CPSC 59700 Project Proposals - Summer 20222

1 CPSC59700 Project Report Guidelines

An important part of the course is a project on a topic of your choice that is closely related to the course material. You are welcome to work individually or in groups of up to two students. Expectations for project outcomes are associated with group size. Projects are required to have final written reports of approximately 8 to 13 pages. These steps should be followed:

1. Step 1 - Definition of the project (Weeks 1 to 3): This is a brainstorming phase during which you will start a discussion thread in BB relating to your research project. By the end of this process and clear project theme should have been defined.
2. Step 2 - Work in Progress (Weeks 3 to 8): During this period students will be working on their research projects and can have online discussions with the instructor.
3. Step 3 - Writing the final Report: This should start by week 6 or week 7 the latest. The report should be written using the IEEE Trans. Journal latex template (file project.zip). A reference document ("How to write a great research paper" by Simon Jones) to be consultedfor your project report writing is posted in BB (Week 2).

This assignment will be done exclusively in Latex. A template file is provided use it to prepare your report document(project.zip). The submission should be a zip file that contains the latex source file and images that were used to create the pdf file. A detailed organization of the reportis given:

- Abstract (300 words max)
- Introduction (1 page)
- The Problem (1 page)
- Related work (2 pages)
- research Idea or solution (1 page)
- The Details (5 to 8 pages)
- Conclusion and further work (1 page)
- references

2 Project 1: Design of a Verilog to Rust Translator

System-on-Chip (SoC) complexity growth has multiplied non-stop, and time-to-market pressure has driven demand for innovation in simulation performance. Logic simulation is the primary method to verify the correctness of such systems. Logic simulation is used heavily to verify the functional correctness of a design for a broad range of abstraction levels. In this research, we will explore a solution that uses high-level parallel abstractions and parallel computing to boost the performance of logic simulation. The first aspect of this research starts by the design of a Verilog to Rust Translator. The existing Verilog to C/C++ Translators are:

- Verilator (https://www.veripool.org/verilator/)
- Verilog2C++ (http://verilog2cpp.sourceforge.net/)
- hdlConvertor (https://github.com/Nic30/hdlConvertor)

This project focuses on the architecture of an efficient algorithm for the implementation of a Verilog to Rust translator with the measurable objectives to provide fully tested and documented Rust translator Libraries to be used in the design effort. The proposed solution should allow greater control over the source code for developers.

3 Project 2: Review of Parallel Algorithms for a Design Distributed Discrete-event Simulator

SoC complexity is increasing rapidly, driven by demands in the mobile market, and increasingly by the fast-growth of assisted- and autonomous-driving applications. SoC teams utilize many verification technologies to address their complexity and time-to-market challenges; however, logic simulation continues to be the foundation for all verification flows and continues to account for more than 90% [1] of all verification workloads. Logic simulation is the primary tool used to validate a widerange of design aspects, foremost among these being the correctness of the system's functionality, both in its behavioral (functional verification) description, as well as in its structural (gate-level verification) one. A large body of research has been devoted to accelerating the logic simulation process [2]. Besides improving the efficiency of algorithms, parallel computing has long been widely considered as the essential solution to provide scalable simulation productivity. Unfortunately, logic simulation has been one of the most difficult problems for parallelization due to the irregularity of problems and the hard constraints of maintaining causal relations [3]. Today commercial logic simulation tools depend on multicore CPUs and clusters by mainly exploiting the task-level par- allelism. In fact, large simulation farms could consist of hundreds of workstations, which would be expensive and power hungry. Meanwhile, the communication overhead might finally outweighthe performance improvement through integration of more machines. In recent years, accelerations of logic simulation have been proposed [4, 5]. The novel simulation architectures maximize the utilization of concurrent hardware resources while minimizing expensive communication overhead. The experimental results show that GPU-based simulators can handle the validationof industrial-size designs while delivering more than an order-of-magnitude performance improvements on average, over the fastest multithreaded simulators commercially available. However, the deployment of such GPU-based simulators is expensive in terms of hardware resources as well astheir integration in commercial simulators. The above approach did not find any breakthrough inIC verification and as such the use of multi-threaded commercial simulators is considered as bestpractice. In this research, we will explore a solution that uses high-level parallel abstractions andheterogeneous embedded parallel computing to boost the performance of logic simulation. The future fast simulation tool will be useful for medium-scale size companies that cannot afford to spend $50K on existing commercial simulators as well as universities to train hardware verification engineers.

The central focus of this research is to develop a fast logic simulator that uses parallelism in abstraction and computation. Most existing simulators leverage parallelism at the gate level and deploy the simulation using multi-threading. The approach to be investigated is based on RTL parallelism partitioning using machine learning techniques to build a fast and effective logic simulator. The fundamental question for this project is:

- How to deploy hardware accelerators within the different modules of the logic simulator such as the parallel Discrete-event simulator kernel?

The proposed architecture of the logic simulator to be developed is given below:

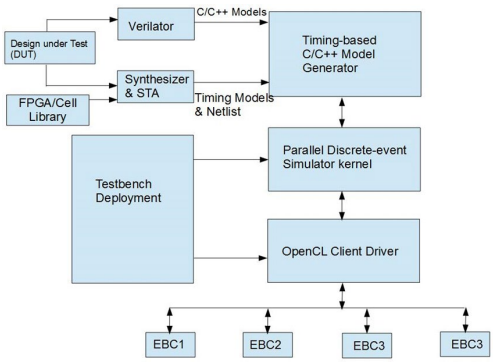

Figure 1: Architecture of the Logic Simulator (EBC: Embedded Computing Module, STA: Static Timing Analyzer)

The research investigation is to propose an effective solution to the research question, which at the end will allow us to develop a cost-effective fast logic simulator with its associated parallel computing platform. The student should present a review of efficient parallel algorithms for the design of a distributed discrete-event simulator. One important algorithm is the Chandy Misra and Bandy (CMB) Algorithm, for which in a parallel simulation process, different modules of a simulated system are abstracted as logic processes.

4 Project 3: Architecture of a Parallel Embedded Computing Hardware for Discrete-event Simulation

SoC complexity is increasing rapidly, driven by demands in the mobile market, and increasingly by the fast-growth of assisted- and autonomous-driving applications. SoC teams utilize many verification technologies to address their complexity and time-to-market challenges; however, logic simulation continues to be the foundation for all verification flows and continues to account for more than 90% [1] of all verification workloads. Logic simulation is the primary tool used to validate a widerange of design aspects, foremost among these being the correctness of the system's functionality, both in its behavioral (functional verification) description, as well as in its structural (gate-level verification) one. A large body of research has been devoted to accelerating the logic simulation process [2]. Besides improving the efficiency of algorithms, parallel computing has long been widely

considered as the essential solution to provide scalable simulation productivity. Unfortunately, logic simulation has been one of the most difficult problems for parallelization due to the irregularity of problems and the hard constraints of maintaining causal relations [3]. Today commercial logic simulation tools depend on multicore CPUs and clusters by mainly exploiting the task-level parallelism. In fact, large simulation farms could consist of hundreds of workstations, which would be expensive and power hungry. Meanwhile, the communication overhead might finally outweigh the performance improvement through integration of more machines. In recent years, accelerations of logic simulation have been proposed [4, 5]. The novel simulation architectures maximize the utilization of concurrent hardware resources while minimizing expensive communication overhead. The experimental results show that GPU-based simulators can handle the validationof industrial-size designs while delivering more than an order-of-magnitude performance improvements on average, over the fastest multithreaded simulators commercially available. However, the deployment of such GPU-based simulators is expensive in terms of hardware resources as well astheir integration in commercial simulators. The above approach did not find any breakthrough inIC verification and as such the use of multi-threaded commercial simulators is considered as bestpractice. In this research, we will explore a solution that uses high-level parallel abstractions andheterogeneous embedded parallel computing to boost the performance of logic simulation. The future fast simulation tool will be useful for medium-scale size companies that cannot afford to spend $50K on existing commercial simulators as well as universities to train hardware verification engineers.

The central focus of this research is to develop a fast logic simulator that uses parallelism in abstraction and computation. Most existing simulators leverage parallelism at the gate level and deploy the simulation using multi-threading. The approach to be investigated is based on RTL parallelism partitioning using machine learning techniques to build a fast and effective logic simulator. The fundamental question for this project is:

- How to build a cost effective and fast embedded computing cluster to be used to leverage parallel computation?

The proposed architecture of the logic simulator to be developed is given below:

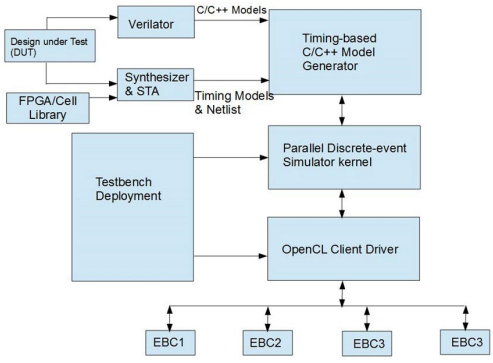

Figure 2: Architecture of the Logic Simulator (EBC: Embedded Computing Module, STA: Static Timing Analyzer)

The research investigation is to propose an effective solution to the research question, which at the end will allow us to develop a cost-effective fast logic simulator with its associated parallel computing platform. The student should research and propose the architecture of a cluster of embedded computing modules as well as the software architecture for parallel computing to be used in the deployment of the logic simulator.

5 Project 4: Comparative Study of C, Ada, and Rust for Embedded Systems Development

Embedded systems are at the heart of a wide area of applications, including avionics/aeronautics, space, transport, automotive, telecommunications, smart cards, consumer electronics. Embedded systems are composed of hardware and software components specifically designed for controlling a given application device. Embedded systems are of strategic importance for those sectors of the economy where the USA has the world industrial leadership. The project should review the characteristics of embedded systems and the associated requirements imposed on programming languages, followed by a comparative survey of the use of C, Ada, and Rust in embedded systems design withthe emphasis on methods for creating safe, reliable, and robust software for embedded systems.

Useful article: A survey of language-based Approaches to cyber-physical embedded systems development by Paul Soulier, Depeng Li, and John R. Williams, TSINGHUA SCIENCE AND TECHNOLOGY, ISSN 1007-0214ll 0211ll pp130-141 Volume 20, Number 2, April 2015.

6 Project 5: Embedded Software Verification

In practice, software testing has been the standard technique for identifying software bugs for a long time. In recent years, theoretical research and experience have shown that with software testing, it is impossible to guarantee high confidence. Formal verification techniques are now used byaerospace, railway, and nuclear applications. Major development groups of large software systems such as Facebook, Microsoft [13,14], and Linux [15] have also embraced formal methods. It is worth mentioning with the development of autonomous vehicles, there will be new requirements for higher software correctness and probably a software certification issue. Formal methods bring a higher reliability, robustness, and confidence in embedded software design. In this research project, we aim to train through research embedded software verification engineers. The first aspect of the project will consist in conducting a comparative study testing vs. formal verification in embedded software verification for C and Ada based programs, and the second aspect will consist in developing and reconfigurable formal embedded software verifier that supports the verification of C, Adaand Rust programs.

In this project students will conduct an extensive study of embedded software verification current practices, with a focus of projects developed in C, which culminates in comparative study between software testing and software checkers (formal verification) [16, 17, 18, 19]: To conduct such evaluations, it is important to propose a framework for test-based falsification (TBF)that interfaces between input programs, test generators, and test cases. Similarly for software

checkers a testing framework should be implemented, which include the following steps [20, 21, 22]: (a) defining the system under verification, (b) define the initial state from which the verification will be performed, (c) define the environment in which the system-under verification will run, (d) define property to be verified.

7 Project 6: Programming Languages for Embedded Software Design

Programming languages are important tools used by firmware developers in the embedded systems design life cycle, this is done by the transformation of requirements and designs into a code an embedded processor can execute. A language that enables a programmer to effectively and efficiently describes a concept and detect errors early in the development process will result in morereliable software [12]. C language is the popular language used in major embedded systems, but memory and type safety are some of its drawbacks. There exist other languages such as ADA thathave a set of unique technical features that make it highly effective for use in large, complex, and safety-critical projects. A newly programming language Rust delivers on speed and interoperability while making memory safety by default. Many issues pertaining to language-based techniques for improving embedded software qualities have relevant solutions, however there are still some open issues that need special attention. To date, languages used in embedded software development do not have mechanisms to specify timing requirements in code. Embedded multicore hardware is now common and developing error free software that exploits this potential parallelism is difficult.In this research, we will aim to develop an embedded software development tool using domain specific languages that integrates C, ADA, and Rust aiming to address the major issues in embedded software design.

Mbeddr[1] is a software tool that aims at creating a different way of developing embedded soft- ware systems. Instead of using archaic modeling tools and manually written C code, it uses the open source JetBrains MPS[2] language workbench to create an integrated approach to embedded development, where C programming, DSLs[3], domain specific extensions to C, product line variability, requirements traceability and model checking are supported directly. Considering the architecture of mbeddr, the research project will explore the possibility of integrating Ada and Rust, or the elaboration of a different architecture. Further work may continue with the development of a multi-language tool for the design of modern embedded software tools from consumer to safety-critical applications, thus supporting C, Ada, and Rust.

8 Project 7: Student's Project

Your project proposal should clearly define a research problem in any area of computer science that you want to address. Any proposal that is generic in nature will not be accepted, for instance a proposal like "Artificial Intelligence and its applications" does not define any research problem to be addressed.

Provide a description of your proposal, it should contain 250 words maximum.

[1]http://mbeddr.com/
[2]https://www.jetbrains.com/mps/
[3]Domain Specific Language

Below are the abstracts of two outstanding projects of Spring 2021:

Project 1: Audio Feature Selection Based on Linear and Non-Linear Segmentation

A proposal is made to use segmentation statistics to highlight the most important and characteristic audio features to use for an audio classification task. Experimentation is done on several problems ranging in difficulty and complexity in order to assess the viability of the proposed method. The Variable Markov Oracle segmentation algorithm is used to compress the data and reflect non-linear dynamics of a signal while rendering the metric Information Rate (IR). Kmeans clusteringis also performed and produces the Inertia (I) measure. These two established measures reflect the quality of segmentation from non-linear and linear perspectives and are used to select features ac- cording to the compressibility and segment coherence. A newly proposed metric combines IR and Ito reflect a combination of linear and non-linear dynamics. Several machine learning algorithms are tested with the proposed and established metrics informing under sampling procedures and feature selection. Promising results were observed that indicate all three measures can be used to select optimal features in some cases. Notably, a moderately successful implementation of COVID-19 cough audio classification was achieved with
this approach.

Project 2: On Modeling Slime Mold for Combinatorial Optimization

Recent works in combinatorial optimization have focused on an unsuspecting amoeboid Physarum polycephalum also commonly referred to as slime mold. The humble slime mold has proven its ingenuity in wet lab experiments by finding optimal paths through a maze, simulating the Tokyo rail system, and solving the Traveling Salesman Problem (TSP). With its remarkable ability to generate networks in a self-reinforced way with lack of environmental information while simultaneous constructing efficient networks for nutrient distribution, it has become the subject of inspiration for many networking algorithms. This paper presents evidence of slime mode relevance associated to an experimental methodology utilizing the Traveling Salesman Problem Library (TSPLIB). It also presents novel approaches based on new evidence for the future development of algorithms.

References

[1] Harry D. Foster, *Trends in functional verification: A 2014 industry study*, In Proceedings of Design Automation Conference, pages 07-11. IEEE, 2015.

[2] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*, John Wiley & Sons, Inc., 2000.

[3] Briner. J. V. Bailey, M. L. and Chamberlin, *Parallel Logic Simulation of VLSI Systems.*, ACM Comput. Surv, 26(3):255-294, 2013.

[4] Andrew D. Debapriya Chatterjee and Valeria Bertacco, *Gate-Level Simulation with GPU Computing*, ACM Transactions on Design Automation of Electronic Systems, 16(3:30):1-26, 2011.

[5] Wang B. Zhu Yuhao and Deng Yandong, *Massively Parallel Logic Simulation with GPUs,* ACM Trans. On Design Automation of Electronic Systems, 16(3): 1-20, 2011.

[6] D. K. Deeptimahanti and M. A. Babar, *An Automated Tool for Generating UML Models from Natural Language Requirements*, 2009 IEEE/ACM International Conference on Automated Software Engineering, Auckland, 2009, pp. 680-682, doi: 10.1109/ASE.2009.48.

[7] Manider Sing, *Using Natural Language Processing and Graph Mining to Explore Inter- Related Requirements in Software Artefacts*, ACM SIGSOFT Software Engineering Notes, Volume 44, Issue 1, March 2019, pp 37.

[8] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Erol-Valeriu Chioasca, Riza T. Batista-Navarro, *Natural Language Process- ing (NLP) for Requirements Engineering: A Systematic Mapping Study*, available at https://arxiv.org/abs/2004.01099.

[9] Andreas Vogelsanf, Kerstin Hartig, Florian Pudlitz, Aaron Schlutter, and Jonas Winker, *Supporting the Development of Cyber-Physical Systems with Natural Language Processing: A Report*, NLP4RE 2019: 2nd Workshop on Natural Language Processing for Requirements Engineering REFSQ, Essen, Germany.

[10] J. Zhao and I. G. Harris, *Automatic Assertion Generation from Natural Language Specifications Using Subtree Analysis*, 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 2019, pp. 598-601, doi: 10.23919/DATE.2019.8714857.

[11] Christopher B. Harris, *Generating Formal Verification Properties from Natural Language Hardware Specification*, Ph D Thesis, University of California, Irvine.

[12] B. Boehm and V. R. Basili, *Software Defect Reduction Top 10 List*, Computer, Vol. 34, no. 1, pp. 135 – 137, 2005.

[13] T. Ball and S. K. Rajamani, *The SLAM project: Debugging system software via static analysis.*, In Proc. POPL, pages 1-3. ACM, 2002.

[14] Z. Pavlinovic, A. Lal, and R. Sharma, *Inferring annotations for device drivers from verification histories*, In Proc. ASE, pages 450-460, ACM, 2016.

[15] A. V. Khoroshilov, V. Mutilin, A. K. Petrenko, and V. Zakharov, *Establishing Linux Driver Verification Process*, In. Proc. Ershov Memorial Conference, LNCS 5947, pages 165-176, Springer 2009.

[16] Dirk Beyer and Thomas Lemberger, *Software Verification: Testing vs. Model Checking – A comparative Evaluation of the State of the Art*, Beyer D., Lemberger T. (2017) Software Verification: Testing vs. Model Checking. In: Strichman O., Tzoref-Brill R. (eds) Hardware and Software: Verification and Testing, HVC 2017, Lecture Notes in Computer Science, vol 10629., Springer.

[17] M. Kim, Y. Kim and H. Kim, *A Comparative Study of Software Model Checkers as Unit Testing Tools: An Industrial Case Study*, IEEE Transactions on Software Engineering, vol. 37, no. 2, pp. 146-160, March-April 2011, doi: 10.1109/TSE.2010.68.

[18] Y. Kim and M. Kim, *SAT-based Bounded Software Model Checking for Embedded Software: A Case Study*, Asia-Pacific Software Engineering Conference (APSEC), Pages: 55 - 62, Dec 1-4, 2014.

[19] Beyer D. (2017), *Software Verification with Validation of Results.*, In: Legay A., Mar- garia T. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2017. Lecture Notes in Computer Science, vol 10206. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-54580-5 20.

[20] Beyer, D., Dangl, M. & Wendler, P. A, *Unifying View on SMT-Based Software Verification.*, J Autom Reasoning 60, 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6.

[21] D. Ratiu, V. Nimal, VeriSure, *Verification-Cases: Characterizing the Completeness Degree of Incomplete Verification for C Programs (Towards Using Formal Verification for Low Criticality Functions)*, Verification and Assurance Workshop 2015.

[22] L. Cordeiro, B. Fischer and J. Marques-Silva, *SMT-Based Bounded Model Checking for Embedded ANSI-C Software*, 2009 IEEE/ACM International Conference on Automated Software Engineering, Auckland, 2009, pp. 137-148, doi: 10.1109/ASE.2009.63.

[23] Garzella J.J., Baranowski M., He S., Rakamarić Z. (2020), *Leveraging Compiler Intermediate Representation for Multi- and Cross-Language Verification*, In: Beyer D., Zufferey D. (eds) Verification, Model Checking, and Abstract Interpretation. VMCAI 2020. Lecture Notes in Computer Science, vol 11990. Springer, Cham. https://doi.org/10.1007/978-3-030-39322-9_5.

[24] *https://www.qemu.org/*

[25] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Begt Werner, *Simics: A Full System Simulation Paftform*, Computers, 35(2): 50-58, 2002.

[26] Wind River SIMICS, *https://www.windriver.com/products/simics/*

[27] CMAP (2021), *Policy update on STEM occupations in the Chicago region*, https://www.cmap.illinois.gov/updates/all/-/asset_publisher/UIMfSLnFfMB6/content/stem-employment-trends-in-the-chicago-region

[28] Pike, Gary & Kuh, George. (2005), *First and Second-Generation College Students: A Comparison of Their Engagement and Intellectual Development*, The Journal of Higher Education. 76. 276-300. 10.1353/jhe.2005.0021.