Array 17 (2023) 100282

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array

When quantum annealing meets multitasking: Potentials, challenges and opportunities

Tian Huang [a], Yongxin Zhu [b,c], Rick Siow Mong Goh [a], Tao Luo [a,*]

[a] *Institute of High Performance Computing, and Agency for Science, Technology and Research, 1 Fusionopolis Way, #16-16 Connexis, Sinapore, 138632, Singapore*
[b] *Shanghai Advanced Research Institute, Chinese Academy of Sciences, Shanghai, China*
[c] *University of Chinese Academy of Sciences, Beijing, China*

ARTICLE INFO

Dataset link: https://github.com/ianmalcolm/QAMTSimulator

Keywords:
Quantum annealing
Multitasking

ABSTRACT

Quantum computers have provided a promising tool for tackling NP hard problems. However, most of the existing work on quantum annealers assumes exclusive access to all resources available in a quantum annealer. This is not resource efficient if a task consumes only a small part of an annealer and leaves the rest wasted. We ask if we can run multiple tasks in parallel or concurrently on an annealer, just like the multitasking capability of a classical general-purpose processor. By far, multitasking is not natively supported by any of the existing annealers. In this paper, we explore Multitasking in Quantum Annealer (QAMT) by identifying the parallelism in a quantum annealer from the aspect of space and time. Based on commercialised quantum annealers from D-Wave, we propose a realisation scheme for QAMT, which packs multiple tasks into a quantum machine instruction (QMI) and uses predefined sampling time to emulate task preemption. We enumerate a few scheduling algorithms that match well with QAMT and discuss the challenges in QAMT. To demonstrate the potential of QAMT, we simulate a quantum annealing system, implement a demo QAMT scheduling algorithm, and evaluate the algorithm. Experimental results suggest that there is great potential in multitasking in quantum annealing.

1. Introduction

In the modern computing era, general-purpose processors, e.g., CPU and GPU, mostly have multitasking capability. That is, it concurrently executes multiple tasks over a certain period of time. This can be realised by either quickly switching between multiple tasks (time-sharing) or running multiple tasks simultaneously (space-sharing) on a single processor. Multitasking capability effectively improves processor resource utilisation.

Quantum annealing is a powerful computing paradigm [1], making use of adiabatic theory and quantum mechanics to solve combinatorial optimisation problems. Most of the existing work on existing quantum annealers assumes that a task occupies the whole annealer exclusively. If you have multiple tasks, your tasks enter a queue and are executed sequentially. A task does not necessarily make good use of all resources in a quantum annealer. This sequential execution manner causes resource utilisation problem.

We explore the feasibility of Multitasking in Quantum Annealers (QAMT). This would improve the resource utilisation of quantum annealers. The capacity of quantum computers has been scaled up several thousand times in the past two decades. We believe that the need for QAMT also increases along with the capacity of a quantum annealer. Time-critical applications such as automation control and autonomous driving would benefit from QAMT, as the execution time of tasks is known. Multitasking would also enable virtualisation of quantum annealers, allowing multiple users to share a quantum annealer without knowing the existence of other users. The sharing of quantum annealers would reduce the cost of usage and spark new applications and opportunities.

Multitasking is not natively supported by existing commercialised quantum annealing systems. For example, in a D-Wave quantum annealer, an instruction always resets the annealer before executing a task, such that the task always occupies all resources in the annealer. There is no scheduling or preemption of tasks according to their priority in timing. The feasibility of multitasking on existing annealers has not been explored. To the best of our knowledge, Pelofske et al. [2] is the only effort that exploits parallelism in quantum annealing. Instantiates multiple copies of a task so that the annealer can produce multiple

* Corresponding author.
E-mail addresses: huangtian44@hotmail.com (T. Huang), zhuyongxin@sari.ac.cn (Y. Zhu), gohsm@ihpc.a-star.edu.sg (R.S.M. Goh), leto.luo@gmail.com (T. Luo).

https://doi.org/10.1016/j.array.2023.100282
Received 18 October 2022; Received in revised form 16 January 2023; Accepted 20 January 2023
Available online 23 January 2023

samples at a time and complete the task more quickly. This method improves resource utilisation but is restricted to a single task.

In this paper, we explore multitasking in Quantum Annealer by identifying task parallelism in a quantum annealer from the aspect of space and time. We propose an instruction-based scheme, which emulates multitasking. To our best knowledge, we are the first to explore the potential of multitasking in quantum annealer. Our contributions can be summarised as follows.

- We identify task parallelism in a quantum annealer from the aspects of space and time. We adapt the concept of task parallelism and task preemption for use in quantum annealing systems.
- We propose an instruction-based scheme to emulate multitasking. Several tasks are packed into one instruction to be executed in parallel. We use predefined sampling periods to emulate task preemption, which is an integral part of dynamic scheduling.
- To facilitate the research and development of the QAMT scheduling algorithm, we develop a simulator. With the simulator, we evaluated the performance of a demo scheduling algorithm on a synthetic dataset. The results suggest great potential of QAMT.

The rest of the paper is organised as follows. Section 2 describes the position of this work in the history of the literature. Section 3 defines the model of a task, a processor, and the execution of the task on the processor. Section 4 defines a multitasking model based on quantum annealer from a hardware point of view. Section 5 formalises the QAMT scheduling problem and reviews existing algorithms that can be adapted for space allocation and time scheduling in QAMT. It also identifies some possible challenges in QAMT scheduling. Section 6 proposes a simulator and performs experiments to demonstrate the potential of QAMT. Section 7 draws a conclusion and discusses the possibility of performing multitasking on other quantum computers.

2. Related work

Classical general-purpose processors, e.g. CPUs, can do multitasking very well. Task parallelism is a well-established research field based on modern computing architecture design [3,4] and scheduling algorithms [5,6]. This paper does not propose new classical computing architectures or new scheduling algorithms for classical computing architectures. Our focus is on realising task parallelism on an annealing-based quantum computer.

The typical procedure for solving a problem instance (or a task) on a quantum annealer is to first represent the instance in the quadratic unconstrained binary optimisation (QUBO) [7] format. This is usually done on the user side. The QUBO problem instance travels through the local interface or the Internet and reaches the input queue of a quantum annealer. The quantum annealer processes the task queue sequentially. When the task is completed, the user fetches the corresponding result.

This exclusive work mode makes sense, since quantum computing is still in its early research and development stage. Scientists want full control over the device for experiments and want to make sure that their experiments are not affected by other users. However, there are motivations to share resources between multiple tasks. First, as the manufacturing technique advances, quantum annealers are getting bigger [8]. A problem instance is less likely to make good use of all computing resources. Sharing a quantum device to reduce the cost of usage could be the main motivation of industrial and individual users. Second, QAMT matches very well with time-critical applications like automation control and autonomous control, which usually require a set of tasks with a known execution time.

Parallel quantum annealing [2] addressed the issue of low resource utilisation in quantum annealing. The authors instantiate multiple copies of a problem instance so that an annealer produces more samples at a time and completes the task faster. Parallel quantum annealing has found its utility in a few applications [9,10]. Niu and Todri-Sanial [11] is a similar work based on gate-based models, which implements multiple quantum circuits on one quantum device. Both Pelofske et al. [2],Niu and Todri-Sanial [11] exploit the parallelism within a task, but do not cover the switching and scheduling between tasks.

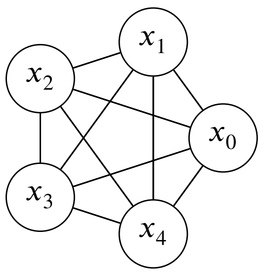


Fig. 1. K_5 complete graph as a QUBO problem.

There is also literature [12,13], which sees the quantum computing system as an accelerator in a modern High-Performance Computing (HPC) infrastructure. The focus is on the design of architectures and programming models needed to integrate near-term quantum computers with supercomputers, and the workflows for potential applications. In-device task-level parallelism is not their focus.

In this paper, we realise multitasking on a quantum annealer. We will point out that the existing implementation is not explicitly designed to share resources among multiple tasks at the same time. We explore the possibility of carrying out multiple tasks on a quantum annealer in a time-sharing and space-sharing manner. We demonstrate the potential of QAMT through simulation-based experiments.

3. Model definition

In this section, we define the task model, processor model, and execution model for quantum annealing.

3.1. Task model

A quantum annealer solves quadratic unconstrained binary optimisation (QUBO) problems [14]. A task to quantum annealer includes the problem itself and a few parameters of the annealer. A general QUBO problem can be expressed as a cost or energy function, as shown in Eq. (1).

$$\texttt{Minimise}\ y = x^t Q x \tag{1}$$

x is a binary column vector of length n. Q is a $n \times n$ upper triangle matrix with real value entries. y is a scalar that represents the energy or the cost. Q is also called the QUBO matrix. The elements on the diagonal are the weights of the linear terms. The elements off the diagonal are the weights of the quadratic terms. We can also treat the QUBO matrix as an adjacency matrix of a graph. Here, we plot a five-variable QUBO matrix as shown in Fig. 1.

The QUBO matrix is the representation of the problem and is passed to a quantum annealer to be solved. Apart from the most important QUBO matrix, a task also includes a few more parameters to solve the problem on the quantum device. A common but incomplete list of parameters includes *embedding*, *number of samples*, global annealing control parameters such as *annealing time*, annealing schedule, and per-resource annealing control parameters, such as offsets and initial states.

The "embedding" specifies the mapping between the logical variables x of the QUBO problem and the physical resource units in the quantum device. Through embedding, we know the number of resources required for the task. Quantum annealers behave as samplers that generate independent configurations from Gibbs distributions [15]. The "number of samples" literally means the number of samples required by the task. According to the extreme value theory, more samples translate into a better chance of getting promising solutions.

The amount of time required to generate each sample is determined by “Annealing time”. In addition, there are a few other parameters that control the annealing process that generate a sample; for example, the annealing schedule and the delay time (for cooling) are global settings shared by all resources within a device. Some parameters, such as annealing offsets and initial states, are on a per-resource basis. Although the parameters are not designed exclusively for multitasking, some of them can be revised for our purposes.

Given an annealing task, we know exactly the amount of resources and the execution time required. This is to be contrasted with a programming in a classical computing diagram, whose execution path and memory requirement could vary along with different inputs.

3.2. Processor model

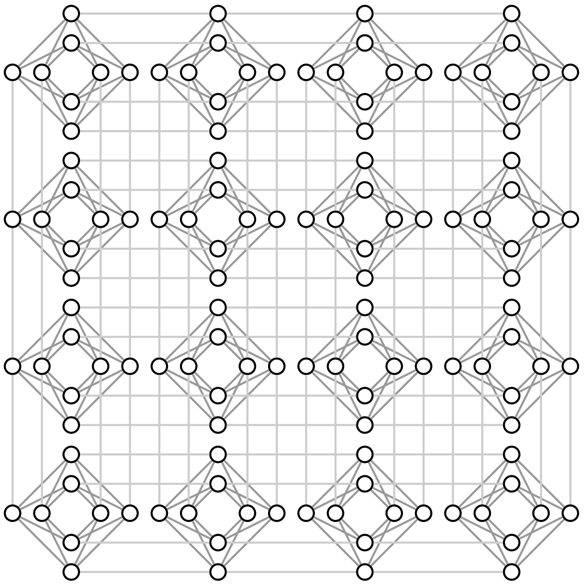
The quantum annealer manufactured by D-Wave is a lattice of interconnected qubits [16]. Qubit is a quantum version of bit compared with its classical counterpart that we find in a CMOS circuit, although its working principle is quite different. Fig. 2(a) shows the topology of a part of the Chimera architecture that D-Wave released in 2017. Each node in the figure is a qubit, which is made of a tiny metal coil and becomes a superconductor and exhibits quantum-mechanical effects when the temperature is below 9.2 kelvin. A qubit corresponds to a binary variable in Eq. (1). One can apply a tunable flux bias to each coil. The strength of a bias to a coil corresponds to a value on the diagonal of the QUBO matrix. If there is tunable inductive coupling between two coils, we say that the two qubits are connected. The strength of the coupling corresponds to the off-diagonal elements of the QUBO matrix. We refer to the qubits and couplers between them as *resources*.

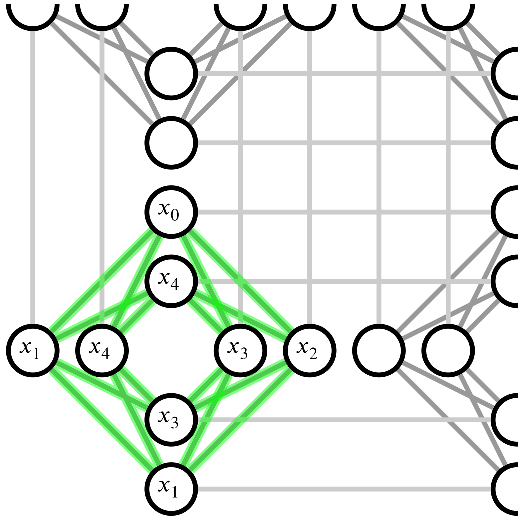
If we take a close look at the topology, we would find that the Chimera architecture consists of unit tiles. Each unit cell is a $K_{4,4}$ graph, i.e., a 4-by-4 complete bipartite graph,[1]. Chimera has 16 × 16 such cells. In Fig. 2(a) we only show a 4 × 4 array for simplicity. There are also connections between cells. Each qubit has two connections to the qubits of neighbouring cells, except for those on the edge of the array.

Fig. 2(b) shows how a complete graph in Fig. 1 can be embedded into this sparsely connected topology. Some variables, for example, x_1, are assigned to two physical qubits. Duplication ensures that all edges in the original problem can be represented by the device topology. The resulting QUBO matrix has eight qubits, which have three more qubits than that of the original problem. Considering that a pair of duplicated qubits could give different answers, we have to add penalty terms to the 8 × 8 QUBO matrix so that disagreement between duplicated qubits is discouraged. The involvement of the duplication is termed as *chain* technique by D-Wave. Without the chain technique, the Chimera architecture cannot even handle a complete graph with three vertices.

Although Chimera has 2048 qubits, the size of a problem that can be mapped onto the device is usually much smaller than 2048, due to the limited number of couplers on the device. With the chain technique, the largest complete graph that can be mapped onto the Chimera architecture is a K_8 complete graph. One cannot map a K_9 problem onto Chimera as there are not enough couplers[2] to represent all connections in the problem. Given the fact that many permutation-based combinatorial optimisation problems and machine learning problems are represented in densely connected graphs or even complete graphs, a single problem instance is not likely to make full use of all the computing resources in a quantum annealer.

The mismatch between the task topology and the processor incurs additional resource consumption. The mismatch is common in real-world applications. Graph embedding itself is a challenging NP-hard problem. Not all problems can be mapped onto the quantum device. The availability of feasible embedding is an integral part of quantum annealing multitasking.


(a) 4x4 unit cells



(b) Close-up of a unit cell

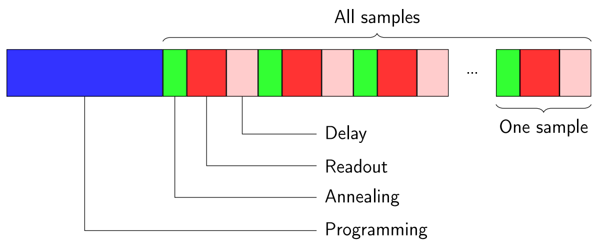
Fig. 2. Chimera architecture and an example of embedding.

Due to manufacturing imperfections, not all qubits and couplers are available to users. For example, Chimera has 2048 qubits and 6016 couplers in its blueprint, but at the time this paper is written, there are 2041 qubits and 5974 couplers available in the DW_2000Q_6 system, which employs the Chimera architecture. The missing qubits and couplers are distributed throughout the annealing device. We speculate that some resources are not in working condition and are masked and hidden from users. Device maintenance could alter the availability of resources as well. One has to consider defects in graph embedding.

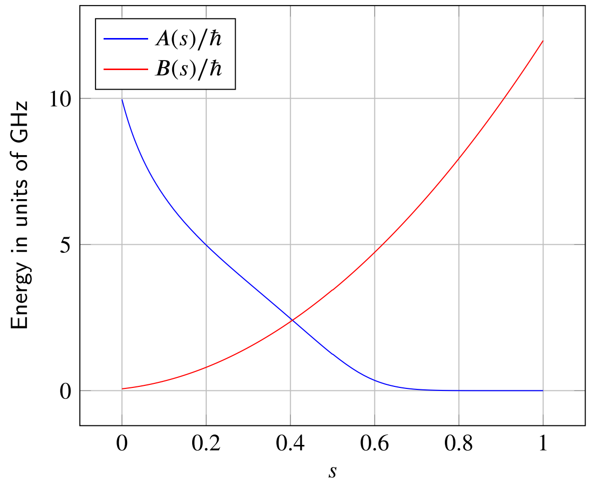
The computing resources in a classical general-purpose processor are usually quite diversified, such as various arithmetic operators for integers and floating-point numbers, logic operators, buffers, and data paths between. But the computing resource in an existing quantum annealer is just a network of qubits, which provides a premise for quantum mechanics to play its magic.

[1] A complete bipartite graph is a graph whose vertices can be divided into two subsets, such that no edge has both endpoints in the same subset, and every pair of vertices in the two subsets are connected with an edge.

[2] As we have demonstrated in Fig. 2(a), a qubits only has access to six couplers, not all 6016 couplers in the whole quantum device.



(a) D-Wave QPU operations



(b) Annealing schedule

Fig. 3. (a) The cycles in a Quantum Machine Instruction (QMI, defined by D-Wave QPU) (b) Annealing schedule for the D-Wave 2000Q QPU, showing energy changes as a function of scaled time. $\hbar$ is the planck constant.

3.3. Execution model

We take the quantum annealers from D-Wave as an example. When a user submits a task, it reaches the D-Wave system and raises a quantum machine instruction (QMI). Once the execution of the instruction starts, it cannot be interrupted. Fig. 3(a), shows a series of operations included in an instruction.

The D-Wave system first pre-processed the QUBO matrix and sent its values to the Digital Analog Converters (DAC) in the QPU, which will late drive the tunable bias and couplers in the annealing process. The blue cycle, called programming, is subdivided into a reset step that erases the previous data stored in all DACs and a step that configures some of the DACs. The digital circuit involved generates heat dissipation, which could affect the subsequent annealing process. A delay helps the QPU regain the working temperature for annealing. During the programming cycle, we cannot do annealing due to the heat dissipation. We refer to the mutual exclusion between programming cycle and annealing cycle as the *blocking* feature of programming cycle in the context of task scheduling.

Next, the QPU repeats the anneal-read cycle and generates samples. The number of samples to generate is specified by the "number of samples" in the task. Each cycle consists of an annealing process (green), a readout (red), and a delay (pink).

$$\mathcal{H}_{ising} = -\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x^{(i)}\right) + \frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i>j} J_{i,j}\hat{\sigma}_z^{(i)}\hat{\sigma}_z^{(j)}\right) \tag{2}$$

The annealing process of a quantum system is where the magic happens. The operator of the system, or Hamiltonian in Eq. (2), describes the time evolution of the annealing process. The first term encodes an equal superposition of all possible answers to the QUBO problem. The second term encodes the QUBO problem to be solved. In the annealing process, the quantum system transits slowly from the equal superposition to a superposition whose state of the lowest possible energy, a.k.a. the ground state, corresponds to the optimal solution to the problem. The schedule of the transition, a.k.a annealing schedule, is controlled by the functions $A(s)$ and $B(s)$ of time s, which are illustrated in Fig. 3(b). According to adiabatic theory, we expect the system to stay in a low-energy state at the end of the annealing process, allowing it to read off a low-energy solution to the problem. The readout generates heat dissipation as well. A subsequent delay helps the quantum system to regain its working temperature.

Apart from the default geometric annealing schedule demonstrated in Fig. 3(b), there are other annealing schedules, such as reversing [17, 18], pausing [19], quenching [20], which are all directly applicable to multitasking, if the tasks share the same annealing schedule.

The annealing schedule is a global setting. The time evolution of the qubits and couplers follows the same configuration. All anneal-read cycles in a QMI follow the same configuration. To use a different annealing schedule, one has to wait until the current QMI completes and start a new QMI with a different configuration. One can use annealing time to either squeeze or stretch the annealing schedule, to make it longer/slower, or shorter/quicker. The annealing time and delay are also global settings, which is similar to how the annealing schedule is shared by all the anneal-read cycles. If we want to share the resources in a device among multiple tasks, these tasks have to share the same annealing schedule, annealing time, annealing delay and other global settings. We say that such tasks are compatible with each other and they belong to a *class* of tasks.

A user can optionally offset the annealing schedule earlier or later by using the annealing offset parameter. One can also specify the initial state of the qubits. Parameters like initial state and annealing offset are, in their nature, on a per-qubit basis, meaning that qubits can have a different offset and initial state. The configuration takes effect from the beginning of a QMI to its end, and cannot be changed halfway.

The length of the programming cycle, indicated by the blue bar in Fig. 3(a), is problem specific. Empirically we find the length of the programming cycle is usually capped at about 12 ms. The length of one annealing process, indicated by a green bar, ranges from 2–2000 μs and is user-specified and independent of the problem. Either the programming cycle or the annealing cycles could dominate the total execution time of a QMI,[3] depending on the settings of the tasks.

The execution of a program in a classical general-purpose processor is complicated but under well control. The transition and branching of digits through various circuit paths are deterministic, observable, and predictable. But for quantum annealing, the computation is like dropping a ball into an opaque Galton board [21]. Once it enters the board, the movement of the ball is not observable and non-deterministic. One can only find the result when the ball hits the button of the board (corresponding to the readout of a sample).

4. Parallelism

In this section, we describe the potential parallelism in quantum annealers from the perspective of space sharing and time sharing.

[3] Readers with interests can find more detailed description of timing from the official website of D-Wave https://docs.dwavesys.com/docs/latest/c_qpu_timing.html

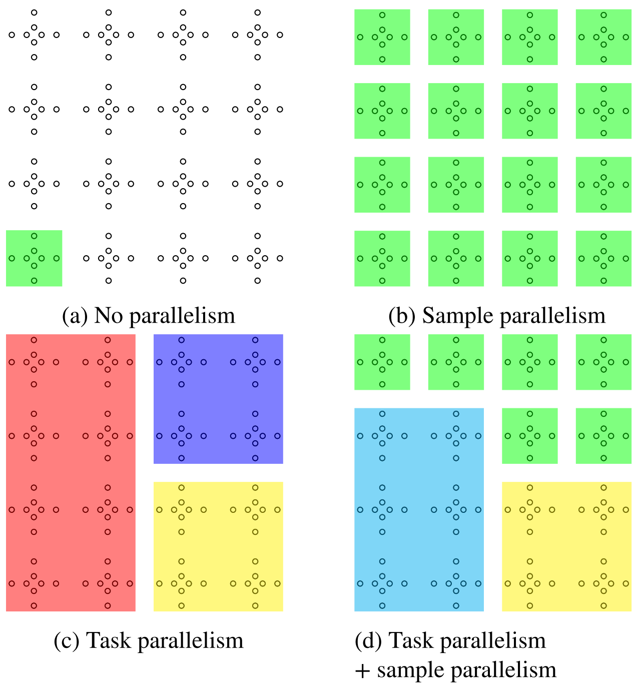


Fig. 4. Space sharing parallelism. These figures are based on the *resource map* of a 4×4 Chimera architecture. Connections between qubits are not shown for simplicity. We use coloured masks to represent different tasks. If a cell is covered by a mask, it is occupied by the corresponding task.

4.1. Space sharing

We first discuss parallelism from a space-sharing perspective. The Quantum Machine Instruction by D-Wave serves well for each task, by resetting all DACs in an annealer and giving control of all resources to a task, even if the task only requires a small number of resources. Fig. 4(a) shows an example. We refer to the array of unit cells as *resource map*. The green task only uses one unit cell and leaves the rest of the resources idle.

Sample Parallelism We can duplicate the green task multiple times to make full use of the resources, as shown in Fig. 4(b). This can be achieved by programming all unit cells using the same QUBO matrix in a single QMI. If the green task requires 256 samples, we can reduce the anneal-read cycle from 256 down to 16 by using duplication, since we can generate 16 samples in each cycle. The feasibility of this has been well discussed in [2]. We refer to this as *sample parallelism*.

Sample parallelism has its advantages. Since all duplicated instances come from the same task, these instances share almost the same parameters and are mostly compatible with each other. The only non-trivial effort is to embed multiple instances onto the device. The feasibility of duplicating a task is based on the assumption that the annealer employs a homogeneous topology globally so that the embedding can be easily applied to different parts of the annealer.

Task Parallelism We can also run multiple tasks simultaneously on a quantum annealer. This can be achieved by fitting the embeddings of multiple tasks into a resource map and programming the annealing device accordingly. The fitting of embeddings is a variation of the bin-packing problem, which will be discussed in Section 5.2. Fig. 4(c) shows an example of task parallelism. The three tasks are just about right to fit in the annealer, make full use of the resources, and receive samples simultaneously. This potentially allows a microsecond-level of real-time response to multiple tasks. The response is gradually optimised as the annealer generates more samples.

The hard limitation of task parallelism comes from the compatibility between tasks. As we have mentioned in Section 3, a configuration is taking effect across the entire QMI and over the entire QPU. If two tasks run in parallel, they must use the same annealing parameters that are shared globally. We observe that default parameter settings provided by the vendor of annealing systems (e.g. D-Wave) usually work well on various kinds of problem and thus lead to the chances of two tasks sharing the same annealing parameters [22–24]. For example, the annealing schedule is the critical setting of an annealing process. The geometric annealing schedule in Fig. 3(b) is sophisticated enough to handle various problems [25]. In addition, instances of the same problem are more likely to share the same annealing schedule. As the manufacturing technique advances, we believe that we will have more independent control over the annealing process for individual tasks.

Task Parallelism + Sample Parallelism We can have task parallelism and sample parallelism at the same time. This is useful twofold. First, some tasks could have higher priority over others. We can duplicate such tasks multiple times for expedition purposes. Second, sometimes an embedding comes with an irregular shape. Having such a task in the annealer results in an irregular-shaped idle space. One can use tasks with small and regular-shaped embedding to fill the irregular space and improve resource utilisation. Fig. 4(d) shows an example. The cyan task takes 2×3 unit cells and leaves a space of 1×2 unit cells. Neither the yellow nor the blue task fit into this area. We can use the green task to fill it.

4.2. Time sharing

We also discuss parallelism from the time-sharing perspective. The current implementation of the D-Wave annealing system does not support the suspension and resumption of tasks. A series of tasks are getting sequentially executed. A task has to wait until the ones before it are all finished, even if it comes with a higher priority. In Fig. 5, we assumes the length of a programming cycle takes 10 ms and one annealing cycle takes 2 ms. As shown in Fig. 5(a). Yellow, blue, and green tasks are available at 0 ms, 20 ms, and 40 ms. The green one has to wait until the previous tasks are complete.

Non preemptive We can reduce the response time by carrying out available tasks in parallel, in a non-preemptive manner. Response time refers to the period between the appearance and the completion of a task. Fig. 5(b) shows an example of non-preemptive task parallelism. Since the red and yellow tasks are both available at 0 ms, we can start the execution of these two tasks at the same time. The blue task is available at 20 ms but has to wait, since the red and yellow task is still running. After the first two tasks are complete, the blue and green tasks start in parallel and run accordingly.

Preemptive We can reduce wait time by starting a task as soon as it becomes available in the scheduling queue. Fig. 5(c) shows an example of preemptive task parallelism. The red and yellow tasks start in parallel, as is usually the case. When the blue task is available at 20 ms, we force a stop to the red and yellow tasks and perform device programming for the first three tasks. They run in parallel for another 10 ms, and then the green task kicks in and forces a stop. This time only the red task is affected since the yellow and blue tasks are complete.

Preemptive task parallelism + sample parallelism With the schedule that incorporates preemptive task parallelism and sample parallelism, we can bring earlier the finish time of tasks. Fig. 5(d) shows an example of preemptive task parallelism and sample parallelism. We start the red and yellow tasks and duplicate them to make full use of the annealer. The blue and green tasks can start immediately when they are available. The Average-case Execution Time (ACET) and Worst-case Execution Time (WCET) are reduced, compared with non-preemptive and preemptive scheduling.

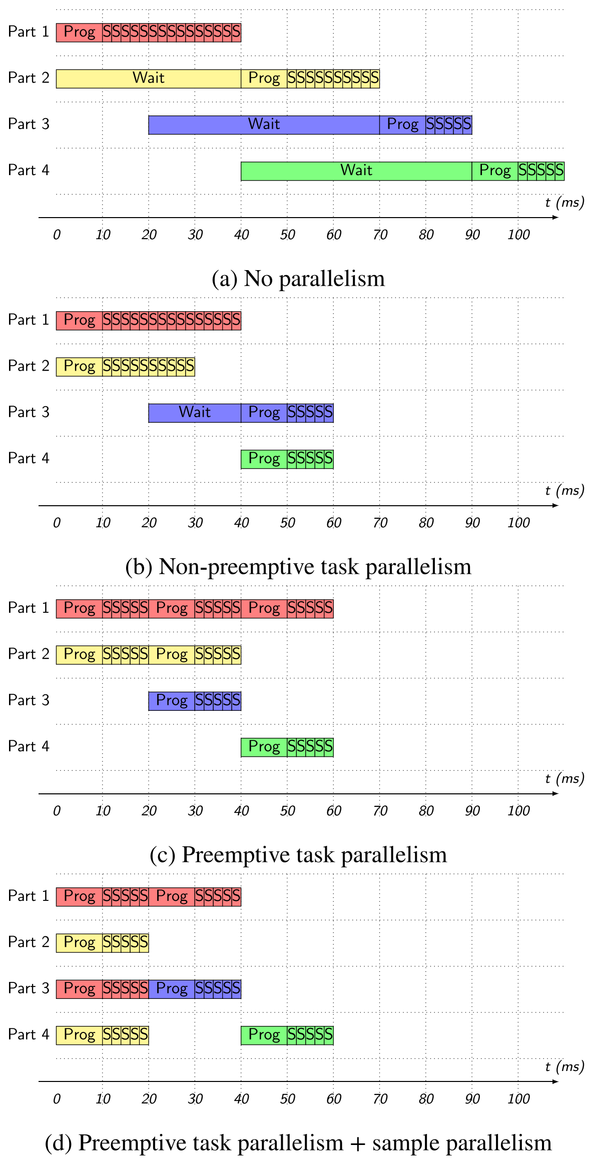


Fig. 5. Time sharing parallelism. Suppose We have a quantum annealer, whose resources are divided into four equal-sized partitions, parts 1 4. We have four tasks with equal-sized QUBO matrices. The number of samples required by the tasks is 15, 10, 5, and 5 respectively. The availability of the tasks are at 0 ms, 0 ms, 20 ms and 40 ms, respectively.

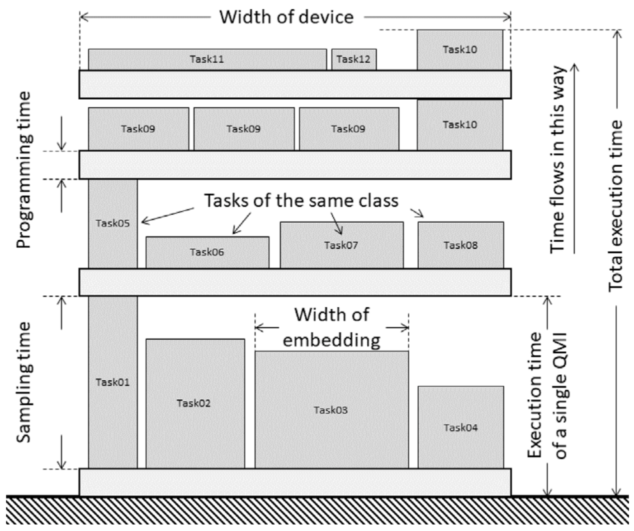


Fig. 6. Quantum Annealing Multitasking Scheduling (QAMTS) problem as a three-dimensional class constrained rack packing (3DCCRP) problem. This is a side view of the three-dimensional problem.

5. Scheduling

Section 4 describes the potential parallelism in a quantum annealer. This section discuss how to realise the parallelism given various constraints/limitations of a quantum annealer, that is a scheduling problem. The time and space sharing representation in Section 4 depicts the parallelism naturally and is widely adopted by literature related to classical processors. But it fails to capture the important constraints/limitation that are unique in QAMT scheduling (QAMTS). For example, Fig. 4 does not emphasise that all space-sharing tasks must share the same global settings of the annealing process, i.e., the class constraints that we mentioned in Section 3.3. Fig. 5 does not emphasise that programming cycle and annealing cycle never go in parallel, i.e. the blocking feature that we mentioned in Section 3.3. Additionally, the concept of preemption mentioned in Section 4 is not natively supported by the quantum annealer. Many of these constraints/limitations are unique to QAMTS and are not commonly seen in the scheduling problems on classical processors. In this section, we first present the formal definition of QAMTS. Based on the generalised definition, we present the challenges raised by the constraints/limitations and name a few potential ways of tackling the challenges.

5.1. 3D class-constrained rack packing problem

First we want to formalise the QAMT Scheduling (QAMTS) problem. We see the three-dimensional class-constrained rack packing problem (3DCCRP) as the best match to QAMTS. It is a variant of strip packing problem with three dimensional space, class constraint and shelf divisions [26]. In reality, we can find similar application of 3DCCRP, e.g., packing as many items as possible onto a rack in a grocery with the constraint that irrelevant items do not sit on the same division. This resembles the problem of maximising resource utilisation of a QPU given a period of time. But in this paper, we have a slightly different goal, which is to minimise the makespan of a given set of tasks. This is equivalent to minimising the height of the rack given a set of items.

We are given some cuboid items $\mathcal{T} = \{t_1, \dots, t_i, \dots, t_{|\mathcal{T}|}\}$. Each item t_i has a class attribute c_i, and a dimension of length l_i, width w_i and height h_i. We are also given a cuboid rack with unlimited height. Shelves (wide and flat boards) are installed horizontally in the rack. Each shelf has a dimension of length L, width W and height h_p. We pack items into the rack by partitioning all items into subsets $\mathcal{I} = \{I_1, \dots, I_j, \dots, I_{|\mathcal{I}|}\}$, such that items in the same subset share the same class. We must ensure that the arrangement of items in any subset I_j is not overlapped, not stacked, and is bounded by $L \times W$. We add/remove shelves to ensure a one-to-one mapping between subsets and shelves. We can also adjust the position of shelves along the height such that the distance between two adjacent shelves equals the height of the tallest item in a corresponding subset. The goal is to find an optimised partition $\tilde{\mathcal{I}}$ such that the height of the rack $H = \sum_{I_j \in \tilde{\mathcal{I}}} \left(h_p + \max_{i \in I_j} h_i \right)$ is minimised. In a canonical problem setting, we may have the following assumptions. The orientation of items is fixed, i.e., they cannot be flipped or rotated. The edges of the items are parallel to those of the rack. An atomic property applies to all items so that each individual cannot be divided into smaller parts.

The above definition corresponds to the non-preemptive task parallelism introduced in Section 4. We can find some correspondences between QAMTS and 3DCCRP. We plot Fig. 6 to facilitate the understanding of QAMTS. An item corresponds to an annealing task. $l_i \times w_i$

describes the shape of the task embedding. h_i represents the time of sampling. By comparing the class attribute of tasks, we know the compatibility between them. Items in a subset are tasks combined in a QMI. Multiple tasks can be combined in a QMI and share a programming cycle, as long as they are compatible, and their embeddings can be fit into a resource map without overlapping. A shelf corresponds to the programming cycle of a QMI, shared by multiple tasks. Device programming takes time, which is expressed as the height of a shelf h_p. The execution time of a QMI I_j is determined by the time for programming h_p and the longest sampling time $\max_{i \in I_j} h_i$. The height of the rack H corresponds to the total execution time of all QMIs, which is a representative metric that indicates the performance of a scheduling algorithm.

3DCCRP naturally reflects the mutual exclusion between programming and annealing cycles, as items are never overlapped with shelves. The class constraint also captures the restriction on the tasks combined in a QMI. We do not find similar settings in the scheduling problems on classical processors. 3DCCRP can be seen as a generalised problem setting for QAMTS, as it is a minimal set of constraints that realise multitasking on quantum annealing. One can find its variants by adding more constraints or giving different objectives. In reality, we can have many opportunities and challenges in optimising the performance of a QAMTS algorithm. We discuss these opportunities and challenges from the aspect of space and time.

5.2. Space allocation

We refer to Space Allocation as assigning resources from a resource map to a task without taking sampling time into account. To embed a task into an annealing device, a conventional way is to use “minor embedding” algorithms, such as Yang and Dinneen [27], which finds a “minor” of an architecture graph that matches the task graph. Minor embedding itself is an NP-hard combinatorial optimisation problem and could take hundreds of seconds to embed a graph with a hundred qubits onto a quantum annealing device. It is impractical for a real-time QAMTS algorithm to find embedding immediately in the presence of a new incoming task since the typical sampling time of a task is usually below a second. Instead, we may ask a user to provide a ready-to-use embedding along with the task so that we can adapt the embedding to the target devices. This is possible if the target device employs a regular structure in its graph topology. Thanks to the unit-tile design in D-Wave quantum annealing devices [28], we can shift an embedding along length and width to find a patch of available resources for the task.

3DCCRP assumes that the orientation of an item is fixed, i.e., one can only shift the item along length and width. But for QAMTS we can make other transformations to a task embedding. D-Wave Chimera architecture, as shown in Fig. 2(a), has a regular structure, that is, unit tiles of $K_{4,4}$ graphs [28]. Given an embedding in this topology, we can find a few transformations that remain to be an embedding in the same topology. Transformations include a cell-aligned shift, a flip along length and width, and a rotation of 90,180,270 degrees along height. The regular structure in the topology gives us more flexibility in the arrangement of tasks, such that we can combine more tasks into a QMI and improves resource utilisation.

We employ alignment of task embedding along the boundary of unit tiles. This is to avoid cross-talk between tasks. Harris et al. [29], Zaborniak and de Sousa [30] suggest that several mechanisms contribute to crosstalk with adjacent qubits through couplers, which scale inversely with the distance in between. There is usually a dense arrangement of qubits and couplers within a unit tile [28], compared with those between tiles. Aligning embeddings along the border of unit tiles would eliminate suspectable crosstalk between tasks. Additionally, this alignment would also simplify the referencing of resources. One can just use one ID to represent a group of qubits and couplers. In the case of D-Wave Chimera architecture, this refers to a $K_{4,4}$ graph.

Resource utilisation is a typical metric for evaluating the quality of a space allocation. Within the scope of space allocation, we define it as the ratio between the unit cells used and the total unit cells available in the device. The time used to find a space allocation should also be considered an important metric. Find and reuse embeddings are at the opposite end of the speed spectrum of space allocation algorithms. There are a few algorithms that can fit 2D items in a container, most of which have a focus on resource utilisation. For example, Next-fit decreasing-height (NFDH), First-fit decreasing-height (FFDH) Best-fit decreasing-height (BFDH) [31] and meta-heuristic algorithms [32]. As the annealing system becomes larger and more complicated, we can also borrow ideas and algorithms from memory management systems [33] to facilitate resource management.

5.2.1. Resource fidelity and availability

Due to the limitations in existing design and manufacturing technology, there is a mismatch between an intended problem and one that is actually implemented on a quantum annealing device [34]. Through characterisation, verification, and validation tools for quantum annealing (QAVV) [35], one can find that some resources have higher fidelity than others in a quantum computer. We would expect better results from the placement of a task in a high-fidelity area. In a fidelity-aware QAMTS, we can use QAVV tools to obtain a profile of the fidelity of resources and allocate tasks accordingly.

Apart from the fidelity issue, manufacturing defect also leads to a gap between the original architecture design and the actual resource available in an annealing device, which we have already described in Section 3.2. A QAMTS algorithm should also be aware of the availability and handle the manufacturing defect in quantum annealers accordingly.

5.2.2. Fragmentation

Task embedding sometimes does not come in regular shape, e.g. rectangular of equal size. If we try to pack these irregular shapes into a rectangular resource map, we are very likely to find gaps between task embeddings that are too small to fit an additional task. We refer to these unusable idle resources as fragmentation for QAMTS. This is similar to fragmentation in a memory management system [36], where fragments of free memory space are seen as wasted resources.

Although it is quite challenging to collect and reuse 2D fragments in QAMTS, we see potential in exploiting fragmentation in the mitigation of resource availability issues mentioned in Section 5.2.1. Since the fragments cannot be reused for other tasks anyway, we can provide a patch of resources in regular shape, which is larger than the embedding. These additional resources can be seen as “internal fragmentation” in a memory management system [37]. We ask users to provide a regular-shaped embedding that includes the additional resources as a reserved backup. By reserving some qubits and couplers in every unit tile, we can use the backup as an alternative to the missing qubits and couplers in the resource map.

In a memory management system, “External Fragmentation” [38] is observed when the memory is allocated in a large number of non-sequential blocks with gaps and cannot be used for a new allocation that requires a large continuous chunk of memory. External fragmentation issue accumulates in a classical computing paradigm, but is so far not a challenge for existing quantum annealing devices. This is because a QMI always start with resetting the whole annealer. Partial device programming is not supported at the moment this paper is written. One can always re-arrange the layout of task embeddings at the beginning of each QMI to merge some of the external fragments. On the other hand, we witness the progress in architecture design for quantum annealing [28], which is constantly reducing the cost of programming. Theoretically, partial device programming generates less heat dissipation than its global counterpart does, and it is a favourable choice for a superconducting computer that works in extremely low temperatures. We believe that the external fragmentation issue would increase along with the availability of partial device programming.

5.3. Time scheduling

Compared with the space allocation mentioned in Section 5.2, the time scheduling aspect of QAMTS emphasises timely execution of the tasks. The response time, that is, the time from the start of the execution of a task to its completion, is one of the most important performance metrics. Statistics, such as the average-case execution time (ACET) and the worst-case execution time (WCET) are widely adopted in the literature [39–41]. For QAMTS, we also value initial waiting time (IWT) [42], i.e., the time from the appearance to the start of execution. We can use the average-case initial waiting time (ACIWT) and the worst-case initial waiting time (WCIWT) to evaluate a QAMTS algorithm.

Due to differences in the sampling time of tasks, the utilisation of resources of a QMI could be restricted. Fig. 6 shows an example. Tasks 5–8 are of the same class and can be combined into one QMI. Task 5 requires a longer sampling time (or larger height) than others. The sampling time of the QMI should be no shorter than that of task 5. This results in some waste of resources assigned to tasks 6–8, illustrated as the space between these tasks and the programming cycle after them. We can mitigate this resource utilisation issue by dividing a task in sampling time. For example, in Fig. 6 task 10 is divided into two segments and is scheduled in two QMIs. This is similar to task scheduling on a classical computer, where a scheduler can suspend a task and resume the execution later. We can also expedite a task by instantiating it multiple times in a QMI. For example, we have three copies of task 9 in the same QMI.

5.3.1. Static and dynamic scheduling

In a static scheduling, a scheduler has the knowledge of tasks, as well as sufficient time to find an optimal schedule of tasks. If the static schedules are used in a recurring situation, e.g., periodic tasks, then it makes sense to use the time-demanding minor embedding algorithms to find an optimised allocation of the resource to the tasks. Otherwise, many of the bin packing algorithms mentioned in Section 5.2 can also be repurposed for static scheduling, if we extend these algorithms to 3D space and treat the third dimension as time.

For dynamic scheduling, a processor does not know about a task before its presence. This includes the time of arrival, sampling time, the shape of embedding, other annealing-related parameters, and user-specified priority. The situation is similar to what a modern OS scheduler is facing. Classical scheduling algorithms, such as First-Come First-Served (FCFS), Shortest-Job-Next (SJN) and Round Robin (RR) [43] are potential candidates for the time scheduling aspect of QAMTS. The time consumption of a QAMTS algorithm should be shorter than or at least comparable to the time of a QMI, which ranges from a few μs to one second[4].

5.3.2. Emulating preemption

We see task preemption (or context switching) as an integral part of dynamic scheduling. A scheduler on a general-purpose classical processor can temporarily interrupt an executing task and resume it at a later time. There is usually dedicated hardware mechanism that support the switches between tasks [44]. However, a QMI in D-Wave annealer is, by its nature, non-preemptive. There is no hardware for switching annealing tasks. A QMI can last a long time. There is no way for an urgent task to preempt the next time slice occupied by another task.

We emulate annealing task preemption by breaking a long-running task into a sequence of short segments, such that other prioritised tasks can interrupt them. From the QMI point of view, this translates to imposing a short run-time limit on individual QMIs. For example, a scheduler can split a QMI with 1000 annealing cycles into a queue of 10 QMIs with 100 annealing cycles and execute them sequentially. At the presence of another prioritised task, the scheduler cannot pause the QMI under execution, but can insert the QMI of the new task to the head of the queue.

The emulation of task preemption introduces additional costs in the programming cycles. The choice of the run-time limit plays a trade-off between timing and resource utilisation. In practise, we can relate the run-time limit to the sampling time required by the tasks in a QMI to play a balance between timing and resource utilisation.

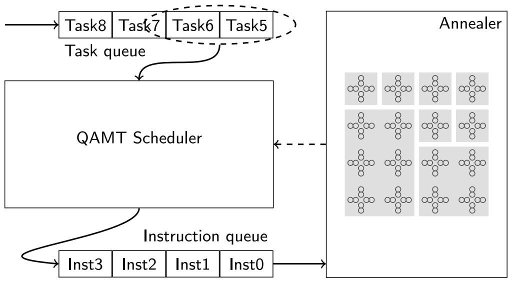


Fig. 7. QAMT Simulator.

6. Simulation

We develop an event-based QAMTS simulator[5] in Python that serves as a platform to evaluate scheduling algorithms for QAMT. Fig. 7 shows the diagram of the simulator. It takes a series of tasks as input. A QAMT scheduler combines one or multiple tasks into an instruction. The scheduler may consider the status of the annealer when translating tasks into instructions. The instructions are fed into an annealer and executed sequentially. We will disclose the source code of the simulator to the public online.

6.1. QAMT scheduling: A demo

To demonstrate the potential of QAMT, we use the simulator to carry out a few experiments. We simulate a D-Wave Chimera architecture, which employs 16×16 $K_{4,4}$ graph unit tiles, and assume that all resources are available and of identical fidelity. For the task dataset, we randomly generate a set of 100 tasks. We assume that all tasks are available at the start of the simulation. The size of embedding follows a uniform distribution between 1×1 and 12×12, with a maximum difference of 2 between the number of rows and columns. The number of samples follows a uniform distribution between 100 and 1000. The distribution of these random numbers is independent. All tasks use 100 μs annealing time and a default geometric annealing schedule. We use random seeds to control the generation process of the dataset. All results presented in this section are statistics on ten different random seeds.

We implement a demo QAMT scheduling algorithm, which employs preemptive task parallelism and sample parallelism. The algorithm is based on Next-fit decreasing size of embedding (NFDE). Given a series of tasks and a QMI, it sorts tasks according to the size of embeddings in descending order and fits tasks into the resource map in the QMI until there is not sufficient room for any additional tasks. The algorithm then starts with a new QMI and tries to fit the rest of the tasks into the QMI. The algorithm repeats until all tasks are fitted. To emulate task preemption, we set the sampling time of each QMI to that required by the task with the largest embedding in that QMI. We use the demo

[4] D-Wave limits the run-time of QPU tasks to one second. A QMI longer than one second will be divided into a few shorter QMIs.

[5] Readers with interest can find the source code via https://github.com/ianmalcolm/QAMTSimulator

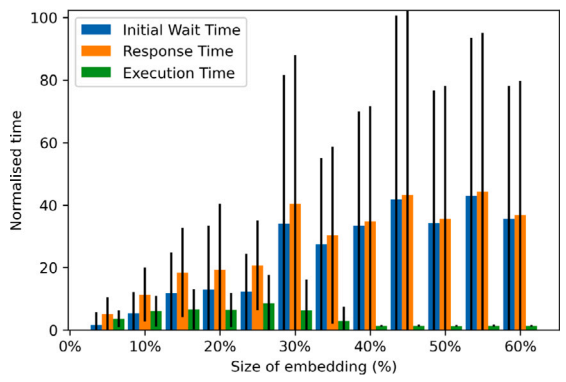


Fig. 8. Task Timing. This figure describes the relation between the size of embedding and various timing. X axis is the size of the embedding of a task in percentage. Y axis represents various timing of the schedule of that task, normalised to the sampling time required by that task. The bars describe the mean and standard deviation of the timing.

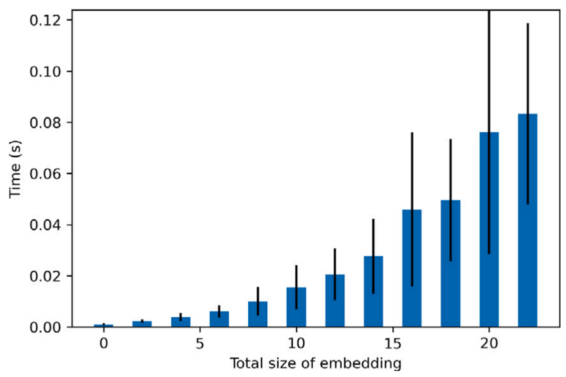


Fig. 9. Scheduler speed describes the relation between the number of resources required by tasks and the amount of time to produce a QMI. X axis represents the total size of embedding of the tasks. Y represents the time in seconds.

algorithm in a simulation of dynamic scheduling, which means the scheduler does not have the knowledge of tasks before the start of the simulation.

The demo scheduling algorithm fulfils all tasks within 2510.000 ms±367.902 ms total execution time,[6] with a resource utilisation of 49.70%±2.43% on our synthetic dataset. This is to be contrasted with 6857.000ms±193.393 ms and 18.07%±1.73% achieved by a naive scheduling algorithm, where the resources are assigned to each task exclusively and executed sequentially. The resource utilisation we use in this section considers both space and time aspects. It is defined as the total size of embedding required by all tasks over some time divided by all resources available in that period. We only see sampling as an effective use of resources. Resource utilisation can be infinitely close to, but never reaches, 100%, due to the existence of programming cycles. In our experimental settings, we assume a fixed cost of 12 ms for programming time and a sampling time range of 10–100 ms for all tasks. One can achieve better resource utilisation by increasing the sampling time if timely executions of tasks are not so critical.

We evaluate the demo scheduling algorithm from the timing of tasks, which is shown in Fig. 8. We include three kinds of timing, Initial Waiting Time, Response Time, and Execution Time. We observe that tasks with larger embeddings have a longer annealing time. This makes sense because fitting large embeddings into a resource map is more challenging. We have already assigned higher priority to large embeddings, which otherwise would suffer a longer initial waiting time. Tasks with a large embedding have shorter execution times, which is a manifest of their priority over smaller ones. We set the sampling time of each QMI to be that of the task with the largest embedding. This is likely to divide other small-embedding but sample-demanding tasks into segments, which would be scheduled in later QMIs. Response time is the sum of the initial wait time and execution time. In general, response time scales along with the increase in the size of the embedding.

We also evaluate the speed of the Python implementation of the demo scheduling algorithm, whose performance is shown in Fig. 9. The complexity of the demo scheduler scales with the number of tasks to be scheduled and scales with the amount of resources required by these tasks. At the beginning of the simulation, all 100 tasks are available to the scheduler. The total amount of resources required by the 100 tasks is roughly equivalent to 25-annealers worth of resources, which corresponds to the upper limit of x-axis in the figure. As more

[6] Please note the run-time of the demo scheduling algorithm is not included in the timing of tasks.

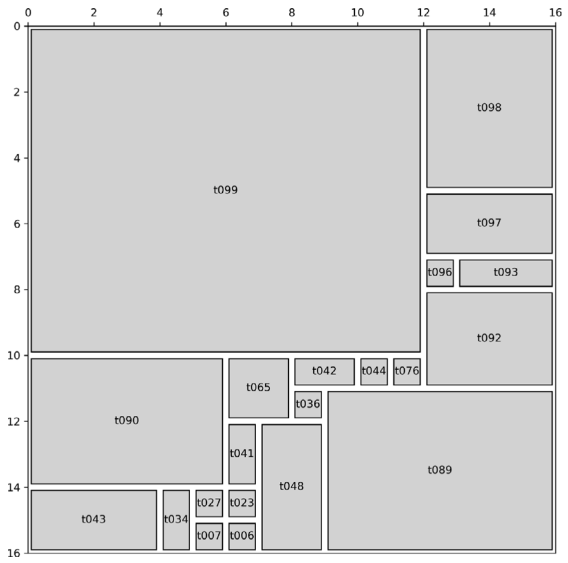


Fig. 10. An example of space allocation. The whole square represents the D-Wave Chimera architecture, which has a 16 × 16 unit tiles. Each grey rectangular represents the embedding of a task.

tasks are getting fulfilled, the time to produce a QMI decreases. On average, the scheduler takes 20.160 ms to produce one QMI; this is to be compared with the running time of a QMI, which ranges from 12+10 to 12+100 ms.

7. Conclusion and discussion

In this paper, we explore the possibility of implementing multitasking in quantum annealing systems. We compare the difference between quantum annealing systems and classical computing systems. We discuss the feasibility of task parallelism from the aspect of space and time and discuss existing algorithms that can be adapted to scheduling and challenges in scheduling. We also develop a simulator to facilitate research on QAMT scheduling and demonstrate the capability of preemptive task parallelism + sample parallelism on a synthetic dataset.

While we are taking D-Wave Chimera topology as a case study and exploring the potential of multitasking, the idea of QAMT can also be applied to more advanced topologies, such as D-Wave Pegasus topology [45] and D-Wave Zephyr topology [46]. A Pegasus topology is consisted of 16 × 16 Pegasus unit cells, each of which contains 24 qubits. Zephyr topology, as the next generation topology, is an array of $(2n+1) \times (2n+1)$ Zephyr unit cells, each of which has 16 qubits. n is a positive integer. The "array of unit cells" design methodology prevails in D-Wave. As we employ alignment of task embedding along the boundary of unit cells, QAMT is directly applicable to quantum computing platforms that adopt this design methodology. We will demonstrate the utility of QAMT on the advanced architecture in our future work.

Apart from annealing-based quantum computers, we also see potential in gate-based quantum computing systems as well. Due to the limitations in existing manufacturing techniques, the probability of decoherence increases rapidly as the scale of a quantum circuit increases. This limits the size of a practical quantum circuit and puts a restriction on the resource utilisation of universal quantum computers. One way is to share a quantum computer among multiple quantum circuits [11].

CRediT authorship contribution statement

Tian Huang: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing. **Yongxin Zhu:** Writing – review & editing. **Rick Siow Mong Goh:** Writing– review & editing, Methodology, Supervision, Funding acquisition. **Tao Luo:** Conceptualization, Writing– review & editing, Methodology, Supervision, Funding acquisition.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work.

Data availability

We will disclose the code/data to the public at https://github.com/ianmalcolm/QAMTSimulator if the paper is accepted.

Acknowledgements

This work was supported by the National Research Foundation Singapore, under its Quantum Engineering Programme 2.0 (National Quantum Computing Hub, NRF2021-QEP2-02-P01) and funding from Agency for Science, Technology and Research (#21709)

Appendix. Visualisation tools

To facilitate the development and evaluation of various scheduling algorithms, we implemented a few visualisation tools along with the simulator. These tools inspect scheduling from a space allocation and time scheduling point of view.

Fig. 10 shows the allocation of space produced by the demo scheduling algorithm. This space allocation belongs to the first QMI in a simulation, where the scheduler is faced with 100 tasks. Although tasks with small embeddings have a lower priority in scheduling, they can easily fit into the gap between those tasks with large embeddings.

Fig. 11 shows the timing aspect of tasks achieved by the demo schedule. We observe that resource utilisation is low in the second half of the scheduling. This is because the remaining tasks have very large embeddings, such that the demo scheduler cannot find a resource-efficient combination. Tasks with small embedding stand a better chance of getting fitted and have already been mostly completed in the first half of the schedule. As we have a uniform distribution in the size of embedding, low resource utilisation in the second half is inevitable. One may argue that some of the bars, for example, the one at 1.7×10^6, only take very few amounts of resources and can be combined with other QMIs for better resource utilisation. This is not feasible as those short bars represent 2D geometries, which do not necessarily fit into one resource map.

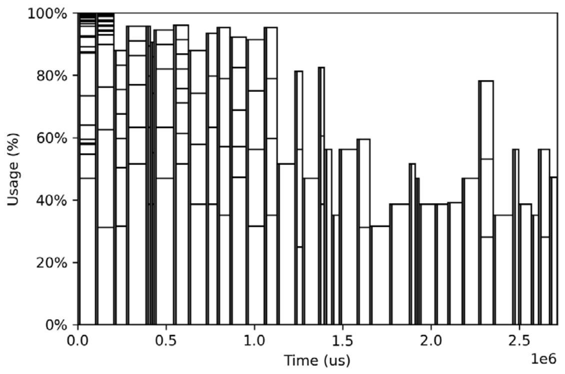


Fig. 11. An example time view of task scheduling. X axis is the time in the simulation. Y axis is the resource utilisation in percentage. Each vertical bar represents a QMI. There is always a grey edge to the left of each bar, which represents the programming stage of it. A long vertical bar splits into a few segments. The vertical length of each segment represents the percentage of resources occupied by a task in that QMI.

References

[1] Hauke P, Katzgraber HG, Lechner W, Nishimori H, Oliver WD. Perspectives of quantum annealing: Methods and implementations. Rep Progr Phys 2020;83(5):054401.
[2] Pelofske E, Hahn G, Djidjev HN. Parallel quantum annealing. Sci Rep 2022;12(1):1–11.
[3] Blake G, Dreslinski RG, Mudge T. A survey of multicore processors. IEEE Signal Process Mag 2009;26(6):26–37.
[4] Hennessy JL, Patterson DA. Computer architecture: A quantitative approach. Elsevier; 2011.
[5] Drozdowski M. Scheduling multiprocessor tasks—an overview. European J Oper Res 1996;94(2):215–30.
[6] Sheikh HF, Ahmad I, Wang Z, Ranka S. An overview and classification of thermal-aware scheduling techniques for multi-core processing systems. Sustain Comput: Inform Syst 2012;2(3):151–69.
[7] Glover F, Kochenberger G, Du Y. A tutorial on formulating and using QUBO models. 2018, http://dx.doi.org/10.48550/ARXIV.1811.11538, URL https://arxiv.org/abs/1811.11538.
[8] Mounier E. Quantum technologies 2021: Market and technology report 2021. Technical report, Yole Development; 2021.
[9] Pelofske E, Hahn G, Djidjev HN. Solving larger optimization problems using parallel quantum annealing. 2022, arXiv preprint arXiv:2205.12165.
[10] Pelofske E, Hahn G, O'Malley D, Djidjev HN, Alexandrov BS. Quantum annealing algorithms for Boolean tensor networks. Sci Rep 2022;12(1):1–19.
[11] Niu S, Todri-Sanial A. Multi-programming cross platform benchmarking for quantum computing hardware. 2022, arXiv preprint arXiv:2206.03144.
[12] Humble TS, McCaskey A, Lyakh DI, Gowrishankar M, Frisch A, Monz T. Quantum computers for high-performance computing. IEEE Micro 2021;41(5):15–23.
[13] Johanssona MP, Krishnasamyb E, Meyerc N, Piechurski C. Quantum computing–a European perspective. PRACE-6IP TR 2021.
[14] Lewis M, Glover F. Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis. Networks 2017;70(2):79–97.
[15] Vuffray M, Coffrin C, Kharkov YA, Lokhov AY. Programmable quantum annealers as noisy gibbs samplers. PRX Quantum 2022;3(2):020317.
[16] Vert D, Sirdey R, Louise S. On the limitations of the chimera graph topology in using analog quantum computers. In: Proceedings of the 16th ACM international conference on computing frontiers. 2019, p. 226–9.
[17] Ohkuwa M, Nishimori H, Lidar DA. Reverse annealing for the fully connected p-spin model. Phys Rev A 2018;98(2):022314.
[18] Fernández-Villaverde J, Hull I. Dynamic programming on a quantum annealer: Solving the RBC model. 2022.
[19] Passarelli G, Cataudella V, Lucignano P. Improving quantum annealing of the ferromagnetic p-spin model through pausing. Phys Rev B 2019;100(2):024302.
[20] Callison A, Festenstein M, Chen J, Nita L, Kendon V, Chancellor N. Energetic perspective on rapid quenches in quantum annealing. PRX Quantum 2021;2(1):010338.
[21] Hoover WG, Moran B, Hoover CG, Evans WJ. Irreversibility in the Galton board via conservative classical and quantum hamiltonian and gaussian dynamics. Phys Lett A 1988;133(3):114–20.
[22] Grant E, Humble TS, Stump B. Benchmarking quantum annealing controls with portfolio optimization. Phys Rev A 2021;15(1):014012.
[23] Inoue D, Okada A, Matsumori T, Aihara K, Yoshida H. Traffic signal optimization on a square lattice using the D-wave quantum annealer. 2020, arXiv preprint arXiv:2003.07527.

[24] Titiloye O, Crispin A. Quantum annealing of the graph coloring problem. Discrete Optim 2011;8(2):376–84.

[25] Huang T, Xu J, Luo T, Gu X, Goh R, Wong W-F. Benchmarking quantum (-inspired) annealing hardware on practical use cases. 2022, arXiv preprint arXiv:2203.02325.

[26] Xavier E, Miyazawa FK. A one-dimensional bin packing problem with shelf divisions. Discrete Appl Math 2008;156(7):1083–96.

[27] Yang Z, Dinneen MJ. Graph minor embeddings for D-wave computer architecture. Centre for Discrete Mathematics and Theoretical Computer Science; 2016.

[28] Bunyk PI, Hoskinson EM, Johnson MW, Tolkacheva E, Altomare F, Berkley AJ, et al. Architectural considerations in the design of a superconducting quantum annealing processor. IEEE Trans Appl Supercond 2014;24(4):1–10.

[29] Harris R, Johansson J, Berkley A, Johnson M, Lanting T, Han S, et al. Experimental demonstration of a robust and scalable flux qubit. Phys Rev B 2010;81(13):134510.

[30] Zaborniak T, de Sousa R. Benchmarking hamiltonian noise in the d-wave quantum annealer. IEEE Trans Quantum Eng 2021;2:1–6.

[31] Lodi A, Martello S, Vigo D. Recent advances on two-dimensional bin packing problems. Discrete Appl Math 2002;123(1–3):379–96.

[32] Hopper E, Turton BC. A review of the application of meta-heuristic algorithms to 2D strip packing problems. Artif Intell Rev 2001;16(4):257–300.

[33] Puaut I. Real-time performance of dynamic memory allocation algorithms. In: Proceedings 14th euromicro conference on real-time systems. euromicro RTS 2002. IEEE; 2002, p. 41–9.

[34] Pearson A, Mishra A, Hen I, Lidar DA. Analog errors in quantum annealing: doom and hope. NPJ Quantum Inf 2019;5:1–9.

[35] Nelson J, Vuffray M, Lokhov AY, Coffrin C. Single-qubit fidelity assessment of quantum annealing hardware. IEEE Trans Quantum Eng 2021;2:1–10.

[36] Shah VH, Shah A. An analysis and review on memory management algorithms for real time operating system. Int J Comput Sci Inf Secur 2016;14(5):236.

[37] Russell DL. Internal fragmentation in a class of buddy systems. SIAM J Comput 1977;6(4):607–21.

[38] Siebert F. Eliminating external fragmentation in a non-moving garbage collector for java. In: Proceedings of the 2000 international conference on compilers, architecture, and synthesis for embedded systems. 2000, p. 9–17.

[39] Zhao W, Kulkarni P, Whalley D, Healy C, Mueller F, Uh G-R. Timing the WCET of embedded applications. In: Proceedings. RTAS 2004. 10th IEEE real-time and embedded technology and applications symposium, 2004. IEEE; 2004, p. 472–81.

[40] Kelter T, Borghorst H, Marwedel P. WCET-aware scheduling optimizations for multi-core real-time systems. In: 2014 international conference on embedded computer systems: architectures, modeling, and simulation (SAMOS XIV). IEEE; 2014, p. 67–74.

[41] Kafshdooz MM, Taram M, Assadi S, Ejlali A. A compile-time optimization method for WCET reduction in real-time embedded systems through block formation. ACM Trans Archit Code Optim (TACO) 2016;12(4):1–25.

[42] Singh A, Goyal P, Batra S. An optimized round robin scheduling algorithm for CPU scheduling. Int J Comput Sci Eng 2010;2(07):2383–5.

[43] Siahaan APU. Comparison analysis of CPU scheduling: FCFS, SJF and round robin. Int J Eng Dev Res 2016;4(3):124–32.

[44] Ford B, Susarla S. CPU inheritance scheduling. In: OSDI. Vol. 96, (22):1996, p. 91–105.

[45] Boothby K, Bunyk P, Raymond J, Roy A. Next-generation topology of d-wave quantum processors. 2020, arXiv preprint arXiv:2003.00133.

[46] Boothby K, King A, Raymond J. Zephyr topology of D-wave quantum processors. D-wave technical report series, 2021.