Parallel Model Checking of B Specifications[⋆]

Jens Bendisposto, Philipp Körner and Michael Leuschel

Heinrich-Heine Universität Düsseldorf
{bendisposto,leuschel}@cs.uni-duesseldorf.de

1 Introduction

Model checking of a specification using ProB can easily become a task with a long running time. However, the task can very easily be parallelized. If we had a transition system (the state space) and an invariant we could partition the vertices of the state space into k partitions of equal size and then check the invariant on k processors in parallel. However, in reality we cannot afford to precompute the state space in advance. We need to explore the state space itself in parallel. This imposes some challenges that the implementation has to address. Firstly, we have to avoid duplicated invariant checking or at least make sure that duplicate invariant checking does not happen too often in practice. Also in contrast to an offline algorithm, the online version cannot split up the workload fairly. This reduces the scalability of the approach. In particular, the scalability depends on the model. For example, let us consider a counter specification that has an integer variable x ranging from zero to a billion. The variable is initialized with the value zero, and the model has a single event that increments x by one. Assuming that we have k processors, the offline version can easily partition the state space and ensure that every processor gets the same amount of work. For instance, it can decide to use $x \bmod k$ to determine the processor for a state. The online algorithm has only a limited sight. It processes only one state at a time because each state has a single successor state.

2 Implementation

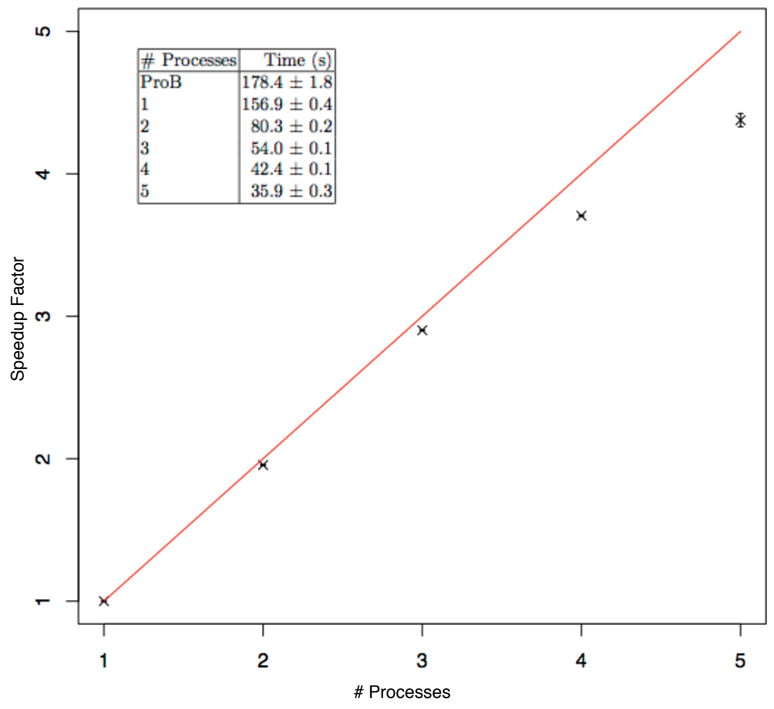
The main goal for the parallelization framework is performance. We have implemented the framework in C using the Zero MQ framework[1]. The typical overhead for processing a state is in the order of 100 μs. This is an order of magnitude lower than the Prolog processing time even for very simple cases.

3 Experimental results

In this extended abstract we will give only some experimental results for a single example: a kernel scheduler taken from [?]. A scheduler is well suited for parallel model checking because of its nondeterministic nature. In this particular model, the state space consists of 24580 states. We used a Mac Pro computer with a 6-Core Intel Xeon processor and 16 GB memory. The results are shown in the figure below. Note that the parallel version is a bit faster than the reference ProB version even if it only uses a single process. The reason is that the model checking algorithm of the parallel version uses a slightly reduced version of invariant checking. For instance, it does not yet handle timeouts in the same way the nonparallel version of ProB does. The red line in the chart shows the optimal

[⋆] This research is being carried out as part of the DFG funded research project GEPAVAS).
[1] http://www.zeromq.org/

scaling[2]. We can see that our implementation is very close to the optimal scaling for up to three processes. Even for five processes the result is very good. We have also carried out more experiments with different models. If the model itself has some nondeterministic branching, i.e., it is suited for an online algorithm, the parallel version of ProB performs well.



# Processes	Time (s)
ProB	178.4 ± 1.8
1	156.9 ± 0.4
2	80.3 ± 0.2
3	54.0 ± 0.1
4	42.4 ± 0.1
5	35.9 ± 0.3

4 Conclusion and future work

We have demonstrated that our implementation scales very well on multicore processors for examples that are suited for an online algorithm. In the talk we will discuss more benchmarks as well as using the parallel model checker for assertion checking and data validation which also works very well. In the future we plan to use the implementation in a distributed way on the Amazon EC2 cloud in order to increase the total number of processes.

[2] At some point the optimal scaling becomes a constant. In the case of the scheduler the maximal time for a single state is about 30 ms, therefore the model checking process can never be faster than 30 ms, which is a scaling factor of about 5300.