QuIDS

A Large-Scale Distributed Framework for Quantum Irregular Dynamics Simulations

Presenter: Joseph Touzet
Joint work with Pablo Arrighi, Amélia Durbec and Oguz Kaya

PhD student

ENS Paris-Saclay, centre Borelli

IPDPS 2025, QCASA-3 workshop, June 3

Motivations for an irregular Quantum simulator

Initial motivation : Quantum Causal Graph Dynamics (QCGD) :

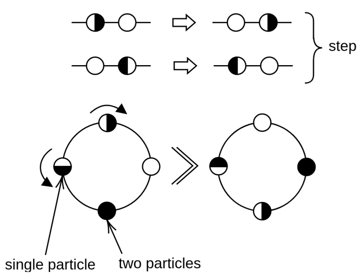
- Python code only for QCGD $\longrightarrow$ too slow and inaccurate, and specific to QCGD
- Infinite dimension state $\longrightarrow$ can't use most off the shelves simulators

Advantages of moving to a distributed, general framework :

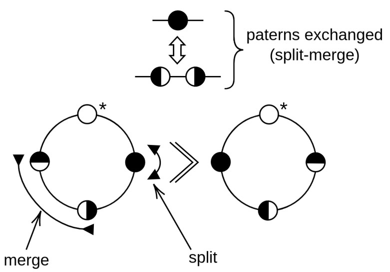
- Faster and more accurate $\longrightarrow$ better science
- Usable for any irregular application $\longrightarrow$ a single efficient framework for many application
- Quantum Turing machine and Loop gravity simulations are existing use-cases
- The ability of simulating irregular system can motivate the creation of new models

Classical Graph Dynamics: Hasslacher-Meyer

- Circular colored graphs
- Can be used as a toy model for the expansion of the universe



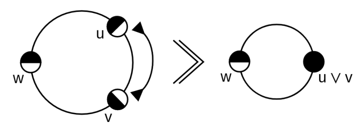
(a) Example of HM *step* evolution



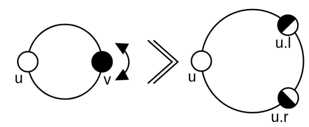
(b) Example of HM *split-merge* evolution

Irregularity in the Hasslacher-Meyer model

- Varying graph size after applying split-merge
- Complex name structure for nodes for reversibility



(a) Example of graph shrinking within the *split-merge* rule - example of node names.



(b) Example of graph growing within the *split-merge* rule - example of node names.

Quantum Causal Graph Dynamics

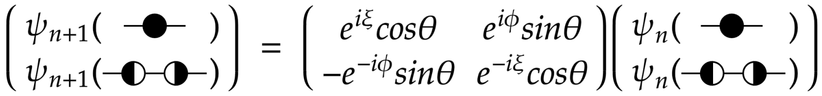


Figure: Example of a quantum rule - *split-merge*

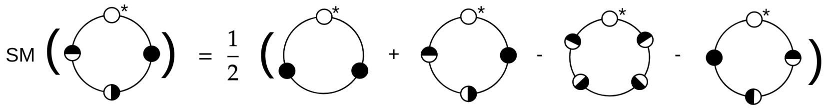


Figure: Example application of a quantum rule - *split-merge* ($\theta = \pi/4$, $\phi = \xi = 0$)

Irregularity of Quantum Causal Graph Dynamics

Memory irregularity

- Big and small objects (1kB - 1MB)
- Objects with irregular sizes

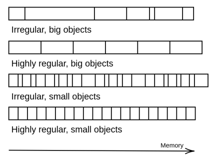


Figure: Examples of contiguous object vectors

Simulation irregularity

- Varying number of "*child*" objects
- Varying number of object collisions

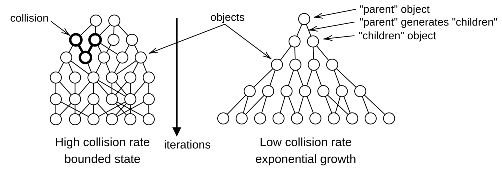


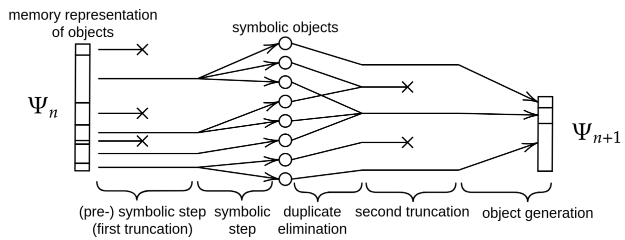
Figure: Two dynamics of different nature

Requirements for a quantum dynamics simulator

- Being as "general purpose" as possible:
  - To simulate *QCGD*, but also *Quantum Turing Machines*, simple *Loop Quantum Gravity* simulations, etc...
  - Allows the development of other irregular quantum models, owing to the capacity to simulate them.
- Performance: Usable accuracy and reasonable execution time:
  - Taking advantage of clusters to gain memory and computational power.
  - Distributing the computation: assigning objects to nodes to compute collisions.
- Exponential growth of the problem size - truncation:
  - Not running out of memory: *under-truncating*.
  - Not letting memory unused: *over-truncating*.
  - Retain accuracy and "representativity".

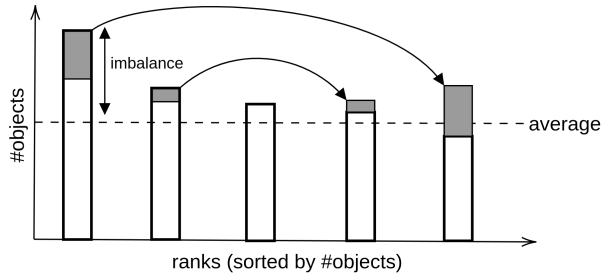
QuIDS framework - General structure

- Objects representing Ψ_n distributed across ranks
- 4 computational steps:
  1. Pre-symbolic step: First truncation and balancing of the number of objects
  2. Symbolic step
  3. Duplicate elimination and second truncation
  4. Final step: generates final object memory representation



QuIDS - Pre-symbolic step

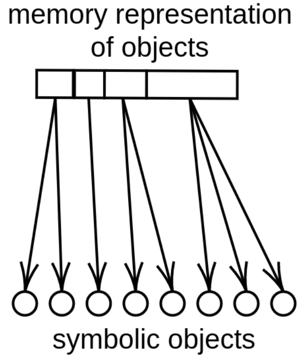
- Load balancing of the number of objects + first truncation
- Iterative balancing of pairs of MPI processes (point to point communication)
- Only step where we communicate objects
- Balancing of the number of children ⇒ better representing the load than the number of objects



QuIDS - Symbolic computation

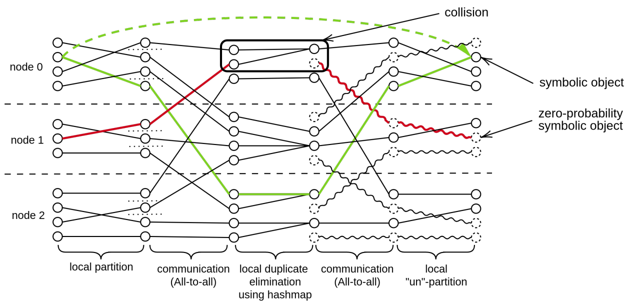
Reasons for a symbolic iteration:

- Memory management accuracy gains:
  - Manipulating fixed-size, smaller intermediary objects
  - Allows for more intermediary objects to be stored ⇒ higher accuracy
  - Knowing the exact size of future objects ahead of their generation
- Simpler distributed duplicate elimination:
  - Sending and receiving small, fixed-size intermediary objects
  - Comparing small objects (hashes)



QuIDS - Elimination of duplicate objects

- Divide objects into local buckets according to their hashes
- Build global exclusive hash sections by merging the buckets
- Each nodes compute collision on a specific exclusive section (collected from all nodes)
- Load balancing through dynamic local object mapping



QuIDS - State truncation

- *predictive* method before the *symbolic step*
- *reactive* method before the *final object generation*
- Types of ranking:
  - Deterministic ranking (N most probable objects selected)
  - Probabilistic ranking (according objects probability): *similar to a Quantum Monte-Carlo-like method*

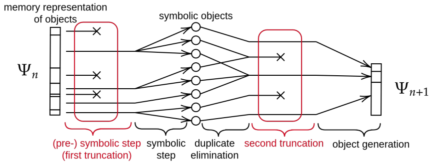
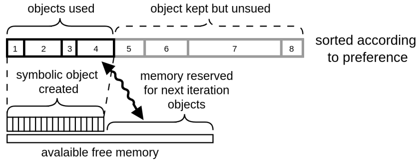
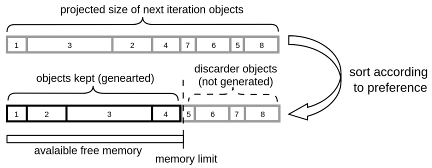


Figure: Major steps of an iteration, showing both truncation



(a) *Predictive* truncation before the *symbolic step*



(b) *Reactive* truncation before *final object generation*

Framework implementation & experimental setup

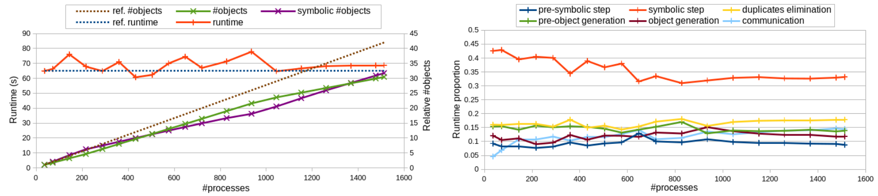
Framework implementation :

- Availability: github.com/jolatechno/QuIDS
- Implemented in C++(17) using MPI

Experimental setup :

- Plafrim Bora cluster
- Up to 43 nodes (1548 cores and 8.256TB of total memory)
- Compiled using g++ 11.3.0, distribution using OpenMPI 4.0.1
- Up to 2 billions objects and 31 billions symbolic objects

Weak scalability

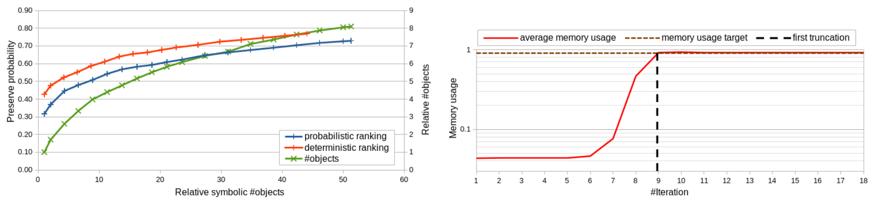


(a) Weak scaling example (low collision rate)

(b) Dissection of the execution time

Accuracy

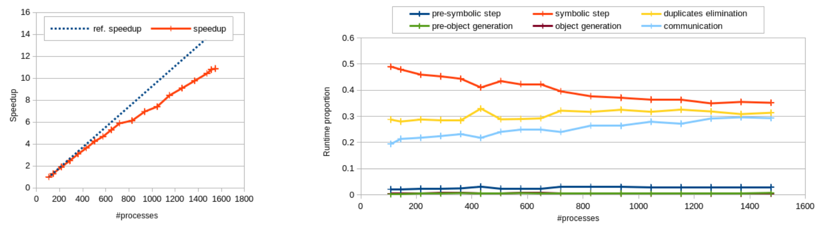
- Differences between ranking algorithms: *probabilistic vs deterministic*
- Preserved probability VS numerical accuracy
- Sustained memory use over 90% without running out of memory



(a) Accuracy scaling example

(b) Memory usage over time

Strong scalability



(a) Strong scaling example

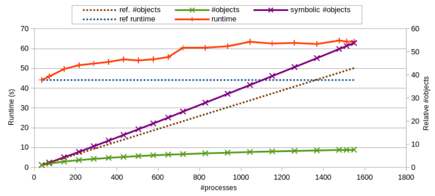
(b) Dissection of the execution time

Conclusion and future work

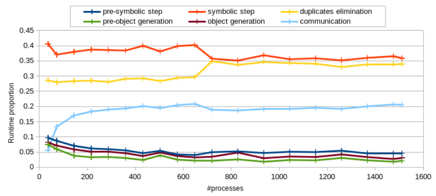
- Simulation of QCGDs for almost any case with usable accuracy
- 70-100% efficient weak-scaling
- Future work:
  - Testing the framework with other irregular problems

Special thanks to the PlaFRIM platform at Inria Bordeaux for providing the compute resources

Support slide 1 - additional performance results

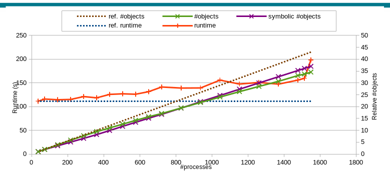


(a) High collision rate weak scaling

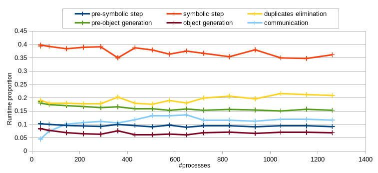


(b) Dissection of the execution time

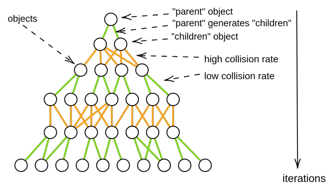
Support slide 2 - alternating dynamics



(a) Alternating dynamic weak scaling



(b) Dissection of the execution time



(c) Schematic of alternating high collision rate and low collision rate dynamics

Support slide 3 - Other QCGDs rules

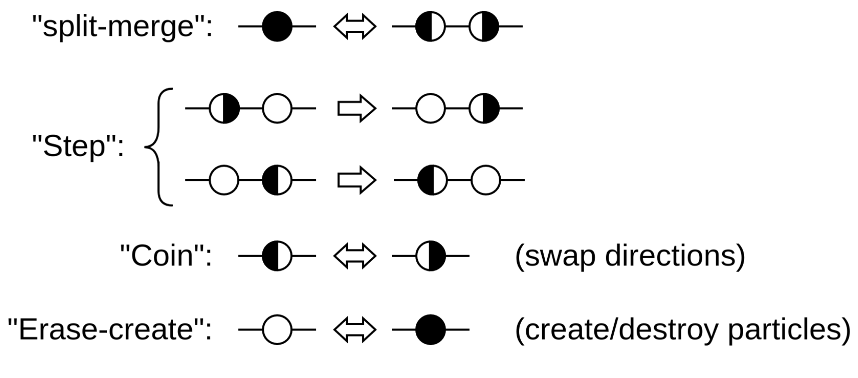


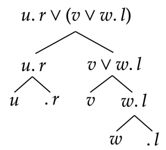
Figure: Other rules used.

Support slide 4 - QCGDs name arithmetic

- Node name represented as tree
- Complex name dynamic - *name arithmetic*

$$\begin{cases} u.l \vee u.r &= u \\ (u \vee v).l &= u \\ (u \vee v).r &= v \end{cases}$$

(a) Rules of the name arithmetic (allowing reversibility)



(b) Example of node name representation (tree)