Quantum approaches for Worst-Case Execution-Times analysis of programs

Gabriella Bettonte

université
PARIS-SACLAY

Quantum approaches for Worst-Case Execution-Times analysis of programs

Approches quantiques pour l'analyse des exécutions pire-cas des programmes

Thèse de doctorat de l'Université Paris-Saclay

École doctorale n° 580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Mathématiques et Informatique
Unité de recherche: Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
Graduate School: Informatique et sciences du numérique
Référent: Faculté des sciences d'Orsay

Thèse préparée à l'Université Paris-Saclay, CEA List,
sous la direction de Stéphane LOUISE (Directeur de recherche CEA, Université Paris-Saclay)
et la co-direction de Renaud SIRDEY (Directeur de recherche CEA, Université Paris-Saclay)

Thèse présentée et soutenue en visioconférence totale, le 23 mars 2023, par

Gabriella BETTONTE

Composition du jury

Liliana CUCU-GROSJEAN Directrice de recherche INRIA, INRIA	Rapportrice
Simon PERDRIX Directeur de recherche INRIA, LORIA	Rapporteur
Dritan NACE Full professor, Université de Technologie de Compiègne	Président
Éric BOURREAU Maitre de conférence, Université de Montpellier	Examinateur

Thèse de doctorat

NNT: 2023UPASG026

Résumé

L'informatique a radicalement changé notre monde : notre efficacité à effectuer des calculs a remarquablement progressé en moins d'un siècle. Des années 60 à nos jours, des nombreux progrès technologiques se sont produites toutes les quelques années. Un indicateur de ces progrès est la réduction continue de la taille des transistors. La loi de Moore prédisait que la taille des transistors diminuerait de moitié tous les deux ans, mais maintenant, elle semble avoir atteint sa limite. De plus, les transistors approchent d'une taille telle que les lois de la mécanique quantique auront un impact sur leur capacité à fonctionner. La communauté scientifique espère que ces effets quantiques, qui nuiraient à la manière classique de faire des calculs, pourraient représenter un avantage en termes d'efficacité pour nouveau paradigme de calcul.

La prise de conscience du potentiel de l'informatique quantique existe depuis 1981, lorsque Richard Feynman a imaginé pour la première fois la construction d'un ordinateur quantique capable de reproduire les principes de la mécanique quantique. Théoriquement, les algorithmes quantiques pourraient résoudre des problèmes plus rapidement que les algorithmes classiques pour certains cas spécifiques. Néanmoins, jusqu'à récemment, le domaine a connu beaucoup de scepticisme quant à ses possibilités pratiques à long terme pour résoudre les problèmes. En particulier, pendant de nombreuses années, les chercheurs ont relevé le défi de construire des ordinateurs quantiques évolutifs et fiables.

Aujourd'hui, de nombreuses entreprises ont obtenu des résultats encourageants et ont réussi à construire des machines quantiques avec suffisamment de qubits pour commencer à mener des expériences intéressantes dessus. L'entreprise canadienne D-wave par exemple, a réalisé des machines analogiques quantiques qui mettent en œuvre l'approche de recuit quantique. Ils suscitent beaucoup d'intérêt parce que leur taille est telle que des problèmes réels et difficiles peuvent y être implémenté. La machine la plus récente contient plus de 5 000 qubits. Les autres machines disponibles sont celles fournies par IBM. Ils implémentent le paradigme quantique à portes, par opposition à l'approche analogique proposée par D-wave. D'autres acteurs de l'informatique quantique, tels que Microsoft, Google et Rigetti, développent des simulateurs quantiques.

L'évaluation des temps d'exécution pire cas (WCETs) est fondamentale pour effectuer l'analyse de faisabilité des applications temps réel. L'analyse WCET fournit des garanties formelles que le temps d'exécution d'un programme répond à toutes les contraintes de timing. En général, il est impossible de calculer le WCET réel d'un programme, donc la plupart des méthodes d'évaluation d'un WCET appliquent des approximations. Ces estimations sont généralement pessimistes afin d'assurer la sécurité. Ainsi, le WCET estimé est habituellement supérieur au WCET réel. Beaucoup d'efforts dans l'exécution de l'analyse WCET consistent à réduire le pessimisme dans l'analyse pour obtenir une valeur estimée suffisamment faible pour être utile.

Nous avons choisi l'évaluation du pire temps d'exécution (WCET) comme application de nos recherches sur l'informatique quantique, car elle est cruciale pour diverses applications en temps réel. En outre, elle offre une vaste gamme de sous-problèmes à examiner, de la complexité polynomiale à la complexité exponentielle. Dans l'histoire des algorithmes quantiques, l'attention est souvent portée sur des problèmes avec une structure mathématique particulière, comme c'est, par exemple, le problème de la factorisation des nombres, abordé par l'algorithme de Shor. L'évaluation des WCET, à l'opposé, n'est pas un problème a priori favorable au contexte quantique, et possède des solutions classiques efficaces déjà éprouvées. Ainsi, il est intéressant d'explorer l'impact de l'informatique quantique sur ce type de problèmes, dans l'esprit de trouver des domaines nouveaux et concrets dans lesquels l'informatique quantique pourrait apporter sa contribution. Si ce n'est pas le cas, la recherche dans ces domaines spécifiques peut aider à définir les limites des applications qui pourraient bénéficier de l'informatique quantique

Dans ce travail de thèse, nous abordons principalement des problèmes des évaluations WCET de temps polynomial en appliquant des approches d'informatique quantique. Travailler avec des problèmes en temps polynomial nous permet de vérifier si les algorithmes quantiques peuvent donner un avantage réel avant de traiter des problèmes plus complexes. En effet, les problèmes polynomiaux permettent de calculer la solution exacte de manière classique et ainsi de pouvoir comprendre pleinement la sortie qui sera comparée à la sortie de l'algorithme quantique.

Dans le contexte de l'évaluation WCET, une influence massive est donnée par le mémoire cache. Aller chercher les données dans la mémoire principale, au lieu de les retrouver stockées dans le cache, impacte de manière non négligeable le temps d'exécution. Nous considérons un modèle simplifié dans lequel l'accès à la mémoire principale est aléatoire. Nous avons considéré le problème du comptage du nombre d'échecs de cache lors de l'exécution d'un programme. Nous avons traité ce problème avec une approche naïve, en essayant de construire une superposition d'états quantiques pour accélérer le calcul. Ce travail représente une première étape fondamentale vers l'exploitation de l'informatique quantique en montrant

un effort d'adaptation d'un problème classique au formalisme quantique.

Dans notre seconde contribution, nous nous sommes à nouveau concentrés sur le problème du comptage des cache misses. Comme cas d'utilisation, nous avons pris des programmes produisant des séquences déterministes d'accès à la mémoire lorsque des points de préemption peuvent survenir à tout moment, provoquant un non-déterminisme. Lorsqu'une préemption se produit, nous supposons que le contenu du cache est vidé. L'idée principale de ce travail est de transformer un algorithme classique que nous avons développé en un algorithme basé sur des portes quantiques. L'algorithme classique pourrait être réécrit comme une séquence d'opérations matricielles mappées dans une séquence de portes quantiques de base. Cependant, toutes les matrices ne sont pas simples à mapper dans une porte quantique de base. Pourtant, nous avons eu une amélioration théorique de l'accélération.

Enfin, nous avons exploré un modèle largement utilisé en informatique quantique : le modèle QUBO. Les machines basées sur le quantum et les machines analogiques quantiques peuvent résoudre un problème QUBO. En particulier, on considère les programmes comme des graphes, dans lesquels chaque nœud correspond à un bloc d'instructions avec un coût spécifique correspondant à son temps d'exécution. Le WCET dans ce problème est donné par le temps d'exécution du chemin le plus couteaux. Un problème aussi simple est linéaire et peut être facilement transformé en QUBO. Potentiellement, cette approche pourrait être élargie a priori pour considérer des problèmes plus complexes qui tiennent compte des effets de cache. Nous avons effectué des expériences à l'aide de machines à ondes D et d'un simulateur IBM, montrant que pour certains cas idéaux, le recuit quantique pourrait représenter un concurrent rapide du recuit simulé classique pour résoudre le problème.

Acknowledgements

I want to thank my thesis director Stéphane Louise, my research stemmed from his work and his tutoring along the way was precious to me. I also want to express my gratitude to my thesis supervisor Renaud Sirdey for his invaluable advice, all the interesting discussions and all his support, both on a professional and a human level.

I am sincerely grateful to Liliana Cucu-Grosjean e Simon Perdix for reviewing this thesis, for their precious feedback on my work, and also for the interest expressed with their stimulating questions during my Ph.D. defence. I want to thank also Dritran Nace and to Eric Bourreau for accepting to be members of the committee at my Ph.D. defence and for the interest demonstrated to my work.

I would like to thank Samuel Evain for the help of organizing a very smooth defence and Benoit Teyssandier for his constant support as chief of my laboratory during my third year at CEA.

A special thank you to my colleagues Mihail Asavoae for being there to help me during hard times on those last three years and Kods Trabelsi for her kind words. To all my Ph.D. colleagues and friends at CEA Daniel Vert, Martin Zuber, Benjamin Binder, Valentin Gilbert, Jonathan Fontaine, Julien Rodriguez, Nermine Ali, Erwan Lenormand: thank you for all the moments we shared.

To Corinne Ancourt, the first person introducing me to research with kindness and enthusiasm, goes my deepest gratitude; I will always keep a dear memory of my time in Fontainebleu. I also want to thank the people met there which stayed with me as friends, Maryna Savchenko, Maksim Berezov, Patryk Kiepas, Monika Rakoczy.

Finally, I am grateful to my family for all the support during my studies, from the beginning many years ago, and to Piercarlo for always being by my side.

Contents

Chapter 1

Introduction

Computer science has radically changed our world: our efficiency in performing computations has advanced remarkably in less than a century. From the Sixties to nowadays, technological breakthroughs have been happening every few years [1]. An indicator of such advancement is the continuous reduction of the size of transistors. Moore's Law predicted successfully that the size of transistors would halve every two years [2], but now it seems to have reached its limit. Furthermore, transistors are approaching a size such that the laws of quantum mechanics will impact their ability to function the way they did.

The scientific community raised the hope that those quantum effects, which would impair the classic way of doing computations, could represent a booster in terms of efficiency if a quantum computer would be built. Theoretically, quantum algorithms could solve problems faster than classical ones for some specific cases, such as Shor's and Grover's algorithms [3][4]. However, quantum machines have proven challenging to build and not easily scalable. Even if an actual advantage seems to exist in utilizing quantum computing, it is not clear how to evaluate this advantage. For such reasons, scientists still struggle to use quantum machines, even for algorithms that, on paper, should improve the speed-up of algorithms.

The quantum scenario nowadays

Nowadays, quantum computing is gaining massive popularity in the computer science community and public opinion. The awareness of the potential of quantum computing had existed since 1981 when Richard Feynman first speculated about building a quantum computer able to reproduce the principles of quantum mechanics. However, until recently, the field has known much skepticism about its long-term practical capabilities to solve problems. In particular, for many years, researchers faced the challenge of building scalable and reliable quantum comput-

ers. Lately, many companies obtained encouraging results and managed to build quantum machines with enough qubits to start conducting interesting experiments on them. D-wave systems have built quantum analog machines that implement the quantum annealing approach. They collect so much interest because their size is such that real-world and challenging problems can be mapped into them. The most recent machine contains over 5,000 qubits. Other available machines are those provided by IBM. They implement the gate-based quantum paradigm, as opposed to the analog approach proposed by D-wave. Other quantum computing stakeholders, such as Microsoft, Google, and Rigetti, are also developing simulators.

Worst-case execution-times evaluation

The Worst-Case Execution-Times (WCETs) evaluation is fundamental for performing feasibility analysis of real-time applications. WCET analysis provides formal guarantees that the execution time of a program matches all the scheduling and timing constraints. In general, it is impossibleto compute the actual WCET of a program, so most methods for evaluating a WCET apply approximations. Those estimates are typically pessimistic in order to provide safety. Thus the estimated WCET is usually higher than the real WCET. Much effort in performing WCET analysis is about decreasing the pessimism in analysis to obtain an estimated value low enough to be valuable to the system designer.

Motivation of this thesis

Along this thesis work, we mainly address polynomial time problems by applying quantum computing approaches. Those problems are quite simple to solve. We believe that working with polynomial-time problems allows us to verify if quantum algorithms can give an actual advantage before dealing with more complex problems. Polynomial problems allow computing the exact solution classically and thus be able to fully understand the output that will be compared to the quantum algorithm's output.

We chose the worst-case execution-time (WCET) evaluation as the application of our research on quantum computing, as it is crucial for various real-time applications. Besides, it offers a vast range of sub-problems to examine, from polynomial to exponential complexity. In quantum algorithms history, attention is often given to problems with a particular mathematical structure, as it is, for instance, the problem of factoring numbers, addressed by Shor's algorithm. The worst-case execution-time evaluation, as an opposite, is not a particularly quantum-friendly

problem, and it owns already proven efficient classical solutions. We considered it was worth exploring the impact of quantum computing on those kinds of arbitrary problems, with the spirit of finding new and unexpected fields to which quantum computing could bring its potential. If not, research on such arbitrary fields could help to set the boundaries of which applications could benefit from quantum computing.

Our contributions

This thesis presents different quantum approaches to perform WCETs evaluations of programs.

Static analysis

In the context of WCET evaluation, a massive influence is given by the cache. Fetching the data from the main memory, instead of finding it stored in the cache, impacts non-negligibly the execution time. We consider a simplified model in which access to the main memory is random. We considered the problem of counting the number of cache misses while executing a program. We dealt with this issue with a naive approach, trying to build a superposition of quantum states to speed up the computation. This work represents a first fundamental step towards exploiting quantum computing by showing an effort to adapt a classical problem to quantum formalism.

Dynamic programming

In our second contribution, we again considered the counting of the cache misses problem. As a use case, we took programs producing deterministic sequences of memory accesses when preemption points can happen anytime, causing non-determinism. When a preemption occurs, we assume the content of the cache to be flushed. The main idea of this work is to transform a classical algorithm we developed into a quantum gate-based algorithm. The classical algorithm could be rewritten as a sequence of matrix operations mapped into a sequence of basic quantum gates. However, not all the matrices are straightforward to map into a basic quantum gate. Still, we had a theoretical improvement in the speed-up.

Optimization problems

Finally, we explored a widely used model in quantum computing: the QUBO model. Quantum-based machines and quantum analog machines can solve a QUBO problem. In particular, we consider the programs as graphs, in which

each node corresponds to a block of instructions with a specific cost corresponding to its execution time. The WCET in this problem is given by the execution time of the most expensive path. Such a simple problem is linear and can be easily transformed into a QUBO. Potentially, this approach could be expanded a priori to consider more complex problems that consider the cache effects. We performed experiments using D-wave machines and IBM simulator, showing that for some ideal cases, quantum annealing could represent a fast competitor of the classical simulated annealing in solving the problem.

Manuscript structure

This thesis is made of two parts. The first part presents a general and technical overview of the context in which this thesis's work has been developed, while the second part presents our contributions.

Part I provides a presentation of the background of this thesis. Chapter 2 presents the basic notions of quantum computing as tools to go through the second part of the manuscript. In Chapter 3, we give a brief overview of the field of study of worst-case execution-time (WCET) evaluations of programs. Chapter 4 brings the reasons for which, in this thesis, we explored the potential of quantum computing applied to the context of WCETs analysis of programs.

Part II consists of three chapters, each of them presenting one of our contributions. Chapter 5 presents the first steps to define a quantum approach to perform static analysis in the case of a program whose execution produces a non-deterministic sequence of memory accesses. Chapter 6 presents the classical dynamic programming algorithm we developed to perform static analysis of programs whose execution produces a deterministic sequence of memory accessed. In particular, we included random preemption points (i.e., execution interruptions) in our model. We ported this classical algorithm to the quantum framework. Chapter 7 presents our work on optimization problems related to the WCETs field of study. We reduced the problem of finding the WCET of a program to a maximization problem on graphs, and we compared the performances while solving with a quantum annealer (D-wave machines) and with QAOA (IBM simulators).

Publications/Talks

Publications

1. Gabriella Bettonte, Valentin Gilbert, Daniel Vert, Stéphane Louise, Renaud Sirdey. Quantum approaches for WCET-related optimization problems, ICCS 2022, Jun 2022, London, United Kingdom.

2. Gabriella Bettonte, Stéphane Louise, Renaud Sirdey. Towards a quantum algorithm for evaluating WCETs, 2nd Quantum Software Engineering and Technology Workshop, IEEE Quantum Week 2021, Oct 2021, Virtual Conference, United States.

3. Gabriella Bettonte, Stéphane Louise, Renaud Sirdey. Two real-time applications of quantum computing for the evaluation of WCETs, Compas2021, Jul 2021, Lyon, France.

Talks

1. Gabriella Bettonte, Stéphane Louise and Renaud Sirdey. Quantum computing for worst-case-execution-time evaluation, MBDA Quantum Conference, Nov 2022, Domaine Saint-Paul, France.

2. Gabriella Bettonte, Stéphane Louise and Renaud Sirdey. Cache modeling: a quantum approach, Workshop on Quantum Computing and Communication, PPAM 2022, Sept 2022, Gdansk, Poland.

3. Gabriella Bettonte, Stéphane Louise, Renaud Sirdey. Towards a quantum algorithm for evaluating WCETs: an overview, 23ème congrès annuel de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, INSA Lyon, Feb 2022, Villeurbanne - Lyon, France.

Part I

Context and Motivations

Chapter 2

Quantum computing

Quantum computing promises that some particular computational tasks might be executed exponentially faster on a quantum processor than on a classical processor. We witness a growing need for algorithms that can process data in less time because data sets are quickly enlarging. Consequentially, there has been an increasing desire to implement quantum algorithms on a large scale. Nowadays, very prominent companies, such as, for instance, Google, IBM, and Microsoft, believe in the potential of quantum computing and invest in it.

2.1 A bit of history

Richard Feynman is considered the pioneer of quantum computing since he proposed in 1981 to exploit the principle of quantum mechanics to perform computations. The issue with classical computers is that they can only offer an approximation of the real world. Feynman believed that a quantum computer, i.e., a computer that follows the quantum mechanics laws, could simulate a real-world quantum system [5].

In 1980 Paul Benioff described a computer's first quantum mechanical model, showing that quantum computers are theoretically possible. In 1985 David Deutsch developed the idea of a universal quantum computer. Deutsch showed that all Turing computable functions are also computable by his universal quantum computer [6].

Nowadays, it is commonly believed that Moore's law is reaching an ending point. The law states the ability to decrease by half the size of processors every two years, but lately, it seems untrue. More than that, engineers have reached a size for which some quantum effects (for instance, tunneling) must be considered. Heisenberg uncertainty defines the eventual limit to the miniaturization we can achieve in engineering [7].

2.1.1 Potential applications

Quantum computers enable substantial speed-up to a relatively modest set of known computational problems. For other problems, quantum computers do not seem to perform better than classical computers [8]. The three main areas that the quantum computing revolution may impact are: cryptography, optimization, and simulation of quantum systems [9].

Cryptography

In 1994 Shor designed an efficient quantum algorithm to find the prime factors of large numbers [3]. Shor's discovery represents a turning point for the area of quantum computing: today, the topic of quantum computing is a central area of physics and computer science, while before, it was quite a niche topic. A large quantum computer could compute the secret key from the public key of the RSA scheme, and hence an attacker could illegitimately decode encrypted messages. While some parts of classical cryptography are not affected by such attacks, much of our online communication is protected by cryptographic procedures based on the hardness of factoring or similar problems. Quantum cryptography exploits quantum effects to design more secure cryptographic systems: measuring an unknown quantum state will disturb it; thus, the honest parties can detect such nuisance. Implementations of these quantum systems already exist.

Optimization

Quantum computers may help to solve extensive search or optimization problems offering considerable speed-ups. Some examples are search problems, finding the minimum or maximum of a given function over some finite domain, finding the shortest path between two points on a map, and approximately solving large systems of linear equations [10]–[14]. Even polynomial speed-ups can make a relevant difference in practice in those contexts, mainly when applied to massive inputs.

Simulation

A third area where quantum computers are likely to have an impact is in simulating the behavior of quantum systems. The simulation of quantum systems is the main reason that drove Richard Feynman to imagine quantum computing [5]. A quantum computer can, in principle, simulate the behavior of any other quantum system efficiently. A large part of the computing time of supercomputers today is spent on simulating quantum systems, and a quantum computer could make a huge difference here. Drug design represents a good example of an application for quantum simulation.

2.1.2 Limitations of quantum computing

The implementation of quantum computers poses considerable experimental and theoretical challenges. Foremost, the quantum system has to be designed to perform a computation in a large enough computational space and with a low enough error rate to provide a speed-up. Then, we ought to formulate a problem that is hard for a classical computer but easy for a quantum computer. Quantum computer hardware must satisfy fundamental constraints [15]:

- the qubits must interact very weakly with the environment to preserve coherence. A quantum system is defined as coherent if isolated from the outside world. Coherence is impacted by any interaction in which the environment measures or acquires information about the system;

- the qubits must interact very strongly with one another to make logic gates and transfer information;

- the states of the qubits must be able to be initialized and read out with high efficiency.

Another issue is the remarkable disparities when selecting different qubits in the implementation design and using different quantum devices to execute the algorithm [16].

Researchers have immensely progressed but building a high-fidelity processor capable of running quantum algorithms in a significant computational space still remains hard. Even if theoretically proven algorithms provide a speed-up, it is hard to reproduce the results on hardware. Some experimental results showed that current quantum computers are running slower than expected [16]. In other words, even though existing quantum computers were capable of producing results with reasonable accuracy for more extended algorithms, they could still end up being much slower than their classical counterparts. In conclusion, today, quantum computers can only be used accurately to solve simple problems with small amounts of data.

2.1.3 Future of quantum computing

The difficulties we described make some researchers doubt quantum computing and that we will fully develop quantum technologies in the foreseeable future [17]. A scenario emerges in which quantum computers will remain rather complex and expensive compared to classical computers. At the same time, it is dubious that

many people will need to own a quantum machine[1]. Today, some companies such as IBM have designed and built a few as-powerful-as-possible quantum computers and are making them widely available, enabling people to use them via the cloud.

It is conjectured that quantum computers cannot offer more than a polynomial advantage for NP-complete problems. Such a speed-up would struggle to compete with the classical heuristic approaches commonly used to solve them. However, even a polynomial speed-up could significantly benefit problems requiring exact solutions or problems that can be classically solved in sub-exponential time [18].

2.1.4 Complexity

The class of problems for which a found solution can be identified as correct in polynomial time is called NP (nondeterministic polynomial time). Many real-world practical problems happen to be NP problems. All the P (polynomials) problems are NP problems: if a problem can be solved quickly, the solution can be verified quickly too.

NP-complete problems are the most challenging NP problems. If an efficient algorithm for any of them were found, it could be adapted to solve all the other NP problems efficiently. The existence of such an algorithm for an NP-complete problem would mean that every NP problem was a P problem (P = NP).

To this day, researchers have designed only a few quantum algorithms that provide a speed-up from exponential to polynomial time for a problem. An efficient quantum algorithm to crack NP-complete problems has not been found or proved that it does not exist. A quantum algorithm capable of efficiently solving NP-complete problems has to exploit the problems' structure, as Shor's algorithm does [8].

A side-effect of the search to develop efficient quantum algorithms is the significant progress in designing fast classical algorithms. When a problem is solved efficiently in quantum computing, it draws more attention and often produces better classical options than existed before [19]. Some of the new efficient classical solutions have been directly inspired by the quantum work, see, for example, the quantum algorithm in [20] and its more efficient classical counterparts in [21].

2.1.5 Quantum supremacy

In recent years, quantum computing research focused on reaching the *quantum supremacy*. Quantum supremacy is achieved when a computational task is performed with an existing quantum device that cannot be accomplished using any

[1] Although such assertions need to be considered with caution as everyone remembers Thomas Watson, chairman of IBM, stating in 1943: "I think there is a world market for maybe five computers.

known algorithm running on a current classical supercomputer in a reasonable amount of time [22]. Harrow and Montanaro [23] have proposed a list of criteria for a quantum supremacy experiment. We need to have the following:

- a well-defined computational problem,
- a quantum algorithm solving the problem which can run on near-term hardware capable of dealing with noise and imperfections,
- several computational resources (time/space) allowed to any classical competitor,
- a small number of well-justified complexity-theoretic assumptions,
- a verification method that can efficiently distinguish the performances of the quantum algorithm from any classical competitor.

Calude and Calude [19] identify some plausible topics that could serve to prove quantum computational supremacy. These topics are evaluated according to a balance of their usefulness and their difficulty to be solved on a quantum computer. We could take as an example the factoring problem. Factoring is a useful application: as we said before, factoring quickly large numbers would break the security of applications based on the RSA scheme. However, factoring is very hard on a quantum computer, at the moment.

Google claimed to have reached quantum supremacy in 2019 [24], and the result has been quite immediately questioned by IBM [25]. The problem Google's quantum machine solved was chosen just to demonstrate the computer's superiority. Otherwise, the problem has not much practical interest. In a nutshell, the quantum computer executed a randomly chosen sequence of instructions, and then all the qubits were measured to produce an output bit string. This quantum computation has very little structure, which makes it harder for the classical computer to deal with it. The answer to this computation is not really informative or helpful. Still, this result is a valuable step in exploring the potential of quantum computing.

Google used a device with 53 qubits and declared that it took just minutes to perform quantum computations that would take today's most powerful supercomputers thousands of years. Although, the IBM team argued that an ideal simulation of the same task could be achieved on a classical system in 2.5 days and with far greater fidelity. They also expect that with additional improvements, they can further reduce the classical cost of the simulation.

To conclude, if we achieved quantum supremacy for some algorithms, they would be on particular problems [8]. A hybrid approach combining quantum and classical computing could be a promising strategy for solving some complex problems, that have been considered intractable using only classical methods [26].

2.2 Basic quantum theory

In this section, we give an overview of quantum computing, for which we will mainly follow the paper of Yanofsky [27].

We can identify three fundamental quantum-mechanical effects for quantum computers: superposition, interference, and entanglement.

Superposition

In classic computing, a binary variable is a perfectly defined binary number: the variable x can only be either 0 or 1. Superposition allows a quantum computer's memory to be in a superposition of many classical states, each state having a certain complex amplitude. For instance, a qubit represents a superposition of the states 0 and 1, creating a quantum state $|\psi\rangle$. This has been accomplished by employing elemental particles such as electrons and photons. The charge or polarization of those particles represents their current state. A qubit can represent two states in superposition, and a quantum computer can potentially complete 2^n computations in one physical step when utilizing n qubits. This phenomenon is the so-called *quantum parallelism*, and it is, theoretically, possible for a quantum computer to achieve complete tasks in exponentially fewer steps than a classical computer. Indeed, an exponential amount of computation has been achieved in the time it takes to compute the function on a single input. That said, it is essential to stress that, unfortunately, superposition alone is not enough to boost computation performance. Indeed, suppose the exponentially rich state is measured. In that case, the entire state collapses into a single randomly chosen input-output pair. It would have been easier to choose the input randomly before applying the computation classically. Quantum interference allows us to make good use of quantum parallelism [15].

Interference

Interference allows different superpositions to combine: positive and negative amplitudes cancel each other out (destructive interference), while amplitudes with the same sign add up (constructive interference). When a superposition of states is built, all the possible solutions have an equal non-zero probability of being observed. Clever exploitation of quantum interference is fundamental to benefit from quantum computing. The goal is to reinforce the probability of obtaining the desired results while reducing or even canceling the probability of obtaining unwanted results [15].

Entanglement

Entanglement allows different parts of the quantum computer, or even different quantum computers far away from each other, to be correlated. Quantum entanglement is a physical phenomenon occurring when a group of particles interacts. It is impossible to describe each particle's quantum state independently of the other's state, even if a significant distance separates the particles.

2.2.1 Quantum states

An n dimensional quantum system is a system that can be observed in one of n possible states. Column vectors of n complex numbers represent the states of a quantum system. We denote these vectors with the ket $|\rangle$ notation:

$$|\varphi\rangle = [c_0, c_1, \ldots, c_j, \ldots, c_{n-1}]^T \tag{2.1}$$

To give an example,

$$|\psi\rangle = [0, 1, \ldots, 0, \ldots, 0]^T \tag{2.2}$$

is saying that our particle will be found in position 1. The state

$$|\psi'\rangle = [0, 0, \ldots, 1, \ldots, 0]^T \tag{2.3}$$

says that the particle is in position j. These two states are examples of what are called *pure states*.

An arbitrary state is written as:

$$|\varphi\rangle = [c_0, c_1, \ldots, c_j, \ldots, c_{n-1}]^T \tag{2.4}$$

Let $\mathcal{S}$ be the sum of the squares of modulus of the c_j, i.e.

$$\mathcal{S} = |c_0|^2 + |c_1|^2 + \cdots + |c_{n-1}|^2 \tag{2.5}$$

If we measure the state described by $|\varphi\rangle$ we would find the particle in position 0 with probability $\frac{|c_0|^2}{\mathcal{S}}$, in position 1 with probability $\frac{|c_1|^2}{\mathcal{S}}$, ..., in position $n-1$ with probability $\frac{|c_{n-1}|^2}{\mathcal{S}}$.

Such states are superpositions: the particle is in more than one position at a time. The ψ means that the particle is in all positions simultaneously and not that the particle is in some single position and the c_j are giving us probabilities of which position.

These superpositions can be manipulated; for instance, they can be added:

$$|\varphi\rangle = [c_0, c_1, \ldots, c_j, \ldots, c_{n-1}]^T \tag{2.6}$$

$$|\varphi'\rangle = [c'_0, c'_1, \ldots, c'_j, \ldots, c'_{n-1}]^T \tag{2.7}$$

then

$$|\varphi\rangle + |\varphi\rangle = [c_0 + c'_0, c_1 + c'_1, \ldots, c_j + c'_j, \ldots, c_{n-1} + c'_{n-1}]^T. \tag{2.8}$$

Also, if there is a complex number $c \in \mathbb{C}$, we can multiply a ket by this c:

$$c\,|\varphi\rangle = [c \times c_0, c \times c_1, \ldots, c \times c_j, \ldots, c \times c_{n-1}]^T \tag{2.9}$$

These operations satisfy the properties of being a complex vector space. So the states of an n dimensional quantum system are represented by the complex vector space $\mathbb{C}^n$. The ket $2\,|\varphi\rangle$ describes the same physical system as $|\varphi\rangle$: for an arbitrary $c \in \mathbb{C}, c \neq 0$ we have that the $|\varphi\rangle$ and the $c\,|\varphi\rangle$ describe the same physical state. Geometrically, it is equal to say that the vector $|\varphi\rangle$ and the extension $c\,|\varphi\rangle$ describe the same physical state. So the only thing that matters is the direction of $|\varphi\rangle$. We might as well work with a normalized $|\varphi\rangle$, i.e.,

$$\frac{|\varphi\rangle}{\sqrt{\mathcal{S}}} \tag{2.10}$$

Given an n dimensional quantum system represented by $\mathbb{C}^n$ and an m dimensional quantum system represented by $\mathbb{C}^m$, we can combine these two systems to form one system. This one system is represented by the tensor product of the two vector spaces:

$$\mathbb{C}^n \otimes \mathbb{C}^m \cong \mathbb{C}^{n \times m} \tag{2.11}$$

Two quantum systems $|\varphi\rangle$ and $|\varphi'\rangle$ combined are represented as:

$$|\varphi\rangle \otimes |\varphi'\rangle = |\varphi, \varphi'\rangle = |\varphi\varphi'\rangle \tag{2.12}$$

In general, there are more elements in the tensor product of the two systems than in the union of each of the two systems. States in $\mathbb{C}^n \otimes \mathbb{C}^m$ that cannot be represented simply as an element in $\mathbb{C}^n$ and an element in $\mathbb{C}^m$ are said to be entangled.

2.2.2 Dynamics

Quantum systems constantly change, and the changes (or operators) on an n dimensional quantum system are represented by n by n unitary matrices. Given a state $|\varphi\rangle$ that represents a system at time t, then the system will be in state $U\,|\varphi\rangle$ at time $t + 1$.

U is unitary, meaning that there is a related matrix that can undo the action that U does here $U^\dagger$takes the result of U's action and returns the original vector.

In the quantum world, the only non-reversible action is measuring. If U operates on $\mathbb{C}^n$ and U' operates on $\mathbb{C}^m$, then $U \otimes U'$ will operate on $\mathbb{C}^n \otimes \mathbb{C}^m$ in the following way:

$$(U \otimes U')(|\varphi\rangle \otimes |\varphi'\rangle) = U|\varphi\rangle \otimes U'|\varphi'\rangle\,. \tag{2.13}$$

2.2.3 Observables

Other operations can be performed on a n dimensional quantum system: observing or measuring the system. When we measure a system, it is no longer in a superposition. The superposition is said to collapse to a pure state.

$$|\varphi\rangle = [c_0, c_1, \ldots, c_j, \ldots, c_{n-1}]^T \rightsquigarrow |\varphi'\rangle = [0, 0, \ldots, 1, \ldots, 0]^T. \tag{2.14}$$

The state could collapse to any of the n pure states. As before, let $\mathcal{S}$ be the sum of all the squares of the modulus, i.e.

$$\mathcal{S} = |c_0|^2 + |c_1|^2 + \cdots + |c_{n-1}|^2 \tag{2.15}$$

This means that there is a $\frac{|c_0|^2}{\mathcal{S}}$ of a chance of the superposition collapsing to the 0th pure state. There is $\frac{|c_1|^2}{\mathcal{S}}$ of a chance of the superposition collapsing to the 1st pure state, and so on. There is no way to know, a priori, to which pure state the state will collapse.

An observable or measurement on an n dimensional system is represented by an n by n hermitian matrix. A n by n matrix A is hermitian, or self-adjoint if $A^\dagger = A$. In other words, $A[j,k] = \overline{A[k,j]}$, meaning that A is hermitian if and only if $A^T = \overline{A}$

For a matrix A in $\mathbb{C}^{n\times n}$, if there is a number c in $\mathbb{C}$ and a vector $|\varphi\rangle$ in $\mathbb{C}^n$ such that

$$A|\varphi\rangle = c|\varphi\rangle \tag{2.16}$$

then c is called an eigenvalue of A and $|\varphi\rangle$ is called an eigenvector of A associated to c. The eigenvalues of a hermitian matrix are all real numbers. Furthermore, distinct eigenvectors with distinct eigenvalues of any hermitian matrix are orthogonal. The hermitian matrices representing observables for an n dimensional system have the further property that there are n distinct eigenvalues and n distinct eigenvectors That means that the set of eigenvectors forms a basis for the entire complex vector space that represents the quantum system we are interested in. Hence if we have an observable A and $|\varphi\rangle$ an eigenvalue of A then $A|\varphi\rangle = c|\varphi\rangle$ for some $c \in \mathbb{C}$. $c|\varphi\rangle$ represents the same state as $|\varphi\rangle$ as we said before. So if the system is in an eigenvector of the basis, then the system will not change.

2.3 Architecture of a quantum computer

Quantum mechanics ideas could be exploited to build a quantum computer. In order to describe a quantum machine, we need the following:

- the number of available qubits,
- the error rate (or noise) and the decoherence time
- the architecture and the qubits connection.

Several hardware companies, such as IBM and Intel, have built gate model quantum computers and are now dealing with the challenge of scaling up their devices. These machines require low temperatures to operate; thus, they need expensive refrigeration technology.

Another approach is quantum annealing: it utilizes quantum fluctuations to find the ground state of a quantum system associated to an optimization problem. Instead of expressing the problem in terms of quantum gates, we consider the problem as an optimization problem, and the quantum annealing computer seeks to find the minimum. D-Wave Systems company builds quantum annealing computers.

2.3.1 Gate-based model quantum computer

The gate model of a quantum computer requires algorithms to be expressed in terms of quantum gates.

Bits and Qubits

A bit describes a system whose set of states is of size two. A bit can be represented by two 2 by 1 matrices. From now on, we write in red the labels for the vectors. We represent the state 0, or $|0\rangle$, as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{matrix} 0 \\ 1 \end{matrix} \tag{2.17}$$

We represent the state 1, or $|1\rangle$, as:

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{matrix} 0 \\ 1 \end{matrix} \tag{2.18}$$

A classical bit is either in state $|0\rangle$ or in state $|1\rangle$. In the quantum world, we have systems where a switch is in a superposition of states $|0\rangle$ and $|1\rangle$. So we define

a quantum bit or a qubit as a way of describing a quantum system of dimension two. We represent any such qubit as a two-by-one matrix with complex numbers:

$$|\phi\rangle = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \begin{matrix} 0 \\ 1 \end{matrix} \tag{2.19}$$

where $|c_0|^2 + |c_1|^2 = 1$ (see 2.10). $|c_0|^2$ is to be interpreted as the probability that after measuring the qubit, it will be found in state $|0\rangle$. $|c_1|^2$ is to be interpreted as the probability that after measuring the qubit, it will be found in state $|1\rangle$. Whenever we measure a qubit, it automatically becomes a bit. The bits $|0\rangle$ and $|1\rangle$ are the canonical basis of $\mathbb{C}^2$. So any qubit can be written as

$$\begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = c_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + c_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.20}$$

There are several ways of writing qubits. For instance $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ can be written as

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.21}$$

Similarly: $\frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ can be written as

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}} |0\rangle - \frac{1}{\sqrt{2}} |1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.22}$$

We have that:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{|1\rangle + |0\rangle}{\sqrt{2}} \tag{2.23}$$

because they both are ways of writing $\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$. In contrast:

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}} \neq \frac{|1\rangle - |0\rangle}{\sqrt{2}} \tag{2.24}$$

The first state is $\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$ and the second one is $\begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$.
However, the two states are related:

$$\frac{|0\rangle - |1\rangle}{\sqrt{2}} = (-1)\frac{|1\rangle - |0\rangle}{\sqrt{2}} \tag{2.25}$$

Some examples of qubit implementations are given:

- an electron might be in one of two different orbits around a nucleus of an atom (ground state and excited state),
- a photon might be in one of two different polarized states,
- a subatomic particle might be in spinning in one of two different directions.

In order to be useful, a quantum device should have more than one qubit. Let us consider a byte, or eight classical bits:

$$01101011 \tag{2.26}$$

We might also write it as:

$$\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix} \tag{2.27}$$

In order to combine systems, we should use the tensor product. We can describe the above byte as

$$|0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \tag{2.28}$$

As a qubit, this is an element of:

$$\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \tag{2.29}$$

This vector space can be written as $(\mathbb{C}^2)^{\otimes 8}$. This is a complex vector space of dimension $2^8 = 256$.This space is isomorphic to $\mathbb{C}^{256}$.

We can describe our byte in the classical world as:

$$\begin{bmatrix}0\\0\\\vdots\\0\\1\\0\\\vdots\\0\\0\end{bmatrix} \begin{matrix}00000000\\00000001\\\vdots\\01101010\\01101011\\01101100\\\vdots\\11111110\\11111111\end{matrix} \tag{2.30}$$

For the quantum world, a general qubit can be written as:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{106} \\ c_{107} \\ c_{108} \\ \vdots \\ c_{254} \\ c_{255} \end{bmatrix} \begin{matrix} 00000000 \\ 00000001 \\ \vdots \\ 01101010 \\ 01101011 \\ 01101100 \\ \vdots \\ 11111110 \\ 11111111 \end{matrix} \tag{2.31}$$

where $\sum_{i=0}^{255} |c_i|^2$. In the classical world, to describe a byte, we need eight bits. In the quantum world, a state of eight qubits is given by writing 256 complex numbers. This exponential growth was one of the reasons why researchers started thinking about quantum computing. To emulate a quantum computer with a 64 qubit register, we would need to store 2^{64} complex numbers.

The qubits:

$$x = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} 00 \\ 01 \\ 10 \\ 11 \end{matrix} \tag{2.32}$$

can be written as:

$$|0\rangle \otimes |1\rangle \,. \tag{2.33}$$

We might also write these qubits as $|0, 1\rangle$ or $|01\rangle$.

Example

A variable x representing two qubits has as law of probability $P_X(0.25, 0.25, 0.25, 0.25)$, meaning that x has a probability of 25% of being equal to 00, of 25% of being equal to 01, of 25% of being equal to 10 and, finally, of 25% of being equal to 11.

We can represent a quantum variable x as a vector of amplitudes, see Fig. 2.1:

$$x = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \begin{matrix} 00 \\ 01 \\ 10 \\ 11 \end{matrix} \tag{2.34}$$

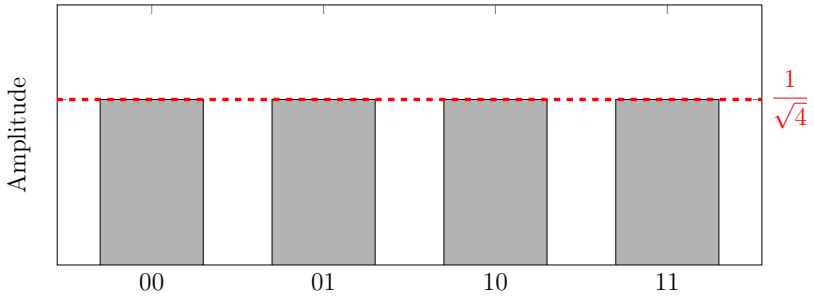


Figure 2.1: Representation of the vector x.

These qubits can be written as

$$\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \tag{2.35}$$

The tensor product of two states is not commutative:

$$|0\rangle \otimes |1\rangle = |0,1\rangle \neq |10\rangle = |1,0\rangle = |1\rangle \otimes |0\rangle \tag{2.36}$$

The first ket means that the first qubit is in state 0 and the second qubit is in state 1. The second ket says that the first qubit is in state 1 and the second qubit is in state 0.

Classical gates

To manipulate classical bits, we use classical logical gates. We represent n input bits as a 2^n by 1 matrix and m output bits as a 2^m by 1 matrix. A 2^m by 2^n matrix takes a 2^n by 1 matrix and outputs a 2^m by 1 matrix. Column vectors represent bits, and matrices represent logic gates.

For example, the NOT gate takes as input one bit and returns as output one bit. The matrix is

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.37}$$

We have that $NOT\,|1\rangle = |0\rangle$ and $NOT\,|0\rangle = |1\rangle$.

Other important gates are:

$$AND = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.38}$$

$$OR = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{2.39}$$

$$NOR = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.40}$$

If we perform a computation, one operation followed by another, it is a sequential operation.

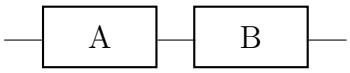


Figure 2.2: Sequential operations.

If matrix A corresponds to performing an operation and matrix B corresponds to performing another operation, then the multiplication matrix $B \times A$ corresponds to performing the operations sequentially.

There are parallel operations:

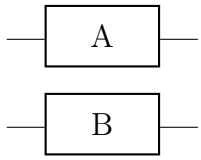


Figure 2.3: Parallel operations.

Here we are doing A to some bits and B to other bits. This will be represented by $A \otimes B$, the tensor product of two matrices. A combination of sequential and parallel operations gates (matrices) is a circuit. Four main steps make a quantum circuit:

- declaration of qubits to use
- qubits initialization
- specification of the problem to solve
- modification of the probability law to obtain the calculation result

The circuit:

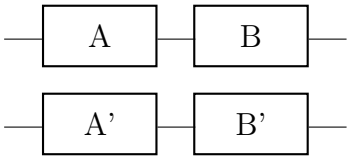


Figure 2.4: Example of circuit.

can be realized as

$$(B \times A) \otimes (B' \times A') = (B \otimes B') \times (A \otimes A') \tag{2.41}$$

Quantum gates

A quantum gate is simply any unitary matrix that manipulates qubits. Some quantum gates are the following.

The Hadamard gate:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{2.42}$$

The Pauli matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.43}$$

The controlled-not gate (Fig. 2.5) has two inputs and two outputs. The top input is the control bit. If $|x\rangle = |0\rangle$, then the output of $|y\rangle$ will be the same as the input. If $|x\rangle = |1\rangle$, then the output of $|y\rangle$ will be the opposite. If we write the top qubit first and then the bottom qubit, then the controlled-not gate takes $|x, y\rangle$ to $|x, x \oplus y\rangle$ where $\oplus$ is the binary exclusive or operation. The matrix that corresponds to this reversible gate is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.44}$$

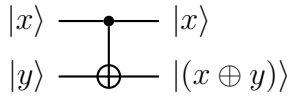


Figure 2.5: Controlled-NOT gate.

An example: Grover's algorithm

Grover's algorithm is a quantum search algorithm from 1996 [4]. We have access to an unsorted quantum database that can be asked if it contains a specific entry. Given an unsorted list of N elements, Grover's algorithm finds with high probability the target element with $O(\sqrt{N})$ operations. Conversely, a classical algorithm requires $O(N)$ operations. Therefore, the quantum algorithm provides a quadratic speed-up over its classical counterparts.

The search problem we consider is finding the target element's index among the list of $N = 2^n$ elements, where n is the number of qubits and N is the list size. The procedure of Grover's algorithm is as follows:

1. Prepare $|0\rangle^{\otimes n}$ where $\otimes$ means tensor, i.e., $|0\rangle^{\otimes n}$ is equivalent to $|0\rangle \otimes |0 \otimes \ldots |0\rangle\rangle$ with n terms.

2. Apply $H^{\otimes n}$ to create a superposition. Suppose the searched element is 10 We are in a situation as in Fig. 2.6

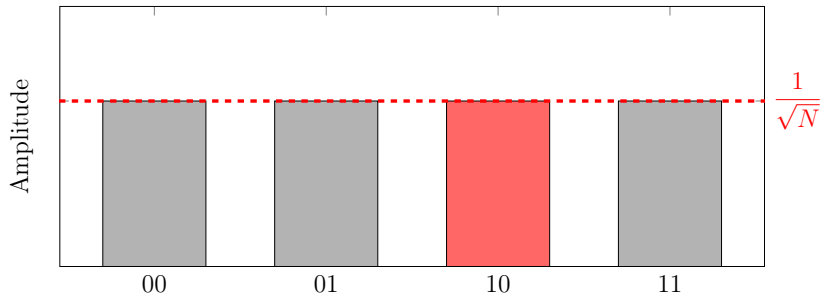


Figure 2.6: The superposition, after step 2 of Grover's algorithm. The target element is 10.

3. Apply the oracle O to mark the target element by negating its sign, i.e., $O|x\rangle = -|x\rangle$ where $|x\rangle$ is the target. See Fig. 2.7

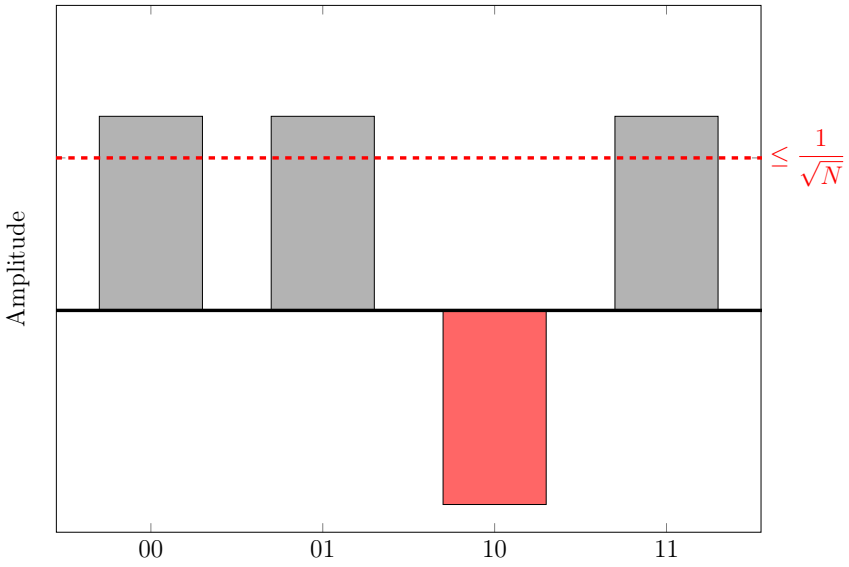


Figure 2.7: Sign negation of the target element.

4. Apply the Grover diffusion operator D to amplify the probability amplitude of the target element. See Fig. 2.8

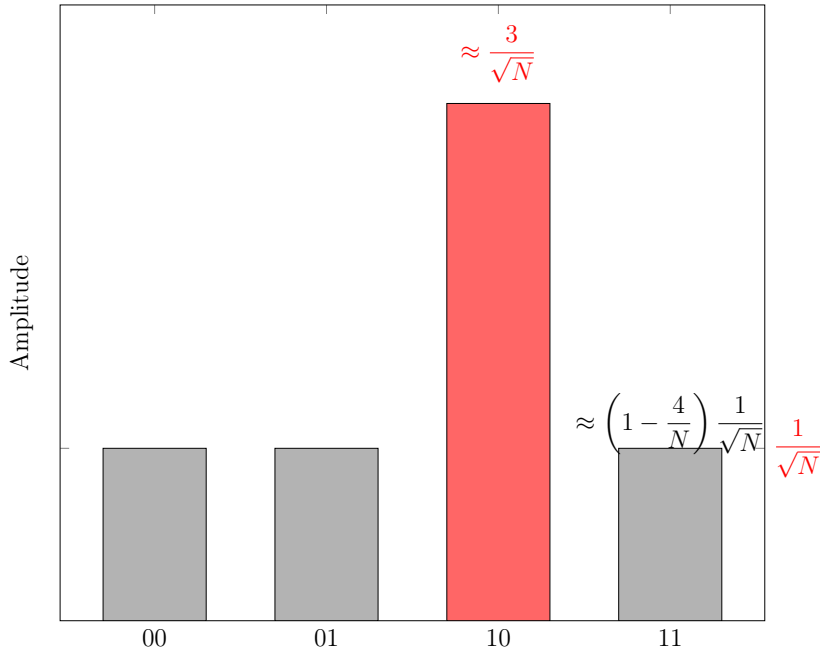


Figure 2.8: Amplification of the probability amplitude of the target element.

5. Repeat Steps 3) and 4) for about $\sqrt{N}$ times.

6. Perform measurements.

After about $\sqrt{N}$ iterations of Steps 3) and 4), we will find the target element with a high probability.

As stated in Nielsen and Chuang [28], it is useful to note that the Grover diffusion operator can be expressed as $2\left|\psi\right\rangle\left\langle\psi\right| - I_N$, where $\left|\psi\right\rangle$ is the uniform superposition of states, and I_N is the N dimensional identity matrix. $2\left|\psi\right\rangle\left\langle\psi\right| - I_N$ operates a reflection about the $\left|\psi\right\rangle$

Grover's algorithm offers a provable speed-up. However, the speed-up is not exponential, and the problem it solves is unrealistic: the cost of constructing the quantum database makes any algorithm advantage negligible. We could do much better by simply creating (and maintaining) an ordered database. Still, using Grover's algorithm as a subroutine for solving problems is efficient and a strategy used on several hybrid quantum-classical algorithms.[19]

2.3.2 Quantum annealing computers

Quantum annealing computer requires problems to be expressed as a minimization problem. Quantum annealing is a computational process that relies on the adiabatic theorem to solve combinatorial optimization problems. As a principle,

it implements a time-dependent Hamiltonian composed of an initial Hamiltonian $\mathcal{H}_0$ and a final one tied to the cost function of the optimization problem. The ground state of the initial Hamiltonian is easy to calculate. For D-Wave quantum computers, the Hamiltonian could be written as:

$$\mathcal{H}_{\text{Ising}}(t) = A(t)\mathcal{H}_0 + B(t)\mathcal{H}_P \tag{2.45}$$

where the functions $A(t)$ et $B(t)$ have to satisfy $B(t=0) = 0$ and $A(t=\tau) = 0$. From $t = 0$ to $t = \tau$, the state $\mathcal{H}(0) = \mathcal{H}_0$ evolves to $\mathcal{H}(\tau) = \mathcal{H}_P$:

$$\begin{cases} \mathcal{H}_{\text{Ising}}(0) = \mathcal{H}_0 \\ \mathcal{H}_{\text{Ising}}(\tau) = \mathcal{H}_P \end{cases} \tag{2.46}$$

where τ is the optimal annealing time.

The adiabatic theorem states that if the time evolution is slow enough (i.e., τ is large enough), then the of $\mathcal{H}_p$ ground state can be obtained with high probability.

If quantum annealing can reach a minimum energy configuration, then the associated state vector solves the equivalent optimization problem.

Chapter 3

Worst-Case Execution-Times analysis

Embedded software systems are almost everywhere to maintain the functioning of the technical devices we use daily. Many of these systems are safety-critical. Some examples are the applications in the avionics and automotive field, where safety is a crucial aspect of the functioning of the systems. Usually, these systems are also time-critical: indeed, they need to perform correct calculations, but also those have to be provided in a timely fashion. In other words, safety and precision have to be guaranteed. The Worst-Case Execution-Times (WCETs) evaluation is a fundamental concern for performing feasibility analysis of real-time applications. This chapter mainly refers to the sources [29]–[32].

3.1 Complexity of the problem

WCET analysis provides formal guarantees for the appropriate timing behavior of a system by computing tight upper bounds for the execution time of a program. The upper curve represents the set of all execution times in figure 3.1, from [29]. The shortest execution time is the best-case execution time (BCET), while the longest time is named the worst-case execution time (WCET). Determining the actual WCET of a problem may be prohibitively difficult.

In general, it is unfeasible to find WCET by analysis because it includes solving the halting problem, which is known to be undecidable. The halting problem consists of determining whether the program will terminate or continue to run forever, starting from the description of an arbitrary program and input. If it were possible to compute the actual WCET of arbitrary programs, it would be possible to determine, in linear time complexity, whether the WCET is a finite quantity, thus solving the halting problem.

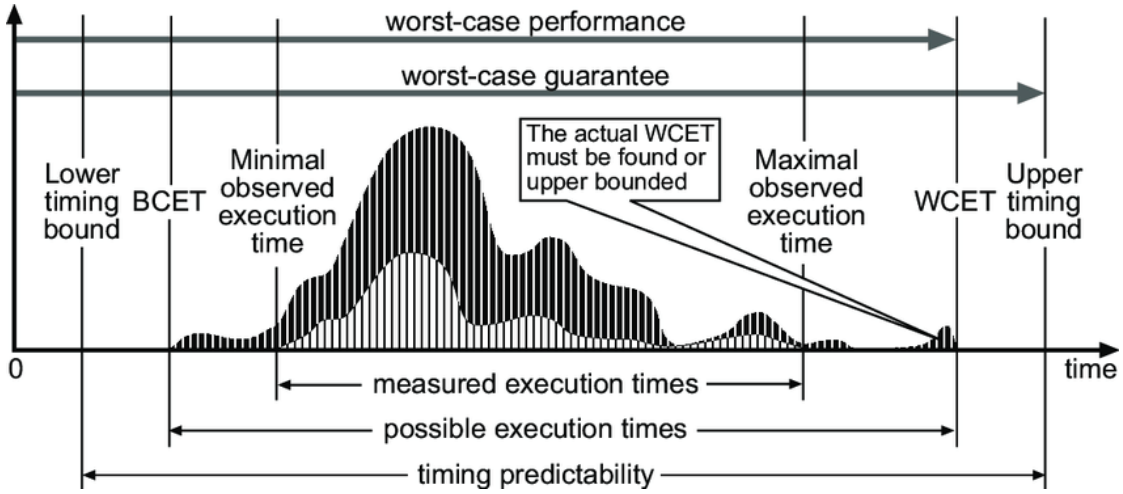


Figure 3.1: The lower curve represents a subset of measured executions. Its minimum and maximum are the minimal observed execution times and maximal observed execution times, resp. The darker curve, an envelope of the former, represents the times of all executions. Its minimum and maximum are the best-case and worst-case execution times, resp., abbreviated BCET and WCET. Figure taken from [29]

However, real-time systems only use a restricted form of programming, guaranteeing that programs always terminate.

Kligerman and Stoyenko [32], as well as Puschner and Koza [33], have listed the conditions for this problem to be decidable:

- absence of recursive function calls;
- absence of dynamic structures;
- bounded loops.

Unfortunately, the worst-case input is generally unknown and hard to derive; otherwise, its corresponding execution time would represent a reliable WCET. One could exhaustively explore all program execution paths. However, this is usually unfeasible because the number of possible paths increases exponentially with the program's size. Therefore, we can claim that, in practice, the exact WCET is often considered unobtainable. WCET could be intrinsic to the program or, depending on the platform the program is running on. The WCET should be provided for any hardware state. The problem is that modern processors have become increasingly complex, full of intellectual-property restrictions and micro-state-perturbing features such as caches, pipelines, branch prediction, preemption, and other speculative components.

3.2 Solving the problem

Most methods for evaluating a WCET apply approximations, so those different techniques produce different estimates for the WCET. Those estimates are typically pessimistic to assure safety. Thus, the estimated WCET will be higher than the real WCET. Much effort in performing WCET analysis is about reducing the pessimism in the analysis to obtain an estimated value low enough to be valuable.

State-of-the-art WCET analysis tools rely on supporting analyses to provide information on the program's execution behavior, such as loop bounds or maximum recursion depths. Typically, both steps need the binary code of the program.

There exist many automated approaches to computing WCET in the state of the art. These include:

- analytical techniques to improve test cases to increase confidence in end-to-end measurements;
- static analysis of the software. These methods do not depend on executing code on real hardware or a simulator. Instead, while considering the task code, they combine the analysis of the set of possible control-flow paths with some abstract model of the hardware architecture to obtain upper bounds. Static methods emphasize safety, guaranteeing that the execution time will not exceed these bounds.;
- a combination of measurements and structural analysis, often referred to as "hybrid" analysis. These methods execute the task or parts on the given hardware or a simulator for some set of inputs. They then take the measured times and derive the maximal and minimal observed execution times for the whole task.

Static timing analysis has been the standard to compute WCET in real-time embedded systems for long time. However, the current complex hardware deepens the limitations of static analysis. In particular, the issue of acquiring detailed information on the hardware has been proven challenging. Some probabilistic approaches for analyzing the timing behavior of next-generation real-time embedded systems have then been proposed [34]–[36].

3.3 Measurement and hybrid techniques

Measurement-based and hybrid approaches measure the execution times of small code segments on the actual hardware. Those pieces of information are then combined in a higher-level analysis. Tools consider the structure of the program

(as loops and branches) to estimate the WCET of the larger program. While it is hard to test the longest path in complex software, it is easier to test the longest path in many more minor components.

Typically, small software sections can be measured automatically by adding markers to the software or with hardware support like debuggers. The result is an execution trace, which includes the path taken through the program and the moments when it was executed. The trace is examined to determine the maximum time each part of the program took to execute, the maximum observed iteration time of each loop, and whether any elements of the software stand untested.

3.4 Static analysis

In computer science, static program analysis is a generic method to determine the properties of the dynamic behavior of a given program without actually executing it [37]. We need an abstract model of the target hardware, the binary executable, and the program's source to perform static analysis. The execution time depends on the program's control flow and the fine-grained behavior of the hardware. A static WCET tool estimates WCET by examining the computer software without executing it directly on the hardware.

Static analysis operates at a high level to determine the structure of a program, considering either the source code or the binary executable. At low-level static analysis exploits the information on the hardware's features. Static analysis provides an upper bound of the execution time of a given task on a given hardware platform by combining those two kinds of analysis.

Static WCET analyses typically consist of three parts. First, control flow and data flow analyses are used to create a program model as a control flow graph.

In the second phase, the micro-architectural analysis determines local timings, taking into account the actual timing behavior of the processor. Finally, the execution time is maximized over all control-flow paths, usually by representing the findings of the other phases as constraints in a linear program that a linear program solver can solve. This technique is called implicit path enumeration technique (IPET). Phases one and three are independent of the actual memory layout of the program on the target machine; phase two depends on the hardware and the granularity of the analysis.

3.4.1 High-level analysis

The high-level analysis addresses program execution and performs control flow analysis, building a control flow graph (CFG) representing the program. Each node is a block of code, i.e., a sequence of instructions. Program path analysis

determines what sequence of instructions will be executed in the worst-case scenario. A program data flow analysis must remove infeasible program paths from the solution search space. Therefore, a mechanism for program path annotations is essential. The number of program paths is exponential with the program size, so an efficient path analysis method is required to avoid exhaustive program path searches.

3.4.2 Integer Linear Programming (ILP) Formulation

.

Linear programming is a generic methodology to code the requirements of a system in the form of a system of linear constraints. In addition, a goal function that has to be maximized or minimized to obtain an optimal assignment of integer values to the system's variables is given. Only linear programs can be solved in polynomial time. ILP formulation should be restricted to small problem instances or to subproblems of timing analysis generating only small problem instances.

The program's control flow is translated into integer linear programs with constraints. The cost function expresses the program's execution time, and which maximal value is an upper bound for all execution times.

ILP has been used to model (straightforward) processors [30]. However, the complexity of solving the resulting integer linear programs did not allow this approach to scale.

For this thesis, we considered simple microarchitecture models that assume the execution time of an instruction to be a constant, i.e., every instruction fetch results in a cache miss.

For this simple case, the total execution time is the sum of the products of instruction counts by their corresponding instruction execution times. The instructions in the same basic block are executed together, so we consider them a single unit. If we let x_i be the execution count of a basic block B_i and c_i be the execution time of the basic block, then given that there are N basic blocks in the program, the total execution time of the program is given as:

$$\text{Total execution time} = \sum_i c_i x_i \tag{3.1}$$

Suppose we can represent the program constraints as linear inequalities. In that case, the problem of finding the estimated WCET of a program is declined to an integer linear programming (ILP) problem, which many existing ILP solvers can solve.

The linear constraints are divided in:

- program structural constraints, which are derived automatically from the program's control flow graph (CFG);

- program functionality constraints, such as loop bounds and other path information, established by the user or extracted from the program semantics.

Let us consider the example (taken from [30]) of a control flow graph in which a conditional statement is nested inside a while loop.

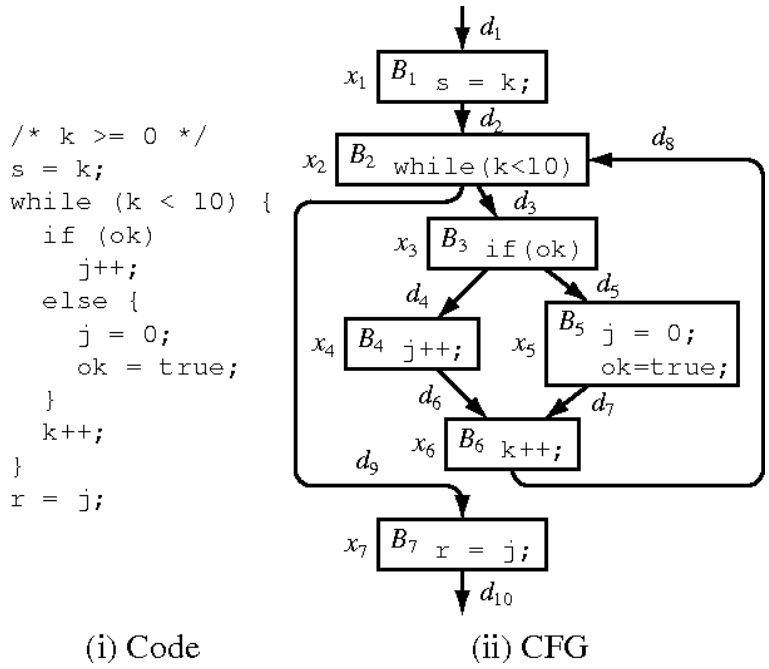


Figure 3.2: Example of a code fragment from [30]

Each node in the control flow graph represents a basic block B_i. A basic block execution count, x_i, is associated with each node. Each edge is marked with a variable d_i, representing the count of times the program control passes through that edge.

The structural constraints of the problem can be deduced from the control flow graph. For each node B_i, its execution count equals the number of times the control enters the node (inflow) and exits it (outflow). The structural constraints of this example are:

$$\begin{cases} d_1 = 1, \\ x_1 = d_1 = d_2 \\ x_2 = d_2 + d_8 = d_3 + d_9 \\ x_3 = d_3 = d_4 + d_5 \\ x_4 = d_4 = d_6 \\ x_5 = d_5 = d_7 \\ x_6 = d_6 + d_7 = d_8 \\ x_7 = d_9 = d_{10} \end{cases} \tag{3.2}$$

3.4.3 Low-level analysis

Low-level analysis determines the timing cost of individual processor instructions on the abstract model of the hardware. The timing cost is not constant in modern hardware. To perform the analysis, we must know the hardware's inner workings (pipeline, caches, etc.).

At the low-level, static WCET analysis is complicated by architectural features that improve the average-case performance of the processor: pipelined instruction execution units and cached memory systems. While speeding up the system's typical performance, these features complicate timing analysis. The cache memory, in particular, is challenging to model accurately. To determine whether the execution of an instruction results in a cache hit, several previously executed instructions must be examined.

Any incorrect prediction will result in large pessimism. Moreover, we face a lack of comprehensive information (intellectual property, patents, differences between specification and implementation). Preemption also substantially affects the analysis. Preemption is the act of temporarily interrupting an executing task to resume it at a later time. Determining tight WCET bounds employing static analysis tools on modern hardware with those features is increasingly problematic.

3.5 Cache memories

Caches are small memories with quicker access times than the main memory. They are employed to avoid long waiting times when accessing memory locations multiple times.

A cache hit happens when the program requires data already in the cache memories. Otherwise, it is a cache miss. See Fig. 3.4.

The following values mainly define the cache memories:

- the line size S_L defines the number of bytes cached together (cache block),

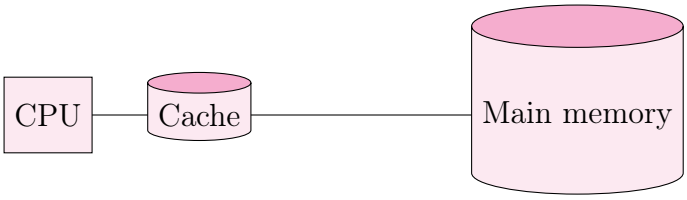


Figure 3.3: Cache memory and main memory.

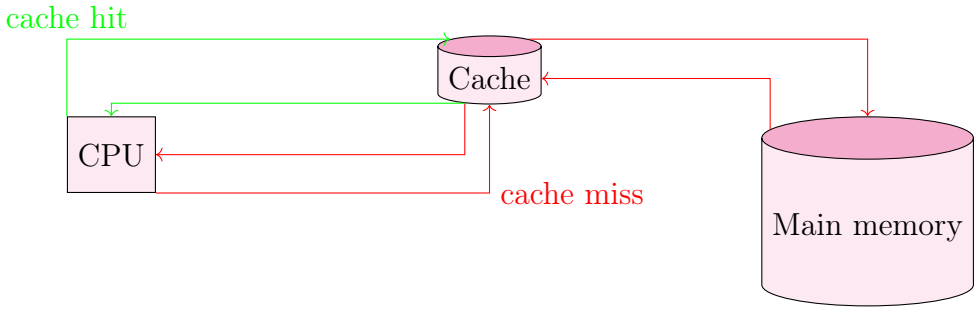


Figure 3.4: The green arrow represents a cache hit: the required data is into the cache. The red arrow represents a cache miss: the data is not into the cache and it has to be fetched from the main memory

i.e., the size of the portion of main memory loaded when a cache miss happens;

- the associativity A characterizes the number of locations into which a memory block can be loaded;
- the capacity S_C is the total number of bytes in the cache; with $n = \frac{S_C}{S_L}$ blocks in the cache.

A set consists of all memory locations that can be loaded into a cache line, and the number of different sets equals $\frac{n}{A}$.

The main mapping strategies are:

- direct mapping: the cache is split into lines, and each memory address maps to a single cache line;
- N-way set associative: the cache is separated into sets, and each memory address maps to a single cache set;
- fully associative: the cache is divided into lines, and each memory address may map to any cache line.

A cache with $A = 1$ is called direct-mapped, and a cache with $A = n$ is called a fully-associative cache. Different mapping strategies trade the likelihood of access conflicts with cache design complexity. For instance, a fully-associative cache assures access conflicts are maximally reduced, but it is very complex to realize in hardware.

Instruction caches exploit the memory's spatial and temporal proximity of executed instructions. Memories are constructed so that instructions located close to each other are not conflicting. Therefore, the behavior of caches depends on the position of each instruction (or basic block) and the relative distance of code fragments executed on the same path. Any change in the program location or the order of code fragments will directly affect the WCET analysis result.

The most common replacement strategy is the Least Recently Used policy (LRU)). The principle of selecting a new set to replace a previously stored cache line is founded on how recent the last access to data in a line was. In the picture, H_0 is the youngest access in history, until H_{A-1} which is the oldest. M stands for the miss state, which means that the considered line in memory is not in the cache. LRU policy means that when an access occurs to a given line in memory, it is copied in the cache at H_0. When another line the considered line history counter is increased by one until it reaches $A - 1$. If another concurrency occurs, then the line is expelled from the cache, and a later access would be a miss.

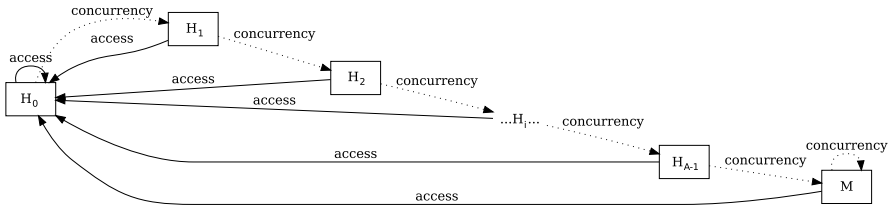


Figure 3.5: LRU remplacement strategy. Fig. from [38].

The LRU must analysis is operated to determine which cache blocks are all hit (AH), i.e., cache blocks which never generate a cache penalty. The must analysis creates abstract cache states, where each memory block maps to the cache line corresponding to its oldest possible age in the cache at the given program point. Cached blocks of the younger generation are degraded, and blocks of age A are evicted if there is a miss.

The may analysis is used to classify cache blocks as all miss (AM). It is similar to the must analysis, with the difference that it associates each memory block with the cache line of the youngest age the memory block may have.

Chapter 4

Conclusion: motivation for this thesis

This chapter summarizes the main reasons supporting this research work. This thesis explores the potential of applying quantum computing to evaluate the worst-case execution time of programs. When quantum computing was initially proposed as a way to improve computations, it was merely a speculative suggestion. A point of change was Shor's algorithm, which, if implemented in a quantum computer, could exponentially speed up a class of cryptoanalysis problems. Back then, Shor's result was still not more than of theoretical interest. In the last decade, researchers have progressed a lot in designing quantum computers, which has reinvigorated the field. Today large investments and efforts are poured into developing such a new computational paradigm. Today, the race to find new possible real-world applications for quantum computing knows a massive acceleration [39].

This thesis has taken the first steps from the work of Stéphane Louise [38]. This work developed a new approach to evaluate cache-related contributions to WCET for single-core execution, for in-order processors, especially in the case of preemptions. His approach is classical, but it has been inspired by the linear operations of Quantum Mechanics. This new method has negligible computing cost compared to the usual approach and represents a fair estimation of WCET in case of preemption. While it is not an entire parallel quantum algorithm, it is the first step in that direction. The idea was that since the formalism in single-history execution is purely linear, obtaining a quantum algorithm should be relatively straightforward.

While the interest in quantum computing is increasing at high speed, there is not enough awareness about what kind of applications can find some advantage in the context of quantum computing. There is often the misconception that quantum computing is a powerful technique that will allow us to solve all the challenging problems that, for many reasons, such as too large an input size, have

been proven difficult to solve with classical computers. Nevertheless, this is an oversimplification. A quantum computer will not be a faster version of a classical computer. Instead, it will be a different sort of computer engineered to handle coherent quantum mechanical waves for various applications.

Sometimes the speed-up that quantum computing could provide is only polynomial. For instance, Grover's algorithm has proven optimal for finding an element in one unsorted array, providing only a polynomial speed-up. Even so, it is essential to remark that even a polynomial improvement could be beneficial for some specific problems. Therefore, in those cases, the effort to obtain a speed-up through quantum computing has its reward.

Till today, scientists have not been able to identify with precision the applications for which quantum computing could be helpful. Some specific topics, like Shor's algorithm, are proven well suited for quantum computing thanks to their intrinsic structure. This thesis has as its aim the willingness to show how an arbitrary problem without a specific quantum friendliness could be tackled to be brought to the quantum framework.

The evaluation of worst-case execution-time seemed us a good playfield for this purpose. It is a problem that has been extensively studied recently, leading to a deep available knowledge of the topic. Concrete applications from automotive, smart city, healthcare, environmental, and infrastructure monitoring require specific timing constraints, similar to real-time applications running on embedded systems. Those domains usually require an HPC infrastructure to run [40]. Today, researchers focus on the challenges of pairing quantum computing systems with modern HPC infrastructure [41]. Integrating quantum computers with HPC systems is indeed within reach of current technologies. Many technical challenges remain to achieve such integration, but the idea's feasibility has proven sound. Furthermore, the field contains many problems of all sorts of complexities and kinds, with many efficient solutions that can inspire future quantum equivalents and also provide a metric of comparison for the performances of those quantum algorithms.

Part II

Contributions

Chapter 5

A quantum approach for the static analysis of cache memory misses

While the interest in quantum computing is constantly rising, the design of quantum algorithms suitable for real applications is still in its infancy. This work presents how to build a superposition of all memory access sequences generated by a program to evaluate Worst-Case Execution-Time (WCET). WCETs are fundamental for validating real-time systems where all time constraints must be met. Since the cache handling in non-deterministic programs substantially impacts the execution time, we use it as a case study to design a quantum algorithm to improve the static analysis of cache misses on programs performing random accesses to the memory. Cache misses are connected to the execution time of a program. For in-order processors, the worst-case execution-time is realized on the path with the higher amount of cache misses. This chapter presents a first attempt to use a quantum algorithm to tackle the problem of evaluating WCETs: it shows how to build a quantum superposition of the cache memory miss values and gives some insights on how this state could be exploited to extract helpful information from the superposition. The content of this chapter has been presented as a short paper on July 2021 at the conference Compas2021 [1].

5.1 Introduction

This work is a first step toward utilizing quantum computing to improve the performances of static analysis of WCETs (Worst-Case Execution Times). In particular, we consider programs that perform non-deterministic accesses to the memory, us-

[1] Gabriella Bettonte, Stéphane Louise, Renaud Sirdey. Two real-time applications of quantum computing for the evaluation of WCETs, Compas2021, Jul 2021, Lyon, France.

ing the count of cache misses as a first proxy evaluation[2] of the WCET (or as the literature calls it, the *Cache memory related delay*). A *dual* approach has already been explored in the literature, with papers about applying static analysis techniques to quantum algorithms to evaluate their performance [42], [43]. However, the application of quantum computing to improve static analysis problems has still to be fully explored, to the best of our knowledge.

We will describe an algorithm that, using a quantum-inspired formalism, builds a superposition of all the possible sequences of memory accesses. The superposition produced by that algorithm is suitable to be used as input for a yet-to-be-determined quantum algorithm that could boost the speed-up, allowing to count the worst number of cache misses with increased efficiency (i.e., using less time) with regards to the classical equivalent.

This chapter is organized as follows. Section 6.2 provides an overview of quantum computing, cache memory, and WCETs. Section 5.3 presents the addressed problem and then formalizes it. Section 5.4 describes the algorithm we designed to build a superposition of sequences of memory accesses. Sections 5.5 and 5.6 contain examples of algorithm application. It is worth noting that, unless specified otherwise, WCET will be a notation for the number of memory cache misses in this chapter. Still, we will draw the link between this number and the actual WCET in the conclusion.

5.2 Background overview

In our discussion, we use concepts from quantum computing and cache memories, which we briefly overview in this section.

5.2.1 On quantum computing

Classic computers rely on bits that are deterministic: they can be either on (in the state 0) or off (in the state 1). On the other hand, qubits have the capacity of being in a superposition of $|0\rangle$ and $|1\rangle$ (in Dirac notation). More precisely,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, |\,\alpha\,|^2 + |\,\beta\,|^2 = 1,\ \text{with}\ \alpha, \beta \in \mathbb{C} \tag{5.1}$$

A part of the interest that quantum computing arises comes from the ability of qubits to generate states of superposition that reflect all the possible outcomes of a given algorithm until a measurement is done, which is usually at the end of the algorithm. This property is called *quantum parallelism* and could theoretically give

[2]This is true, especially for in-order execution for which the execution-time grows monotonically with the number of memory cache misses.

a massive performance boost for quantum algorithms. However, the advantages of quantum computing are not without caveats: only some classes of problems can be solved by quantum computing with a significant gain in terms of efficiency with respect to classical computing. Indeed, one crucial research issue related to quantum computing is defining with precision those kinds of problems [28].

To have an idea of the momentum the scientific community gives to quantum computing [44], the Quantum Algorithm Zoo [45] (at the time of writing) cites 430 papers and counting on quantum algorithms. Companies such as Google, IBM, and Rigetti Computing are investing many resources in research on quantum technologies. Among all quantum algorithms, we can identify two main essential blueprints. The first one is defined by quantum algorithms able to reach an exponential speed-up over classical algorithms on precisely defined and heavily structured mathematical problems, e.g., Shor's algorithm for integer factorization. The second one is defined by quantum algorithms with a polynomial speed-up over classical ones, e.g., Grover's algorithm for searching an unstructured array. Still, many questions remain open about the real-world applications and benefits of quantum computing [46].

5.2.2 On cache memory and WCETs

Real-time systems are computer systems for which processing outcomes must also meet timing constraints which otherwise would jeopardize the real-time system or its environment, potentially including the life of human beings. Evaluating a WCET for programs and tasks is a fundamental issue in verifying this kind of system. In this context, the behavior of cache memories usually has the most significant impact on the execution time of programs and must be considered while evaluating worst-case performance.

Cache memories are employed to address the problem represented by the so-called memory wall: the gap between the speed of processors and the memory latency increases by order of magnitude every few years. The solution, applied mainly since the '80s, has been to add small amounts of high-speed memory close to the processing parts –the cache memories– to provide faster access to a local copy of often or recently used values in memory, thus speeding up the average access time to memory.

The data accessed – and data close to the accessed location, to exploit the principle of locality – are copied into the cache memory. The position of the copy inside the cache memory is decided by several parameters, including its so-called associativity A –which determines the number of lines in which a particular copy of the memory can reside– size, and replacement policy.

5.3 A simple model of program

As a first step, we consider evaluating the WCET for a simple model of program that considers only a linear sequence of memory accesses, some of which are deterministically and others non-deterministically chosen from a given subset of memory addresses.

We define $\mathcal{E}$ as the set of all the possible memory elements the program has access to:

$$\mathcal{E} = \{e_0, e_1 \cdots, e_{N-1}\} \tag{5.2}$$

We denote the sequence of accesses to the cache the program does as:

$$\mathcal{C}_1\mathcal{C}_2 \cdots \mathcal{C}_M \tag{5.3}$$

Any access to the cache $\mathcal{C}_i$ can be deterministic or chosen arbitrarily in some subsets of $\mathcal{E}$:

$$\mid \mathcal{C}_i \mid = \begin{cases} 1 & \text{deterministic access} \\ n, n > 1 & \text{non-deterministic access} \end{cases} \tag{5.4}$$

with

$$i = 1, \ldots, M \tag{5.5}$$

$$\forall i \quad \mathcal{C}_i \subseteq \mathcal{E} \tag{5.6}$$

We will study the case of direct mapped cache memory (i.e., associativity $A = 1$: a block of the main memory can be mapped only to a specific block/line of the cache memory). We remind that cache memories constitute a means to provide a local copy of often or recently used values in memory and decrease the average access time to memory. The allocation of copies in a cache memory is done in so-called lines. The location of these lines is defined by several parameters, including the cache memory associativity A, which determines the number of lines in each cache set, its size, and its replacement policy. For an A-way associative cache with l lines of b bytes per line, then each set defines a $l \cdot b$ space of addresses. The cache has a total number of lines $L = l \cdot A$ and a total size of $S = b \cdot L = b \cdot l \cdot A$. The replacement policy determines which previously used cache line must be replaced when a new snippet from memory must be copied into the cache after a miss [47].

Despite its simplicity, this non-deterministic model of program is difficult to statically analyze for the evaluation of WCET with classical algorithms[3]. Indeed, the non-deterministic memory accesses produce a large set of possible execution paths for which a non-exhaustive analysis can significantly overestimate the WCET.

[3]It is worth noting that despite its apparent simplicity, any single-threaded program is amenable to our model, by replacing at relevant points in the execution, accesses on different branches of the control-flow graph by a corresponding number of non-deterministic accesses.

5.4 Algorithm

The sequence of memory accesses performed by the program contains non-deterministic accesses, which our quantum algorithm superposes to take into account, in *one* execution, the number of cache misses for each possible path. Our final goal is to design an algorithm that returns (with non-negligible probability) the largest number of cache misses over every path, which should give important hints about the path of worst-case execution time for in-order processors.

5.4.1 First step: encoding

The first operation the algorithm performs is encoding the addresses of all the elements in $\mathcal{E}$ and the cache state s. With this simple model, we do not need to know where the elements in the cache are stored. As there are several options to encode the elements of $\mathcal{E}$ and s, we describe below the ones we considered while designing this algorithm.

Direct encoding for the accessed element α

We encode all the elements in $\mathcal{E}$ as vectors of N qubits, for which any possible memory access possesses its line of encoding.

$$\begin{aligned} e_0 &\longrightarrow 000\cdots1 \\ e_1 &\longrightarrow 00\cdots10 \\ &\vdots \\ e_{N-1} &\longrightarrow 10\cdots0 \end{aligned}$$

Therefore the access α to the cache, being an element of $\mathcal{E}$, is represented using a N qubits vector:

$$\alpha = \alpha[0], \alpha[1], \cdots, \alpha[N-1] \tag{5.7}$$

where $\exists!\ i$ s.t. $\alpha[i] = 1$ and $\forall j \neq i, \alpha[j] = 0$.

Compact encoding for the accessed element α

The elements in $\mathcal{E}$ are vectors of $\lceil \log_2 N \rceil$ qubits with an incremental label in an associated binary numeration.

$$\begin{aligned} e_0 &\longrightarrow 000\cdots1 \\ e_1 &\longrightarrow 00\cdots10 \\ &\vdots \\ e_{N-1} &\longrightarrow 11\cdots1 \end{aligned}$$

Therefore the access α to the cache, being an element of $\mathcal{E}$, is represented using a $\lceil \log_2 N \rceil$ qubits vector:

$$\alpha = \alpha[0], \alpha[1], \cdots, \alpha[\lceil \log_2 N \rceil] \tag{5.8}$$

The direct encoding provides a low number of quantum gates but also a high number of necessary qubits as a drawback. Conversely, a compact encoding has the advantage of requiring a lower number of necessary qubits but then, the number of quantum gates used to deal with them increases.

Encoding for the cache state s

To represent the state of the cache we chose to stick with a simple encoding. The cache state s is a N qubit vector.

$$s[j] = \begin{cases} 1 & \text{if } e_j \in \mathcal{E} \text{ is in the cache} \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

5.4.2 Second step: superposition

Since the program performs some non-deterministic accesses to the memory (i.e., $\mid \mathcal{C}_i \mid > 1$, for some i), there exist multiple possible linear sequences. We aim to exploit the quantum parallelism to compute a superposition of the total number of cache misses over each possible execution path. It is not difficult to produce a superposition of all the possible elements that can be selected for each $\mid \mathcal{C}_i \mid > 1$ employing a small network of Hadamard gates. The upper bound of the number of possible paths is:

$$R = \prod_i |C_i| \tag{5.10}$$

5.4.3 Third step: cache misses counter

For every access to the memory in each possible sequence of accesses, the algorithm checks if a cache miss or hit occurred and modifies the cache's state accordingly. After a cache miss, the accessed element is added to the cache to its assigned place. As consequence, the vector representing the cache state has to show the presence of such element inside the cache, after fetching it from the memory. The cache state vector represents the presence into the cache of an element by a 1 at the position corresponding to the element. In contrast, after a cache hit, the cache state does not change (since it is direct-mapped, every element has a unique possible position in the cache). We designed a circuit $\mathcal{Q}$ that checks if a cache miss occurred, takes note of it, and creates a new cache state that contains the considered element. It

is worth noting that we have to make a new cache state instead of updating the previous one to maintain the reversibility of the quantum circuit. The circuit $\mathcal{Q}$ is the concatenation of the circuits $\mathcal{M}$ (i.e. the MISS operator) and $\mathcal{N}$ (i.e. the NEXT operator) below.

MISS operator

The MISS operator $\mathcal{M}$ is designed to evaluate if the requested data is available in the cache (cache hit) or if the program has to perform an access to the main memory (cache miss). The inputs of the miss operator are:

- the accessed data α;
- the actual state of the cache s;
- a qubit b to store the result.

At the beginning $b = |0\rangle$, while, after the application of the miss operator, the qubit b will be in the state $|1\rangle$ if a cache miss occurred or be left to the state $|0\rangle$ if a cache hit occurred.

$$\mathcal{M} : |\alpha, s, b\rangle \longrightarrow |\alpha, s, b \oplus \text{miss}(\alpha, s)\rangle \tag{5.11}$$

where α is the accessed data at time t and s is the state of the cache when the program has access to α.

$$\text{miss}(\alpha, s) = \begin{cases} 1 & \text{if } s[\alpha] = 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.12}$$

NEXT operator

The operator NEXT $\mathcal{N}$ is built to create a new cache state s_{new} from the previous one s. The inputs of the next operator $\mathcal{N}$ are:

- the accessed data α;
- the current state of the cache s
- the information about if α generated a cache miss (i.e. the result of the MISS operator): b
- the new state of the cache s_{new}

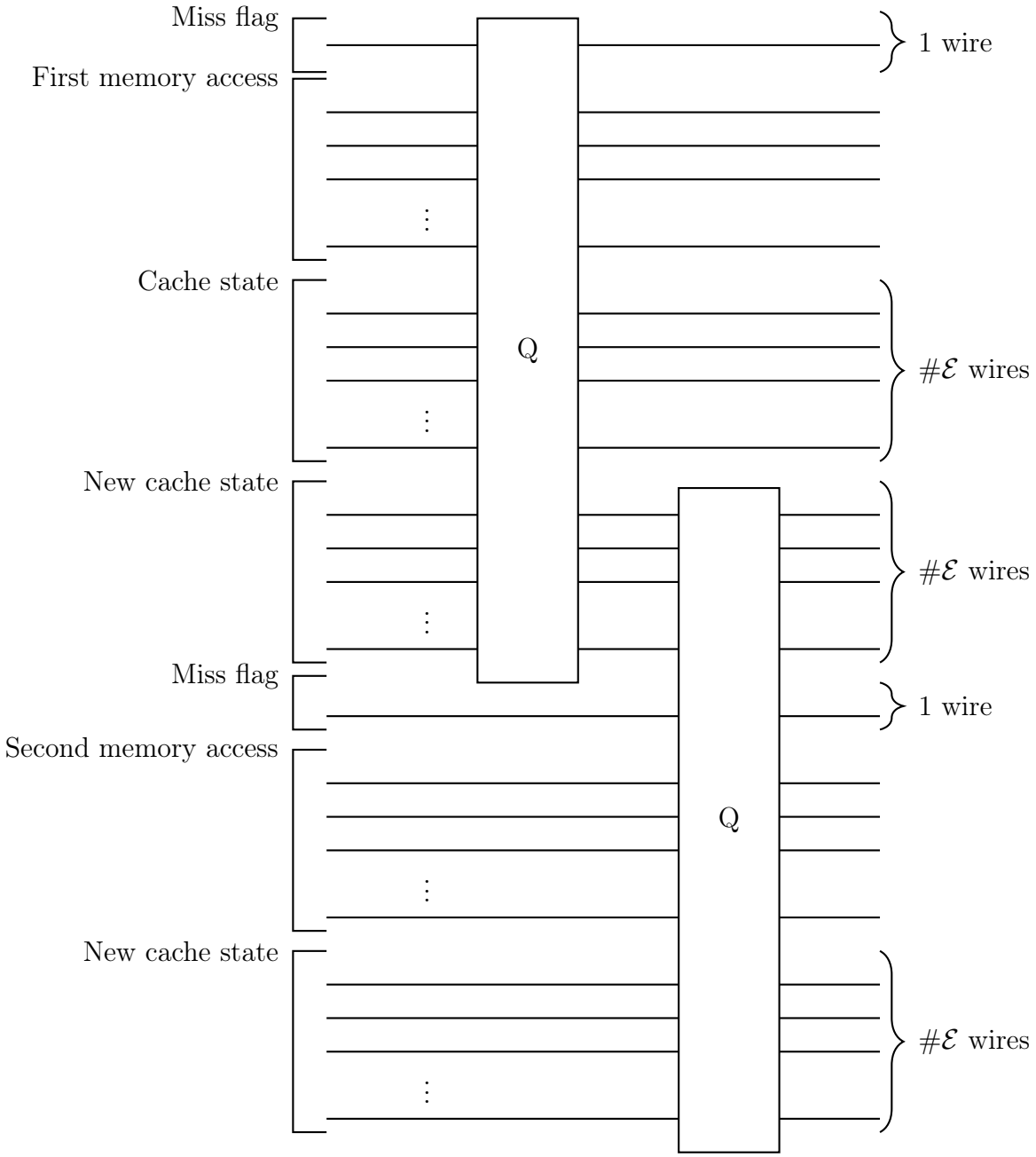
The operator $\mathcal{N}$ needs the accessed element α, the current state of the cache s. It is necessary to know also if the previous access to the cache has generated a cache hit or miss. In other words, $\mathcal{N}$ needs b (i.e. the result of the $\mathcal{M}$ operator). Then, $\mathcal{N}$ creates the new cache state s_{new} by adding, bit-a-bit to the cache state s, a vector v. The vector v is all zeros, but at the place of index α. We have that $v[\alpha]$ is the result of the function miss, described above: $\text{miss}(\alpha, s)$.To put it in another way, this corresponds to add $+1$ if the accessed data generated a cache miss, giving the information that, from that moment on, the data α is inside the cache. If the accessed data generated a cache hit, the content of the cache does not change, so s_{new} will be only a copy of s (the vector v in this case is made of all zeros).

$$\mathcal{N} : |\alpha, s, b, s_{new}\rangle \longrightarrow \left|\alpha, s, b, s \oplus \begin{bmatrix} 0 & \cdots & 0 & \text{miss}(\alpha, s) & 0 & \cdots & 0 \end{bmatrix}\right\rangle \tag{5.13}$$

where α is the accessed data at the time t and s is the state of the cache at the time t.

It is required to repeat the $\mathcal{Q}$ circuit m times (m being the number of accesses the program performs after the first non-deterministic access), with $m < M$.

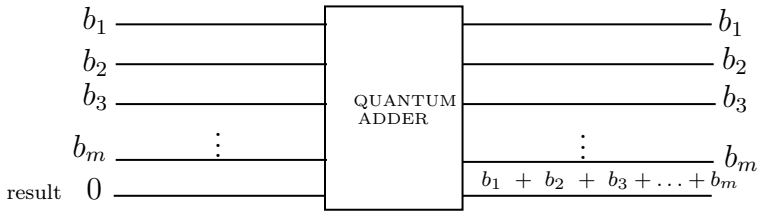
We provide below the details of the design of the $MISS$ and $NEXT$ operators for both the encoding (direct and compact) of the accessed element α. For the cache state s we stick with the direct encoding. The figure below represents the main flow of the algorithm. The circuit $\mathcal{Q}$ is made by the concatenation of the operator $\mathcal{M}$ and the operator $\mathcal{N}$. $\mathcal{Q}$ takes as input the accessed element, the actual cache state, the new cache state, and a miss flag wire. We perform $\mathcal{Q}$ for every access of the sequence of memory accessed. In this figure, two iterations of the algorithm are represented. The number of qubits required to represent the considered access or cache state is on the right.


Miss flag
First memory access
1 wire
Cache state
Q
#$\mathcal{E}$ wires
New cache state
#$\mathcal{E}$ wires
Miss flag
1 wire
Second memory access
Q
New cache state
#$\mathcal{E}$ wires

5.4.4 Fourth step: Sum operator

After performing the third step circuit, we have m cache misses wires (*i.e.* qubits): the wires associated with a cache miss are in the state $|1\rangle$ while the wires associated with a cache hit are in the state $|0\rangle$.

The idea is that we can use one of the many quantum sum operators in the literature, e.g., [48], to perform the sum of the qubits.



The obtained value, plus the cache misses we counted before the first non-deterministic access, is the total number of memory-cache misses of the sequence The total number of cache misses can be tied to an estimation of the (cache-related) worst-case execution-time for the program considered in the case of in-order processors. We want to point out that we are not providing any post-processing for extracting the maximum number of cache misses from the superposition built by our algorithm. In essence, the work presented in this chapter is a preliminary first contact with applying quantum computing to WCET problems and we focused primarily on building meaningful superpositions. Unfortunately, as it is generally the case with quantum computing, exploiting such superpositions to extract meaningful results is difficult to achieve and it remains unclear how to do so from the quantum states obtained in this chapter.

5.5 First application: direct encoding for the access α

In this section we examine the application of the algorithm when encoding the access α in the *direct* way. Since, in this case, the cache state s and the access α are encoded in the same way, to check if α generated a cache hit, it suffices to multiply bit a bit α with s and check if one of the product returns 1. Thus, we

have:

$$\text{miss}(\alpha, s) = \neg(\sum_{i=0}^{N-1} (s[i] \wedge \alpha[i]) \tag{5.14}$$

The number of (non-auxiliary) qubits needed by the miss operator $\mathcal{M}$ is $N + N + 1$ as:

- α: N qubits
- s: N qubits
- b (the result): 1 qubit.

The number of (non-auxiliary) qubits needed by the next operator $\mathcal{N}$ is $N + N + N + 1$ as:

- α: N qubits
- s: N qubits
- b: 1 qubit
- s_{new}: N qubits.

5.5.1 Example

Let us consider a set of two possible elements $\mathcal{E} = \{A, B\}$. We encode such elements as:

$$A = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{5.15}$$

and

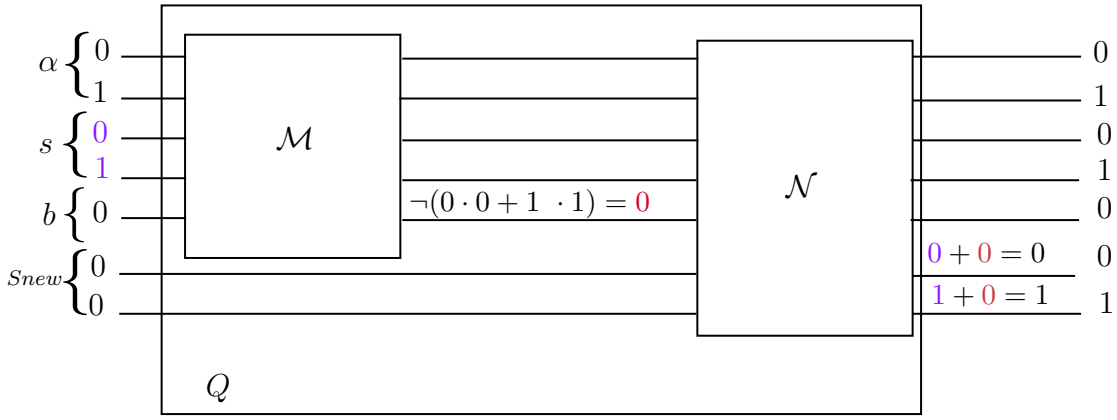
$$B = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{5.16}$$

The cache state s is a vector of dimension 2.

$$s = \begin{cases} \begin{bmatrix} 0 & 0 \end{bmatrix} & \text{if the cache is empty} \\ \begin{bmatrix} 0 & 1 \end{bmatrix} & \text{if A is inside the cache} \\ \begin{bmatrix} 1 & 0 \end{bmatrix} & \text{if B is inside the cache} \\ \begin{bmatrix} 1 & 1 \end{bmatrix} & \text{if both A and B are inside the cache} \end{cases} \tag{5.17}$$

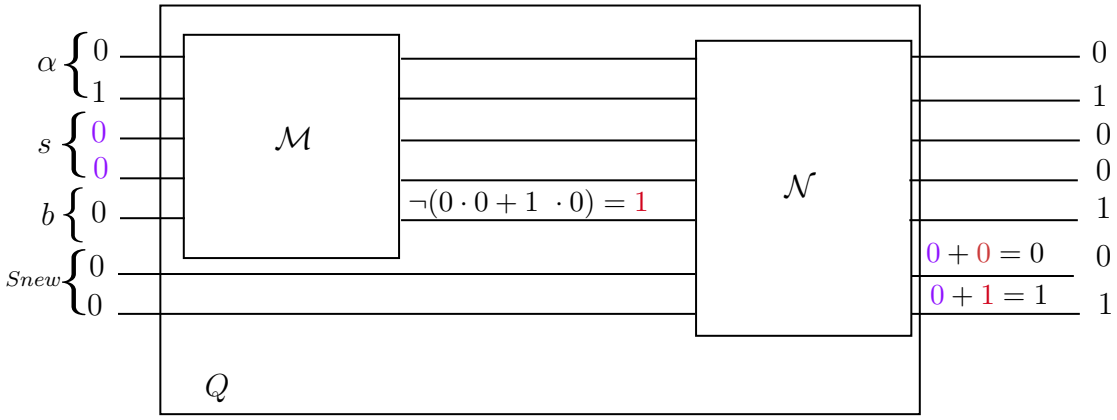
If a cache HIT happens

We suppose the program access to A (thus $\alpha = \begin{bmatrix}0 & 1\end{bmatrix}$), while inside the cache there is already A (thus, for instance, $s = \begin{bmatrix}0 & 1\end{bmatrix}$). This situation generates a cache hit. The circuit represents the action of the MISS operator $\mathcal{M}$ and the NEXT operator $\mathcal{N}$. The operator $\mathcal{M}$ returns a cache hit: $\neg(\alpha[0]\cdot s[0]+\alpha[1]\cdot s[1]) = \neg(0\cdot 0+1\cdot 1) = 0$. The NEXT operator $\mathcal{N}$ creates a new cache state that is a copy of the previous one. $\mathcal{Q}$ is the concatenation of $\mathcal{M}$ and $\mathcal{N}$.



If a cache MISS happens

Let us now suppose that the cache is empty (i.e. $s = \begin{bmatrix}0 & 0\end{bmatrix}$ while we access to $\alpha = \begin{bmatrix}0 & 1\end{bmatrix}$.
This situation generates a cache miss. The circuit represents the action of the MISS operator $\mathcal{M}$ and the NEXT operator $\mathcal{N}$. The operator $\mathcal{M}$ returns a cache miss: $\neg(\alpha[0]\cdot s[0]+\alpha[1]\cdot s[1]) = \neg(0\cdot 0+1\cdot 0) = 1$. This means that $\alpha = A$ is added to the cache. Thus, in this case $v = \begin{bmatrix}0 & 1\end{bmatrix}$ and the NEXT operator $\mathcal{N}$ creates a new cache state s_{new} where it is represented that now into the cache there is the element $\alpha = A$, i.e. $s_{new} = \begin{bmatrix}0 & 1\end{bmatrix}$. $\mathcal{Q}$ is the concatenation of $\mathcal{M}$ and $\mathcal{N}$.



5.6 Second application: compact encoding for the access α

For each of the element that could be into the cache, we need to build a miss operator, able to check its presence into the cache.

$$\forall e \in \mathcal{E}, m_e = \begin{cases} 1 & \text{if } e \text{ is } \underline{\text{not}} \text{ inside the cache} \\ 0 & \text{otherwise} \end{cases} \tag{5.18}$$

The miss operator for a general element α is:

$$\text{miss}(\alpha, s) = m_{e_1} \oplus m_{e_2} \oplus \cdots \oplus m_{e_M} \tag{5.19}$$

The number of (non-auxiliary) qubits required by the MISS operator $\mathcal{M}$ is $(n + N + 1)$, as:

- α: $n = \lceil \log_2 N \rceil$
- s: N qubits
- b: 1 qubit

The number of (non-auxiliary) qubits required by the NEXT operator $\mathcal{N}$ is $(n + N + N + 1)$, as:

- α: $n = \lceil \log_2 N \rceil$
- s: N qubits
- b: 1 qubit
- s_{new}: N qubits

5.6.1 Example

We consider a 1-way associative cache with two lines. We suppose to have the following sequence of accesses:

$$ABABE^*AE^* \tag{5.20}$$

where

$$E^* = \{E, F\} \tag{5.21}$$

With the compact encoding, to code the access α, we need two qubits: $\alpha_0\alpha_1$.

A	$\begin{bmatrix}0 & 0\end{bmatrix}$
B	$\begin{bmatrix}0 & 1\end{bmatrix}$
E	$\begin{bmatrix}1 & 0\end{bmatrix}$
F	$\begin{bmatrix}1 & 1\end{bmatrix}$

We need four qubits to store the state s of the cache: $s = s_A s_B s_E s_F$.

$$s_\alpha = \begin{cases} 1 & \text{if } \alpha \text{ is in the cache} \\ 0 & \text{otherwise} \end{cases}.$$

In our example we have four possible situations that can happen: *ABABEAE*, *ABABEAF*, *ABABFAE* and *ABABFAF*. There are only deterministic accesses at the beginning: we have two cache misses (A, B) and two cache hits (A, B).

$$0000 \xrightarrow{\text{access to A}} 1000 \xrightarrow{\text{access to B}} 1100 \xrightarrow{\text{access to A}} \cdots$$
$$\downarrow \text{cache miss} \qquad \downarrow \text{cache miss}$$
$$+1 \qquad +1$$

$$\cdots \xrightarrow{\text{access to A}} 1100 \xrightarrow{\text{access to B}} 1100$$
$$\downarrow \text{cache hit} \qquad \downarrow \text{cache hit}$$
$$+0 \qquad +0$$

We apply the Hadamard gate on the second qubits to build a superposition of E and F.
The miss operator $\mathcal{M}$ for our example is:

$$|\alpha, s, b\rangle \longrightarrow |\alpha, s, b \oplus \text{miss}(\alpha, s)\rangle \tag{5.22}$$

where:

$$\text{miss}(\alpha, s) = m_A \oplus m_B \oplus m_E \oplus m_F \tag{5.23}$$

More precisely:

- $m_A = \neg\alpha_0 \wedge \neg\alpha_1 \wedge \neg s_{00}$
- $m_B = \neg\alpha_0 \wedge \alpha_1 \wedge \neg s_{01}$
- $m_E = \alpha_0 \wedge \neg\alpha_1 \wedge \neg s_{10}$
- $m_F = \alpha_0 \wedge \alpha_1 \wedge \neg s_{11}$

The number of (non-auxiliary) qubits needed by the operator $\mathcal{M}$ is 7:

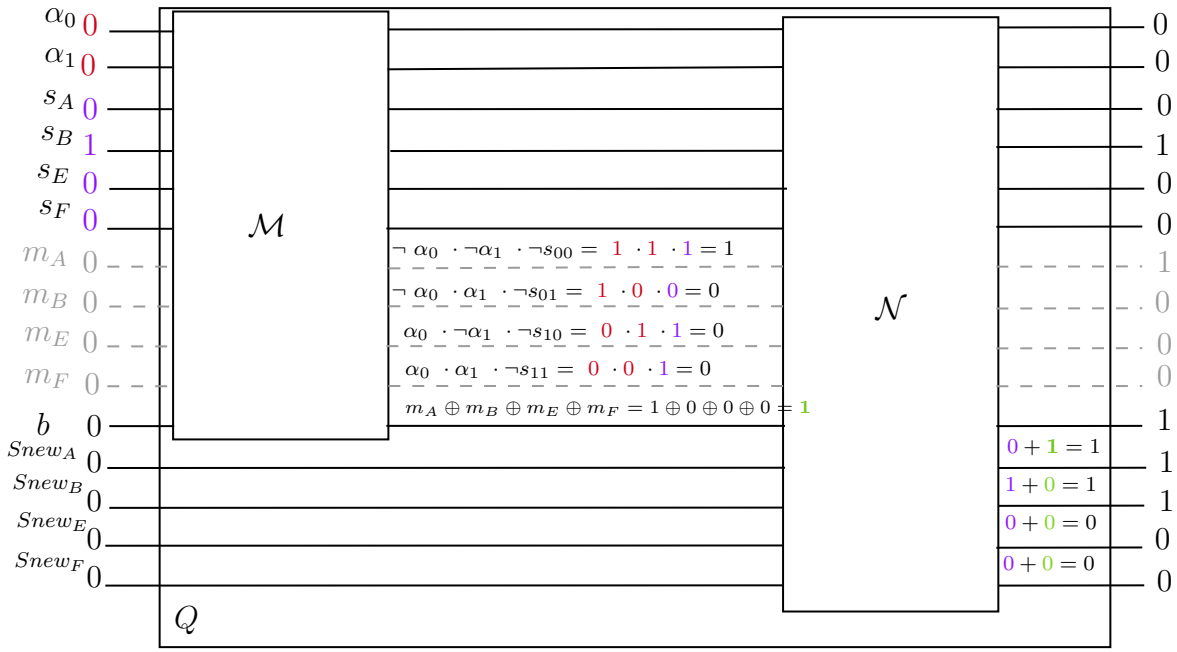
- access α: 2 qubits
- cache state s: 4 qubits
- target b: 1 qubit

The number of (non-auxiliary) qubits needed by the operator $\mathcal{N}$ is 11:

- access α: 2 qubits
- cache state s: 4 qubits
- target b: 1 qubit
- new cache state s: 4 qubits

If a cache miss happens

In the circuit below we consider an access to the element A (i.e. $\alpha = \begin{bmatrix}0 & 0\end{bmatrix}$), while inside the cache there is only B (i.e $s = \begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}$. This situation generates a cache miss. After applying the $MISS$ operator, q will be equal to 1. The $NEXT$ operator creates a new cache, representing the presence of A and B (i.e. $s_{new} = \begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}$).



5.7 Conclusion

To explore the potential advantages of quantum computing for static analysis, we chose to consider programs performing non-deterministic accesses to the memory as a first case study. Since static analysis on those programs generates a considerable amount of equally important execution paths, it is tempting to use quantum parallelism to build a superposition of analysis results (here, the number of cache misses) for all these paths. Overall, this chapter presents a preliminary work and there still are some aspects, in particular with respect to post-processing of the quantum states, that require further exploration to derive a fully functional algorithm. For instance, we should try to encode the cache state s in a compact way because it would potentially be more efficient in terms of the number of qubits required (yet, this can be expected to result in a gate increase trade-off). To do so, we should find a way to connect the elements represented into the cache state (*i.e.* the index of the accessed element in $\mathcal{E}$) and the corresponding elements in $\mathcal{E}$. Also, a complete formalization of the creation of the state superposition in the second step of the algorithm must be dealt with. We considered a straightforward program to which we applied our algorithm in this chapter. We should define an approach for more complex programs. For example, we could beforehand split the program into smaller pieces onto which we would perform the quantum static analysis described in this chapter instead of directly applying it holistically to the program. Beyond those technical details, some conceptual questions must be raised as well. We have

yet to provide post-processing for the superposed state whose construction was shown in this chapter, as the ultimate goal is to extract beneficial analysis results, as we state in section 5.4.4. There are pieces of evidence that quantum computing can significantly accelerate certain computations, even if there is no formal proof of that, except for the case of Grover's algorithm. It is also worth mentioning that exponential differences in speed-ups between classical and quantum computing are known only for problems with no formally established complexity lower bounds. For the future, to obtain a speed-up over classical algorithms, we could try to take advantage of Simon's algorithm, which is an oracle algorithm that provides an exponential speed-up over classical computation. However, this will most likely apply to programs with specific access patterns. Another option could be to obtain a polynomial speed-up over classical computation, as researchers do today.

Chapter 6

Porting a classical dynamic programming algorithm for WCET to the quantum framework

In this chapter, we propose a quantum-based solution to the problem of counting the cache hits, an important issue when analyzing real-time embedded applications. This field has already seen the development of *quantum-inspired* classical algorithms, which are competitive with the state of the art. We designed a polynomial-time dynamic programming algorithm for computing the lowest number of cache hits realized by a deterministic sequence of memory accesses in the presence of preemptions. Our contribution consists of porting that algorithm to the quantum framework, improving the algorithm's complexity from $O(N^3)$ to $O(N^2 + N)$. The content of this chapter has been published as a long paper in the proceedings of the Quantum Software Engineering and Technology Workshop (QSET21), co-located with IEEE International Conference on Quantum Computing and Engineering (QCE21) (IEEE Quantum Week 2021) [1].

6.1 Introduction

For the most part, the interest addressed to quantum computing comes from the ability of qubits to store a superposition state that reflects all the possible inputs/outputs of a given algorithm until a measurement is done. This property is called *quantum parallelism* and can, in some instances, give a massive performance boost for quantum algorithms. However, the advantages of quantum computing do

[1] Gabriella Bettonte, Stéphane Louise, Renaud Sirdey. Towards a quantum algorithm for evaluating WCETs, 2nd Quantum Software Engineering and Technology Workshop, IEEE Quantum Week 2021, Oct 2021, Virtual Conference, United States.

not come without caveats: only some classes of problems can be solved by quantum computing, with a definite gain in terms of efficiency with respect to classical computing. Indeed, one crucial research issue related to quantum computing is defining with precision such problems.

However, we can identify two main design blueprints within the variety of quantum algorithms. The first one is defined by quantum algorithms capable of reaching an exponential speedup over classical algorithms on precisely defined and heavily structured mathematical problems like Shor's algorithm for integer factorization. The second one is defined by quantum algorithms with a quadratic speedup over classical ones like Grover's algorithm for searching an unstructured array. Still, many questions remain about the real-world applications and benefits of quantum computing, especially for more arbitrary problems and fields without any a priori quantum-friendliness.

We consider the problem of worst-case execution time (WCET) evaluation by static analysis of programs. The WCET analysis is highly relevant to the design of safety-critical real-time systems, which must respect all the timing constraints to match safety properties. Also, WCET analysis has recently seen the developments of *quantum-inspired* classical algorithms, which are competitive (in terms of precision and efficiency) with state-of-the-art approaches. Furthermore, the problems arising in WCET evaluation cover a wide range of complexity classes, from undecidability in the general case to NP-hardness and polynomial-time solvability in some restricted cases [29]. As such, it appears to provide a relevant playground to put the quantum computing promise to the test, although other fields may be as relevant and *should* be explored as well.

In this direction, in this chapter, we tackle only a restricted setup with the most simple program model: the evaluation of the worst-case number of cache misses of programs performing deterministic sequences of memory accesses in case of arbitrary preemptions. Preemption is the act of interrupting one task to allow the execution of another task on a machine. There is a strong connection between the number of cache misses done by a program and its execution time, as uncached memory accesses are highly time-consuming on modern processors. In other words, we consider a single path program (linear sequences of instructions), and the complexity of the model comes from the arbitrary-placed interruptions due to other programs running on the same system, which create interferences in the cache memory. Our model considers K preemption points, which can occur at any time in the sequence, making the cache behavior non-predictable.

In this chapter, we propose a dynamic programming classical algorithm with polynomial-time complexity to compute the minimal number of cache hits in the sequence of memory accesses as a proxy measure of the WCET for the considered program. Although our model is standard for the WCET community, we consider

it, as well as the classical polynomial-time algorithm solving it, as the basis to derive a lower complexity (still polynomial-time) hybrid quantum-classical algorithm. In doing so, we demonstrate a first benefit of explicitly using the quantum computing paradigm in the WCET calculations field, albeit in a simplified setting.

Dynamic programming is an algorithmic technique to solve a problem consisting in finding one optimal solution by solving a family of more manageable sub-problems. We designed a dynamic programming algorithm to compute the minimal number of cache hits while executing a deterministic sequence with K preemptions. This technique to solve WCETs is consistent with other classical solutions in the literature [49]. Still, we want to emphasize that designing the best classical algorithm to solve the considered problem is not the point of our work: we are focused on the portability of classical algorithms to the quantum framework.

This chapter is organized as follows: Section 6.2 provides a brief overview of cache memories and preemptions. Then, Section 6.3 places the work in the context of the state-of-the-art and presents our program model. In Section 6.4 we propose a dynamic programming algorithm for evaluating WCET. Section 6.5 then contains the quantum version of that algorithm. Lastly, in Section 6.6, we compare the complexity of the two algorithms, and in Section 6.7, we provide some perspectives for future works.

6.2 Background on cache and preemption

Cache memories impact the variability of execution times since the access time between an element stored in the cache memories and an element that is not can be up to two orders of magnitude. As cache memories are limited in size, the hardware uses the history of previous accesses in the cache to decide which currently stored elements should be replaced when a new one must be stored in the cache memory. For instance, LRU (Least Recently Used) cache policy privileges, as a replacement candidate, the oldest used line of memory in a set. In the general case, when the program access an element already in the cache, it is called a *cache hit*; otherwise, *cache miss*. Cache misses impact the execution time because the missing element needs to be fetched from the main memory, which induces additional delays.

The advantage of preemptions is the possibility of optimal computing power utilization. In particular, in fully preemptive systems, the running task can be interrupted at any time by another task with higher priority and be resumed to continue when all higher priority tasks have been completed [50]. When the task is preempted, the memory blocks corresponding to the task are usually considered as flushed from the cache memory[2], between the time the task is preempted and the

[2]Whilst it is possible that, for a given set of tasks, preemption would preserve some cache

time the task resumes execution. Therefore a substantial amount of time is spent to restore the previous content of the cache, greatly increasing the task's execution time [51]. In certain cases, preemption has not a great impact because it happens at a moment of the program's execution in which data stored in the cache are not useful: for instance, before a cache miss. Still, in general, preemption damages program locality and therefore it causes a degradation of system predictability, making WCETs not easy to characterize and predict [52] [53] [51] [50].

6.3 Deterministic memory access with preemptions

We consider programs that perform deterministic accesses to the memory that can be interrupted at any time, using the count of cache misses as a first proxy evaluation of the WCET (or, as the literature calls it, the Cache memory preemption delay). A "dual" approach has already been explored in the literature, with papers about applying static analysis techniques to quantum algorithms to evaluate their performance and formally analyze their functional properties [42], [43]. On the other hand, the applications of quantum computing to improve the static analysis of programs has been only scarcely explored [46] and even lesser so is the issue of static analysis of cache misses and WCET, where only *quantum-inspired* classic algorithm have been proposed [38]. So many questions remain open about the applications and benefits of quantum computing for software engineering issues [46]. Numerous research works exist in improving classical polynomial algorithms with a quantum-inspired approach, allowing to gain a polynomial factor of complexity [21], [54]–[60]. In this chapter, we propose a dynamic programming algorithm for computing the minimal number of cache hits and improve it into a quantum-classic hybrid version, leading to lower complexity.

As a model case, we consider the problem of evaluating the WCET of a deterministic sequence of memory accesses in case of preemptions. We denote the sequence of memory accesses as $a_0, a_1, \ldots, a_{N-1}$ and we are supposed to know if each of them is a cache miss or a cache hit in an execution without preemptions (this is done in linear time as a pre-processing step). We denote $X[i]$ the i-th access of the sequence and

$$X[i] = \begin{cases} 1 \ \textit{if} \text{ cache miss} \\ 0 \ \textit{if} \text{ cache hit.} \end{cases}$$

P is the set of possible contents of those memory accesses $p_0, \ldots, p_{M-1}$. Any memory access a_i belongs to the set P, i.e. $a_i \in P$. The number of preemptions

lines, the situation when one wants to calculate WCET of a task in the general case requires the hypothesis that no useful cache line would be retained by a preemption.

interrupting the program is a fixed number of K. Preemption can happen at any time, leading to the model's non-predictability. We also suppose that when preemption occurs, the cache's content is flushed from it.

6.4 Our classical starting point

We designed a dynamic programming algorithm to compute the minimal number of cache hits while executing a deterministic sequence with K preemptions. The minimal number of cache hits corresponds to the maximal number of cache misses.

First, we scan the sequence of memory accesses, as it would be without any point of preemption, and we store the information of being a cache hit or miss into an array X: if the considered access is cache miss $X[i] = 1$, $X[i] = 0$ otherwise. We also compute the $N_i[p_t] > i$, the table of indexes of the first access to the object p_t, after the i-th access, into the memory access sequence. If there is no other access to p_t, we write ∞.

We suppose that we know the solution of the problem for the sequence $a_{i+1}, \cdots, a_{N-1}$ and all the numbers of preemptions k, where $0 \leq k \leq K$. In other words, it is known:

- $S(i+1, p_t, k)$: the smallest number of cache hits for the object p_t and k preemptions;
- $Y(i+1, k) \geq i+1$: index of the next preemption in this solution.

We want to determine the $S(i, p_t, k)$ and the $Y(i, k), \forall i, k$ from the $S(i+1, p_t, k)$ and $Y(i+1, k)$. Many different cases have to be considered.

6.4.1 Case 1: cache miss

For a given $p_t \in P$ and a given $0 \leq k \leq K$, how to determine $S(i, p_t, k)$ and $Y(i, k)$ from $S(i+1, p_t, k)$ and $Y(i+1, k)$, in the case of a cache miss when executing the linear sequence, without preemptions. In other words, when $X[i] = 1$).

Independently from the previous preemptions that may have happened, when we call the data a_i a cache miss is generated. A preemption point can arrive just before a_i (what we call sub-case β) or not (what we call sub-case α).

Sub-case α

In this case, there is not preemption just before calling a_i. This means that the number of cache hit will not be modified:

$$S(i, p_t, k) = S(i+1, p_t, k) \quad \forall t, k. \tag{6.1}$$

More than that, the index of the next point of preemption is not modified, because the number of preemptions didn't change:

$$Y(i,k) = Y(i+1,k) \quad \forall k. \tag{6.2}$$

Sub-case β

This case describes the situation in which a point of preemption happens, just before calling a_i. As condition we have that $k > 0$. For $k = 0$, we consider $S(i, p_t, 0) = \infty$, for all p_t.
We loose 1 cache hit if the next access to p_t was a cache hit (in the linear sequence X) and if the next preemption is after this access. If the next preemption is before this access, we already subtracted it from the cache hit counter. In symbols:

$$\begin{array}{l} \forall p_t, k \\ \textbf{if } X[N_i[p_t]] = 0 \textbf{ and } Y(i+1,k-1) > N_i[p_t] \\ \quad \textbf{then } S(i,p_t,k) = S(i+1,p_t,k-1) - 1 \\ \quad \textbf{otherwise } S(i,p_t,k) = S(i+1,p_t,k-1) \end{array} \tag{6.3}$$

We have also to change the index of the next preemption:

$$Y(i,k) = i \quad \forall k. \tag{6.4}$$

6.4.2 Case 2: cache hit

For a given $p_t \in P$ and a given $0 \leq k \leq K$, how to determine $S(i, p_t, k)$ and $Y(i, k)$ from $S(i+1, p_t, k)$ and $Y(i+1, k)$, in the case of a cache hit when executing the linear sequence, without preemptions. In other words, when $X[i] = 0$).

A preemption point can arrive just before a_i (what we call the sub-case β) or not (what we call sub-case α).

Sub-case α

In this case, there is no preemption just before calling a_i.
This means that the number of cache hits will not be modified, for all the objects $p_t \in P$ if the considered object p_t it is not the called one a_i. In symbols:

$$\begin{array}{l} \forall k, t \text{ such that } p_t \neq a_i \\ \quad S(i,p_t,k) = S(i+1,p_t,k). \end{array} \tag{6.5}$$

Otherwise, if the considered object p_t is the called element a_i, the counter of cache hits have to be increased by one. Indeed, since no preemption happened and

the object a_i gave a cache hit in the sequential execution, it will gives a cache hit as well. In symbols:

$$S(i, \mathbf{a_i}, k) = 1 + S(i+1, \mathbf{a_i}, k) \quad \forall k. \tag{6.6}$$

We have also that the index of the next preemption does not change, because a preemption did not happen.

$$Y(i,k) = Y(i+1,k) \quad \forall k. \tag{6.7}$$

Sub-case β

This case describes the situation in which a point of preemption happens, just before calling a_i. As condition we have that $k > 0$. For $k = 0$, we consider $S(i, p_t, 0) = \infty$, for all p_t.
The counter of cache hits has to be decreased by one if the next access to p_t was a cache hit (in the linear sequence X) and if the next preemption is after this access. If the next preemption is before this access, we already subtracted it from the cache hit counter. In symbols:

$$\begin{array}{l} \forall k, \forall t \text{ such that } p_t \neq a_i \\ \textbf{if } X[N_i[p_t]] = 0 \textbf{ and } Y(i+1, k-1) > N_i[p_t] \\ \quad \textbf{then } S(i, p_t, k) = S(i+1, p_t, k-1) - 1 \\ \quad \textbf{otherwise } S(i, p_t, k) = S(i+1, p_t, k-1) \end{array} \tag{6.8}$$

and, because of the preemption, the counter of cache hits for the called object a_i become:

$$S(i, a_i, k) = S(i+1, a_i, k-1). \tag{6.9}$$

We have also to change the index of the next preemption:

$$Y(i,k) = i \quad \forall k. \tag{6.10}$$

6.4.3 Algorithm

For both cases 1 and 2, the choice between the sub-case α and the sub-case β is done in order to minimize the number of cache hits. In other words, for a given k we choose the option that minimize:

$$\sum_t S(i, p_t, k). \tag{6.11}$$

In the situation in which the sub-case α and β give the same amount of cache hits, the choice between them is done arbitrarily.

We use those recursion rules to compute the $S(*, p_t, k)$ and the $Y(*, k)$ from right (the end of the sequence) to the left (beginning of the sequence), knowing that that:

- $S(N-1, p_t, k)$ and $Y(N-1, k)$, $\forall k$ are trivial to compute;
- The solution of the problem is the smallest number of cache hits for all the objects and for $k = K$, i.e. $\sum_t S(i, p_t, K)$.

Since M and K are bounded by N, the time and memory complexity of the algorithm is $O(N^3)$

6.4.4 Example of application of the classic algorithm

For example, we consider a cache of size 2, with $K = 2$ (i.e., two preemptions occur). The set of possible contents of the cache is $P = \{p_0, p_1, p_2\} = \{A, B, C\}$. The sequence of memory accesses is $a_0a_1a_2a_3a_4 = ABABC$. The first two accesses and the last one are cache misses, while the third and fourth accesses are cache hits.

The worst case scenario is when the two point of preemptions delete both of the two cache hit, resulting in a sequence of all cache misses. In particular this result is given by different situations:

- ApBpABC
- ABppABC
- ABpApBC
- ABpABpC

where p represents a point of preemption. We can summarize saying that the worst case scenario is given by the situation:

A(miss)B(miss)**p(preemption)**A(miss)B(miss)C(miss).

and the second point of preemption can be anywhere, it will give 0 cache hits anyway. After the algorithm performs the first scan of the sequence of memory accesses, we have $X = 11001$.
The table N is:

$$N = \begin{array}{|c|c|c|c|} \hline \text{i} \setminus p_t & p_0 \text{ (A)} & p_1 \text{ (B)} & p_2 \text{ (C)} \\ \hline 0 & 2 & 1 & 4 \\ 1 & 2 & 3 & 4 \\ 2 & \infty & 3 & 4 \\ 3 & \infty & \infty & 4 \\ 4 & \infty & \infty & \infty \\ \hline \end{array} \tag{6.12}$$

We provide below the details for all the iterations.

First round $i = 4$

The first iteration corresponds to the 4th access into the sequence, i.e. $a_4 = C$. In the linear executions sequence this access corresponds to a cache miss. It is clear that, no matter if a preemption happens or not, the cache hits counter is 0 for all p_t and k.

$$S(4, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash \text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 0 & 0 & 0 \\ p_1 \text{ (B)} & 0 & 0 & 0 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.13}$$

The index of the next preemption is:

$$Y(4, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 4 & 4 \\ \hline \end{array} \tag{6.14}$$

Second round $i = 3$

The second iteration corresponds to the access into the sequence $a_3 = B$. In the linear executions sequence this access corresponds to a cache hit. We will apply the rules described by the Case 2 in 6.4.2. We have to analyze both sub-cases α and β.

- **Case** α **(without preemption)**, we have:
  - $S(3, A, k) = S(4, A, k)$, for all k
  - $S(3, C, k) = S(4, C, k)$, for all k
  - $S(3, B, k) = 1 + S(4, B, k)$, for all k.

$$S(3, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash \text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 0 & 0 & 0 \\ p_1 \text{ (B)} & 1 & 1 & 1 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.15}$$

  Then we have the table of index of the next preemption $Y(i, k) = Y(i+1, k)$, $\forall k$

$$Y(3, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 4 & 4 \\ \hline \end{array} \tag{6.16}$$

- **Case** β **(with preemption)**, we have:

- $S(3, p_t, 0) = \infty$, for all p_t;
- $X[N_3(A)] = X[\infty]$ is not defined, then $S(3, A, k) = S(4, A, k-1)$, for $k = 1, 2$;
- $X[N_3(C)] = X[4] = 1$, then $S(3, C, k) = S(4, C, k-1)$, for $k = 1, 2$;
- $a_3 = B$, then $S(3, B, k) = S(4, B, k-1)$, for $k = 1, 2$.

$$S(3, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash\text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & \infty & 0 & 0 \\ p_1 \text{ (B)} & \infty & 0 & 0 \\ p_2 \text{ (C)} & \infty & 0 & 0 \\ \hline \end{array} \tag{6.17}$$

Then we have the table of index of the next preemption $Y(i, k) = i$, $\forall k$

$$Y(3, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & 3 & 3 & 3 \\ \hline \end{array} \tag{6.18}$$

Final tables $S(i, p_t, k)$ and $Y(i, k)$, made by choosing between the tables of sub-cases, column by column, the option that minimize $\sum_t S(i, p_t, k)$. In this example, we take the first column from the sub-case α (without preemption) and the second and the third ones from the sub-case β (with preemption).

$$S(3, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash\text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 0 & 0 & 0 \\ p_1 \text{ (B)} & 1 & 0 & 0 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.19}$$

$$Y(3, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 3 & 3 \\ \hline \end{array} \tag{6.20}$$

Third round $i = 2$

The third iteration corresponds to the access into the sequence $a_2 = A$. In the linear executions sequence this access corresponds to a cache hit. We will apply the rules described by the Case 2 in 6.4.2. We have to analyze both sub-cases α and β.

- **Case** α **(without preemption)**, we have:
  - $S(2, B, k) = S(3, B, k)$, for all k
  - $S(2, C, k) = S(3, C, k)$, for all k
  - $S(2, A, k) = 1 + S(3, A, k)$, for all k.

$$S(3,p_t,k) = \begin{array}{|l|l|l|l|}\hline \text{t }\backslash\text{k} & 0 & 1 & 2 \\ \hline p_0\ (\text{A}) & 1 & 1 & 1 \\ p_1\ (\text{B}) & 1 & 0 & 0 \\ p_2\ (\text{C}) & 0 & 0 & 0 \\ \hline \end{array} \tag{6.21}$$

Then we have the table of index of the next preemption $Y(i,k) = Y(i+1,k)$, $\forall k$

$$Y(2,k) = \begin{array}{|l|l|l|l|}\hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 3 & 3 \\ \hline \end{array} \tag{6.22}$$

- **Case β (with preemption)**, we have:
  - $S(2,p_t,0) = \infty$, for all p_t;
  - $X[N_2(B)] = X[3] = 0$ and $Y(3,0) = \infty > N_2[B] = 3$ then $S(2,B,1) = S(3,B,0) - 1 = 1 - 1 = 0$;
  - $X[N_2(B)] = X[3] = 0$ and $Y(3,1) = 3 \not> N_2[B] = 3$ then $S(2,B,2) = S(3,B,1) = 0$;
  - $X[N_2(C)] = X[4] = 1$, then $S(2,C,k) = S(3,C,k-1)$, for $k = 1,2$;
  - $a_2 = A$, then $S(2,A,k) = S(3,A,k-1)$, for $k = 1,2$.

$$S(2,p_t,k) = \begin{array}{|l|l|l|l|}\hline \text{t }\backslash\text{k} & 0 & 1 & 2 \\ \hline p_0\ (\text{A}) & \infty & 0 & 0 \\ p_1\ (\text{B}) & \infty & 0 & 0 \\ p_2\ (\text{C}) & \infty & 0 & 0 \\ \hline \end{array} \tag{6.23}$$

Then we have the table of index of the next preemption $Y(i,k) = i$, $\forall k$

$$Y(2,k) = \begin{array}{|l|l|l|l|}\hline \text{k} & 0 & 1 & 2 \\ \hline & 2 & 2 & 2 \\ \hline \end{array} \tag{6.24}$$

Final tables $S(i,p_t,k)$ and $Y(i,k)$, made by choosing between the tables of sub-cases, column by column, the option that minimize $\sum_t S(i,p_t,k)$. In this example, we take the first column from the sub-case α (without preemption) and the second and the third ones from the sub-case β (with preemption).

$$S(2,p_t,k) = \begin{array}{|l|l|l|l|}\hline \text{t }\backslash\text{k} & 0 & 1 & 2 \\ \hline p_0\ (\text{A}) & 1 & 0 & 0 \\ p_1\ (\text{B}) & 1 & 0 & 0 \\ p_2\ (\text{C}) & 0 & 0 & 0 \\ \hline \end{array} \tag{6.25}$$

$$Y(2,k) = \begin{array}{|l|l|l|l|}\hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 2 & 2 \\ \hline \end{array} \tag{6.26}$$

Fourth round $i = 1$

The fourth iteration corresponds to the access into the sequence $a_1 = B$. In the linear executions sequence this access corresponds to a cache miss. We will apply the rules described by the Case 1 in 6.4.1. We have to analyze both sub-cases α and β.

- **Case** α **(without preemption)**, we have:
  - $S(1, A, k) = S(2, A, k)$, for all k.
  - $S(1, B, k) = S(2, B, k)$, for all k
  - $S(1, C, k) = S(2, C, k)$, for all k

$$S(1, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash \text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 1 & 0 & 0 \\ p_1 \text{ (B)} & 1 & 0 & 0 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.27}$$

Then we have the table of index of the next preemption $Y(i, k) = Y(i+1, k)$, $\forall k$

$$Y(1, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 2 & 2 \\ \hline \end{array} \tag{6.28}$$

- **Case** β **(with preemption)**, we have:
  - $S(1, p_t, 0) = \infty$, for all p_t;
  - $X[N_1(A)] = X[2] = 0$ and $Y(2, 0) = \infty > N_1[A] = 2$ then $S(1, A, 1) = S(2, A, 0) - 1 = 1 - 1 = 0$;
  - $X[N_1(A)] = X[2] = 0$ and $Y(2, 1) = 2 \not> N_1[A] = 2$ then $S(1, A, 2) = S(2, A, 1) = 0$;
  - $X[N_1(B)] = X[3] = 0$ and $Y(2, 0) = \infty > N_1[B] = 3$ then $S(1, B, 1) = S(2, B, 0) - 1 = 1 - 1 = 0$;
  - $X[N_1(B)] = X[3] = 0$ and $Y(2, 1) = 2 \not> N_1[A] = 3$ then $S(1, B, 2) = S(2, B, 1) = 0$;
  - $X[N_1(C)] = X[4] = 1$, then $S(1, C, k) = S(2, C, k - 1)$, for $k = 1, 2$;

$$S(1, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash \text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & \infty & 0 & 0 \\ p_1 \text{ (B)} & \infty & 0 & 0 \\ p_2 \text{ (C)} & \infty & 0 & 0 \\ \hline \end{array} \tag{6.29}$$

Then we have the table of index of the next preemption $Y(i,k) = i$, $\forall k$

$$Y(1,k) = \begin{array}{|c|c|c|c|}\hline \text{k} & 0 & 1 & 2 \\ \hline & 1 & 1 & 1 \\ \hline \end{array} \tag{6.30}$$

Final tables $S(i,p_t,k)$ and $Y(i,k)$, made by choosing between the tables of sub-cases, column by column, the option that minimize $\sum_t S(i,p_t,k)$. In this example, we take the first column from the sub-case α (without preemption) and the second and the third ones from the sub-case β (with preemption).

$$S(1,p_t,k) = \begin{array}{|l|l|l|l|}\hline \text{t \textbackslash k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 1 & 0 & 0 \\ p_1 \text{ (B)} & 1 & 0 & 0 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.31}$$

$$Y(1,k) = \begin{array}{|c|c|c|c|}\hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 1 & 1 \\ \hline \end{array} \tag{6.32}$$

Fifth round $i = 0$

The fifth iteration corresponds to the access into the sequence $a_0 = A$. In the linear executions sequence this access corresponds to a cache miss. We will apply the rules described by the Case 1 in 6.4.1. We have to analyze both sub-cases α and β.

- **Case α (without preemption)**, we have:
  - $S(0,A,k) = S(1,A,k)$, for all k;
  - $S(0,B,k) = S(1,B,k)$, for all k:
  - $S(0,C,k) = S(1,C,k)$, for all k

$$S(0,p_t,k) = \begin{array}{|l|l|l|l|}\hline \text{t \textbackslash k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 1 & 0 & 0 \\ p_1 \text{ (B)} & 1 & 0 & 0 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.33}$$

Then we have the table of index of the next preemption $Y(i,k) = Y(i+1,k)$, $\forall k$

$$Y(0,k) = \begin{array}{|c|c|c|c|}\hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 1 & 1 \\ \hline \end{array} \tag{6.34}$$

- **Case β (with preemption)**, we have:

- $S(0, p_t, 0) = \infty$, for all p_t;
- $X[N_0(A)] = X[2] = 0$ and $Y(1, 0) = \infty > N_0[A] = 2$ then $S(0, A, 1) = S(1, A, 0) - 1 = 1 - 1 = 0$;
- $X[N_0(A)] = X[2] = 0$ and $Y(1, 1) = 1 \not> N_0[A] = 2$ then $S(0, A, 2) = S(1, A, 1) = 0$;
- $X[N_0(B)] = X[1] = 1$ then $S(0, B, k) = S(1, B, k - 1)$, for $k = 1, 2$;
- $X[N_1(C)] = X[4] = 1$, then $S(1, C, k) = S(2, C, k - 1)$, for $k = 1, 2$;

$$S(0, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash \text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & \infty & 0 & 0 \\ p_1 \text{ (B)} & \infty & 1 & 0 \\ p_2 \text{ (C)} & \infty & 0 & 0 \\ \hline \end{array} \tag{6.35}$$

Then we have the table of index of the next preemption $Y(i, k) = i$, $\forall k$

$$Y(0, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & 0 & 0 & 0 \\ \hline \end{array} \tag{6.36}$$

Final tables $S(i, p_t, k)$ and $Y(i, k)$, made by choosing between the tables of sub-cases, column by column, the option that minimize $\sum_t S(i, p_t, k)$. In this example, we take the first and second column from the sub-case α (without preemption) and the third one from the sub-case β (with preemption).

$$S(0, p_t, k) = \begin{array}{|l|l|l|l|} \hline \text{t } \backslash \text{k} & 0 & 1 & 2 \\ \hline p_0 \text{ (A)} & 1 & 0 & 0 \\ p_1 \text{ (B)} & 1 & 0 & 0 \\ p_2 \text{ (C)} & 0 & 0 & 0 \\ \hline \end{array} \tag{6.37}$$

$$Y(0, k) = \begin{array}{|l|l|l|l|} \hline \text{k} & 0 & 1 & 2 \\ \hline & \infty & 1 & 0 \\ \hline \end{array} \tag{6.38}$$

The solution is given by the sum of the values in the last column of the last table. Then,the minimal number of cache hits for the sequence $ABABC$ is $0 + 0 + 0 = 0$ cache hits.

6.5 Going quantum

This section presents the steps towards a *quantum version* of the previous algorithm. We already know the cache's behavior in case of the program's execution without preemptions (i.e., the array X) and the table N, both deterministic. While searching for the minimal number of cache misses, in the classic algorithm, we perform three nested loops:

- over the sequence memory accesses (N iterations)
- over all the possible objects that can be into the cache (M iterations)
- over all the k with $0 \leq k \leq K$ ($K+1$ iterations)

To compute the solution (i.e., the number of cache hits corresponding to K preemption and the last memory access), the algorithm needs all the number of cache hits corresponding to k with $0 \leq k \leq K$. We reformulated the dynamic programming classic algorithm into a hybrid quantum-classical algorithm in which the third loop (the one over the number of preemptions) is suitable to be executed on a quantum computer.

6.5.1 Superpositions construction

To explain the construction of the superpositions in the quantum version of the previous algorithm, we exploit the previous example to make easier going through the details of the algorithm and give a glimpse of its functioning.
We recall from the previous example that the sequence of accesses is $a_0a_1a_2a_3a_4 = ABABC$. The set of object is $P = \{A, B, C\}$. We build a superposed state represented as $|k, S\rangle$ where k is the number of considered preemptions and S is the corresponding number of cache hits. In other words, for each number of preemptions k, it is known the minimum possible number of cache hits while performing the sequence of memory accesses. We have to consider $k = \{0, 1, 2\}$. The number of cache hits, for this example, can be equal to $\{0, 1\}$. Thus, for our example, we consider the following table for the first round of the algorithm (i.e. when $i = 4$).

A	$\frac{1}{\sqrt{3}}\|0,0\rangle + 0\|0,1\rangle + \frac{1}{\sqrt{3}}\|1,0\rangle + 0\|1,1\rangle + \frac{1}{\sqrt{3}}\|2,0\rangle + 0\|2,1\rangle$
B	$\frac{1}{\sqrt{3}}\|0,0\rangle + 0\|0,1\rangle + \frac{1}{\sqrt{3}}\|1,0\rangle + 0\|1,1\rangle + \frac{1}{\sqrt{3}}\|2,0\rangle + 0\|2,1\rangle$
C	$\frac{1}{\sqrt{3}}\|0,0\rangle + 0\|0,1\rangle + \frac{1}{\sqrt{3}}\|1,0\rangle + 0\|1,1\rangle + \frac{1}{\sqrt{3}}\|2,0\rangle + 0\|2,1\rangle$

For $i = 3$, we have $a_3 = B$ that corresponds to a cache hit. To pass from table for $i = 4$ to the table for $i = 3$, we switch the two first entries for the object B (we have indeed a cache hit when calling the second B, when the number of preemption k is 0). The table for $i = 3$ is:

A	$\frac{1}{\sqrt{3}}\|0,0\rangle + 0\|0,1\rangle + \frac{1}{\sqrt{3}}\|1,0\rangle + 0\|1,1\rangle + \frac{1}{\sqrt{3}}\|2,0\rangle + 0\|2,1\rangle$
B	$0\|0,0\rangle + \frac{1}{\sqrt{3}}\|0,1\rangle + \frac{1}{\sqrt{3}}\|1,0\rangle + 0\|1,1\rangle + \frac{1}{\sqrt{3}}\|2,0\rangle + 0\|2,1\rangle$
C	$\frac{1}{\sqrt{3}}\|0,0\rangle + 0\|0,1\rangle + \frac{1}{\sqrt{3}}\|1,0\rangle + 0\|1,1\rangle + \frac{1}{\sqrt{3}}\|2,0\rangle + 0\|2,1\rangle$

6.5.2 Case 1: a cache miss happens

Let us analyze the case in which a cache miss happens (i.e. in the case $X[i] = 1$), see the classical counterpart in 6.4.1.

Sub-case α: no preemption

We recall that, from equation 6.1, the sub-case α that describes the case in which no preemption happens just before the called data a_i corresponds to perform $S(i, p_t, k) = S(i+1, p_t, k), \forall t, k$. This corresponds to applying an identity matrix (for our example, of dimension 6) for each object in P.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.39}$$

For instance, if we are in the sub-case α for a input superposition, for an object, as:

$$\frac{1}{\sqrt{3}} |0,0\rangle + 0 |0,1\rangle + \frac{1}{\sqrt{3}} |1,0\rangle + 0 |1,1\rangle + \frac{1}{\sqrt{3}} |2,0\rangle + 0 |2,1\rangle \tag{6.40}$$

at the next iteration, we obtain the same superposition. In other words, we appliedy the identity matrix I_6, for each object, on the corresponding vector of amplitudes, in other to pass to the next iteration:

$$\begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} \tag{6.41}$$

Sub-case β: preemption

For the sub-case β, we recall equation 6.3. In this sub-case, we have a condition:

$$\textbf{if } X[N_i[p_t]] = 0 \textbf{ and } Y(i+1, k-1) > N_i[p_t] \tag{6.42}$$

- If this condition in 6.42 is FALSE we apply:

$$S(i, p_t, k) = S(i+1, p_t, k-1). \tag{6.43}$$

To perform the equation 6.43, for each object p_t, we define the operator:

$$D = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.44}$$

It is easy to see that the operator D is not more than an permutation from the values of $S(i+1, p_t, k-1)$ to $S(i, p_t, k)$, for an object $p_t \in P$. For instance, if we are in the sub-case β, where the condition 6.42 is false for a input superposition as:

$$\frac{1}{\sqrt{3}}|0,0\rangle + 0\,|0,1\rangle + \frac{1}{\sqrt{3}}|1,0\rangle + 0\,|1,1\rangle + \frac{1}{\sqrt{3}}|2,0\rangle + 0\,|2,1\rangle \tag{6.45}$$

we obtain again, for this case, the same superposition. In other words, we apply the D matrix on the corresponding vector of amplitudes:

$$\begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} \tag{6.46}$$

and we obtain:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} \tag{6.47}$$

- If otherwise the condition 6.42 is TRUE, we apply:

$$S(i, p_t, k) = S(i+1, p_t, k-1) - 1 \tag{6.48}$$

To perform the equation, for our example, 6.48 we define the operator G, to

apply after the operator D, where:

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{6.49}$$

For instance, if we are in the sub-case β, where the condition 6.42 is true for a input superposition as:

$$0\,|0,0\rangle + \frac{1}{\sqrt{3}}\,|0,1\rangle + 0\,|1,0\rangle + \frac{1}{\sqrt{3}}\,|1,1\rangle + 0\,|2,0\rangle + \frac{1}{\sqrt{3}}\,|2,1\rangle \tag{6.50}$$

we obtain:

$$\frac{1}{\sqrt{3}}\,|0,0\rangle + 0\,|0,1\rangle + \frac{1}{\sqrt{3}}\,|1,0\rangle + 0\,|1,1\rangle + \frac{1}{\sqrt{3}}\,|2,0\rangle + 0\,|2,1\rangle\,. \tag{6.51}$$

In other words, we applied the G matrix, after the D matrix, on the corresponding vector of amplitudes:

$$\begin{bmatrix} 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \end{bmatrix} \tag{6.52}$$

We obtain:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} \tag{6.53}$$

Please note that the matrix G has the shape of a permutation because it is necessary to maintain the reversibility. This does not represent a problem because in the classical algorithm we never perform 6.48 when $S(i+1, p_t, k-1) = 0$. To clarify, if we consider another example as:

$$0\,|0,0\rangle + 0\,|0,1\rangle + \frac{1}{\sqrt{3}}\,|0,2\rangle + 0\,|1,0\rangle + 0\,|1,1\rangle + \frac{1}{\sqrt{3}}\,|1,2\rangle + 0\,|2,0\rangle + 0\,|2,1\rangle + \frac{1}{\sqrt{3}}\,|2,2\rangle\,. \tag{6.54}$$

In this case, the matrix G would be:

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{6.55}$$

and we would obtain:

$$0\,|0,0\rangle + \frac{1}{\sqrt{3}}\,|0,1\rangle + 0\,|0,2\rangle + 0\,|1,0\rangle + \frac{1}{\sqrt{3}}\,|1,1\rangle + 0\,|1,2\rangle + 0\,|2,0\rangle + \frac{1}{\sqrt{3}}\,|2,1\rangle + 0\,|2,2\rangle\,. \tag{6.56}$$

The Case 2 (if a cache hit happens) is quite similar, we are not developing the details.

6.5.3 Choice between the sub-case α and the sub-case β

In the Algorithm, at the end of each round we need to choose between the output of the sub-case α and the output of the sub-case β. In the classical algorithm, for a given k, we take the option that minimize $\sum_t S(i, p_t, k)$. We now provide an equivalent, suitable to be implemented in a quantum circuit. For simplicity, let us consider a simplified example, with respect the previous one. We suppose to have only two elements to consider A and B. We suppose to have again $k = \{0, 1, 2\}$ and $S = \{0, 1\}$. The superpositions describing the number of cache hits for all the possible k-s for the two elements A, B are, for instance:

A	$0\|0,0\rangle + \frac{1}{\sqrt{3}}\,\|0,1\rangle + \frac{1}{\sqrt{3}}\,\|1,0\rangle + 0\,\|1,1\rangle + \frac{1}{\sqrt{3}}\,\|2,0\rangle + 0\,\|2,1\rangle$
B	$\frac{1}{\sqrt{3}}\,\|0,0\rangle + 0\,\|0,1\rangle + \frac{1}{\sqrt{3}}\,\|1,0\rangle + 0\,\|1,1\rangle + \frac{1}{\sqrt{3}}\,\|2,0\rangle + 0\,\|2,1\rangle$

The two superposition are independent and we can perform the tensor product:

$$\begin{bmatrix} 0 \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \\ |20\rangle \\ |21\rangle \end{matrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \\ \frac{1}{\sqrt{3}} \\ 0 \end{bmatrix} \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \\ |20\rangle \\ |21\rangle \end{matrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \frac{1}{3} \\ 0 \\ \frac{1}{3} \\ 0 \\ \frac{1}{3} \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} |0000\rangle \\ |0001\rangle \\ |0010\rangle \\ |0011\rangle \\ |0020\rangle \\ |0021\rangle \\ |0100\rangle \\ |0101\rangle \\ |0110\rangle \\ |0111\rangle \\ |0120\rangle \\ |0121\rangle \\ \vdots \end{matrix}. \tag{6.57}$$

If we write the states of the superposition for an element as:

$$|\text{index}_{\text{element}}, \text{value}_{\text{element}}\rangle \tag{6.58}$$

we can express the states of the resulting superposition of the tensor product in 6.57 as:

$$|i_A v_A i_B v_B\rangle \tag{6.59}$$

We suppose now to have to choose between the output of the sub-case α and the output of the sub-case β. We represent the states of the superposition resulting as output of the sub-case α as:

$$|i_A v_A i_B v_B\rangle \,. \tag{6.60}$$

In particular, for $i_A = i_B = l$, where l is a fixed value of k, the only non-zero amplitudes are those with states:

$$|lt_A[l] lt_B[l]\rangle \,. \tag{6.61}$$

We represent the states of the superposition resulting as output of the sub-case β as:

$$|i'_A v'_A i'_B v'_B\rangle \,. \tag{6.62}$$

In particular, for $i'_A = i'_B = l$, where l is a fixed value of k, the only non-zero amplitudes are those with states:

$$|lt'_A[l] lt'_B[l]\rangle \,. \tag{6.63}$$

If we perform the tensor product between the superposition resulting as output from the sub-case α and the superposition resulting as output from the sub-case β we obtain a superposition which states can be represented as:

$$|i_A v_A i_B v_B i'_A v'_A v'_B v'_B\rangle \,. \tag{6.64}$$

Then, for $i_A = i_B = i'_A = i'_B = l$ where l is a fixed value of k, the only non-zero amplitudes in the superposition resulting as output of the tensor product are those with states:

$$|lt_A[l]lt_B[l]lt'_A[l]lt'_B[l]\rangle\,. \tag{6.65}$$

The choice between the option α and the option β is done performing the following comparison:

$$\text{COMPARE}(t_A[l] + t_B[l], t'_A[l] + t'_B[l]) \tag{6.66}$$

choosing the option that minimize the sum.

Example

To clarify, let us consider an example. We can rewrite the superposition for the element A such as:

$$\begin{aligned} 0\,|0,0\rangle + \frac{1}{\sqrt{3}}\,|0,1\rangle + 0\,|1,0\rangle + \frac{1}{\sqrt{3}}\,|1,1\rangle + \frac{1}{\sqrt{3}}\,|2,0\rangle + 0\,|2,1\rangle = \\ \frac{1}{\sqrt{3}}\,|0,t[0]\rangle + \frac{1}{\sqrt{3}}\,|1,t[1]\rangle + \frac{1}{\sqrt{3}}\,|2,t[2]\rangle\,. \end{aligned} \tag{6.67}$$

In other words, 6.67 represents the superposition for the array:

$$\begin{cases} t[0] = 1 \\ t[1] = 1 \\ t[2] = 0. \end{cases} \tag{6.68}$$

Note that he array in 6.68 corresponds to the first line of the matrix in 6.13. We can rewrite the superposition for the element B such as:

$$\begin{aligned} \frac{1}{\sqrt{3}}\,|0,0\rangle + 0\,|0,1\rangle + 0\,|1,0\rangle + \frac{1}{\sqrt{3}}\,|1,1\rangle + \frac{1}{\sqrt{3}}\,|2,0\rangle + 0\,|2,1\rangle = \\ \frac{1}{\sqrt{3}}\,|0,s[0]\rangle + \frac{1}{\sqrt{3}}\,|1,s[1]\rangle + \frac{1}{\sqrt{3}}\,|2,s[2]\rangle \end{aligned}\,. \tag{6.69}$$

In other words, 6.69 represents the superposition for the array:

$$\begin{cases} s[0] = 0 \\ s[1] = 1 \\ s[2] = 0. \end{cases} \tag{6.70}$$

Note that he array in 6.68 corresponds to the second line of the matrix in 6.13. If now we perform the tensor product between the two superpositions 6.67 and 6.69

we obtain:

$$\begin{aligned}\frac{1}{3}(&|0,t[0],0,s[0]\rangle + |0,t[0],1,s[1]\rangle + |0,t[0],2,s[2]\rangle + \\ &+ |1,t[1],0,s[0]\rangle + |1,t[1],1,s[1]\rangle + |1,t[1],2,s[2]\rangle + \\ &+ |2,t[2],0,s[0]\rangle + |2,t[2],1,s[1]\rangle + |2,t[2],2,s[2]\rangle)\end{aligned} \tag{6.71}$$

The only states (with amplitude non-zero) that matter to us (those with $i_A = i_B$) are:

$$\begin{cases} |0,t[0],0,s[0]\rangle \\ |1,t[1],1,s[1]\rangle \\ |2,t[2],2,s[2]\rangle \end{cases} \tag{6.72}$$

Equation 6.72 express the states with non-zero amplitude and $i_A = i_B$ for the superposition resulting as output of the sub-case α. For the sub-case β, for we will have a similar result. Equation 6.73 express the states with non-zero amplitude and $i'_A = i'_B$ for the superposition resulting as output of the sub-case β.

$$\begin{cases} |0,t'[0],0,s'[0]\rangle \\ |1,t'[1],1,s'[1]\rangle \\ |2,t'[2],2,s'[2]\rangle \end{cases} \tag{6.73}$$

We perform the tensor product between the superposition produced by the sub-case α and the superposition produced by the sub-case β. We obtain a resulting superposition which states are:

$$\begin{cases} |0,t[0],0,s[0],0,t'[0],0,s'[0]\rangle \\ |0,t[0],0,s[0],1,t'[1],1,s'[1]\rangle \\ \vdots \\ |1,t[1],1,s[1],0,t'[0],0,s'[0]\rangle \\ |1,t[1],1,s[1],1,t'[1],1,s'[1]\rangle \\ \vdots \\ |2,t[2],2,s[2],2,t'[2],2,s'[2]\rangle \end{cases} \tag{6.74}$$

In particular, the resulting superposition is:

$$\begin{aligned}\frac{1}{9}(&|0,t[0],0,s[0],0,t'[0],0,s'[0]\rangle + |0,t[0],0,s[0],1,t'[1],1,s'[1]\rangle + \dots \\ &+ |1,t[1],1,s[1],0,t'[0],0,s'[0]\rangle + |1,t[1],1,s[1],1,t'[1],1,s'[1]\rangle + \dots \\ &+ |2,t[2],2,s[2],2,t'[2],2,s'[2]\rangle)\end{aligned} \tag{6.75}$$

The only states that matter to us are those with non-zero amplitudes, when $i_A = i_B = i'_A = i'_B$:

$$\begin{cases} |0, t[0], 0, s[0], 0, t'[0], 0, s'[0]\rangle \\ |1, t[1], 1, s[1], 1, t'[1], 1, s'[1]\rangle \\ |2, t[2], 2, s[2], 2, t'[2], 2, s'[2]\rangle \end{cases} \tag{6.76}$$

We need an auxiliary qubit to store the boolean value resulting from the evaluation of the condition and make the choice between the sub-case α and the sub-case β a reversible operation. Let us perform a small change of notation to take into consideration the boolean value b. We consider from now on $t^{(b)}$ for the element A and $s^{(b)}$ for the element B. Thus, we have:

$$\begin{cases} t^{(0)} = t \\ t^{(1)} = t' \end{cases} \tag{6.77}$$

and

$$\begin{cases} s^{(0)} = s \\ s^{(1)} = s' \end{cases} \tag{6.78}$$

Then, for a fixed k, from 6.76 and considering the new notation in 6.77 and 6.78, we have:

$$\left|k, t^{(b)}[k], k, s^{(b)}[k], k, t^{(1-b)}[k], k, s^{(1-b)}[k], b\right\rangle. \tag{6.79}$$

Let us suppose, for instance, $b = |0\rangle$. If $t^{(0)}[k] + s^{(0)}[k] < t^{(1)}[k] + s^{(1)}[k]$, then $b = |0\rangle$ and the state stays as it is. In the final output, for the considered k, we will have :

$$\left|k, t^{(0)}[k], k, s^{(0)}[k], k, t^{(1)}[k], k, s^{(1)}[k], 0\right\rangle \tag{6.80}$$

Otherwise if $t^{(1)}[k] + s^{(1)}[k] < t^{(0)}[k] + s^{(0)}[k]$,then $b = |1\rangle$ and for the considered k we obtain:

$$\left|k, t^{(1)}[k], k, s^{(1)}[k], k, t^{(0)}[k], k, s^{(0)}[k], 1\right\rangle \tag{6.81}$$

For the next iteration, we work on the states without considering the qubit b. In this way, the size of states increases but not size of computations. For the other states in 6.75 such that it is not true that $i_A = i_B = i'_A = i'_B$, we simply consider that the circuit returns an arbitrary value of b (e.g. $b = |0\rangle$) since we are not interested in those states.

Example

As example, we suppose that we have to choose between the superposition produced by the sub-case α and the one produced by the sub-case β. We consider the tensor product of those two and the resulting superposition. The only states

in the resulting superposition that have non-zero amplitude and that respect our desired condition of equality between indices are :

$$\begin{cases} \left|0, t^{(0)}[0], 0, s^{(0)}[0], 0, t^{(1)}[0], 0, s^{(1)}[0], 0\right\rangle \\ \left|1, t^{(0)}[1], 1, s^{(0)}[1], 1, t^{(1)}[1], 1, s^{(1)}[1], 0\right\rangle \\ \left|2, t^{(0)}[2], 2, s^{(0)}[2], 2, t^{(1)}[2], 2, s^{(1)}[2], 0\right\rangle. \end{cases} \tag{6.82}$$

Then, for example, we suppose that:

$$\begin{cases} t^{(0)}[0] + s^{(0)}[0] < t^{(1)}[0] + s^{(1)}[0] \\ t^{(1)}[1] + s^{(1)}[1] < t^{(0)}[1] + s^{(0)}[1] \\ t^{(0)}[2] + s^{(0)}[2] < t^{(1)}[2] + s^{(1)}[2]. \end{cases} \tag{6.83}$$

Thus, the circuit described above operates on the states and in the result we have as states of matter the following ones:

$$\begin{cases} \left|0, t^{(0)}[0], 0, s^{(0)}[0], 0, t^{(1)}[0], 0, s^{(1)}[0], 0\right\rangle \\ \left|1, t^{(1)}[1], 1, s^{(1)}[1], 1, t^{(0)}[1], 1, s^{(0)}[1], 1\right\rangle \\ \left|2, t^{(0)}[2], 2, s^{(0)}[2], 2, t^{(1)}[2], 2, s^{(1)}[2], 0\right\rangle. \end{cases} \tag{6.84}$$

6.5.4 Some clarifications on the algorithm

While describing the algorithm we discussed the two possible sub-cases that may happen when executing a program. A point of preemption may (sub-case β) or may not (sub-case α) happen just before performing a memory access. We presented two cases as we computed both of them, over duplicating the qubits, to simplify the description. However, such simplification does not reflect a necessary step to perform into the algorithm. Indeed, at each iteration of the algorithm we can rather compute:

$$x = cf(z) + (1 - c)g(z) \tag{6.85}$$

where x is the output superposition, z is the input superposition, c is the condition that chooses between the sub-case α and the sub-cases β, $f(z)$ is the function that does not modify the superposition (sub-case α) and finally $g(z)$ is the function performing the SWAP between the qubits (sub-case β).

While executing the algorithm, we use auxiliary qubits. We can choose to keep these auxiliary qubits unchanged. Alternatively, since we only perform reversible operations and if there is an incentive to do it, it is always possible to de-intricate these qubits, by performing a correct set of operations.

6.5.5 Post-processing

After applying the previous quantum-algorithm, for each k, we have a superposition of states $|k, S_{min}(k)\rangle$ where $S_{min}(k)$ is the minimal number of cache hits if k preemptions occur and $0 \leq k \leq K$. We need to extrapolate the minimal number of cache hits corresponding to K preemption from superposition (i.e. $S_{min}(K)$.

In particular, we expect to have as input for the oracle the superposition:

$$\begin{aligned} &0\,|0,0\rangle + 0\,|0,1\rangle + \cdots + \alpha_0\,|0, S_{min}(0)\rangle + \cdots \\ &+ 0\,|k, N-1\rangle + 0\,|k,0\rangle + 0\,|k,1\rangle + \cdots + \alpha_k\,|k, S_{min}(k)\rangle + \cdots + 0\,|k, N-1\rangle + \cdots \\ &+ 0\,|K,0\rangle + 0\,|K,1\rangle + \cdots + \alpha_K\,|K, S_{min}(K)\rangle + \cdots + 0\,|K, N-1\rangle \end{aligned} \tag{6.86}$$

where $\alpha_0, \ldots, \alpha_K$ are the only non-zero amplitudes. We have that $\alpha_0 = \alpha_1 = \cdots = \alpha_K = \frac{1}{\sqrt{K+1}}$. Thus, the superposition in 6.86 omitting the states with zero amplitude, is:

$$\frac{1}{\sqrt{K+1}}\,|0, S_{min}(0)\rangle + \cdots + \frac{1}{\sqrt{K+1}}\,|k, S_{min}(k)\rangle + \cdots + \frac{1}{\sqrt{K+1}}\,|K, S_{min}(K)\rangle \tag{6.87}$$

We suppose then to build an oracle O_G that selects the states with *index* corresponding to K, in order to isolate the value of interest $S_{min}(K)$ (i.e. the number of cache hits for K preemptions). If, at the end of the algorithm, the result is $|x, y\rangle$ where $x \neq K$, it means that the searching algorithm did not work, and we should apply the algorithm again. The final result is the sum, for each element in P, of its relative $S_{min}(K)$.

Example of oracle O_G

When applying the classical dynamic programming algorithm to the example proposed in 6.4.4, at the end of algorithm we obtain as result the table in 6.37. We expect, for the same example, the quantum version of the algorithm to return a uniform superposition for the element A such as:

$$\frac{1}{\sqrt{3}}\,|0,1\rangle + \frac{1}{\sqrt{3}}\,|1,0\rangle + \frac{1}{\sqrt{3}}\,|2,0\rangle\,. \tag{6.88}$$

An amplitude amplification procedure amplifies the amplitude of the marked item, so that measuring the final state will return the right item with near-certainty. For our example, the target item is $|2,0\rangle$. So we must apply a sign inversion on this specific component of the state, as shown below:

$$\frac{1}{\sqrt{3}}\,|0,1\rangle + \frac{1}{\sqrt{3}}\,|1,0\rangle - \frac{1}{\sqrt{3}}\,|2,0\rangle \tag{6.89}$$

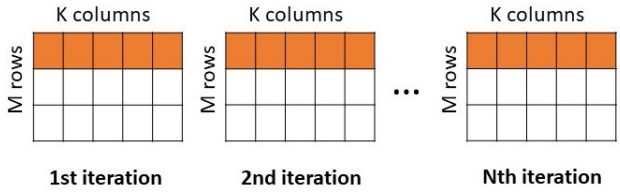
and then perform an inversion about the mean:

$$-0.19\,|0,1\rangle + -0.19\,|1,0\rangle + 0.96\,|2,0\rangle\,. \tag{6.90}$$

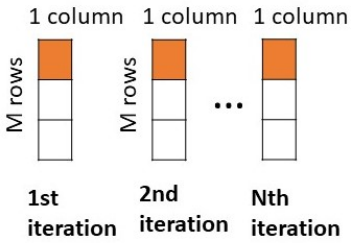
Thus, measuring the final state yields a probability $(0.96)^2 = 0.92$ of obtaining the expected final state $|2,0\rangle$

6.6 Complexity analysis

In order to compute the final solution, at each iteration of the first loop, we create a table of dimensions $M \times (K+1)$, containing the number of the cache hits for each item in P and each k, with $0 \leq k \leq K$. Each iteration of the second loop is independent of the others, meaning that we can compute any row of the next table independently from the other rows. Then, the complexity of "updating" a row (most inner loop) is $O(K)$. Therefore the dynamic programming classical algorithm has a complexity of $O(M \times N \times K)$ (see Fig 6.1a). In the context of porting classical applications to quantum computing, we designed a "hybrid quantum-classical version" of the previous algorithm, whose output is the number of cache hits corresponding to K preemption. We consider a row in one table as a superposition of states representing all the numbers of preemptions k and all the possible values of the number of cache hits. In this way, the complexity of the act of "updating" a row becomes $O(1)$ (see Fig. 6.1b). Therefore, the complexity of the resulting quantum algorithm becomes $O(M \times N)$. As a drawback, we need to post-process the result we obtained in the previous step: superpositions of dimension $K \times U$, where $U < N$ is the maximum number of cache hits. However, we only need the value of the number of cache hits when K preemptions occur (i.e., only one value per row in the last set of rows). Therefore, we can use Grover's algorithm for each row to extract the value of interest in the associated superposition. This means that this post-processing has complexity $\sqrt{K \times U} < \sqrt{K \times N} < \sqrt{N^2} = N$. Hence, the complexity of the whole quantum algorithm is decreased to $O(N^2 + N)$ compared to $O(N^3)$ for the classical algorithm.



(a) Classic algorithm



(b) Quantum algorithm

Figure 6.1: In orange: the number of cache hits for one object in P and all k'

6.7 Conclusions and perspectives

In this chapter, as a first step to deal with WCET-related problems employing quantum-classic hybrid algorithms, we worked on a highly simplified program model since we just considered single-path programs, yet in the presence of arbitrary preemptions. A natural perspective would be to generalize this approach to more complex program models allowing for some non-determinism in the control flow. However, such a generalization would require careful control of the size of the quantum state representing the cache hit counters array to avoid a complexity blow-up at the post-processing amplification step.

Also, we have investigated how to port a polynomial-time dynamic programming algorithm to the quantum framework. In essence, most such algorithms follow the same regular pattern of several nested loops updating an array data structure with the final result obtained in only one of the entries of the last array. As such, the approach in this chapter consists in turning the inner loop of our algorithm into a *quantum parallel for* associated with a Grover-style amplification on the *single* result of interest. Consequently, the approach in this chapter may potentially be generalized to other (exponential-time or polynomial-time) dynamic programming algorithms to derive polynomial quantum speedups.

Chapter 7

Quantum approaches for WCET-related optimization problems

This chapter explores the potential of quantum computing on a WCET[1]-related combinatorial optimization problem applied to a set of several polynomial special cases. We consider the maximization problem of determining the most expensive path in a control flow graph. In these graphs, vertices represent blocks of code whose execution times are fixed and known in advance. We port the considered optimization problem to the quantum framework by expressing it as a QUBO. We then experimentally compare the performances in solving the problem of classic Simulated Annealing (SA), Quantum Annealing (QA), and Quantum Approximate Optimization Algorithm (QAOA). Our experiments suggest that QA represents a fast equivalent of simulated annealing. Indeed, we measured the approximation ratio on the results of QA and SA, showing that their performances are comparable, at least on our set of simplified problems. This chapter corresponds a paper published in the proceedings of the International Conference on Computational Science (ICCS) [2].

7.1 Introduction

The interest given to quantum computing primarily comes from the ability of qubits to store a superposition state that reflects all the possible inputs and outputs

[1]Worst-Case Execution Time (of a program).

[2]Gabriella Bettonte, Valentin Gilbert, Daniel Vert, Stéphane Louise, Renaud Sirdey. Quantum approaches for WCET-related optimization problems, ICCS 2022, Jun 2022, London, United Kingdom.

of a given algorithm until a measurement is performed. This property is called *quantum parallelism* and can, in certain cases [3], [4], give a performance boost for quantum algorithms. However, the advantages of quantum computing do not come without caveats. Only some classes of problems can be solved by quantum computing, with a definite efficiency increase compared to classical computing [61]. One crucial research issue related to quantum computing is determining with precision which problems are better solved by means of quantum approaches [61].

This chapter explores the potential of quantum computing by examining problems involved with determining the Worst-Case Execution Time (WCET) of a restricted set of programs. The problems arising in WCET evaluation cover a wide range of complexity classes, from undecidability in the general case to *NP*-hardness and polynomial-time solvability in some restricted cases [29]. As such, WCET evaluation appears to provide a relevant playground to put the quantum computing promise to the test. The execution time of a program running on a machine depends on multiple factors, such as the initial system state, the hardware, and the input data. The validation of a real-time system requires knowing the WCET. However, the analysis of the exact WCET of a program is complicated by dynamic hardware mechanisms. A possible way to cope with the analysis complexity of WCET is by omitting such hardware mechanisms [62].

All the possible combinations of input data and execution paths need to be considered for computing the actual WCET of a program. This computation of an exact WCET is usually unfeasible with classic computers because the number of possible program paths can grow exponentially with the program size (notwithstanding decidability issues in the general case). Classical computing performs static analysis that approximates the WCET without actually executing the program. This approximation has to be an upper bound of the actual value of the worst-case execution time to assure the system's safeness. However, these WCET approximations should also not be overly pessimistic to avoid system overdimensioning.

Yet, performing an exhaustive examination, even implicitly, of all possible program paths is a *sufficient* condition, even if not necessary, for understanding the worst-case scenario [62]. Thus, quantum computing could allow computing better worst-case-execution-time approximations, thanks to its promising computational power higher than classical computing. In a nutshell, the *program path analysis* method determines which sequence of instructions requires the highest execution time. The problem of determining a program's WCET generally is undecidable. Still, if there are no recursive function calls, dynamic structures, and unbounded loops in the program, it becomes decidable [32], [33]. Those strong hypothesis are enforced to be true in the field of real-time embedded systems, which is the primary target of WCET research.

In this chapter, as a first step, we focus on a simplified program model: we consider only programs with IF-ELSE conditions (i.e., programs in which statically bounded loops have been unrolled), assuming uninterrupted executions and independence from the hardware. We assume the execution time of an instruction to be a constant: each instruction leads to a cache miss, and all data have to be fetched from the main memory. It is worth noticing that all these hypotheses make the problem solvable in polynomial time. Nevertheless, considering the limitation of existing quantum hardware, we argue that they represent an interesting model to be transposed to the quantum framework for benchmarking and that quantum computing approaches should be also evaluated against polynomial problems [63] rather than only on NP-hard ones [64]: Performing well on the former is presumably necessary to perform well on the latter. Furthermore, distance to optimality is of course easier to measure when attempting to solve polynomial-time problems on quantum hardware.

In particular, we examined well-known approaches for the estimation of worst-case execution times [29], [30], [65], [66]. Using Integer Linear Programming (ILP) problems, we can naturally describe the structure of our problem and the set of possible program paths, reducing the issue of estimating the WCET of a program into an optimization problem. The sum of the execution time of the executed *basic blocks* gives the cost function. Our goal is to find the maximum of this function. It is possible to adapt this optimization problem to the quantum computing framework through the penalty function method. The cost function and the constraints of the original problem are represented using linear and quadratic terms. Thus, the problem can be reformulated as a QUBO (Quadratic Unconstrained Binary Optimization) problem [67], [68] and solved by Quantum Approximate Optimization Algorithm (QAOA) and Quantum Annealing.

We solve the problem with different methods: Quantum annealing on D-Wave machines, classical Simulated Annealing and Quantum Approximate Optimization Algorithm.The aim is to compare the performance of these methods (in term of optimization quality). Our tests suggest that D-Wave machines, in ideal cases, achieve the performances of classical Simulated Annealing while being much faster.

The chapter is organized as follows. Section 7.2 provides the necessary preliminaries. Section 7.3 defines our problem and reformulates it in the form of a QUBO problem. Section 7.4 describes our evaluation parameters and the machines for the experiments. Section 7.5 collects the results of our experiments. Section 7.6 concludes the chapter.

7.2 Combinatorial optimization

Combinatorial optimization computes the maximum or minimum of a function over a discrete domain. A combinatorial optimization problem is expressed as:

$$z^* = \arg\max_{z \in S} \; f(z) \tag{7.1}$$

$$\begin{cases} g_l(z) = 0, & l = 1, \ldots, L \\ h_m(z) < 0 & m = 1, \ldots, M \end{cases} \tag{7.2}$$

where z is a discrete integer variables, $f(z)$ is the cost function, and $\mathcal{S}$ the set of decision variables satisfying the equality and inequality constraints given in (7.2) [69].

An integer linear programming (ILP) formulation is the mathematical formulation of an optimization problem in which variables are restricted to integer values and the constraints and cost function are linear [70]. ILP canonical form is expressed as:

$$\max \; c^T x \tag{7.3}$$

subject to

$$\begin{cases} Ax = b, \\ 0 \leq x, \qquad x \in \mathbb{Z}^n \end{cases} \tag{7.4}$$

A Quadratic Unconstrained Binary Optimization (QUBO) problem is a combinatorial problem defined through an upper triangular matrix $Q \in \mathbb{R}^{N \text{x} N}$ and a vector x of binary variables. The goal of the optimization problem is to determine the vector of binary variables $\forall i, x_i \in \{0, 1\}$ that minimizes (or maximizes) the objective function :

$$\sum_i Q_{ii} x_i + \sum_{i<j} Q_{ij} x_i x_j. \tag{7.5}$$

Using the penalty function method, we can rewrite any optimization problem into one without any constraints. For instance, given the equality constaint $g(z) = 0$, we can transform (7.1) into:

$$z^* = \arg\max_{z} \; f(z) + \lambda g(z). \tag{7.6}$$

In this work, we solve the QUBO problem to find the most expensive execution path by using quantum annealing (QA) and the Quantum Approximate Optimization Algorithm (QAOA). We compare the results with these of Classical Simulated Annealing (SA).

At this point, it is worth noting that any QUBO cost function can be transformed into a generalized 2D-Ising Hamiltonian with a simple transformation of variable $x_i = \frac{1+\sigma_i^z}{2}$ with $\sigma_i^z \in \{-1, 1\}$:

$$\mathcal{H}_P = \sum_i^n h_i \sigma_i^z + \sum_{i<j}^n J_{ij} \sigma_i^z \sigma_j^z \tag{7.7}$$

7.2.1 Quantum annealing

Quantum annealing is a computational process that relies on the adiabatic theorem to solve combinatorial optimization problems. As a principle, it implements a time-dependent Hamiltonian composed of an initial Hamiltonian $\mathcal{H}_0$ whose ground state is easy to calculate and a final one tied to the cost function of the optimization problem as seen in equation (7.7): $\mathcal{H}_{\text{Ising}}(t) = A(t)\mathcal{H}_0 + B(t)\mathcal{H}_P$ such that $\mathcal{H}_{\text{Ising}}(0) = \mathcal{H}_0$ and $\mathcal{H}_{\text{Ising}}(\tau) = \mathcal{H}_P$ where τ is the optimal annealing time.

We choose $\mathcal{H}_0 = -\sum_i^n \sigma_i^x$ whose ground state corresponds to an equal superposition of the states of the computational basis. The adiabatic theorem states that if the time evolution is slow enough (i.e., τ is large enough), then the (global) optimal solution can be obtained with high probability. In practice, the final result is conditioned by the size of the spectral gap, the evolution time, the environmental or intrinsic decoherence effects, and the size of the coherence domain of the qubits on the chip.

If the quantum annealing can reach a minimum energy configuration, then the associated state vector solves the equivalent QUBO problem. Reformulating combinatorial problems in QUBO form preserves the underlying structure of the objective function [71].

7.2.2 Quantum Approximate Optimization Algorithm

The Quantum approximate optimization algorithm (QAOA)[72] is a quantum-classical algorithm used to solve optimization problems. It can be seen as an ansatz for the simulation of the Adiabatic process on a gate-based quantum computer. The approximation of the process is done using a parameter p stating the number of steps for the simulation: A p-depth QAOA consists of alternatively apply the two unitary propagators associated with both Hamiltonians $U(\mathcal{H}_P, \gamma)$ and $U(\mathcal{H}_M, \beta)$ on the initial state $|s\rangle$ which is a uniform superposition of all states of the computational basis.

$$|\psi\rangle = U(\mathcal{H}_M, \beta_p)U(\mathcal{H}_P, \gamma_p)...U(\mathcal{H}_M, \beta_1)U(\mathcal{H}_P, \gamma_1)\,|s\rangle \tag{7.8}$$

Each classical optimization round is used to optimize angles $\beta_1..\beta_p$ and $\gamma_1..\gamma_p$ to maximize the expectation value obtained from the run of the quantum circuit.

7.3 Problem designing: from control flow graphs to QUBO

We consider a simple micro-architecture [65] such that each basic block B_i of the program takes a constant time c_i to execute. A basic block is defined as a sequence of consecutive instructions. The flow of control enters into the block at the beginning and leaves at the end, without halt or possibility of branching except in the end. Let variable x_i be the execution count of the basic block B_i and N be the number of basic blocks in the program. In this chapter, we assume that $x_i \in \{0, 1\}$, meaning that each basic block could be executed only once. The total execution time of the program is given by the linear expression:

$$\text{Program execution time} = \sum_{i}^{N} c_i x_i. \tag{7.9}$$

The problem has intrinsic constraints: not every execution flow is possible. For instance, if there is an IF-ELSE condition in our code, only the block respecting that condition is executed. Thus, one part of the program is ignored, depending on the input data. The WCET is given by the cost of the most expensive flow in terms of execution time. Notice that, in our model, we consider that the solution is unique. Our goal is to find the sequence of variables $x_0, \cdots, x_{N-1}$ such that it maximizes the cost function. To clarify all this, let us consider an example of a program with an IF condition: concretely, we have a sequential path that at some point splits into two branches and then reconnects again to the main path.

In this particular example we have only two possible paths of execution and the objective of our optimisation problem is to find the execution path that gives us the maximal cost:

$$\arg\max \ (c_1x_1 + c_2x_2 + c_3x_3 + c_5x_5 \ , \ c_1x_1 + c_2x_2 + c_4x_4 + c_5x_5) \tag{7.10}$$

We call x the vector representing the execution path, with $x_i = 1$ if the corresponding basic block B_i is executed and $x_i = 0$ otherwise. The solution is:

$$x = \begin{cases} \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \end{bmatrix} \text{ if } (c_1x_1 + c_2x_2 + c_3x_3 + c_5x_5 > c_1x_1 + c_2x_2 + c_4x_4 + c_5x_5) \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \end{bmatrix} \text{ otherwise.} \end{cases}$$

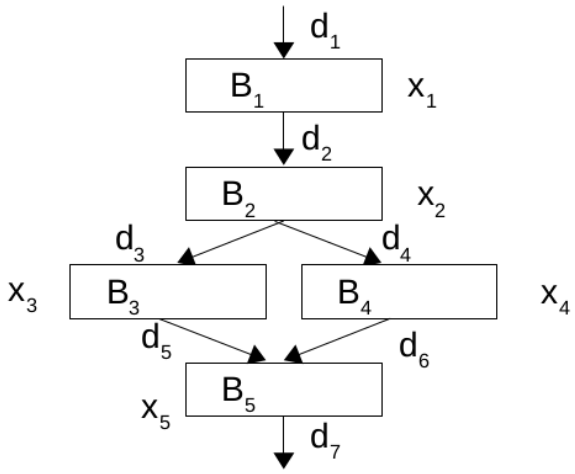


Figure 7.1: Control flow diagram of a simple program structure

In this example, the solution would be obtained simply considering the path that includes the most expensive branch of the IF condition. However, let us consider the constraints of this graph to compute the solution. The nodes of this graph are the blocks of code. The edges are the d_is, representing the action of *entering* or the *quitting* of a basic block. This graph represents the execution of a program, so we know for sure that the first and the last block have to be executed, so the edges entering and quitting these nodes will be equal to one. Only one branch of the condition will be executed (if, for instance, $d_3 = 1$, then $d_4 = 0$). For this example, the set of constraints to consider is:

$$\begin{cases} d_1 = 1, \\ x_1 = d_1 = d_2 \\ x_2 = d_2 = d_3 + d_4 \end{cases} \quad \begin{cases} x_3 = d_3 = d_5 \\ x_4 = d_4 = d_6 \\ x_5 = d_5 + d_6 = d_7 \end{cases} \tag{7.11}$$

We transform this optimization problem to a QUBO moving the constraints to the cost function (penalty method). We omit the constant values in the cost function and we consider $x_i^2 = x_i, \forall i$ because $x_i \in \{0, 1\}$. Our optimization problem thus becomes:

$$\begin{aligned} &\arg\max_x(\textstyle\sum_i^5 c_i x_i - \lambda(1 - x_3 - x_4)^2) \\ &= \arg\max_x(\textstyle\sum_i^N c_i x_i - \lambda(1 - x_3^2 - x_4^2 + 2x_3x_4)) \\ &= \arg\max_x(c_1x_1^2 + c_2x_2^2 + (c_3 + \lambda)x_3^2 + (c_4 + \lambda)x_4^2 - 2\lambda x_3x_4 + c_5x_5^2)) \end{aligned} \tag{7.12}$$

Thus, for the considered problem, the matrix Q is:

$$Q = \begin{pmatrix} c_1 & 0 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 & 0 \\ 0 & 0 & c_3 + \lambda & -2\lambda & 0 \\ 0 & 0 & 0 & c_4 + \lambda & 0 \\ 0 & 0 & 0 & 0 & c_5 \end{pmatrix} \tag{7.13}$$

where $c_1, \ldots, c_5 \in \mathbb{N}$ and $x = (x_1, x_2, x_3, x_4, x_5)^T$.

This chapter focuses on a finite set of study cases to perform our experiments. The most basic program is not more than a linear sequence of instructions, thus with a deterministic and constant execution time, (Fig. 7.2(a)). We explored programs made by chains of consecutive IFs (Fig. 7.2(b)). This problem is interesting because, essentially, any program could be reduced to a loop with a condition in it. So, by unrolling loops, a program could be seen as a chain of conditions. Still, we want to underline that the problem is polynomial: the solution is easily founded by considering the most expensive branch at each IF.

Then we analyzed an expanded version by allowing the exclusive conditions to be more than two SWITCHes (Fig. 7.2(c)). Here again, the solution is given by the path that takes the most expensive branch at each IF condition, so the problem is still polynomially solvable. However, both examples give us the possibility of building an interesting benchmark for quantum computing.

Enlarging slightly more the focus on the targeted problem, we allowed the IFs blocks to have other nested IFs blocks inside them (Fig. 7.2(d)). This situation is slightly more complex than the previous ones because the paths need to be enumerated to find the actual worst-case path. All these cases of study are called *series–parallel graphs*.

The IFs chain and the SWITCH study case are easily generalizable from the matrix (7.13). The nested IF study case is more complex because it is not enough to choose the most expensive branch at each step. Thus, it is not possible to determine a pattern. We develop explicitly here the Q matrix only for a particular example. The Q matrix for the graph shown in Fig. 7.2(d) is:

$$Q = \begin{pmatrix} c_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_1 & -2\lambda & 2\lambda & 2\lambda & 0 \\ 0 & 0 & c_2 + 2\lambda & -2\lambda & -2\lambda & 0 \\ 0 & 0 & 0 & c_3 & -4\lambda & 0 \\ 0 & 0 & 0 & 0 & c_4 & 0 \\ 0 & 0 & 0 & 0 & 0 & c_5 \end{pmatrix}$$

7.3.1 Choice of the lambda value

Let $\{c_1, \ldots c_k\} \in C$ be the set of weights for each node of the graph. $\{x_1, \ldots x_k\}$ are boolean values defining if the basic block is executed. In the single IF case, the

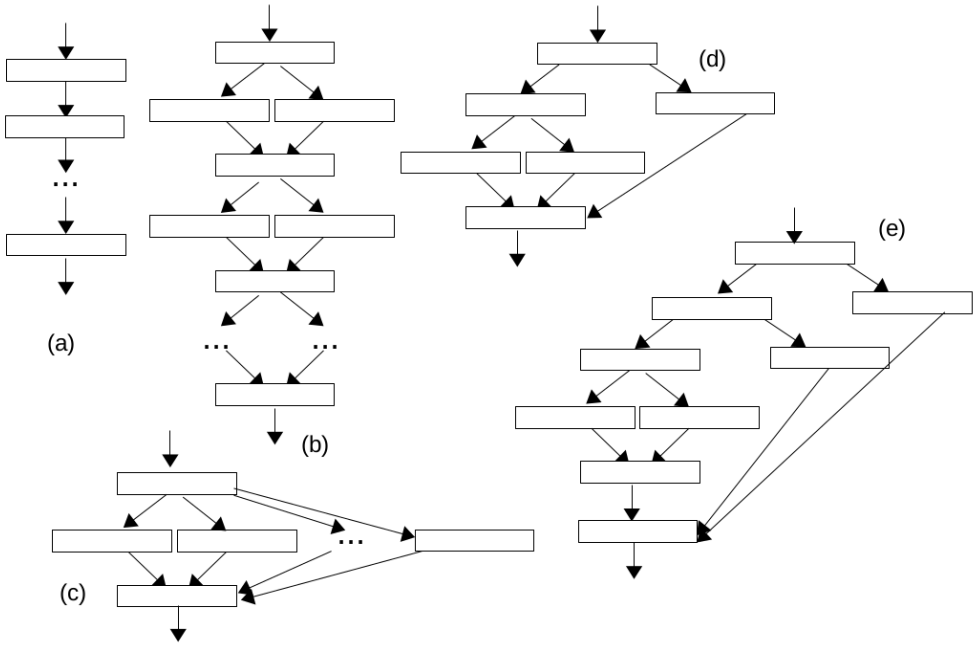


Figure 7.2: Series–parallel graphs

local cost function to the IF case is expressed as $(c_i + \lambda)x_i + (c_j + \lambda)x_j - 2\lambda x_i x_j$ with c_i, c_j the weigths of each path and x_i, x_j the boolean values. The factor λ is appropriate if it follows the set of conditions:

$$\begin{aligned} &\forall\ c_i, c_j \in \mathbb{R}^+ \text{we must have :} \\ &c_i + \lambda > c_i + \lambda + c_j + \lambda - 2\lambda \Leftrightarrow \lambda > c_j \\ &c_j + \lambda > c_i + \lambda + c_j + \lambda - 2\lambda \Leftrightarrow \lambda > c_i \end{aligned} \tag{7.14}$$

For simplicity, we use a single λ for the whole expression, even if it contains multiple IF cases. We also consider that λ is an integer as each graph weight is integer. Therefore, λ should be greater than each weight of the graph yielding:

$$\lambda = \max(C) + 1 \tag{7.15}$$

This choice of λ appears to behave well with the nested IFs and SWITCH too.

7.4 Benchmark Metrics and Computers

We compare the performances of SA, QA and QAOA while solving the optimisation problem of finding the most expensive execution path in the case tests

we mentioned above: IF chains, SWITCH and nested IF. This section presents metrics, algorithms and quantum computers used for the benchmark.

7.4.1 Benchmark metrics

Problems presented in this chapter are toy problems in which optimal solutions can be found in polynomial time. However, their simplicity allows us to fully evaluate and understand the results of our experiments. Before executing the experiments, each of our optimization problems is transposed into their minimization form. To compare our results, we took into consideration two parameters:

- The approximation ratio (7.16). It represents the quality of the solution found compared to the whole energy landscape of the problem. E is the energy obtained from a single simulation, E_{min} is the energy of the optimal solution and E_{max} the energy of the worst solution.

$$r = \frac{E - E_{max}}{E_{min} - E_{max}} \tag{7.16}$$

  This parameter is interesting because, although the solution may not be the exact one, it could be very close.

- The optimal solution probability. It represents the proportion of optimal solutions (solutions having the energy E_{min}) obtained during the simulation.

7.4.2 Simulated Annealing

We configure the simulated annealing with an exponential decrease of temperature $T_1 = 0.95 * T_0$. In the following experiments, we set the thermal equilibrium at $T = 10^{-3}$. These iterations are fixed with the number of input variables n and may vary between $n^{0,5}$ to $n^{1,5}$. We consider SA running in a degraded mode where the number of iterations per temperature step is inferior to n. Each simulation is based on 100 runs of the SA to extract the mean of energy and the probability of getting the optimal solution.

7.4.3 Quantum annealing with D-Wave systems

Our experiments on D-Wave systems involve adiabatic quantum computing used to find the vector that minimizes the input cost function. At the moment, they represent the most advanced quantum machines having thousands of qubits. However, they require problems under the form of QUBOs and the topology of their chips limits their performance [73]. D-Wave systems used during our benchmarks are:

- **DW_2000Q_6** with 2048 qubits (2041 usable qubits) and 6 connections between each qubit (cf. Chimera topology).

- **Advantage_system4.1** with 5760 qubits (5627 usable qubits) and 15 connections between each qubit (cf. Pegasus topology).

For each experiment, results are computed with and without gauge inversion. The principle of a gauge inversion is to apply a Boolean inversion to the σ_i operators in our Hamiltonian. This technique preserves the optimal solution of the problem while limiting the effect of local biases of the qubits, as well as the machine accuracy errors [22]. Following the commonly used procedure (e.g. [74]), we randomly selected 10% of the physical qubits used as spin inversion for each instance. Each simulation is based on 1000 runs of D-Wave systems to extract the mean of energy and the optimal solution probability.

7.4.4 Simulation of QAOA

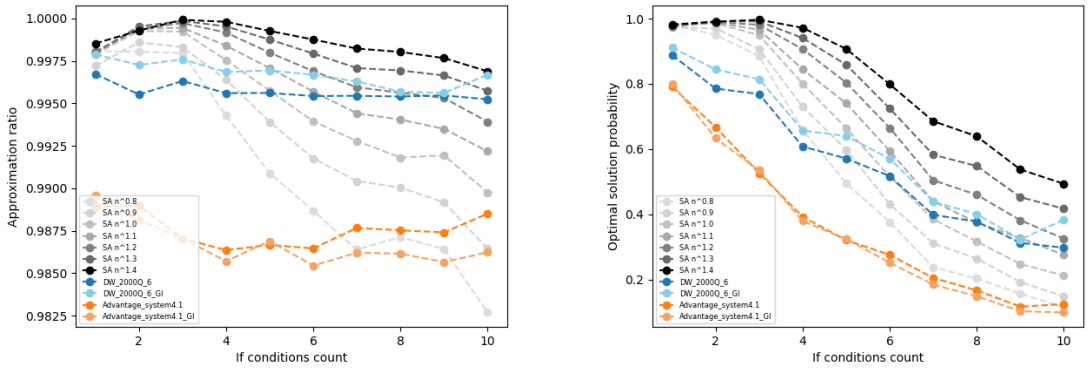
The simulation of QAOA is performed using the Qiskit library [75]. QAOA circuits are built from QUBO instances using penalty terms to express constraints. In our experiments, weights are specified with integers, hence $\gamma \in [0, 2\pi]$ and $\beta \in [0, \pi]$. We did not find patterns to perform interpolation optimization as in [76]. We followed the *parameter fixing strategy* [77] to set angles at p-depth. This method starts at $p = 1$ and randomly generates 100 pairs of angles γ_1 and β_1. Then, we run a local optimizer on each of these pairs. We used COBYLA, a gradient-free optimizer. We run 1024 times the QAOA circuit at each optimization step to sample the mean expectation value corresponding to γ and β angles. At the end of the 100 optimization loops, we get 100 optimized pairs of angles. We select γ_1^* and β_1^* such as they minimize the value of the cost function. This process is then repeated at $p = n$, initializing the problem with $\gamma_1^*...\gamma_{n-1}^*$ and $\beta_1^*...\beta_{n-1}^*$ and 100 pairs of angles γ_n and β_n. For the simulation of the QAOA we used the *aer_simulator*, which provides a good speed performance.

7.5 Experimental results

To represent the cost of each basic block, we generated random integer, such that $c_0, ..., c_n \in \{1, \ldots, 50\}$, and used them as input for our experiments. Each problem is designed to have only a single optimal solution, meaning that each branch candidate for solution should have a different global cost. In section 7.5.1 and section 7.5.2, each data point is smoothed over 30 randomly generated instances.

7.5.1 IF chains

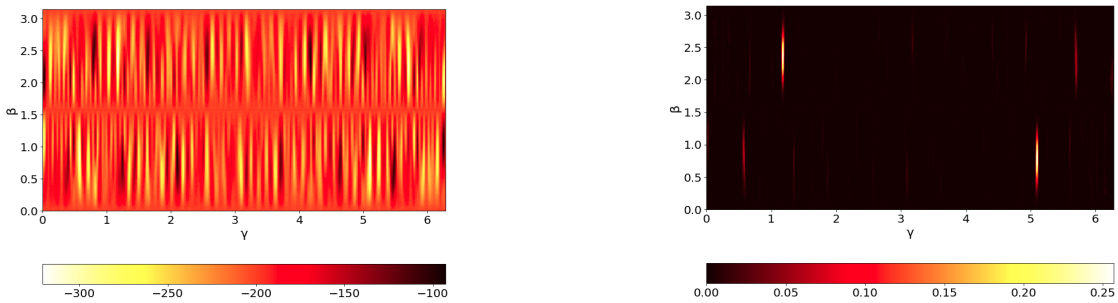
Each IF chain (Fig. 7.2(a)) is composed of a succession of several IF conditions. This study case represents an ideal problem for the D-Wave as the corresponding QUBO matrix is sparse. Hence, the encoding on D-Wave systems does not require any duplication of qubits, neither for the 2000Q_6 nor the Advantage4.1 system. We start from one IF condition for this benchmark and grow the problem up to 10 IF conditions. A single block separates each IF condition block. We benchmark in Fig. 7.3 D-Wave systems against the resolution with classical simulated annealing. As the problem grows, QA seems to outperform SA progressively. For the IF chain, D-Wave system 2000_Q_6 systematically outperforms the Advantage_system4.1. Although the Advantage system is more recent than the 2000Q_6 system, it seems more sensitive to noise. We do not simulate QAOA for IF chains as it gives rather poor results compared to SA and D-Wave systems.



(a) (b)

Figure 7.3: Benchmark of the QUBO resolution by D-Wave systems and SA for IF chains from 1 to 10 IF conditions. Each D-Wave simulation is done with and without GI (Gauge Inversion). n is the number of blocks in the if chain. SA number of iteration per temperature step is expressed according to n. a) Mean of approximation ratio. b) Mean of the probability to get the optimal solution.

Fig. 7.4 shows the QAOA energy landscape at $p = 1$ for a 3-if chain picked randomly. The heatmap exhibits many local minima, which impact the angle optimization of QAOA. This heatmap is more complex than heatmaps usually obtained for the well-studied Max-cut problem ([77],Fig. 5a). This complex energy landscape seems to be closely related to penalty terms that impact the whole energy landscape. Moreover, minimizing the mean energy does not always lead to an increased probability of getting the optimal solution.



(a) Approximation ratio

(b) Optimal solution probability

Figure 7.4: QAOA heatmap at $p = 1$ for a random 3 IFs chain. Approximation ratio is computed from the mean of the expectation value at angles γ and β.

7.5.2 SWITCH

SWITCHes presented in Fig. 7.2(b) are harder to solve for D-Wave system since the plurality of choices leads to a dense matrix and requires higher connectivity between qubits. The impact of the density on QAOA is lower thanks to SWAP gates. In this execution case, we notice that the D-Wave Advantage4.1 performs better than the D-Wave 2000Q_6 for SWITCH cases 3 and 4. This improvement is due to the qubit duplication occurring on D-Wave 2000Q_6 whereas there is no qubit duplication on the Advantage4.1 for these instances (Fig. 7.5e). On larger instances (from 10 to 15 SWITCHes), the advantage of the D-Wave Advantage4.1 provided by its number of connections is questionable. However, when the case is not ideal for D-Wave systems, the performances are poor against SA, even when SA is running under a degraded mode ($n^{0.5}$ iterations per temperature step).

7.5.3 Nested IFs

Nested IFs cases increase the density of the QUBO matrix. Table **??** shows two cases of nested IFs. The results are smoothed over 30 costs generated randomly. We compare the results obtained with SA, QA, and QAOA simulator. As for SWITCH cases, nested IF cases are difficult to be solved for D-Wave systems. They result to be not competitive against simulated annealing. QAOA simulations provide results of a lower performance compared to the D-Wave systems. The results obtained with QAOA start to decrease at $p = 6$. This decrease may be due to the difficulty of COBYLA to optimize γ and β angles at this depth. As the optimization at depth p always starts from angles $\beta_1^*...\beta_{p-1}^*$ and $\gamma_1^*...\gamma_{p-1}^*$, the global optimization may be stuck at local minima imposed by the parameters found at $p-1$. The energy landscape of problems with penalty terms would deserves in-depth study to understand and find patterns of optimization of QAOA angles.

Table 7.1: Nested IFs simulation, Case 1

Nested if results	**Case 1** (see fig.2d)	
	Mean energy	**Opt sol prob**
SA $n^{0.5}$	0.985	0.66
SA $n^{0.6}$	0.986	0.69
SA $n^{0.7}$	0.989	0.76
D-Wave 2000Q	0.985	0.57
D-Wave Advan.	0.983	0.54
QAOA $p = 4$	0.958	0.44
QAOA $p = 5$	0.965	0.49
QAOA $p = 6$	0.926	0.40

Table 7.2: Nested IFs simulation, Case 2

Nested if results	**Case 2** (see fig.2e)	
	Mean energy	**Opt sol prob**
SA $n^{0.5}$	0.999	0.934
SA $n^{0.6}$	0.999	0.923
SA $n^{0.7}$	0.999	0.956
D-Wave 2000Q	0.993	0.56
D-Wave Advan.	0.975	0.29
QAOA $p = 4$	0.909	0.10
QAOA $p = 5$	0.920	0.13
QAOA $p = 6$	0.855	0.06

7.5.4 Real case

We provide a concrete application inspired by the bubble sort algorithm provided in the Mälardalen WCET research group [78][79]. The algorithm's goal is to sort an array of integers in ascending order. We consider an ideal scenario with an in-order, single-issue Arm processor similar to an M_0 with prefetched cache memory. Blocks composing the graph are built from the instructions obtained after the compilation of C code (using *gcc* Arm 8.3 with *-O2* level of optimization). The number of micro-instruction in each basic block defines its cost as given by our in-house WCET explorer with block identification and pipeline simulation. For the sake of simplicity, we limited to 3 the number of elements in the vector to sort. Table 7.3 shows the obtained result while solving the problem with SA and QA. From Table 7.3 we conclude that the 2000Q system performs quite well, even duplicating 4 qubits. The Advantage system, while duplicating only 1 qubits performs poorly. We may conclude that duplicating on Advantages machines downgrade drastically

the performances of the machine.

Table 7.3: Real case simulation

Method	**Mean energy**	**Optimal solution prob**	**Qubits used**	**Max qubits duplication**
SA $n^{0.5}$	0.944	0.58	/	/
SA n^{1}	0.972	0.79	/	/
SA $n^{1.1}$	0.990	0.91	/	/
D-Wave 2000Q_6	0.987	0.868	15	1
D-Wave Advantage4.1	0.901	0.325	12	1

7.6 Conclusion

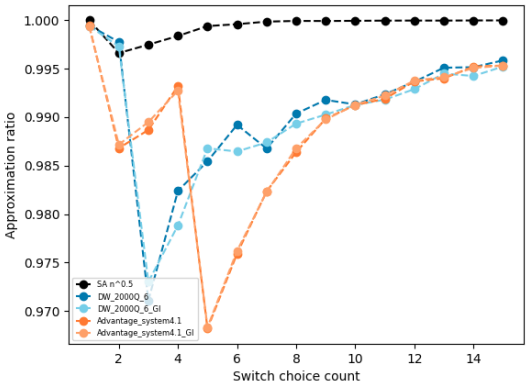
In this chapter, we explored the potential of quantum computing in dealing with the combinatorial optimization problem of finding the most expensive execution path in a graph on a restricted set of simple instances. Our work offers an example of using quantum computing in a new application field. The performances we obtained using D-Wave machines and the QAOA do not suggest that quantum computing is the silver bullet solution that will replace the classical computing solid and road-tested techniques. Still, we claim the results we obtained are encouraging enough to explore further the potential of quantum computing on the proposed problem (for instance, the backtracking quantum algorithm). Indeed, our experiments suggest that quantum computing gives us a fast equivalent of simulated annealing for ideal problems, such as the IFs chains.

We can draw some conclusions from the experience of this work on D-Wave machines. Their topology is quite limiting, and it is not straightforward to adapt the considered optimization problem to it. We noticed that the Advantage4.1 machine outperforms the 2000Q machine when the considered problems involve qubit duplications on 2000Q device and not on the Advantage4.1 machine. It may be worth exploring the performances of these machines while solving problems that need more qubit duplications to be adapted to both topologies. The goal is to find parameters that show a priori which D-Wave machine is the most suitable to solve the considered problem. The chain of IFs is ideal for the topology of D-Wave systems, and their results are competitive with SA. Concretely, SA on 5600 variables

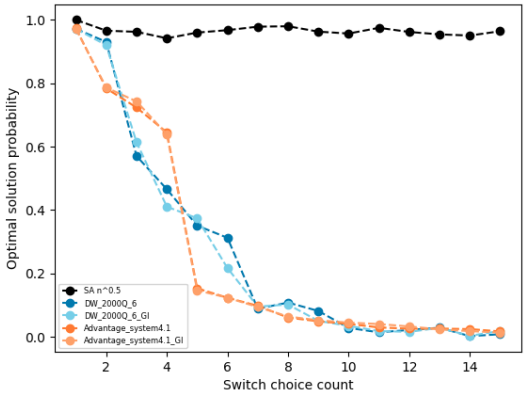
at n^2 would need billions of evaluations of the cost function against millisecond runtime for D-Wave systems. However, we stress again that the problem is polynomially solvable using methods other than SA, and the interest is the possibility of building a benchmark for quantum computers.

We can as well draw some conclusions about QAOA. At $p = 5$ the performances of QAOA are almost the same as D-Wave. QAOA does not have the problem of qubit duplication since we can circumvent the chip's topology with SWAP gates. Still, we performed our experiments on a simulator and not on a physical machine as D-Wave systems. An idea to investigate is to restrict the search on the feasible subspace of the problem [80]. In our case, this would be a subspace where every solution preserves the flow constraint.

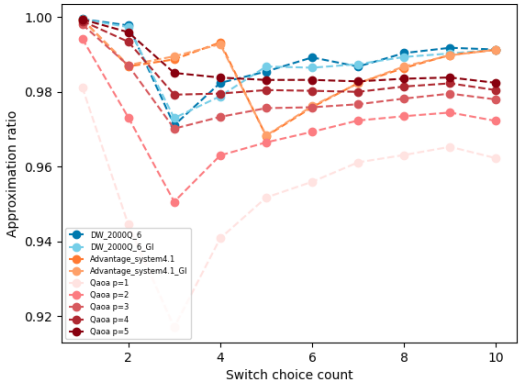
Another aspect that may be explored in the future is the behavior of our models considering the effect of cache memories. Additionally, our choice of λ has no guarantees of being optimal, even if it provides good results. As a perspective, it may be interesting to perform a pre-processing to find, through simulations, the optimal λ. However, the effort of finding the optimal λ could overcome its benefits.



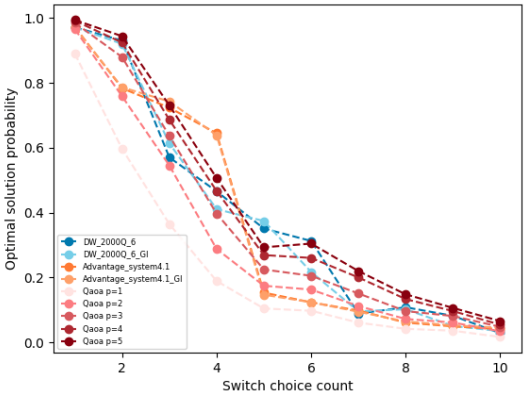
(a) Energy ratio SA and D-Wave



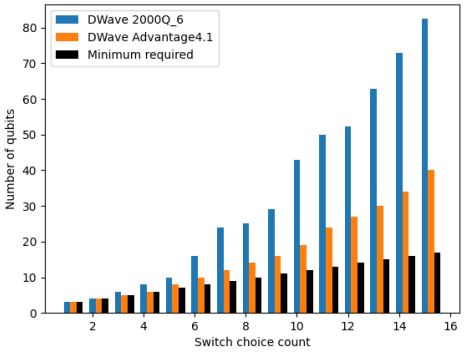
(b) Optimal solution probability



(c) Energy ratio QAOA and D-Wave



(d) Optimal solution probability



(e) Qubits used by D-Wave systems

Figure 7.5: Comparison of switch instances solved with classical SA, D-Wave systems and QAOA simulator. (e) Mean of the number of qubits used by D-Wave systems to solve switch instances against the optimal number of qubits on ideal fully connected topology.

Chapter 8

Conclusion and perspectives

This thesis work focuses on tackling a concrete problem, namely the WCET evaluation, using quantum computing approaches. Rather than just indulging in the most followed patterns, as it would have been exploiting Shor- or Grover-like algorithms as subroutines for other algorithms, we decided to exhaustively explore the mainstream current quantum techniques to dive into the problem.

Until a few years ago, the possibility of using quantum machines was limited, but nowadays, many companies in the quantum computing field provide access to quantum machines, such as the D-wave annealers and IBM simulators we used for our tests. The possibility to access those resources, aside from helping the author's work, testifies how much the whole quantum computing field has advanced in the last decades. Still, our experience allowed us to draw some conclusions and identify two main issues with current quantum computers: hardware and formalism.

Currently, quantum hardware is limited by several technical issues. Not only quantum computers have at their disposal only a few qubits that are not connected enough, but it is also often necessary to duplicate qubits to map a problem onto a quantum machine, thus reducing even more the number of logical qubits available for the actual computation. In addition, the computation process suffers from noise issues in the sense that these machines are only very imperfect ersatzes their respective theoretical models, especially with respect to coherence, resulting in sometimes disappointing experimental results.

Concerning formalism, to map the target problem onto a quantum computer, it is necessary to modify it by reformulating most of its hard constraints into soft ones. For instance, theoretically, the QUBO model is universal and allows representing all problems (in NP). However, in practice, we had to remove many constraints and add penalty factors to our model to make it fit. The resulting model, albeit fitting the machine constraints, ultimately became too ill-conditioned to solve real-world instances satisfactorily. This trade-off is acceptable since our research's focus was to fully explore one particular problem from a quantum com-

puting point of view and confirm that it is possible to use today's quantum computers to solve simple problems rather than have a breakthrough in the WCETs solving techniques.

Once we recognize how inherently different quantum computing is from classical computing, it becomes apparent that to be able to harness the potential of the former, we need to grow our skills and knowledge on how to formalize and solve problems on quantum computers. Thus, tackling concrete problems through quantum approaches is essential, and so it is for universities and companies in the quantum computing field to keep pushing the limits of today's machines and implement new, more flexible ones with other quantum paradigms. Also, with a deeper understanding of this technology, we should aim to be able to establish *a priori* if an application is suitable for the quantum computing framework. Moreover, when a particular application is judged suitable to be implemented in quantum computing with actual benefit, it would be essential to be able to address the quantum approach and machine that could give the maximal benefit. In that sense, this thesis belongs to the first line of works that attempt to build a practically relevant benchmark for quantum computing.

Bibliography

[1] R. Allan, D. Cable, T. Franks, and P. Kummer, *A history of computing at daresbury laboratory*, 2007.

[2] A. Thackray, D. C. Brock, and R. Jones, *Moore's law: The life of gordon moore, silicon valley's quiet revolutionary*, 2016.

[3] P. W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, 1997.

[4] L. K. Grover, *A fast quantum mechanical algorithm for database search*, 1996.

[5] R. P. Feynman, *Simulating physics with computers*, 1981.

[6] D. Deutsch, *Quantum theory, the church-turing principle and the universal quantum computer*, 1985.

[7] J. Powell, *The quantum limit to moore's law*, 2008.

[8] S. Aaronson, *The limits of quantum computers*, 2008.

[9] R. de Wolf, *The potential impact of quantum computers on society*, 2017.

[10] A. Montanaro, *Quantum algorithms: An overview*, 2015.

[11] C. Dalyac, L. Henriet, E. Jeandel, *et al.*, *Qualifying quantum approaches for hard industrial optimization problems. a case study in the field of smart-charging of electric vehicles*, 2022.

[12] R. Shaydulin, H. Ushijima-Mwesigwa, C. F. A. Negre, I. Safro, S. M. Mniszewski, and Y. Alexeev, *A hybrid approach for solving optimization problems on small quantum computers*, 2019.

[13] L. Zhu, H. L. Tang, G. S. Barron, *et al.*, *Adaptive quantum approximate optimization algorithm for solving combinatorial problems on a quantum computer*, 2022.

[14] S. Harwood, C. Gambella, D. Trenev, A. Simonetto, D. Bernal, and D. Greenberg, *Formulating and solving routing problems on quantum computers*, 2021.

[15] G. Brassard, I. Chuang, S. Lloyd, and C. Monroe, *Quantum computing*, 1998.

[16] A. Mandviwalla, K. Ohshiro, and B. Ji, *Implementing grover's algorithm on the ibm quantum computers*, 2018.

[17] M. I. Dyakonov, *Prospects for quantum computing: Extremely doubtful*, 2014.

[18] E. Bernstein and U. Vazirani, *Quantum complexity theory*, 1997.

[19] C. S. Calude and E. Calude, *The road to quantum computational supremacy*, 2017.

[20] I. Kerenidis and A. Prakash, *Quantum recommendation system*, 1998.

[21] E. Tang, *A quantum-inspired classical algorithm for recommendation systems*, 2019.

[22] S. Boixo, T. Albash, F. Spedalieri, N. Chancellor, and D. Lidar, *Experimental signature of programmable quantum annealing*, 2013.

[23] A. W. Harrow and A. Montanaro, *Quantum computational supremacy*, 2017.

[24] F. Arute and et al, *Quantum supremacy using a programmable superconducting processor*, 2019.

[25] E. Pednault, J. Gunnels, D. Maslov, and J. Gambetta, *On quantum supremacy*, 2019.

[26] P. P. Angara, U. Stege, H. A. Müller, and M. Bozzo-Rey, *Hybrid quantum-classical problem solving in the nisq era*, 2020.

[27] N. S. Yanofsky, *An introduction to quantum computing*, 2007.

[28] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, 2010.

[29] R. Wilhelm, J. Engblom, A. Ermedahl, *et al.*, *The worst-case execution-time problem—overview of methods and survey of tools*, 2008.

[30] Y.-T. Li, S. Malik, and A. Wolfe, *Efficient microarchitecture modeling and path analysis for real-time software*, 1995.

[31] J. Fellmuth, T. Göthel, and S. Glesner, *Instruction caches in static wcet analysis of artificially diversified software*, 2018.

[32] E. Kligerman and A. D. Stoyenko, *Real-time euclid: A language for reliable real-time systems*, 1986.

[33] P. Puschner and C. Koza, *Calculating the maximum execution time of real-time programs*, 1989.

[34] L. Cucu-Grosjean, L. Santinelli, M. Houston, *et al.*, *Measurement-based probabilistic timing analysis for multi-path programs*, 2012.

[35] J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla, *On the comparison of deterministic and probabilistic wcet estimation techniques*, 2014.

[36] F. Cazorla, E. Quiñones, T. Vardanega, *et al.*, *Proartis: Probabilistically analyzable real-time systems*, 2013.

[37] P. Cousot and R. Cousot, *Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Los Angeles, California, 1977.

[38] S. Louise, *A first step toward using quantum computing for low-level wcets estimations*, 2019.

[39] E. Grumbling and M. Horowitz, *Quantum computing: Progress and prospects*, 2019.

[40] F. Reghenzani, G. Massari, and W. Fornaciari, *Timing predictability in high-performance computing with probabilistic real-time*, 2020.

[41] T. S. Humble, A. McCaskey, D. I. Lyakh, and M. Gowrishankar, *Quantum computers for high-performance computing*, 2021.

[42] A. Facon, S. Guilley, M. Lec'Hvien, A. Schaub, and Y. Souissi, *Detecting cache-timing vulnerabilities in post-quantum cryptography algorithms*, 2018.

[43] A. J. Abhari, S. Patil, D. Kudrow, *et al.*, *Scaffcc: A framework for compilation and analysis of quantum computing programs*, 2014.

[44] S. Bravyi, D. Gosset, R. König, and et al., *Quantum advantage with noisy shallow circuits*, 2020.

[45] *The quantum algorithm zoo.* `http://math.nist.gov/quantum/zoo/`.

[46] R. J. Hall, *A quantum algorithm for software engineering search*, 2009.

[47] D. Patterson and J. Hennessy, *In praise of computer organization and design: The hardware/ software interface, fifth edition*, 2014.

[48] A. Cherkas and S. Chivilikhin, *Quantum adder of classical numbers*, 2016.

[49] J. Cavicchio, C. Tessler, and N. Fisher, *Minimizing cache overhead via loaded cache blocks and preemption placement.* 2015.

[50] G. Buttazzo, M. Bertogna, and G. Yao, *Limited preemptive scheduling for real-time systems, a survey,* 2013.

[51] C.-G.Lee, J.Hahn, Y.-M.Seo, *et al.*, *Analysis of cache-related preemption delay infixed-priority preemptive scheduling*, 1998.

[52] H. Ramaprasad and F. Mueller, *Tightening the bounds on feasible pre-emption points*, 2006.

[53] ——, *Bounding worst-case response timefor tasks with non-preemptive regions*, 2008.

[54] A. Manju and M. Nigam, *Applications of quantum inspired computational intelligence: A survey*, 2012.

[55] S. B. Ramezani, A. Sommers, H. K. Manchukonda, S. Rahimi, and A. Amirlatifi, *Machine learning algorithms in quantum computing: A survey,* 2020.

[56] H. Talbi, A. Draa, and M. Batouche, *A new quantum-inspired genetic algorithm for solving the travelling salesman problem*, 2004.

[57] D. Jethwani, F. L. Gall, and S. K. Singh, *Quantum-inspired classical algorithms for singular value transformation,* 2020.

[58] G. Ripoll and J. Jose, *Quantum-inspired algorithms for multivariate analysis: From interpolation to partial differential equations*, 2021.

[59] N. Chia, H. Lin, and C. Wang, *Quantum-inspired algorithms for multivariate analysis: From interpolation to partial differential equations,* 2021.

[60] A. Gilyen, Z. Song, and E. Tang, *An improved quantum-inspired algorithm for linear regression,* 2020.

[61] T. F. Rønnow, Z. Wang, J. Job, *et al.*, *Defining and detecting quantum speedup*, 2014.

[62] J.-C. Liu and H.-J. Lee, *Deterministic upperbounds of the worst-case execution times of cached programs*, 1994.

[63] D. Vert, R. Sirdey, and S. Louise, *Benchmarking quantum annealing against "hard" instances of the bipartite matching problem*, 2021.

[64] Ö. Salehi, A. Glos, and J. A. Miszczak, *Unconstrained binary models of the travelling salesman problem variants for quantum optimization*, 2022.

[65] Y.-T. Li, S. Malik, and A. Wolfe, *Cache modeling for real-time software: Beyond direct mapped instruction caches*, 1996.

[66] H. Theiling and C. Ferdinand, *Combining abstract interpretation and ilp for microarchitecture modelling and program path analysis*, 1998.

[67] A. Lucas, *Ising formulations of many np problems*, 2014.

[68] C. C. Chang, C. C. Chen, C. Koerber, T. S. Humble, and J. Ostrowski, *Integer programming from quantum annealing and open quantum systems*, 2020.

[69] M. Zaman, K. Tanahashi, and S. Tanaka, *Pyqubo: Python library for mapping combinatorial optimization problems to QUBO form*, 2021.

[70] S. Bradley, A. Hax, and T. Magnanti, *Applied mathematical programming*, 1977.

[71] F. Glover, G. Kochenberger, and Y. Du, *A tutorial on formulating and using qubo models*, 2018.

[72] E. Farhi, J. Goldstone, and S. Gutmann, *A quantum approximate optimization algorithm*, 2014.

[73] D. Vert, R. Sirdey, and S. Louise, *On the limitations of the chimera graph topology in using analog quantum computers*, ACM, 2019.

[74] T. Albash and D. Lidar, *Demonstration of a scaling advantage for a quantum annealer over simulated annealing*, 2018.

[75] J. Gambetta and et al., *Qiskit: An open-source framework for quantum computing*, 2022.

[76] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, *Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices*, 2020.

[77] X. Lee, Y. Saito, D. Cai, and N. Asai, *Parameters fixing strategy for quantum approximate optimization algorithm*, 2021.

[78] J. Gustafsson, A. Betts, A. Ermeddahl, and B. Lisper, *The malardalen wcet benchmarks - past, present and future*, 2010.

[79] Malardalen, *Wcet benchmarks,* `http://www.mrtc.mdh.se/projects/wcet/benchmarks.html`.

[80] S. Hadfield, Z. Wang, B. O'Gorman, E. Rieffel, D. Venturelli, and R. Biswas, *From the quantum approximate optimization algorithm to a quantum alternating operator ansatz*, 2019.

université PARIS-SACLAY

DÉPARTEMENT
Sciences et Technologies de l'Information et de la Communication

Titre: Approches quantiques pour l'analyse des exécutions pire-cas des programmes.
Mots clés: Informatique quantique, Applications du calcul quantique, WCET, QUBO, Programmation dynamique.

Résumé: L'informatique quantique gagne en popularité dans la communauté informatique. La prise de conscience du potentiel de l'informatique quantique a commencée en 1981, lorsque Richard Feynman a imaginé la construction d'un ordinateur quantique. Cependant, le domaine a connu beaucoup de scepticisme quant à ses capacités pratiques à long terme pour résoudre les problèmes. En particulier, les chercheurs tente de relever le défi de construire des ordinateurs quantiques scalables et fiables. Dernièrement, de nombreuses entreprises ont obtenu des résultats encourageants et ont construit des machines quantiques avec suffisamment de qubits pour commencer à mener des expériences intéressantes dessus. Nous avons choisi l'évaluation du pire temps d'exécution (WCET) comme application de nos recherches sur l'informatique quantique, car elle est cruciale pour diverses applications temps réel. L'analyse WCET garantit que le temps d'exécution d'un programme respecte toutes les contraintes d'ordonnancement et de timing. Dans l'histoire des algorithmes quantiques, l'attention a souvent été accordée aux problèmes avec une structure mathématique particulière. L'évaluation des WCET, à l'opposé, n'est pas un problème a priori favorable au contexte quantique, et possède des solutions classiques efficaces déjà éprouvées. Ainsi, il est intéressant d'explorer l'impact de l'informatique quantique sur ce type de problèmes, dans l'esprit de trouver des domaines nouveaux et concrets dans lesquels l'informatique quantique pourrait apporter sa contribution. Si ce n'est pas le cas, la recherche dans ces domaines spécifiques peut aider à définir les limites des applications qui pourraient bénéficier de l'informatique quantique. Cette thèse présente différentes approches quantiques pour effectuer des évaluations WCETs de programmes pour des modèles simplifiés.

Title: Quantum approaches for Worst-Case Execution-Times analysis of programs.
Keywords: Quantum computing, Application of quantum computing, WCET, QUBO, Dynamic programming.

Abstract: Quantum computing is gaining popularity in the computer science community. The awareness of the potential of quantum computing started in 1981, when Richard Feynman first speculated about building a quantum computer. However, until recently, the field has known much skepticism about its long-term practical capabilities to solve problems. In particular, researchers are still facing the challenge of building scalable and reliable quantum computers. Lately, many companies have obtained encouraging results and built quantum machines with enough qubits to start conducting interesting experiments. We chose the worst-case execution-time (WCET) evaluation as the application of our research on quantum computing, as it is crucial for various real-time applications. WCET analysis guarantees that a program's execution time matches all the scheduling and timing constraints. In quantum algorithms history, attention was often given to problems with a particular mathematical structure. The WCETs evaluation, as an opposite, is not a particularly quantum-friendly problem, and it has already proven efficient classical solutions. Hence, it is worth exploring the impact of quantum computing on those kinds of problems, with the spirit of finding new and concrete fields to which quantum computing could bring its potential. If not, research on such specific fields will help to set the boundaries of which applications could benefit from quantum computing. This thesis presents different quantum approaches to perform WCETs evaluations of programs under simplified assumptions.

Maison du doctorat de Université Paris-Saclay
2e étage, aile ouest, École normale supérieure Paris-Saclay
4 avenue des Sciencs
91190 Gif-sur-Yvette, France