Solving Boolean Satisfiability in Parallel using a QUBO Reduction

Daudi Wampamba Bandres
CS with AI
COMP4027 psydw3@nottingham.ac.uk

Introduction

The boolean Satisfiability problem is a well studied NP-Hard decision problem solvers

Parallel & Quantum Computing

Parallel computing has become a pivotal technology in the modern world, playing an increasing role in recent years as the main method of improving the performance of classical computers. The increasing complexity of computational problems, has pushed industry and research to look to the efficient parallelisation of work as a solution to dealing with ever larger problems.

While parallel computing has gained prominence as a reliable method for enhancing classical computer performance, quantum computing remains in its experimental infancy, characterized by the unique abilities that quantum effects enable that offer the potential for groundbreaking speed-ups. These two computing paradigms have given rise to divergent strategies in algorithm development and problem-solving approaches.

Quadratic Unconstrained Binary Optimization & Maximum Satisfiabililty

Quadratic Unconstrained Binary Optimization (QUBO) problems are a class of mathematical optimization problems that involve finding the best combination of binary variables to minimize or maximize a quadratic objective function. In a QUBO problem, each variable represents a binary decision. The objective function, which is quadratic in nature, quantifies the relationship between these binary variables and aims to optimize some real-world or abstract problem. Such problems have a wide range of applications, including portfolio optimization, scheduling, and circuit design, among others.

One notable feature of QUBO problems is their suitability for quantum computers, specifically through a technique known as quantum annealing. Quantum annealing is a quantum computing approach designed to efficiently solve optimization problems like QUBO. It leverages the principles of quantum mechanics to encode the QUBO solution in qubits that find solutions by , making it potentially faster than classical computers for certain optimization tasks.

The intersection of QUBO problems and quantum computing has sparked significant research interest. This interest has also lead to an interest in solving QUBO problems using parallel computing, not only quantum computing. Parallel computing involves breaking down a problem into smaller tasks that can be executed simultaneously, thus potentially speeding up computation. Various parallelization techniques have been developed in the context of QUBO problems. Some of these techniques aim to simulate the quantum annealing process on classical, non-quantum hardware, while others explore novel approaches that harness the unique computational power of quantum computers to directly solve QUBO problems.

The Maximum Satisfiability (Max-SAT) problem, an NP-Hard Optimization problem similarly to QUBO, has traditionally been approached with predominantly single-threaded solvers. Existing parallel techniques, centered around clause-sharing and portfolio-based methods, have faced challenges in achieving efficient core utilization and adapting to SIMD approaches for GPU acceleration.

The main output of this project is a fully featured Max-SAT solver that employs a reduction from Max-SAT to QUBO as a means to enable the simple parallelisation. Leveraging the substantial body of research surrounding QUBO, this approach aims to not only to create a parallel Max-SAT solver, but also compare this approach with existing apporaches, including classical, parallel, quantum apporaches.

Aims and Objectives

The aim of this project is to produce a parallel and quantum-capable Max-SAT solver.

- Investigating existing solutions for a suitable reduction from Max-SAT to QUBO and developing a Max-SAT to QUBO reduction with a complexity that allows the parallel algorithm to maintain it’s time complexity.
- Implementing and optimising an algorithm for solving QUBO in parallel based on the large body of existing work.

- Test and evaluate the performance of the resulting parallel Max-SAT solver, using the Max-SAT Evaluations set
- Conduct an in-depth comparative evaluation of the efficacy of the solver in comparison to existing approaches, including other parallel algorithms and quantum computing approaches.

External Aspect

This project carries a broad external applicability, with potential implications for a diverse range of stakeholders. The ability to efficiently parallelize Max-SAT solvers, extends its relevance beyond the research realm. It may find utility among industries grappling with complex optimization challenges, such as logistics and supply chain management, where the rapid and parallel resolution of Max-SAT problems is critical. Moreover, the project could draw interest from fields beyond computer science, including operations research and engineering.

Motivation

Related Work

Description of the work

Methodology

Design

The solver is designed to be flexible and allow for the implementation of many alternative backends allowing for different reduction and solving algorithms to easily be compared to one another. The general flow for solvving a problem is shown below.

Input from the user is Processed into the SAT Problem instance. The SAT Problem is then reduced into a QUBO problem using one of the specified algorithms. We can then solve the reduced QUBO problem with another specified algorithm. Once we have a solution, we present it to the user.

Reduction Algorithms

The plan is for the solver to implements 5 reduction algorithms:

- **Choi** [1] which uses a reduction from K-SAT to 3-SAT to MIS then finally QUBO
- A novel method which reduces K-SAT to Max-2-SAT and then to QUBO
- **Chancellor** [2] which directly encodes problems in a Hamiltonian function that defines the QUBO Matrix. This is the current state-of-the-art method, and the resulting QUBO Matrices are notably smaller than that of Choi.
- **Nusslein 2022** [3] is similar to Chancellor, but is supposed to scale better for QUBO formulations where the resulting QUBO graph has a number of edges that is sub-quadratic i.e. $|E| = \Theta(|V|)$
- **Nusslein 2023** [4] is a preprint which is supposed to produce smaller QUBO matrices than Chancellor with similar characteristics

QUBO Solving Algorithms

The plan is for the solver to implement 6 QUBO solving algorithms:

- **Simulated Annealing**
- **Parallel Exhausive Search** [5]
- **MOPSO** [6]
- **Momentum Annealing** [7]
- **Simulated Quantum Annealing** [8]
- **Divers Adaptive Bulk Search** [9]

Implementation

The Solver is written in Rust. Rust was chosen as it has good tools for abstraction of problems and a strong type system that makes it easy to encode problems in. Additionally it has really good tools for concurrency which make coding the parallel sections much easier.

Input/Output

Following typical SAT solver convention and in-line with the competition requirements, the solver uses the DIMACS Input/Output format for inputing problem instances. Included in this project is a python file `generate_cnf.py` for generating random cnf instances.

Problem & Solution Encoding

Problems are implemented as Rust traits which makes it easy to implement different problems which can be reduced into one another in definitive ways. There are two specific interfaces for problems. `Problem` which every problem implements, and `ReducibleProblem` which defines how a problem should be reduced into another problem.

```
pub trait Problem<SolutionType, EvaluationType> {
    fn solve(&self) -> SolutionType;
    fn validate_solution(&self, solution: &SolutionType) -> EvaluationType;
}

pub trait ReducibleProblem<T: Copy, TSolutionType, TEvaluationType, USolutionType,
UEvaluationType>: Problem<TSolutionType, TEvaluationType> {
    fn solve_with_reduction(&self, reduction: T) -> TSolutionType;
    fn solve(&self) -> TSolutionType;
    fn reduce(&self, reduction: T) -> Box<dyn Problem<USolutionType, UEvaluationType>>;
    fn convert_solution(&self, reduction: T, solution : USolutionType) -> TSolutionType;
}
```

This makes it trivial to add new Problems to the solver as intermediary reductions for example the SAT to QUBO Reduction (simplified)

```
pub enum SatToQuboReduction {
    Choi,
    Satellite,
    Chancellor,
    Nuesslein2022,
    Nuesslein2023,
}

impl ReducibleProblem<...> for KSatProblem {
    fn reduce(&self, reduction: SatToQuboReduction) -> Box<dyn Problem<QuboSolution, i32>> {
        // reduce code
    }

     fn convert_solution(&self, reduction: SatToQuboReduction, solution : QuboSolution) ->
KSatSolution {
        // convert solution code
    }
}
```

The underlying structure of problems can vary a lot, but rust allows us to be flexible with that as long as we implement the traits above. The QUBO problem uses a sparse matrix representation in order to improve memory efficiency. The problem also stores how it should be solved alongside it.

```
pub enum QuboProblemBackend {
    ParallelExhaustiveSearch,
    MopsoParallel,
    // ...
}
// ...
pub struct QuboProblem {
    problem_matrix: matrix::SparseMatrix<i32>,
    problem_backend: QuboProblemBackend
}
```

The SAT problem doesn't implement any solve function as it is tangential to this project. SAT instances are stored as the number of variables and a list of of lists of variables analagous to conjunctive normal form, each variable having whether it is negated (`true` for not negated, `false` for negated) and a which variable number it corresponds to.

```
pub struct KSatVariable(pub bool, pub usize);
pub struct KSatProblem(pub usize, Vec<Vec<KSatVariable>>);
```

Solutions can be any rust type which makes defining problems very flexible, for example when solving a SAT problem there are 3 possible outcomes, `SAT` or `UNSAT` or `UNKNOWN`, however for QUBO problems, there is only one form of solution which is the list of binary variables.

```
pub enum KSatSolution {
    Sat(Vec<bool>),
    Unsat,
    Unknown
}
// ...
pub struct QuboSolution(pub Vec<bool>);
```

Solving

Work Plan

Phase One: Research [November]

- Conduct a literature review to identify potential parallel algorithms for solving QUBO problems.
- Summarize the strengths and weaknesses of these algorithms.
- Create a comprehensive list of references in this area.
- Investigate existing methods for reducing Max-SAT problems to QUBO.
- Examine the theoretical underpinnings of these reductions and their practical applicability.
- Identify key challenges and opportunities in this field.
- Compile the findings into an interim report discussing the chosen research direction.

Phase Two: Design & Implementation [December - January]

- Design a Max-SAT to QUBO reduction algorithm.
- Specify the algorithm's logic, data structures, and optimization techniques.
- Ensure the algorithm's compatibility with parallelization.
- Write the code for the Max-SAT to QUBO reduction algorithm.
- Test the algorithm with sample Max-SAT instances to validate its functionality.
- Plan the architecture of the parallel QUBO solver.
- Choose appropriate parallelization techniques (e.g., multi-threading, SIMD, GPU acceleration).
- Define how the solver will utilize multiple cores or processors effectively.
- Code the parallel QUBO solver following the design specifications.
- Ensure that the solver can handle a range of QUBO instances in parallel.
- Conduct initial performance tests for the parallel solver.
- Combine the Max-SAT to QUBO reduction algorithm with the parallel QUBO solver.
- Verify that the integrated solver functions correctly.
- Test the solver on various benchmark problems to assess its initial performance.

Phase Three: Testing, Optimization, and Refinement [February - March]

- Collect and analyze data on the solver's performance, including execution times and solution quality.
- Begin writing detailed documentation on the solver's mechanism, inputs, and outputs.
- Identify performance bottlenecks and areas for improvement.
- Implement optimization techniques to enhance the solver's efficiency.
- Conduct comparative tests to assess the impact of optimizations.
- Execute extensive testing on the refined solver using a diverse set of Max-SAT problems.
- Evaluate the solver's performance against existing solvers and quantum computing approaches.
- Record and analyze the results.
- Compile all the research, development, and testing findings into a comprehensive final report.

Bibliography

[1] V. Choi, “Different Adiabatic Quantum Optimization Algorithms for the NP-Complete Exact Cover and 3SAT Problems”, *Proceedings of the National Academy of Sciences*, no. 7, Feb. 2011, doi: 10.1073/pnas.1018310108.

[2] N. Chancellor, S. Zohren, P. A. Warburton, S. C. Benjamin, and S. Roberts, “A Direct Mapping of Max k-SAT and High Order Parity Checks to a Chimera Graph”, *Scientific Reports*, no. 1, p. 37107, Nov. 2016, doi: 10.1038/srep37107.

[3] J. Nüßlein, T. Gabor, C. Linnhoff-Popien, and S. Feld, “Algorithmic QUBO formulations for \textit{k} -SAT and hamiltonian cycles”, Boston Massachusetts: ACM, Jul. 2022, pp. 2240–2246. doi: 10.1145/3520304.3533952.

[4] J. Nüßlein, S. Zielinski, T. Gabor, C. Linnhoff-Popien, and S. Feld, “Solving (Max) 3-SAT via Quadratic Unconstrained Binary Optimization”. Accessed: Nov. 30, 2023. [Online]. Available: http://arxiv.org/abs/2302.03536

[5] M. Tao *et al.*, “A Work-Time Optimal Parallel Exhaustive Search Algorithm for the QUBO and the Ising model, with GPU implementation”, New Orleans, LA, USA: IEEE, May 2020, pp. 557–566. doi: 10.1109/IPDPSW50202.2020.00098.

[6] N. Fujimoto and K. Nanai, “Solving QUBO with GPU parallel MOPSO”, Lille France: ACM, Jul. 2021, pp. 1788–1794. doi: 10.1145/3449726.3463208.

[7] T. Okuyama, T. Sonobe, K.-i. Kawarabayashi, and M. Yamaoka, “Binary optimization by momentum annealing”, *Physical Review E*, no. 1, p. 12111, Jul. 2019, doi: 10.1103/PhysRevE.100.012111.

[8] D. Volpe, G. A. Cirillo, M. Zamboni, and G. Turvani, “Integration of Simulated Quantum Annealing in Parallel Tempering and Population Annealing for Heterogeneous-Profile QUBO Exploration”, *IEEE Access*, pp. 30390–30441, 2023, doi: 10.1109/ACCESS.2023.3260765.

[9] K. Nakano *et al.*, “Diverse Adaptive Bulk Search: a Framework for Solving QUBO Problems on Multiple GPUs”, 2022, doi: 10.48550/ARXIV.2207.03069.