Multithreaded parallelism for heterogeneous clusters of QPUs

Philipp Seitz
Technical University of Munich
TUM School of Computation, Information and Technology
Department of Computer Science
Boltzmannstraße 3, 85748 Garching, Germany
0000-0003-3856-4090

Manuel Geiger
Technical University of Munich
TUM School of Computation, Information and Technology
Department of Computer Science
Boltzmannstraße 3, 85748 Garching, Germany
0000-0003-3514-8657

Christian B. Mendl
Technical University of Munich
TUM School of Computation, Information and Technology
Department of Computer Science and *TUM Institute for Advanced Study*
Boltzmannstraße 3, 85748 Garching, Germany
0000-0002-6386-0230

***Abstract*—In this work, we present MILQ, a quantum unrelated parallel machines scheduler and cutter. The setting of unrelated parallel machines considers independent hardware backends, each distinguished by differing setup and processing times. MILQ optimizes the total execution time of a batch of circuits scheduled on multiple quantum devices. It leverages state-of-the-art circuit-cutting techniques to fit circuits onto the devices and schedules them based on a mixed-integer linear program. Our results show a total improvement of up to 26 % compared to a baseline approach.**

***Index Terms*—Quantum Computing, High Performance Computing, Scheduling**

I. Introduction

Quantum computing promises exponential speedup compared to classical computing for certain computational tasks [1]. To realize such a quantum advantage with fault tolerance, the estimated number of required qubits is roughly 10^8 [2], considerably more than the number provided by currently available devices (for example, IBM Osprey with 433 qubits).

This is due to overhead introduced by error correction, which is necessary because of the error-prone hardware realizations of qubits. In the near term, only small, noisy quantum devices are available, and their access is limited. Still, many supercomputing centers have already started integrating quantum devices on their premises. Multiple physical realizations of quantum computers exist (called modalities), such as superconducting qubits [3], neutral atoms [4], or ion traps [5], each with unique characteristics. To diversify their portfolio, supercomputing centers prepare heterogeneous infrastructures supporting multiple modalities. In this article, we propose a multithreading scheme for such a platform to maximize utilization. We also implement a prototype, available as a repository on GitHub at https://github.com/qc-tum/milq.

II. Motivation

In the noisy intermediate-scale quantum (NISQ) era, quantum resources are still scarce. While the individual device sizes keep growing, access remains limited. Unfortunately, most circuits do not fit perfectly on a device. If the circuit is too small, this leads to underutilization. If the circuit is too large, it can be cut into fitting parts, but likely with a remainder that underutilizes the quantum processing unit (QPU). This is especially wasteful, considering that devices are exclusively assigned to users. As an extreme example, running a variational algorithm can block a device for a long time, even if enough qubits are available to run other simulations. MILQ solves this issue by considering all submitted circuits before running them. It resizes circuits appropriately and uses available resources in parallel. As a result, it is possible to have multiple circuit instances running on one QPU, hence the term "multithreading".

Due to its modular nature, MILQ extends beyond NISQ. One can integrate communication overhead as an additional constraint when considering distributed quantum systems. It is also relevant as a case study since many techniques apply to near-term integration scenarios. Many supercomputing centers have started adopting QPUs as novel accelerators. In this domain, scheduling is a common problem. MILQ is an initial attempt at providing a runtime component that can be expanded further.

III. Background and Related Work

MILQ combines techniques from various application areas. In this section, we provide the necessary background knowledge and investigate comparable solutions. The scheduling component is based on a mixed-integer linear program (MILP), hence the name MILQ.

The scheduling of quantum circuits is still an emerging problem. No established tool exists; most existing schedulers

are based on simple first-in, first-out concepts. One can consider scheduling as a part of the mapping problem if one includes the mapping over distributed devices [6]. A quantum circuit can be arbitrarily distributed, assuming gate teleportation between devices is possible, which is not necessarily true. Bhoumik et al. [7] consider the scheduling problem regarding error mitigation by finding optimal mappings from (cut) subcircuits to multiple QPUs. Their work is based on an integer linear program, which optimizes for circuit fidelity.

A. Scheduling

General scheduling is an optimization problem studied in operations research and computer science. Using the Graham $\alpha|\beta|\gamma$ classification scheme [8], the problem of interest in this article can be described as $R|\text{res}_1|C_{\max}$. It is a variant of the unrelated parallel machine problem, which has been studied extensively in literature [9]–[12]. R indicates m completely independent machines (in our case the QPUs), which use a single shared resource res_1, namely qubits. The problem is optimized for the makespan $C_{\max}$, the latest completion time of any job. Unfortunately, it is NP-hard to solve this exactly [12]. Hence, such problems are usually stated as a MILP and solved using heuristic approaches. Given the resource constraints, the goal is to provide a schedule where each job is assigned to one machine, and the overall execution time is minimized. Standard techniques involve Simulated Annealing, Tabu Search, and genetic algorithms [13]. Depending on the availability of the job information, algorithms either work offline when all information is known ahead of time or online when information is only accessible right before scheduling. A variant of the online version considers batches of jobs that are scheduled at the same time.

In computer science, scheduling is relevant in operating systems and high performance computing (HPC). Users of an HPC cluster submit jobs to the system, which are initially queued. Then, they are assigned a portion of the available resources for a fixed amount of time. Typically, the exact resource assignment is not disclosed to the user. The requests are usually handled by a resource management tool, most commonly SLURM [14], which also provides diagnostics and monitoring utilities. System utilization and overall runtime are the relevant metrics.

B. High Performance Computing and Quantum Computing

Integrating quantum computing (QC) into HPC is an ongoing process. Emerging microarchitectures and multiple integration scenarios are part of the challenges in this domain [15]. QC is transitioning out of the laboratories, and computing centers are starting to integrate quantum hardware into their premises. There is no established straightforward blueprint for combining classical with quantum infrastructure. In this state of uncertainty, the computing centers provide multiple modalities with varying capabilities. Emerging standards for common interfaces, like QIR [16], abstract the hardware details from the user. Still, most algorithms are run on a single device based on hardware availability. This leads to underutilization, as the entire device is reserved for the whole duration of the algorithm. As a user, this is also frustrating; the time-to-solution increases due to the longer queueing times.

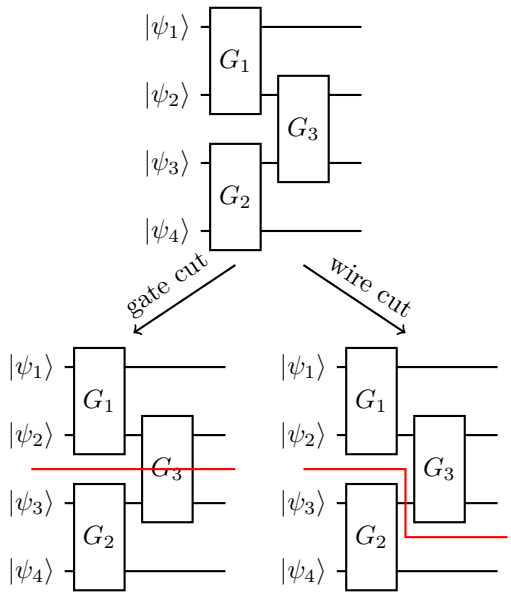


Fig. 1. Circuit knitting in two variants: gate and wire cuts are possible.

C. Circuit Knitting

In the NISQ era, hardware is physically limited, restricting the possible circuits in two ways. First, the *width* of a circuit is restricted by the number of available qubits; second, the *depth* of a circuit is limited by the decoherence times of the qubits. *Circuit Knitting* (also called *Circuit Cutting*) is a method to resize circuits in both dimensions. This comes at the cost of additional sampling overhead and classical computation.

Wire cuts reduce circuit depth by cutting wires in a circuit and executing the resulting partial circuits at different times [17]. *Gate cuts*, on the other hand, reduce circuit width by decomposing (multi-qubit) gates [18]. Both techniques reconstruct the expectation value of the original circuit by sampling from a quasi-probability distribution from the resulting sub-circuits. The procedures are depicted in Fig. 1 in a simplified manner. Recent works [19], [20] keep improving sampling overhead with advanced techniques.

IV. Proposal

MILQ is a standalone project built on the infrastructure of the popular quantum software framework Qiskit [21]. The intended workflow is summarized in Fig. 2. Users build their circuits with Qiskit and specify multiple hardware backends. For example, two jobs *A* and *B* are submitted to a system that combines the two fictional devices *QPU 1* and *QPU 2*. The circuits are compiled individually before being submitted to the system; any compilation is treated as a black box. After compilation, the circuits are submitted as a job to a joint interface. The system greedily resizes circuits using knitting

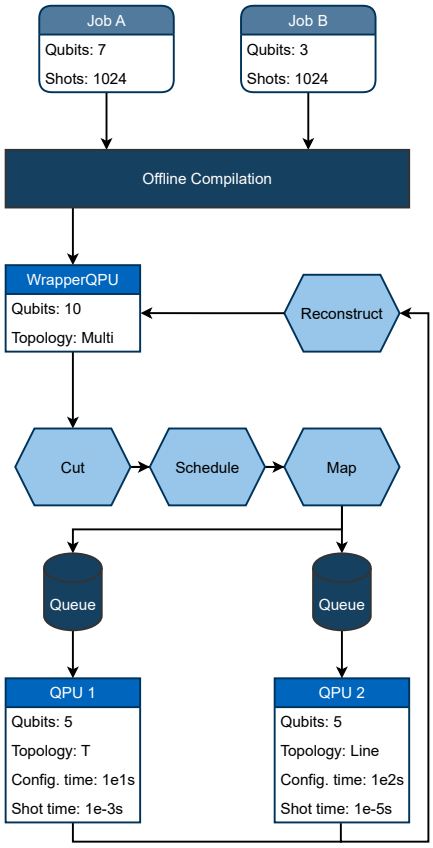


Fig. 2. Simplified overview of the intended workflow.

techniques to fit the available hardware and keeps track of execution for reconstruction during postprocessing. The cutting component supports only gate-cutting for the moment. Only necessary cuts are selected based on the size of the QPUs; the optimality conditions to reduce the sampling overhead are not considered yet. In this scenario, multiple two-qubit gates could be cut simultaneously by one *black box cut* [22]. In certain cases, one *joint cut* suffices, which is advantageous compared to individual cuts. We will discuss the selection of the cuts in Section IV-B. For scheduling, MILQ solves a linear programming task, which we describe in Section V in more detail. The goal of the optimization is to minimize the overall execution time. Once the hardware is determined, a hardware-specific compilation procedure prepares the circuits for execution. After obtaining the results, MILQ reconstructs the measurement data and assigns it to the correct circuits.

A. Components

MILQ comprises three main components: a *QPU wrapper*, a *scheduler*, and a *compilation pipeline*. The *QPU wrapper* is an abstraction layer, imitating the behavior of a single QPU. This minimizes the necessary code changes when switching hardware providers. Still, MILQ can use the available hardware information when producing a schedule, and hardware of different modalities is supported.

We assume that there exists a modular *compilation pipeline*. Modularity provides the possibility of running compilation steps at different times. While we treat most of the compilation as a black box, MILQ assumes that compilation can be split up over several phases: *offline compilation*, which is hardware agnostic, and *online compilation*, which can use hardware information. This mimics the workflow of an HPC application, where jobs are fully compiled before submission. The knitting module in MILQ (based on [23]) requires n-qubit gates to be decomposed first. This can be achieved offline, using synthesis tools or the built-in functionality from Qiskit [21]. Hiding implementation details is one benefit of the abstraction layer. For example, hardware-specific optimizations and necessary modifications, such as mapping, are abstracted.

During the scheduling process, MILQ analyzes a batch of circuits and provides an optimal schedule based on a MILP (see Section V). The scheduler assumes that circuits already fit the available hardware. The system queries the backends for each circuit's estimated processing and setup times. We use dummy data based on the circuit depth as accurate data from the hardware interface is unavailable. The scheduler supports two modes of operation: *simplified* and *extended*. In the simplified variant, we assume that the setup times depend only on the current job independent of its predecessor, which relaxes the constraints resulting from Equation (1).

B. Cutting Considerations

There are three main criteria when selecting a cut: The resulting circuit sizes, classical communication, and sampling overhead. Some circuit-cutting techniques require classical communication, using gate [24] or state [25] teleportation techniques to improve sampling overhead. Based on the communication patterns, this requires some form of synchronization on the quantum devices. Such dependencies can be considered during the scheduling but drastically increase the complexity. Due to the technical infeasibility of real-time communication, we do not consider this scenario. Wire-cutting also introduces precedence relationships to the schedule and depends on preparation operations, which have yet to be widely available in hardware. Once this becomes more commonplace, including wire cuts will be valuable for scheduling as they allow the circuit execution to be *preempted* and continued later.

Cutting gates generates multiple circuit instances with slight differences, reflecting the components of the initial circuit. When reconstructing the original probability distribution, more Monte-Carlo samples from the created circuits are necessary, which induces sampling overhead. The number of samples scales with the number of cut gates, the type of gates, and the cutting technique. In the worst case, cutting n gates U_i will generate an overhead of $\prod_i^n \kappa_i^2$, where κ_i is the one-norm of the coefficients of the quasiprobability decomposition of the gate U_i [25]. Overhead can be reduced depending on the gate type and by cutting gates in parallel [26]. The selection of cuts can be implemented as a constraint set but would require knowledge about the gates. Due to the already high complexity

of the model and nonexistent implementations, we only encode sampling overhead as part of the execution time of circuits.

An additional consideration is increased noise by running circuits in parallel. Due to the various sources of noise, especially crosstalk, placing the circuits on neighboring qubits affects the overall quality. Even allowing for spacing and selecting the least noisy qubits increases noise [27]. When choosing the cuts, this is an additional criterion to acknowledge. The notion of noise could also be implemented as part of the scheduling but would require enforcing connectivity constraints. Also, this would need an estimate of which hardware is most suitable for a given circuit. From a hardware point of view, it might be possible to physically separate the qubits according to their circuit in some modalities. Such physical partitions could significantly lower the chance of unwanted interactions.

Our greedy cut selection targets circuit size. One limiting factor for running quantum experiments is the number of available qubits. By selecting cuts such that the circuits will be guaranteed to fit the hardware, we hope for the near-term usability of our tool.

V. Problem Statement

We formulate the scheduling of circuits as an (offline) MILP. The problem is a more constrained variant of the well-studied unrelated parallel machines problem discussed in Section III-A. Each job (quantum circuit) can be assigned to one machine (QPU) in this model. Depending on this machine, each job has a unique processing time. Each job has a machine- and sequence-dependent setup time. This mimics, for example, the reconfiguration of the arbitrary waveform generators (AWGs) between experiments. To model qubit numbers, each machine has a fixed capacity, which cannot be exceeded at any time. The notation is summarized in Table I. $\mathbb{M}$ and $T_{\max}$ have to be tuned following the magnitude of the input parameters p_{im} and s_{ijm}. One key difference compared to existing models is the relaxation of the succession constraints. Typically, each job has one predecessor and up to one successor. In our scenario, however, multiple jobs can run in parallel on one machine, meaning one job can have multiple successors. Hence, we use the following definition for the successor relation (cf. Table I):

$$y_{ijm} = 1 \iff c_i < c_j \wedge \nexists k \in J : c_i < c_k < b_j \wedge \gamma_{ijm} \wedge \gamma_{ikm} \tag{1}$$

Paraphrased, a job j is the successor of job i on machine m when no other job k was completed in between. To allow for more than one successor, we also relax the completion time constraint (C5) such that having multiple predecessors does not add a penalty by having to set up twice. With this, we implicitly assume that the setup time for multiple circuits can be combined. Otherwise, mapping all possible combinations of circuits to the MILP would exponentially increase the complexity of the problem.

A. MILP Formulation

The notation in Table I and problem formulation are derived from Al-harkan and Qamhan [11]. The optimization problem is formulated as follows:

$$\min(c_{\max}) \tag{OBJ}$$

Subject to:

$$c_j \leq c_{\max} \quad \forall j \in J \tag{C1}$$

$$c_0 = 0 \tag{C2}$$

$$\sum_{m \in M} x_{jm} = 1 \quad \forall j \in J \tag{C3}$$

$$\sum_{m \in M} z_{jmt} \leq 1 \quad \forall j \in J, \forall t \in T \tag{C4}$$

$$c_j \geq b_j + \sum_{m \in M} p_{jm} \cdot x_{jm} + \sum_{i \in J \cup \{0\}} \sum_{m \in M} s_{ijm} \cdot y_{ijm} \quad \forall j \in J \tag{C5}$$

$$b_j \geq c_i + \mathbb{M} \cdot \left(\sum_{m \in M} y_{ijm} - 1 \right) \quad \forall j \in J, \forall i \in J \cup \{0\} \tag{C6}$$

$$\sum_{m \in M} \sum_{t \in T} z_{jmt} = c_j - b_j + 1 \quad \forall j \in J \tag{C7}$$

$$\sum_{t \in T} z_{jmt} \leq \mathbb{M} \cdot x_{jm} \quad \forall j \in J, \forall m \in M \tag{C8}$$

$$c_j \geq t \cdot \sum_{m \in M} z_{jmt} \quad \forall j \in J, \forall t \in T \tag{C9}$$

$$s_j \leq t \cdot \sum_{m \in M} z_{jmt} + \mathbb{M} \cdot \left(1 - \sum_{m \in M} z_{jmt} \right) \quad \forall j \in J, \forall t \in T \tag{C10}$$

$$\sum_{j \in J} q_j \cdot z_{jmt} \leq Q_m \quad \forall m \in M, \forall t \in T \tag{C11}$$

$$1 \leq \sum_{i \in J \cup \{0\}} \sum_{m \in M} y_{ijm} \quad \forall j \in J \tag{C12}$$

$$\mathbb{M} \cdot x_{jm} \geq \sum_{i \in J \cup \{0\}} y_{ijm} \quad \forall j \in J, \forall m \in M \tag{C13}$$

$$\mathbb{M} \cdot x_{jm} \geq \sum_{i \in J \cup \{0\}} y_{jim} \quad \forall j \in J, \forall m \in M \tag{C14}$$

$$z_{jm0} = y_{0jm} \quad \forall j \in J \forall k \in M \tag{C15}$$

$$\mathbb{M} \cdot \alpha_{ij} \geq b_j - c_i \quad \forall i, j \in J, i \neq j \tag{C16}$$

$$\mathbb{M} \cdot \beta_{ij} \geq c_j - c_i \quad \forall i, j \in J, i \neq j \tag{C17}$$

$$\gamma_{ijm} \geq x_{im} + x_{jm} - 1 \quad \forall i, j \in J, i \neq j, \forall m \in M \tag{C18}$$

$$\delta_{ijkm} \geq \alpha_{kj} + \beta_{ij} + \gamma_{ijm} + \gamma_{ikm} - 3 \quad \forall i, j, k \in J, i \neq j, \forall m \in M \tag{C19}$$

$$y_{ijm} \geq \alpha_{ij} + (1 - \sum_{k \in J} \delta_{ijkm}) + \gamma_{ijm} - 2 \quad \forall i, j \in J, \forall m \in M \tag{C20}$$

The overall goal, formulated in the objective function (OBJ),

TABLE I
MILP NOTATION

Metavariables	
$\mathbb{M}$	A big number
$T_{\max}$	A large number of time slots
0	Dummy job
Input Parameters	
J	Set of jobs (circuits)
M	Set of machines (QPUs)
p_{im}	Processing time of job i on machine m
s_{ijm}	Setup time of job j after job i on machine m
q_i	Required resources (qubits) of job i
Q_m	Available resources (qubits) of machine m
Indices	
i, j, k	Index of jobs, $i, j, k \in J$
m	Index of machines, $m \in M$
t	Timesteps, $t \in T = \{0, 1, 2, \ldots, \mathrm{T}_{\max}\}$
Binary Decision Variables	
x_{im}	1 if job i is scheduled on machine m
y_{ijm}	1 if job j is is a successor of job i on machine m
z_{imt}	1 if job i is scheduled on machine m at timestep t
Binary Helper Variables	
α_{ij}	1 if job i completes before job j starts
β_{ij}	1 if job i completes before job j completes
γ_{ijm}	1 if jobs i, j are scheduled on machine m
δ_{ijkm}	1 if job k completes after i completes but before j starts on m
Real Decision Variables	
c_j	Completion time of job j
$c_{\max}$	Makespan
b_j	Start time of job j

is to minimize the makespan. The constraints (C1) bound it on the maximal completion time of any job. Constraints (C3) and (C4) ensure that a job is executed on one machine at a time. (C5) calculate the completion time, and (C6) ensures that all predecessors of a job are done before starting it. Constraints (C7) ensure entire processing of a job; (C8) align the time-dependent execution variables such that they match the chosen machine. (C9) and (C10) fix the execution between start end completion. (C11) constrain the resource usage. The previous constraints could easily be adapted to time-dependent resource availability to model erroneous qubits or partial recalibration procedures. The constraints (C12)–(C19) ensure the validity of the successor relationship. Each regular job has a predecessor (C12), which is on the same machine ((C13) and (C14)). Jobs running at $t = 0$ are constrained to use the dummy job with (C15). (C16)–(C19) set up the helper variables, and the successor is finally set in the constraint (C20).

B. Example Assignment

As a simple example, we look at integer processing and setup times. This drastically reduces the number of necessary time-step variables. In this example, we revisit the setting in Fig. 2 for the circuits and QPUs. Initially, circuit *A* does not fit on any available device. By performing a greedy cut on *A*, MILQ generates four five-qubit jobs and four two-qubit jobs. Together with the three-qubit job *B*, these jobs are subsequently distributed across two five-qubit devices.

We generate synthetic processing times by randomly applying uniformly sampled variations to the circuit size. In reality, the difference in execution time between single-circuit executions is negligible. Still, this can accumulate significantly when considering the high number of shots in NISQ experiments. Setup times are generated pairwise, depending on the size of both circuits. In reality, MILQ could query this information during scheduling if the backends can estimate this accurately. Also, the setup time would depend on all combinations between predecessors and successors. For this example, we use the maximum of the possible setup times.

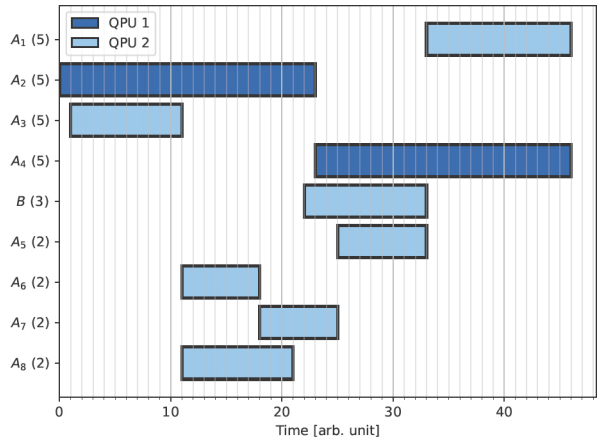


Fig. 3. Schedule of the sample problem obtained by the *simple* schedule.

The resulting MILP comprises 3188 (1208) variables and 3272 (1355) constraints for the *extended* (*simple*) algorithm. We run the example using the Gurobi [28] solver on a single 80-way Intel Ice Lake node. The resulting *simple* schedule is depicted in Fig. 3 and takes approximately 4 minutes to be generated. Due to resource limitations, we stop the *extended* model when the gap between the upper bound and the incumbent falls below 20 %, which still takes 8.6 hours. As parameters, we choose $\mathbb{M} = 1000$ and $T_{\max} = 64$.

VI. RESULTS

To validate MILQ, we benchmark multiple batches of random circuits. We compare the resulting schedule of MILQ with a baseline implementation of a simplified algorithm. The system under test is the *scheduling* component of MILQ. Every other component is fixed, especially the circuit knitting and mapping components. Our primary focus lies on the overall makespan, but we also provide insight into the real-time performance of the algorithms.

A. Hardware Analysis

Unfortunately, setup and processing times are often not directly available from vendors, even less so preliminary estimates of those numbers for a given job. Hence, we fall back to using dummy values. We argue, however, that modeling setup and processing times as machine- and job-specific and, in the first case, additionally as sequence-dependent values is a reasonable choice. On the one hand, gate times between different hardware implementations can vary by orders of magnitude. For instance, gate times on superconducting platforms can range from 10 nanoseconds up to microseconds [3].

The processing time of a circuit mainly depends on its depth and the number of shots. On the other hand, we justify sequence-dependent setup times by platform-specific hardware configurations. For instance, platforms utilizing neutral atoms can reconfigure the layout of atom traps as part of their experimental setup. This reconfiguration allows for three-dimensional topologies and scales according to the number of layers in the mapping [29]. Although current commercial compilers do not yet support such functionalities, it is a plausible feature for the future, making it an ideal scenario for testing MILQ.

B. Benchmarks

As a benchmark, we look at randomly selected circuits of various sizes using the MQT Bench [30] tool. We generate ten batches of seven circuits with the maximum size restricted to the largest available device. This mimics the situation after resizing the circuits. From the MQT Bench, we select the "random" option with optimization level zero. We compare two scenarios, one with the same configuration as in Section V-B (two QPUs with capacity 5) and the second scenario with three QPUs with the sizes (5, 6, 20). For both scenarios, the two models *simple* and *extended* are evaluated against a *baseline* algorithm (see Section VI-C). We randomly generate processing and sequence-dependent setup times in both scenarios based on the hardware estimations for superconducting platforms. Additionally, we run a set of trials with real-valued times and an independent set of integer-valued times. The simplified model assumes only machine-dependent setup times, which we generate by taking the maximum $s_{jm} = \max_{i \in J} s_{ijm} \forall j \in J$. After generating the schedules, we recalculate the makespan based on the original setup times, using the successor relation from Equation (1).

C. Baseline

As a baseline algorithm, we use an adapted version of first-fit decreasing bin packing [31]. This uses the simplified assumption that processing and setup times are independent of jobs and machines and are equally long. Each bin represents a QPU, and copies thereof model the simplified succession of jobs. Algorithm 1 summarizes the procedure; a copy of all instances is added instead of opening a single bin to ensure equal distribution over all devices. This provides a naive approach while still allowing multithreading. Compared to sequential schedules for each device, this is already an improvement.

Input : $J :=$ set of jobs, $B :=$ list of bins
Output: Bins filled with jobs
$J' \leftarrow$ `sorted(`*J, qubit_count descending*`)`;
$open \leftarrow B$;
$closed \leftarrow \emptyset$;
foreach $job \in J'$ **do**
  $bin \leftarrow$ `FindFirstFitting(`*job, open*`)`;
  if bin *is none* **then**
    $new \leftarrow B$;
    $bin \leftarrow$ `FindFirstFitting(`*job, new*`)`;
    extend $open$ with new;
  end
  add job to bin;
  if bin *is full* **then**
    remove bin from $open$;
    add bin to $closed$;
  end
end
add all $open$ to $closed$;
return $closed$;

Algorithm 1: Scheduling with first-fit decreasing bin packing.

D. Simulation Results

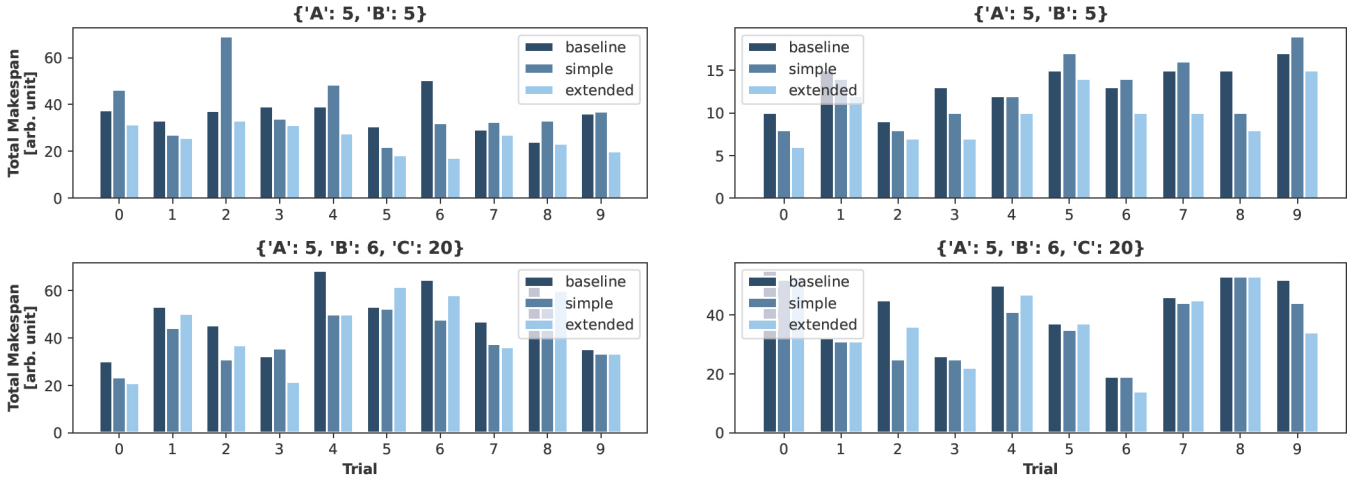
Fig. 4 shows the results of the makespan optimization. The full data is available in the code repository.

The *extended* model consistently outperforms the *simple* and *baseline* algorithms. The difference is more noticeable when we can parallelize multiple circuits. In the setting with two QPUs, circuits are similarly sized, and the makespan can be reduced by around $25\,\%$ by the *extended* model for real and integer inputs. Due to the size imbalance, the second setting has potentially sequential circuits on the largest device, reducing the gain to $12\,\%$ on average; for the real-valued input, the *extended* model performs slightly better. This also explains the outliers for the *simple* model, where the setup times are potentially assumed to be worse for one device, which is then avoided entirely. As a result, sequential execution decreases the performance, but a reduction of $11\,\%$ on average can still be achieved. In the worst-case setting (two devices with real-valued inputs), the baseline is faster than the *simple* approach.

Besides usability, the time to solution is also an essential factor. Due to the exponential overhead, the *simple* model is roughly four magnitudes slower, and the *extended* model is six magnitudes slower than the baseline. Especially in the desired scenarios with considerable parallelization potential, the *extended* model needs to evaluate multiple solutions. A bottleneck is the configuration and invocation of Gurobi, which we call through a Python interface using the default parameters. An approximation would likely suffice in a production environment, which we plan to implement in future iterations of MILQ.

VII. Future Work

The current state of MILQ primarily serves as a proof of concept rather than a full-fledged implementation. There are several potential avenues for improvement. Initially, the scheduling functionality is limited to a batched or offline environment. Real systems with continuous job submissions regularly trigger rescheduling based on a previous schedule. Such updates, as well as considering other criteria like preemption and priority, are necessary improvements. Integrating circuit knitting with the scheduler, rather than keeping them separate components, could enhance efficiency, especially in optimizing



(a) Results for real-valued s_{jm} and p_{jm}.

(b) Results for integer-valued s_{jm} and p_{jm}

Fig. 4. Makespan results for the three configurations *baseline*, *simple* and *extended* in two different settings.

cutting decisions for available hardware. The exact solving of MILP poses limitations when handling larger problems, suggesting the potential integration of heuristics for quicker solutions.

Expanding MILQ's scope beyond isolated QPUs is a future goal, with plans for interprocess communication, exemplified by upcoming technologies like the IBM Flamingo chip [32]. To accommodate such advancements, substituting the knitting module with a distribution component becomes necessary. Additionally, as hardware sizes increase, MILQ could schedule (partial) operations on error-corrected qubits instead of entire circuits. However, it is important to note that within MILQ's current scope, there is no notion of logical circuits that cannot be cut or distributed over multiple devices.

MILQ's optimization ideally relies on accurate processing and setup time estimates. Unfortunately, such precise data is largely unavailable for most systems. A standardized interface for QPUs would greatly aid in leveraging real-time data to influence optimal scheduling decisions. This interface could also provide live fidelity information, which could be used during the mapping and cutting steps.

VIII. Conclusion

In this article, we present MILQ, a software tool that can address multiple quantum hardware backends at the same time. It automatically distributes batches of circuits over the available hardware, which are not necessarily the same type. We formulate a MILP to solve the scheduling problem, which arises when the circuits are resized to fit the hardware. We prioritize minimizing the overall execution time and, therefore, emphasize hardware utilization. In a set of benchmarks, we show an average makespan improvement of $20\,\%$ compared to a baseline algorithm. MILQ works as an end-to-end tool but can easily be integrated into existing infrastructures.

Acknowledgment

The research is part of the Munich Quantum Valley (MQV), which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus. Moreover, this project is also supported by the Federal Ministry for Economic Affairs and Climate Action on the basis of a decision by the German Bundestag through project QuaST, as well as by the Bavarian Ministry of Economic Affairs, Regional Development and Energy with funds from the High-tech Agenda Bayern. Further funding comes from the German Federal Ministry of Education and Research (BMBF) through the MUNIQC-SC project. The authors would like to thank Christian Ufrecht and Daniel Scherer for their circuit cutting demonstration and the Quantum Computing group, especially Peter Eder, for their valuable feedback.

References

[1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Santa Fe, NM, USA: IEEE, 1994, pp. 124–134. [Online]. Available: https://doi.org/10.1109/SFCS.1994.365700

[2] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Phys. Rev. A*, vol. 86, p. 032324, Sep 2012. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.86.032324

[3] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020. [Online]. Available: https://doi.org/10.1146/annurev-conmatphys-031119-050605

[4] X. Wu, X. Liang, Y. Tian, F. Yang, C. Chen, Y.-C. Liu, M. K. Tey, and L. You, "A concise review of rydberg atom based quantum computation and quantum simulation*," *Chinese Physics B*, vol. 30, no. 2, p. 020305, feb 2021. [Online]. Available: https://dx.doi.org/10.1088/1674-1056/abd76f

[5] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, p. 021314, 05 2019. [Online]. Available: https://doi.org/10.1063/1.5088164

[6] M. Bandic, L. Prielinger, J. Nüßlein, A. Ovide, S. Rodrigo, S. Abadal, H. van Someren, G. Vardoyan, E. Alarcon, C. G. Almudever, and S. Feld, "Mapping quantum circuits to modular architectures with qubo," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 01, 2023, pp. 790–801. [Online]. Available: https://doi.org/10.48550/arXiv.2305.06687

[7] D. Bhoumik, R. Majumdar, A. Saha, and S. Sur-Kolay, "Distributed scheduling of quantum circuits with noise and time optimization," 2023, unpublished. [Online]. Available: https://doi.org/10.48550/arXiv.2309.06005

[8] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II*, ser. Annals of Discrete Mathematics, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, vol. 5, pp. 287–326. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016750600870356X

[9] L. Fanjul-Peyro and R. Ruiz, "Size-reduction heuristics for the unrelated parallel machines scheduling problem," *Comput. Oper. Res.*, vol. 38, no. 1, p. 301–309, jan 2011. [Online]. Available: https://doi.org/10.1016/j.cor.2010.05.005

[10] F. J. Rodriguez, M. Lozano, C. Blum, and C. García-Martínez, "An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem," *Comput. Oper. Res.*, vol. 40, no. 7, p. 1829–1841, jul 2013. [Online]. Available: https://doi.org/10.1016/j.cor.2013.01.018

[11] I. M. Al-harkan and A. A. Qamhan, "Optimize unrelated parallel machines scheduling problems with multiple limited additional resources, sequence-dependent setup times and release date constraints," *IEEE Access*, vol. 7, pp. 171 533–171 547, 2019. [Online]. Available: https://doi.org/10.1109/ACCESS.2019.2955975

[12] J. K. Lenstra, D. B. Shmoys, and É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical Programming*, vol. 46, no. 1, pp. 259–271, Jan 1990. [Online]. Available: https://doi.org/10.1007/BF01585745

[13] C. Glass, C. Potts, and P. Shade, "Unrelated parallel machine scheduling using local search," *Mathematical and Computer Modelling*, vol. 20, no. 2, pp. 41–52, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0895717794902054

[14] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. [Online]. Available: https://doi.org/10.1007/10968987_3

[15] T. S. Humble, A. McCaskey, D. I. Lyakh, M. Gowrishankar, A. Frisch, and T. Monz, "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 5, p. 15–23, sep 2021. [Online]. Available: https://doi.org/10.1109/MM.2021.3099140

[16] QIR Alliance, *QIR Specification*, 2021, also see https://qir-alliance.org. [Online]. Available: https://github.com/qir-alliance/qir-spec

[17] T. Peng, A. W. Harrow, M. Ozols, and X. Wu, "Simulating large quantum circuits on a small quantum computer," *Phys. Rev. Lett.*, vol. 125, p. 150504, Oct 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.125.150504

[18] K. Mitarai and K. Fujii, "Constructing a virtual two-qubit gate by sampling single-qubit operations," *New Journal of Physics*, vol. 23, no. 2, p. 023021, feb 2021. [Online]. Available: https://dx.doi.org/10.1088/1367-2630/abd7bc

[19] C. Ufrecht, M. Periyasamy, S. Rietsch, D. D. Scherer, A. Plinge, and C. Mutschler, "Cutting multi-control quantum gates with ZX calculus," *Quantum*, vol. 7, p. 1147, Oct. 2023. [Online]. Available: https://doi.org/10.22331/q-2023-10-23-1147

[20] H. Harada, K. Wada, and N. Yamamoto, "Optimal parallel wire cutting without ancilla qubits," 2023, unpublished. [Online]. Available: https://doi.org/10.48550/arXiv.2303.07340

[21] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.2573505

[22] L. Schmitt, C. Piveteau, and D. Sutter, "Cutting circuits with multiple two-qubit unitaries," 2023, unpublished. [Online]. Available: https://doi.org/10.48550/arXiv.2312.11638

[23] L. Bello, A. M. Brańczyk, S. Bravyi, A. Carrera Vazquez, A. Eddins, D. J. Egger, B. Fuller, J. Gacon, J. R. Garrison, J. R. Glick, T. P. Gujarati, I. Hamamura, A. I. Hasan, T. Imamichi, C. Johnson, I. Liepuoniute, O. Lockwood, M. Motta, C. D. Pemmaraju, P. Rivero, M. Rossmannek, T. L. Scholten, S. Seelam, I. Sitdikov, D. Subramanian, W. Tang, and S. Woerner, "Circuit Knitting Toolbox," 2023. [Online]. Available: https://zenodo.org/doi/10.5281/zenodo.7987996

[24] C. Piveteau and D. Sutter, "Circuit knitting with classical communication," *IEEE Transactions on Information Theory*, pp. 1–1, 2023. [Online]. Available: https://doi.org/10.1109/TIT.2023.3310797

[25] L. Brenner, C. Piveteau, and D. Sutter, "Optimal wire cutting with classical communication," 2023, unpublished. [Online]. Available: https://doi.org/10.48550/arXiv.2302.03366

[26] C. Ufrecht, L. S. Herzog, D. D. Scherer, M. Periyasamy, S. Rietsch, A. Plinge, and C. Mutschler, "Optimal joint cutting of two-qubit rotation gates," 2023, unpublished. [Online]. Available: https://doi.org/10.48550/arXiv.2312.09679

[27] J. S. Baker, G. Park, K. Yu, A. Ghukasyan, O. Goktas, and S. K. Radha, "Massively parallel hybrid quantum-classical machine learning for kernelized time-series classification," 2023, unpublished. [Online]. Available: https://doi.org/10.48550/arXiv.2305.05881

[28] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[29] D. Barredo, V. Lienhard, S. de Léséleuc, T. Lahaye, and A. Browaeys, "Synthetic three-dimensional atomic structures assembled atom by atom," *Nature*, vol. 561, no. 7721, pp. 79–82, Sep 2018. [Online]. Available: https://doi.org/10.1038/s41586-018-0450-2

[30] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing," *Quantum*, vol. 7, p. 1062, Jul. 2023. [Online]. Available: https://doi.org/10.22331/q-2023-07-20-1062

[31] B. S. Baker, "A new proof for the first-fit decreasing bin-packing algorithm," *Journal of Algorithms*, vol. 6, no. 1, pp. 49–70, 1985. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0196677485900185

[32] J. Gambetta. (2022) Expanding the ibm quantum roadmap to anticipate the future of quantum-centric supercomputing. [Online]. Available: https://research.ibm.com/blog/ibm-quantum-roadmap-2025