violated with the mate pair constrains. PCAP (Huang et al., 2003) used mate-paired information to correct contigs and to link contigs into scaffolds. Different from these assemblers, MAP uses mate pairs as a core measure to construct contigs when repeats hamper the assembly. Based on mate-paired information, MAP designs a series of procedures to implement the layout stage. With regard to the OLC strategy, MAP uses the strategy similar to Celera. Thus in many cases, MAP generates contigs in similar size as Celera does. However, the former uses mate pair information to prevent error contigs from being formed, while the latter uses the information to correct and break wrong contigs. These two strategies are quite distinct, but both can avoid errors and in some cases have the similar results. However, the MAP strategy is more reasonable to avoid error as compared to Celera. For example, lacking sufficient mate pair information, Celera was not able to detect the potential errors which generated via other operation (such as best overlap graph strategy in CABOG) by its method, while MAP will sacrifice the size but avoid potential errors by leaving alone the conservative unitigs. In addition, MAP does not include the scaffolder module as Bambus 2 does. Bambus 2 uses the scaffold graph to report the regions of variation in closely related strains, and to distinguish between them. Since the analysis of the strain variation among metagenomes could be carried out by a stand-alone software such as Strainer (Eppley et al., 2007). We have demonstrated that our strategy can achieve higher assembly accuracy in terms of both reads and nucleotide sites, while maintaining high assembly capacity. However, the application of MAP algorithm is sensitive to several factors such as the mate pair length and sequence coverage, which are the state of the art challenges in metagenomic assembly. As the MAP algorithm finds mate pair paths in the overlap graph, too long length of the mate pairs may lead to too many nodes traversed between them, and complicating the process of finding mate pair paths. In addition, low sequence coverage may greatly reduce the number of overlaps, making the overlap graph being sparse. Thus, a deeper sequencing may increase the efficiency of MAP, especially in the short read data. It should be noted that without mate pair information, MAP assembles the reads into unitigs, functioning like the assembler Minimus (Sommer et al., 2007). It is noted that unlike the metagenome projects sequencing on Sanger platform in early years, recently emerged metagenome projects using 454 sequencing platform were sometimes inclined not to take the mate pair approach to sequence the DNA fragments mainly due to the lower costs. However, robust mate pair protocols are increasingly regarded as essential to new sequencing technologies for their wider application in de novo sequencing, especially to metagenomic assembly process (Kunin et al., 2008; Pop et al., 2008; Miller et al., 2010). The study in this paper demonstrates that using mate pair information to resolve the repeat in the overlap graph is the promising method for metagenomic assembly. Moreover, our attempt in this paper provides valuable support for the mate pair approach in the trade-off in sequencing protocol selection for 454 sequencing platform or other next generate sequencing platform in metagenome projects.

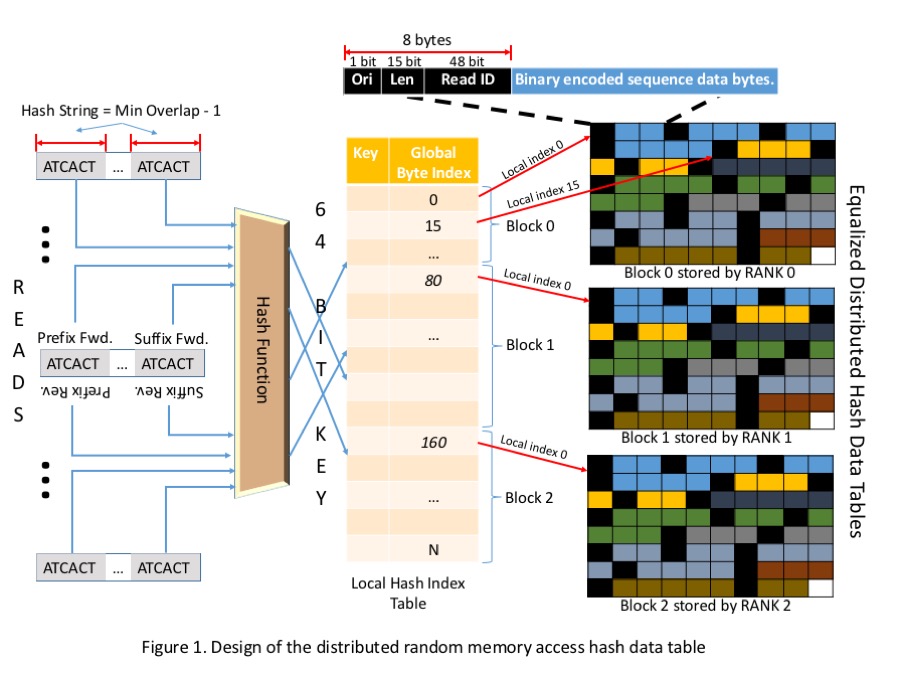
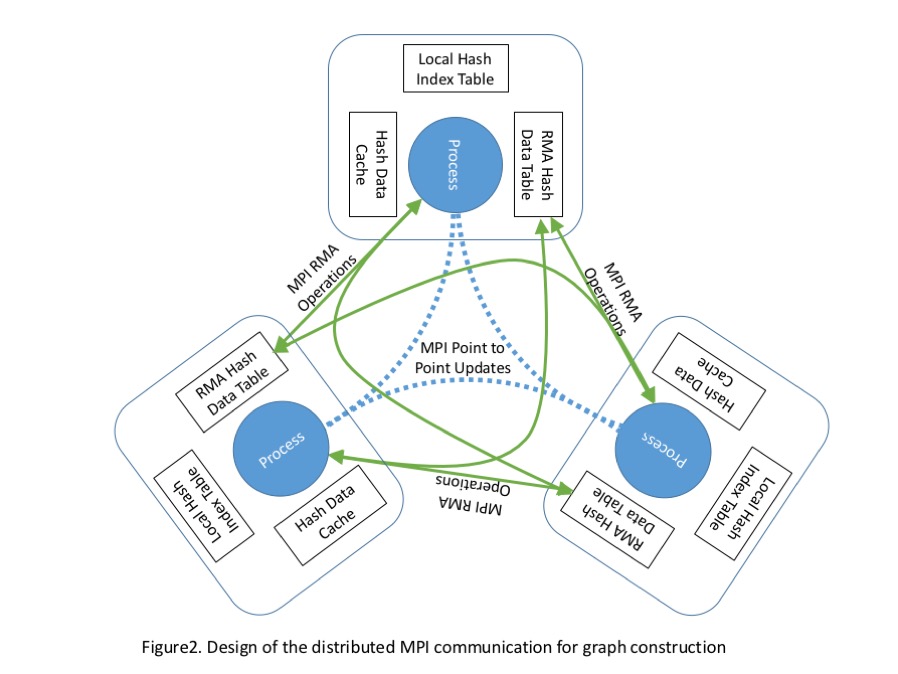
## Omega Algorithm

Omega can accept multiple input datasets with different insert sizes and variable read lengths in fasta or fastq format. The assembly and scaffolding are performed in eight steps :

1. **Hash table construction.** All unique reads are loaded to the memory and indexed in a hash table. Let *K* be the user-defined minimum overlap length. The keys of the hash table are DNA sequence substrings of length *K −*1. Each read is inserted to the hash table with four keys: prefix and suffix of length *K −*1 of both forward sequence and reverse complement sequence of the read. A value in the hash table is an array of pointers to the reads associated with the corresponding key. The hash table is initialized to be eight times of the total read number. Hash collision is resolved using linear probing. The hash table allows a nearly constant time search of all reads by their prefixes or suffixes. A read that is a substring of another read is called a *contained read*. To identify all contained reads of read *r*, every proper substring *s* of length *K* in read *r* is searched in the hash table. This produces a short list of reads that contains *s* as a prefix or suffix, which is then compared with read *r* to identify the contained reads of read *r*. The contained reads are used for coverage depth calculation and mate-pair linkage analysis below.
2. Contained Read Removal:

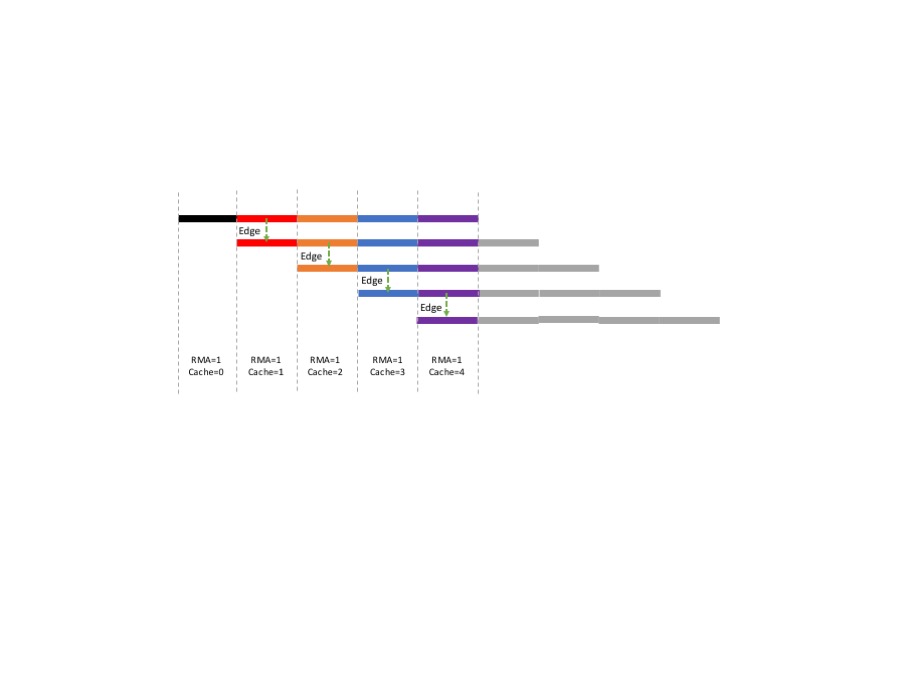


**Fig. 1.** Omega3 Assembly Flowchart

1. **Overlap graph construction.** Each read is represented by a vertex in a bi-directed overlap graph. An edge is inserted between two vertices if the two corresponding reads have an exact-match overlap of at least *K* bp. The bi-directed edges represent the four different orientations in which two reads can overlap: suffix with prefix (**•→−−→•**), suffix of the reverse complement with prefix (**•←−−→•**), suffix with prefix of the reverse complement (**•→−−←•**) and suffix of the reverse complement with prefix of the reverse complement (**•←−−←•**). To efficiently find all reads overlapping with a read *r*, every proper substring *s* of length *K −*1 in read *r* is searched in the hash table, and all retrieved reads are compared with the read *r*. If a read has the exact match with read *r* for their remaining overlap, an edge is inserted between the two reads’ corresponding vertices. After inserting all edges of read *r*, all transitive edges incident on read *r* are removed using a linear algorithm as described (Myers, 2005)(Haider *et al.*, 2014). Briefly, suppose that *r* is connected with two other reads, *a* and *b*. If there is also an edge between *a* and *b* to form a triangle with *r* and the sequence represented by the edge (*r, b*) is the same as the sequence represented by the path through (*r, a*) and (*a, b*), then (*r, b*) is identified as a transitive edge and is deleted. Removing all transitive edges significantly simplifies the overlap graph without losing any information.
2. **Composite edge contraction.** While the bi-directed edges can be traversed in both directions, the vertices can be traversed only by entering a vertex in an in-arrow and exiting in an out-arrow (**−→•→−**) or by entering a vertex in an out-arrow and exiting in an in-arrow (**−←•←−**). A valid path in the overlap graph represents an assembled DNA sequence containing proper overlapping reads with appropriate orientation and sufficient overlap length. After removing transitive edges, simple vertices have exactly one in-arrow and one out-arrow, representing only one possible way to traverse such simple vertices. A read in a simple vertex uniquely overlaps with one other read in either direction. To simplify the overlap graph, a simple vertex, *r*, along with its in-arrow edge (*u, r*) and out-arrow edge (*r, w*), are replaced by a composite edge (*u, w*) in the overlap graph. The composite edge (*u, w*) contains the read *r*and all ordered reads in edge (*u, r*) and (*r, w*). The edge (*u, w*) has the same arrow types to *u* and *w* as the original edges, (*u, r*) and (*r, w*), respectively. Simple vertices are merged into composite edges iteratively, until there is no simple vertex remaining in the overlap graph.
3. **Sequence variation removal.** Sequence variations originate from uncorrected sequencing errors and natural sequence polymorphisms in microbial communities. Many reads with sequence variations do not overlap with any other reads and are represented as isolated vertices in the overlap graph. Reads with the same sequence variation may overlap with one another, which creates small branches and bubbles in the overlap graph. Small branches are short dead-end paths that contain <10 reads. Bubbles are two edges that connect the same two vertices with the same arrow types. The overlap graph is systematically traversed to trim off small branches and remove the edges containing less reads in bubbles. This may create new simple vertices that are then removed by repeating the composite edge contraction.
4. **Minimum cost flow analysis.** Each edge in the overlap graph is associated with a string copy number, representing how many times the edge’s sequence is present in the metagenome. String copy numbers of edges are estimated based on the topology of the overlap graph using minimum cost flow analysis as described ([Haider, 2012](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-3); [Myers, 2005](http://bioinformatics.oxfordjournals.org/content/30/19/2717.long#ref-9)). Composite edges with sequences >1000 bp are set to have a minimum flow of 1, requiring such edges’ sequences to be present in the metagenome at least once. The minimum flow for short edges (<1000 bp) is set to 0. The CS2 algorithm (Goldberg, 1997) is used to optimize the amount of flow passing through every edge such that the total cost of the flow network in the overlap graph is minimized. Edges with more than one unit of flow correspond to repeat regions shared among multiple genomes or multiple places in a single genome. Edges with zero flow represent short sequences that are not needed to connect long sequences together and are ignored. Tree structures in the overlap graph are simplified using the flows. A tree comprises two edges, (*p, t*) and (*q, t*), merging to a third edge (*t, r*), and the flow on (*t, r*) is equal to the total flow on (*p, t*) and (*q, t*). Such a tree is reduced to two new edges (*p, r*) and (*q, r*) that both contain the reads in vertex *t* and edge (*t, r*).
5. **Merging of adjacent edges with mate-pair support.** The insert size of each paired-end dataset is estimated separately to accommodate a mixture of datasets with different insert sizes. The overlap graph at this stage has long composite edges that contain both reads of many mate-pairs. The insert sizes of such pairs are determined from their relative locations on the long edges and are pooled to estimate the mean *μ* and SD *σ* of all mate-pairs’ insert sizes in each dataset. Mate-pairs that span multiple edges are used to merge adjacent edges in the overlap graph. For each of such mate-pairs, all possible paths of length within range (*μ* − 3*σ*, *μ* + 3*σ*) are enumerated. If all paths of a mate-pair travel through two adjacent edges, (*m, r*) and (*r, n*), the connection between these two edges is considered to be supported by this mate-pair. After processing all mate-pairs, if the connection between (*m, r*) and (*r, n*) is supported by more than three mate-pairs, these two edges are merged to one edge (*m, n*) containing a duplicated *r*.
6. **Scaffolding of long edges with mate-pair support.** Scaffolding uses mate-pairs that have no valid path between their paired reads in the overlap graph because of a gap in genome coverage. Scaffolding is attempted for every pair of non-adjacent edges >1000 bp. A mate-pair is considered to support the scaffolding of two edges if its two reads are uniquely mapped to the two edges at an appropriate distance apart. After processing all mate-pairs, the scaffolds of long edges with support of more than three mate-pairs are accepted.
7. **Resolving ambiguity by coverage depth.** Many unresolved vertices in the overlap graph have two incoming edges and two outgoing edges, which often originate from a short repeat region between two different genomes. The two genomes may have different coverage depths to separate their edges. The coverage depth is calculated for every position along an edge to estimate the mean *δ* and SD *θ* of coverage depth along the edge. Only unique reads in an edge are considered for coverage depth calculation. A pair of adjacent edges on an unresolved vertex are merged if | *δ*1 –*δ*2 | < *θ*1 + *θ*2.

We performed two major experiments to test the performance and accuracy of Omega3. We now describe the algorithm in the layout stage integrated into matepaired information to find the optimal paths to construct final contigs. The flowchart of the algorithm is presented in Figure 1 too. In the current OLC approach, the overlap graph is used to facilitate the assembly process. Conceptually, reads and overlaps are represented in the graph G by nodes and bi-directed edges respectively. The arrows of both ends of the edge are determined by the way how two reads overlap (See Figure 2). Herein, a dovetail path is defined as an acyclic path with each node has only one arrow outward it and one arrow inward it. Thus, a dovetail path can determine a certain contig by means of threading the reads corresponding to the nodes in this path. Thus the goal of the layout stage is to separate the graph into disconnected dovetail paths. However, since there may be quite many misleading edges in the graph that represent the false overlaps mainly originated from two repetitive DNA regions or similar fragments of different genomes, this goal seems to be a formidable task. To this end, MAP is designed to determine the optimal dovetail paths as following steps: 1) Construct overlap graph Based on the output of the overlap stage, a bi-directed graph, named overlap graph, is constructed where the nodes and edges represent reads and overlaps, respectively. The overlaps where one read is completely contained in the other read are not included in graph construction. 2) Merge unitigs Unitig is designated to represent the contig in which the reads have no contested overlaps with any other reads (Myers et al., 2000). In overlap graph, a dovetail path without any other edges intersected can directly lead to a unitig, and the path is called a simple path. Herein, MAP uses the transitive reduction algorithm to remove the transitive edges (Myers, 2005). If three nodes A, B, C are linked by overlaps A-B, B-C, and A-C, and the overlaps are mutually consistent among three reads, then the edge A-C is said a transitive edge. Then a simple path may be replaced by a node representing the corresponding unitig, so that the graph can be reduced. Thus the node in graph may be either a read or a set of reads in a simple path. 3) Find mate pair paths To further select the path from the graph with many conflicting edges, the algorithm then goes to find the paths with mate pair threading. As mate-paired reads come from the same longer DNA fragment, the path with mate pair threading can be regarded as an authentic path corresponding a true DNA fragment. A path is defined as a mate pair path, if both ends are linked by mate pairs, and distance of the mate reads in the path is consistent with the mate pair length (Figure 3a, b). A description of the method of mate pair path finding can be referred to in Methods in Supplementary materials. After the mate pair paths are decided in the graph, the edges that are not only excluded (i.e. not passed) in any mate pair paths but also intersect other mate pair paths are removed (Figure 3c). After this step many paths (probably not all) become simple paths. There may be still mate pair paths that intersected each other. For example, multiple mate pair paths may across each other and a pair of nodes may involve more than one mate pair path connecting them. In these cases, nodes in the graph are classified into stem nodes and bridge nodes, in which the bridge nodes are defined as not only being in the middle of the path but also having the length shorter than that of mate pairs while the remaining nodes are classified as stem nodes. 4) Separate middle shared nodes If a bridge node is shared by more than one mate pair paths, we separate these mate pair paths by duplicating this bridge node for each mate pair path in the graph (Figure 4A). 5) Pack bubbles If two stem nodes are linked by several paths including bridge nodes, thus forming a so-called bubble structure (Miller et al., 2010), MAP connects the stem nodes by replacing those paths (named bridge paths) with an altered node linking to the stem nodes, named bubble node, which does not represent a sequence but several sequences that link both the stem nodes (Figure 4B).

## Hash Table and Distributed Read Data Storage

The hash table described in section 3.1.1 is the major read storage that can be distributed and stored across all nodes (Fig 1.). All reads are loaded to the memory and indexed in a partially distributed hash table for fast lookup. Let *K* be the user-defined minimum overlap length. The keys of the hash table are DNA sequence substrings of length *K –*1; 4 for each read corresponding to the prefix and suffix of length *K −*1 of both forward sequence and reverse complement sequence respectively. To reduce the number of times the read is stored in the hash table we apply a hash function that generates for same key for the prefix of the forward sequence and the suffix od the reverse complement sequence. Similarly, the key generated for the suffix of the forward sequence and prefix of the reverse complement sequence is the same. This allows us to store the read only twice in the hash table instead of all 4 times for all 4 substrings. We store a single orientation bit along with other read metadata such as read identifier and length for differentiating between the prefix or suffix. This bit is used to correctly compute the orientation of a hit when querying the hash table.

One major design consideration while distributing the hashed read across multiple processes is that sequences hashing to the same value must the stored together on the same node. Otherwise, each hash lookup will result is more than one costly remote memory access operation. A main consideration in our design of the distributed hash table is to ensure minimum number of remote access operation. Therefore, we introduced a local hash index table to store the global index of each hash block. This local hash index table initially counts the number of bytes required to store all the read entries for each hash key in a preprocessing step. This information allows to equally distribute the hash data across nodes and then a cumulative sum operation on the local hash index table gives us the global index for each hash key. This global index can be used to determine the rank of the MPI process storing the hash data and the local index of the hash data within that process’s remote memory access window. The length of the block of data to be fetched can be computed by looking and the global index of the next hash key and subtracting from it the queried global index. With these three bits of information i.e. rank, local index and length we can initiate a MPI\_Get operation to lookup remote memory and receive all the reads hashed to a particular key in one operation.

The same design of the hash table is used by the OpenMP and MPI HighMem versions with all the data blocks being stored on a single node and therefore requiring no remote memory operations.

**Fig. 5. Relation between τ and *t*.** This example has only two continuous Steppers, S1 and S2.

## Distributed Data Transfer and Synchronization

The distributed hash data storage described in section 3.2 is highly efficient and performs no more than 1 remote memory operation for one hash lookup. Hash lookups are used for determining the potential set of overlapping reads for a given read. Computing all overlapping edges of a read requires N-K+1 hash lookups with each such operation requiring 1 costly remote access. Consider the read alignment in the figure Fig. 5. The large number hash lookups for each read will also again quried be performed for neighboring reads and again repeated for R3. To avoid such repeated remote memory accesses, we use a hash data cache that stores the ash lookup results locally. We use least recently used cache algorithm to cache remote data and improve performance.

Our MPI remote memory access algorithm assumes that MPI\_THREAD\_SERIALIZED is available and implemented by the MPI library.

The graph construction process in each MPI process is further multithreaded. The main thread does not perform any computation but is only responsible for point to point MPI communication to reduce redundant neighborhood computation.

# Results & Implementation

We performed two major experiments to test the performance and accuracy of Omega3. We compare the accuracy of Omega3 against 6 other popular assemblers – SPADes (Bankevich *et al.*, 2012), ABySS , MegaHit (Li *et al.*, 2015), MetaVelvet (Namiki *et al.*, 2012), RAY (Boisvert *et al.*, 2012) and HipMer (Georganas *et al.*, 2015). The performance comparisons for the OpenMP version of Omega3 is shown against all the 6 above assembler. The MPI HighMem and RMA version of Omega3 are compared against RAY and HipMer which can scale to many nodes.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Metrics | | Omega | SPADes | ABySS | MegaHit | MetaVelvet | RAY | HipMer |
|  | Dataset: 64 Genome | | | | | | | |
| # of Contigs | |  |  |  |  |  |  |  |
| N50 | |  |  |  |  |  |  |  |
| Misassembly | | 342 | 415 | 437 | 449 | 643 | 772 | 1069 |
| NGA50 | | 82522 | 57896 | 55234 | 78313 | 28242 | 19886 | 17567 |
| Genome Frac. | | 92.33 | 93.28 | 89.70 | 95.29 | 89.09 | 86.45 | 88.23 |
|  | Dataset: MC06 | | | | | | | |
| # of Contigs | |  |  |  |  |  |  |  |
| N50 | |  |  |  |  |  |  |  |
| Misassembly | | 10121 | 10355 | 4280 | 19103 | 4891 | 6756 | 8099 |
| NGA50 | | 45332 | 35296 | 17406 | 36818 | 5551 | 10998 | 9925 |
| Genome Frac. | | 34.36 | 32.66 | 24.33 | 30.64 | 26.85 | 30.22 | 28.01 |
|  | Dataset: MC04 | | | | | | | |
| # of Contigs | |  |  |  |  |  |  |  |
| N50 | |  |  |  |  |  |  |  |
| Misassembly | | 9139 | 9868 | 4530 | 17886 | 4814 | 7122 | 8265 |
| NGA50 | | 80762 | 60797 | 29663 | 52541 | 5831 | 26540 | 11652 |
| Genome Frac. | | 32.36 | 32.53 | 24.60 | 30.61 | 27.01 | 29.34 | 30.87 |
|  | Dataset: CAMI | | | | | | | |
| # of Contigs | |  |  |  |  | -- |  |  |
| N50 | |  |  |  |  | -- |  |  |
| Misassembly | | 6740 | 7084 | 13110 | 3141 | -- | 4511 | 6552 |
| NGA50 | | 91243 | 90043 | 88581 | 86627 | -- | 73889 | 56232 |
| Genome Frac. | | 98.84 | 89.74 | 82.57 | 92.39 | -- | 89.04 | 91.09 |
|  | Dataset: MOCK250 | | | | | | | |
| # of Contigs | |  |  |  |  |  |  |  |
| N50 | |  |  |  |  |  |  |  |
| Misassembly | | 165 | 118 | 65 | 30 | 68 | 211 | 177 |
| NGA50 | | 201789 | 172861 | 95881 | 6973 | 10133 | 6973 | 7723 |
| Genome Frac. | | 89.48 | 90.95 | 85.46 | 88.27 | 89.14 | 88.31 | 89.02 |

## Comparison of Assembler Accuracy

We compare Omega3 accuracy with 5 other assemblers and report.

### 3.4.1 Evaluation Metrics

We use MetaQUAST tool to assess the quality of assembly (Gurevich *et al.*, 2013). We select 3 major assembly quality parameters to report from the various metrics that are reported by MetaQUAST. The quality parameters are:

1. # misassemblies: This is the measure of the the number of contigs that are incorrectly joined. This metric is the sum of all relocations, inversions and translocations errors. This metric is the best measure of correctness of an assembly. Local base calling errors are not included in this metric.
2. Median NGA50: NGA50 is calculated after splitting incorrect contigs and scaffolds to align then all to the reference genomes. The N50 score of these split contigs and scaffolds is NGA50. Since, the NGA50 is reported for each reference in the metagenome individually. We pick the median NGA50 of all the reference organisms that are at least covered 10 times by the read library. This metric is a good metric to measure contiguity of the assembly. Due to the fact that incorrect contigs and scaffolds are broken to align correctly with the reference this metric cannot influenced by incorrect long contigs.
3. Genome Fraction (%): Genome fraction is the ratio of bases in the reference that appear at least once in the assembled contigs. This metric is a good measure of completeness of the assembly. In cases of complex metagenome communities with large number of low coverage organisms this value is very useful to determine the capability of an assembler to assemble low coverage organisms.

Further details on QUAST assembly metric can be found in (Gurevich *et al.*, 2013).

### 3.4.2 Test Datasets & Experiment Setting

The comparison is conducted on 5 mock datasets of various sizes and community complexities. Although a mock dataset does not capture all genetic variations (and associated complexity) occurring in natural microbial populations, it is a way to validate the correctness of assemblies produced by the assemblers used for testing and fairly compare the correctness generated by the assemblers. Table 1 shows the input parameters of the test datasets. The 64 genome dataset is a small low complexity dataset with 64 genomes covered almost uniformly. The MC04 and MC06 are mock datasets are simulated high complexity datasets with 868 microbes in the community. These two mock datasets are drawn from the same set of reference organisms however the abundance of the organisms are different. The mock 250 medium volume dataset is a real Illumina sequenced library from a low complexity mock community of only 26 reference organisms. The CAMI dataset is a medium complexity mock dataset with large sequence volume. These 4 datasets were trimmed and filtered using BBtools(cite?) and error corrected using the Tadpole(cite?).

**Table 1. Description of Test Datasets. The datasets are Illumina sequenced reads.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sl. No. | Name | Seq.  Param.(bp) | # of Reads  (Mbp) | # of  References | Read Disk Space(GB) |
| 1 | 64 Genome | 2 x 100 | 214 | 64 | 24 |
| 2 | MC04 | 2 x 150 | 128 | 144\*(868) | 42 |
| 3 | MC06 | 2 x 150 | 128 | 137\* (868) | 42 |
| 4 | MOCK 250 | 2 x 250 | 210 | 26 | 108 |
| 5 | CAMI | 2 x 150 | 725 | 304 | 241 |

The datasets were chosen to reflect both the sequencing volume of metagenomics reads and complexity of metagenomics communities.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Metrics | | Omega | SPADes | ABySS | MegaHit | MetaVelvet | RAY\* | HipMer\* |
|  | Dataset: 64 Genome | | | | | | | |
| Time (sec.) | | 8843 | 10500 | 15447 | 5146 | 35210 | 12231 | 2421 |
| Memory (GB) | | 22.31 | 20.34 | 18.23 | 13.56 | 22.28 | 92.76 | 102.03 |
|  | Dataset: MC06 | | | | | | | |
| Time (sec.) | | 21660 | 79601 | 65350 | 10080 | 54750 | 82980 | 3255 |
| Memory (GB) | | 38.71 | 127.45 | 134.55 | 113.78 | 83.86 | 120.35 | 118.09 |
|  | Dataset: MC04 | | | | | | | |
| Time (sec.) | | 22980 | 81753 | 37214 | 9420 | 58108 | 81091 |  |
| Memory (GB) | | 39.10 | 92.12 | 80.45 | 113.21 | 83.96 | 119.89 |  |
|  | Dataset: CAMI | | | | | | | |
| Time (sec.) | | 292962 | 93577 | 159516 | 53920 | -- | 133132 |  |
| Memory (GB) | | 226.04 | 111.33 | 92.37 | 99.00 | -- | 190.22 |  |
|  | Dataset: MOCK250 | | | | | | | |
| Time (sec.) | | 170751 | 17909 | 43295 | 9440 | 80749 | 113451 |  |
| Memory (GB) | | 116.34 | 36.47 | 19.69 | 41.05 | 77.92 | 69.23 |  |

## Comparison of Computational Performance

In order to assess the scalability of Omega MPI versions, we use one simulated large dataset with complex metagenome complexity.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Table 4. Benchmark results Omega scalability | | | | | | | | |
| #of  Cores | Omega MPI  HighMem | | Omega MPI  RMA | | RAY | | HipMer | |
|  | Mem. | R.T. | Mem. | R.T. | Mem. | R.T. | Mem. | R.T. |
| 1 |  |  |  |  |  |  |  |  |
| 32 |  |  |  |  |  |  |  |  |
| 64 |  |  |  |  |  |  |  |  |
| 128 |  |  |  |  |  |  |  |  |
| 256 |  |  |  |  |  |  |  |  |
| 512 |  |  |  |  |  |  |  |  |
| 1024 |  |  |  |  |  |  |  |  |

Parallel performance experiments are conducted on Cori, the Cray XC30 located at NERSC. Cori has a peak performance of 2.57 petaflops/sec, with 5,576 compute nodes, each equipped with 120 GB RAM and two 12-core 2.4GHz Intel Ivy Bridge processors for a total of 133,824 compute cores, and interconnected with the Cray Aries network using a Dragonfly topology. For our experiments, we use Cray’s parallel Lustre /scratch3 file system, which has 144 Object Storage Servers providing 144-way concurrent access to the I/O system with an aggregate peak performance of 72 GB/sec. To analyze performance behavior of the OpenMP version of the Omega3 we recorded the peak memory usage and wall clock execution time of the assemblies from all the assemblers discussed in section 3.4. We can see that performance of the Omega3 is directly proportional to the volume of the sequence data to be assembled. For example we see that Omega3 takes much longer to assemble the low complexity mock 250 dataset as compared to the more complex MC06 dataset(Table ? column ?). However, the performance of the other assemblers which are de-Drujin graph based are dependent more on the complexity of the metagenome community than the volume of the sequence data. For example we see a reverse trend in the resource utilization of SPADes which takes far longer to assemble the complex MC06 dataset than the low complex mock250 dataset even though the sequence volume of the mock250 dataset is over twice that of the MC06 dataset.

To investigate the scalability of our approach for *de novo* metagenome assembly, we recorded the peak memory usage and wall clock time of all the three assemblers for the MC06 dataset. We decided to examine the MC06 dataset described in 3.2.2 which is a mock dataset simulatin5rg complex low abundance metagenomes with fairly large sequence volume. This dataset was assembled with the two Omega3 MPI versions, Ray Meta and HipMer. The Omega3 HighMem version significantly reduces the execution time required compared to the OpenMP version and has a much lower memory footprint and if the only viable assembler for this dataset at small core numbers. The Omeag3 RMA version further reduced the memory requirements of the assembly program and is capable of distributing almost 70% of the data storage and can still provide good execution times as the number of nodes are made available.

This is table foot note sample text This is table foot note sample text This is table foot note sample text

Conclusion

Next-generation short-read sequencing technology has resulted in explosive growth of sequenced DNA. However, de novo assembly has been unable to keep pace with the flood of data, due to vast computational requirements and the algorithmic complexity of assembling large-scale genomes and metagenomes. In this work we address this challenge head on by developing HipMer, an end-to-end high performance de novo assembler designed to scale to massive concurrencies. Our work is based on the Meraculous assembler, which has been shown to be one of the top de novo approaches in recent Assemblathon competitions [6]. We developed several novel algorithmic advancements by leveraging the efficiency and programmability of UPC, including optimized high-frequency k-mer analysis, communication avoiding de Bruijn graph traversal, advanced I/O optimization, and extensive parallelization across the numerous and complex application phases. We emphasize that distributed hash tables lie in the heart of HipMer and the main operations on them are irregular lookups. Therefore our algorithms avoid synchronization and message matching logic that would be imposed by a two-sided communication model and instead employ the asynchronous one-sided communication capabilities of UPC. The global address space is also convenient for these algorithms, since variables may be directly read and written by any processor.

Overall results show unprecedented performance and scalability, attaining an overall runtime of 8.4 minutes for the human DNA at 15K cores on the Cray XC30, compared with 10.8 hours for Ray and 23.8 hours for the original Meraculous application. Additionally, we explored performance on the grand-challenge wheat genome, which, to date, has been too large and complex for most modern de novo assemblers. Our results demonstrated impressive scalability, allowing the completed wheat assembly in just 39 minutes using 15K cores. Furthermore, we have shown that the distributed memory implementation of HipMer can successfully assemble contigs from one of the largest, most complex and challenging, deeply sequenced metagenome datasets in less than 11 minutes using 20K cores. We have begun to apply the Meraculous algorithm to the new challenge of metagenome assembly and plan to continue to develop the necessary algorithmic changes for scaffolding, variant resolution and clustering, and will then adapt this code into an end-to-end high performance metagenome assembler. Using our HipMer technology enables — for the first time — assembly throughput to exceed the capability of all the world’s sequencers, thus ushering in a new era of genome analysis. Additionally, HipMer makes it possible to improve assembly quality by running tuning parameter sweeps that were previously prohibitively expense. The combination of high performance sequencing and efficient de novo assembly is the basis for numerous bioinformatic transformations, including advancement of global food security and a new generation of personalized medicine.

This study described an extension of the single-genome assembler Velvet to a metagenome assembler, MetaVelvet, and showed its effectiveness in several experiments using simulated datasets and human gut microbial sequence datasets. This work is the first step in the construction of a de novo metagenome assembler for the assembly of metagenomes from mixed ‘short’ sequence reads of multiple species.

Meta-IDBA ([6](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-6)) is a currently available and practically executable *de novo*metagenome assembler. From the results using the simulated datasets, it can be seen that MetaVelvet outperformed Meta-IDBA. In fact, the Meta-IDBA project ([6](http://nar.oxfordjournals.org/content/40/20/e155.long#ref-6)) pointed out that one of their future works is to make use of uneven abundance ratios in metagenomic datasets. On the other hand, Meta-IDBA is designed to solve the metagenome assembly problem caused by polymorphisms in similar species in metagenomic environments. In this aspect, Meta-IDBA might be more useful for analyzing slight variants in the genomes of subspecies within a same species. [Figure 13](http://nar.oxfordjournals.org/content/40/20/e155.long#F13) shows the comparisons of the N50 scores of contigs between MetaVelvet assemblies and Meta-IDBA assemblies. Since Meta-IDBA does not have the scaffolding function, this result shows a fair comparison on assembly performances between both assemblers. Nevertheless, MetaVelvet achieved higher N50 scores than Meta-IDBA at the order, family and genus levels. On the other hand, Meta-IDBA produced higher N50 score for contigs at the species level, which confirms the specific feature of Meta-IDBA mentioned above.

Although we have designed the primary procedure for MetaVelvet, a key issue is how to identify and deal with ambiguous nodes, termed chimeric nodes in this article, with multiple incoming and outgoing edges in the de Bruijn graph. Most nodes in the de Bruijn graph are unique nodes with only one incoming edge and one outgoing edge, and as such, are reliable for building contigs. The single-genome assemblers recognize the ambiguous nodes as repeat nodes, where a repeat node represents a sequence that occurs several times in the genome, which is correct in single-genome assembly. MetaVelvet explicitly identifies chimeric nodes as causing misassemblies that combine reads from distinct species to generate chimeric scaffolds. MetaVelvet then identifies and separates the chimeric node into two unique nodes using node coverage differences. The results showed that this simple strategy worked well for metagenome assembly. The results demonstrate that MetaVelvet is a potentially valuable tool that can be widely used in metagenomic analyses. In particular, a significant increase of the number of predicted complete protein-coding genes is valuable in the search for novel enzymes in metagenome research.

Acknowledgements

We thank Dr. XX for their help to use and modify the source codes of…

Funding

This work has been supported by the …..

*Conflict of Interest:* none declared.

References

Bankevich,A. *et al.* (2012) SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *J. Comput. Biol.*, **19**, 455–477.

Boisvert,S. *et al.* (2012) Ray Meta: scalable de novo metagenome assembly and profiling. *Genome Biol.*, **13**, 1–13.

Georganas,E. *et al.* (2015) HipMer: An Extreme-scale De Novo Genome Assembler. In, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’15. ACM, New York, NY, USA, p. 14:1--14:11.

Goldberg,A. V (1997) An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *J. Algorithms*, **22**, 1–29.

Gurevich,A. *et al.* (2013) QUAST: quality assessment tool for genome assemblies. *Bioinforma.* , **29**, 1072–1075.

Haider,B. *et al.* (2014) Omega: an Overlap-graph de novo Assembler for Metagenomics. *Bioinforma.* , **30**, 2717–2722.

Li,D. *et al.* (2015) MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinforma.* , **31**, 1674–1676.

Myers,E.W. (2005) The fragment assembly string graph. *Bioinforma.* , **21**, ii79-ii85.

Namiki,T. *et al.* (2012) MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic Acids Res.*, **40**.