An Automated Parallel Simulation Flow for Heterogeneous Embedded Systems

Seyed Hosein Attarzadeh Niaki, Ingo Sander
Unit of Electronic Systems
School of Information and Communication Technology
KTH Royal Institute of Technology, Stockholm, Sweden
Email: {shan2, ingo}@kth.se

***Abstract*—Simulation of complex embedded and cyber-physical systems requires exploitation of the computation power of available parallel architectures. Current simulation environments either do not address this parallelism or use separate models for parallel simulation and for analysis and synthesis, which might lead to model mismatches. We extend a formal modeling framework targeting heterogeneous systems with elements that enable parallel simulations. An automated flow is then proposed that starting from a serial executable specification generates an efficient MPI-based parallel simulation model by using a constraint-based method. The proposed flow generates parallel models with acceptable speedups for a representative example.**

I. Introduction

One of the key steps in the design of embedded and Cyber-Physical System (CPS) is development of executable system models which can be used to validate systems' functionality via simulation. Depending on the level of details captured in a model, its simulation may run much slower than the real system. On the other hand, continuous increase in complexity of such systems makes the simulation runs even longer, making exploitation of the computation power offered by parallel, multi- and many-core platforms a necessity.

Parallel and distributed simulation techniques have been centers of interest for a long time especially for discrete-event models in areas such as High Performance Computing (HPC). Usually, a parallel simulation model of a system is developed using libraries (e.g., MPI), compiler directives (e.g., OpenMP), or a higher level parallel simulation environment (e.g., Parsec [1]) and the compiled model is run on a parallel machine.

However, models of embedded systems are used not only for simulation, but also for analysis and synthesis. Developing separate models for these objectives is cumbersome and raises the chance of incompatibilities between them. It is beneficial to have a single model of a system which can be used both for efficient parallel simulation and application of formal analysis and synthesis techniques. To the best of our knowledge, this problem is not addressed in the available modeling frameworks, simulation environments, and published literature.

We propose a method for automatic construction of efficient parallel and distributed simulation models from formal high-level executable models. The SystemC-based implementation of the **For**mal **Sy**stem **De**sign (ForSyDe) modeling framework [2] is used to capture an initial executable specification of a system which can be validated by serial simulation using the SystemC simulation kernel. The developed models are able to export their own structure as an intermediate format which is used for analysis and synthesis. We use this format and a constraint programming-based approach to partition a specification between different cores/machines statically, followed by generation of the simulator code for individual computing elements.

978-3-9815370-0-0/DATE13/

The contributions of this work are

- extending a formal system-level modeling framework with new elements that enable parallel simulations;
- formulating a Constraint Satisfaction Problem (CSP) to partition a process network for efficient parallel simulation;
- introducing an automated flow for generation of parallel simulation models from executable specifications; and
- applying the proposed flow in a case study.

II. The Modeling Framework

This section briefly describes the ForSyDe [3], [4] modeling framework which is selected as the formal basis and design entry point of our flow.

A system model is structured as a concurrent hierarchical process network. Such a network consists of *processes*, connected by *signals*. Processes communicate and synchronize *only* using signals. As a result, there is no global state in the system which leads to more inherent parallelism in the specification. Processes are either *composite processes*, which are created by composing other processes, or *leaf processes*, which are directly created using *process constructors*.An example of a system specification is given in Fig.1 in which p_{123} is a composite process formed by composition of leaf processes p_1 to p_3.

Models of Computation (MoCs) describe the semantics of computation and concurrency of the processes in the system. We use MoCs to model the timing abstraction in each component of a heterogeneous system model explicitly. Each leaf process in the process network belongs to a specific MoC. For example, in Fig.1, processes p_1 to p_3 could belong to the Continuous Time (CT) MoC, which is suitable for modeling analog systems, or the Synchronous Data Flow (SDF) MoC, which fits best for modeling analyzable streaming applications. ForSyDe also supports modeling of heterogeneous systems by means of Domain Interfaces (DIs), which are processes that relate signals belonging to different MoCs together.

In order to create a leaf process in the model, the designer must use a process constructor. A process constructor is a

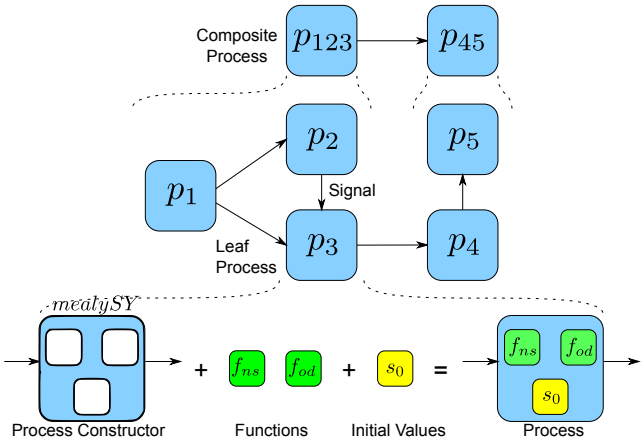

Fig. 1. Example of a system model in ForSyDe.

formally defined construct which the designer chooses from the ForSyDe library, provides it with side-effect-free functions and/or initial values, and obtains a valid process. As an example, in Fig.1, p_3 is created using the `mealySY` process constructor. The designer supplies the next-state function f_{ns}, the output-decoding function f_{od}, and an initial state s_0 to `mealySY` and gains a process with the semantics of a Mealy state machine in the Synchronous (SY) MoC. This key difference compared to other approaches enables construction of formal and analyzable models. The concept of wrappers [5] can also be used to wrap external models e.g., a TLM IP or an instruction-set-simulator; enabling simulation of lower-level platform components.

ForSyDe is also implemented as a C++-based class library on top of the IEEE standard SystemC Domain Specific Language (DSL). In this case, an inermediate execution model (more precisely, simulation model) based on a Kahn Process Network (KPN) with blocking write to bounded FIFOs is used to map heterogeneous models of computation on top of the SystemC kernel [2]. Signals are mapped to SystemC FIFOs and processes are realized by SystemC modules that include a single SystemC process which invoke user-provided functions.

In order to perform further analysis and synthesis on a model specified in ForSyDe, there is a need for a tool which exports structural and behavioral aspects of the system to appropriate formats. Following [6], we call the ability of an executable specification to query its own specification using a reflective mechanism, *introspection*. In ForSyDe-SystemC, processes collect the structural information, such as interconnection of the components and channel types, automatically before starting the simulation. On the other hand, in ForSyDe, the behavioral aspects of a model are mainly isolated as functions passed to process constructors, which makes them easy to access automatically by means of simple user annotations and naming conventions. In this way, a ForSyDe-SystemC model can inspect its own behavior and structure, and export it as suitable formats to be read by external tools and flows. This method is presented in [2] and is used also in this work.

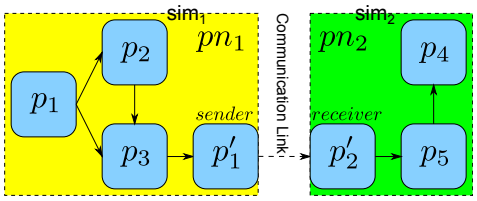

Fig. 2. A partitioned process network for parallel simulation.

III. Extending the Framework for Parallel Simulation

Process networks, especially the way they are defined in ForSyDe, are inherently parallel due to the fact that communication and synchronization are performed *only* using signals. Theoretically, every single process can be simulated on a separate processing element provided that it can communicate with other processes via FIFOs with blocking semantics. However, this can easily become inefficient because of the communication overhead or unfeasible due to the limitation of available processing elements. We would like to partition the complete process network of the system model into sub-networks and run sub-simulations for each sub-network. Special sender/receiver processes need to be defined in the framework which communicate with the corresponding receiver/sender processes on the cutting points of the process network. In this way we can control the granularity of the simulation parallelism explicitly. Fig.2 depicts the idea for the process network of Fig.1 which is partitioned into two sub-networks pn_1 and pn_2, simulated using simulator instances sim_1 and sim_2.

We extend the modeling framework by defining two new process constructors. The $sender$ constructs processes appears as a sink in a sub-network of a parallel simulation model but packs its input events as transmission packets. Similarly, the $receiver$-based processes which look like a source process and unpack the received packets as a signal of events in ForSyDe. Despite the syntactical resemblance, both of the processes are semantically equivalent to a combinational process ($comb$) with identity function (see Fig.2). In addition, we require that an implementation of these constructors in a parallel simulation environment ensures that the semantics of the communication (and synchronization) between a sender and receiver process using a communication link adheres to the semantics of communication (and synchronization) between two ordinary processes using signals. This method of synchronization among sub-simulations is commonly called “conservative synchronization” in the parallel simulation terminology [7].

We chose to implement the $sender$ and $receiver$ process constructors in the SY MoC of ForSyDe-SystemC using the Message Passing Interface (MPI) library as the communication and synchronization mechanism between the sub-simulations. Given a rank r and a tag t, a $sender$/$receiver$ communicates its output/input and synchronizes with a $receiver$/$sender$ in a sub-simulation with MPI rank r using the message tag t. The message tag is unique for each communication link and is used to distinguish between other send and receives in a sub-simulation. A $sender$/$receiver$ uses the

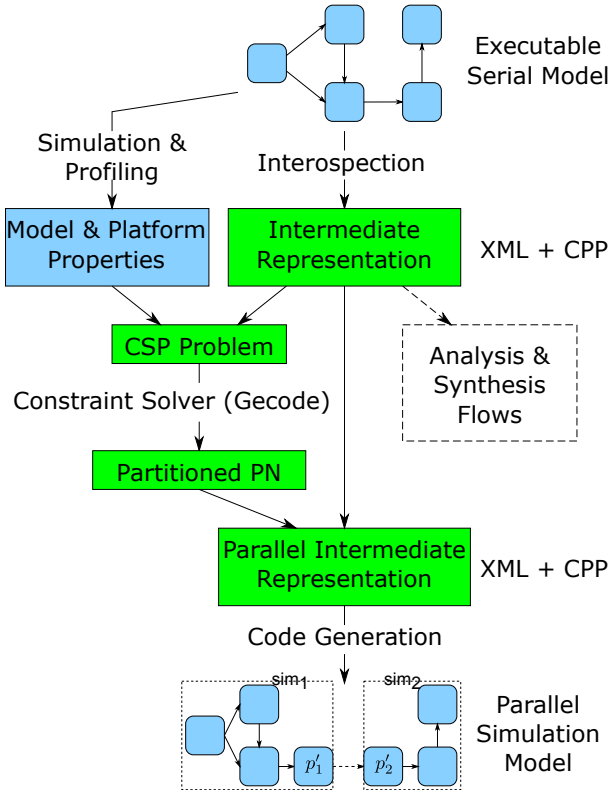

Fig. 3. A parallel simulation flow for embedded systems starting from formal executable models. The partitioning problem is formulated as a Constraint Satisfaction Problem (CSP).

`MPI_Send`/`MPI_Recv` command with blocking semantics in the implementation of `prod`/`prep` phase of its abstract semantics internally.

IV. The Parallel Simulation Flow

The proposed flow for parallel simulation is illustrated in Fig.3. In this section we present a general overview of the flow followed by a detailed description of the challenging steps.

The designer starts the modeling using ForSyDe-SystemC library and constructs an executable model which runs over the serial SystemC simulation kernel. Using the introspection feature of the library, an intermediate representation is generated which can also be used for analysis and synthesis. Given a set of simulation environment characteristics such as the communication cost per unit of data and the number of available processing cores, and the intermediate representation of the system, a Constraint Satisfaction Problem (CSP) is set up to identify a partitioning of the process network which leads to an optimal parallel simulation performance. In this work, we use the Gecode [8] constraint solver to find a solution for the CSP problem (See Sec.V for more details). Having a solution for the process network partitioning problem, an intermediate representation of the parallel simulation model is generated. This is achieved by setting a separate simulation model for each partition of the process network and inserting the newly introduced *sender* and *receiver* processes on each of the cut signals. This step is discussed in more detailed later in this section. Finally, the ForSyDe-SystemC code for each of the sub-simulations is generated which can be compiled and executed in parallel on the parallel platform.

During insertion of the *sender* and *receiver* processes, the communication parameters passed to their process constructor, namely the rank of the source/target and the message tag, should be properly set. More formally, assume p_i is originally connected to p_j via a signal s_{ij}. For a partitioning that assigns p_i to sim_k and p_j to sim_l, p_i will be connected to a sender process that communicates to $rank = l$ with a unique tag t_{ij}. Similarly, p_j will be connected to a sender process that communicates to $rank = k$ with the same tag t_{ij}.

V. Partitioning the Process Network

The missing piece of the simulation flow presented in Fig.3 is the formulation of a CSP problem for partitioning a system process network into sub-networks for parallel simulation. First, we introduce what are the inputs and outputs of the problem and then the problem variables are defined. We specify a set of constrains which separate valid partitioning and a cost function which is used for evaluating the solutions against each other. Finally, we state which searching and branching strategies are used for the constraint solver.

a) Inputs and Outputs: The inputs of the problem consist of *1*) the maximum number of available parallel processors M; *2*) Number of processes N; *3*) e_i (with $1 \leq i \leq N$) denoting the average execution time of each process p_i; and *4*) $c_{i,j}$ the average communication time from p_i to p_j, assuming p_i and p_j belong to different sub-simulations. If p_i is not connected to p_j in the original specification, then $c_{i,j} = 0$.

b) Constraint Variables: 1) n is an integer variable denoting the number of network partitions (sub-simulations) with a domain of $[1, M]$; *2*) m_is (with $1 \leq i \leq N$) are a set of integers with the domain $[1, M]$ where m_i states to which sub-network the process p_i is assigned; and *3*) *cost* which is the associated cost to be minimized for each partitioning and is used for optimization.

c) Constraints: The first constraint limits the m_is to the number of available partitions:

$$m_i \leq n; 1 \leq i \leq n \tag{1}$$

In order to break the symmetry and reduce the size in the design space, we fix the first process to the first sub-network:

$$m_1 == 1 \tag{2}$$

The cost function for which the optimization is performed is an estimation of the total execution time after partitioning. Here, we assume that the communication cost of the processes in the same partition is negligible compared to the execution time of the processes and the communication overhead between the partitions.

$$\begin{aligned} cost == avg_e + \frac{\sum_{i=1}^{n} |\sum_{j=1}^{N} e_j|_{m_j=i} - avg_e|}{n} \\ + \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} c_{i,j}|_{m_i \neq m_j}}{|\{c_{i,j} \neq 0 | m_i \neq m_j, i, j \in [1, N]\}|} \end{aligned} \tag{3}$$

where $avg_e = \frac{\sum_{i=1}^{N} e_i}{n}$ is the average execution time of sub-simulations. The equation sums the average computation time

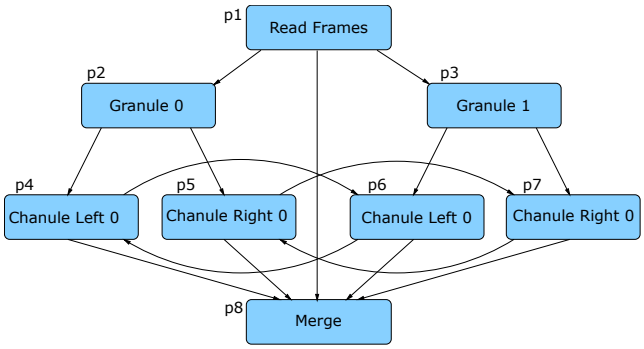

Fig. 4. The process network of the MP3 decoder system.

of each sub-network plus a penalty for unbalanced distribution of processes and the average communication overhead for communication between the sub-networks.

Implementation decisions might impose additional constraints. For example, the current implementation of ForSyDe-SystemC represents signals in the CT MoC as functions over time [2]. C++ does not support serialization of function objects for communication and thus, we need to restrict the mapping of CT process clusters to the same sub-simulation.

We chose to branch first on the number of partitions n starting from the minimum value, and then on m_is with smallest domain and minimum value. A branch and bound algorithm is used for searching the design space.

VI. Case Study

The presented design flow is tested on an MP3 decoder system model. The system's process network is shown in Fig.4. It consists of 8 key functional processes which implement the decoding algorithm. First, an executable ForSyDe-SystemC model of the system is run under the profiler to extract the average execution time of each actor. We used the Callgrind profiler [9] which can report the inclusive execution time of each function. Looking at the results we see that around 93% of the simulation time is spent in the computational functions, which supports our decision in neglecting the communication delays of the processes in the same sub-simulation. The parallel simulation platform in our case is a Linux workstation powered by an Intel Quad core CPU running the OpenMPI environment. The average communication overhead of an MPI send-receive pair per byte is measured separately using a simple program which was $7 \times 10^{-9sec/Byte}$ in our case. This is used together with the profiling results to estimate the inter-process communication delay between sub-simulations. These results were printed as the inputs to a Gecode script which we use to solve our CSP. Running the Gecode script for this problem was instantaneous and resulted for optimal partitonings for 2- to 4-processor parallel simulations. The results are presented in Table I as the constraint variables m_i. For each case, the corresponding sub-simulation systems were generated, compiled and executed under the MPI runtime as an MIMD program. A speed-up of 1.7x was observed for the 2-core solution but there was no performance gain in the simulation time for the 4-core case.

TABLE I
Optimal partitionings for the MP3 decoder example.

# of sub-sims	m_is
2	$\{1, 2, 2, 1, 1, 2, 2, 1\}$
3	$\{1, 2, 2, 1, 2, 3, 3, 1\}$
4	$\{1, 2, 3, 1, 2, 3, 4, 4\}$

VII. Related Work

Some of the related works such as Parsec [1] provide a dedicated input languages and execution environments. [7] gives a good overview on these works. These approaches do not provide sufficient means for modeling heterogeneous systems and their input models are not suitable application of analysis and synthesis methods. Modification of the SystemC kernel for distributed simulation based on the Parallel Discrete-Event Simulation (PDES) model has also been performed. Approaches like [10] modify the simulation kernel, take a conservative approach for synchronization, and add new SystemC construct to the language to explicitly control the level of parallelism. Also in [11], the kernel is modified but the distribution is done automatically and is hidden from the user. Unlike the aforementioned approaches, we do not modify the SystemC kernel and use an offline method for distributing the processes among the computing nodes.

VIII. Conclusion

We have presented a parallel and distributed simulation flow to be integrated in the design flow of heterogeneous embedded systems. It partitions a verified serial model by solving a Constraint Satisfaction Problem (CSP) and generating simulation code for the partitioned system. The advantages of the flow are that it starts from the same model that is used also from analysis and synthesis; it is ready for automation and no user intervention is required; it uses the inherent parallelism in the original specification and hence, does not require any parallel programming expertise.

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y.-A. Chen, X. Zeng, J. Martin, and H. Y. Song, "Parsec: a parallel simulation environment for complex systems," *Computer*, 1998.
[2] S. Attarzadeh Niaki, M. Jakobsen, T. Sulonen, and I. Sander, "Formal heterogeneous system modeling with systemc," in *Proc. of FDL*, 2012.
[3] I. Sander and A. Jantsch, "System modeling and transformational design refinement in ForSyDe," *IEEE TCAD*, 2004.
[4] A. Jantsch, *Modeling Embedded Systems and SoCs*. Morgan Kaufmann, 2004.
[5] S. H. Attarzadeh Niaki and I. Sander, "Co-simulation of embedded systems in a heterogeneous moc-based modeling framework," in *International Symposium on Industrial Embedded Systems (SIES)*, 2011.
[6] H. D. Patel and S. K. Shukla, *Ingredients for Successful System Level Design Methodology*, 1st ed. Springer, Jun. 2008.
[7] A. Sulistio, C. S. Yeo, and R. Buyya, "A taxonomy of computer-based simulations and its mapping to parallel and distributed systems simulation tools," *Software: Practice and Experience*, 2004.
[8] C. Schulte, G. Tack, and M. Lagerkvist, "Modeling and Programming with Gecode," 2012.
[9] J. Weidendorfer, "Sequential performance analysis with callgrind and kcachegrind," *Tools for High Performance Computing*, 2008.
[10] B. Chopard, P. Combes, and J. Zory, "A conservative approach to SystemC parallelization," in *Computational Science ICCS 2006*. Springer Berlin / Heidelberg, 2006.
[11] D. R. Cox, "RITSim: distributed systemC simulation," *master's thesis*, 2005.