ORIGINAL ARTICLE

HUMAN ANALYSTS AT SUPERHUMAN SCALES:

What Has Friendly Software To Do?

Élénie Godzaridis,[1] Sébastien Boisvert,[2,3]* Fangfang Xia,[4] Mikhail Kandel,[5,6] Steve Behling,[6] Bill Long,[6] Carlos P. Sosa,[6,7] François Laviolette,[8] and Jacques Corbeil[2,3]*

Abstract

As analysts are expected to process a greater amount of information in a shorter amount of time, creators of big data software are challenged with the need for improved efficiency. Ray, our group's usable, scalable genome assembler, addresses big data problems by using optimal resources and producing one, correct and conservative, timely solution. Only by abstracting the size of the data from both the computers and the humans can the real scientific question, often complex in itself, eventually be solved. To draw a curtain over the specific computational machinery of big data, we developed RayPlatform, a programming framework that allows users to concentrate on their domain-specific problems. RayPlatform is a parallel message-passing software framework that runs on clouds, supercomputers, and desktops alike. Using established technologies such as C++ and MPI (message-passing interface), we handle the genomes of hundreds of species, from viruses to plants, using machines ranging from desktop computers to supercomputers. From this experience, we present insights on making computer time more useful—and user time much more valuable.

Introduction

It is no secret that life sciences have made their entry into the world of big data.[1–7] Modern sequencing technologies produce terabytes of data in the span of a few days.[8,9] Many large-scale projects have been set up to take advantage of these technologies in fields as varied as ecology,[10,11] infectious diseases,[12–15] and phylogeny.[16–22] Analysts have lamented about the unyielding nature of their tools.[23,24] Sequencing the human genome might once have taken 10 years, cost millions, and yielded countless appearances in mainstream media, but today it takes 1 week, costs approximately $10,000, and is barely worth a mention in the Methods section. Yet the classes of algorithms used to piece it together are largely unchanged.[25–31]

An approach used to deal with the large scale of the data produced is to similarly increase the scale of the resources devoted to it, for example, by using supercomputers.[32] Several genome assemblers exist that can use an MPI (message-passing interface[33]) library to scale over thousands of computing nodes.[34–39] Kiki is also a genome assembler that uses MPI (https://github.com/GeneAssembly/kiki). Ray, which we developed, parallelizes every step of its execution.

[1]*Department of Strategic Technology, Bentley Systems, Inc., Quebec, Canada.*
[2]*Faculty of Medicine, Laval University, Québec, Canada.*
[3]*Department of Infectious and Immune Diseases, CHUQ Research Center, Québec, Canada.*
[4]*Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, Illinois.*
[5]*Department of Electrical and Computer Engineering, University of Illinois, Champaign, Illinois.*
[6]*Cray Inc., St. Paul, Minnesota.*
[7]*Biomedical Informatics and Computational Biology (BICB), University of Minnesota Rochester, Minneapolis, Minnesota.*
[8]*Department of Computer Science and Software Engineering, Faculty of Science and Engineering, Laval University, Québec, Canada.*
*These authors contributed equally to this work.

Notably, ABySS (genome assembler that uses MPI for building the genome graph) was used to assemble the very large genome of the white spruce.[40]

The genome assembly problem is commonly represented as a graph problem.[26,28,31,41,42] With the input sequence reads, a graph is constructed, and then paths are discovered in this graph. There are different types of graphs for genome assembly, and it was shown that the two popular ones (de Bruijn graph and unitig/overlap graph) are closely connected.[31] Genome assembly is difficult because repeats in genomes are usually longer than the readouts obtained by DNA sequencing.

The methods used in Ray to parallelize the assembly problem over as many processing elements as are available are readily applicable to most types of graph problems. For example, the approach of dividing the exploration of a graph between several actors by giving them different starting points was first reported in 2009 by Jackson and colleagues[36] using an IBM Blue Gene/L.

"A SIGNIFICANT PART OF THE CHALLENGES INVOLVED IN WRITING SCALABLE, PARALLEL SCIENTIFIC SOFTWARE ARE COMMON TO ALGORITHMS AND APPLICATIONS."

Here we cover not only increasing scalability through parallelization but also various constraints facing big data analysis software such as ours, and we present the ways we have counteracted them. These constraints cover the costs in time and resources of running the software, of course, but also the incidental costs to the user of setting up the analysis environment, accessing the results, and choosing the parameters.

For example, one of the foremost issues to be taken into consideration when publishing scalable analysis software for large datasets is that of portability. Not everyone interested in scaling out their analysis can or will use supercomputers; indeed, there is a growing interest in running all kinds of biological analyses on the cloud.[43–45] Ray's design has allowed it to be easily ported over to a cloud environment. As an example, Ray is available in the DNAnexus platform (https://platform.dnanexus.com/app/ray), which allows apps to be started with a few mouse clicks. Even among supercomputers, special care has to be taken so that the software is flexible enough to maintain its performance on different types of machines such as the Cray XE6, the IBM Blue Gene/Q, and university computing infrastructures. It has to be friendly enough so that the time saved for the user by the increased performance is not offset by the time cost of getting the software to run in the first place.

We also introduce Ray Cloud Browser, a web-based genome visualization tool that allows users to more closely examine and validate their results.

Definitions

Here, graphs are utilized for representing the genomic data and for optional communication tasks such as message routing. A directed graph $G=(V, A)$ is a set of vertices V and a set of directed relationships (called arcs) $A \subseteq V \times V$. A de Bruijn graph is a directed graph defined for an alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ and an integer $k \in \mathbb{N}$. Its vertices are the sequences of k elements from the alphabet. There is an arc $(u,v) \in$ between two vertices u and v if and only if the last $k-1$ elements of u and the $k-1$ elements of v are the same. In genome research, the alphabet is $\Sigma = \{A, T, C, G\}$ (nucleotides), and the integer k usually falls roughly within the range from 19[46] and 150—the current maximum length for the sequencing platform that provides the highest yield. In the problems that Ray aims to solve, the graph is a distributed subgraph (meaning that its vertices are not stored all in one location). The complete de Bruijn graph is not required because only the DNA sequences that occur in a given sample are relevant—any genome contains a small proportion of all possible sequences of length k if k is large enough. A *path* P of a graph is an ordered list of vertices $\langle v_1, v_2, v_3, \ldots, v_{|P|} \rangle$ such that any two consecutive vertices v_i, v_{i+1} are linked by an arc. Here, we define a traversal as the process by which a path is constructed. More precisely, in the context of *de novo* genome assembly, the traversal is guided by heuristics based on spatial information provided by input data.[35] Reads (usually at least 100 nucleotides) are larger than vertices (for example $k=31$). Furthermore, heuristics also use long-range information provided by paired reads and mate pairs.

A parallel job comprises an ordered set of processes, which are called ranks. A message is a unit of information passed from one rank to another. The granularity refers to the trade-off between the amount of communication and the amount of processing in any given period. A coarse granularity means that processing is not heavily interleaved with communication and that communication occurs once in a while. A fine granularity is characterized by a high level of overlap between communication and processing.

Building a Message-Passing Framework

A significant part of the challenges involved in writing scalable, parallel scientific software are common to algorithms and applications. Because factors such as the size of messages, frequency of communications, and number of potential recipients are largely dependent on the scale of the problem rather than its specific application domain, they are ideal candidates for abstraction.

RayPlatform, the framework Ray is built on (see Fig. 1), abstracts and optimizes three different aspects of the performance of an application built on message passing, that is, nontrivially parallel: communications between the application's ranks (see Talk is Not Cheap section), task switching and scheduling (see A Question of Efficiency), and data access and retrieval (see The True Weight of Big Data). The source code of RayPlatform is licensed under the GNU Lesser General Public License (version 3) and is available at https://github.com/sebhtml/RayPlatform. A minimalist example using RayPlatform is available at https://github.com/sebhtml/RayPlatform-example.

Ray 2.2.0 contains 26 plugins that are registered on the RayPlatform using its application programming interface (API). In that regard, the source code of Ray focuses on solving biological problems such as *de novo* genome (or metagenome) assembly, taxonomic profiling, and other tasks[39] instead of solving framework-related problems, which are handed over to RayPlatform. The source code of Ray is licensed under the GNU General Public License (version 3) and is available at https://github.com/sebhtml/ray.

Talk Is Not Cheap

As demonstrated before,[47] communication overhead is a looming specter—casting its shadow over the hopes of supercomputing scalability and performance. When tackling a problem with many nodes (motherboards), though now equipped with more memory and processing threads, it is not uncommon to witness a slowdown in computation. Misplaced data, stalled execution flow, or high-latency networks can all affect the entire computing system. One strategy to reduce the number of messages that transit in an interconnect is to use message aggregation.

The best supercomputers available implement highly optimized hardware message routing: Cray XT/XE[48] and IBM Blue Gene/Q[49] use multidimensional tori while cutting-edge Cray XC[50] and IBM PERCS[51] systems use a transparently managed dynamic hierarchy of all-to-all networks. In these cases, there is dedicated proprietary hardware that ensures messages arrive in a timely and consistent manner, along with packaging expertise to understand the proper placement of computer components. However, not every computing facility is equipped with such sophisticated communication networks and some interconnects have poor performance in the presence of an any-to-any communication pattern. With an any-to-any communication pattern, any rank can send a message directly to any other rank. To counteract this effect,

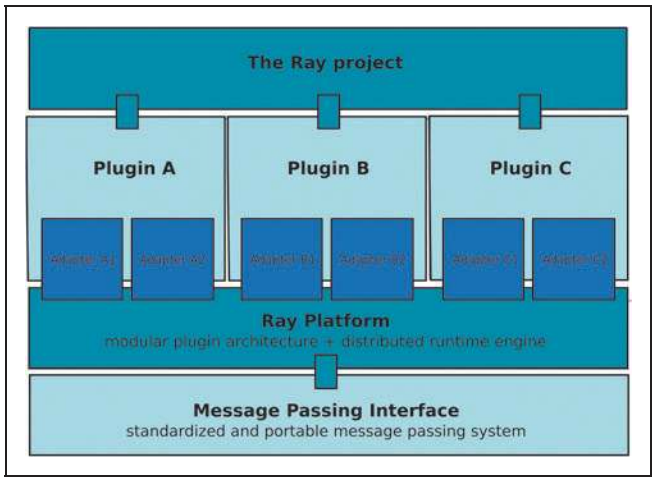

FIG. 1. The relationship between Ray and RayPlatform.

there are several levels of control over communication intensity and efficiency in RayPlatform. First, different kinds of software routing graphs can be used to alleviate the load on some parts of the network (see Ref.[52] for an introduction to this subject) in order to bring down the numbers of direct physical connections each rank is required to maintain. This facility can also balance the messages that transit through a rank. In RayPlatform, software message routing is a service provided by a virtual message router. In addition to all-to-all and per-node routing schemes, RayPlatform supports Kautz graphs (offering the smallest diameter), de Bruijn graphs, polytope routing,[52–54] and torus graphs (modeling the physical interconnect of many supercomputers).[49,50] On Université Laval's supercomputer, Colosse (960 compute nodes, 7680 processor cores), this yields a five-times-lower latency (500 to 90 microseconds) for a 512-rank job. Ray components use a tighly coupled communication pattern. Therefore, reducing the point-to-point communication latency reduces the computation time.

Messages are aggregated to get economies of scale, and ranks cannot receive more than one message per cycle. This allows ranks to use the work quantum they use each cycle to advance their own problem, rather than just answer requests for data from other ranks. This communication strategy works well when the work is balanced across ranks. In Ray, work is derived from ownerships of vertices in the de Bruijn graph. Ownership is determined by hashing a vertex, and therefore the vertices in the graph are uniformly distributed on ranks, and so is the communication. The data are stored in an in-memory distributed hash table (DHT) with double hashing[55] and incremental resizing.

"MISPLACED DATA, STALLED EXECUTION FLOW, OR HIGH-LATENCY NETWORKS CAN AFFECT THE ENTIRE COMPUTING SYSTEM."

RayPlatform also includes a prototype for mini-ranks to adequately leverage systems with a higher number of cores per machine but a slower interconnect between individual nodes. Mini-ranks are ranks, which is to say they send messages to each other and do their own computing. One of the differences is that mini-ranks can be pooled together inside a process to reduce the number of MPI processes doing calls to MPI functions. There is no routing between mini-ranks of the same rank, and the communication is guaranteed to be in-memory, which lessens the load on the network. This hybrid programming model allows us to extend RayPlatform's paradigm to handle varying combinations of network and computing performance and to keep using the MPI paradigm to write the software. In RayPlatform, mini-ranks are implemented as POSIX threads.

A Question of Efficiency

Because the kind of computing power required for big data applications is the kind of computing power that, very often, is only available for rental (by-the-hour usage on shared infrastructure), as is the case in clouds and academic clusters, one source of waste is paying for nodes that don't do their fair share of work. As a result, load balancing is often a prime concern, and it's also difficult to manage. Even when no data dependency exists between the many execution threads, efficiency requires splitting the data so that the varying amounts of work required by each datum add up to the same amount for each rank; this would be a variation on the NP-hard knapsack problem.[56] In molecular dynamics, this very problem was addressed in CHARM++ using task migration, where workers move between computing nodes in order to maintain a balanced load across every compute nodes.[57,58]

> "RAYPLATFORM ALSO INCLUDES A PROTOTYPE FOR MINI-RANKS TO ADEQUATELY LEVERAGE SYSTEMS WITH A HIGHER NUMBER OF CORES PER MACHINE BUT A SLOWER INTERCONNECT BETWEEN INDIVIDUAL NODES."

While traversing a path in the graph does not necessarily depend on intermediate results obtained by the computation of any other path, the problem still requires constant communication to deal with the massive amount of data. If there were, for example, dedicated data stores that provided all of the necessary sequences to every rank – eliminating its dependency on other compute units – there would need to be more cores that would spend the vast majority of their time waiting for messages, which is inefficient.

RayPlatform has a relatively conservative approach when it comes to scheduling. All ranks run the same code, and with the exception of very few moments in the algorithm (when all the work is done, for example), do the same work on different parts of the data. Each rank has an allocated opportunity, once per cycle, to receive a message, and then process it. When more than one message is available, only the first is read. The remaining messages will wait for the next iteration. Each rank has a quantum of work that is really advancing one of several ongoing tasks, each assigned to one worker of a pre-allocated pool. For example, a quantum of work can consist of extracting a subsequence from a DNA read. The rank then sends messages, and if a task that is part of the rank's work is waiting on another rank's reply, it is suspended and other tasks are chosen instead. The granularity is really small to allow each rank to timely probe incoming messages as computation and communication take place in the same thread on each rank.

A state machine manages the steps of the program. These steps and events are grouped into three types: master modes, slave modes, and message tags. Associated handlers are registered with the RayPlatform API by the application at running time. All steps can be done in parallel, and when all the work for a step is done, and/or additional conditions are met, all ranks advance to the next state. The advantage of this method is that it is very generic: no prior knowledge regarding data dependencies within a given step of the algorithm is required, or at least these dependencies are delegated from the framework to the application. For example, from a data point-of-view, one first needs to load the sequence reads in memory before building a distributed de Bruijn subgraph. This data dependency is managed by the state machine, and a given rank knows when it's ready. When everyone is ready, the next state is affected. The scheduling scheme in Ray is broadly applicable, provided that tasks can be encoded with granularity.

The True Weight of Big Data

A problem with big data, especially when paired with large computing needs, is that bigger data is harder to move. A single machine can hardly hold all of the data, not even all of the data it needs, at the same time, and depending on data structures and network interconnects, acquiring that data is costly.

There are several ways around this problem. Faster, shared data stores can be used in the form of large amounts of shared memory or specific, high-performance networked file systems. They are often expensive, and because they require specific maintenance they are not generally offered as part of cloud services, nor are they available by merely

connecting together bits of commodity hardware. For data transport, DNA sequencing instruments can send data in increments to the cloud where they can be processed (examples: Galaxy,[59] DNAnexus,[60–62] Illumina BaseSpace,[63] GenomeSpace[64]).

The data can be preprocessed and structured to make retrieval faster, perhaps at the cost of deletion or insertion times. Several solutions oriented toward the big data market exist in terms of providing fast, concurrent databases with good access times,[65–67] and this might be the best option for cloud storage provided there is a value in keeping the structured data after running an algorithm on it.

"DATA CAN ALWAYS BE PREPROCESSED, STATE CAN ALWAYS BE STORED, AND WHEN IN DOUBT, YOU CAN ALWAYS ADD MORE CORES."

Finally, locality of similar data can be improved. Depending on the data type, there are algorithms that can do this—it is a solved problem on three-dimensional meshes.[68] This particular approach is very data dependent, and similarity of the data is often what needs to be computed in the first place. Ray keeps the data stored in the same ranks that are responsible for the computation, distributed uniformly. With fast interconnects or efficient routing algorithms, because the individual vertices in the graph are small, this is not a great problem during execution.

Keeping a Large-Scale Application Useful

The proper use of computing time and resources is an area of focus in high-performance computer software research. However, in a world where a machine's time costs less than a dollar an hour, and human wages are climbing globally, it is worthwhile to consider the gain in efficiency from the proper use of human resources. To keep an application useful several levers can be exploited: ease of set-up, accuracy of results, and availability of information are just three. We will describe how Ray benefits from, and how it gained, its usability.

Human Learning: An Embarrassingly Nonparallel Problem

From the standpoint of a software developer, most tasks are ripe for optimization. Data can always be preprocessed, state can always be stored (for example, using checkpoints), and when in doubt, add more cores. This rule does not hold for set-up time, because adding more analysts won't make them understand the software's parameters and chosen file format faster.

Genomics knows many file formats for sequences (fasta, fastq,[69] sff,[70] color-space[71]), alignments (sam,[72] bam,[73] psl[74]), variations (vcf,[75] gvf[76]), and annotations (gff,[77] bed,[78] wig). Raw DNA sequences from published research are archived in the Sequence Read Archive.[79] The packaging of data is a problem for big data analysts because in the absence of standards, every software will design their own (often overfitted to their own problem). In the presence of standards, analysts will feel constricted by the data representation and have trouble conceptualizing a way to coerce the results into a format that is usable by other programs.

Ray accepts all kinds of file formats and outputs in the standard fasta format. Its plugins that perform taxonomic profiling[39] output data in XML. For genome assembly, Ray has only one important parameter: the length of the subsequences (*k*-mers), which will be familiar to anyone experienced with assembly and is easily explained to a biologist (see Refs.[46,80,81]). Other parameters are adjusted automatically with the data.

Better Results or Faster Results?

Much is made to improve software performance, especially on pay-per-use resources. More often, users, especially in the scientific field, will demand that the accuracy of the results match that of the more commonplace imperative algorithms they used to run on their desktops.

When our team took part in the Assemblathon 2 challenge (http://assemblathon.org), which aims to identify the best DNA assembly methods for vertebrate genomes, it was found that no assembler managed to obtain consistently best-in-class results on all datasets. However, Ray was ranked first overall for the snake genome dataset.[82]

While Ray's primary selling points are its great scalability (see Fig. 2 and Ref.[39]) and portability, not even being fast on larger data can let analysis software off the hook altogether. And we all have to ensure at least state-of-the-art precision. While this can be difficult for particularly interlinked data, the task was facilitated by the abstraction provided by RayPlatform and its state machine.

Being Open Closes No Doors

Ultimately, the best way to make the most of users' time is to listen to them; Ray is open-source software, and in addition the team maintains two mailing lists and handles tracking issues. This offers an interface for users to express their problems or propose their ideas and promotes Ray's (and, by extension, RayPlatform's) growth toward ever more efficiency by keeping its purpose clear—to provide good data analysis

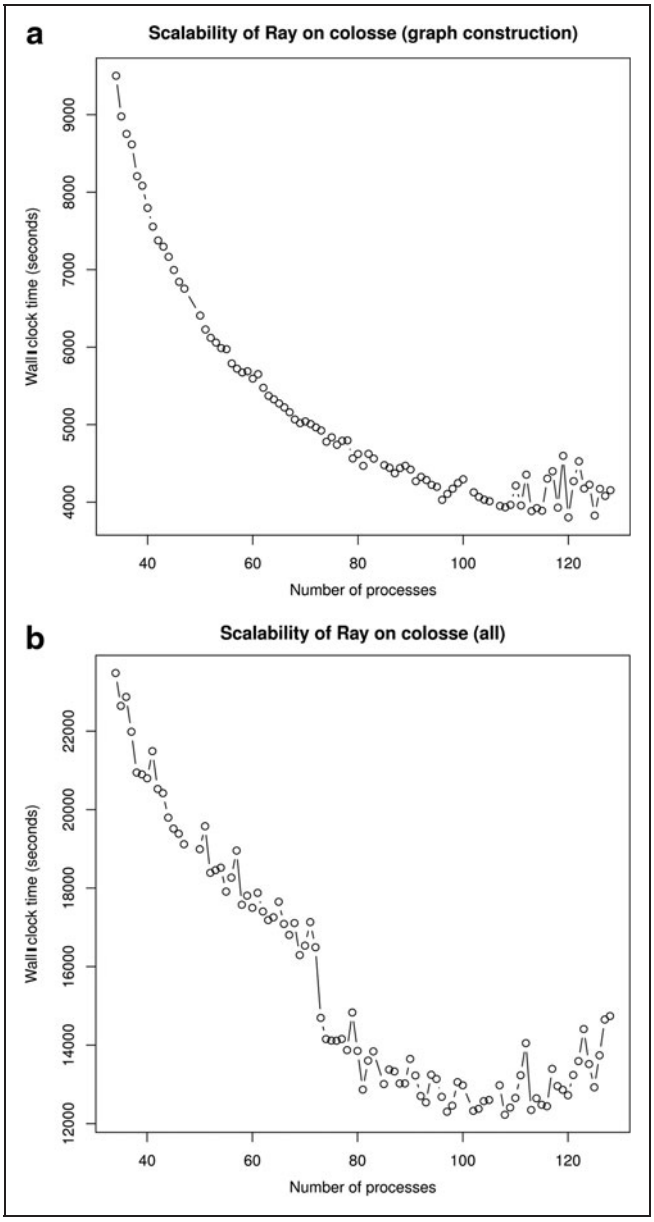

FIG. 2. Scalability of Ray: The scalability was measured in the sample SRS011098 from the Human Microbiome Project.[10] Using the same input data, different times were obtained by varying the number of processes. Panel **(a)** shows the time required to build the genome graph for various numbers of processes. In this panel, the time asymptote is 4,000 seconds because some parts of the computation are not parallel (Amdahl's law). In panel **(b)** the time asymptote is reached at a lower number of processes because the time includes all the computing steps of an assembly with Ray.

on nearly every machine for every user for problems related to *de novo* genome assembly.

For data analysis and validation, the time cost of having to download the results before looking at them can be prohibitive. We developed Ray Cloud Browser (http://genome.ulaval.ca:10090/client/ and https://github.com/sebhtml/Ray-Cloud-Browser), a streaming solution that allows the user to watch the reconstructed genome as an actual graph from any web browser. Ray Cloud Browser is shown in Figure 3. This web-based streaming visualizer called Ray Cloud Browser was developed as a companion to the assembler to allow users to walk through assembly paths themselves, just as the algorithm does. Not only is the concept of the assembly problem more clearly expressed when removed from its data-manipulation trappings, the interface allows analysts to publish their results on a server for end users to consume without having to download giant files. Beyond validation, analyses can be performed in a colored version of the graph. Such graphs can be generated by Ray,[39] ABySS,[34] or by Cortex in the case of multiple samples.[83] This kind of openness—between software providers and analysts and between analysts and end-users—leads to faster results and will generate faster decisions based on big data.

The Road Ahead

Improving our Ray genome assembler code base is part of the roadmap. Using Ray Cloud Browser, we are able to visually investigate particular cases in the data where our algorithms can be improved.

Future work for our framework—RayPlatform—includes load balancing of workers so that processing tasks are migratable between MPI ranks. Previous research articles from the CHARM++ project will assist us in that regard.[57,58]

Conclusions

When dealing with large data and complex systems, users really want three attributes in particular: scalability, usability, and correctness. To have it all, we have designed RayPlatform, a portable parallel framework on which the genome assembler Ray[35,39] is based. Other genomic applications can use this framework, such as our metagenome assembler and profiler.[39] Our approach does not aim at identifying big data and leveraging bigger systems; rather, it deals with many systems for many data. An interlocked graph is split into tractable chunks of data, and a network of computers into communicable units. With fine granularity, ranks work and communicate as little as possible in the shortest possible amount of time. This allows for scaling both problem and resources from small data, to big data, to bigger data, and beyond, without a paradigm change, allowing the details of the approach, and not its size, to truly fit the problem.

We have presented the larger design principles at work in RayPlatform and the reasoning behind them: getting the most efficiency out of both machine and user, keeping usability and portability at the forefront. As data gets bigger

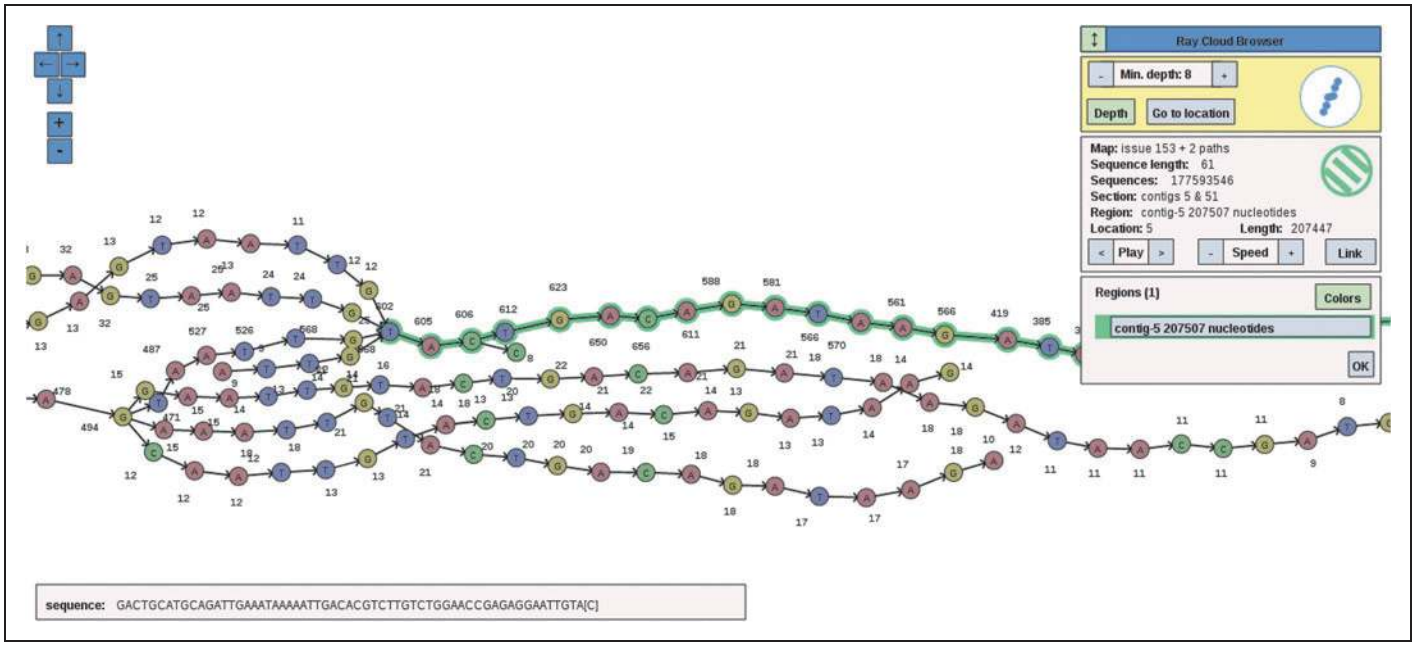

FIG. 3. A screenshot of Ray Cloud Browser. The green path is a sequence assembled by Ray from reads. Numbers near vertices are sequencing depths. Vertices with low sequencing depths are sequencing errors.

and analysts scarcer, it's not only computing time that matters.

Acknowledgments

We are grateful to Rick L. Stevens and Frédéric Raymond for helpful comments. S.B. is recipient of a Frederick Banting and Charles Best Canada Graduate Scholarship Doctoral Award (200910GSD-226209-172830) from the Canadian Institutes for Health Research (CIHR). J.C. is the Canada Research Chair in Medical Genomics. M.K. was supported by Cray, Inc. This research was supported in part by the Fonds de recherche du Québec—Nature et technologies (grant 2013-PR-166708 to F.L. and J.C.) and by the Discovery Grants Program (Individual, Team and Subatomic Physics Project) from the Natural Sciences and Engineering Research Council of Canada (grant 262067 to FL). Computing time on the Colosse supercomputer at Université Laval was provided by Calcul Québec / Compute Canada (projects nne-790-ab & nne-790-ac). Computations were also carried out on the Mammouth-parallèle II (mp2) supercomputer at Université de Sherbrooke (Calcul Québec / Calcul Canada). The operations of Colosse are funded by the Canada Foundation for Innovation (CFI), the National Science and Engineering Research Council (NSERC), NanoQuébec, and the Fonds Québécois de Recherche sur la Nature et les Technologies (FQRNT).

Author Disclosure Statement

The authors declare no competing financial interests exist.

References

1. Marx V. Biology: The big challenges of big data. Nature 2013; 498:255–260.
2. Schatz MC, Langmead B, and Salzberg SL. Cloud computing and the DNA data race. Nature Biotechnology 2010; 28:691–693.
3. Schadt EE, Linderman MD, Sorenson J, et al. Computational solutions to large-scale data management and analysis. Nature Reviews Genetics 2010; 11:647–657.
4. Stein LD. The case for cloud computing in genome informatics. Genome Biology 2010;11:207+.
5. Dudley, JT Butte AJ. *In silico* research in the era of cloud computing. Nature Biotechnology 2010; 28:1181–1185.
6. Stein L. Creating a bioinformatics nation. Nature 2002; 417:119–120.
7. Henry CS, DeJongh M, Best AA, et al. High-throughput generation, optimization and analysis of genome-scale metabolic models. Nat Biotech 2010; 28:977–982.
8. Bentley DR, Balasubramanian S, Swerdlow HP, et al. Accurate whole human genome sequencing using reversible terminator chemistry. Nature 2008; 456:53–59.
9. Gathering clouds and a sequencing storm. Nature Biotechnology, 2010; 28:1.
10. The Human Microbiome Project Consortium. Structure, function and diversity of the healthy human microbiome. Nature 2012; 486:207–214.
11. Smillie CS, Smith MB, Friedman J, et al. Ecology drives a global network of gene exchange connecting the human microbiome. Nature 2011; 480:241–244.

12. Rothberg JM. Leamon JH. The development and impact of 454 sequencing. Nat Biotechnol 2008; 26:1117–1124.
13. Pybus OG, Rambaut A. Evolutionary analysis of the dynamics of viral infectious disease. Nat Rev Genet 2009; 10:540–550.
14. Nakamura S, Maeda N, Miron IMM et al. Metagenomic diagnosis of bacterial infections. Emerging Infectious Diseases 2008; 14:1784–1786.
15. Mardis ER. New strategies and emerging technologies for massively parallel sequencing: applications in medical research. Genome Medicine 2009; 1(4).
16. Caporaso JG, Kuczynski J, Stombaugh J, QIIME allows analysis of high-throughput community sequencing data. Nature Methods 2010; 7:335–336.
17. Lozupone C, Knight R. UniFrac: a new phylogenetic method for comparing microbial communities. Applied and Environmental Microbiology 2005; 71:8228–8235.
18. Lozupone C Hamady M, and Knight R. UniFrac–an online tool for comparing microbial community diversity in a phylogenetic context. BMC Bioinformatics 2006; 7:371+.
19. Lozupone CA. Knight R. Global patterns in bacterial diversity. Proceedings of the National Academy of Sciences 2007;104:11436–11440.
20. Costello EK, Lauber CL, Hamady M et al., Bacterial community variation in human body habitats across space and time. Science 2009; 326:1694–1697.
21. Ley RE, Lozupone CA, Hamady M, et al. Worlds within worlds: evolution of the vertebrate gut microbiota. Nature Reviews Microbiology 2008; 6:776–788.
22. Wu D, Hugenholtz P, Mavromatis K, et al. A. Eisen. A phylogeny-driven genomic encyclopaedia of Bacteria and Archaea. Nature 2009; 462:1056–1060.
23. McPherson JD. Next-generation gap. Nature Methods 2009; 6:S2–S5.
24. Mardis E. The $1,000 genome, the $100,000 analysis? Genome Medicine 2010; 2:84+.
25. Pevzner PA. 1-Tuple DNA sequencing: computer analysis. J Biomol Struct Dyn 1989; 7:63–73.
26. Idury RM, Waterman MS. A new algorithm for DNA sequence assembly. Journal of Computational Biology 1995; 2:291–306.
27. Myers EW, Sutton GG, Smith HO, Adams, and Venter JC. On the sequencing and assembly of the human genome. Proc Natl Acad Sci U S A 2002; 99:4145–4146.
28. Pevzner PA, Tang H, and Waterman MS. An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences 2001; 98:9748–9753.
29. Miller JR, Koren S, and Sutton G. Assembly algorithms for next-generation sequencing data. Genomics 2010; 95:315–327.
30. Pop M. Genome assembly reborn: recent computational challenges. Brief Bioinform 2009; 10:354–366.
31. Myers EW. The fragment assembly string graph. Bioinformatics 2005; 21:ii79–ii85.
32. Loken C, Gruner D, Groer L, et al. SciNet: Lessons learned from building a power-efficient top-20 system and data centre. Journal of Physics: Conference Series 2010; 256:012026+.
33. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.0, 2012.
34. Simpson JT, Wong K, Jackman SD, et al. ABySS: A parallel assembler for short read sequence data. Genome Research 2009; 19:1117–1123.
35. Boisvert S, Laviolette F, Corbeil J. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies. Journal of Computational Biology 2010; 17:1519–1533.
36. Jackson BG, Schnable PS, and Aluru S. Parallel short sequence assembly of transcriptomes. BMC Bioinformatics 2009; 10 Suppl 1:S14+.
37. Jackman S, Birol I. Assembling genomes using short-read sequencing technology. Genome Biology 2010;11: 202+.
38. Jackson BG, Schnable PS, and Aluru S. Assembly of large genomes from paired short reads. In Rajasekaran S, ed, Bioinformatics and Computational Biology, Vol. 5462, 30–43. Springer Berlin Heidelberg 2009.
39. Boisvert S, Raymond F, Godzaridis E, et al. Ray Meta: scalable de novo metagenome assembly and profiling. Genome Biology 2012; 13:R122+.
40. Birol I, Raymond A, Jackman SD, et al. Assembling the 20 Gb white spruce (Picea glauca) genome from whole-genome shotgun sequencing data. Bioinformatics 2013; 29:1492–1497.
41. Hutchinson G. Evaluation of polymer sequence fragment data using graph theory. Bulletin of Mathematical Biophysics 1969; 31:541–562.
42. Gallant JK. The complexity of the overlap method for sequencing biopolymers. J Theor Biol 1983; 101:1–17.
43. Qiu X, Ekanayake J, Beason S, et al. Cloud technologies for bioinformatics applications. In Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers. New York: ACM 2009.
44. Krampis K, Booth T, Chapman B, et al. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. BMC Bioinformatics 2012;13:42+.
45. Bateman A, Wood M. Cloud computing. Bioinformatics 2009;25:1475.
46. Flicek D, Birney E. Sense from sequence reads: methods for alignment and assembly. Nature Methods 2009; 6:S6–S12.
47. Pham CD. Comparison of message aggregation strategies for parallel simulations on a high performance cluster. In Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems 2000;358–365.

48. Scott SL, et al. The Cray T3E Network: Adaptive routing in a high performance 3D torus. In Proceedings of Hot Interconnects IV Symposium (15–16 August 1996), pp. 147–156.
49. Chen D, Eisley NA, Heidelberger P, et al. The IBM Blue Gene/Q interconnection network and message unit. In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. New York: ACM 2011.
50. Faanes G, Bataineh A, Roweth D, et al. Cray cascade: a scalable HPC system based on a Dragonfly network. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Los Alamitos, CA: IEEE Computer Society Press 2012.
51. Arimilli B, Arimilli R, Chung V, et al. The PERCS High-Performance Interconnect. In Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects, 75–82. Washington, D.C., IEEE Computer Society.
52. Ben-Ameur W, Kerivin H. Routing of uncertain traffic demands. Optimization and Engineering 2005; 6:283–313.
53. Ben-Ameur W. Between fully dynamic routing and robust stable routing. In Design and Reliable Communication Networks, 2007, 1–6. Washington, D.C., IEEE.
54. Ben-Ameur W, Żotkiewicz M. Robust routing and optimal partitioning of a traffic demand polytope. International Transactions in Operational Research 2011 18:307–333.
55. Knuth DE. Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition). Boston, MA, Addison-Wesley Professional 1998.
56. Karp RM. Reducibility among combinatorial problems. In Jünger M, Liebling TM, Naddef D, et al., eds. 50 Years of Integer Programming 1958–2008, 219–241. Springer Berlin Heidelberg 2010.
57. Kale LV, Krishnan S. CHARM++: a portable concurrent object oriented system based on C++. In Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications, 91–108. New York: ACM 1993.
58. Kale LV and Zheng G. Charm++ and AMPI: Adaptive runtime strategies via migratable objects. In *Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applications.* 2009, pp. 265–282.
59. Goecks J, Nekrutenko A, Taylor J, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biology 2010; 11:R86+.
60. Baker M. Next-generation sequencing: adjusting to data overload. Nature Methods 2010; 7:495–499.
61. Pollack A. DNA sequencing caught in deluge of data. New York Times, 2011, P.1.
62. Marx V. Genomics in the clouds. Nature Methods 2013; 10:941–945.
63. Stockton J. BaseSpace Roadmap. In Plant and Animal Genome XXI Conference. Plant and Animal Genome 2013.
64. Reich M, Liefeld J, Thorvaldsdottir H, et al. GenomeSpace: An environment for frictionless bioinformatics. In Proceedings of the 103rd Annual Meeting of the American Association for Cancer Research, Vol 72, 3966+. American Association for Cancer Research 2012.
65. Cattell R. Scalable SQL and NoSQL data stores. SIGMOD Rec 2011; 39:12–27.
66. DeCandia G, Hastorun D, Jampani M, Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev. 2007; 41:205–220.
67. Calder B, Wang J, Ogus A, et al. Windows Azure Storage: a highly available cloud storage service with strong consistency. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, 143–157, New York: ACM 2011.
68. Lawder JK, King PJH. Querying multi-dimensional data indexed using the Hilbert space-filling curve. SIGMOD Rec. 2001; 30:19–24.
69. Cock PJA, Fields CJ, Goto N, et al. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Research 2010; 38:1767–1771.
70. Margulies M, Egholm M, Altman WE, et al. Genome sequencing in microfabricated high-density picolitre reactors. Nature 2005; 437:376–380.
71. Rumble SM, Lacroute P, Dalca AV, et al. SHRiMP: Accurate mapping of short color-space reads. PLoS Comput Biol 2009; 5:e1000386+.
72. Li H, Handsaker B, Wysoker A, et al. The Sequence Alignment/Map format and SAMtools. Bioinformatics 2009; 25:2078–2079.
73. Barnett DW, Garrison EK, Quinlan AR, et al. BamTools: a C++ API and toolkit for analyzing and managing BAM files. Bioinformatics 2011; 27:1691–1692.
74. Kent WJ. BLAT–the BLAST-like alignment tool. Genome Res 2002; 12:656–664.
75. Danecek P, Auton A, Abecasis G, et al. The variant call format and VCFtools. Bioinformatics 2011; 27:btr330–2158.
76. Reese M, Moore B, Batchelor C, et al. A standard variation file format for human genome sequences. Genome Biology 11:R88+.
77. Rutherford K, Parkhill J, Crook J, et al. Artemis: sequence visualization and annotation. Bioinformatics 2000; 16:944–945.
78. Quinlan AR, Hall IM. BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics 2010; 26:841–842.
79. Leinonen R, Sugawara H, and Shumway M. The Sequence Read Archive. Nucleic Acids Research 2011;39: D19–D21.
80. Chikhi, R Medvedev P. Informed and automated k-mer size selection for genome assembly. Bioinformatics, 2013; btt310+.

81. Compeau PEC, Pevzner PA, and Tesler G. How to apply de Bruijn graphs to genome assembly. Nature Biotechnology 2011; 29:987–991.
82. Bradnam KR, Fass JN, Alexandrov A, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. Giga Science 2013; 2:10.
83. Iqbal Z, Caccamo M, Turner I, et al. De novo assembly and genotyping of variants using colored de Bruijn graphs. Nature Genetics 2012; 44 226–232.

Address correspondence to:

Sébastien Boisvert
Infectiologie et Immunologie
Centre de Recherche du CHUQ
2705, boulevard Laurier, R-5711
Québec, Québec G1V 4G2
Canada

E-mail: sebastien.boisvert.3@ulaval.ca