QUIDS: A Large-Scale Distributed Framework for Quantum Irregular Dynamics Simulations

Joseph Touzet, Oguz Kaya, Pablo Arrighi, Amélia Durbec

QUIDS: A Large-Scale Distributed Framework for Quantum Irregular Dynamics Simulations

Joseph Touzet
ENS Paris-Saclay, Centre Borelli
Gif-sur-Yvette, France
joseph.touzet@ens-paris-saclay.com

Oguz Kaya
Université Paris-Saclay, LISN, CNRS
Gif-sur-Yvette, France
oguz.kaya@universite-paris-saclay.fr

Pablo Arrighi
Inria, Université Paris-Saclay, LMF, CNRS
Gif-sur-Yvette, France
pablo.arrighi@universite-paris-saclay.fr

Amélia Durbec
ICL, Junia, Université Catholique de Lille, LITL
Lille, France
amelia.durbec@protonmail.com

***Abstract*—In traditional quantum computing, e.g. in the quantum circuit model, the size of the data structure describing basis elements is well known, because the dimensionality is fixed. General quantum systems, however, exhibit basis elements of variable size, and state spaces having dynamically unbounded, possibly infinite dimensionality, e.g. for quantum Turing machines or quantum field theories. When seeking to simulate them classically, this imposes an irregularity on both the memory representation of basis elements and the sparsity of the quantum transformations they undergo. Moreover, the high dimensionality of these problems often makes them memory intensive, potentially requiring truncation methods during the simulation. One prototypical example of this would be quantum causal graph dynamics (QCGD), which feature superpositions of colored graphs of different shapes and sizes, driven by the application of local quantum transformations. Numerical observations show that their reversible counterparts typically grow in size; understanding how this is affected in the quantum regime is an arduous computational challenge requiring a particular HPC expertise. In this work, we address this challenge by developing a computational framework for a scalable simulation of such general irregular quantum systems in distributed-memory parallel environments. We lay out the computational challenges arising from the nature of such simulations and then propose effective parallelization, load balancing, memory management, and parallel sampling strategies to accelerate them. We report parallel scalability and accuracy results for up to 1548 MPI processes on a parallel cluster using our framework for the QCGD simulation.**

***Index Terms*—quantum computing, irregular quantum simulation, quantum causal graph dynamics, distributed-memory parallelism.**

I. INTRODUCTION

A. On quantum mechanics

Quantum mechanics governs the evolution of all fundamental particles and their interactions. Simulating quantum mechanics would unleash an array of applications, such as computer-assisted design of molecules, drugs, and materials. Unfortunately, quantum mechanics is arduous to simulate on a classical computer due to the superposition principle. Indeed, the superposition principle has it that if $(|i\rangle)_{i\in S}$ are the possible basic states of a system, then so are all possible norm-one linear combinations $|\psi\rangle = \sum_i \alpha_i |i\rangle$. For instance, consider n particles, where each can be in spin up or down, i.e. $S = \{0,1\}^n$, which makes for 2^n basic states. The classical description of an arbitrary quantum state $|\psi\rangle$ is then a rather lengthy $(\alpha_i)_{i\in S}$ vector of 2^n complex numbers. This exponential blow in memory size is the reason quantum mechanics is extremely difficult to simulate on a classical computer—but it is also the natural source of parallelism that powers the quantum computing advantage. To efficiently simulate quantum mechanics, one should resort to a quantum computer: this was Feynman's insight and his sole motivation to invent quantum computers at all. Nowadays, quantum simulation through classical computing is considered to be a medium-term application of the thriving field of quantum computing. However, this also means that now, more than ever, we need to scale up and push our abilities to their limits to classically simulate quantum mechanics, for instance, in order to be able to design and validate the first quantum simulation devices that will see the light.

B. On Monte Carlo simulation

It is crucial to emphasize the difference in nature between a quantum state $|\psi\rangle$ as described by (α_i), and a joint probability distribution $p = (p_i)$. After all, the memory size of the description of some arbitrary p is also exponential in the number of random variables. Yet, stochastic processes can easily be simulated with great accuracy through Monte Carlo methods. So why should quantum processes be any different? The difference boils down to a simple fact: probabilities are positive numbers, and they always add up constructively. This allows us to interpret $p = (p_0\ p_1)$ as 'either 0, or 1', i.e., substituting vectors for random choices; adding up the results observed over several runs. In contrast, magnitudes may have opposite signs that cancel out. For example, suppose that the state $|0\rangle$ maps to $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, and that $|1\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, thereby following the rather common Hadamard evolution. Then, after two time steps, $|0\rangle$ will always be taken back to itself, as the two scenarios $|0\rangle \longrightarrow \frac{1}{\sqrt{2}}|1\rangle \longrightarrow -\frac{1}{2}|1\rangle$

and $|0\rangle \longrightarrow \frac{1}{\sqrt{2}}|0\rangle \longrightarrow \frac{1}{2}|1\rangle$ interfere destructively. An unweighted Monte Carlo approach would fail to see this, as some runs would produce $|1\rangle$ and declare it reachable. A weighted Monte Carlo approach would obtain $\frac{1}{2}|1\rangle$ at times, $-\frac{1}{2}|1\rangle$ at others, failing to converge on whether $|1\rangle$ is reachable, in a way that is highly sensitive to overcounting in general. An exact-counting, weighted Monte Carlo may sometimes succeed in trading space complexity for time complexity [1], e.g., for sampling problems on fixed quantum circuits featuring particular sparsity conditions. In this paper, we do not assume that we are in such a specific case, and take the common view that, ultimately, destructive interference forces us to interpret $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ as "both 0 and 1, with some weights", thus having to keep track of (α_i).

C. On irregular quantum systems

Indeed, the quantum circuit case is specific in many ways. For instance, the description of its basis elements is of fixed size n. Moreover, the position as well as the nature of the next quantum gate to be applied are also fixed by the fixed classical layout of the local quantum gates. General quantum systems do not have such features. For instance, in a quantum Turing machine [2], the non-blank part of the tape may grow over time and even find itself in a quantum superposition of sizes. Even its head may find itself in quantum superpositions of locations, thereby inducing the application of a local quantum gate at different, superposed cells. The quantum field theories that provide our current best understanding of quantum particles do not have such features either (making them very hard to simulate [3], [4]), as initially well-located fundamental particles may spread out, infant virtual particles or not, trigger a variety of interactions at different places or not, in a quantum superposition. Our aim in this paper is to push the limit of the classical simulation of such irregular quantum systems; i.e. general systems whose basic elements are variable in size, and whose different sizes may indeed be reached throughout the evolution. Thus, although we may be able to provide an upper bound on the dimension of the subspace that is effectively being used at any point in time, the overall state space of such irregular quantum systems is in fact a countably infinite vector space. Whilst we do assume a well-defined unitary procedure to work out how each basic state will evolve in time, it may or may not have a description in terms of local quantum gates, and if it does, the layout may in general depend on the basic state at hand.

D. On causal graph dynamics

In addition to general quantum systems, our motivation for tackling this class of simulation problems lies particularly in our desire to explore quantum versions of some intriguing, classical reversible graph dynamics, namely the Hasslacher-Meyer (HM) model [5]. The HM model features cyclic graphs, with particles hopping from one vertex to another; see fig. 1. There are just two particle species: the left-movers and the right-movers. Thus, each vertex may be found in one of four possible states: empty, occupied by a left-mover, a right-mover, or both a left- and a right-mover. The HM model's dynamics alternates in two steps. During the "shifting" step S, left-movers move left, and right-movers move right, as shown fig. 1 (top left). Note that this step is reversible. During the "interaction step" I, the patterns shown in fig. 1 (bottom left) are replaced synchronously with one another across the cyclic graph. Note that this step is again reversible. In other words, particles move synchronously. When they cross, and depending upon the way that they cross, two vertices of the graph merge into one, or one vertex splits into two. The size of

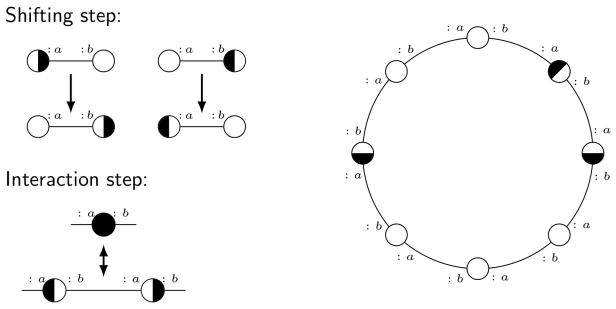


Fig. 1. *(Left.)* The HM model is an alternation of shifting steps (S) and interaction steps (I). *(Right.)* A possible graph.

the cycle will therefore grow or shrink accordingly. Intuitively, because HM is fully reversible, one might conjecture that the size oscillates at times, or perhaps that the number of graphs that tend to shrink equals the number of graphs that tend to grow. This, however, turns out not to be the case: almost every graph drawn at random has a tendency to grow, with $s(t) \propto \sqrt{t}$ [5].

This is surprising: due to reversibility, past and future ought to be symmetric, yet typical graphs tend to grow towards the future. This curious fact is only accessible to us through experiments: we do not know of a proof that typical graphs will grow. The HM model thus constitutes a particularly simple, numerical model that reconciles seemingly contradictory features in cosmology: i) the reversibility of the laws of physics, ii) the expansion of the universe, and iii) the emergence of an arrow of time from past to future. However, the fundamental reason for the reversibility of the laws of physics is the unitarity of quantum evolutions. Moreover, the universe is expanding slowly, yet exponentially as $s(t) \propto (1+\varepsilon)^t$, rather than quadratically. Could we engineer a quantum version of the HM model to obtain both unitarity and a low exponential expansion rate? Again, since no proof is known for the growth of the classical reversible HM model, we cannot expect to answer these questions analytically in the quantum setting, hence the need to tackle the problem of the classical simulation of irregular quantum dynamics. Answering them will in fact be left for a subsequent physics paper. Here, it suffices to say that the classical simulation quantum causal graph dynamics (QCGD) has proved challenging enough to motivate the development of a large-scale distributed simulation framework, and has provided us with prototypical examples as well as a wide test bench.

II. A QUANTUM MODEL

We present a series of quantum evolutions that will enrich the HM model. The simplest of which consists in simply assigning magnitudes to each complementary transformation of the HM model, which will be later called 'split-merge' for its ability to 'split' (add) or 'merge' (remove) vertices.

A. Quantum walks

Let us ignore the interaction step in fig. 1 for a moment. What we are left with is a particle system with synchronous and reversible updates. Such models have been studied abundantly in physics under the name of lattice-gas automata and in computer science under the name of reversible cellular automata. A well-known way to make this quantum is to precede the shifting step S, with a "coin step" C, mapping $|\CIRCLE\rangle$ to $\alpha|\CIRCLE\rangle + \beta|\LEFTcircle\rangle$. For this to be a valid quantum evolution, we must also map $|\LEFTcircle\rangle$ to $\gamma|\CIRCLE\rangle + \delta|\LEFTcircle\rangle$, in such a way that the corresponding operator over the Hilbert space $\mathcal{H}_{\{\CIRCLE,\LEFTcircle\}}$ be unitary:

$$U = \begin{pmatrix} \alpha & \gamma \\ \beta & \delta \end{pmatrix} = e^{i\xi} \begin{pmatrix} cos\theta & sin\theta e^{i\phi} \\ sin\theta e^{-i\phi} & -cos\theta \end{pmatrix}$$

where the RHS is the standard parametrization of $U(2)$. When $\theta = \pi/4, \phi = \xi = 0$ this is the Hadamard coin H. Beware that during the coin step C, U gets applied synchronously at every location (technically, one would write $C = \bigotimes_{v \in V(G)}(U \oplus Id_{\CIRCLE,\RIGHTcircle})$). For instance, C does

$$|\CIRCLE - \CIRCLE\rangle \mapsto \alpha^2|\CIRCLE - \CIRCLE\rangle + \alpha\beta|\CIRCLE - \LEFTcircle\rangle + \beta\alpha|\LEFTcircle - \CIRCLE\rangle + \beta^2|\LEFTcircle - \LEFTcircle\rangle.$$

The model hereby obtained is known as a quantum walk (QW), which have been intensively studied both as a means of formulating quantum algorithms or casting quantum simulation particles [6].

B. Quantum causal graph dynamics

Say that we now alternate the coin step C, the shifting step S and finally the interaction step I. The SC, QW-like step will create superpositions of being a collision pattern, or not. The I step will shrink the graph, or grow the graph, or have nowhere to act upon, in a quantum superposition, as in Fig. 2. This will lead to quantum superpositions of graphs [7] which

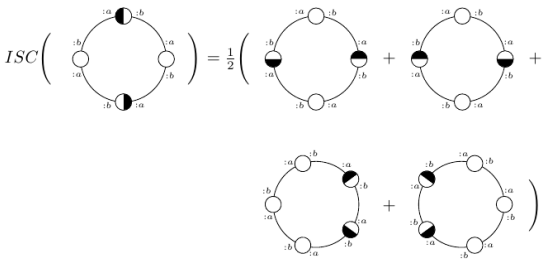


Fig. 2. *Example of the application of a QW SC with a classical interaction rule I.*

yield a first example of a QCGD [8].

Moreover, as mentioned earlier, QWs were defined just by transposing two patterns wherever they occur. We can apply the same idea to make a quantum version of the interaction step of fig. 1, called from now on a "split-merge". That is, we map $|\RIGHTcircle\rangle$ to $\alpha'|\RIGHTcircle\rangle + \beta'|\CIRCLE - \LEFTcircle\rangle$, etc., completing this into a unitary U'. Again, the unitary interaction U' is applied synchronously at every occurrence of the pattern. A crucial point is that this prescription is unambiguous because the two patterns do not overlap. The rule that transposes the pattern $\RIGHTcircle$ and $\LEFTcircle$ is called "erase-create", which does not preserve the number of particles. It can be made quantum and parametrized by a unitary U'' in the same way. We use it with some quantum magnitude, as a means to a certain particle density in spite of graph growth.

III. COMPUTATIONAL CHALLENGES

The simulation of irregular quantum systems such as QCGD poses various computational challenges that necessitate a particular treatment to accelerate the simulation through effective parallelism and optimized kernels. As previously alluded to, such systems involve many objects of varying size and magnitude to represent the quantum state, and the local application of dynamics on each object generates new objects that can vary greatly in size and count (as in the case of split-merge or erase-create). Moreover, among generated objects, many duplicates may exist, which need to be merged for the accuracy of the simulation. All of these aspects pose serious challenges in a distributed simulation setting to balance the computational and communication load as well as memory usage across processes. Moreover, the unbounded dimensionality in such simulations renders the existing parallel quantum simulation frameworks impractical for this purpose.

Our main contribution in this paper is the design and implementation of scalable parallel algorithms tailored to address the unique computational challenges of the simulation of such irregular quantum systems, as well as a distributed simulation software framework QUIDS, which provides an abstraction layer to integrate other irregular quantum dynamics (IQD) applications into the framework by domain experts.

IV. RELATED WORK

Classical simulations of quantum systems is an extremely well-studied domain, with numerous software frameworks ranging from MATLAB/Python implementations to high-performance parallel libraries with shared memory/distributed memory/GPU parallelism [9]. The focus of such libraries varies from the simulation of quantum computers and circuits using optimized HPC kernels [10] to the simulation of high-dimensional Hamiltonians using tensor networks with optimized matrix/tensor operations for effective low-rank compression and fast execution on modern architectures. Such frameworks, however, are not amenable to our use case in general due to its particular nature (unbounded dimensionality, extreme sparsity, and irregularity). QCGD investigates the evolution of quantum systems expressed as a quantum superposition of graphs under various dynamics, which is among

the IQD applications that motivate this work and has been the subject of recent studies [8], [11], [12]. Many outstanding questions in this domain can only be answered through large-scale quantum simulation, which renders QCGD particularly interesting in the context of this paper's focus.

V. Distributed QuIDS framework

A. General description of the framework

We describe the main steps of a single iteration of the parallel IQD simulation, which starts with a set of objects distributed across processes and generates a new set of distributed objects for the subsequent parallel iteration. Throughout the paper, we call an "iteration" the application of a single "dynamic" (or rule, described in the next section) to all objects to generate a new set, which consists of three major blocks: symbolic computation, duplicate elimination, and object generation.

In the spirit of many sparse irregular computations (e.g., sparse direct solvers, parallel sparse matrix-matrix multiplication), we employ a symbolic compeutation step within the iteration for the subsequent computation involving duplicate elimination and object generation, which provides many advantages (w.r.t. a "direct" computation of objects), such as

- pre-computing the layout of data structures, which enables a tight and contiguous memory allocation and fast in-place object generation in parallel,
- performing load balancing on symbolic objects (of small and fixed size) instead of real objects (of much greater size depending on the simulation/application), thereby avoiding expensive data movements across processes,
- generating significantly higher number of total objects before the duplicate object elimination and truncation. This is because we perform all elimination/truncation on the symbolic objects, which are significantly smaller than the real ones, and generate solely those to be kept for the next iteration. Having more objects also increases the overall accuracy of the simulation.

Once symbolic objects are created and duplicate objects merged (by adding their magnitudes and eliminating all but one copy), the remaining set of symbolic objects is truncated based on their magnitude so that there is sufficient memory to generate all objects in the next object generation step. Finally, the object generation step creates the "physical" objects corresponding to the remaining symbolic objects after elimination and truncation, filling out the data structures containing objects and their magnitudes.

B. Representation of a dynamic

For the purpose of our framework, we call an object to which a dynamic is applied a "parent", and the resulting objects from this application "children". We also define a dynamic by two functions that are applied to a parent object to generate its children: i) NUM-CHILDREN(obj) that returns the number of child objects the dynamic would create if applied to the parent object obj, and ii) CHILD(obj, id_{lc}, ψ) that generates a particular child with a local identifier. $id_{lc}, 0 \leq id_{lc} <$ NUM-CHILDREN(obj) among all children. Here, ψ is an optional parameter for the magnitude of obj; if provided, CHILD also returns the magnitude of the child object created. The ensemble of dynamics forms a sparse representation of the global quantum operator of the system. There could be many dynamics of different nature (e.g., having explosive or retracting characteristics on the number of objects) applied in a particular sequence throughout the simulation. In any case, our algorithm is oblivious to the exact nature of a dynamic and objects, as long as the dynamic provides these two functions required for our framework.

TABLE I
DESCRIPTION OF SYMBOLS USED IN ALGORITHMS.

Symbol	Description
$n_{obj}^{(0/1)}$	Number of objects at the current/next iteration.
$obj^{(0/1)}$	Vector containing the memory representations of objects of the current/next iteration.
$s^{(0/1)}$	Vector containing the size of the memory representation of each object in $obj^{(0/1)}$.
$\psi^{(0/1)}$	Vector containing the magnitudes (or specific component of the wave function) of objects at the current/next iteration.
$\widetilde{n_{obj}}$	Number of objects generated in the symbolic step.
$\widetilde{\psi}$	Vector containing the symbolic object magnitudes.
$\widetilde{H}$	Vector containing the hashes of symbolic objects.
$\widetilde{n_c}$	Vector containing the number of children of specific objects at the current/next iteration.
$\widetilde{id_{lc}}$	Vector containing the local id (between 0 and $n_c^{(c/n)}$) of child objects among their siblings (having the same parent object).
$\widetilde{id_p}$	Vector containing the parent object id of a specific symbolic object. Together with $\widetilde{id_{lc}}$, it allows to fully identify a child.
$\widetilde{s}$	Vector containing the size of the real object corresponding to each symbolic object (if it were to be formed).

C. Data structures

In table I, we describe the data structures used in the parallel IQD simulation. The steps of parallel simulation are described in an single-program multiple-data (SPMD) manner in the algorithms; each process possesses its own set of data structures provided in table I running the same steps of the parallel algorithm, exchanging data when necessary in the explicit collective communication steps. All vectors in this table are held as contiguous data structures in the memory.

For any variable x in the first half of table I (representing real objects), $x^{(0)}$ and $x^{(1)}$ respectively represent the state of the variable in the current and next iteration. Here, we have n_{obj} the total number of objects, obj the contiguous compressed data structure representing objects, and s and ψ their size (bytes) and magnitude (in the wave function). For a variable x, $\widetilde{x}$ denotes the variable in the symbolic step (provided in the second half of table I, corresponding to symbolic objects). Such variables are used to represent the

Algorithm 1: PRE-SYMBOLIC-STEP

1 **for** $i \leftarrow 0$ **to** $n_{obj}^{(0)} - 1$ **do**
2 $\quad \widetilde{n_c}[i] \leftarrow$ NUM-CHILDREN$(obj^{(0)}[i])$
3 BALANCE-CHILDREN$(obj^{(0)}, n_{obj}^{(0)}, \widetilde{n_c})$
4 $\psi^{(0)} \leftarrow$ TRUNCATE-SYMBOLIC$(s^{(0)}, \psi^{(0)})$
5 $\widetilde{n_{obj}} \leftarrow 0$
6 **for** $i \leftarrow 0$ **to** $n_{obj}^{(1)} - 1$ **do**
7 $\quad$ **if** $\psi^{(0)}[i] \neq 0$ **then**
8 $\quad\quad$ **for** $j \leftarrow 0$ **to** $\widetilde{n_c}[i]$ **do**
9 $\quad\quad\quad \widetilde{id_p}[\widetilde{n_{obj}}] \leftarrow i$
10 $\quad\quad\quad \widetilde{id_{lc}}[\widetilde{n_{obj}}] \leftarrow j$
11 $\quad\quad\quad \widetilde{n_{obj}} \leftarrow \widetilde{n_{obj}} + 1$

Algorithm 2: Balancing algorithm for children

1 BALANCE-CHILDREN$(obj^{(0)}, n_{obj}^{(0)}, \widetilde{n_c})$
2 $\quad$ **repeat**
3 $\quad\quad \{pair, \Delta_c\} \leftarrow$ FIND-BALANCE-PAIR$(\sum_i \widetilde{n_c}[i])$
4 $\quad\quad$ **if** $\Delta_c > 0$ **then**
5 $\quad\quad\quad \Delta_{obj} \leftarrow$ FIND-SUFFIX-SUM-IDX$(\widetilde{n_c}, \Delta_c)$
6 $\quad\quad\quad$ SEND-TAIL-OBJ$(obj^{(0)}, \widetilde{n_c}, \Delta_{obj}, pair)$
7 $\quad\quad\quad n_{obj}^{(0)} \leftarrow n_{obj}^{(0)} - \Delta_{obj}$
8 $\quad\quad$ **else**
9 $\quad\quad\quad \{objs, n_c\} \leftarrow$ RECEIVE-TAIL-OBJ$(pair)$
10 $\quad\quad\quad obj^{(0)} \leftarrow obj^{(0)} \bigcup objs$
11 $\quad\quad\quad \widetilde{n_c} \leftarrow \widetilde{n_c} \bigcup n_c$
12 $\quad\quad\quad n_{obj}^{(0)} \leftarrow n_{obj}^{(0)} + |objs|$
13 $\quad$ **until** *load balance is established.*

symbolic objects resulting from the application of a dynamic, without generating or storing the entire object.

D. Parallel algorithms

1) Pre-symbolic step: The main goal of this step is to pre-process distributed objects so that the subsequent generation of symbolic objects could be done in parallel efficiently (i.e., with good load-balancing) and reliably (i.e., avoiding memory under- and over-utilization).

A symbolic object consists of four variables; object magnitude ($\widetilde{\psi}[i]$), object hash ($\widetilde{H}[i]$), identifier of the parent object ($\widetilde{id}_p[i]$), and a local (child) identifier among its siblings ($\widetilde{id}_{lc}[i]$). Symbolic objects are small in size compared to real objects; however, depending on the dynamic, the number of symbolic objects stemming from a real object can be very large (potentially exponential w.r.t. the size of real objects) and vary greatly, which poses two challenges. First, there could be a significant imbalance among processes in terms of the number of symbolic objects to be generated. Second, processes might have insufficient memory to generate all symbolic objects.

The pre-symbolic step starts with a pass over all objects in lines 1-2 of algorithm 1, computing the number of symbolic child objects to be generated for each parent object. Then, a load-balancing step follows in line 3, which exchanges real objects to make sure that each process generates a similar number of child objects in the symbolic step. This exchange occurs through BALANCE-CHILDREN given in algorithm 2, which ensures that the relative imbalance in the counts of symbolic objects per node does not exceed a given threshold (which we set to 5%). When it does, the k$^{\text{th}}$ heaviest and lightest processes are paired through FIND-BALANCED-PAIR, which returns the $pair$ process id and the number of child objects that the process should send ($\Delta_c > 0$) or receive ($\Delta_c < 0$). The heavier process then finds the number of real objects (Δ_{obj}) to send from the tail of its object list $obj^{(0)}$ that have about Δ_c children in total, which is calculated in FIND-SUFFIX-SUM-IDX through a binary search on the suffix sum of the children counts (n_c). This is the only place where we communicate real objects, which is costly. However, our load-balancing methods on symbolic objects in the latter steps ensure that BALANCE-CHILDREN converges only in a few iterations in practice.

Balancing the number of children is expected to balance the computation/memory for the symbolic step; however, it does not guarantee that there is enough memory to generate all symbolic objects. For this reason, in line 4, we perform a truncation on the real objects of the current iteration using TRUNCATE-SYMBOLIC, which returns object magnitudes $\psi[i]$, setting it to 0 if $obj^{(0)}[i]$ is not to be kept.

For truncation, we use all remaining memory in the node, minus a small safety margin of $< 10\%$ for auxiliary variables/arrays, to determine how many objects could be kept. We expect that the truncation function "scores" objects based only on their magnitude ($\psi^{(0)}[i]$), generating a natural order of "preference". We then employ this in an algorithm that performs a binary search on successive bisections of these scores, finding the maximum number of objects such that the total size does not exceed the available memory and there is still additional free memory left to store real objects $obj^{(1)}$ of the next iteration. In doing so, we reserve the same amount of memory for $obj^{(1)}$ as $obj^{(0)}$ after truncation (i.e., $\sum_{i,\ \psi^{(0)}[i] \neq 0} s^{(0)}[i]$). Our algorithm is akin to the expected linear-time selection algorithm and avoids explicit sorting to find this threshold; we skip its details for brevity. This second condition also maximizes the amount of data retained, assuming a steady-state (i.e., memory is fully saturated), keeping the number of objects constant throughout the transformations. In this way, the loss of accuracy in this symbolic truncation and the truncation right before the object generation is better balanced. Finally, in lines 5-11, we process the remaining objects to fill in the parent and local child identifiers ($\widetilde{id}_p$ and $\widetilde{id}_{lc}$) of all symbolic objects and simultaneously find the total number of symbolic objects to be generated in each process.

2) Symbolic step: The symbolic step is described in algorithm 3, which iterates on all symbolic object indices, generating each child object from its parent using the dynamic

Algorithm 3: SYMBOLIC-STEP

1 **for** $i \leftarrow 0$ **to** $\widetilde{n_{obj}} - 1$ **do**
2 $\quad j \leftarrow \widetilde{id_p}[i]$
3 $\quad \{obj, \widetilde{\psi}[i]\} \leftarrow \text{CHILD}(obj^{(0)}[j], \widetilde{id_{lc}}[i], \psi^{(0)}[j])$
4 $\quad \widetilde{s}[i] \leftarrow \text{SIZE}(obj)\ \widetilde{H}[i] \leftarrow \text{HASH}(obj)$

and computing its magnitude ($\widetilde{\psi}[i]$). Next, the size of the object is stored ($\widetilde{s}[i]$) for the subsequent object generation step. Here, though we generate all "real" objects for the next iteration, we do it one at a time, which means that this step only needs $\max_i \widetilde{s}[i]$ additional storage, which is feasible.

As alluded to in section I, a dynamic may create many identical objects with different magnitudes. Such objects would need to be "merged", by adding their magnitudes and eliminating all copies but one, to ensure the accuracy of the simulation, for two reasons. First, the truncation algorithm might have a propensity to favor objects having higher magnitude, hence potentially eliminating many or all such copies having a magnitude smaller than the "merged" object that would be kept otherwise. Second, such copies occupy memory that could be used to store other "meaningful" objects, which would demand fewer object truncations and hence better simulation accuracy.

For this purpose, we compute and store the hash of each object ($\widetilde{H}[i]$) in the symbolic step at line 4, which will then be used to merge those having identical hash values in DUPLICATE-ELIMINATION. Though this approach is not entirely accurate per se, our rationale is that i) the simulation is already an approximation by nature as we ought to heavily truncate generated objects; therefore, the perturbation due to few objects lost during hash collision should be tolerable in most cases, and ii) one can either increase the hash size or employ a secondary/tertiary hash to reduce the number of such collisions drastically to the point of becoming negligible/nonexistent if needed. This is due to the well-known Birthday Paradox phenomenon: using an N-bit hash, the number of collisions will be small statistically speaking as long as the number of objects is less than $2^{\sqrt{N}}$.

3) Duplicate elimination: The duplicate elimination step provided in algorithm 4 merges symbolic objects with identical hashes $\widetilde{H}[i]$ using a hashmap ($\widetilde{map}$) by adding their magnitudes $\widetilde{\psi}[i]$ and leaving a unique copy for the subsequent pre-object generation step. In a distributed setting, a natural question that arises is which process should merge which objects/hash values, a decision which has strong implications on parallel performance as it determines the computational and communication load of each process.

In this regard, we exploit the property that a "good" hash function ought to produce "bits" that are completely random w.r.t. the input. In line 1 of algorithm 4, we group symbolic objects through FORM-BUCKETS into $n_{buckets}$ buckets using the last $\log_2(n_{buckets})$ bits of their hash values. For this purpose, each process reorders its symbolic objects so that they become continuous w.r.t. bucket indices, which is performed

Algorithm 4: DUPLICATE-ELIMINATION

1 $\rho_{send} \leftarrow \text{FORM-BUCKETS}(\widetilde{H},\ n_{buckets})$
2 $\rho_{global} \leftarrow \text{ALL-REDUCE}(\rho_{send}, +)$
3 $\Pi \leftarrow \text{BALANCE-BUCKETS}(\rho_{global},\ n_{procs})$
4 $\rho_{send} \leftarrow \text{MERGE-BUCKETS}(\rho_{send}, \Pi)$
5 $\rho_{recv} \leftarrow \text{ALL-TO-ALL}(\rho_{send})$
6 $\{\widetilde{H}, \widetilde{\psi}\} \leftarrow \text{ALL-TO-ALLV}(\widetilde{H}, \widetilde{\psi}, \rho_{send}, \rho_{recv})$
7 **for** $i \leftarrow 0$ **to** $\sum \rho_{recv}$ **do**
8 $\quad$ **if** $\widetilde{H}[i] \notin \widetilde{map}$ **then**
9 $\quad\quad \widetilde{map}[\widetilde{H}[i]] \leftarrow i$
10 $\quad$ **else**
11 $\quad\quad j \leftarrow \widetilde{map}[\widetilde{H}[i]]$
12 $\quad\quad$ **if** $\text{WORK-STEAL}(j,\ i)$ **then**
13 $\quad\quad\quad \widetilde{\psi}[i] \leftarrow \widetilde{\psi}[i] + \widetilde{\psi}[j]$
14 $\quad\quad\quad \widetilde{\psi}[j] \leftarrow 0$
15 $\quad\quad\quad \widetilde{map}[\widetilde{H}[i]] \leftarrow i$
16 $\quad\quad$ **else**
17 $\quad\quad\quad \widetilde{\psi}[j] \leftarrow \widetilde{\psi}[j] + \widetilde{\psi}[i]$
18 $\quad\quad\quad \widetilde{\psi}[i] \leftarrow 0$
19 $\widetilde{\psi} \leftarrow \text{ALL-TO-ALLV}(\widetilde{\psi},\ \rho_{recv},\ \rho_{send})$

in linear time with two passes on the objects. The function returns the number of local symbolic objects in each bucket, ρ_{send}, which is then aggregated across all processes in the following ALL-REDUCE step in line 2 to obtain the global object count per bucket (ρ_{global}). Here, buckets are already expected to have mostly similar number of objects due to the hash property. We nonetheless increase the granularity slightly by using $n_{buckets} > kn_{procs}$ for a small constant $k > 1$ to have more leeway in achieving load balance during the assignment of buckets to processes next. In line 3 of algorithm 4, we compute this assignment (Π) through BALANCE-BUCKETS so that each process ends up with a similar number of symbolic objects to merge. A bin-packing algorithm could be used to this end; however, we rather opt for Nicol's chains-on-chains partitioning (CCP) algorithm [13] for two reasons. First, when the bucket weights do not vary substantially, as is in our case, CCP has strong theoretical guarantees for the maximum imbalance and works remarkably well. Second, assigning a continuous set of buckets ensures that the follow-up ALL-TO-ALLV communication of symbolic objects in lines 6 can be performed without any data reshuffling. We do this partitioning redundantly in each process as the CCP cost is negligible ($O(n_{buckets} + n_{procs}^2 \log_2^2(n_{procs}))$ w.r.t. the rest of the computation on objects, and parallel implementations could be employed if needed. In line 4, the set of contiguous buckets assigned to each process is merged into one, providing one bucket to be sent from each process to each process. Lines 5-6 exchange symbolic objects first be sent communicating send counts (ρ_{send}) to obtain receive counts (ρ_{recv}) through ALL-TO-ALL, then performing an ALL-TO-ALLV exchange of symbolic objects.

In lines 7-18, each process performs a merge of identical objects, aggregating magnitudes in the unique copy to be kept and eliminating others by setting their magnitude ($\widetilde{\psi}[i]$) to 0. This elimination could potentially disturb the load balance in case some processes lose too many objects in elimination w.r.t. others. To prevent this, we employ a work-stealing heuristic which is employed when a hash conflict is detected between an object j and its previously inserted "unique" copy i. The WORK-STEAL oracle decides if j should be marked as unique instead, which happens when the owner of j has less unique objects *locally* (to avoid communication) within the merger process than the owner of i. Line 19 then performs another ALL-TO-ALLV to redistribute updated magnitudes ($\widetilde{\psi}$), which mirrors the communication in line 6.

4) Pre-object-generation step: At this point, we have a unique copy of symbolic objects distributed evenly across processes, ready to be transformed into real objects. However, it is unlikely to have enough memory to construct all these objects; hence, a truncation step will again be needed. This is performed through TRUNCATE-REAL, which is in essence the same as TRUNCATE-SYMBOLIC in algorithm 1, except that it dedicates *all* remaining free memory (minus the same safety margin) to the generation of real objects. In lines 2-6, we then

Algorithm 5: PRE-OBJECT-GENERATION

```
1 ψ̃ ← TRUNCATE-REAL(s̃, ψ̃)
2 n_obj^(1) ← 0
3 for i ← 0 to ñ_obj − 1 do
4     if ψ̃[i] ≠ 0 then
5         ψ^(1)[n_obj^(1)] ← ψ̃[i]
6         s^(1)[n_obj^(1)] ← s̃[i]
7         n_obj^(1) ← n_obj^(1) + 1
8 REALLOCATE(obj^(1), s^(1))
```

compress the lists of magnitudes $\widetilde{\psi}$ and sizes $\widetilde{s}$ w.r.t. objects with zero probability (either from duplicated elimination in section V-D3, or from earlier truncation) to obtain their final compact analogs $\psi^{(1)}$ and $s^{(1)}$ for object generation. Once we have the final set of real objects to be generated and their size, REALLOCATE prepares the pointers for the compressed data structure ($obj^{(1)}$) accommodating the objects of the next iteration.

5) Generating next-iteration objects: We finally generate the final objects on $obj^{(1)}$ by iterating over each symbolic object that is kept and applying the dynamic to its parent object, as shown in lines 1-5 in algorithm 6. We then finalize the iteration by normalizing the magnitudes of all objects in lines 7-9, which serves to counteract the loss of norm due to truncation and keep a "meaningful" quantum state.

E. Object ranking for truncation

Both steps of TRUNCATE-SYMBOLIC in algorithm 1 and TRUNCATE-REAL in algorithm 5 expect a ranking of objects

Algorithm 6: OBJECT-GENERATION

```
1 j ← 0
2 for i ← 0 to ñ_obj − 1 do
3     if ψ̃[i] ≠ 0 then
4         obj^(1)[j] ← CHILD(obj^(0)[id̃_p[i]], id̃_lc[i])
5         j ← j + 1
6 P_total ← ALL-REDUCE(Σ_i |ψ^(1)[i]|^2, +)
7 for i ← 0 to n_obj^(1) − 1 do
8     ψ^(1)[i] ← ψ^(1)[i]/√P_total
```

based on their importance for the accuracy of the simulation. Here, we discuss two possible approaches of different nature for such a ranking.

1) Deterministic ranking : The most intuitive way to rank objects to maximize accuracy would be to favor those with higher magnitude. Despite being logical and fast, one drawback of this method is its potential to introduce some biases in the simulation. One such example would be when all objects with a specific property also happen to have small magnitudes; truncation using such a deterministic ranking would entirely eliminate this property from the simulation.

2) Probabilistic ranking : To avoid aforementioned bias scenarios, we can opt for a probabilistic ranking, similar to a quantum Monte Carlo method [14], where we rank objects higher with a probability proportional to their magnitude in the wave function. This is usually performed iteratively via a method such as the Metropolis–Hastings algorithm [15], which would be far too costly in our case given the number of objects we have. We rather employ a simpler alternative using which we observed a similar general behavior in our experiments. Through some experimentation, to rank the objects, we tailored the distribution

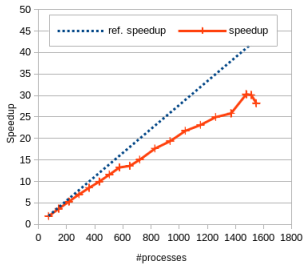
$$X(p) = -U_{[0,1]}/p$$

with $U_{[0,1]}$ denoting the uniform distribution in $[0, 1]$. We tested this in order to match three different distributions of object magnitudes/probabilities; uniform, exponential, and inverse polynomial, and managed to obtain a close-to-perfect correspondence in all cases - with an exact result for a single element to be sampled, and almost exact (less than 1% relative deviation) sampling for up to 20% of the total. This novel strategy can easily be executed in parallel by performing random sampling in each node followed by keeping the N highest-valued objects that can fit in the memory.

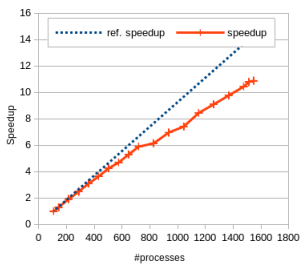
VI. EXPERIMENTS

A. Experimental setup

We tested the scalability of our framework for the special case of QCGD simulations [8], which proves to be challenging from a computational point of view. This is because i) the objects necessitate an irregular memory representation and accesses, ii) the simulation tends to exhibit an exponential



(a) split-merge, 16 iterations, starting graph with 14 vertices.



(b) erase-create, 13 iterations, starting graph with 14 vertices.

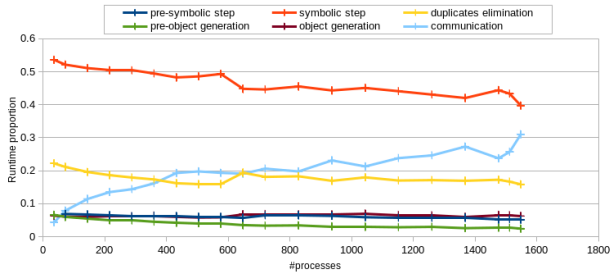
Fig. 3. Strong scaling results using split-merge and erase-create dynamics. "ref. speedup" represents the perfect speedup.

growth in object counts, and iii) generated objects tend to have a highly variable number of children. That said, we also managed to successfully run our framework on other examples such as the simulation of simple quantum gates with the ability to add/destroy qubits, which we will not discuss here.

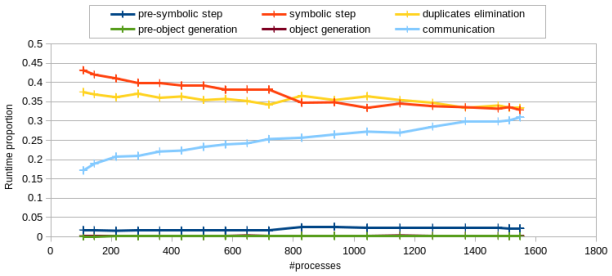
We ran our code on a parallel compute platform having a maximum of 44 nodes (of which a maximum of 43 were accessible to use simultaneously) and 192GB of memory per node (8TB of total memory for 43 nodes) and two sockets populated with 18-core intel Xeon Skylake Gold 6240 CPUs@2.6GHz each, with turbo boost and hyperthreading disabled for reliable timings. The nodes are connected through an OmniPath 100Gb/s network switch, allowing a high-bandwidth and low-latency omnidirectional communication. Our code is implemented in C++20 and parallelized using MPI. For each experiment, we fully saturate nodes by assigning one MPI process per CPU core, using up to 1548 total MPI processes.

B. Strong scalability

We first provide the strong scaling results for this case using 1 to 43 nodes (or 36 to 1548 MPI processes, respectively). For erase-create, we tried to simulate an "exact" computation without truncation (so we have the same "problem" no matter the number of nodes used), which required at least 3 nodes to fit in the memory. For split-merge, truncation is indispensable due to the unlimited growth of object counts; therefore, we instead fixed the maximum amount of total objects to a number that would fit into a single node (having 192GBs of memory) across all runs. Simulations using both split-merge in fig. 3a and erase-create in fig. 3b scale all the way up to the maximum number of MPI processes, with a slight flattening in the speedup curve after around 600-700 MPI processes and a minor slowdown for split-merge using 43 nodes. Next, in figs. 4a and 4b, we break down the total execution time into five distinct steps of the simulation, plus the total communication time. As the number of processes increases, we observe a slight increase in the communication percentage, which is expected due to all-to-all exchanges in most communication steps, and a slight decrease in the symbolic step. The proportion of other steps stays mostly stable, with only small fluctuations across runs. These figures



(a) Dissection of runtime for split-merge in fig. 3a.



(b) Dissection of runtime for erase-create in fig. 3b.

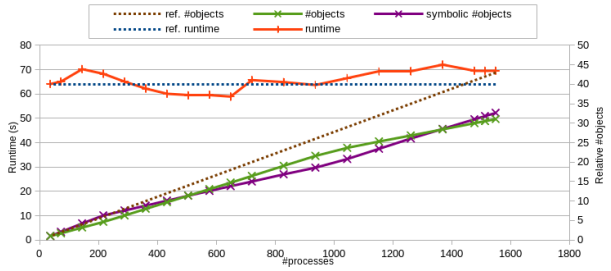
Fig. 4. Dissection of the total runtime for split-merge and erase-create strong scaling results.

also underline the computational difference between these two types of dynamics; a high collision-rate (erase-create) results in a high duplicate-elimination cost compared to a low collision-rate (split-merge) dynamic where the symbolic step dominates as it generates more objects.

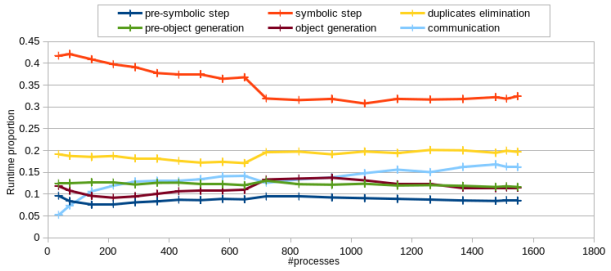
C. Weak scalability

We now present weak scaling results for our parallel algorithms. As the problem size should increase proportionally to the number of processes for weak scaling, we let the dynamic fill the memory and truncate when needed. For the purposes of weak scalability, we define the "size" of the problem as the total memory occupied by the objects, which increases proportionally to the number of nodes we employ. We evaluate the weak scaling using two classes of dynamics; namely, those with low-collision (split-merge) and high-collision (erase-create) rates.

1) Weak scalability in memory-limited simulations with low collision rate : We first evaluate the weak scaling in a simulation using a dynamic with a non-bounded attainable state, i.e., split-merge, provided in figs. 5a and 5b. In fig. 5a, we observe an excellent weak scaling with the conservation of the runtime up to 1000 processes, then a slight increase towards the end. Considering the runtime dissection in fig. 5b, we see that this is mostly due to a slight increase in the total communication time, which is again to be expected due to all-to-all communication routines. In fig. 5a, we also plot the evolution of the total number of symbolic and real objects relative to the single node execution, and observe a sublinear growth in both of them at the same rate. The rest of fig. 5b tells a similar story to fig. 4b, with the proportion of symbolic step decreasing as the communication increases, duplicate



(a) Weak scaling runtime and #objects. "ref. runtime" represents the execution for a perfect weak scaling whereas "ref. #objects" denotes a perfectly linear increase in the number of symbolic/real objects w.r.t. #processes. vertices.

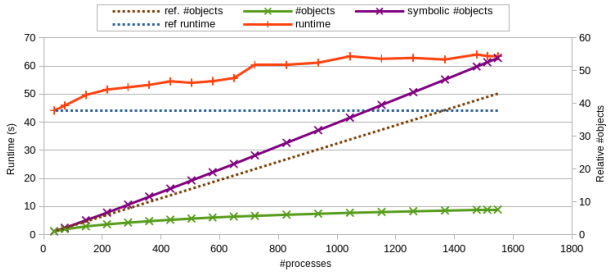


(b) Dissection of the total runtime.

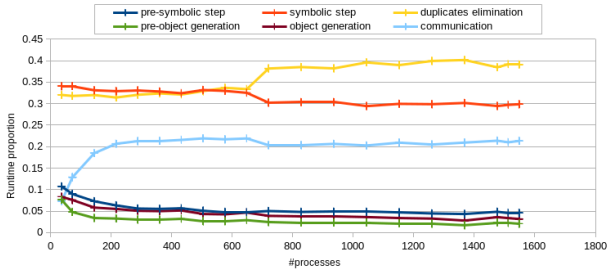
Fig. 5. Weak scaling results using the split-merge dynamic. Results represent the parallel runtime for 10 iterations using an initial graph having 14 vertices.

elimination step slightly increasing, and other steps staying mostly stable except small fluctuations.

2) Weak scalability in memory-limited simulations with high collision rate : Here, we perform a similar weak scaling evaluation in a high collision rate simulation using the erase-create dynamic, results of which are provided in fig. 6a. We employ a slightly bigger initial graph with a larger attainable space that cannot be fully stored (before duplicate elimination) even using the entire cluster; therefore, memory becomes fully saturated in all executions as intended. In fig. 6a, we again observe a respectable weak scaling, with a parallel runtime slightly increasing up to 600 processes, then staying constant. The dissection of runtime provided in fig. 6b shows a slightly higher communication w.r.t. fig. 5b (0.21 vs 0.16) as well as a more dominant duplicate elimination step, which is to be expected due to the higher collision rate of the erase/create dynamic. In fig. 6a, we also observe that the number of objects grows very slowly, especially after 600 processes, which frees up more memory space for symbolic objects, leading to their superlinear growth. This is because the attainable space is small enough that an increasingly smaller fraction of symbolic objects can represent it. The breakdown of the runtime in fig. 6b is comparable to the timings given in fig. 5b, with both duplicate elimination and the symbolic steps representing a much larger proportion of the runtime, as the collision rate is higher in this case (around 99.9% compared to 1%). For this reason, the number of symbolic objects to generate is high compared to the number of real objects to generate.



(a) Weak scaling runtime and #objects.



(b) Dissection of the total runtime.

Fig. 6. Weak scaling using the erase-create dynamic. Results represent the parallel runtime for 9 iterations using an initial graph having 17 vertices.

TABLE II
THE EFFECT OF THREE LOAD BALANCING STRATEGIES ON PERFORMANCE IN THREE SIMULATION CASES INVOLVING WEAK/STRONG SCALING FOR ERASE-CREATE (EC) AND STRONG SCALING FOR SPLIT-MERGE (SM), USING 23 NODES. BB, WC, AND BC RESPECTIVELY REPRESENT BALANCE-BUCKETS, WORK-STEAL, AND BALANCE-CHILDREN LOAD-BALANCING METHODS. RESULTS REPRESENT THE SPEEDUP W.R.T. THE BASELINE EXECUTION EXCLUDING ALL THREE METHODS.

Simulation	BB	BB + WS	BB + BC	BB + WS + BC
Weak SM	0.99	1.24	1.03	1.16
Weak EC	0.98	1.09	1.07	1.06
Strong EC	1.02	3.02	3.43	3.46

D. Load balancing

In table II, we show the impact of load balancing techniques we employ on symbolic (BALANCE-BUCKETS and WORK-STEAL) and real objects (BALANCE-CHILDREN). We provide the speedup with respect to the baseline execution excluding these steps (where exactly k contiguous buckets are statically assigned to each process for symbolic objects, and no work stealing or exchange of real objects is performed) using one, two, or three of these load balancing techniques. We first note that BALANCE-BUCKETS did not provide a substantial gain/loss on performance (a 2% gain, or a 1-2% loss due to the CCP computation and ALL-REDUCE communication overhead). This is mostly because the static partitioning (assigning k buckets per process) provides an excellent load balance since the randomization due to hashing almost perfectly balances the bucket weights in the case of QCGD, meaning that the optimal CCP solution yields the same/similar partition as the static partitioning. We believe that it would be prudent nonetheless to employ CCP in case buckets vary significantly in size in certain applications (i.e., involving a high number of identical

objects whose hash fall in the same bucket). Note that this test was not performed for the strong scaling of the low collision rate (split-merge) simulation, because the number of objects per node is artificially limited, which prevents reliably measuring potential load imbalances since any extra objects that could cause imbalance are pruned due to this artificial limit.

Next, we observe that employing WORK-STEALING and BALANCE-CHILDREN with BALANCE-BUCKETS results in a remarkable speedup for the strong scaling case using erase-create (2.97x and 3.35x), which is to be expected since even a slight imbalance in the number of objects can entail a snowball effect and create a dramatic difference in the number of objects per process after a few iterations. We also observe a gain in weak scaling cases, yet to a much lesser extent (3-24%). This is because in the weak scaling case, whenever an imbalance occurs, lightly loaded processes immediately "fill" the available space by performing fewer truncations at the end of the iteration, establishing the load balance for the next iteration, thereby avoiding a snowball effect. This, however, is not without a cost; heavily loaded processes will perform more truncation in contrast, leading to the elimination of important objects that would be kept otherwise (i.e. in a load balanced case), and thereby potentially affecting the simulation accuracy. Combining all three load-balancing strategies gave the best result in the strong scaling case, and similar results to the combination of two methods in the weak scaling case.

E. Memory management

We evaluated the efficiency of the memory usage of our framework in a simulation using the split-merge dynamic run for 18 iterations using 4 compute nodes. We set the memory safety margin to 10% in the truncation steps. We observe an exponential growth in the memory usage up until iteration 9, where the first truncation occurs, then a stable saturation of the memory space, which is indispensable to ensure a better simulation accuracy as discussed. We also observed a balanced memory usage between nodes with less than 1% difference owing to the load balancing done at different simulation stages.

VII. CONCLUSION

In this paper, we present a distributed simulation framework, QUIDS, for the simulation of large-scale quantum systems having sparse irregular operators with potentially unbounded dimensionality. We outline computational challenges stemming from a such simulation and introduce efficient parallel algorithms that incorporate symbolic computation, truncation, duplicate detection and elimination, communication and load-balancing, as well as memory management and parallel sampling strategies. These techniques result in high efficiency and good strong and weak scalability up to 1548 MPI ranks on a parallel cluster. Additionally, we provide an accompanying software library that offers the necessary abstractions to integrate other IQD applications with minimal effort and provides drop-in distributed parallelism, making HPC accessible to many domain experts in this promising field.

ACKNOWLEDGMENT

This work has been partially funded by the European Union through the MSCA SE project QCOMICAL, by the French National Research Agency (ANR): projects TaQC ANR-22-CE47-0012 and within the framework of "Plan France 2030", under the research projects EPIQ ANR-22-PETQ-0007, OQULUS ANR-23-PETQ-0013, HQI-Acquisition ANR-22-PNCQ-0001 and HQI-R&D ANR-22-PNCQ-0002, SELESTE ANR-20-CE46-0008-01, and by the ID #62312 grant from the John Templeton Foundation, as part of the 'The Quantum Information Structure of Spacetime' Project (QISS). The opinions expressed in this project/publication are those of the author(s) and do not necessarily reflect the views of the John Templeton Foundation. The experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine.

REFERENCES

[1] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, "Quantum supremacy is both closer and farther than it appears," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1807.10749

[2] E. Bernstein and U. Vazirani, "Quantum complexity theory," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM New York, NY, USA, 1993, pp. 11–20.

[3] W. Fan and F. Liu, "A numerical method for solving the two-dimensional distributed order space-fractional diffusion equation on an irregular convex domain," *Applied Mathematics Letters*, vol. 77, pp. 114–121, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893965917303142

[4] G. Magnifico, T. Felser, P. Silvi, and S. Montangero, "Lattice quantum electrodynamics in (3+1)-dimensions at finite density with tensor networks," *Nature Communications*, vol. 12, 2021.

[5] B. Hasslacher and D. A. Meyer, "Modelling dynamical geometry with lattice gas automata," June 1998, expanded version of a talk presented at the Seventh International Conference on the Discrete Simulation of Fluids held at the University of Oxford.

[6] J. Kempe, "Quantum random walks: an introductory overview," *Contemporary Physics*, vol. 44, no. 4, pp. 307–327, 2003.

[7] P. Arrighi, M. Christodoulou, and A. Durbec, "On quantum superpositions of graphs, no-signalling and covariance," *arXiv preprint arXiv:2010.13579*, 2020.

[8] P. Arrighi and S. Martiel, "Quantum causal graph dynamics," *Physical Review D.*, vol. 96, no. 2, p. 024026, 2017, pre-print arXiv:1607.06700.

[9] Quantiki, "List of quantum computing simulators," https://quantiki.org/wiki/list-qc-simulators, last updated: march 2020.

[10] M. Smelyanskiy, N. P. D. Sawaya, and A. Aspuru-Guzik, "qhipster: The quantum high performance software testing environment," 2016. [Online]. Available: https://arxiv.org/abs/1601.07195

[11] P. Arrighi, C. Cedzich, M. Costes, U. Rémond, and B. Valiron, "Addressable quantum gates," 2021. [Online]. Available: https://arxiv.org/abs/2109.08050

[12] T. Konopka, F. Markopoulou, and S. Severini, "Quantum graphity: A model of emergent locality," *Phys. Rev. D*, vol. 77, p. 104029, May 2008. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.77.104029

[13] A. Pinar and C. Aykanat, "Fast optimal load balancing algorithms for 1d partitioning," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, 12 2002. [Online]. Available: https://www.osti.gov/biblio/835143

[14] M. Caffarel and P. Claverie, "Development of a pure diffusion quantum Monte Carlo method using a full generalized Feynman-Kac formula. I. Formalism," *Chem. Phys.*, vol. 88, no. 2, pp. 1088–1099, Jan. 1988.

[15] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, Jun. 1953.