Rapid System-Level Performance Evaluation and Optimization for Application Mapping onto SoC Architectures

Sumit Mohanty and Viktor K. Prasanna
Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
{smohanty, prasanna}@usc.edu

Abstract— **System-on-Chip (SoC) architectures integrate several heterogeneous components onto a single chip. These components provide various capabilities such as dynamic voltage scaling, reconfiguration, multiple power states, etc. that can be exploited for performance optimization during application design. We propose a Generic Model (GenM) which captures the capabilities of a large class of SoC architectures and facilitates evaluation and optimization of performance during application design. GenM model is used as an abstraction to identify various well-defined optimization problems for application mapping onto SoC architectures. Using GenM, we developed an interpretive simulator, High-level Performance Estimator (HiPerE). It integrates component specific performance estimates to rapidly evaluate performance at the system level. The MILAN framework enables hierarchical simulation through the integration of HiPerE and low-level component specific simulators into a unified environment. Hierarchical simulation facilitates efficient design space exploration during application mapping onto SoC architectures.**

I. Introduction

System-on-Chip (SoC) architectures integrate several heterogeneous components [8] such as general purpose processors, reconfigurable logic, DSPs, ASICs, memory, among others. These components provide several features such as dynamic voltage variation, reconfigurability, multiple clock domains, and multiple implementation alternatives that can be exploited during the application design process to achieve optimization [9]. However, these features introduce a multitude of design challenges.

To address these challenges, high-level abstraction and system-level performance estimation are necessary to evaluate the overall effect of various component specific design decisions and also to identify bottlenecks and optimization possibilities. System-level performance estimation and efficient traversal of design space for SoC architectures are challenging due to lack of high-level abstraction for SoC architectures, time-consuming low-level (cycle-accurate or RT-level) simulations, and lack of standard interface between different component specific simulators.

Generic Model (GenM) provides an abstraction of the SoC architectures that identifies the key architectural features that can be exploited for performance optimization. GenM model can be used to analyze latency and energy during application mapping onto SoC architectures. We can identify various well-defined optimization problems based on GenM specifically for SoC architectures. The rapid performance estimation tool based on GenM, **Hi**gh-level **Per**formance **E**stimator (HiPerE), combines component specific performance estimates to evaluate performance at the system-level using interpretive simulation. Given a task graph with N nodes and Q edges, the complexity of performance evaluation using HiPerE is $\theta(N+Q)$. HiPerE also provides an activity report that captures the state of each component during application execution. In this paper, performance refers to both latency and energy dissipation.

The work described in this paper is part of the MILAN project. MILAN is a **M**odel based **I**ntegrated simu**LA**tio**N** framework for embedded system design and optimization through integration of various simulators into a unified environment [1]. Using the MILAN framework, a designer formally models the target application and architecture. The information captured in the models are used to drive various simulators from a single interface. Hierarchical simulation refers to performing simulation of a task or a set of tasks at different levels of abstraction and implementation. Multiple levels of abstraction makes it possible to control the speed, required modeling effort, and

This work is supported by the DARPA Power Aware Computing and Communication Program under contract F33615-C-00-1633 monitored by Wright Patterson Air Force Base and in part by the National Science Foundation under award No. 99000613.

accuracy of the simulations. In MILAN, hierarchical simulation is a two-level technique with HiPerE at the top level and the component specific simulators at the bottom level.

Hierarchical simulation is exploited to perform efficient design space exploration (DSE) in MILAN. DSE evaluates a design space (set of designs). In order to obtain a design space, the designer can either choose to evaluate all possible mappings (designs) of the application onto the architecture or use an optimization technique initially to identify a small set of candidate designs. Once the set of designs to be evaluated are selected, HiPerE is used to evaluate these designs and eliminate those that do not meet performance requirements. Later, more detailed simulations (cycle-accurate, RT-level, etc.) are used to evaluate the designs selected by HiPerE to identify the final design.

This paper is organized as follows. The next section discusses some related efforts. GenM is described in Section III. System-level performance estimation using HiPerE is discussed in Section IV. Section V describes hierarchical simulation for rapid design evaluation using MILAN. We conclude in Section VI.

II. Related Work

There have been several efforts to address application design and optimization targeting embedded systems in general and also specific to SoC architectures. Hybrid System Architecture Model (HySAM) model was proposed for latency evaluation and optimization of reconfigurable devices [3]. GenM model extends HySAM to include a processor supporting DVS and a memory with different power states. Givargis et al. proposed a technique for cache and bus power estimation based on sample simulations [5]. In this effort, low-level simulation at some design points are performed to derive a graph which is used to estimate performance of all possible designs. This technique cannot be scaled for heterogeneous SoC architecture as it is not possible to derive well-behaved functions based on sample simulation. Sinha et al. presented power and time estimation based on instruction level profiling [11]. This technique can only be applied to the ISA-based processors. There also exists several cycle-accurate simulators for processors to estimate performance [2] [10]. However, these simulators are time consuming and therefore cannot be used to traverse a large design space. Our effort is complementary as these simulators provide the component specific performance estimates used by HiPerE and hierarchical simulation.

There are several efforts for system-level estimation in the hardware-software codesign community [6] [7]. Integration of various component specific simulators using the Ptolemy framework was proposed by Lajolo et al. [7]. For on-chip communication architectures, an analysis methodology was presented by Lahiri et al. [6]. These efforts focus only on specific architecture such as RISC processor with cache and memory. Our methodology based on GenM model addresses a large class of SoC architectures.

SPADE is a hierarchical system-level simulation methodology [8]. SPADE uses trace-driven simulation and look-up tables of pre-computed estimates to derive system-level estimations. The applications are modeled using Kahn Process Network. Our methodology differs from SPADE in several aspects. Due to MILAN, our methodology can choose between a variety of application models to identify the one best-suited for the target application. In addition to the application model, we provide an explicit resource model to capture the capabilities of the target architecture. This facilitates identification of well-defined optimization problems. MILAN also provides a unified environment to integrate various simulators. This capability enables automatic estimation of performance of a task on a specific hardware to update the initial estimates in order to obtain better system-level estimates using HiPerE.

III. Generic Model for Application Mapping onto SoC Architectures

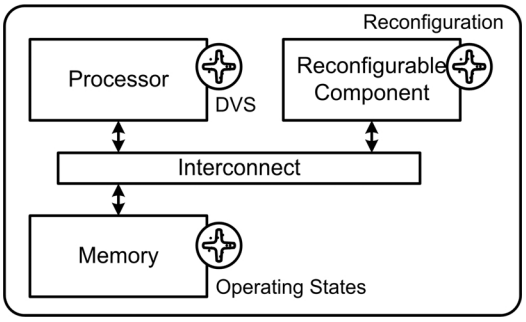

Fig. 1. Components of the GenM Model

The Generic Model (GenM) models SoC architectures to capture various capabilities that can be exploited for performance optimization during application mapping onto SoC architectures. It consists of three components: a processor, a reconfigurable logic (RL), and a memory, all connected through an interconnect (Figure 1). GenM models discrete dynamic voltage scaling (DVS) for the processor, power states for the memory, and reconfiguration for the RL. GenM model consists of the following parameters:

V : array of operating states of the processor, $[V_0 \dots V_{v-1}]$, where V_0 represents the idle state

$\hat{V}^t_{ij}(\hat{V}^e_{ij})$: time (energy) costs for voltage variation from V_i to V_j

C : array of operating states of the RL, $[C_0 \dots C_{c-1}]$, where C_0 represents the idle state

$\hat{C}^t_{ij}(\hat{C}^e_{ij})$: time (energy) costs for reconfiguration from C_i to C_j

$e^P_{id}(e^R_{id})$: processor (RL) energy dissipation in the idle state

M : array of memory states, $[M_0 \dots M_{m-1}]$

$\hat{M}^t_{ij}(\hat{M}^e_{ij})$: time (energy) costs for changing memory state from M_i to M_j

α_i : memory energy consumption per time unit in state M_i

η (ζ) : unit data transfer time (energy) between the processor/RL and the memory

In order to model application mapping onto SoC architectures, GenM model needs some additional parameters. These parameters capture few application details and provide performance estimates for various task to computing element mapping and are referred to as *Performance Parameters*. These parameters are:

T : set of tasks, $T_1 \dots T_n$

$\theta^i_{in}(\theta^i_{out})$: amount of data input (output) to (from) task T_i from (to) memory

$t_{ij}(e_{ij})$: time (energy) for executing task T_i on the processor in operating state V_j, $j = 1, \dots, v-1$

$t'_{ij}(e'_{ij})$: time (energy) for executing task T_i on the RL in operating state C_j, $j = 1, \dots, c-1$

Energy estimates (e_{ij} and e'_{ij}) refer to average energy dissipated by the task evaluated by averaging the estimates based on a set of input data. We also assume that when there is no task to execute, the processor is in the idle state. The idle state and the other states of the processor determined by specific operating voltages are referred to as the operating states of the processor. Similar assumption holds for the RL. The idle state of the RL and other states determined by specific configurations are referred to as the operating states of RL.

Let S denote the set of all possible system states. Each system state s, $s \in S$, is represented by a tuple ($I^v(s)$, $I^c(s)$, $I^m(s)$), where $I^v(s)$, $I^c(s)$, and $I^m(s)$ are integers and represent the operating states of the processor, RL, and memory respectively. We have $0 \leq I^v(s) \leq v-1$, $0 \leq I^c(s) \leq c-1$, and $0 \leq I^m(s) \leq m-1$. Therefore, the total number of distinct states is vcm. The transition between different system states incurs certain time and energy costs which depend on the source and destination states of the transition. Let q_{ij} (r_{ij}) denote the energy (time) costs for the system state for transition from s_i to s_j. q_{ij} is calculated as $\hat{V}^e_{I^v(s_i),I^v(s_j)} + \hat{C}^e_{I^c(s_i),I^c(s_j)} + \hat{M}^e_{I^m(s_i),I^m(s_j)}$. r_{ij} can also be calculated similarly. In the above formalism, we have assumed communication between tasks is performed via shared memory. However, this assumption is a limitation of the model. It is not necessary to define the problems.

GenM can be used for (a) rapid estimation of performance in terms of energy and latency, (b) development of efficient application designs using a high-level abstraction that ignores specific low-level details of the target SoC architectures and applications, and (c) development of optimization techniques for mapping and scheduling applications onto SoC architecture.

In the following, we provide an example optimization problem defined using GenM. This problem can be solved using dynamic programming [12]. The problem is to *find a mapping of a linear pipeline of tasks to system states, such that the overall energy consumption of the system is minimized.* For the sake of illustration of the model in formulating this problem, we assume that there is only one memory state, M_0, available and there is only one instance of the pipeline to be executed at any time.

We now provide a formal definition of the problem based on the parameters of the GenM model. For a linear pipeline of tasks, $T_1, \dots, T_n$, let the tasks be executed in linear order i.e. task T_i must be executed before T_{i+1}, $i = 1, 2, \dots, n-1$. Therefore, at any time, only one task can be executed by the system, either on the processor or RL. Consequently, for any system state s, either $I^v(s) = 0$ or $I^c(s) = 0$, but not both. Thus, the total number of possible system states in this particular problem is $v + c - 2$. For a given mapping, let x_i denote the system state in which task T_i is executed. Let E_{i,x_i} denote the total energy consumption for executing T_i. E_{i,x_i} is the sum of the energy consumed by:

1. data input from memory to the computing element where T_i is mapped
2. execution of T_i on the processor or RL
3. the processor or RL in the idle state
4. the memory during the execution of T_i
5. data movement to memory to store the output of T_i

The above five components can be calculated as

follows:

1. $\theta^i_{in} \cdot \zeta$
2. $e_{i,I^v(x_i)}$ if $I^v(x_i) \neq 0$, or $e'_{i,I^c(x_i)}$, otherwise
3. $t_{i,I^v(x_i)} \cdot e^R_{id}$ if $I^v(x_i) \neq 0$, or $t'_{i,I^c(x_i)} \cdot e^P_{id}$, otherwise
4. $t_{i,I^v(x_i)} \cdot \alpha_0$ if $I^v(x_i) \neq 0$, or $t'_{i,I^c(x_i)} \cdot \alpha_0$, otherwise
5. $\theta^i_{out} \cdot \zeta$

Let x_0 denote the initial system state. The overall system energy consumption is evaluated as:

$$E_{total} = \sum_{i=1}^{n} (E_{i,x_i} + q_{x_{i-1},x_i})$$

Because transitions between system states consume certain amount of time and energy that depend on the specific source and destination states, a simple greedy heuristic does not guarantee an optimal solution. On the contrary, dynamic programming can be used to find the optimal mapping in $O((v + c - 2)^2 \cdot n)$ time.

Several interesting extensions to the above problem can be defined using GenM for mapping applications onto SoC architectures [12]:

• The total execution time of the pipeline is to be minimized instead of the overall energy consumption of the system.

• The overall system energy consumption is to be minimized while the total execution time of the pipeline is subject to some pre-specified upper-bound.

• Multiple instances of the pipeline can be concurrently executed by the system. (This results in parallel execution of tasks on both the processor and RL.)

In the next section, we describe how we have exploited the GenM model to design the High-level Performance Estimator (HiPerE).

IV. System-Level Performance Estimation

As discussed earlier, one of the major challenges in system-level performance estimation is lack of standard interface among the component specific simulators which makes it difficult to integrates the simulators to simulate a SoC architecture. HiPerE addresses this issue by combining component specific performance estimates through interpretive simulation to derive system-level performance values.

The GenM model describing the target SoC architecture is the primary input to HiPerE. In addition, the performance parameters are also provided as an input. In our methodology, various optimizations may be performed before invoking HiPerE. In case an optimization is performed, a subset of designs identified by the optimization technique are evaluated by HiPerE. A designer can also choose not to perform any optimization and apply a brute force technique to evaluate each possible design exploiting the rapid estimation capability of HiPerE.

For performance estimation of a given design, HiPerE needs the mapping (specified by the design). Mapping identifies the computing element a task is mapped to and provides the operating voltage (or configuration) if the element is the processor (or the RL). HiPerE uses the mapping information to identify the appropriate component specific estimates (t_{ij} or t'_{ij} for latency and e_{ij} or e'_{ij} for energy). The designer provides initial values for all the performance parameters. Later, component specific performance estimation is used to estimate more accurate values for t_{ij}, t'_{ij}, e_{ij}, and e'_{ij}. In addition to these inputs, the application task graph which captures dependency among tasks is also provided. The task graph provides the order of execution (using topological sort) for the tasks. For the memory component, the designer provides a schedule of power states. Currently, we support change of power state for the memory only at the task boundaries.

The output of HiPerE is system-level energy and latency estimates. Along with these estimations, HiPerE also generates an activity report for each component in the target architecture. An activity report identifies various voltage settings, configurations, and power states for the processor, RL, and the memory component respectively during the course of execution. It also provides the duration of idle time (if any) between execution of tasks for the processor and the reconfigurable component.

A. Component Specific Performance Estimation

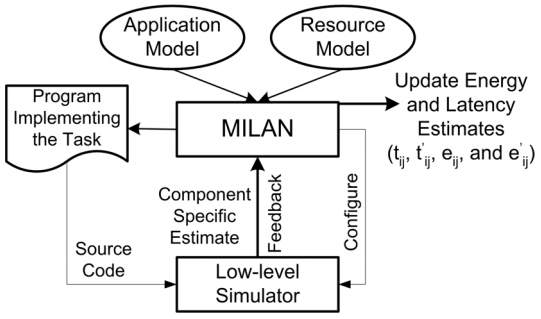

Fig. 2. Component Specific Performance Estimation using MILAN

Component specific performance estimation refers to the evaluation of performance parameters t_{ij}, e_{ij}, t'_{ij},and e'_{ij} specific to a task in a particular voltage setting or configuration. There are several techniques to estimate component specific performance values such as *Complexity Analysis, Graph Interpolation [5], Trace Analysis* [11] [8], and *Cycle-accurate Simulation* [2] [10]. While complexity analysis does not require a simulator, all the other techniques use a simulator

based on an architecture model at an appropriate level of abstraction.

We exploit the isolated simulation feature of the MILAN framework to perform component specific simulation. This feature refers to the ability to simulate a single application task on a specific hardware component. The resulting performance estimates are used to automatically update the performance parameters. The MILAN framework features an application model, a resource model, and a mapping model [9]. The resource model consists of all the parameters of the GenM model and various additional parameters that are used to drive simulators. The application model captures the application details as a data flow graph. Performance parameters, T_i, θ^i_{in}, and θ^i_{out} are also part of the application model. Other performance parameters are part of the mapping model. The designer also provides implementation of each task, for example, in C or Java.

Once a task has been selected for isolated simulation, based on the computing element it is mapped to, MILAN generates an appropriate simulator-configuration file and a source file (in a high-level language) that implements the task. While modeling the application, the designer provides source and destination scripts for each task that generate input for the task and consume output from the task. These two scripts are used by MILAN during the generation of a program that implements the task. For example, if FFT is a task mapped onto a MIPS processor and SimpleScalar is the chosen simulator, MILAN generates a C code implementing FFT and a SimpleScalar configuration file. After the simulation is performed, the performance estimate is provided as a feedback to MILAN which is used to update the initial performance estimates provided by the designer.

Component specific performance estimation is used to improve the accuracy of the initial estimates provided in the GenM model. We assume that when a designer provides a GenM model for a specific problem the performance estimates (initial values) are also provided.

Before moving to system-level performance estimation, we derive composite performance estimate for each task. Composite performance estimate includes all the set-up cost for task execution including the cost of execution. This estimate includes cost for execution, data access, memory activation, and reconfiguration or voltage variation. For example, assume that task T_i is mapped onto the RL with configuration C_j and C_k be the previous configuration. If we assume that no memory power state transition occurred, the composite latency performance (ΓC_i) can be evaluated as:

$$\Gamma C_i = t'_{ij} + \left(\theta^i_{in} + \theta^i_{out}\right) \cdot \eta + \hat{C}^t_{jk}$$

Similar composite estimates (EC_i) is derived for energy dissipation of task T_i. In the following subsection, the component specific performance estimate of a task refers to the composite performance estimate for that task.

B. System-Level Performance Estimation

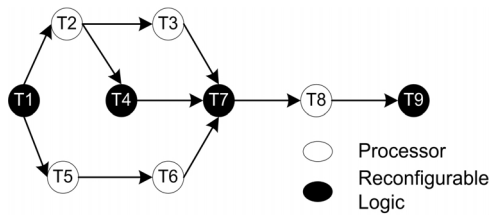

Fig. 3. A Sample Task Graph

We employ the following technique to evaluate system-level energy and latency and to generate the activity report. Figure 3 shows a sample task graph with tasks mapped on to either the processor or the RL. Our technique is as follows:

Let MP_i denote the mapping (given as input) for task T_i, where $MP_i = 1$ when T_i is mapped on the processor, and $MP_i = 2$ when T_i is mapped on the RL. MP_i can be obtained using the techniques in [12]. However, MP_i is provided as an input to system-level performance estimation. Let α denote the list of tasks, $\{T_{\pi_1}, T_{\pi_2}, \cdots, T_{\pi_n}\}$ obtained by a topological sort of the original task graph. Let A_1 and A_2 denote the earliest available time for executing a task on the processor and RL, respectively. Initially $A_1 = A_2 = 0$. Let t_{π_i} denote the completion time of task T_{π_i}. In the following algorithm, the earliest start time is calculated for each task without violating the dependency information provided in the application task graph. It is also assumed that a non-preemptive scheduling policy is used by each computing element. The pseudo-code for the algorithm is provided below.

for $k \leftarrow 1$ to n **do**
  Let β be the set of immediate predecessors of T_{π_k}
  The earliest start time for T_{π_k} is
    $\tau = max\{max_{T_i \in \beta}\{t_i\}, A_{MP_{\pi_k}}\}$
  Set $t_{\pi_k} = \tau + \Gamma C_{\pi_k}$ and $A_{MP_{\pi_k}} = t_{\pi_k}$

As a result, $max\{A_1, A_2\}$ is the system-level latency. The idle time of the processor is calculated as $IT_1 = A_1 - \sum_{MP_i=1} \Gamma C_i$. Therefore, the idle energy dissipation of the processor is $IE_1 = IT_1 \cdot e^P_{id}$. Similarly, the idle time and energy dissipation of the

RL, IT_2 and IE_2, are also calculated. Total energy dissipation is the sum of individual component specific energy dissipation and energy dissipation when the component is in the idle state. Thus total energy is evaluated as:

$$\sum_{i=1}^{n} EC_i + IE_1 + IE_2$$

Clearly, the complexity of the above method is $\Theta(N+Q)$, where N and Q are the numbers of nodes and edges in the task graph, respectively.

The activity report is generated based on the processed task graph with the mapping information and the time of completion for each task. The designer can exploit the activity report to identify bottlenecks and optimization opportunities. One possible optimization is to take advantage of the idle time and use a lower DVS setting to execute a task slowly in order to save energy.

HiPerE is implemented using Java and can be run on both Unix and Windows platform. HiPerE is also integrated into the MILAN framework. Therefore, it is possible to automatically generate input for HiPerE and execute it to obtain the performance estimates. We are currently developing the feedback mechanism to automatically store the HiPerE output in the MILAN framework.

C. Illustrative Example

In order to illustrate the use of HiPerE, we modeled a signal processing application (AzimuthElevation) that evaluates the azimuth and elevation of an object within an image with respect to a reference coordinate (Figure 4). The application consists of two major components, Azimuth and Elevation. Azimuth consists of 6 tasks (shown in Figure 4). Elevation has the same task graph as Azimuth but differs only in the type of BandPassFilters used. In these examples, we show the use of component specific estimations to derive system-level estimates, the accuracy of HiPerE estimation, and the effect of hierarchical simulation. Details of hierarchical simulation will be discussed in the next section.

We conducted experiments with two different architectures, a MIPS processor operating at a frequency of 600 MHz and a StrongARM (SA) processor operating at three different frequencies 206, 162, and 59 MHz. We considered four different designs. The schedule was the same for all designs and was evaluated using topological sort. The four different mappings were the complete system mapped onto a MIPS operating at 600 MHz or a SA operating at 206, 162, or 59 MHz. The energy and latency estimates obtained

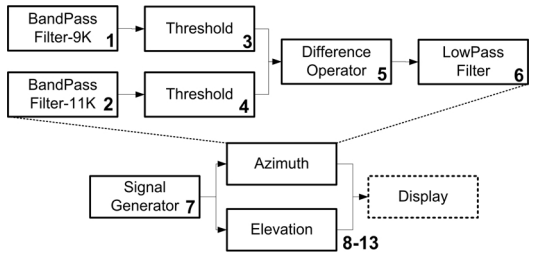

Fig. 4. Application Task Graph

using HiPerE were compared with the complete application simulation using low-level simulators. For low-level simulation we used SimpleScalar [10] and Wattch [2] for time and energy simulation for MIPS processor and JouleTrack [11] for simulation of the SA-1100 (StrongARM) processor.

HiPerE was given two sets of values for each mapping. For the values in the RE (Rough Estimates) columns for SA, we simulated a task at various operating frequency using JouleTrack and derived an equation that evaluates energy (or time) as a function of frequency. We used these equation to obtain estimates for a task at different frequencies. A similar technique has been used in [5]. We used a single equation to characterize all the four band pass filters used in the application. These equations are $Energy = 0.0227f^2 - 2.7422f + 312.93$ and $Time = 0.2695f^2 - 105f + 12102$ where f is the operating frequency. Similar technique was also used for SimpleScalar simulation. On the other hand, the values in the MILAN columns were obtained using the MILAN isolated simulation technique applied for each task. We describe part of the GenM model for mapping of AzimuthElevation in Table I. This table lists performance estimates (t_{ij}) for latency using the rough estimation technique. These values were later modified using the isolated simulation technique. The complete GenM model consists of estimates for latency and energy for all tasks for all the four different mappings.

TABLE I
PART OF THE GENM MODEL FOR AZIMUTHELEVATION

latency values	MIPS	SA	SA	SA
in mili seconds	@600	@206	@162	@59
BandPassFilter	0.22	2095	2312	6898
Threshold	0.02	94	120	330
LowPassFilter	0.44	2748	3030	9040
SignalGenerator	0.63	5613	6188	18393

Table II shows the estimation when the application

is executed on a MIPS processor. As expected, the estimation using the component specific simulation results in higher accuracy (error $< 3\%$). However, the error is high ($> 20\%$) when the estimates based on rough calculations are used.

TABLE II
COMPARISON OF HIPERE ESTIMATES WITH SIMPLESCALAR AND WATTCH

MIPS Processor	Complete Low-level Simulation	Estimation using HiPerE			
		RE	Error %	MILAN	Error %
Time (μs)	412	520	26	424	2.9
Energy (μJ)	17820	22408	25	18354	3

The results for the StrongARM processor are more interesting. We conducted experiments for three different frequency of operations. Using component specific simulation of each individual task the error is less than 7% for all the 6 estimates when compared with the value estimated through low-level simulations of the complete application. However, when we compared the HiPerE result through the use of rough-estimates the error varies from -3.2% to 11%.

TABLE III
COMPARISON OF HIPERE ESTIMATES WITH JOULETRACK

StrongARM Processor	Complete Low-level Simulation	Estimation using HiPerE			
		RE	Error %	MILAN	Error %
@206 (μs)	18228	20247	11	19350	6.1
@206 (μJ)	6432	6807	5.8	6868	6.7
@162 (μs)	23179	22462	-3.2	24609	5.8
@162 (μJ)	4046	4437	9.6	4321	6.8
@59 (μs)	63646	66721	4.6	67578	5.8
@59 (μJ)	2015	2199	9.1	2152	6.8

V. APPLICATION OPTIMIZATION USING MILAN

MILAN is a **M**odel based **I**ntegrated simu**LA**tio**N** framework for embedded system design and optimization [1]. The designer formally models the target application, underlying hardware, and constraints (latency, throughput, energy, etc.) through a graphical interface provided by MILAN. 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. Model interpreters are used to translate the models for use in the system's execution environment.

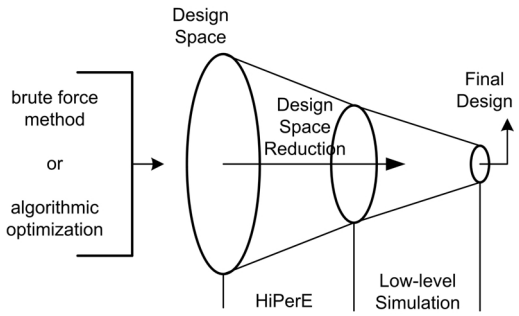

Fig. 5. Hierarchical Simulation for DSE in MILAN

Hierarchical simulation is used to efficiently evaluate a design space. A design space represents a set of designs enumerated by different values of parameters such as configuration, operating voltage, choice of implementation that are available in a typical SoC architecture. Evaluation of a design space (specially the large ones) requires efficient traversal of the space. Hierarchical simulation refers to performing simulation at multiple abstraction levels that makes it possible to control the speed, required modeling effort, and accuracy of the simulation results. Hierarchical simulation exploits the ability of HiPerE to provide quick system-level performance estimates. HiPerE constitutes the top level of hierarchical simulation (Figure 5). In MILAN, hierarchical simulation consists of two levels. The bottom level consists of all the component specific simulators. There are several simulators available for individual components (processor, reconfigurable logic, and memory) at different levels of abstraction. These simulators provide a trade-off between speed an accuracy. During DSE, the designs are evaluated against the performance constraints provided by the designer. DSE based on hierarchical simulation is as follows:

- The designer models (in a graphical interface) the application and the target architecture using the application, resource, and mapping model (available in MILAN). These models capture a design space enumerating all possible designs. Initially, the designer provides a rough estimate of various parameters (specifically performance values t_{ij}, t'_{ij}, e_{ij}, and e'_{ij}) described in the GenM model.
- The designer uses one or more optimization technique(s) and identifies a set of designs as the output of the optimization. Let these designs be the initial set of designs.

or

The designer chooses to use a brute force method and uses all possible designs as the initial set of designs.

• HiPerE provides an initial system-level performance estimate based on the initial values of component specific performance estimates and the activity reports for each design in the initial set. These results are used to select a set of candidate designs from the design space based on the performance constraints.
• The designer can choose to perform component specific estimation for some tasks in order to obtain more accurate estimates and execute HiPerE again. This results in more accurate estimation of system-level performance which can be used to refine the initial selection by HiPerE.
• In the final step, the designer identifies simulators with appropriate speed and accuracy to evaluate the designs selected by HiPerE. In case there is no single simulator for the complete SoC architecture, HiPerE can be used to integrate the results from individual simulators. In hierarchical simulation, the final step uses the most accurate simulators available and identifies the final design based on the simulation result and performance constraints.

An illustrative example of hierarchical simulation is shown in section IV.C. The RE columns in Tables III and II are based on offline estimates for the tasks in AzimuthElevation application. Error in the estimates by HiPerE is between $3.2 - 11\%$ for StrongARM and $> 20\%$ for MIPS processor. The values in the MILAN column are the result of component specific performance estimation for each task. Therefore the accuracy of estimates using HiPerE goes up for StrongARM (error $\pm 7\%$) and MIPS processor (error $+3\%$). The comparison of both these estimates are with the result of low-level simulation of the complete task using SimpleScalar and JouleTrack.

VI. Conclusion

We presented a high-level abstraction of SoC architectures, GenM model, and rapid system-level performance estimation based on GenM. HiPerE is a high-level interpreter which is used for design space exploration in MILAN using hierarchical simulation suitable for efficient application mapping onto SoC architectures. The GenM model provides abstraction to derive well-defined optimization problems.

Rapid performance estimation at system-level has wide range of applications. We provided a hierarchical approach to DSE by initially using DESERT, an ordered binary decision diagrams based symbolic search tool, to rapidly prune the design space, followed by the use of HiPerE, and finally by the use of low-level simulators to select the final design [9]. We are also investigating detailed modeling of FPGA for energy efficient algorithm design and optimization to compliment the effort (application optimization) discussed in this paper. In this effort, we propose to generate parameterized design with performance as a function of the design parameters [4]. Such a design will be used for application mapping onto SoC architectures.

Acknowledgement: We would like to thank Yang Yu and Jingzhao Ou for their valuable input in defining the GenM model and identification of optimization problems based on the GenM model.

References

[1] Agrawal A., Bakshi A., Davis J., Eames B., Ledeczi A., Mohanty S., Mathur V., Neema S., Nordstrom G., Prasanna V., Raghavendra C., Singh M., "MILAN: A Model Based Integrated Simulation for Design of Embedded Systems," Language Compilers and Tools for Embedded Systems, 2001.

[2] Brooks D., Tiwari V., and Martonosi M., "Wattch: A framework for architectural-level power analysis and optimizations," 27th Intl. Symposium on Computer Architecture, 2000.

[3] Bondalapati K. and Prasanna V. K., "DRIVE: An Interpretive Simulation and Visualization Environment for Dynamically Reconfigurable Systems," Intl. Workshop on Field Programmable Logic and Applications, 1999.

[4] Choi S., Mohanty S., Jang J., and Prasanna V. K., "Domain-Specific Modeling for Rapid System-Level Energy Estimation of Reconfigurable Architectures," Intl. Conf. on Engineering of Reconfigurable Systems and Algorithms, 2002.

[5] Givargis T. D., Vahid F., and Henkel J., "Evaluating Power Consumption of Parameterized Cache and Bus Architectures in System-on-a-chip Designs," IEEE Transactions on Very Large Scale Integration Systems, August 2001.

[6] Lahiri K., Raghunathan A., and Dey S., "System-level Performance Analysis for Designing On-Chip Communication Architectures," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, June 2001.

[7] Lajolo M., Raghunathan A., Dey S., and Lavagno L., "Efficient Power Co-estimation Techniques for System-on-Chip Design," Design, Automation & Test in Europe, 2000.

[8] Lieverse P., Wolf P., Deprettere E., and Vissers K., "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems," Journal of VLSI Signal Processing for Signal, Image and Video Technology, November 2001.

[9] Mohanty S., Prasanna V. K., Neema S., and Davis J., "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation," Language Compilers and Tools for Embedded Systems, 2002.

[10] SimpleScalar, http://www.simplescalar.com.

[11] Sinha A. and Chandrakasan A. P., "JouleTrack-A Web Based Tool For Software Energy Profiling," Design Automation Conference, 2001.

[12] Yu Y., Ou J., Mohanty S., and Prasanna V. K., "Energy Efficient Mapping of Applications onto SoC Architectures," in preparation, Technical Report, Dept. of Electrical Engg., University of Southern California, 2002.