Towards A Heterogeneous Simulation Kernel for System Level Models: A SystemC Kernel for Synchronous Data Flow Models*

Hiren D. Patel Sandeep K. Shukla
FERMAT Laboratory
Virginia Polytechnic Institute and State University
{hiren,shukla}@vt.edu

ABSTRACT

As SystemC gains popularity as a modeling language of choice for system-on-chip (SOC) designs, heterogeneous modeling in SystemC and efficient simulation become increasingly important. However, in the current reference implementation, all SystemC models are simulated through a non-deterministic Discrete-Event simulation kernel, which schedules events at run-time. This sometimes results in too many delta cycles hindering the simulation performance of the model. The SystemC language also seems to target this simulation kernel as the target simulation engine. This makes it difficult to express different Models Of Computation naturally in SystemC. In an SOC model, different components may need to be naturally expressible in different Models Of Computations. Some of these components may be amenable to static scheduling based simulation or other pre-simulation optimization techniques. Our goal is to create a simulation framework for heterogeneous SystemC models, to gain efficiency and ease of use within the framework of SystemC reference implementation. In this paper, we focus on Synchronous Data Flow (SDF) models, where the rates of data produced and consumed by a data flow node/block are known a priori. In digital signal processing (DSP) applications where relative sample rates are specified for each DSP component, such models are quite common. Compile time knowledge of these rates allow the use of static scheduling resulting in significant improvement in simulation efficiency. We describe an alternate SystemC kernel that exploits such static scheduling of SDF models. Our experiments show improvement in simulation time over the original models and over the latest efficiency results from [20].

*We acknowledge the support of NSF CAREER grant CCR-0237947, NSF NGS program grant ACI-0204028, and an SRC Integrated Systems grant.

GLSVLSI'04, April 26–28, 2004, Boston, Massachusetts, USA.

Categories and Subject Descriptors

I.6.2 [**Computing Methodologies**]: Simulation and Modeling Simulation Languages

General Terms

Algorithms, Design, Experimentation, Verification

Keywords

Models Of Computation, SystemC, Heterogeneous, Synchronous Data Flow, Embedded System Design, Simulation Efficiency

1. INTRODUCTION

System level modeling for System-on-Chip and hardware & software co-design has been gaining importance due to rising complexity of such systems, continual advances in semiconductor technology, and the productivity gap in design of complex systems. In the past few years, we have seen the introduction of many system level design frameworks and languages such as SystemC and SpecC [14, 21]. There also is an effort to lift the abstraction level in the hardware description languages, exemplified in the efforts to standardize SystemVerilog [22]. The success of any of these languages is predicated upon the adoption as a standard design entry language by the industry and resulting closure of the productivity and verification gaps. SystemC is poised as one of the strong contenders for such a language. SystemC is an open-source scheme that allows designers to simulate their system level models in a very VHDL [24] and Verilog [23] like manner. In this paper, we argue that going to a higher level abstraction for closure of the productivity gap, while keeping the Discrete-Event simulation semantics, are contradictory goals. In fact, most system models for SOCs are heterogeneous in nature, and encompass multiple Models Of Computation in its different components. As a result, we have to start thinking about simulation semantics of such a langauge away from the semantics of hardware simulation only, and we have to be able to provide a way to express and simulate other models of computation. The inspiration of such a system is drawn upon the success of the Ptolemy II framework in specification and simulation of embedded heterogeneous software systems [6]. However, since SystemC is targeted to be the language of choice for semiconductor manufacturers as well as system designers, including embedded software, our goals are more ambitious. In this paper we report a first step towards this goal by introducing

heterogeneity in SystemC by adding a Synchronous Data Flow (SDF) kernel along with the distributed Discrete-Event (DE) kernel. We also show that the previous attempts in SDF modeling with SystemC [5, 13] are not adequate for simulation speed and expressibility.

One of the first attempts at designer guidelines for heterogeneous modeling in SystemC is shown in [5]. Here, they model SDF systems using *SC_THREAD* processes and blocking read and write *sc_fifo* channels. This scheme maps the SDF model onto the Discrete-Event kernel using dynamic scheduling at run-time opposed to the conventional static scheduling. The kernel has to accommodate the SDF model with the underlying DE kernel incurring unnecessary delta cycles. [5, 13] present a method whereby SDF systems can be modeled in SystemC. However, we argue that this is an inefficient method of implementing SDF in SystemC nor is it natural. Not only do the adder [5] and OFDM [13] examples use blocking *sc_fifo* type channels but they also use *SC_THREAD*s introducing context-switch overhead. We understand that these examples in [5, 13] do not exploit the SDF MOC and their approach defeats the purpose of modeling SDF systems because once recognized as SDF, they can and should be statically scheduled. [5] promotes that tools may be designed to exploit static scheduling algorithms to perform efficient SDF simulation, but they only speculate some source level transformation techniques in enabling SDF systems with the underlying DE reference implementation kernel. These models can be converted such that there is no need for *SC_THREAD* or *SC_CTHREAD* processes, communication signals to pass control, and most importantly dynamic scheduling. In this paper, we present an SDF kernel interoperable with the existing DE kernel, that eliminates the need for dynamic scheduling and complex programming idiosyncrasies specific for SDF models. We propose source level hints to be provided by the model designer to help express SDF more naturally and to make the new simulation kernel execute special functionalities.

Synchronous Data Flow MOC is not a new idea and extensive work has been done in that area. [1] shows work on scheduling SDFs, buffer size minimization and other code optimization techniques. We employ technology presented in that work, such as their scheduling algorithm for SDFs. However, the focus of [1] pertains to synthesis of embedded C code from SDF specifications. This allows their SDF representations to be abstract, expressing only the nodes, their connections, and token values. This is unlike the interaction we have to consider when investigating the simulation kernel for SystemC. We need to manipulate the kernel upon computing the schedules requiring us to discuss the Data Flow theory and scheduling mechanisms. Our aim is not in solely providing evidence to show that changes made in the kernel to accommodate different MOCs improves simulation efficiency because this is already known. Instead, we aim at introducing a heterogeneous modeling and simulation framework in SystemC that also results in simulation efficiency. From the results, it may be inferred that the gain in efficiency is not significant, but the models presented are mainly single-SDF models. Constructing models with multiple SDF blocks will show a significant increase in simulation speed as per Amdahl's law [12].

1.1 Why Heterogeneous Kernel in SystemC?

The language infrastructure and object-oriented nature of C++ used to define SystemC extends usability farther than any existing hardware description language at present. SystemC facilitates embedded systems designers with a freely available simulation and modeling framework. Generally, simulation is the foremost stage in the development cycle of a system, where designers can check for functional correctness, sufficient and appropriate communication interactions, and overall correct conceptualization of the system. Consequently, due to the amount of time the product spends in the design phase, simulation efficiency is a gating factor to the time-to-market. SystemC has gained sufficient industrial momentum for its usage for design validation. This provides an added motivation to investigate SystemC.

Although meant to be a system level design language, the current SystemC simulation kernel uses a non-deterministic Discrete-Event scheduler to simulate the modeled system. This is very well suited for designs at the register transfer level of abstraction, similar to VHDL or Verilog models. However, since SystemC is also planned as a language at higher levels of abstraction, notably at the system level, such DE based simulation is not the most efficient for system level simulation. This is because heterogeneous Models Of Computation [9] are better suited for different parts of a system design. For example, an SOC design might consist of DSP cores, microprocessor models, bus models etc. Each of these may have their appropriate MOC, and hence ability to simulate in a heterogeneous kernel environment will speed up the simulation, as well as allow designers to put together a heterogeneous model without worrying about the target simulation kernel.

In order to build such an infrastructure, we have undertaken the task of building a SystemC simulation kernel that on the one hand is interoperable with the existing simulation kernel of the reference implementation, but on the other hand employs different simulation kernel functions depending on which part of the design is being simulated. In this paper, we focus on building the simulation kernel appropriate for the Synchronous Data Flow (SDF) [9, 11, 6] models. We also propose a manner in which the designers code their models that encapsulate an SDF MOC at the source code level, so that the simulation kernel identifies when to employ this particular kernel during simulation. The long term effort is in providing the design automation community with an open source multi-domain modeling framework. The current SystemC standards are being challenged by implementing features to incorporate SDF designs because of the flexibility the modeler gains when using our kernel. However, we maintain all existing SystemC standards and plan to submit our changes to the SystemC standards committee. We also plan on distributing our heterogeneous SystemC kernel to the SystemC community.

1.2 An Example of Heterogeneous Model

We construct an example of a heterogeneous model of an image Converter shown in Figure 1. This system begins by downloading encrypted images that are decrypted and converted to a specific type and then encrypted again to be uploaded back to the source. This is a multi-MOC model where the first DE block is responsible for downloading encrypted images and decrypting them. These images are passed onto the SDF block that performs the Sobel edge detection algorithm on the image. This is our preferred conversion type. Output from Sobel is sent to the final DE block that encrypts the converted image and uploads it back to the download source. The Sobel SDF component is separated into three functional blocks as shown in Figure 1. The DE component pushes the image values into the data path queue to the CleanEdges block. This clears the edges of the image matrix and forwards the values through the Channel to the Sobel operator block. This Sobel block per-

forms the Sobel computations and pushes the results to the following DE component.

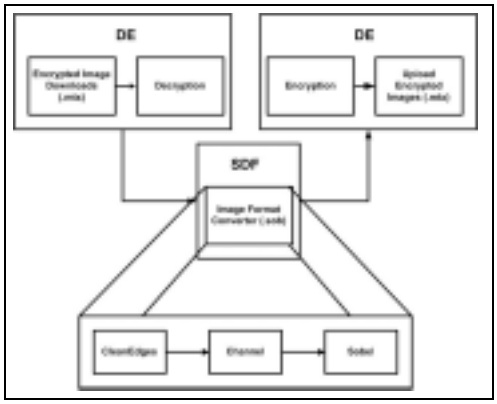


Figure 1: Image Converter Diagram

The interesting aspect about this model is the interaction between the DE and SDF blocks. By interaction we mean the data transfer. SystemC channels may be employed as the interaction medium between the DE and SDF blocks, but we advise using *queue<...>* type queues to push the data to and receive from the SDF top-level (this is a process that encapsulates all the SDF blocks). Both the block responsible for pushing data into the SDF and the SDF must have access to the queue and a control signal can be used to trigger the top-level process once data is ready on the queue for the SDF to consume. However, the modeler must ensure that there is sufficient data ready on the queue before setting the control signal to execute the SDF top-level block. Otherwise, the modeler will experience segmentation faults unless explicitly handled in the model. This is a simple example showing how we implement the Converter, a heterogeneous model using our improved modeling and simulation framework.

1.3 Organization

This paper is organized as follows: In Section 2 we discuss some related work and in Section 3 we present relevant preliminary background. Section 4 discusses the Discrete-Event and Synchronous Data Flow kernels, in Section 5 we describe some experimental results to show the efficiency of simulation obtained, and Section 6 concludes the paper with discussion and future work.

2. RELATED WORK

Research related to simulation efficiency in SystemC is a relatively new area of research and related work in re-threading based simulation speed up have been conducted [20, 19, 18, 17]. To the best of our knowledge no attempt has been made in implementing other Models Of Computation at kernel level for SystemC. Though the idea of heterogeneous modeling is not explicit to the SystemC language, the concept of heterogeneous modeling is popular as demonstrated by the success of Ptolemy II [6]. Ptolemy II allows designers to select directors that in relation to SystemC act as the kernel.

The Synchronous Data Flow MOC is a subset of the Data Flow paradigm [3, 11, 1]. This paradigm dictates that a program is divided into blocks and arcs, representing functionality and data paths, respectively. The program is represented as a directed graph connecting the function blocks with the data arcs. A Synchronous Data Flow model imposes further constraints by defining the block to be invoked only when there is sufficient input samples available to carry out the computation by the function block, and blocks with no data input arcs can be invoked at any time.

Lee et al. in [10, 1] describe a method whereby static scheduling of these function blocks can be computed during compile time rather than run-time, which with appropriate modifications can be used in the implementation of an SDF kernel for SystemC. The method utilizes the predefined consumption and production rates to construct a set of linear homogeneous system of equations and reducing the solution of this system of equations to an integer-valued solution producing the *repetition vector* [1] for the SDF model.

Static scheduling requires a solver to yield the *repetition vector*. The *repetition vector* must consist of non-zero integer values. Our approach identifies these system of 2-variable equations as linear Diophantine [2, 4, 7, 8] homogeneous system of equations since they must have a non-zero and an integer valued solution. As it is known, not all homogeneous systems have a non-zero integer solution. Combining the fact that all equations germane in SDF are 2-variable equations with certain properties that make an iterative solution possible, one can overcome the complexity of solving linear Diophantine equation systems in general. An algorithm using Hilbert's basis computation provides a solution for first order Diophantine equations that suffice our needs of solving the static scheduling problem. [15] provides a Hilbert's solver programmed in C that we employed for computing the SDF *repetition vector*.

3. PRELIMINARY BACKGROUND

There are two problems to be solved in creating an *executable schedule* for an SDF graph (SDFG) for execution. Since, the execution of an SDFG involves computing a *repetition vector* and a *firing order*, the two problems are:

1. Computing the number of times each SDF block has to be fired that we refer to as the *repetition vector*.

2. Finding the order in which the SDF blocks have to be executed, which we term *firing order*.

Solution to the above two problems yields an *executable schedule* for an SDF graph, assuming that the SDFG is also consistent.

3.1 SDF Scheduling: Repetition vector

The first issue of creating a valid *executable schedule* is discussed in [10] which utilizes the predefined consumption and production rates to construct a set of linear Diophantine [2] homogeneous system of equations and represent it in the form of a topology matrix Γ. It was shown in [10] that in order to have a solution, Γ must be of rank $s-1$ where s is the number of blocks in the SDFG. Solving this system of equations to an integer-valued solution produces the *repetition vector* for the SDF model. An algorithm using Hilbert's basis [2] solves these linear Diophantine equations using the Completion procedure [2, 4, 15]. We use a modification of the algorithm in [2] and add a heuristic specific to SDF models, but due to lack of space we guide the reader to reference [16] for the details.

3.2 SDF Scheduling: Firing Order

The second issue of determining the *firing order* of execution of these function blocks is resolved by using an algorithm in [1]. This algorithm handles the *firing order* of the SDF

blocks and can appropriately schedule cyclic SDFGs as well. Since SDFGs are not limited to directed acyclically connected graphs, we have to ensure that cycles are handled. In fact, many systems require feedback loops and some even require non-trivial cycles. Bhattacharyya, Murthy and Lee in [1] developed scheduling algorithms for SDF of which one scheduling algorithm determines the *firing order* of non-trivial (cyclic or acyclic) SDFGs. In [1], they introduce *delay* that represents the number of tokens the modeler has to inject into a particular arc in order for the model to execute. The concept of *delay* is necessary when considering cyclical SDFGs. Due to the additional constraint set by the SDF paradigm that every block only executes when it has sufficient input tokens on its input arcs, thus a cycle without a *delay* causes the execution of the SDFG to deadlock. The *delay* acts as an initial token on that arc to allow the simulation to begin in these circumstances. The *repetition vector* from the Diophantine solver is necessary for this algorithm to create a valid *firing order*. The algorithm's initial step is in traversing through all the blocks saving the *delay*s, after which blocks that have incoming edges with sufficient modeler injected tokens on the arc are pushed onto a *ready* queue. This incorporates arcs that result to a cycle. The algorithm takes a block from the *ready* queue and recalculates the state of the tokens in the SDF model assuming that this block is fired. This is followed by recalculating whether every reachable block from the block popped off the *ready* queue has sufficient tokens to be scheduled. Once scheduled, this procedure is repeated. A detailed description of this algorithm along with the pseudo-code is available in [1].

4. SYSTEMC DE KERNEL VS. SYSTEMC SDF KERNEL

The existing SystemC scheduler employs the Evaluate - Update paradigm to process the modules in the model as shown in Figure 2. During initialization, the Evaluate phase schedules all processes that are ready-to-run, though the order selection of the execution of the processes is unspecified. The executing process can cause other processes to be ready-to-run in the same delta cycle, due to immediate event notifications or in future cycles using delayed notifications. When all the ready-to-run processes for that specific delta cycle have been evaluated, the scheduler enters the Update phase by updating the signal values calculated in the Evaluate phase. According to the possibility of pending delayed notifications, the scheduler determines the processes that are ready-to-run due to these notifications after which the scheduler begins iterating through the Evaluate-Update phases until the end of the simulation. The end of simulation is realized when there are no timed notifications, but if there are pending timed notifications, then the scheduler advances the current simulation time to the earliest pending timed notification and continues to iterate through the Evaluate-Update phases until all timed notifications have been processed. According to Figure 2, an SDF design modeled with the underlying DE kernel causes unnecessary traversals through the Evaluate-Update stages due to dynamic scheduling of the blocks. Our focus is in reducing this for SDF models such that the SDF blocks are Initialized, Evaluated and then Updated completing the execution of the SDF model. This reduces the number of delta cycles incurred during simulation of an SDF model making simulation more efficient.

After investigating the current Discrete-Event scheduler we envisioned the SDF kernel to follow the same Evaluate-Update

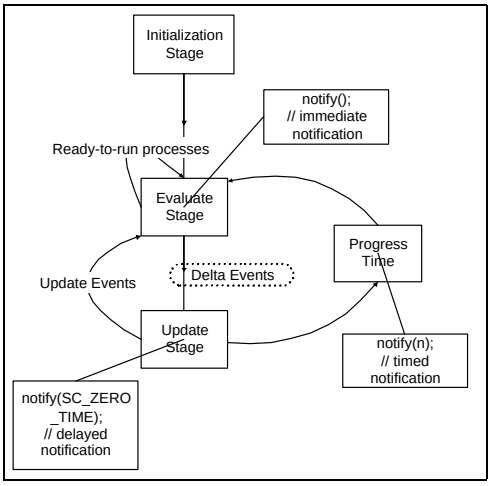


Figure 2: Simulation Kernel

paradigm with added specifications. These specifications involve computing the *executable schedule* for every SDFG in the model during initialization, storing the processes specific to SDF in an appropriate data structure and executing them in the computed order when the SDF model is to be executed. In addition to adding these specifications, we provide guidelines in creating models for SDF using our SDF kernel.

4.1 Implementation Details

To understand the kernel implementation it is necessary to have brief knowledge of the modeling requirements for an SDF design using our SDF kernel. We describe this by showing Figure 3 as a Finite Impulse Response model. The Stimulus, FIR and Display are SDF blocks limited to being *SC_METHOD* process types and in separate modules. This means each SDF block requires one module declaration. Each of these modules communicate through C++ *queue<...>* type queues and are encapsulated in a top-level process of any type. The entry function, which describes the behavior of that SDF block is bound to an *SC_METHOD* process type. This entry function must have a call to *sdf_trigger(...)* to inform the kernel that this process is a top-level process for an SDF model. This top-level process can be signaled to execute using signal/channels from other Discrete-Event modules. Detailed modeling guidelines are documented in [16].

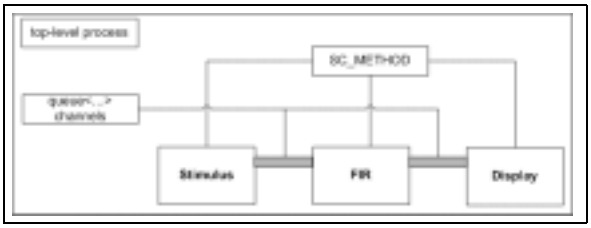
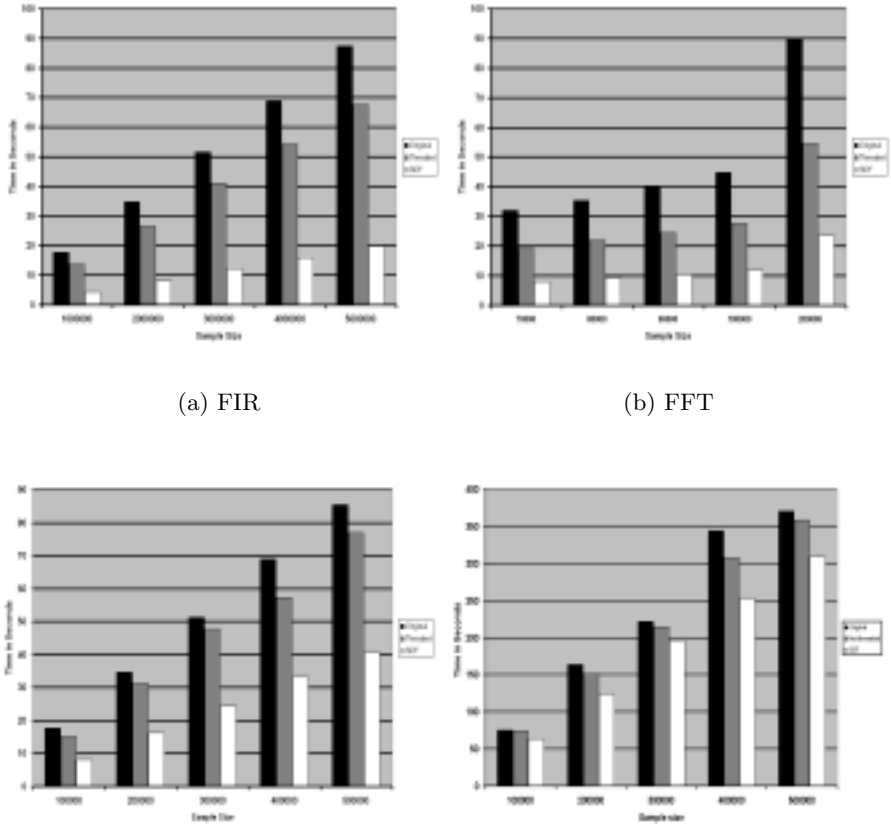


Figure 3: FIR Model

To re-iterate, the Discrete-Event and our Synchronous Data Flow execution semantics are the same, in terms of both MOCs following the Evaluate-Update paradigm. However, algorithms for the *firing order* and *repetition vector* are executed during initialization of the kernel. This allows for immediate vali-



(a) FIR (b) FFT

(c) Sobel (d) Converter

(Black = Original DE models, Dark Gray = Non-Threaded models, White = SDF models)

Figure 4: Results from Experiments

dation whether the SDF models in the complete model are *executable*. With incorrect construction of the SDF model, there can be inconsistent SDF graphs that we identify during initialization causing termination of the simulation along with an appropriate error message. Once the construction of *executable schedule*s is successful, the initialization of all ready-to-run processes is performed. However, during this initialization we separate all processes marked as SDF blocks. To clarify, SDF blocks are modules that make up an SDF model that are then encapsulated in a top-level process. The SDF blocks are removed and stored separately. However, the top-level process remains as a normal DE process. We do this to maintain the existing execution semantics. When the simulation kernel schedules the top-level process to execute, its entry function is fired. We guide the modeler to call the *sdf_trigger(...)* function that takes in the module name as a parameter, locates that SDF model and executes its corresponding SDF blocks. As mentioned earlier, these SDF blocks are only processes of type *SC_METHOD* that allow for a start to end execution. Once *sdf_trigger(...)* returns, the top-level process either suspends or completes its execution and resumes execution of the kernel through its Evaluate-Update stages.

5. EXPERIMENTS AND RESULTS

In order to evaluate the efficiency enhancement by our SDF kernel, we set out to experiment with a few models that are amenable to SDF style modeling. We show the results of simulating the same models for three distinct modeling styles, for different sample sizes of data. In [20] around 50% or more improvement in simulation efficiency over threaded models have been reported. Our aim has been to improve upon those results reported in [20], which we call "Non-Threaded" models.

Four systems are modeled using the SDF kernel and the SDF modeling style. They are Finite Impulse Response Filter (FIR), Fast Fourier Transform (FFT), a Sobel edge detection system and a heterogeneous model that we call the Converter that was introduced in Figure 1.

As seen in the graphs, every model showed improvement in the amount of simulation time that was taken over the original model and the non-threaded model. The three bars on each chart refer to the time taken in seconds for the entire model to be executed in their respective modeling environments. The leftmost bar being the original and middle bar being non-threaded are modeled using the discrete event kernel and the rightmost bar is the synchronous data flow using the SDF kernel. The bar charts show that increasing the number of sample size will still preserve the efficiency presented by

the set of collected data. The FIR and FFT yielded approximately 75% improvement in simulation time compared to the original model and the Sobel yielded a 53% improvement in simulation time. The Converter model yielded 13% improvement. When the results from the SDF kernel is compared to the non-threaded models, the FIR, FFT, Sobel and Converter, showed 70%, 57%, 47% and 8% improvement in simulation time respectively. We attribute the inconsistent decrease in simulation efficiency amongst models to Amdahl's law. For example, the SDF block in the Converter model serves only a small portion of the entire system allowing for only that much improvement in total simulation performance. This means that when we construct models with numerous SDF blocks, the simulation efficiency of that model would be significantly larger. These experiments were executed on a Linux 2.4.18 platform with an *Intel® Pentium®* IV CPU 2.00GHz processor, 512KB cache size, and 512MB of RAM.

6. CONCLUSION AND FUTURE WORK

In this paper we present an SDF kernel that works in unison with the existing DE kernel to allow designers to model heterogeneous systems limited by the MOCs implemented so far. We provide a static scheduling method during compile time for the SDF kernel that improves simulation efficiency over models implemented using the original DE kernel. The SDF nature of the models is a prerequisite for this improvement in performance and modeling these systems. Using the DE kernel for SDF models proves to be to some extent unnatural, such as the need for communication signals for control passing. Granted that the SDF kernel introduces some modeling guidelines, but the efficiency gained by the new modeling paradigm overweighs the introduction of these rules. Furthermore, it provides heterogeneity to some limited extent in SystemC's simulation framework. Efforts to standardize the SystemC kernel is necessary such that an application-protocol interface (API) can exist to provide a tidy mechanism of producing alternative kernel implementations and incorporating them with the existing SystemC distribution. To our knowledge there has been no attempt at introducing heterogeneity in SystemC.

The successful co-existence of SystemC DE and SDF kernels introduces a realm of possibilities for multi-domain modeling. Primarily, industries that have IP-cores written in C/C++ languages that depend on MOCs other than DE can be integrated with SystemC given that support is provided by SystemC for that particular MOC. Efficient and fast simulations can be executed reducing the bottleneck in the design cycle. The capability of modeling in SystemC extends to synthesis where tools supporting SystemC synthesis may be able to accommodate heterogeneity.

Currently, we are working on restructuring the SystemC kernel entirely to allow the ability to model behavioral hierarchy, which may extend across various MOCs. Structural hierarchy is not enough for behavioral decomposability and behaviorial hierarchy is necessary to provide hierarchical MOC models. This requires restructuring the communication between events/signals/channels and the class hierarchy to allow for a tidy execution of multiple MOCs. We are considering this alteration in concert with implementing Continuous Time, Finite State Machine, Kahn Process Network and Communicating Sequential Processes Models Of Computation with SystemC. We believe this will further solidify the modeling of SOCs at system level with SystemC by making the design of heterogeneous system models even simpler and efficient.

7. REFERENCES

[1] S. Bhattacharyya, P. Murthy, and E. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Publishers, 1996.

[2] M. Clausen and A. Fortenbacher, *Efficient solution of linear Diophantine equations*, Journal of Symbolic Computation **8** (1989), no. 1-2, 201–216.

[3] E. Lee et al, *Heterogeneous Concurrent Modeling and Design in Java: Introduction to Ptolemy II*, Memorandum UCB/ERL M03/27, July 2003.

[4] A. Fortenbacher, *Algebraische unifikation*, Master's thesis, Universitat Karlsruhe, 1983.

[5] T. Grotker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.

[6] Ptolemy Group, *Ptolemy II*, Website: http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[7] T. Guckenbiehl and A. Herold, *Solving linear diophantine equations*, Tech. Report SEKI-85-IV-KL, Universitat Kaiserslautern, 1985.

[8] G. Huet, *An algorithm to generate the basis of solutions to homogeneous linear Diophantine equations*, Information Processing Letters, April 1978, 7(3).

[9] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, *Models of Computation for Embedded System Design*, Website: citeseer.nj.nec.com/lavagno98model.html, 1998.

[10] E. A. Lee and D. G. Messerschmitt, *Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing*, In Proceedings of IEEE Transactions on Computers, NO. 1, vol. Vol. C-36, 1987.

[11] Edward A. Lee and Alberto L. Sangiovanni-Vincentelli, *Comparing Models of Computation*, In Proceedings of the International Conference on Computer-Aided Design (ICCAD), 1996, pp. 234–241.

[12] A. Michalove, *Amdahl's Law*, Website: http://home.wlu.edu/ whaleyt/classes/parallel/topics/amdahl.html.

[13] B. Niemann, F. Mayer, F. Javier, R. Rubio, and M. Speitel, *Refining a High Level SystemC Model*, Kluwer Academic Publishers, 2003, In SystemC: Methodologies and Applications, Ed. W. Muller and W. Rosenstiel and J. Ruf.

[14] OSCI, *SystemC* , Website: http://www.systemc.org.

[15] D. Pasechnik, *Linear Diophantine Equation Solver*, Website: http://www.thi.informatik.uni-frankfurt.de/ dima/software.html.

[16] Hiren D. Patel, *HEMLOCK: HEterogeneous ModeL Of Computation Kernel for SystemC*, Master's thesis, Virginia Polytechnic Institute and State University, December 2003, Website: http://fermat.ece.vt.edu.

[17] N. Savoiu, S. K. Shukla, and R. K. Gupta, *Automated Concurrency Re-assignment in High Level System Models for Efficient System Level Simulation*, In Proceedings of Design Automation and Test (DATE02), 2002.

[18] ———, *Concurrency in System Level Design: Conflict between Simulation and Synthesis goals*, International Workshop on Logic and Synthesis (IWLS 02), 2002.

[19] ———, *Efficient Simulation of Synthesis Oriented System Level Designs*, In the proceedings of International Symposium on System Synthesis (ISSS'02), 2002.

[20] S. Sharad and S. K. Shukla, *Efficient Simulation of System Level Models via Bisimulation Preserving Transformations*, Tech. report, FERMAT Lab Virginia Tech., 2003-07.

[21] SPECC, *SpecC*, Website: http://www.ics.uci.edu/specc/.

[22] SystemVerilog, *System Verilog*, Website: http://www.systemverilog.org/.

[23] VERILOG, *Verilog*, Website: http://www.verilog.com/.

[24] VHDL, *VHDL*, Website: http://www.vhdl.org/.