

A HIERARCHICAL SIMULATION FRAMEWORK FOR APPLICATION DEVELOPMENT ON SYSTEM-ON-CHIP ARCHITECTURES

Vaibhav Mathur and Viktor K. Prasanna

Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-2562
{vaibhav, prasanna}@usc.edu

ABSTRACT

We propose a hierarchical simulation methodology to assist application development on System-on-Chip architectures. Hierarchical simulation involves simulation of a SoC based system at different levels of abstraction. Thus, it enables a system designer to exploit simulation speed vs. accuracy of results trade-offs. Vertical simulation is a special case of hierarchical simulation, where a feedback mechanism between the different simulation levels helps in “interpreting” the results of stand-alone simulations in the system-wide context. The paper presents an approach to perform vertical simulation of a class of applications under a simplified scenario.

I. INTRODUCTION

Performance requirements under ubiquitous computing, and convergence of communication and computing technologies have resulted in the emergence of System-on-Chip (SoC) architectures. Application development on such architectures involves a delicate balance between high performance requirements and constraints on area, power, etc. State-of-the-art design tools and methodologies are not adequate to manage the design complexity of SoCs. Current design processes are based on independent design flow for each architecture component and have not co-evolved with changing system designs and requirements. Programming models and design tools for each component are independently utilized to map an application, and system integration is performed later. System-wide performance analysis is typically a manual process. It involves the use of component specific simulators in isolation, and is tedious since each simulator has a different input/output interface. This approach results in sub-optimal design because multi-objective optimization requires exhaustive traversal of a large design space. Several environments [9], [15] have emerged that concentrate more on co-simulation and system synthesis than on high level system design. Design problems (e.g. improper application-to-architecture mapping, insufficient resources to meet the performance requirements, etc.) detected during such co-simulation leads to tedious and time consuming redesign of the system [6].

This work is supported by the US DARPA Power Aware Computing and Communication Program under contract F33615-C-00-1633 monitored by Wright Patterson Air Force Base.

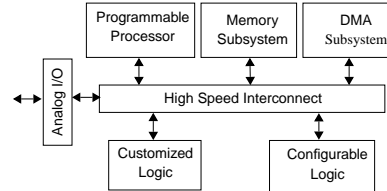


Fig. 1. A typical SoC architecture

The above scenario necessitates a system level design approach. A unified simulation environment is needed that provides performance estimates (latency, throughput, average power, etc.), for a given application-to-architecture mapping, at desired detail and cost (time) of simulation. This will enable rapid evaluation of performance trade-offs for alternate mappings at a high level and early in the design cycle without performing time-intensive low-level simulations. This paper outlines a hierarchical simulation methodology for this purpose.

The work described in this paper is part of the MILAN¹ project, which is a collaborative research effort between the University of Southern California (USC) and the Institute for Software Integrated Systems (ISIS) at Vanderbilt University.

MILAN is a **Model based Integrated simuLAtion** framework for embedded system design and optimization [2]. The designer formally models the target application, underlying hardware, and constraints (latency, throughput, power, etc.) through a graphical interface provided by MILAN. The information is stored in a model database that can be accessed through a simple Component Object Model (COM) [13] interface. The model information is translated into suitable input formats required by the integrated simulators. Thus, MILAN has the capability to drive multiple simulators with different input/output formats from a single system specification.

Hierarchical simulation means simulating a design at different levels of abstraction in terms of both the structure and the behavior of the underlying component(s). *Vertical simulation* is a special case of hierarchical simula-

¹milan (hindi): meeting, unification. <http://milan.usc.edu/>

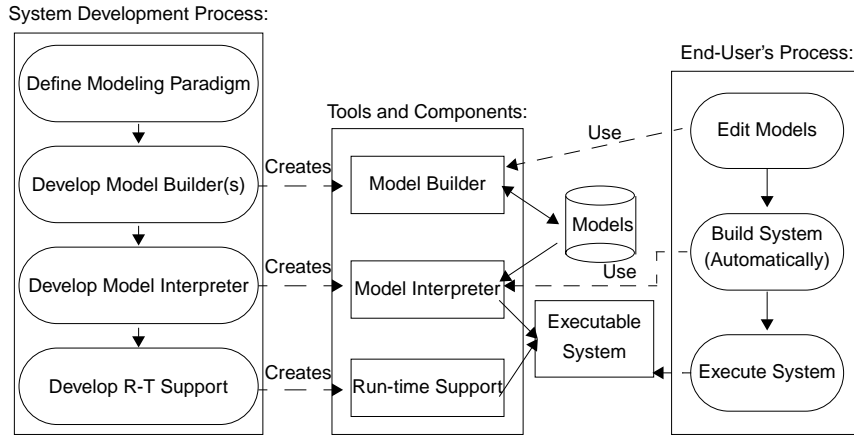


Fig. 2. Model Integrated Computing

tion, where a designer is interested not only in simulating a system sub-component (a task-resource mapping) at multiple levels of granularity, but also in “meaningful” interpretation of low-level simulation results from a system-wide perspective. For example, the throughput of one system sub-component might affect the rate of availability of data to a data dependent system sub-component. Also, the computation of a particular task might depend on input data values, that can sometimes only be determined by simulating “real-world” system operation via realistic data sets. Vertical simulation is useful when (a) high level end-to-end performance estimates are to be further refined through low-level simulation, and (b) a particular system sub-component is to be simulated at a fine granularity, taking into account system-wide effects such as those mentioned above. A high level estimation methodology and a feedback mechanism, explained in detail in Sections II and III, helps in achieving both these objectives.

Vertical simulation will typically be useful in the later phase of the design cycle, when high-level, system-wide performance estimates for a particular mapping are to be refined through low-level simulations. High-level simulations are coarse-grained approximations which yield very rapid results with possible compromise in accuracy. Low-level simulations typically are highly accurate but are time consuming. Hierarchical simulation with interpretation of simulation results allows the user to exploit the trade-offs between simulation speed and accuracy of results.

MILAN focuses primarily on integration of simulators for Instruction Set Architecture (ISA) based processors (e.g. RISC, DSP), application-specific cores, configurable logic (e.g. FPGA), memories, and interconnects. Numerous commercial platforms targeted towards communication and networking applications, have been introduced lately. A typical platform is shown in Figure 1. The Universal Microsystem (UMS) [8], the Reconfigurable Com-

munications Processor [7], the Jazz PSA [11], and the Platform FPGA [16], are a few examples. These platforms include programmable processor(s), configurable logic, customized logic, embedded memories, high speed interconnect, and analog I/O components on the same chip. MILAN integrates simulators for digital components only; simulators for analog components are not considered.

The paper is organized as follows. Section II describes the MILAN project and the underlying Model Integrated Computing (MIC) [18] design approach. Section III defines hierarchical and vertical simulation concepts. Section IV discusses a prototype implementation of vertical simulation for a class of applications. Section V has concluding remarks.

II. THE MILAN PROJECT

The focus of the MILAN project [2] is on developing formal modeling paradigms that will enable simulator integration and efficient application-to-architecture mapping through automatic design space exploration. MILAN adopts Model Integrated Computing (MIC) [18] as the core design technology (Figure 2). MIC is especially valuable for the design of computer-based systems with strong interdependence between the hardware and the software components. By formally modeling all aspects (application, resource, behavior, constraints, etc.) of a system and using well-defined rules to generate new systems or manage existing ones, it is possible to avoid the errors that arise when requirements change and the system has to be redesigned or re-implemented. While the initial modeling effort might be costly compared to ad hoc approaches, the benefits are apparent for a system that evolves over time.

An environment that supports the MIC allows the designer to create domain-specific models at the required level of abstraction, validate these models, and perform various computational transformations on them (Figure 2).

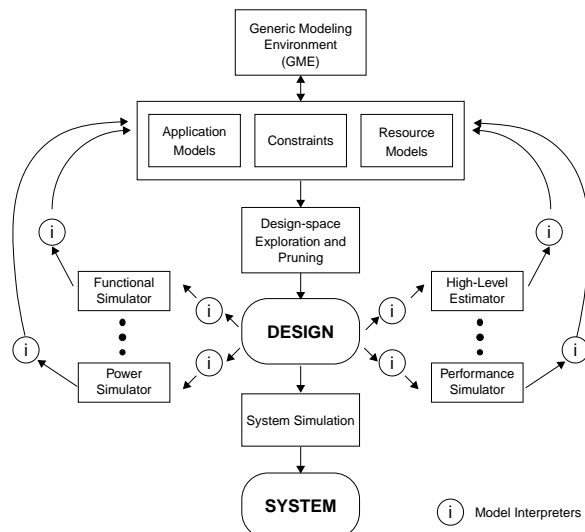


Fig. 3. MILAN Architecture

A *metamodel* is a formal description of the modeling environment’s model construction semantics, i.e. the *modeling paradigm*. Syntactically and semantically correct models are created using modeling paradigms. The metamodel defines the syntactical building blocks available to the designer for describing (instantiating) the system. It also defines the domain-specific composibility rules and the constraints that enforce some of the semantics in the system specification. Model interpreters are the software components that translate the models for use in the MILAN execution environment. A model database is a key component that stores the models and the translated information in a canonical form that provides a common representation for the information that are used in driving various simulators. The model database also stores the simulation results.

The Generic Modeling Environment (GME) [10] is a configurable graphical tool suite supporting MIC. The configuration of the environment to support domain-specific modeling is performed in a formal manner through the use of metamodels. The metamodeling language is the UML class diagram notation [5]. Well-formedness rules that are also part of the metamodels are specified using the Object Constraint Language (OCL). These constraints, along with the syntactical rules of the domain language, are enforced by the automatically generated target environment. MILAN exploits the MIC technology to configure an environment tailored for embedded system design, evaluation, and optimization. The framework incorporates power as an important design metric. Power estimation and optimization is supported through integration of existing component specific power simulators, based on system-wide power models.

Figure 3 shows the architecture of MILAN and also depicts the system design flow from the users’ perspec-

tive. The graphical interface is provided by GME configured to support the modeling paradigms developed for MILAN. Semantic information in the modeling paradigms is captured through metamodels. The metamodels configure the GME for creating domain-specific models. Following models for the design of embedded systems based on SoC architectures are currently supported in MILAN.

Resource models [3] describe available hardware components and their interconnectivity in a hierarchical block diagram-like notation. *Application* models are based on a hierarchical signal flow representation with important extensions. Most notably, the modeling language allows for the specification of explicit design or implementation alternatives of any component. This enables modeling of the entire design space of the application as opposed to a point solution. To manage this design space, application requirements, resource constraints, and other specifications are captured explicitly through OCL, in the metamodels. *Performance* modeling of SoC architectures involves characterizing desired performance metrics of a given mapping in terms of architecture parameters. The performance model leverages from prior USC work in high level performance models for traditional and reconfigurable architectures [1][14]. The *communication* model provides a common formalism to enable inter-operability of simulators that represent the same information in different formats.

The design-space exploration and pruning tool takes the potentially very large design space and applies the constraints using a symbolic constraint satisfaction technique to find the set of solutions that satisfy all the constraints. The goal of design space exploration is to identify a small number of valid candidate designs.

MILAN supports an estimation methodology that integrates various component-specific performance models (power, time) and enables system-wide performance evaluation. The methodology is based on a high level system-wide performance model to evaluate complex application to architecture mapping choices and different schedules of execution. Although, ideally a low-level (detailed) system wide performance model can be defined, it is impractical since diverse architecture features (reconfigurable vs. customized logic, variable vs. static parameters etc.) cannot be captured uniformly. Moreover, simulations based on it will be cost (time) intensive. The design of a *High Level Performance Estimator* (HiPerE) based on a high level performance model is motivated by these issues.

MILAN supports different classes of simulators, besides the HiPerE. Functional simulators (e.g., MATLAB or SystemC), verify the functionality of the application. Several low-level simulators (power, performance) will be supported in future. The HiPerE along with the low-level simulators integrated in MILAN facilitates multi-level simulation, which exploits the trade-off between the accuracy of results and the simulation speed.

III. HIERARCHICAL AND VERTICAL SIMULATION

MILAN has the following capabilities to assist application development under a unified simulation environment.

- *Simulator Integration*: Integrating multiple component specific simulators and driving these simulators with different input/output formats using a single system specification is defined as *Simulator Integration*. The system is specified in terms of application and resource models, and end-to-end performance requirements. The models are stored in the model database that enables seamless simulator integration by providing a common information repository. The model interpreters translate this information into the syntax and the semantics of the inputs required to drive the simulators.

- *Hierarchical simulator integration*: “Hierarchical” refers to multiple levels of abstraction of the models describing the system. Hierarchical simulation implies simulating the system or a system sub-component at different levels of abstraction and implementation. Thus, providing hierarchical simulation capability for a particular task-to-resource mapping means providing the user with a choice of simulators for that resource, at different levels of granularity (assuming they are integrated into the framework). *Simulator integration of component-specific or system-wide high level and low level simulators is termed as hierarchical simulator integration*. High-level simulators are typically based on a few key parameters of the system (or system sub-component) and are designed to provide rapid performance estimates, possibly at the expense of accuracy. Low-level simulators (such as cycle-accurate simulators) are highly accurate but can require an order of magnitude more simulation time and a much larger set of input parameters to be specified, compared to high-level simulators. The hierarchical, block-diagram like specification of the resources themselves also provides different levels of abstraction corresponding to the different granularities of resource representation required by high-level and low-level simulators.

- *Vertical simulation*: Hierarchical simulator integration enables the simulation of a system sub-component (a task mapped onto a compute resource) at different levels of granularity, using any of the integrated simulators for that component, thereby allowing the designer to exploit the simulation time vs. accuracy of results trade-off. There are two scenarios when stand-alone, multi-granular simulation is not useful: (a) when the designer is interested in refining high-level estimates of a particular mapping, provided by HiPerE, and (b) when the designer wants detailed and realistic statistics about a particular system sub-component. *A meaningful interpretation of the results of hierarchical (multi-granular) simulation, in the system-wide context is defined as vertical simulation*. Vertical simulation uses the hierarchical simulator integration capability and provides

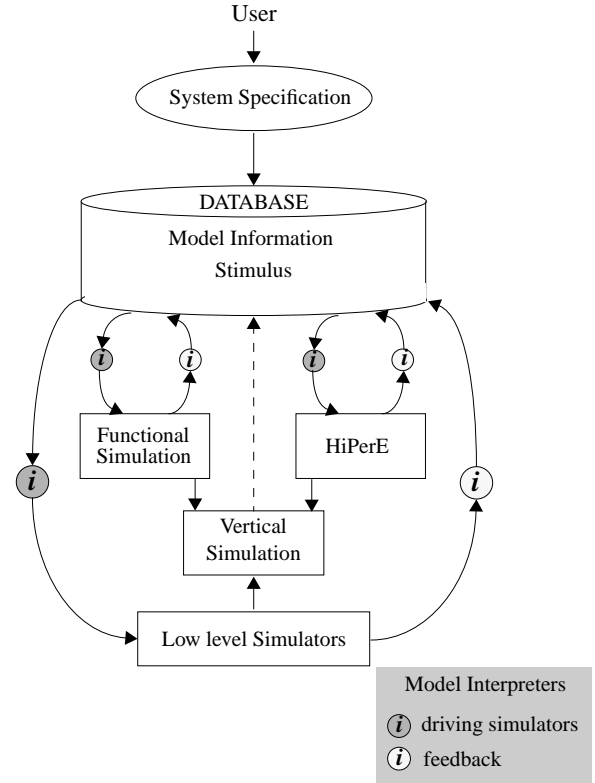


Fig. 4. Hierarchical Simulation Framework

a mechanism to refine a system sub-component or the end-to-end performance estimates of a candidate mapping. The HiPerE system-wide performance estimation methodology does not rely on cycle-accurate simulation of a given application on the target resource at the time HiPerE is invoked. The HiPerE requires pre-characterized costs from experimental results, vendor-supplied information for library components, or any other “offline” method. In the later stages of the design cycle, simulation time will be of less concern because the design space would have reduced to a smaller, manageable set of mappings that satisfy broad system requirements. In the first design scenario as mentioned above, the designer will be interested in refining the HiPerE estimates for a given mapping, by running low-level simulations for selected components. The model database not only stores the models but also the data values processed by the system sub-component, and the results of simulation. One of the crucial information required for invoking any simulator is the incoming data set(s), which form a *stimulus* to the system sub-component (a task-to-compute resource mapping). The stimulus is obtained either through functional simulation of the application or by using data generator scripts for the predecessor tasks in the path of application execution. These methods are explained in detail in Section IV. Since the simulation results are also stored in the database, simulating a component at

low-level implies updating the performance cost associated with it. *By invoking the appropriate low-level simulator, updating the performance costs, and then invoking HiPerE, refined estimates can be obtained (for the first scenario). This bottom-up feedback needs model interpreters that can suitably interpret the low-level simulation results and update the model database.*

The second scenario is much more challenging because it requires a top-down feedback from the HiPerE to enable realistic low-level simulation, as against the bottom-up feedback from the low-level simulation results for refining HiPerE estimates. A realistic simulation is one that provides performance statistics closest to those exhibited by the final system when it is deployed in the real world. Two of the important considerations in arriving at such realistic low-level simulation results are the availability of realistic data values that will be processed by that task, and modeling of system-wide effects such as input/output delays due to sampling rates of sensors, constraints on buffer size, etc. Although the HiPerE and low-level simulation results are both stored in the model database, a common repository by itself is not sufficient to help the user get a realistic set of performance statistics for a particular task, with system-wide effects taken into consideration. *Vertical simulation techniques are required to meaningfully interpret low-level simulation results with top-down feedback from the HiPerE.* The next section addresses this issue in more detail.

There is another aspect to integrating simulators that is outside the scope of the MILAN project. When more than one component-specific low-level simulators are invoked *simultaneously* and system-wide performance is evaluated through real-time interaction among the different simulators, the capability is termed *horizontal simulation*. Although horizontal simulation, if implemented “correctly”, can accurately simulate the actual interaction among sub-components, concurrent execution of component-specific simulators poses multitude of challenges. For example, most widely-used simulators are not designed to interact at run-time, with other systems. Modifying such simulators can prove to be time-consuming and not always feasible. Specifying a common API for simulators to conform to, is an equally challenging task. Finally, there are simulation speed (time) issues that can arise when simulators have to synchronize, say, at every cycle.

IV. VERTICAL SIMULATION: A SIMPLE IMPLEMENTATION

Using the MILAN modeling paradigms, the system designer specifies the target application, underlying resources, the task-to-resource mapping, a schedule of execution of the tasks, and performance constraints. The information is stored in the model database, and model interpreters extract this information and translate it into suitable

formats for driving the integrated simulators. Hierarchical simulation is facilitated through this mechanism, as explained in the previous section. This section outlines vertical simulation implementation for a simplified scenario. Following assumptions are made.

- Application is modeled as a Directed Acyclic Graph (DAG). Each vertex in the DAG represents a task and edges between the vertices represent the data transfer between the tasks. Every task in the DAG is “atomic”. This means that a task cannot be further decomposed into sub-tasks. If the task is dependent on data from multiple tasks, we assume that the task is activated (simulated) when the data from all its predecessor tasks (the stimulus) in the DAG is available.
- Currently, the data processing rate by all the tasks is assumed to be same and constant during the entire execution of the application. There is no change in the rate of processing dependent on a control data.
- Resources consist of compute resources (RISC core, FPGA, etc.), storage elements (memories), and communication channels (buses). Explicit communication through the channels occurs between the compute resources.
- The task-to-resource mapping and the schedule of execution of the tasks on the compute resources together define an execution model for the application. Only one task is executed on a compute resource at any time. There is no resource sharing with others tasks during the execution i.e., the tasks do not compete for resources such as compute cycles, memory etc. The preceding (and succeeding) tasks in the schedule can however be executed on the same resource. Since there is no concurrent execution of the tasks on the same resource, the results of stand-alone simulation (that typically does not model concurrency and resource sharing) accurately represent the computation cost of the task being simulated.

A linear task graph with only one task mapped onto a compute resource at a time, trivially satisfies the assumptions under the application and the execution model. The sequential data dependence between the tasks satisfies the requirement that the stimulus (consisting of just one data value) is available before the task is initiated.

For vertical simulation, the user selects a particular task and an associated low-level simulator for the resource onto which that task is mapped. Invocation of a simulator requires the following inputs: (a) The stimulus (the incoming data set to the task) (b) the simulator configuration parameters, and (c) an appropriate implementation of the task (e.g. C code for a RISC simulator or a VHDL code for an FPGA simulator). These three inputs are provided by the model interpreter.

The stimulus S_i to a task i denoted by vertex V_i is modeled by two vectors: $S_i = [\{d_1^i, d_2^i, \dots, d_k^i\}, \{t_1^i, t_2^i, \dots, t_k^i\}]$, where d^i represents input data value, t^i represents the arrival time of the value, and k is the in-degree of V_i . The

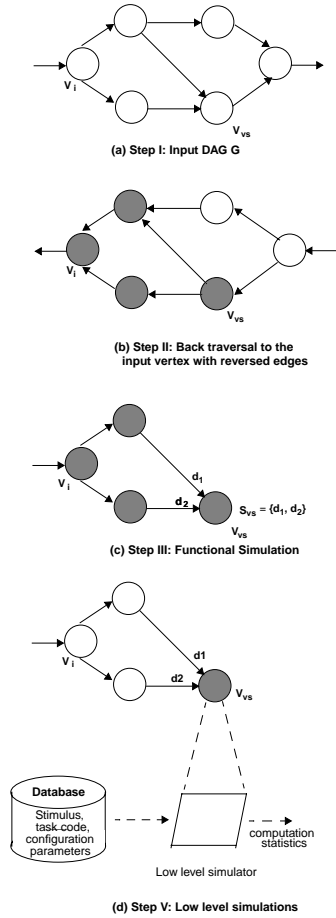


Fig. 5. Steps for vertical simulation

rate of arrival and the data values are crucial for realistic simulation of the task. Our simple vertical simulation technique focuses on task latency and not throughput, i.e. the arrival rate of data at a node is not currently modeled. For determining the stimulus, the following method is used (see Figure 5).

The task to be vertically simulated V_{vs} , and the input vertex V_i for the entire application are marked in the application DAG G . To determine all data flows that lead from V_i to V_{vs} , the direction of the edges is reversed and the graph is traversed starting from V_{vs} . All the vertices on all the paths leading from V_{vs} to the input vertex V_i are determined. Functional simulation is then performed for all these vertices (tasks) according to the dependencies specified by the original graph G , and the data stimulus for V_{vs} is obtained.

Another method for obtaining the stimulus is requiring the user to associate a script with every task. This script generates “dummy” output that represents the actual output, had the task been executed with real input data. If such scripts are provided, obtaining the data stimulus for a specific task will involve executing the dummy data gener-

ators only for its immediately preceding tasks. Functional simulation is still preferable, because the computation cost of a task is most accurately obtained through simulation that uses data values generated during the actual application execution. For example, consider an image processing application. Computation of the correlation coefficients for an image vector depends on the intensity of the pixels, which in turn depends on the relative distance on the RGB color scale. Generating dummy vectors that represent actual images “seen” by the system in the real world can be difficult or even impossible.

While the stimulus obtained through functional simulation and generator scripts is stored in the model database, some typical data values generated during prior application development, and tested for the worst or the best case performance satisfaction can as well be stored. These can be reused for the current application development process as stimulus for low-level simulations.

Once the stimulus is obtained, the appropriate low-level simulator is invoked. The results from low-level simulation need to be suitably interpreted and used to update the computation costs associated with the task in the model database. *This is a bottom-up feedback mechanism, where results from low-level simulation are used to obtain more accurate system-wide performance estimates through HiPerE.* Stand-alone simulation of a task, however, does not model the system-wide effects on task execution, such as input/output delays due to sampling rates of sensors, constraints on buffer size, etc. The simulation also does not capture the state of the processing element between task executions. For example, the cache contents after a task completes execution affect the initial memory access costs (compulsory misses) for the next task that executes on the same resource. These access costs will be available through HiPerE and will also be stored as a stimulus in the database. However, this stimulus is required with the output of the low-level simulations. *HiPerE provides such top-down feedback necessary for system-wide interpretation of the low-level simulation results.* The estimates of system-wide effects provided by HiPerE, such as input/output data delays, are used in conjunction with stand-alone simulation results to arrive at more accurate performance statistics for the system sub-component. The issues in implementing vertical simulation for a general scenario are discussed in Section V.

The vertical simulation outlined above is being implemented in MILAN. The MILAN application modeling paradigm allows graphical representation of the target application as a DAG, with task implementations specified for each node of the DAG (C source code, HDL implementations, etc.). The current version of the resource modeling paradigm is capable of representing uniprocessor architectures as modeled by the SimpleScalar simulator [17], a prototype integration of which has been completed for MI-

LAN. Currently, for specifying task-to-resource mapping, a task node in the application graph is explicitly associated with the underlying resource label. MATLAB has been integrated into MILAN for functional simulation. These capabilities of MILAN enable a user to functionally simulate the entire application, and perform low-level simulation for a selected task-resource pair. Vertical simulation, as described previously, requires additional capabilities that are being implemented. These include graph traversal from the specified node in the direction of the edges, functional simulation of an application sub-graph, design and implementation of model interpreters for bottom-up feedback, etc. A preliminary design of HiPerE has been completed, and implementation is under progress.

Vertical simulation for application development provides the capability to perform multi-level simulation for a particular application-to-resource mapping. The designer can evaluate hardware vs. software speedup for a task and can change the mapping for that task only, or for the entire system if desired. An FFT algorithm implemented as Decimation in Time and Frequency (DITF) is less computationally intensive (in terms of floating point operations) as compared to DIT or DIF radix-2 implementations [4]. A designer can evaluate the end-to-end performance enhancements using such alternate algorithmic implementations.

V. CONCLUDING REMARKS

This paper discussed the concept and need for hierarchical and vertical simulation, and a simple implementation in the context of MILAN. Some important issues that are applicable in a more general scenario remain to be addressed. A few of the important characteristics of the general scenario are:

- Tasks are executed concurrently on the same resource.
- The latency and throughput of a task change depending on control data. This in turn, affects the rate of availability of data to the data dependent tasks in the application execution.
- Communication between compute resources is through different mechanisms, such as global memory. This impacts the end-to-end latency, power consumption, etc.

We are currently enhancing HiPerE to address the issues in the general scenario. The feedback mechanism (bottom-up and top-down) as discussed in the paper will be implemented and enhanced. This will involve writing appropriate model interpreters. In future, the refinement of the high level model parameters themselves will be performed, based on low-level simulation results. We envision that our framework will address the system level design challenge facing the SoC community by providing an accurate mechanism to evaluate interactions between SoC components at various levels of granularity, by exploiting the simulation time vs. accuracy of results trade-off.

VI. ACKNOWLEDGMENTS

We would like to thank Amol Bakshi (USC), Sumit Mohanty (USC), and Akos Ledeczki (ISIS). Amol provided invaluable feedback on simulator integration, and hierarchical simulation integration aspects of MILAN (the subject of this paper). Sumit provided inputs on HiPerE, and Akos provided inputs on the MILAN architecture.

REFERENCES

- [1] Algorithms for Data Intensive Applications on Intelligent and Smart Memories (ADVISOR), Univ. of Southern California. <http://advisor.usc.edu>.
- [2] A. Bakshi, V. K. Prasanna, A. Ledeczki, et al., "MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems," ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'2001), Utah, June 2001.
- [3] A. Bakshi and V. K. Prasanna, "Abstract Resource Representations for Custom Design of System-on-Chip Architectures," submitted to IFIP VLSI-SOC 2001, Montpellier, France, December 2001.
- [4] M. Balducci, A. Ganapathiraju et al., "Benchmarking of FFT Algorithms," IEEE Southeastcon '97, Engineering New Century, Proceedings, pp. 328-330.
- [5] G. Booch et al., "The Unified Modeling Language User Guide," Addison-Wesley Pub Co., 1999.
- [6] H. Chang et al., "Surviving the SOC Revolution A Guide to Platform-Based Design," Kluwer Academic Publisher, Massachusetts, USA, 1999.
- [7] Chameleon Systems Inc., <http://www.chameleonsystems.com>.
- [8] Cradle Technologies, <http://www.cradle.com>.
- [9] Synopsys EagleI, http://www.synopsys.com/products/hwsw/eagle_ds.html.
- [10] Generic Modeling Environment, <http://www.isis.vanderbilt.edu/projects/gme/default.html>.
- [11] Improv Systems Inc., <http://www.improvsys.com>.
- [12] The MILAN Project, <http://milan.usc.edu>.
- [13] Microsoft Component Object Model, <http://www.microsoft.com/com/>
- [14] Models, Algorithms and Architectures for Reconfigurable Computing Project, <http://maarc.usc.edu>.
- [15] CoWare N2C, <http://www.coware.com/cowareN2C.html>.
- [16] Xilinx Inc., <http://www.xilinx.com>.
- [17] SimpleScalar Tool Set, <http://www.cs.wisc.edu/mscalar/simplescalar.html>.
- [18] J. Sztipanovits, G. Karsai, "Model-Integrated Computing," IEEE Computer, April, 1997.