TOWARDS PARALLELIZATION OF SIMULATION-BASED REACHABILITY

By

Bolun Qi

Senior Thesis in Computer Engineering

University of Illinois at Urbana-Champaign

Advisor: Sayan Mitra

May 2016

Abstract

To check bounded time invariant properties of models with nonlinear dynamics, one promising method is called simulation-based verification. This involves: (a) generating numerical simulations of the ODE from a finite set of representative initial states that cover the whole (uncountably many) initial set, say, (b) bloating each of these simulations by some factor such that the bloated tubes together over-approximate the reachable states from, and (c) checking if this computed over-approximation is adequate for proving invariance; otherwise, add more representative initial states to obtain a more precise over-approximation and repeat from (a). Compare-Execute-Check-Engine (C2E2) is such a tool for verifying bounded time dynamical and hybrid systems models using simulation-based reachability analysis. To make the reachable set computation more accurate, it is preferable to start from smaller initial sets, which requires C2E2 to partition large initial sets into smaller covers at first. However, the number of initial covers will increase exponentially with dimensionality of the system and the fidelity of each cover. Currently, C2E2 does invariant checking for the reachable sets from each cover sequentially, which makes the running time increase proportionally to the initial covers.

Simulation-based approaches are naturally parallelizable due to the fact that simulations and reachable set computation for each initial condition considered can be computed independently. In this paper, we introduce the parallelization of C2E2 to utilize the computational power of multi-core CPU and improve the efficiency of tools. The parallel algorithm is implemented using thread library OPENMP. We evaluate the improvement of performance on four different models. Comparison with the sequential counterpart shows a maximum speedup of 7.3x on a four core Intel CPU I7-4790K processor.

Subject Keywords: Reachability analysis, Verification, Hybrid Systems, Parallel Programming, OPENMP

Acknowledgments

I would first like to thank my advisor, Associate Professor Sayan Mitra, for his continued support and guidance throughout the research and thesis writing. Without his helping, this thesis would not have been possible.

I would also like to thank Phd students Chuchu Fan, Parasara Sirdhar Duggirala and Associate Professor Mahesh Viswanathan for their guidance with implementing exciting features of C2E2 and their outstanding paper on which my thesis is based.

Contents

1. Introduction

Reachability analysis is a standard technique for safety verification. Verifying nonlinear, switched and hybrid system models using numerical simulation has been recently studied in detail some papers [1,2,3,4,5,8,9,14,15]. Since exact computation of reachable states is in general intractable, simulation-based over-approximation computation methods have designed to solve this problem. The simulation-based procedure for this problem first chooses finite sampling from the initial set and generates numerical approximations of the behaviors. Then, it computes an over-approximation of reachable states from initial states by bloating these simulation traces based on annotation from users. If this over-approximation proves safety or produce a counterexample, then the algorithm decides the safety of system. Otherwise, it draws more samples from the initial set and repeats the whole process. Precision and scalability have been the two challenges with such simulation based methods. Once the dimensions of system increase or the precision of the system increase, the number of sampling points increases exponentially, which means the running time of the tool increases exponentially.

In order to bring efficiency to this algorithm, we need to utilize the nature of simulation-based approaches, which are naturally parallelizable due to the fact that simulations and reachable set computation for each initial condition considered can be computed independently. The advent of multi-core architecture has provided tremendous computing power. Our goal is to utilize these powerful parallel architectures to speed up the performance of simulation-based approaches, and design better implementation of tools to maintain scalability when running in CPU with more threads. In particular, We use thread library OPENMP [6] to parallelize simulations and

reachable set computation for each initial point, and I change the implementation of the tool to optimize performance.

The organization of this thesis is as follows. Chapter 2 presents an overview of C2E2 and thread library OPENMP. Chapter 3 presents background on the verification problem, the details of the parallelized algorithm and its implementation. Chapter 4 illustrates the performance gain and scalability for different systems. Chapter 5 concludes the thesis.

2. Background

2.1 Overview of Simulation-based Reachability tool C2E2

C2E2 is a tool for checking bounded time invariant properties of nonlinear hybrid automaton models through reachability analysis.

The architecture of the tool is shown in Figure 1. The frontend provides a model parser, connects to the verification engine, and provides a property editor and a plotter for the verification result. Once the frontend reads the input file which can be either a Stateflow model or hyxml file, it generates simulation code based on a differential equation of system. The properties are obtained from the input file or from the user through the property editor. The simulation code is compiled using a validated simulation engine using Computer Assisted Proofs in Dynamic Groups (CAPD) library [10]. This compiled code and the property are used by the C2E2 verification algorithm during runtime. C2E2 also uses GNU Linear Programming Kit (GLPK) [7] and OPENMP to analyze reachtubes in parallel. The verification result and computed reachable set are sent to the frontend for visualization.

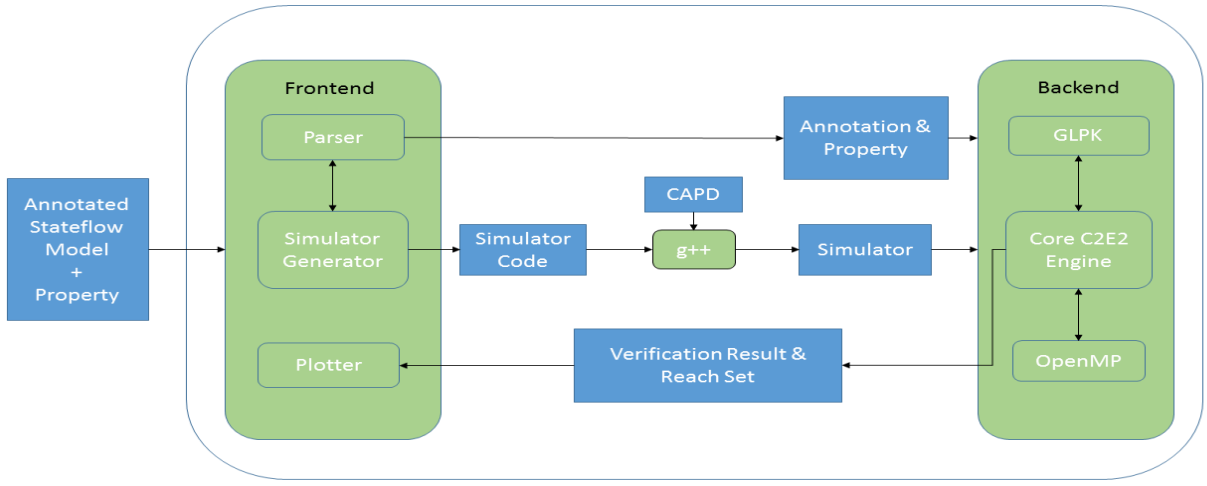

Figure 1. Architecture of C2E2

2.2 OPENMP Overview

OPENMP is an application programming interface (API) that supports shared memory multiprocessing programming [6]. OPENMP is a shared memory model, where all threads have access to the same, globally shared, memory. Workload in OPENMP is distributed between threads. Within parallel regions, variables can be either shared among all threads or duplicated privately for each thread, and threads can communicate with others by sharing variables. OPENMP programs work according to the fork-join model. The execution begins as a single process, where the master thread takes in charge. When execution enters parallel region, the master thread will create teams of threads and execute together. At the end of the parallel region, threads in the team will synchronize and only the master thread continues execution. The advantages of OPENMP are good performance and scalability when supported by many compilers and the small programming effort. Adding support for parallel programming to C/C++ in a very clean way. Unlike other thread libraries, small changes can make an existing program run on multiple processors in parallel.

3. Parallel Algorithm and Implementation

3.1 Background

Consider an autonomous dynamic system:

$$\dot{x} = f(x)$$

A trajectory of the system is a function $\xi: \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ such that $\forall x_0 \in \mathbb{R}^n$ and $\forall t > 0$, and $\xi(x_0, t)$ satisfies the differential equation, where x_0 is an initial point of the system.

3.1.1 Simulation

A simulation (x_0, τ, ϵ, T) of the system $\dot{x} = f(x)$ is a sequence of time stamped sets $(R_0, t_0), (R_1, t_1), \cdots (R_n, t_n)$. The sets satisfy the following properties: (1) Each R_i is a compact set in $\mathbb{R}^n$, and the diameter of R_i is less than ϵ, where ϵ is the error bound of simulation. (2) t_n is equal to T and for each i, $t_{i+1} - t_i = \tau$, where τ is the time step of simulation. (3) The trajectory starting from x_0 within time $[t_i, t_{i+1}]$ is contained in R_i.

C2E2 uses simulation library CAPD [10] and ODEint [13] to compute the simulation trajectory of linear and nonlinear dynamical system models. CAPD generates a sequence of states and error bounds, and R_i is represented as hyper rectangles.

3.1.2 Reachtube

The Reachtube (D, τ, T) of system $\dot{x} = f(x)$, where the set of initial states $D \subseteq \mathbb{R}^n$, is a sequence of time stamped sets from (R_0, t_0) to (R_n, t_n). The sets satisfy the following properties. (1) Each R_i is a compact set in $\mathbb{R}^n$, (2) t_n is equal to T and for each i, $t_{i+1} - t_i = \tau$, where τ is the time step of the simulation, (3) $\forall x_0 \in D$ and $\forall t \in [t_i, t_{i+1}]$, the solution $\xi(x_0, t) \in R_i$.

Reachtubes are generally difficult to compute. The algorithm in C2E2 requires the user to provide an annotation called discrepancy function to compute reachtubes from simulation trajectories. Details of calculating reachtubes are described in [3,8].

3.1.3 Annotation

$\beta: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ is called the discrepancy function for system $\dot{x} = f(x)$ if it satisfies the following properties:

(1) $\forall x, x' \in \mathbb{R}^n$, and $\forall t > 0$, $\|\xi(x,t) - \xi(x',t)\| \leq \beta(x,x',t)$,

(2) as $x \rightarrow x', \beta(x,x',t) \rightarrow 0$,

(3) $\forall \varepsilon > 0, \forall x, x' \in \mathbb{R}^n, \exists \delta \; such \; that \; \forall t, \|x - x'\| < \delta \Rightarrow \beta(x,x',t) < \varepsilon$.

The annotation β gives an upper bound on the distance between two neighboring trajectories, which is provided by the users.

3.1.4 Verification Problem

A bounded-time safety verification problem is addressed by the following variables:

1) An n dimensional dynamical system such as $\dot{x} = f(x)$,
2) A compact set $\Theta \in \mathbb{R}^n$, which is called initial set,
3) An (possibly) open set $U \in \mathbb{R}^n$, which is called unsafe sets,
4) A time bound $T > 0$.

A state x is reachable if $\exists x_0 \in \Theta$ and a time $t \in [0,T]$ such that $x = \xi(x_0,t)$. A set of all reachable states from Θ is donated by $Reach(\Theta,T)$. Given the system and initial set, we want to design algorithms to decide whether every reachable state (within time T) is safe, or in other words, whether $Reach(\Theta,T) \cap U = \emptyset$. A sequence of papers [3,9] presented algorithms for

solving this problem for nonlinear dynamic, switched, and hybrid systems. In this thesis, I will present an approach to parallelize the original algorithms, helping improve the efficiency of existing methods.

3.2 Parallelized Algorithm

The parallelized verification algorithm is shown in Algorithm 1. It takes as inputs the initial set Θ, the unsafe set U, the time-bound T and delta δ.

Three data structures are used in this algorithm. First, C is returned by the $Partition$ function, which is a set of δ-balls around initial states θs, the union of which completely covers initial set Θ. The second structure R is the union of reachtubes from each cover that has been proven safe. The last structure N comprises of those reachtubes that partially intersect with unsafe set U but the corresponding simulation is not contained in the unsafe set.

Algorithm 1: Parallelized Verification Algorithm
Input: Θ, U, T, δ
$\varepsilon \leftarrow \varepsilon_0$; $C \leftarrow \emptyset$; $R \leftarrow \emptyset$; $N \leftarrow \emptyset$
$C \leftarrow <Partition(\Theta, \delta), \delta, \varepsilon>$
While $C \neq \emptyset$ **do**
    OMP Task
        For each $(\theta, \delta, \epsilon) \in C$ **do**
            $\Psi = \{(R_i, t_i)\}_{i=1}^{k} \leftarrow Simulate(\theta, \tau, \epsilon, T)$
            $\beta = ComputeAnnotation(\Psi, \delta, \epsilon)$
            $D \leftarrow \Psi \oplus \beta$
            If $D \cap U = \emptyset$ **then**
                $R \leftarrow R \cup D$
                $C \leftarrow C \backslash (\theta, \delta, \epsilon)$
            Else if $\exists k, R_k \subseteq U$ **then**
                Return (UNSAFE, Ψ)
            Else
                $N \leftarrow N \cup D$
    End of OMP
    If $N \neq \emptyset$ **do**
        $C \leftarrow < Partition\left(\Theta, \frac{\delta}{2}\right), \frac{\delta}{2}, \varepsilon >$
        $R \leftarrow \emptyset$
        $N \leftarrow \emptyset$
End of While
Return (Safe,R)

OMP label is the parallel region of OpenMP. Discarding the OMP part in the algorithm, the algorithm works as follows:

1) At the beginning of the algorithm, C contains a sequence of covers $(\theta, \delta, \epsilon)$.
2) For each $(\theta, \delta, \epsilon)$ in C, a simulation Ψ, which is a sequence of $\{(R_i, t_i)\}$, is generated.
3) Bloat the simulation trace by some factor such that the result is guaranteed to be an overapproximation of $Reach(B_\delta(\theta), T)$.
4) Check whether the bloated tube intersects with U. If the bloated reachtube from θ is disjoint with unsafe set U, it is proven safe. If any piece of simulation trace is contained in U, then the system is declared unsafe with the counter-example. Otherwise, the safety of the reachtube is unknown, and the reachtube will be pushed to unknown set N.
5) If the unknown set N is not empty, we will refine Θ such that it will be covered by finer covers, and go to step 2.

The previous algorithm of C2E2 evaluates each $(\theta, \delta, \epsilon)$ in C sequentially. As the number of initial covers increases exponentially with increasing dimension of the system and size of the initial set, the running time of verification increases exponentially as well. Simulation-based approaches are naturally parallelizable due to the fact that simulations and reachableset computation for each initial condition considered can be done independently. To improve the performance of tools, I use the thread library OPENMP to evaluate the initial covers in parallel.

Within parallel regions as denoted by OMP, the master thread will first go through each $(\theta, \delta, \epsilon)$ in C and create one task for each $(\theta, \delta, \epsilon)$, then store these tasks in a task list. Child threads will execute tasks from the task list until is empty. This is accomplished using OPENMP Task API.

The TASK construct defines an explicit task, which may be executed by the encountering thread,

Algorithm 2: Parallelized Verification Algorithm with Lock
Input: Θ, U, T, δ
$\varepsilon \leftarrow \varepsilon_0;\ C \leftarrow \emptyset;\ R \leftarrow \emptyset;\ N \leftarrow \emptyset$
$C \leftarrow <Partition(\Theta, \delta), \delta, \varepsilon>$
While $C \neq \emptyset$ **do**
  OMP Task
    For each $(\theta, \delta, \epsilon) \in C$ **do**
      $\Psi = \{(R_i, t_i)\}_{i=1}^{k} \leftarrow Simulate(\theta, \tau, \epsilon, T)$
      $\beta = ComputeAnnotation(\Psi, \delta, \epsilon)$
      $D \leftarrow \Psi \oplus \beta$
      If $D \cap U = \emptyset$ **then**
        OMP Lock
          $R \leftarrow R \cup D$
          $C \leftarrow C \backslash (\theta, \delta, \epsilon)$
        End of OMP Lock
      Else if $\exists k, R_k \subseteq U$ **then**
        Return (UNSAFE, Ψ)
      Else
        OMP Lock
          $N \leftarrow N \cup D$
        End of OMP Lock
  End of OMP
  If $N \neq \emptyset$ **do**
    $C \leftarrow < Partition\left(\Theta, \frac{\delta}{2}\right), \frac{\delta}{2}, \varepsilon >$
    $R \leftarrow \emptyset$
    $N \leftarrow \emptyset$
End of While
Return (Safe,*R*)

or deferred for execution by any other thread in the team. By setting each argument in the function call to either private variable or shared variable we can prevent race condition. After that, OPENMP will synchronize all the threads. Note that in the algorithm, multiple threads can access *R* and *N* at the same time, which will create the race condition. A Critical section is essential to prevent such race condition. This is achieved using OPENMP lock. Each thread can only access critical section after it gets the lock, and it releases the lock when it exits the critical section. The modified algorithm with lock in shown in Algorithm 2.

3.3 Implementation

When I implemented the parallelized version of C2E2, I noticed that there are some deficiencies in the current implementation and these drawbacks weaken the tool in term of both performance and scalability.

3.3.1 Thread Safe GLPK

C2E2 uses the GLPK library, an open source linear programming solver library, to analyze the intersection between reachtube rectangles and unsafe set *U*. However, I notice that the GLPK library is not thread safe due to its implementation of memory management functions. GLPK uses shared data that suffers from the race condition in multithreaded execution. Therefore, I created a critical section when using GLPK library to analyze reachtube, which may reduce the speed of the tool. To solve this problem, I analyzed GLPK memory management implementations, making them thread-local to ensure thread safety. The thread-safe GLPK are used per thread to analyze different reachtubes in parallel. Table 1 shows running time for the Brusslator model that running in thread safe GLPK and thread unsafe GLPK. The overall performance increases around 4 percent in 8 threads case.

Table 1. Performance gain for two dimensional dynamic system with thread safe GLPK

Brusslator	1 thread	2 threads	4 threads	8 threads
Thread unsafe GLPK	561.83 s	332.34 s	265.34 s	250.14 s
Thread Safe GLPK	561.69 s	328.79 s	257.69 s	240.52 s

3.3.2 Executable and File IO

C2E2 is designed to verify different models, where each model is a unique system. In order to accommodate different systems, C2E2 generates simulation, invariant checking and guard checking source code based on the model, and these source codes will be compiled to executable during runtime. When the verification process starts, the C2E2 backend verification engine will call these executables and use file I/O to transfer data. This implementation is problematic when integrated into parallel C2E2 because the disk resources and CPU resources are limited. The running time for calling executables and file I/O for different numbers of threads are listed in Table 2.

Table 2. Performance for calling executable and file write with different number of threads

	1 thread	2 threads	4 threads	8 threads
Executable	0.0024 s	0.0026 s	0.003 s	0.0052 s
File Write	0.00018 s	0.00019 s	0.00025 s	0.00037 s

As we can see from Table 2, the running time of file writing and calling executables in 8 threads is doubled compared to the 2 threads case. For each cover point, C2E2 writes files 4 times, reads files 4 times and calls the executables 3 times. The scalability of the parallelized algorithm is significantly affected due to the heavy load caused by calling executables and file I/O.

In order to make the parallelized C2E2 scalable, I used dynamic libraries to reduce the number of file I/O and executable calls during runtime. Dynamic libraries are libraries that are loaded by programs when they started. When a new dynamic library is installed properly, all programs that links to this library and start afterward automatically use the new dynamic library. It makes your program more flexible by allowing you override old functionality you have. The idea is that to

compile source codes generated for each model not to executable, but to a dynamic library. Then I linked the back-end engine to this dynamic library. Data transfers between dynamic library and back-end happen in memory instead of disk. Using dynamic libraries, the number of executable calls is reduced to one time for each cover point since simulator is still left as an executable. File write and file read are also reduced to one time.

Performance and scalability of the parallelized algorithm increase significantly as shown in Table 3. The running time is calculated for five stages. The first stage is simulation, where C2E2 simulates initial points, and it has one executable and one file write. The second stage is bloating, where C2E2 bloats reachtube by using user-provided annotation, and this stage has one file read. The third stage is verification, where C2E2 first checks invariant and guard and then checks if reachtube intersects with unsafe set. Without dynamic library, this stage contains two file writes, two file reads and two executables. With dynamic library, these file I/O and executable have been eliminated. The fourth and fifth stages execute sequentially. The fourth stage checks if there exists a mode transition for the hybrid system. The last stage refines the initial set if there are some rectangles of the reachtube that partially intersect with unsafe set.

Table 3. Performance table for two dimensional dynamic systems without dynamic library

Brusselator	1 thread	2 threads	4 threads	8 threads
Running time	561.69 s	328.79 s	257.69 s	240.52 s
Simulation	172.33 s	82.58 s	49.15 s	42.9 s
Bloating	24.54 s	15.24 s	9.75 s	7.4s
Verification	363.53 s	230.65 s	198.41 s	189.38 s
Next Set	2.4e-05 s	2.5e-06 s	2.2e-05 s	3.4e-05 s

Refine	0.23 s	0.14 s	0.13 s	0.14 s

Table 4. Performance table for two dimensional dynamic system with dynamic library

Brusselator	1 thread	2 threads	4 threads	8 threads
Running time	246.74 s	138.85 s	87.82 s	77.55 s
Simulation	172.05 s	83.53 s	50.06 s	42.7 s
Bloating	24.51 s	16.21 s	9.12 s	7.36 s
Verification	49.95 s	38.54 s	28.49 s	27.21 s
Next Set	1.9e-05	1.8e-05	3.2e-05	2e-05s
Refine	0.13s	0.13s	0.13s	0.13s

Tables 3 and 4 shows the performance of Brusselator without and with dynamic library. Comparison of the two tables shows that performance has increased significantly using dynamic library. Eight threads with dynamic library is 3.1 times faster than. The performance has scaled up 2.33 times on comparing the 8 threads with thread without dynamic library, and the performance scales up 3.31 times with dynamic library.

4. Performance analysis and discussion

To measure the performance of the new parallelized algorithm and implementation, I test it on the benchmarks of Brusselator, Moore Greitzer, and Linear thermostat. The Brusselator is a theoretical model for a type of autocatalytic reaction with two variables in total. The Moore-Greitzer is a model that attempts to model the non-linear phenomena of rotating stall and surge of an axial compression system with two variables. Linear thermostat is a simple hybrid system simulating the temperature control system of the building with one variable.

All benchmarks run on Intel Core I7-4790k, 4 core, 8 threads, 4.0 GHZ, 16 GB RAM processor. Table 4 shows the running time of the original and the parallelized version of C2E2 on Brusselator. TableS 5 and 6 show running time for Moore-Greitzer and Linear thermostat.

Table 5. Performance table for Moore Greitzer -- two dimensional dynamic non-linear system

MooreGreitzer	Original	1 thread	2 threads	4 threads	8 threads
Running Time	184.12 s	90.15 s	60.16 s	38.88 s	32.28 s
Simulation	53.85 s	53.57 s	38.69 s	23.00 s	18.26 s
Bloating	12.73 s	12.84 s	7.56 s	4.54 s	3.39 s
Verification	117.43 s	24.65 s	13.90 s	11.32 s	10.59 s
NextSet	1.3e-05 s	1.9e-05 s	1.8e-05 s	2.2e-05 s	2.3e-05 s
Refine	0.042 s	0.043 s	0.039 s	0.039 s	0.039 s

Table 6. Performance table for Thermostat -- one dimensional hybrid linear system

Thermostat	Original	1 thread	2 threads	4 threads	8 threads
Running Time	55.23 s	45.40 s	30.69 s	19.62 s	17.60 s
Simulation	26.79 s	27.89 s	20.62 s	12.92 s	11.90 s
Bloating	12.87 s	13.04 s	7.47 s	5.01 s	4.67 s
Verification	15.53 s	4.46 s	2.53 s	1.51 s	1.01 s
NextSet	0.002 s	0.003 s	0.003 s	0.004 s	0.004 s
Refine	0.00 s	0.00 s	0.00 s	0.00 s	0.00 s

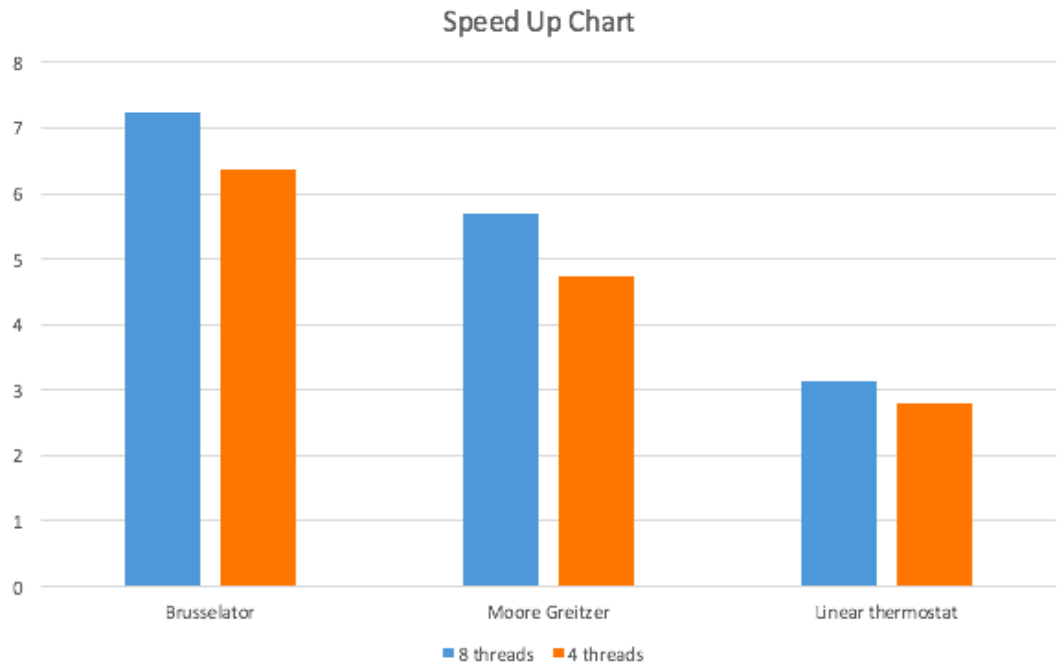

Figure 2 Performance gain for three systems

Figure 2 illustrates the speed-up obtained from the new implementation and new algorithm. The speed-up bar compares the running time of original C2E2 and parallelized C2E2 with 8 threads. A maximum speed-up of 7.3x is obtained for the two-dimensional systems. The performance

gain is different for different systems, and it depends on complexity and dimensionality of the system. Non-linear models will have better scalability since the simulation for non-linear models' simulation need larger computational power.

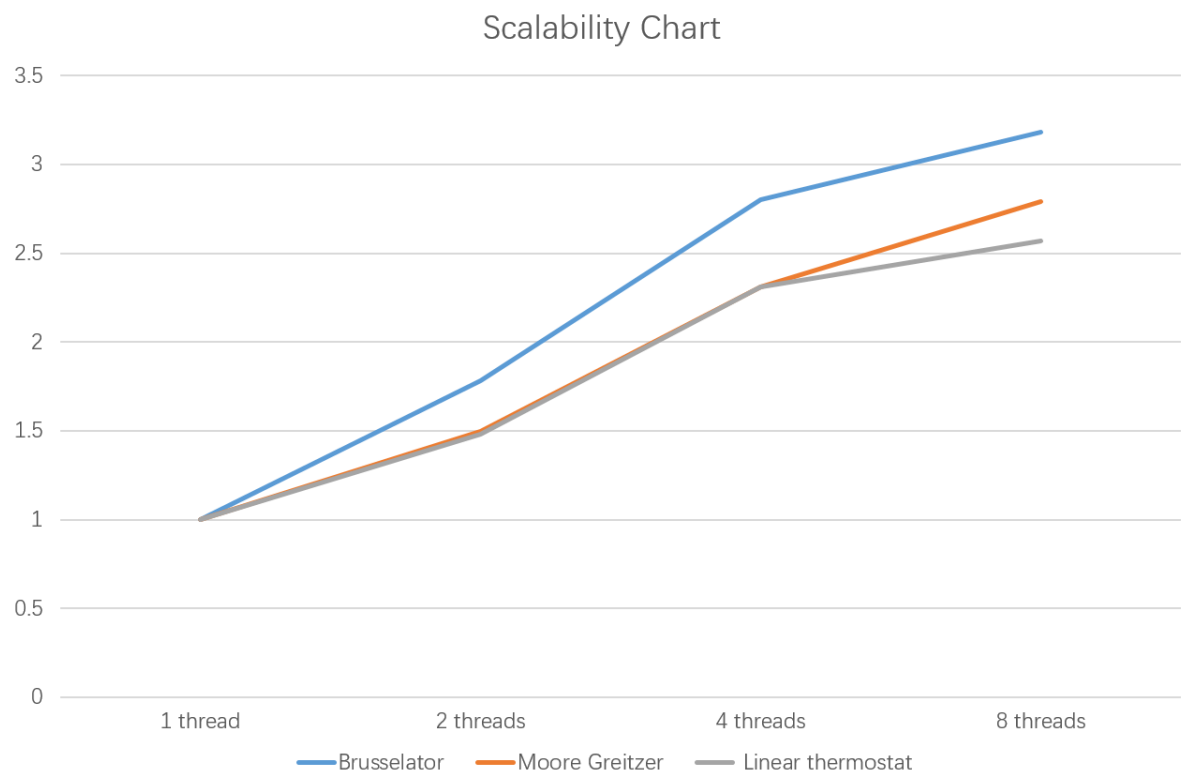

Figure 3. Scalability of three systems

Figure 3 illustrates the scalability obtained from the new implementation and new algorithm. We are comparing the running time of 4 and 8 threads cases with original C2E2 running time. Observe that the performance of parallelized algorithms improves with the increase of the number of threads in the machine. Note that the scalability decreases significantly from 4 threads to 8 threads. This is because Intel I7 4790K only has four physical cores. If we just increase threads of tool, the system will follow round-robin scheduling and keep performing context switch, which has some effect on the performance. The performance chart signifies that the

performance of parallel C2E2 can scale automatically with future multicore processors with a higher degree of parallelism.

5. Conclusion and future work

In this thesis, I presented a parallelized implementation of the verification algorithm used in the tool C2E2 and several optimizations. Simulation-based reachability is a good candidate for parallelization because each initial cover is independent to its neighbor. From the profiling of parallel execution, we learned that the draw-back of scalability are from massive file IO and running binary executable from code. Therefore, we decided to use dynamic libraries to minimize the file IO and executable. Results show that the performance of the verification algorithms for systems can be considerably improved using modern multi-core processors. The use of CPU has shown a promising performance gain in reachability analysis. The parallelized algorithm is implemented in the tool C2E2.

The future work of this project is to achieve higher order parallelism. One option is to utilize GPU computational power. GPUs have many more computational units than CPUs, API such as CUDA [12] have been utilized in verification tools such as XSpeed [11]. One way to use CUDA is to replace the current simulator CAPD with ODEINT [13], where ODEINT can utilize the power of GPUs by means of CUDA and Thrust, which is a STL-like interface for the native CUDA API.

References

[1] Donzé, A., & Maler, O. (2007). Systematic simulation using sensitivity analysis. In *Hybrid Systems: Computation and Control* (pp. 174-189). Springer Berlin Heidelberg.

[2] Duggirala, P. S., Fan, C., Mitra, S., & Viswanathan, M. (2015, July). Meeting a powertrain verification challenge. In *Computer Aided Verification* (pp. 536-543). Springer International Publishing.

[3] Duggirala, P. S., Mitra, S., & Viswanathan, M. (2013, September). Verification of annotated models from executions. In *Proceedings of the Eleventh ACM International Conference on Embedded Software* (p. 26). IEEE Press.

[4] Fan, C., Duggirala, P. S., Mitra, S., & Viswanathan, M. (2015). Progress on powertrain verification challenge with C2E2.

[5] Huang, Z., & Mitra, S. (2014, April). Proofs from simulations and modular annotations. In *Proceedings of the 17th international conference on Hybrid systems: computation and control* (pp. 183-192). ACM.

[6] Dagum, L., & Enon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, *5*(1), 46-55. [7]

Makhorin, A. (2014). GLPK (GNU Linear Programming Kit), 2000.
Chicago

[8] Huang, Z., Fan, C., Mereacre, A., Mitra, S., & Kwiatkowska, M. (2014, July). Invariant verification of nonlinear hybrid automata networks of cardiac cells. In *Computer Aided Verification* (pp. 373-390). Springer International Publishing.

[9] Duggirala, P. S., Wang, L., Mitra, S., Viswanathan, M., & Munoz, C. (2014). Temporal precedence checking for switched models and its application to a parallel landing protocol. In *FM 2014: Formal Methods* (pp. 215-229). Springer International Publishing.

[10] CAPD. Computer assisted proofs in dynamics. Available: http://www.capd.ii.uj.edu.pl/, 2002.

[11] Ray, R., Gurung, A., Das, B., Bartocci, E., Bogomolov, S., & Grosu, R. (2015). XSpeed: Accelerating Reachability Analysis on Multi-core Processors. In *Hardware and Software: Verification and Testing* (pp. 3-18). Springer International Publishing.

[12] CUDA, Available: http://www.nvidia.com/object/cuda_home_new.html

[13] ODEINT Available: http://headmyshoulder.github.io/odeint-v2/

[14] Duggirala, P. S., Mitra, S., Viswanathan, M., & Potok, M. (2015). C2E2: a verification tool for stateflow models. In *Tools and Algorithms for the Construction and Analysis of Systems* (pp. 68-82). Springer Berlin Heidelberg.

[15] Fan, C., & Mitra, S. (2015). Bounded Verification with On-the-Fly Discrepancy Computation. In *Automated Technology for Verification and Analysis* (pp. 446-463). Springer International Publishing.