Genomic Applications on Cray supercomputers: Next Generation Sequencing Workflow

Mikhail Kandel
Department of Electrical and Computer Engineering
University of Illinois
Urbana-Champagne, IL
USA
kandel3@illinois.edu
Stephen Behling, Nathan Schumann and Bill Long
Cray Inc
Saint Paul, MN
USA
sbehling@cray.com, nds@cray.com, longb@cray.com
Carlos P. Sosa
Cray Inc and University of Minnesota Rochester
Saint Paul, MN
USA
cpsosa@cray.com
Sébastien Boisvert and Jacques Corbeil
Département de Médecine Moléculaire, Université Laval,
Québec, Canada
sebastien.boisvert.3@ulaval.ca,
jacques.corbeil@crchul.ulaval.ca
Lorenzo Pesce
Computation Institute, University of Chicago, Chicago, IL
USA
lpesce@uchicago.edu

***Abstract*—Recent progress in DNA sequencing technology has yielded a new class of devices that allow for the analysis of genetic material with unprecedented speed and efficiency. These advances, styled under the name Next-Generation Sequencing (NGS), increasingly shift the burden from chemistry done in a laboratory to a string manipulation problem, well suited to High-Performance Computing (HPC). By breaking up DNA into millions of small strands (20 to 1000 bases) and reading them in parallel, the rate at which genetic material can be acquired has increased by several orders of magnitude at the expense of a new distinction between raw and processed genetic data. The technology that generates raw genomic data is becoming increasingly fast and inexpensive when compared to the rate that this data can be analyzed. In general, assembling small reads into a useful form is done by either assembling individual reads (*de novo*) or mapping these pieces against a reference (mapping). In this paper, we present our experience with Ray, a parallel short-read *de novo* assembler code. We also present a configuration for an NGS workflow based on Sonexion storage and Cray supercomputers.**

Keywords—Next-generation sequencing; assembler; DNA; high-performance computing; parallel; genome

I. Introduction

Today NGS produces large quantities of small fragments of DNA, called reads. In general, assembling small reads into a useful form is done by either assembling individual reads (*de novo*) or mapping these pieces against a reference (mapping). However, the success of the new technology to generate data faster has come at a price. Sequencers will produce reads that are too small (< 150 base pairs (bp)) or overlap-layout-consensus assemblers [1]. Alternately, de Bruijn graph-based assemblers have proven to be successful at assembling short reads [1,2].

One of such assemblers is Ray [3,4]. There are multiple software packages to perform *de novo* assembly [1]. However, Ray is a highly parallel assembler designed to leverage high-performance computing architectures. In addition, Ray has been developed to assemble reads obtained from different sequencing technologies. Boisvert et al. [3] have shown that using a mixture of 454 and Illumina reads is possible to assemble genomes with greater accuracy. Most importantly, they have shown that by using this hybrid approach they were able to reduce the number of *contigs* (large sequences reconstructed from reads of DNA) and the number of errors.

Recently Bradnam et al. reported as part of the Assemblathon 2 paper [5] that Ray was ranked among the best assemblers in terms of correctness and gene content. As we follow the trend for the last few years, the data deluge is not going to stop. To keep up with the exponential growth in data production, it is critical to have tools that can not only accurately but efficiently make use of the data. It is important to continue developing Ray to achieve the goal of assembling a human genome in a few hours. Presently Ray can achieve it in 10 h and other assemblers in days. Ultimately, this kind of tool will help toward the goal of personalized medicine. In addition, being able to assemble multiple genomes (metagenome) will help in the similarity analysis of genomes against genomes.

II. Next Generation Sequencing Workflow

The process of sequencing a genome requires several steps. These steps are summarized in Fig.1. The first step consists in collecting a sample to obtain DNA. This sample is fragmented into small sequences. The small sequences are composed of a combination of the corresponding four nucleotides denoted by the letters {A, C, T, G} or {N} which denotes an unknown base. This process eventually produces the so called reads that are used by the different tools to assemble the genome [6].

Reads are in general 36 to less than 500 bases [2]. However, recent technologies allow for larger reads [7].

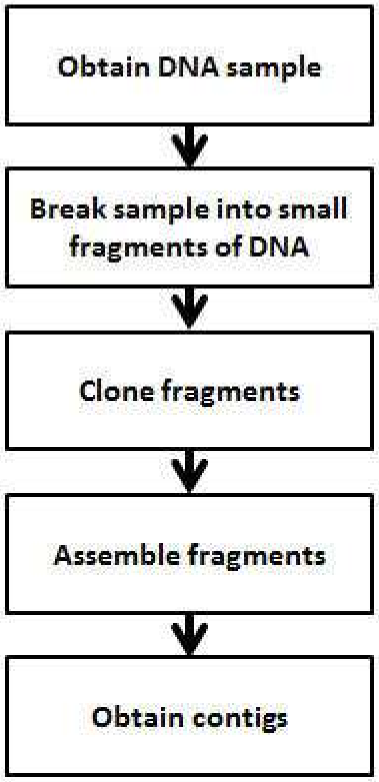

Fig.1. Summary of the process of sequencing

The steps summarized in Fig. 1 do not reflect all of the technologies required from an IT point of view to complete the NGS process. In the next section we briefly describe what they are and what components are required from beginning to end.

A. *IT Sequencing Technologies*

The IT user environment is shown in Fig. 2. This environment illustrates the multiple paths that data follows and the IT resources required. The tasks that are related to the sequencer are in yellow. The tasks identified as part of the data center are highlighted in green. Finally, desktop tasks are highlighted in blue.

Fig. 2 illustrates that there are three main components in the NGS environment.

- Sequencer
- Desktop
- Data Center

There are a variety of sequencers and different laboratories have adopted different technologies. Some of the commonly available sequencers come from companies like 454 Life Sciences, Illumina, Life Technologies, PacBio and Ion Torrent [8]. The flowchart in Fig.3 is based on the Illumina sequencer. One of the initial tasks in this diagram is to convert image files to Illumina base calling files [9].

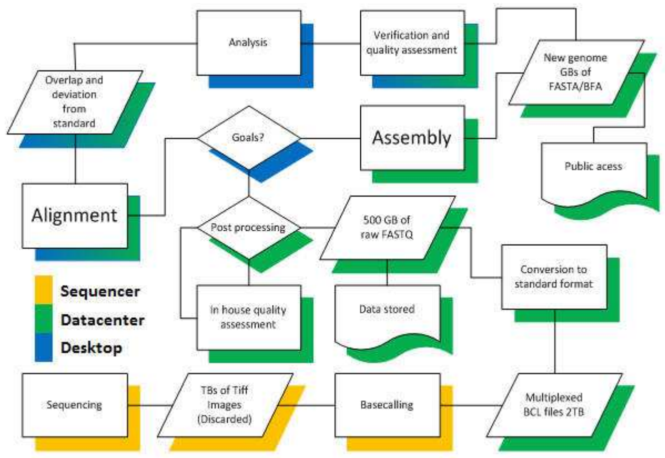
Analysis
Verification and quality assessment
Overlap and deviation from standard
New genome GBs of FASTA/BFA
Goals?
Assembly
Public acess
Alignment
Post processing
500 GB of raw FASTQ
Conversion to standard format
Sequencer
Datacenter
Desktop
In house quality assessment
Data stored
Sequencing
TBs of Tiff Images (Discarded)
Basecalling
Multiplexed BCL files 2TB

Fig. 2. NGS workflow based on IT requirements.

Subsequent file conversions can be accomplished using the resources provided by the data center. This is how it is performed in the flowchart presented in Fig. 2. As part of the tasks carried out in the data center, the diagram illustrates assembly as one of the key tools for genome sequencing. As previously indicated we illustrate the use of a *de novo* assembler, in particular, Ray [3]. However, before we proceed to describe Ray, we'll illustrate the data center components based on a system similar to the configuration used for Ray in this study.

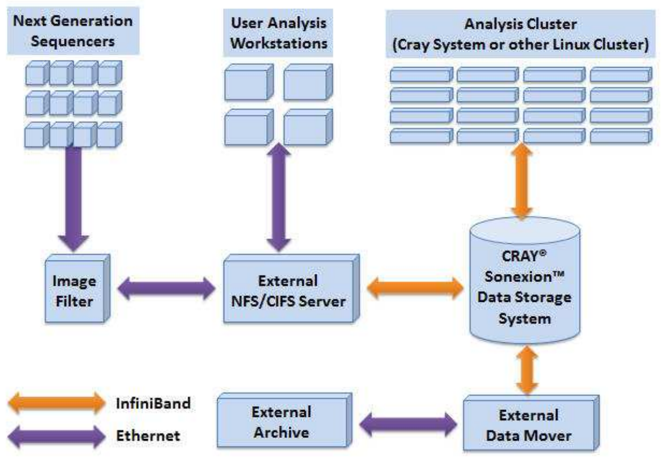
Next Generation Sequencers
User Analysis Workstations
Analysis Cluster (Cray System or other Linux Cluster)
Image Filter
External NFS/CIFS Server
CRAY® Sonexion™ Data Storage System
InfiniBand
Ethernet
External Archive
External Data Mover

Fig. 3 NGS data center configuration.

The Sanger FASTQ [10] files can be stored in the Cray Sonexion scale-out Lustre storage system for fast I/O [11]. All the applications are installed on the Cray server. The configuration uses InfiniBand to connect storage to the external data movers as well as the Cray server.

In this diagram the sequencer exports data to the workstation to filter images. Sonexion is mounted directly using Common Internet File System (CIFS) file-sharing protocol in Windows over Ethernet. No local disk cache is necessary for the workstation unless required. In this case the application is reading and writing data directly to the Sonexion via InfiniBand. The end-user workstation mounts Sonexion directly using CIFS over Ethernet Native with Lustre support for Hierarchical Storage Management (HSM), a data storage system that automatically moves data between storage media.

B. *Cray XE6 Server*

The Ray benchmarks shown in this study were run on a Cray XE6 system with AMD Opteron™ Interlagos IL-16 processors with a clock frequency of the core of 2.1 GHz. There are two sockets per node and the number of cores per socket is 16. The cache sizes correspond to L3 cache with 8 MB shared per die, L2 cache: 2 MB per core module and L1 cache data 16 KB, Instructions: 64KB. Peak performance per core is 16.8 GFlop/s. The interconnect is Cray Gemini, 3D Torus with 48 switch ports per Gemini chip and, 160 GB/s internal switching capacity per chip with latency of < 1.5 µs [12].

The Cray XE6 system runs the Cray Linux Environment (CLE) operating system on the login nodes and a lightweight kernel called CNL on the compute nodes. In this work we used CLE release 4.0 [12].

C. *Data*

The data used in this study to test a modified version of Ray 2.0.0 corresponds to the Illumina-based metagenomic sequencing, assembly and characterization of 3.3 million non-redundant microbial genes, obtained from 576.7 Gigabases of sequence, from fecal samples of 124 European individuals [13]. In this study we used the sample ERS006494 with runs ERR011117.sra through ERR0111123.sra.

III. *DE NOVO* ASSEMBLERS LANDSCAPE

There are many genome tools currently available. As a result of the fast growth in raw data coming from faster sequencing machines, there has also been an increase in the number of tools to assemble genomes. In whole-genome sequence assembly (WGSA) the goal is to assemble *contigs*. What characterizes *de novo* assembly is that there is no reference genome. As described in the literature [1], assemblers follow the heuristic that “if two reads share a sufficiently long subsequence then they can be assumed to have originated from the same location in the genome” [1]. This heuristic overlooks the case of large DNA sequence duplication in the genome and other confounding features, but is a good starting point.

A. *Greedy Algorithm-Based Assemblers*

These assemblers use the Greedy algorithm to identify the shortest common sequence[1]. This method builds a solution by continuously expanding fragments. Two nucleotide fragments with high overlap scores are merged into one larger fragment. This new fragment is added to the set of fragments, the process of merging fragments is repeated until no more fragments can be merged. The result of this process is a collection of *contigs* or larger DNA strings [1]. Advantages and disadvantages of this approach have been documented in the literature [1,6]. Narzisi and Mishra [1] have reported TIGR [14], PHRAP [15], CAP3[16] uses Greedy only for the first step, PCAP [17], and Phusion [18] as assemblers based on this methodology.

B. *Graph-Based Assemblers*

In simple terms, in this approach the genome corresponds to identifying a path in a graph [2]. The overlap-layout-consensus (OLC) was one of the early methodologies utilized to build graphs [6]. The graph is constructed by nodes representing reads and edges representing overlaps [1]. Examples of assemblers based on this methodology are: CELERA [19], CABOG [20], ARACHNE [21], Minimus [22], and Edena [23].

The advent of the next-generation sequencing technology introduced shorter reads. This made it difficult for assemblers based on the OLC methodology to cope with large genomes and insufficient overlaps [1]. This problem was overcome by the adoption of assemblers based on the de Bruijn graph methods [2]. This methodology considers nodes as k-mers and the edges are placed between pair of k-mers that show in consecutive reads [2]. For a comprehensive review of assemblers based on de Bruijn methods see [1,2].

IV. PARALLEL RAY: *DE NOVO* ASSEMBLER

De novo assembly is computationally intensive, requiring parallel execution for reasonable run times. Ray is parallelized using the message-passing interface (MPI) [24]. It is implemented using peer-to-peer communication. Ray is built on top of RayPlatform a message-passing-interface programming framework. Fig. 4 illustrates Ray’s architecture.

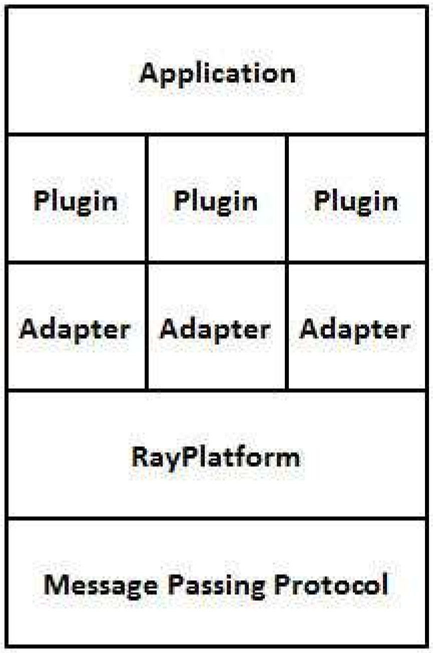

Fig. 4 Ray software-stack. RayPlatform is a programming framework that provides easy access to the message-passing protocol. Currently it is MPI.

RayPlatform is a development framework that simplifies distributing data via a lower level message passing protocol such as MPI. Content creation is done by creating plugins that

S.B. is recipient of a doctoral award from the Canadian Institutes of Health Research (200910GSD-226209-172830). JC is the holder of the Canada Research Chair in Medical Genomics.

can be added on the RayPlatform compute engine. The message-passing protocol becomes transparent to the developer. Ray bundles the following functionality that developers and users can access in one single package:

- *De novo* genome assembly
- *De novo* meta-genome assembly
- *De novo* transcriptome assembly
- Quantification of *contig* abundances
- Quantification of microbiome consortia members
- Quantification of transcript expression
- Taxonomy profiling of samples
- Gene ontology profiling of samples
- Profiling with any database of choice

Recently a mini-ranks programming model has been implemented in RayPlatform. Mini-ranks are a hybrid programming model where MPI ranks and mini-ranks co-exist. Mini-ranks are run on separate threads that are spawn by the MPI ranks. This enables the usage of a hybrid paradigm with MPI and threads.

A. *Message-Passing Main Loop*

The driver parallel loop is analogous as if each rank has its own message inbox and its own message outbox where received messages go in the inbox and sent messages go in the outbox. This loop is illustrated in Fig. 5.

```
while(running){
    receiveMessages();
    processMessages();
    processData();
    sendMessages();
}
```

Fig. 5. Main parallel construct in Ray.

This can be considered the general architecture utilized in Ray.

B. *Distributed Storage*

Distributed storage engine used by Ray is a distributed sparse hash table that uses these features:

- Incremental resizing
- Double hashing
- Buckets are in groups
- Distributed

The run-time options include `hash-table-buckets` that set the initial number of buckets which has to be a power of 2. The default value: 262144. `Hash-table-buckets-per-group` sets the number of buckets per group for sparse storage, default value is 64 and it must be between 1 and 64. `Hash-table-load-factor-threshold` sets the load factor threshold for real-time resizing, default value is 0.6 and must be between 0.5 and 1. `Hash-table-verbosity` activates verbosity for the distributed storage engine.

C. *Ray Functionality*

In addition to its parallel capabilities, Ray can be utilized as a k-mer counter. It also builds a k-mer graph (subgraph of a full de Bruijn graph). Once the graph has been assembled, Ray finds paths in the graph. The code is implemented in C++ and it utilizes MPI version 2.2. Various network interconnects are supported via MPI libraries. K-mers are using distributed sparse hash tables with double hashing via `MyHashTable`. Ray also utilizes smart pointers and garbage collection using real-time memory defragmentation & compaction via `DefragmentationGroup`. To avoid storing most of the erroneous k-mers in memory Ray uses a distributed Bloom filter. Ray utilizes virtual communication (`VirtualCommunicator`). Ray supports substitution DNA sequencing errors (Illumina) and very short indels DNA sequencing errors (Pacific Biosciences, 454) [8].

D. *Ray Profiling*

Ray generates some profiling information by default without introducing overhead. The first is the network testing. Before doing the biologically-work, Ray tests the network. Each MPI rank sends a number of messages to measure the latency of a round trip. So the point-to-point latency is actually half this value. The result is written to RayOutput/NetworkText.txt. The network test can also dump detailed data with the option -`write-network-test-raw-data`.

RayPlatform implements an array of communication graphs such as complete, de Bruijn, Kautz, hypercube, polytope, random, group [3,4]. It can be activated with -route-messages. The model is set with -`connection-type <type>`. The model complete is the default. The best is the hypercube/regular polytope because it can do load balancing of routed messages. This is useful on supercomputers with network hardware that does not support too many communication peers. The Cray XE6 does not require this option because the Gemini Interconnect [25] is already optimized for any-to-any communication patterns with a hardware torus.

In Ray, all the code paths must use the format imposed by RayPlatform. All the code is put inside functions/methods starting with call_followed by the signal name. The Ray code actually runs in a supervisor implemented in RayPlatform. It deleguates the signals such as a received message, or a slave mode or master mode that must be executed for one tick. All scheduling information is written in RayOutput/Scheduling/*. These reports provide, for each MPI rank, with the granularity in nanoseconds, the number of messages sent/received per second, the number of ticks in the supervisor, the total number of milliseconds for any given slave mode or master mode, and so on. Statistics on messages sent or received are written in RayOutput/MessagePassingInterface.txt

The option -`show-communication-events` activates the reporting of all communications (send and receive operations). The option `-run-profiler` will run Ray in slow mode, but the supervisor will collect a lot of profiling information. There is also a compile time option to add collectors in the code. The option is `PROFILER_COLLECT=y` (or `-D CONFIG_PROFILER_COLLECT`).

The file RayOutput/ElapsedTime.txt contains a human-readable report of the time required by each step. In the standard output, Ray reports its memory usage for each MPI rank. When compiling the code, turning on link time optimization and using the native instruction set is suggested.

V. RESULTS AND DISCUSSION

In general, multiple tools have been developed as a result of the fast growth in raw data coming from faster sequencing machines. In particular, there has been an increase in the number of tools to assemble genomes. However, work is required to assess the accuracy of these new assemblers. The first set of results reported by the team that developed Ray appeared in 2010 [3] and more recently in the Assemblethon 2 paper [5]. In their paper Boisvert et al. [3] have compared Ray against some of the state-of-the-art assemblers. They compared against ABySS [4,5], EULER-SR[6] and Velvet [7]. To assess the quality of their results they selected a set of metrics such as the number of *contigs* having at least 500 base pairs (bp), the number of bp, the mean size of *contigs*, the N50, the largest *contig* size, the genome coverage, incorrect *contigs*, mismatches and indels. The datasets they selected correspond to S. pneumonia R6 divided into three subsets: SpSim, SPErSim and SpPairedSim. For further details and additional meaning of the metrics and datasets the reader should refer to [3].

The second key Ray paper corresponds to their work on *de novo* metagenome assembly [3]. In this paper they have illustrated that Ray can accurately assemble a three-billion-read metagenomic experiment in 15 hours with 1,024 cores using 1.5 GB of memory per core [3].

Table 1. Ray profile (ERS006494).

MPI Tasks	**128**	**256**	**512**	**1024**
	Elapsed Time in Sec.			
Network testing	10	29	66	78
Counting sequences to assemble	24	**81**	446	24
Sequence loading	843	747	433	**229**
K-mer counting	1174	563	258	**110**
Coverage distribution analysis	4	2	1	1
Graph construction	2580	1236	**578**	254
Null edge purging	**1217**	1072	1009	986
Selection of optimal read markers	861	420	210	**119**
Detection of assembly seeds	810	405	218	134
Est. outer distances for paired reads	84	54	33	22
Bidirectional extension of seeds	1458	897	585	638
Merging of redundant paths	1422	1050	515	456
Generation of *contigs*	37	39	36	37
Scaffolding of *contigs*	616	**434**	315	305
Counting sequences to search	0	0	0	0
Graph coloring	2	2	0	0
Counting *contig* biological abundances	103	78	70	42
Counting sequence biological abundances	0	0	0	1
Loading taxons	2	1	1	0
Loading tree	2	2	0	0
Processing gene ontologies	4	2	1	1
Computing neighbourhoods	0	0	1	0
Total	11254	7117	**4807**	3478

This work has been extended to run on Cray supercomputers. Table 1 shows the run time profile produced by Ray. This is the default profiling with little or no overhead. The first line in Table 1 corresponds to each MPI rank sending a number of messages to measure latency of a round trip. When the number of MPI ranks increase, the elapsed time decreases.

Table 1 also illustrates how CPU usage is distributed among the different tasks in the code. The top tasks that consume CPU for more than 1000 seconds correspond to k-mer counting, graph construction, null edge purging, bidirectional extension of seeds, and merging of redundant paths. K-mer counting and graph construction scale linearly. On the other hand, the scalability for null edge purging is limited after 128 MPI tasks. This behavior was due to a bug and it has been fixed in later versions. The Bidirectional extension shows a parallel speed up to 512 cores.

Fig. 6 shows the total scalability of Ray for this particular example. Ray can scale to 1024 cores. Additional optimization in some of the timing routines could improve scalability. In this work optimization for network routing, memory allocation, and tasks allocations showed a 20% improvement as illustrated in Fig. 6. Better task allocation was carried out via Cray's aprun options [12].

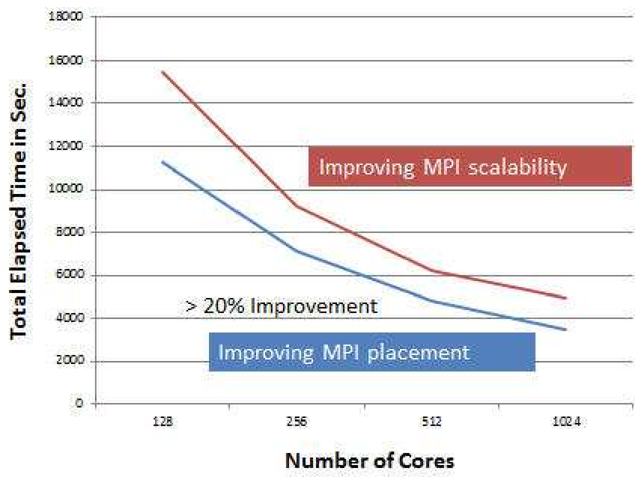

Fig. 6. Illustrates Ray scalability using the Qin et al. human gut microbiome dataset (ERS006494).

Finally, as part of the Assemblathon 2 [3] competition many assemblers currently utilized throughout the world were tested. Some of the relevant questions investigated included speed, hardware requirements, flexibility toward new read technologies, and composition of the assembled sequence. The objective as described in the Assemblathon 2 paper [5] was to test state-of-the-art methods. The data used for testing consisted of three vertebrate species: bird, fish and snake. This careful and systematic study concluded by ranking Ray very highly. "The Ray assembly was ranked 1st overall, and also ranked 1st for all individual measures except multiplicity (ranked 7th)".

VI. Conclusions

The advances in Next Generation Sequencing machines technology have provided critical tools for deciphering DNA sequences of vital importance in biology. The cost of one Mb of DNA sequence has gone down from about $5,000 in 2001 to approximately $0.78 in 2009 [26]. On the other hand, this has come at a cost. Assembler programs have been created to assist in the process of assembling genomic data. However, as data coming from sequencers outpaces Moore's law, it is critical to develop tools and procedures that can accurately and efficiently keep pace with the data production. Ray provides to the sequence assembly field the next quantum leap into the development of the next generation of assemblers by mixing sequencing technologies. It reduces the number of errors and the number of *contigs*. However, more importantly by recognizing that key to the design of an assembler is to leverage the architecture underneath the software. Ray represents a major step forward in overcoming some of the major challenges facing genome assembling today. This is particularly true for large datasets that otherwise are intractable.

Acknowledgment

We would like to thank Cray Inc for providing all the computation resources and funding for MK internship. S.B. is recipient of a doctoral award from the Canadian Institutes of Health Research (200910GSD-226209-172830). JC is the holder of the Canada Research Chair in Medical Genomics. We also would like to thank the to thank the Compute Canada / Calcul Canada (nne-790-ab) and the Minnesota Supercomputing Institute for allocating resources for this project. Finally, the Biomedical Informatics and Computational Biology Program, University of Minnesota Rochester is acknowledged for continued support.

.

References

[1] G. Narzisi and B. Mishra, "Comparing *De novo* Assembly: The Long and Short of It," PLoS ONE, vol. 6, pp. 1–14, April 2011.

[2] J. Henson, G. Tischler, and Z. Ning, "Next-Generation Sequencing and Large Genome Assemblies", Pharmacogenomics, vol. 13, pp. 901-915, 2012..

[3] S. Boisvert, F. Laviolette, and J. Corbeil, "Ray: Simultaneous Assembly of Reads from Mix of High-Throughput Sequencing Technologies.", Journal of Computational Biology, vol 17, pp. 1519-1533, 2010.

[4] S. Boisvert, F. Raymond, E. Godzaridis, F. Laviolette, J. Corbeil, "Ray Meta: Scalable *de novo* Metagenome Assembly Profiling", Genome Biology, vol 13, pp. 1-33, 2012.

[5] K R. Bradnam, J N. Fass, A Alexandrov, P Baranay, M Bechner, İ Birol, S Boisvert, J A. Chapman, G Chapuis, R Chikhi, H Chitsaz, W- Chou, J Corbeil, C Del Fabbro, T. R Docking, R Durbin, D Earl, S Emrich, P Fedotov, N A. Fonseca, G Ganapathy, R A. Gibbs, S Gnerre, É Godzaridis, S Goldstein, Ma Haimel, G Hall, D Haussler, J B. Hiatt, I. Y. Ho, J Howard, M Hunt, S D. Jackman, D B. Jaffe, E Jarvis, H Jiang, S Kazakov, P J. Kersey, J O. Kitzman, J R. Knight, S Koren, T-Wah Lam, D Lavenier, F Laviolette, Y Li, Z Li, B Liu, Y Liu, R Luo, I MacCallum, M D. MacManes, N Maillet, S Melnikov, B M. Vieira, D Naquin, Z Ning, T D. Otto, B Paten, Octávio S. Paulo, Adam M. Phillippy, Francisco Pina-Martins, Michael Place, Dariusz Przybylski, Xiang Qin, Carson Qu, Filipe J. Ribeiro, Stephen Richards, Daniel S. Rokhsar, J. Graham Ruby, Simone Scalabrin, Michael C. Schatz, David C. Schwartz, Alexey Sergushichev, Ted Sharpe, Timothy I. Shaw, Jay Shendure, Yujian Shi, Jared T. Simpson, Henry Song, Fedor Tsarev, Francesco Vezzi, Riccardo Vicedomini, Jun Wang, Kim C. Worley, Shuangye Yin, Siu-Ming Yiu, Jianying Yuan, Guojie Zhang, Hao Zhang, Shiguo Zhou, Ian F. Korf1, "Assemblathon 2: evaluating *de novo* methods of genome assembly in three vertebrate species.", arXiv:1301.5406.

[6] M. Pop, "Genome Assembly Reborn: Recent Computational Challenges", Briefings in Bioinformatics, vol. 6, pp. 354-366, 2009.

[7] M. Heger, "PacBio Users Report Progress in Long Reads for Plant Genome Assembly, Tricky Regions of Human Genome", In Sequence, March 6, 2013.

[8] M. L Metzker,"Sequencing Technologies – The Next Generation", Nature Review, vol. 11, pp. 31-48, 2010.

[9] M. Kircher, U. Stenzel, and J. Kelso, "Improved Base Calling for the Illumina Genome Analyzer Using Machine Learning Strategies", Genome Biology, vol. 10, R83, 2009.

[10] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, Nucleic Acids Research, vol. 38, pp. 1767-1771, 2009.

[11] Solution Brief, "Cray Storage Solutions fro Life Sciences", http://www.cray.com/Assets/PDF/products/storage/SolutionBrief_SonexionandLifeSciences.pdf

[12] Cray Applications Developer's User Guide, S–2396–601, Cray Inc, 2011.

[13] J. Qin, et al., " A Human Gut Microbial Gene Catalogue Established by Metagenomic Sequencing", Nature, vol. 464, pp.59-65, 2010.

[14] G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage, "TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects.", Genome Science and Technology, vol. 1, 9-19, 1995..

[15] P. Green, Phrap documentation. Available: Http://www.phrap.org, 1996./

[16] X. Huang, and A. Madan A, "CAP3: A DNA Sequence Assembly Program.", Genome Research vol. 9, pp. 868–877, 1996.

[17] X. Huang, J. Wang, S. Aluru, S.P. Yang, I. Hillier, " PCAP: AWhole-Genome Assembly Program.", Genome Research vol. 13, 2164–2170, 2003.

[18] J.C. Mullikin, and Z.Ning, "The Phusion Assembler." Genome Research vol. 13, 81–90, 2003.

[19] E. W. Myers, G. G. Sutton, A. L., Delcher, I. M. Dew, D. P. Fasulo, "A Whole-Genome Assembly of Drosophila", Science, vol. 287, pp. 2196-2204, 200.

[20] J. R. Miller, A. L. Dekker, S. Koren, E. Venter, B. P. Walenz, Aggressive Assembly of Pyrosequencing Reads with Mates", Bioinformatics, vol. 24, pp. 2818-2824, 2008.

[21] S. Batzoglou, D. B. Jaffe, K. Stanley, J. Buttler, S. Gnerre, "ARACHNE: A Whole-Genome Shorgun Assembler", Genome Research vol. 12, pp. 177-189, 2002.

[22] D. Summer, A. Delcher, S. Salzberg, M. Pop, "Minimus: a Fast, Lightweight Genome Assembler, BMC Bioinformatics vol. 8, pp. 64, 2007.

[23] D. Hernandez, P. Francois, L. Farinelli, M. Sters, J. Schrenezel, "*De novo* Bacterial Genome Sequencing: Millions of Very Short Reads Assembled on a Desktop Computer", Genome Research, vol. 18, pp. 802-809, 2008.

[24] MPI : A Message-Passing Interface Standard Version 2.2, Message Passing Interface Forum, September 4, 2009

[25] R. Alverson, D. Roweth, L. Kaplan, "The Gemini System Interconnect", High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium, 18-20 Aug. 2010.

[26] Human Genome Research Institute, http://www.genome.gov/sequencingcosts/