TUHH
Hamburg University of Technology

Bachelor Thesis

Leonard Techel

Low-Connectivity State Space Exploration using Swarm Model Checking on the GPU

August 2, 2021

supervised by:
Prof. Dr. Sibylle Schupp
Sascha Lehmann

Hamburg University of Technology (TUHH)
Technische Universität Hamburg
Institute for Software Systems
21073 Hamburg

STS
Software
Technology
Systems

Abstract

Using small, independent verification tests, model checking large models with billions of states can be parallelized on GPUs. This approach works great on models with high connectivity. However, it fails when a model has only few edges between states or large portions of the state space are hidden behind bottleneck structures.

Past work on the *Grapple* model checker has tried different approaches including depth-limiting and alternating between breadth-first and depth-first search.

The main goal of this thesis is to: (1) create a systematic way of classifying a model as *low-connectivity* (2) minimize the amount of verification tests needed to maximize the state space coverage of said models. To do that, we provide an implementation of the *Grapple* model checker.

Contents

List of Figures

1 Introduction

In an explicit-state model checker, state space exploration of large models with billions of states is a time-consuming problem. Swarm Verification

Using swarm verification, the problem can be split into small, independent verification tests. Past work has shown that by executing these verification tests in parallel on GPUs, a high-speed model checker can be implemented [2].

To create the small, independent verification tests, diversification is used. Each verification test only covers a subset of the state space. Together, the verification tests nearly achieve full state space coverage. This approach works great on models with high connectivity. However, it fails when a model has only few edges between states or large portions of the state space are hidden behind bottleneck structures.

2 Related Work

3 Background

3.1 Model Checking

Model checking is a formal verification method that checks whether a state machine satisfies a specification. For example, the state machine of an elevator may be verified to meet the safety property of not opening the doors between floors. To do so, a *model checker* searches the state space for counterexamples, also called violations. When a violation is found, the path of state transitions that lead to the violation is reported back. Model checking can thus be divided into three main problems: Description of models through state machines, definition of the specification through temporal logic and algorithms that verify whether a state machine models a specification.

There are two main branches of model checking: Explicit-State Model Checking and Symbolic Model Checking. Explicit-State Model Checking can only verify finite state machines. In particular, the model has to have finite states, each state needs to be representable by a finite-size tuple containing its atomic propositions and the model changes state through execution of state transitions. To overcome these limitations and verify potentially infinite-size state machines or systems of unknown structure, Symbolic Model Checking uses the abstraction of *symbols*, each representing a set of states and transitions. Within this thesis, we are only considering explicit-state model checking.

Each state in a model is labelled with atomic propositions that hold true while the state is active. An example for such propositions are the current values of variables in a program at a given state. In a specification, different types of properties can then be expressed onto these propositions. Three common properties are reachability, safety and liveness: Reachability means that an atomic proposition holds true at some state in the future. Safety means that an atomic proposition holds true at all states in the future. Liveness means that an atomic proposition holds true infinitely often in the future, meaning that it does not happen that the atomic proposition never holds true. Within this thesis, we are only considering reachability and safety properties.

A challenge all model checking algorithms have to face is the *state explosion problem*. In an asynchronous model of n processes, each consisting of m states, the number of states grows exponentially by the number of processes, namely m^n. This means that even for small models, it is often not possible to fit all reachable states of the system into a computer's memory. Therefore, every model checking algorithm needs to reduce the state space in some sense. However, even then, a non-parallel algorithm may need a lot of physical time for exhaustive verification of the state space. In exhaustive verification, all states are visited and checked for a violation. [1, 3]

3.1.1 Parallelized model checking

Model checking can be speed up by parallelizing the state space exploration. For example, the Spin model checker uses a parallel breadth-first search, as described in the paper "Parallelizing the Spin Model Checker" by G. J. Holzmann [4]. In summary, this parallel

BFS algorithm works by allowing lock-free communication between threads using a shared, multidimensional queue array where, for each pair of threads, there is only one piece of memory to communicate over. Writing and reading to this piece of memory is coordinated by splitting the algorithm into two alternating phases. We are going to cover this algorithm more in-depth later on.

A major challenge in parallelized BFS is the communication overhead between threads: Shared memory does not allow to easily split the work onto a cluster of heterogeneous processors or the massively parallel architecture of a GPU on which thousands of threads can exist simultaneously.

3.1.2 Swarm Verification

Swarm Verification solves the challenge of parallelizing state-space search by splitting the state space exploration into many small, independent, memory-limited tasks called *Verification Tests* (VTs). Each VT only covers a small subset of the total state space and, using diversification techniques, uses a different search path.

The trick is that we do not care about exhaustive, 100% state space coverage: Instead, by executing all VTs and collecting their results, we still achieve nearly full state space coverage.

As the VTs are independent of each other, we can easily execute them on heterogeneous computers. Even further, as VTs are also memory-limited, we can massively parallelize them on devices with very limited resources like GPUs. [5]

3.2 CUDA

3.3 Grapple model checker

4 Implementation

4.1 Model Definition and State Generation

The states of a model support two operations: *successorGeneration* returns a successor to the current state, *violates* returns whether the state violates.

4.2 Queues

The queues support three operations: *push* adds an element at the back, *pop* removes and returns the first element from the front, *empty* tells whether the queue is empty.

4.3 Hash table

The hash tables support one operation: *markVisited* marks a state as visited and returns whether it was already visited before.

4.4 State Space Exploration

5 Experiments

5.1 Low-Connectivity Models

5.2 Optimizing Grapple on Low-Connectivity Models

6 Conclusion

Bibliography

[1] Edmund M. Clarke, Thomas A. Henzinger, and Helmut Veith. "Introduction to Model Checking". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Cham: Springer International Publishing, 2018, pp. 1–26. ISBN: 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8_1.

[2] Richard DeFrancisco et al. "Swarm model checking on the GPU". In: *International Journal on Software Tools for Technology Transfer* 22.5 (Oct. 2020), pp. 583–599. ISSN: 1433-2787. DOI: 10.1007/s10009-020-00576-x.

[3] Gerard J. Holzmann. "Explicit-State Model Checking". In: *Handbook of Model Checking*. Ed. by Edmund M. Clarke et al. Cham: Springer International Publishing, 2018, pp. 153–171. ISBN: 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8_5.

[4] Gerard J. Holzmann. "Parallelizing the Spin Model Checker". In: *Model Checking Software*. Ed. by Alastair Donaldson and David Parker. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 155–171. ISBN: 978-3-642-31759-0. DOI: 10.1007/978-3-642-31759-0_12.

[5] Gerard J. Holzmann, Rajeev Joshi, and Alex Groce. "Swarm Verification". In: *2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. 2008, pp. 1–6. DOI: 10.1109/ASE.2008.9.