Design and Implementation of a Compiler for Simulation of Large-scale Models

Tomás Pelayo, Unai Arronategui[0000−0003−4457−3938], José Ángel Bañares[0000−0002−4198−8241], and José Manuel Colom[0000−0001−5066−4030]

Aragón Institute of Engineering Research (I3A)
University of Zaragoza, Zaragoza, Spain
{779691,unai,banares,jm}@unizar.es

Abstract. As systems become increasingly sophisticated, their state spaces become larger and the use of analysis and simulation techniques is impossible by a single-processor machine. Distributed simulation seems the natural way to afford this problem. However, the main bottleneck of distributed simulation is the management, compilation and deployment of large scale models. This paper presents the experimentation results of a compiler of large scale Petri-net based models.

Keywords: Distributed simulation · Timed Petri nets · Large-scale compilation

1 Introduction

As systems become complex their modelling involves the management of large scale models, and their analysis and simulation becomes infeasible. Modular design and distributed simulation (DS) are the silver-bullets to manage large-scale modelling ans simulation. . However the complexity of distributed simulation maintains recurrent challenges. The pessimism has spread among the M&S community due to the difficulties to bring the field to a commercial success for performance prediction of complex, scalable systems [2].

The cloud has proven to be a suitable platform for distributed simulation of discrete event systems [5]. However, the incorporation of economic cost and energy consumption, in addition to the traditional speed-up metric have made DS even more complicated, especially if we are considering large-scale models.

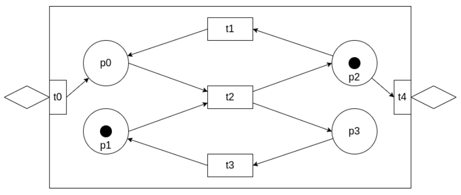
Petri nets (PN) are a well-known formalism for M&S of complex systems such as the cloud [4]. PN formal verification methods and simulation allow quantitative and qualitative analysis. Our previous works has defined a PN-based framework for the M&S of large scale models that provides a conceptual modelling language, a representation code for efficient distributed interpretation, mechanisms for load-balancing, and the overall methodology [1]. This paper focuses on the main bottleneck to develop large-scale models: A compiler and distributed linker to translate large scale models to executable models on a distributed execution platform.

The remainder of this paper is organised as follows: The first section briefly presents the incorporation of modular and hierarchical structuring primitives for defining large scale models. Section 3 briefly describes the design and implementation details of a compiler and distributed linker, and presents some preliminary experimental results. Finally, in section 4 we provide some final remarks.

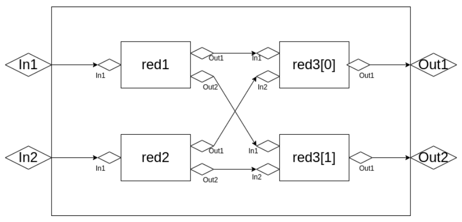
2 Modular Design of complex and large scale TPN models

Complex Systems are composed by a large number of entities whose behaviour and interactions (communications, competing for resources, etc.) can be modelled by Timed Petri nets (TPN). To build large scale distributed simulations it is necessary to decompose the conceptual model into components, each of which can be separately compiled to generate executable code for simulation. The composition of TPN-based specifications can be done by the incorporation of structuring mechanism (components) and mechanism for TPN composition [3] based on place/transition fusion, or arc connections. The interface of components includes input and output port specifications, and primitives for composing component behaviours by means of port connections.

A basic component can have `in`, `out` and `inout` ports. They specify sending and receiving events between components, and they are translated internally to places, transitions, and arcs. The mapping of `in` and `out` ports translates into a fusion of places, and `inout` port mappings represents a rendez-vous that can be translated into a fusion of transitions. Figure 1a represents a graphical representation of a basic component. A composite represents an aggregation of components, the port mapping between them, and what subcomponent ports configure the composite interface. Figure 1b shows the graphical representation of a composite.



(a) Basic component sample



(b) Composite component sample.

3 Model Elaboration, compilation, and distributed linked

Code execution for the simulation is based on the PN interpretation, which allow to separate the model specification from the simulator. It is essential for scaling simulations and simplifying the dynamic deployment of simulation code on distributed execution platforms. To provide an efficient interpretation, the specification must be translated to a model representation that allow the efficient interpretation of the model (fast update of the list of enabled transition).

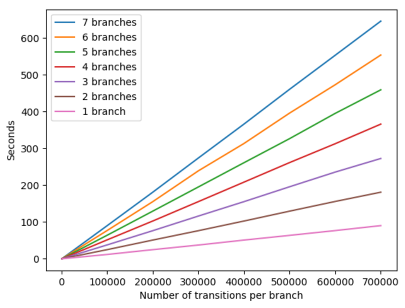
In previous works [1], we presented a TPN representation suitable for efficient distributed simulation based on linear enabling functions (LEFs). In a first step, the compiler translates the hierarchical description of components are flattened by a called *elaboration* process that produces a flat TPN representation. Then, The compiler translates the intermediate PN representation to an event dependency network, where each transition is translated to a structure that represents the transition enabling state, and how to update the enabling state of those transitions affected by the occurrence/firing of this transition.

Last step is the translation of code for a centralized simulation, that is, all the computational load of the simulation falls on a single machine, or simulation in a distributed simulator using several nodes. In the case of a distributed simulation, components specified in the top level of the hierarchy is used as the initial model partition to be deployed in the execution platform. To establish the communication channels, we have opted for the composition of components by fusion of transitions. In this way, the linking process recovers the lef structures of connected transitions, it generates the lef structure of the fusion transitions, and replaces fusion transitions by the transition that represents the *rendezvous* that is placed in any of the connected components.

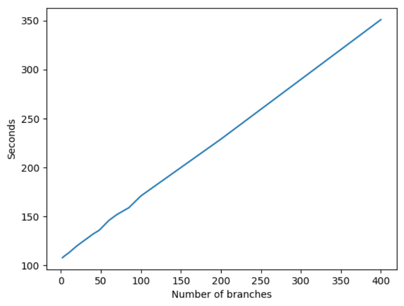
The use of array, iterators and distributed link has an important impact in the the required memory to generate the code. Vectors allow load the component only once in memory and use it as a reference for the different vector instances (modules). Replicated modules are connected by the distributed linker.

Our experimental evaluation is based on a synthetic PN that can be easily parametrised with the number of branches, which are chains of events that can be executed in parallel without violating the causality constraint, and the number of transitions represents the simulation workload of each branch. Branches are synchronized at the beginning and the end. This is a good sample to test compiler scalability increasing the number of branches, and the number of transitions by branch. The compiler and linker have been implemented in Ocaml. These experiments has been executed in a PC with a Intel Core i7-11700 2.50GHz processor, 128 GB of RAM and a Ubuntu 20.04 operating system.

Figure 2a shows compilation times of the branches model with different number of branches and transitions by branch. Distributed linking opens the possibility to generate code beyond the memory limits of a computer. In order to test this new capacity of the compiler, the tests has been carried out with the branch model. The model has been be divided into branches with synchronization points. It is intended to place each branch in a different node. Each branch has been generated with 1,000,000 transitions occupying the lef data representation 210 MB in memory in each simulation node. Different number of branches have been compiled and linked. The execution times obtained are shown in Figure 2b.



(a) Compilation time with branches.



(b) Compilation & link time in function of #branches.

4 Conclusions and Future work

This paper has shown a first approach to generate concise specifications of large scale PN models, and the first experimental results of an scalable compiler. Future work includes to improve the language expressiveness, and to develop an information system to manage large scale models and help the dynamic configuration of distributed simulations.

Acknowledgments This work was co-financed by the Aragonese Government and the European Regional Development Fund "Construyendo Europa desde Aragón" (COSMOS research group); and by the Spanish program "Programa estatal del Generación de Conocimiento y Fortalecimiento Científico y Tecnológico del Sistema de I+D+i ", project PGC2018-099815-B-100.

References

1. Bañares, J.Á., Colom, J.M.: Model and simulation engines for distributed simulation of discrete event systems. In: GECON 2018 - International Conference on the Economics of Grids, Clouds, Systems, and Services. pp. 77–91. Springer (2018)
2. Ferscha, A., Johnson, J., Turner, S.J.: Distributed simulation performance data mining. Future Generation Computer Systems 18(1), 157–174 (2001), i. High Performance Numerical Methods and Applications. II. Performance Data Mining: Automated Diagnosis, Adaption, and Optimization
3. Gomes, L., Barros, J.P.: Structuring and composability issues in petri nets modeling. IEEE Trans. Ind. Informatics 1(2), 112–123 (2005)
4. Louhichi, W., M.Berrima, Robbana, N.B.R.: A timed and colored petri nets for modeling and verifying cloud system elasticity. International Journal of Electrical and Computer Engineering 16(3), 24 – 33 (2022)
5. Vanmechelen, K., De Munck, S., Broeckhove, J.: Conservative distributed discrete-event simulation on the amazon ec2 cloud: An evaluation of time synchronization protocol performance and cost efficiency. Simulation Modelling Practice and Theory 34, 126–143 (2013)