Experimental Grapple Implementation

Leonard Techel, 21510495

June 26, 2021

This document denotes the necessary steps for an experimental implementation of the Grapple model checker.

Contents

1 Model Definition

Models are either defined using a model specification language like Promela/DVE or inferred from program code using e.g. LLVM. For our task, using models written in a specification language is sufficient as we are only going to evaluate theoretical models with known state space size.

Model databases:

- BEEM DVE models: `https://paradise.fi.muni.cz/beem/`
- Example PROMELA models: `https://github.com/nimble-code/Spin/tree/master/Examples`

Models are usually precompiled from their specification language into C code or an LLVM representation.

The Waypoints model used in the Grapple paper is taken from [5].

2 State Generation

The state generation strategies of model checkers can be divided into *a priori* and *on-the-fly* generation. In an a priori generation, the state space graph is completely known before running the verification algorithm. In an on-the-fly generation, the successors of each state are created on the fly during verification.

The Grapple model checker depends on on-the-fly state generation on the GPU. In theory, the C code generated from a model can be run directly on the GPU. However, as the generated C code representing a model from tools like SPIN and DIVINE heavily depends on branches (if-else conditions, switch statements), executing it on a GPU using CUDA comes with a huge performance penalty as it executes all branches and later decides which are valid. As of this, the on-the-fly state generation for GPUs algorithm from [1] needs to be used.

- Task 1.1: Implement Algorithm 3 from [1]
- Task 1.2: Find an easy, systematic way to re-use `.dve/.pml` models with this approach

3 Queues

The state-space exploration loop of the Grapple model checker is based on the parallel BFS algorithm from [4]. In order for the parallel threads to communicate, the algorithm is using two $N \times N$ queues where N denotes the number of threads. These queues are implemented best using CUDA's Atomic Functions [2]. Within Python + Numba, multidimensional NumPy arrays can be used in conjunction with the Numba implementation [7] of those functions.

- Task 2.1: Implement Queues

Prior art of such a queue can be found in the divine-cuda software. [1]

4 Search Algorithm

Having the model, state generation and queues ready, it is time to finally implement the basic Grapple model checker [3]:

- Task 3.1: Prepare the data structures on the host, e.g. input/output queues, initial state, ...
- Task 3.2: Implement the BFS Grapple search algorithm and find out how the queues are swapped
- Task 3.3: Execute the model check

[1] `https://divine.fi.muni.cz/download/discontinued/divine-cuda.tar.gz`

- Task 3.4: Compare results with the evaluation from [3]

The `mix(a, b, state)` function is taken from [6] and can be implemented using a Numba CUDA device function.

As of their size, the input and output queues need to be within the GPUs *global memory* and, as such, actually have the shape $K \times N \times N \times I$ where K denotes the number of VTs / Warps, N denotes the number of threads per VT and I the number of queue slots per thread per VT.

The (global) waypoints should be stored within the GPUs *constant memory* and are checked within the VTs threads.

The hash tables of each VT is stored within the *shared memory* of its block/warp.

5 Variations of the search algorithm

Within [3], multiple variants of the search algorithm are evaluated.

- Task 4.1: Implement PDS and compare results with the evaluation from [3]
- Task 4.2: Check the other variations, i.e. process-PDS, scatter-PDS, depth-limited PDS, ...

References

[1] Ezio Bartocci, Richard DeFrancisco, and Scott A. Smolka. "Towards a GPGPU-Parallel SPIN Model Checker". In: *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*. SPIN 2014. San Jose, CA, USA: Association for Computing Machinery, 2014, pp. 87–96. ISBN: 9781450324526. DOI: 10.1145/2632362.2632379.

[2] *CUDA C++ Programming Guide*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[3] 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.

[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 Techniques". In: *IEEE Transactions on Software Engineering* 37.6 (2011), pp. 845–857. ISSN: 1939-3520. DOI: 10.1109/TSE.2010.110.

[6] Bob Jenkins. *A Hash Function for Hash Table Lookup*. URL: https://burtleburtle.net/bob/hash/doobs.html.

[7] *Numba for CUDA GPUs*. URL: https://numba.readthedocs.io/en/0.53.1/cuda/.