Granularities and messages: from design to abstraction to implementation to virtualization

Length: 1 hour

Élénie Godzaridis
Strategic Technology Projects
Bentley Systems, Inc.

Sébastien Boisvert
PhD student, Laval University
CIHR doctoral scholar

Meta-data

- Invited by Daniel Gruner (SciNet, Compute Canada)
- https://support.scinet.utoronto.ca/courses/?q=node/95
- **Start: 2012-11-26 14:00 End: 2012-11-26 16:00**
- **Seminar by Élénie Godzaridis, Sébastien Boisvert , developers of the parallel genome assembler "Ray".**
- **Location: SciNet offices at 256 McCaul Street, Toronto, 2nd Floor.**

Introductions

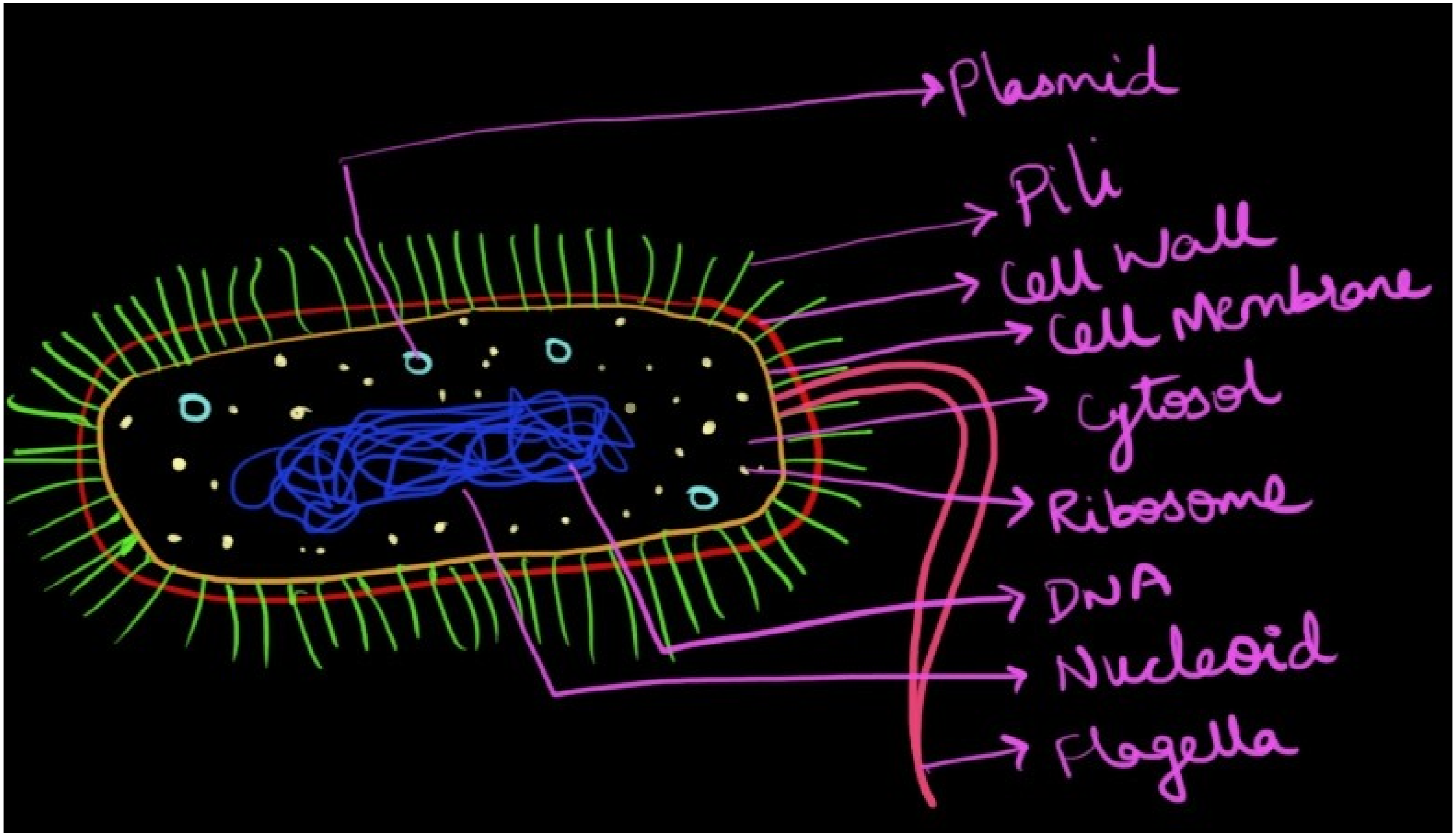
- Who are we ?
- **Sébastien:** message passing, software development, biological systems, repeats in genomes, usability, scalability, correctness, open innovation, Linux
- **Élénie:** software engineering, blueprints, designs, books, biochemistry, life, rendering engines, geometry, web technologies, cloud, complex systems

Approximative contents

- Message passing
- Granularity
- Importance of having a framework
- How to achieve useful modularity at running time / compile time ?
- Important design patterns
- Distributed storage engines with MyHashTable
- Handle types: slave mode, master mode, message tag
- Handlers
- RayPlatform modular plugin architecture
- Pure MPI apps are not good enough, need threads too
- Mini-ranks
- Buffer management in RayPlatform
- Non-blocking shared message queue in RayPlatform

- Problem definition

Why bother with DNA ?



de novo genome assembly

Why is it hard to parallelize ?

- Each piece is important for the big picture
- Not embarrassingly parallel
- Approach: have an army of actors working together by sending messages
- Each actor owns a subset of the pieces

de Bruijn graphs in bioinformatics

- Alphabet: {A,T,C,G}, word length: k
- Vertices V = {A,T,C,G}^k
- Edges are a subset of V x V
- (u,v) is an edge if the last k-1 symbols of u are the first k-1 symbols of v
- Exemple: A**TCGA** -> **TCGA**T
- In genomics, we use a de Bruijn subgraph using k-mers for vertices and (k+1)-mers for edges
- k-mers and (k+1)-mers are sampled from data
- Idury & Waterman 1995 Journal of Computational Biology

Why is assembly hard ?

- Arrival rate of reads is not perfect
- DNA sequencing theory
- Lander & Waterman (1988) Genomics 2 (3): 231–239.

Professor E. Lander
(Photo: Wikipedia)

Professor M. Waterman
(Photo: Wikipedia)

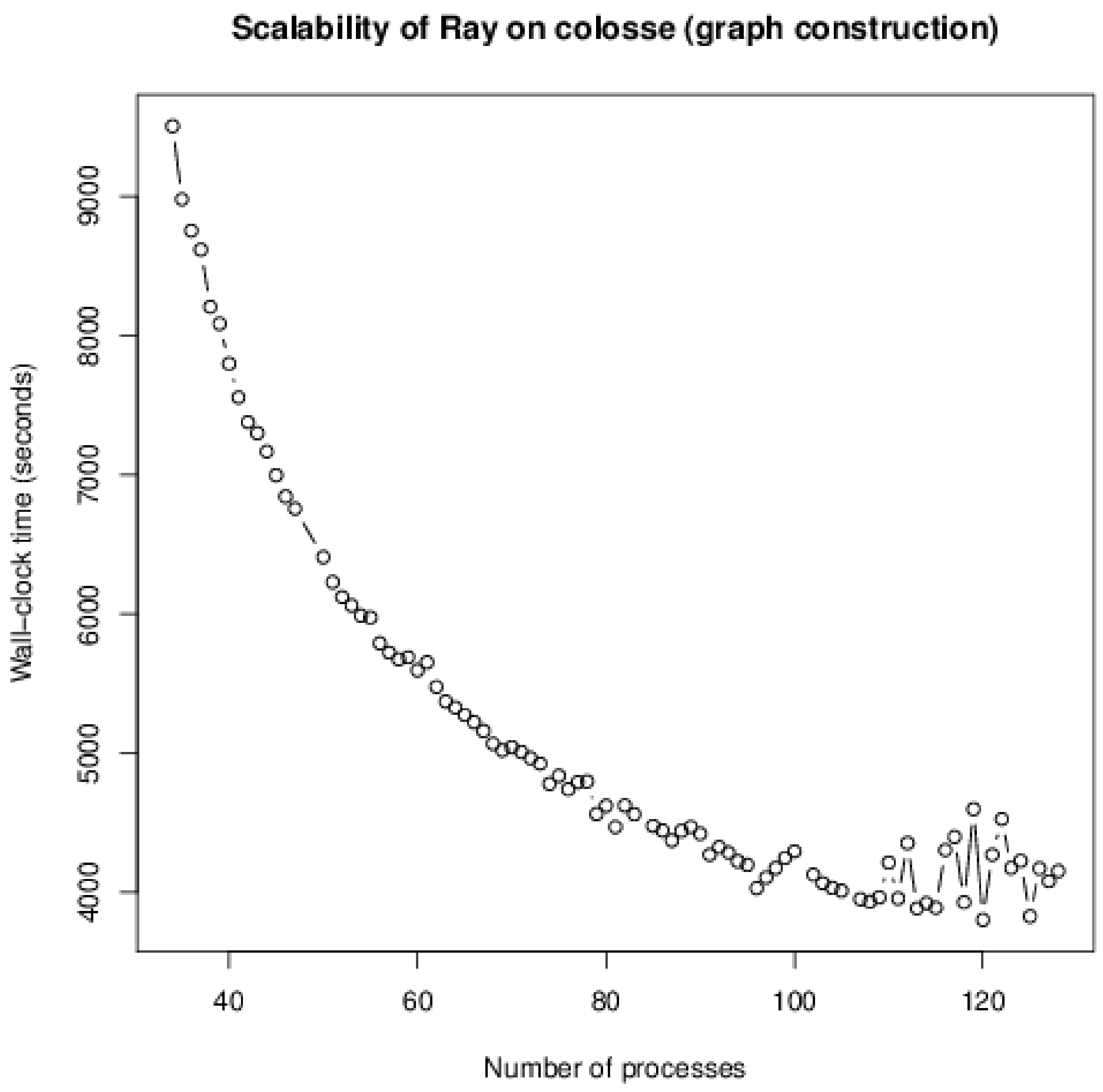
- Granular run-time profiles on Blue Gene/Q

Latency matters

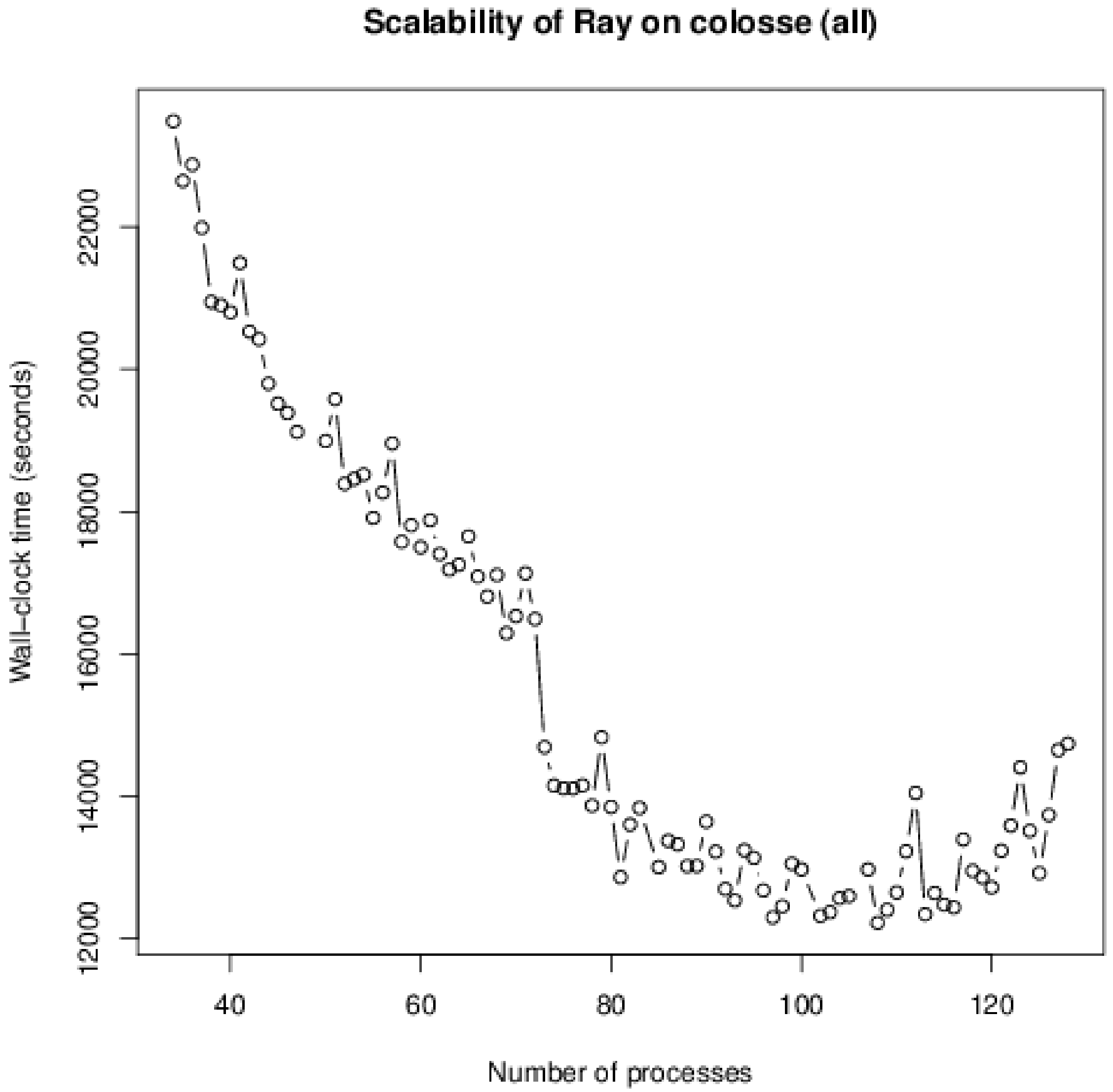
- To build the graph for the dataset SRA000271 (human genome, 4 * 10^9 reads), with 512 processes
  - 159 min when average latency is 65 us (Colosse)
  - 342 min when average latency is 260 us (Mammouth)
- 4096 processing elements, Cray XE6, round-trip latency in application -> 20-30 microseconds (Carlos Sosa, Cray Inc.)

Building the distributed de Bruijn graph

- metagenome
- sample SRS011098
- 202 * 10^6 reads



Overall (SRS011098)



- Message passing

Message passing for the layman

Olga the crab (*Uca pugilator*)
Photo: Sébastien Boisvert, License: Attribution 2.0 Generic (CC BY 2.0)

Message passing with MPI

- MPI 3.0 contains a lot of things
- Point-to-point communication (two-sided)
- RDMA (one-sided communication)
- Collectives
- MPI I/O
- Custom communicators
- Many other features

MPI provides a flat world

Figure 1: The MPI programming model.

```
                +--------------------+
                |   MPI_COMM_WORLD   |
                +---------+----------+          MPI communicator
                          |
     +------+------+------+--+------+------+
     |      |      |         |      |      |
   +---+  +---+  +---+     +---+  +---+  +---+
   | 0 |  | 1 |  | 2 |     | 3 |  | 4 |  | 5 |
   +---+  +---+  +---+     +---+  +---+  +---+      MPI ranks
```

Point-to-point versus collectives

- With point-to-point, the dialogue is local between two folks
- Collectives are like meetings – not productive when too many of them
- Collectives are not scalable
- Point-to-point is scalable

- Granularity

Granularity

- Standard sum from 1 to 1000
- Granular version: sum 1 to 10 on the first call, 11 to 20 on the second, and so on
- Many calls are required to complete

- From programming models to frameworks

Parallel programming models

- 1 process with many kernel threads on 1 machine
- Many processes with IPC (interprocess communication)
- Many processes with MPI (message passing interface)

MPI is low level

- Message passing does not structure a program
- Needs a framework
- Should be modular
- Should be easy to extend
- Should be easy to learn and understand

- How to achieve useful modularity at running time / compile time ?

Model #1 for message passing

- 2 kernel threads per process (1 for busy waiting for communication and 1 for processing)
- Cons:
  - not lock-free
  - prone to programming errors
  - Half of the cores busy wait (unless they sleep)

Model #2 for message passing

- 1 single kernel thread per process
- Comm. and processing interleaved
- Con:
  - Needs granular code everywhere !
- Pros
  - Efficient
  - Lock-free (less bugs)

Models for task splitting

- **Model 1: separated duties**
- Some processes are data stores (80%)
- Some processes are algorithm runners (20%)
- Con:
  - Data store processes do nothing when nobody speak to them
  - Possibly unbalanced

Models for task splitting

- **Model 2: everybody is the same**
- Every process has the same job to do
- But with different data
- One of the processes is also a manager (usually # 0)
- Pros
  - Balanced
  - All the cores work equally

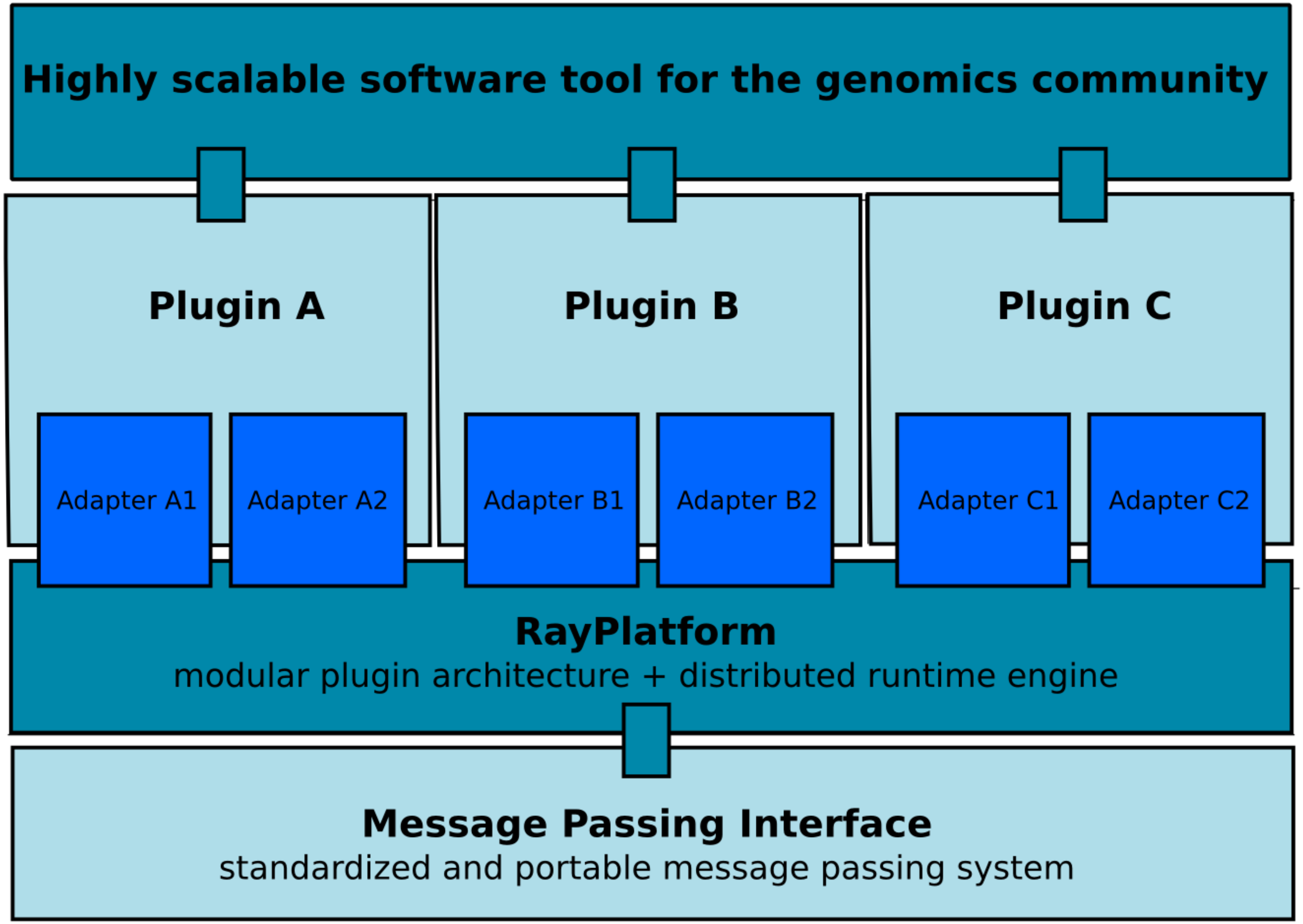
Memory models

- 1. Standard: 1 local virtual address space per process
- 2. Global arrays (distributed address space)
  - Pointer dereference can generate a payload on the network
- 3. Data ownership
  - Message passing
  - DHTs (distributed hash tables)
  - DHTs are nice because the distribution is uniform

- RayPlatform modular plugin architecture

RayPlatform

- Each process has: inbox, outbox
- Only point-to-point
- Modular plugin architecture
- Each process is a state machine
- The core allocates:
  - Message tag handles
  - Slave mode handles
  - Master mode handles
- Associate behaviour to these handles
- GNU Lesser General Public License, version 3
- https://github.com/sebhtml/RayPlatform


Highly scalable software tool for the genomics community
Plugin A
Plugin B
Plugin C
Adapter A1
Adapter A2
Adapter B1
Adapter B2
Adapter C1
Adapter C2
RayPlatform
modular plugin architecture + distributed runtime engine
Message Passing Interface
standardized and portable message passing system

Quick facts:

◦ RayPlatform assists developers during the content creation
◦ A software tool for the genomics community is implemented as plugins
◦ Plugins are executed by a distributed runtime engine
◦ Communication is portable thanks to MPI

- Important design patterns

- State
- Strategy
- Adapter
- Facade

- Handlers

Definitions

- Handle: opaque label
- Handler: behaviour associated to an event
- Plugin: orthogonal module of the software
- Adapter: binds two things that can not know each other
- Core: the kernel
- Handler table: tells which handler to use with any handle
- Handler table is like interruption table

- Handle types: slave mode, master mode, message tag

State machine

- A machine with states
- Behaviour guided by its states
- Each process is a state machine

Main loop

- while(isAlive()){

```
    receiveMessages();
    processMessages();

    processData();
    sendMessages();

}
```

Virtual processor (VP)

- Problem: kernel threads have a overhead, but
- Solution: thread pools retain the benefits of fast task-switching
  - each process has many user space threads (workers) that push messages
- The operating system is not aware of workers (user space threads)

Virtual communicator (VC)

- Problem: sending many small messages is costly
- Solution: aggregate them transparently
- Workers push messages on the VC
- The VC pushes bigger messages in the outbox
- Workers are user space threads
- States: Runnable, Waiting, Completed

Regular complete graph and routes

Complete graph for MPI communication is a bad idea !

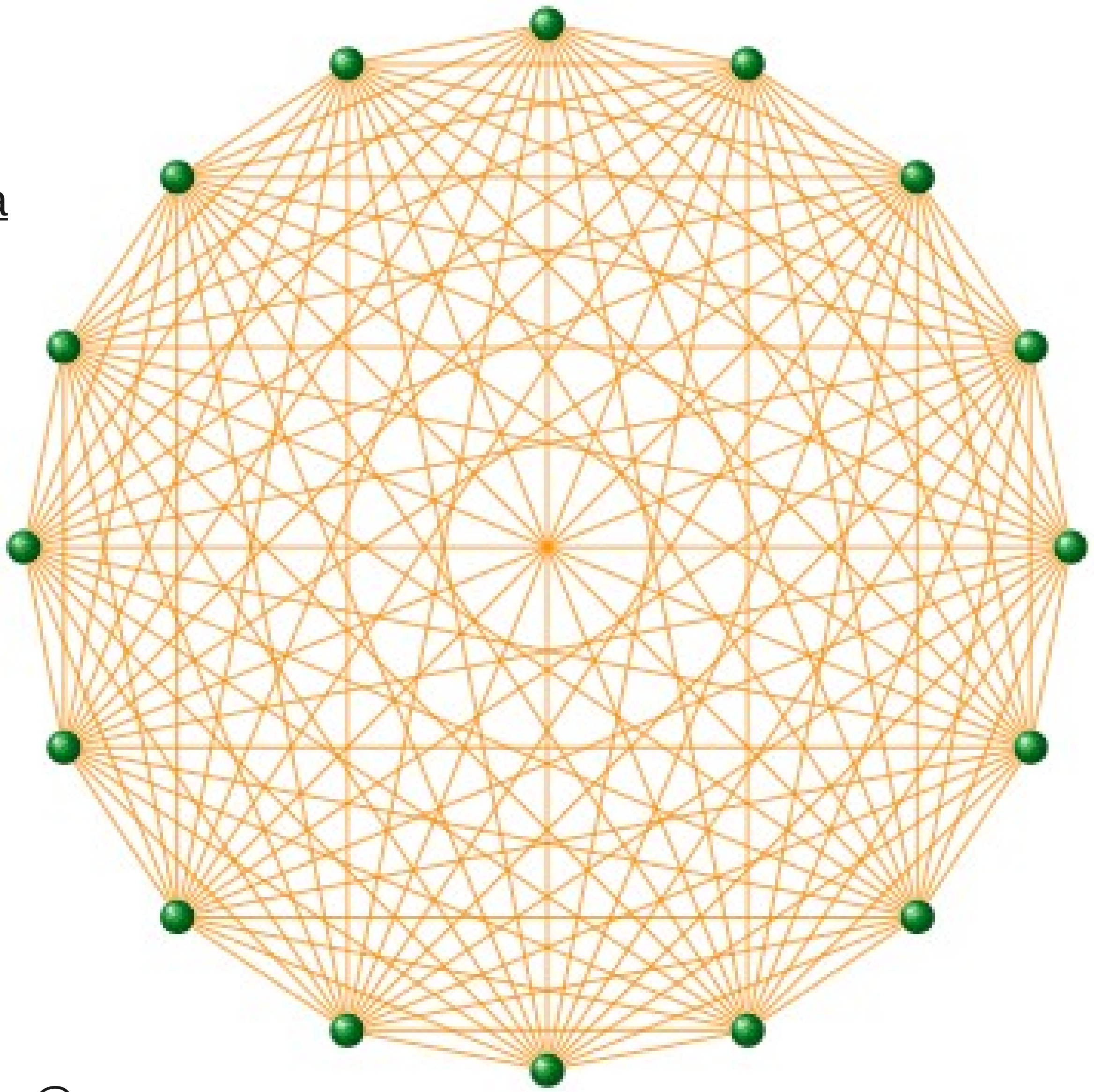


Image by: Alain Matthes — al.ma@mac.com

Virtual message router

- Problem: any-to-any communication pattern can be bad
- Solution: fit the pattern on a better graph
- 5184 processes -> 26873856 comm. edges ! (diameter: 1)
- With surface of regular convex polytope: 5184 vertices, 736128 edges, degree: 142, diameter: 2

Profiling is understanding

- RayPlatform has its own real-time profiler
- Reports messages sent/received, current slave mode at every 100 ms quantum

Example

- Rank 0: RAY_SLAVE_MODE_ADD_VERTICES Time= 4.38 s Speed= 74882 Sent= 51 (**processMessages**: 28, **processData**: 23) Received= 52 Balance= -1

  Rank 0 received in **receiveMessages**:

  Rank 0   RAY_MPI_TAG_VERTICES_DATA 28

  Rank 0   RAY_MPI_TAG_VERTICES_DATA_REPLY   24

  Rank 0 sent in **processMessages**:

  Rank 0   RAY_MPI_TAG_VERTICES_DATA_REPLY   28

  Rank 0 sent in **processData**:

  Rank 0   RAY_MPI_TAG_VERTICES_DATA 23

- Pure MPI apps are not good enough, need threads too

Routing with regular polytopes

- Polytopes are still bad
- all MPI processes on a machine talk to the Host Communication Adapter
- Threads ?

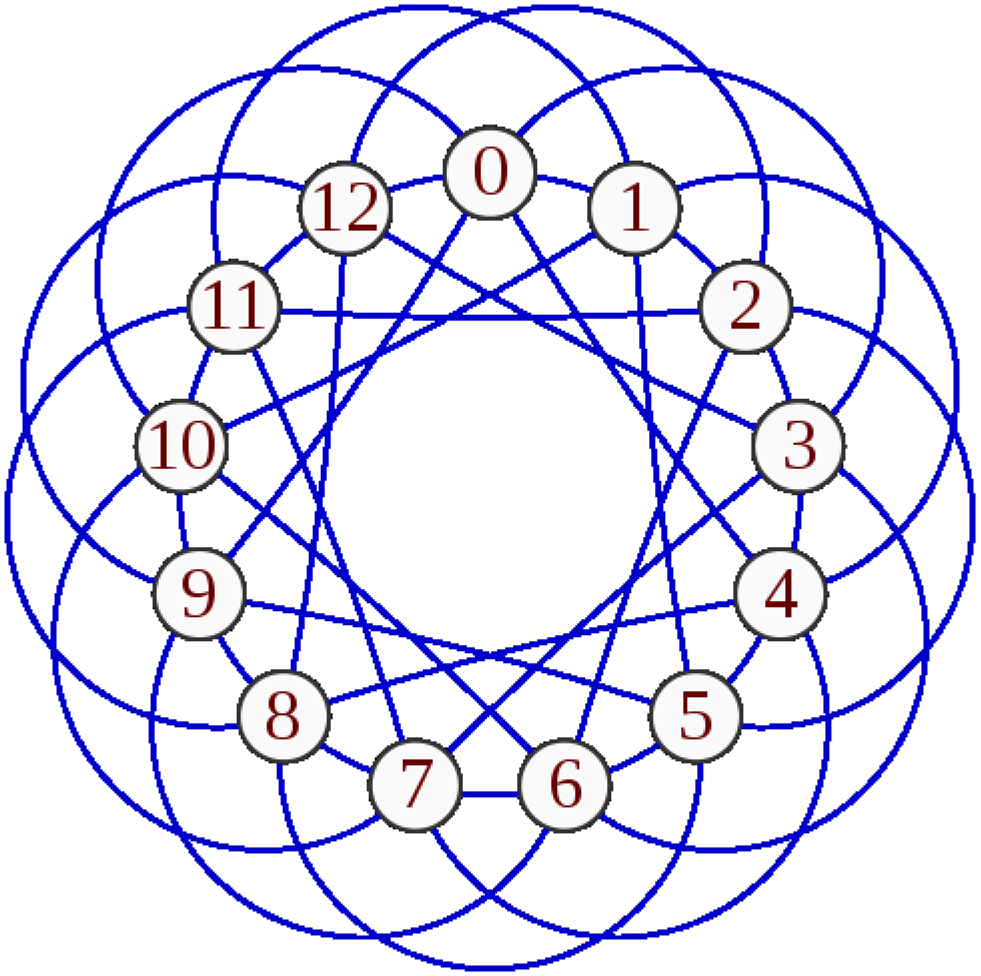


Image: Wikipedia

- Mini-ranks

Roadblocks with MPI processes

- The IBM PowerPC A2 *may be**** better at scheduling 16 processes with 4 threads each than scheduling 64 processes

*** Hypothesis

Hierarchical message distribution systems

Mini-ranks hybrid programming model

Figure 2: The MPI programming model, with mini ranks.

```
            +--------------------+
            |   MPI_COMM_WORLD   |           MPI communicator
            +---------+----------+
                      |
    +------+------+---+--+------+------+
    |      |      |      |      |      |
  +---+  +---+  +---+  +---+  +---+  +---+
  | 0 |  | 1 |  | 2 |  | 3 |  | 4 |  | 5 |  MPI ranks (1 RankProcess.cpp instance per rank)
  +---+  +---+  +---+  +---+  +---+  +---+     with the thread from main() for  MPI calls
  |   |  |   |  |   |  |   |  |   |  |   |
  | 0 |  | 4 |  | 8 |  |12 |  |16 |  |20 |  |
  | 1 |  | 5 |  | 9 |  |13 |  |17 |  |21 |  | => mini-ranks, in pthreads
  | 2 |  | 6 |  |10 |  |14 |  |18 |  |22 |  |
  | 3 |  | 7 |  |11 |  |15 |  |19 |  |23 |  |  (1 MiniRank.cpp instance per minirank,
  |   |  |   |  |   |  |   |  |   |  |   |  |   1 Application running with its RayPlatform
                                                                      ComputeCore.cpp)
```

This hybrid model was devised by Sébastien Boisvert, Fangfang Xia and Rick Stevens.
It is implemented in RayPlatform and a manuscript is in preparation

-mini-ranks-per-rank

- In pure MPI mode:
  ```
  mpiexec -n 2400 \
  MyApplication ...
  ```
  => 2400 MPI processes

- In mini-ranks mode:
  ```
  mpiexec -n 100 -bynode \
  MyApplication -mini-ranks-per-rank 23 ...
  ```
  => 100 MPI processes, 23 threads per MPI process for mini-ranks, the control thread of main() does MPI calls

- RayPlatform runtime engine will pick up “-mini-ranks-per-rank” and do its magic

- Buffer management in RayPlatform

•Amortized buffer management

- Needs to know when space in the ring buffer given to MPI_Isend can be reused
- Amortized management of dirty buffers
- Buffers are either BUFFER_STATE_DIRTY or BUFFER_STATE_AVAILABLE
- You just don't know how many buffers you need before running a job

- Non-blocking shared message queue in RayPlatform

Best way to synchronize mini-rank threads

- The best way is to do nothing at all !
- Non-blocking circular message queue
- Allows 1 consumer and 1 producer simultaneously
- Algorithms and concepts described by Kjell Hedström

  http://www.codeproject.com/Articles/43510/Lock-Free-Sing
- Source code for MessageQueue written from scratch in RayPlatform (license: LGPL3)

- Distributed storage engines

Hash tables in RayPlatform

- Custom code: MyHashTable.h, MyHashTableGroup.h
- C++ template
- Sparse (Knuth model, 64 buckets / group)
- Distributed (DHT)
- Open addressing (double hashing)
- Double hashing has no clustering
- But is bad with CPU cache
- Incremental resizing

Distributed storage engine

- Reads are distributed uniformily
- K-mers are distributed uniformily
- Only 1 of any 2 reverse complement k-mers stored
- Annotations on objects (be it reads or k-mers)
- Virtual coloring of k-mers
- Compact edge representation (Simpson et al. 2009 Genome Research)

Sequencing errors

- Bloom filter, 2 operations: hasItem?, insertItem!
- No false negatives, few false positives
- In bioinformatics (Pell et al. PNAS 2012)
- Each Ray process has a Bloom filter
- Weeds out most of the k-mers occurring once

Data structures

- “Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”
  -- Linus Torvalds

https://plus.google.com/u/0/+LinusTorvalds/posts

Some results with Ray Meta

- All these results are on Colosse
- Round-trip in-application point-to-point latency > 100 microseconds for 512-process jobs
- 3 000 000 000 reads from a 1000-bacterium metagenome, 15 hours on 1024 cores
- 400 000 000 reads from 100-bacterium metagenome, 14 hours, 128 cores
- Includes also k-mer based profiling (genome abundance, taxonomy, gene ontology)

Acknowledgements / Invitation

- Daniel Gruner (invitation and arrangements)
- Ramses van Zon (reviewed slides)

Acknowledgements / Funding

- SB: Doctoral award, **Canadian Institutes for Health Research**, September 2010 – August 2013
- Jacques Corbeil: **Canada Research Chair in Medical Genomics**
- Discovery Grants Program (Individual, Team and Subatomic Physics Project) from the **Natural Sciences and Engineering Research Council of Canada** (grant 262067 to François Laviolette)

CIHR IRSC
Canadian Institutes of Health Research
Instituts de recherche en santé du Canada
Institute of Genetics
L'Institut de génétique

NSERC
CRSNG

Acknowledgements / Product team

- Sébastien Boisvert (designer, developer, release technician, community manager)
- Élénie Godzaridis (parallel designs, works in the industry)
- Prof. François Laviolette (graph specialist)
- Prof. Jacques Corbeil (genomician)
- Maxime Boisvert (design tricks, consultant in the industry)
- Dr. Frédéric Raymond (end user / stakeholder)
- Pier-Luc Plante (intern)

Acknowledgements / CPU time

- 2011: 50 core-years on Colosse
- 2012: 250 core-years on Colosse
- Compute Canada (Colosse, Mammouth Parallèle II, Guillimin)
- Calcul Québec, CLUMEQ, RQCHP
- Canadian Foundation for innovation for the 32-core 128-GB SMP machine
- Collaboration with Cray Inc. for the Cray XE6 (with Carlos Sosa)

Questions and answers