Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation*

S. Mohanty, V. K. Prasanna
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089
[smohanty, prasanna]@usc.edu

S. Neema, J. Davis
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN 37235
[sandeep.k.neema, james.r.davis]
@vanderbilt.edu

ABSTRACT

In addition to integrating different Intellectual Property cores, heterogeneous embedded systems provide several architecture knobs such as voltage, operating frequency, configuration, etc. that can be varied to optimize performance. Such flexibilities results in a large design space making system optimization a very challenging task. Moreover, such systems operate in mobile and other power constrained environments. Therefore, in addition to rapid exploration of a large design space a designer has to optimize both time and energy performance. To address these issues, we propose a hierarchical design space exploration methodology. Our methodology initially uses symbolic constraint satisfaction to rapidly prune the design space. This pruning process is followed by a system wide performance estimation to further reduce the number of candidate designs. Finally, detailed simulation using low-level simulators are performed to select an appropriate design. Our methodology is implemented by integrating two tools, DESERT and HiPerE, into the **M**odel based **I**ntegrated simu**LA**tio**N** (MILAN)[1] framework. DESERT uses Ordered Binary Decision Diagrams based symbolic search to rapidly explore a large design space and identifies candidate designs that meet the user specified performance constraints. HiPerE provides rapid estimation of system wide energy and latency based on component level simulations and also facilitates energy optimization. MILAN provides the required modeling support for these tools and also facilitates component specific multi-granular simulations through seamless integration of various simulators.

*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.

[1]milan (*Hindi*): unification, bringing together

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
LCTES'02–SCOPES'02, June 19-21, 2002, Berlin, Germany.
Copyright 2002 ACM 1-58113-527-0/02/0006 ...$5.00.

Categories and Subject Descriptors

I.6.0 [**Simulation and Modeling**]: Design Space Exploration - *constraint based design exploration, multi-granular simulation, rapid performance estimation*

General Terms

Performance, Design

Keywords

Design Space, Performance Estimation, Binary Decision Diagram, Symbolic Search, Modeling, Model Integrated Computing, Multi-granular Simulation

1. INTRODUCTION

Design space exploration is the process of analyzing several "functionally equivalent" implementation alternatives to identify an optimal solution. In the *traditional design process* the designer often starts with an informal specification and develops a reference executable in some high-level language such as VHDL, Verilog, or C, typically referred to as the reference model. This model is then verified for functional correctness according to the system specification and is used to get rough estimates of its performance requirements. This initial step is followed by manual or semi-automatic generation of several alternative designs which are subjected to a series of time-consuming and typically ad hoc evaluations. Finally the most suitable design is chosen based on various performance metrics such as performance, cost, power, reliability, and flexibility.

Embedded applications consisting of stream-based data processing of various types (e.g., text, speech, image, music, video) are widely used in portable devices such as wireless phones, personal digital assistants, laptops, etc. High throughput and stringent real time requirements are especially important in such data-intensive tasks demanding high processing power. On the other hand, energy efficiency is as important as throughput for portable devices as they operate in power constrained environments (batteries, and solar power, etc.). Therefore, a balanced architecture which consumes minimum energy and provides the required performance (or throughput) is the most desirable.

System-on-Chip (SoC) architectures are complex chips that

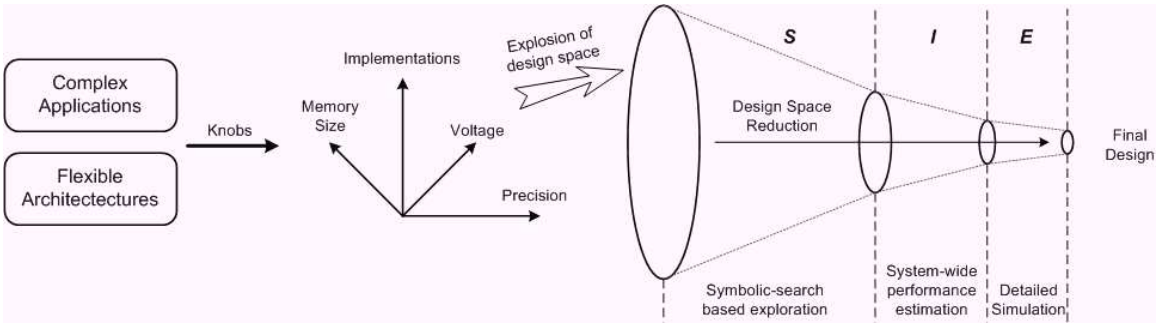

Figure 1: Hierarchical Design Space Exploration

integrate all major functional elements of a complete end product onto a single chip. Such functional elements are realized on a SoC as intellectual property (IP) blocks. IP blocks include general purpose processors, memories, application-specific processors, and customized logic, among others. Xilinx Virtex II Pro, Chameleon Systems Reconfigurable Communication Processor, and Trimedia VLIW Processors from Trimedia Technologies are some such architectures. For example, the Virtex II Pro embeds up to four Power PC processors within a reconfigurable fabric. These state-of-the-art architectures provide several novel features such as variable operating voltage and frequency, reconfigurability, different operational states, variable number of active memory banks, etc. Such flexibility and a wide choice of processing elements make SoC architectures a suitable platform for implementing complex embedded systems. However, the design space, that must be explored to arrive at a suitable design, has grown enormously due to large number of architecture knobs (Figure 1).

The drawback of the traditional design process becomes obvious as it is applied to these state-of-the-art architectures. For example, suppose that an application of size n tasks is to be mapped onto an architecture with k processing elements. The number of possible mappings can be computed using Stirling's number of the second kind, $S(n,m)$, which denotes the number of ways to partition the set of n elements into m non-empty sets [3] where,

$$S(n,m) = \frac{1}{m!}\sum_{i=0}^{m-1} -1^i \left(\begin{array}{c} m \\ i \end{array} \right) (m-i)^n$$

Let us assume that each of the k processing modules can have on average p number of different states (settings of architecture knobs). For example, if the StrongARM processor can be operated at four different frequencies, it has four different states. Further, it is possible to use "some" or "all" of the computing element available on a heterogeneous system for an implementation. Thus the number of possible solutions (ways an application can be mapped) is given by:

$$N_{mapping}(n,k,p) = \sum_{x=1}^{k} p^x S(n,x)$$

Applying this formula, a design with 4 tasks on an architecture with 3 processing modules and each having 4 possible configurations can result in 500 designs. This can prove to be a very large number even for the most efficient simulator if one is to use system simulation to evaluate all possible designs.

A common approach in traditional design techniques is to use detailed simulation for design space exploration. While several component specific simulators are available, it is not easy to integrate these simulators to develop a detailed system level simulator for a typical multi-component heterogeneous embedded system. Moreover, it is impractical to use these simulators to evaluate a large number of design choices (even in the order of 10s) due to prohibitively high simulation time.

Thus the exponential growth in the design space, increasing complexity of applications, and lack of rapid system level evaluation tool demand a novel methodology to address these issues. This methodology should be able to rapidly analyze a large number designs and evaluate system wide performance in terms of energy and time performance at multiple levels of abstraction. Our approach (Figure 1) to this end can be summarized as follows:

- *(step S)* apply various user specified design and performance constraints to rapidly eliminate the designs that do not meet the given constraints
- *(step I)* eliminate sub-optimal designs (from the remaining) based on system wide latency and energy estimates
- *(if required)* iterate through above two steps by updating component specific performance estimates through more accurate simulations
- *(step E)* perform low-level simulations on the remaining designs to identify the design that meets all the performance constraints

In this paper, we present integration of two design tools DESERT and HiPerE into the MILAN[2] framework which in conjunction with the integrated simulators provide the necessary environment for efficient design space exploration.

DEsign **S**pace **E**xplo**R**ation **T**ool (DESERT) is a domain-independent constraint-based rapid design space exploration tool, developed in prior research efforts at ISIS, Vanderbilt University [6]. It uses symbolic methods based on Ordered Binary Decision Diagrams (OBDDs) for constraint satisfaction [9]. The highlight of the symbolic constraint satisfaction method is the ability to apply constraints to the entire space

[2]http://milan.usc.edu

without enumerating point solutions, whereas an exhaustive search by enumeration through the space generally exhibits exponential time complexity. Symbolic analysis methods represent the problem domain implicitly as mathematical formulae and the operations over the domain are performed by symbolic manipulation of mathematical formulae.

High-level **Per**formance **E**stimator (HiPerE) is a rapid high-level performance estimation tool which, given a mapping, schedule of execution, and component specific performance for computation, communication, and storage elements evaluates the system wide performance values taking into account available parallelism, idle time energy dissipation, and state transition and and reconfiguration overheads. HiPerE also facilitates energy optimization by exploiting various low-power operating states available in the various components of a heterogeneous architecture. Details of HiPerE can be found in [17].

Per our prior analysis using Sterling's number, even a high-level performance estimator will be unable to analyze the large number of design spaces (which can go up to the order of 10^n, where $n \gg 10$) efficiently. Thus the combination of *symbolic search* and a high-level performance estimator provides an advantage. While the symbolic search tool rapidly applies constraints to eliminate invalid designs, the HiPerE tool evaluates the remaining designs based on the system wide time and energy performance. HiPerE can also be used along with any optimization mechanism or tool to verify the system wide effects of various optimizations.

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]. The designer formally models the target application, underlying hardware, and constraints (latency, throughput, energy dissipation, etc.) through a graphical interface provided by MILAN. The models are stored in a model database. The model information is translated through model interpreters into suitable input formats required by the integrated simulators. MILAN adopts Model Integrated Computing (MIC) as the core design technology [25]. The Generic Modeling Environment (GME) is a configurable graphical tool suite supporting MIC [11]. GME allows the designer to create domain-specific models at the required level of abstraction [14]. A *metamodel* (modeling paradigm) is a formal description of model construction semantics. Once the metamodel is specified by the user, it can be used to configure GME itself to present a modeling environment specific to the problem domain. Every target system is specified in MILAN as a model. Model interpreters are the software components that translate the information captured in models as per the requirement of the MILAN execution environment. Both DESERT and HiPerE are domain independent tools. The target system is described using domain specific modeling paradigm and are translated into inputs specific to each tool through model interpreters.

The paper is organized as follows. Next section discusses some of the related efforts. Various MILAN models are described in Section 3. An overview of the design space exploration and brief descriptions of the DESERT and HiPerE tool are provided in Section 4. The MILAN framework and its ability to integrate various simulators and tools is discussed in Section 5. We conclude in Section 6.

2. RELATED WORK

Several research efforts have addressed the issue of design space exploration and performance analysis of the embedded systems. The CODEF tool allows design space exploration based on a complete specification of partitioning/scheduling and interconnection synthesis [2]. The focus is on time and area constraints, and primarily targets the design of dedicated systems. Baghdadi et al. proposed a design space exploration framework based on a fast and accurate estimation approach primarily targeted towards homogeneous multiprocessors with fixed architecture [3]. This effort makes use of a novel idea of being able to perform a few cycle-accurate simulations to extracts *all* timing constraints necessary for *all* possible implementations. All these efforts target specific non-programmable architectures and focus only on time optimization. Thus they may not be suitable for energy optimal application implementation using the state-of-the-art programmable heterogeneous architectures.

Other system level design frameworks such as PMOSS and COSMYA target uniprocessor systems [10, 12]. PMOSS targets evaluation of speed-up due to a coprocessor. COSMYA calculates separate metrics for software, hardware, and communication based on profiling and low-level simulation to analyze performance. POLIS combines high-level and low-level simulation but targets only a specific architecture of one microprocessor accompanied by several custom coprocessors [5]. These efforts target certain class of embedded systems such as Application Specific Integrated Circuits and micro-controllers. The MILAN projects is an effort to combine some of the techniques developed in the above mentioned researches into one framework targeted towards design and optimization using heterogeneous embedded systems. Energy and time are the primary consideration during optimal application design using MILAN [4].

The Artemis project and related SPADE methodology address the issue of evaluation of embedded system architectures at multiple abstraction levels [15, 21]. However, there are some key differences. Artemis project proposes application modeling based on Kahn Process Network and a large library of architecture models at different granularity, which can be used to exploit time versus accuracy tradeoff. Design space exploration in the context of ARTEMIS means to be able to quickly evaluate a large number of designs using coarse-level architecture models and moving towards more accurate modeling as the number of designs reduces. While our approach has a similar concept, we draw a clear boundary between rapid analytical search and multi-level performance estimation. During analytical search we do not evaluate system wide performance *per se*, but rather eliminate designs based on explicit user level performance constraints. The multi-granular simulation follows this step. Further, due to MIC based design, a user can easily add new application and architecture modeling paradigms, thus making MILAN a more generic solution. MILAN also supports a plug-n-play environment that facilitates integration of any design tools such as DESERT or HiPerE (discussed in Section 5).

3. MODELING THE DESIGN SPACE

MILAN uses a representation method that partitions the system into three distinct classes of models: application models, resource models, and constraints. Application mod-

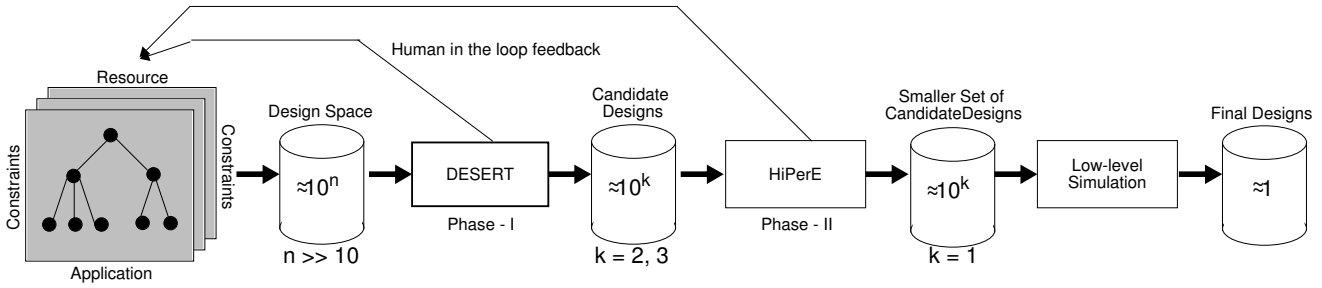

Figure 2: Three phase Design Space Exploration in MILAN

els describe the task to be performed while the resource models describe the available physical hardware. Constraints specify performance and power requirements the system must meet. In addition, a mapping model facilitates mapping between application tasks and the processing elements within resource models to represent the possible resource assignment choices.

Application models are currently constructed using enhanced data flow diagrams. Some of the key extensions to the typical data flow diagrams include:

- **hierarchy** to help handle system complexity,
- both **asynchronous** and **synchronous data flow**, as well as their composition,
- strongly **typed** data flow,
- modeling application functionality that is to be implemented in **configurable hardware**, and
- explicit design and implementation **alternatives** to capture the design space of the application as opposed to a point solution.

While the designer can model an application exploiting the hierarchical nature of our data flow models, the source code scripts are provided only at the lowest levels of the hierarchy. This allows higher level functionalities to be described by composing algorithms using lower level components.

Asynchronous data flow components are written to a specific run time kernel. Currently, MILAN supports a Java real time kernel and the Active kernel [7]. Synchronous data flow components are not written to a specific interface - the model interpreters generates a static schedule for synchronous data flow components. It is also possible to generate a functional description of the application in high-level languages such as C, Java, or MATLAB using model interpreters associated with MILAN. In some cases, the system may require a mixture of asynchronous and synchronous data flow models. While these cases are handled, their discussion is beyond the scope of this paper.

The MILAN application model also *isolated simulation*. This feature refers to the ability to simulate a single application task on a specific hardware component to perform a component specific simulation and update the performance estimates used by DESERT and HiPerE. The user can provide a high level script for a data flow model that enables simulators, during isolated simulations, to make use of the high level script instead of the detailed composition of all the components. These high-level scripts are associated with the predecessors and the successors of the task that needs to be simulated. Further details regarding isolated simulations can be found in [16].

Another important data flow extension involves the use of alternatives. Alternatives are used to represent different solutions to the same problem. Depending on other system details, one solution may be better than others. Using alternatives allows the user to specify the design space of their application. This design space encapsulates the different design decisions for each application task that must be evaluated to identify the best design.

Design constraints capture system requirements such as timing, performance, power, cost, etc. Moreover, resource constraints and other information also need to be specified as part of the models. Resource constraints specify the rules for resource composition out of hardware building blocks. The Object Constraint Language (OCL) is used to specify constraints in a formal manner. These constraints are specified in the application models and are used by the DESERT tool in evaluating which system instances meet the system specifications. Using OCL ensures we make use of standard languages whenever possible in MILAN. This helps keep the modeling language as natural for the end user as possible.

Resource models capture the details of the hardware components in the target system. The modeling language is based on a hierarchical classification of the hardware components typically found in the embedded systems. Resource model has two aspects, structural and parametric. The structural aspect describes the composition of an architecture through modeling entities representing the class of components. At the highest level of abstraction they are divided into computational, storage, and communication elements. Several specialized classes of components are also defined within each of these categories [1].

The parametric aspect captures programmable/variable architecture parameters such as operating voltage, frequency, cache structure, number of memory banks, input and output ports, etc. Further, the various operation states possible for a component is also captured in the resource model. Other information related to operation states such as state-transition cost, idle-stage energy dissipation, expected throughput, etc, are also part of the resource modeling paradigm. These information are used by HiPerE during system wide performance estimation and energy optimization.

Finally, the user must model which data flow components can be implemented on which available resources. MILAN provides a mapping model to associate data flow components with the resources. A single data flow component can

be mapped to several resources to represent various choice of implementation. That is, the selected data flow component can be implemented on any of the resources, but only one of the resources for an instance of the system. Additionally, performance and power estimates are also provided in the mapping model. These attributes represent the performance and power characteristics of the data flow component executing on the mapped resource. This information is also used by DESERT in determining which candidate designs meet the system performance constraints and HiPerE to evaluate system performance. Mapping model also determines the communication channel to be used for the data flow between different task components. The communication channel determines the latency and energy associated with data transfer.

The separation of application, resource, and mapping models separates the analysis of the functional behavior and performance behavior of a system. The functional behavior of a system is captured in the application models and is independent of the underlying hardware. Therefore, only application model is enough to verify correctness through functional simulation using high-level languages such as MATLAB, C, and Java. Functional simulation can also be used to identify the schedule of execution (specifically for asynchronous data flow graphs), details of data flow between the components, etc. that are used during the high-level estimation using HiPerE (Section 4). Further, this separation allows independent composition and modification of the resource and application models. Also, once the schedule is known, analysis of performance can be accomplished without further evaluation of the functional behavior.

During simulations the resource model is used to drive the simulators while the application models provide the necessary high-level programs. For example, for simulations using SimpleScalar the resource model provides the MIPS architecture details such as cache configuration, pipeline details, etc. while the application model provides the "C" code.

The various models in the MILAN framework capture the information necessary to drive DESERT, HiPerE, and the low-level simulators. In the following section we discuss how these information are utilized to perform design space exploration using MILAN.

4. DESIGN SPACE EXPLORATION (DSE)

Design space exploration is performed in three phases in MILAN (Figure 2). The first phase is based on analytical techniques. This phase does not use estimation or simulation, rather uses mathematical formulation to rapidly evaluate an extremely large design space. The second phase is based on various estimation techniques based on simulation models at various abstraction levels. The third phase uses low-level detailed simulations. The models of any system in the MILAN framework capture the complete design space in terms of alternate implementations and range of values for architecture parameters. During modeling phase individual designs are not identified. The tool performing the first level DSE analyzes a design space and identifies a set of designs for the next phase. A design is a mapping of each application task onto a processing elements with specified values for each architecture parameter associated with that processing element. The second phase of DSE analyzes this set of designs through rapid system wide performance estimations. This phase does not deal with design space rather analyzes individual designs. However, one can perform further optimizations to any of these design based on the results provided by the second phase. The final phase uses low-level simulation tools to verify performance estimates and choose a single design if a set of design candidates are identified at the end of the second phase.

The three phase process described above facilitates a hierarchical design space exploration and exploits the speed versus accuracy tradeoff to gradually reduce the design space to arrive at an appropriate design. In the following, we explain how such an hierarchical DSE is realized by integrating DESERT, HiPerE, and various simulators into the MILAN framework.

4.1 DSE using DESERT and HiPerE

DESERT performs the *Phase-I* DSE and HiPerE performs the *Phase-II* DSE. *Phase-I* is design space exploration through constraint satisfaction. This phase identifies all of the potential designs that meet the system performance specifications while eliminating those designs that do not meet the constraints. Typically the constraints are tuned (relaxed or tightened) so that at the end of *Phase I* the designer should be left with $\approx$ 100 designs. The number of designs is chosen so that the rest of the process can be completed in "reasonable" time. The set of designs selected by *Phase-I* are evaluated based on system wide performance estimation in *Phase-II*. The key components in this phase are HiPerE and a set of component specific simulators. Finally the designs chosen by *Phase-II* are subjected to low-level simulations to identify the optimal design in *Phase-III*. This design process is shown in Figure 2.

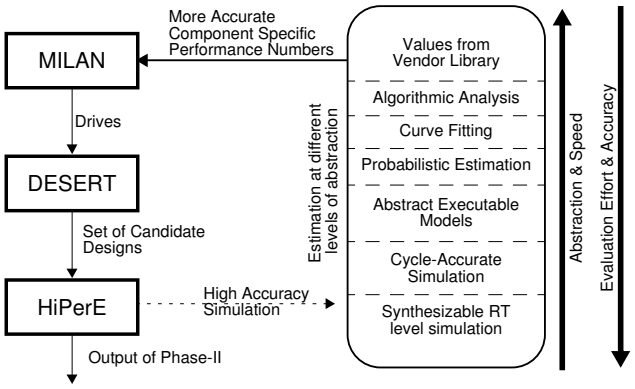

Figure 3: DSE using component specific simulation at multiple abstraction

Both DESERT and HiPerE use component specific performance estimates to perform design space exploration and system wide performance estimation respectively. Various choices to obtain component specific performance numbers is the key differentiator between different accuracy levels of design space exploration or performance evaluation (Figure 3). Perhaps the fastest way to obtain performance number is back-of-the-envelope calculation based on algorithm analysis. On the other hand the slowest but perhaps the most accurate result can be obtained using a simulator based on register-transfer level model of a computational element. MILAN framework supports isolated simulation so that a user can identify a single task and obtain a high-level code

which has source and sink script along with the script for the task under evaluation [16]. This can be executed on a detailed simulator to obtain performance values that can be used as a "better" input to HiPerE. Various other techniques which fall in between these two methods are curve fitting, probabilistic estimation, cycle accurate simulation, etc. (see Figure 3).

The design space exploration in both the phases exploit the availability of simulations or estimations at different levels of accuracy (Figure 3). Initially, both DESERT and HiPerE use the rough estimates of time and energy performance provided in the mapping model. Following the selection of a set of candidate designs in the second phase the designer can choose to run both DESERT and HiPerE again with more accurate estimates for some mappings obtained through component specific simulations. With every iteration the accuracy level of the component specific simulation is increased (or a new set of simulations are performed for a different set of mappings) and the number of designs to be evaluated is decreased by applying "more" strict performance constraints. Eventually the designer terminates the exploration if a pre-determined number of candidates are identified for final selection using low-level simulations.

4.2 OBDD based Design Space Exploration

This section elaborates upon Phase I of design-space exploration. The phase I uses DESERT, an interactive symbolic constraint satisfaction tool. DESERT treats the constraint satisfaction problem as a finite set manipulation problem, where the design space is a finite set and constraints specify relations within this set. Two key problems necessary for solving this finite set manipulation problem have been addressed: (a) symbolic representation of the space, and (b) symbolic representation of the constraints. We describe each one in brief below.

4.2.1 Symbolic Representation of the Design space

The key to exploiting the power of symbolic Boolean manipulation is to express a problem in a form where all of the objects are represented as Boolean functions [9]. By introducing a binary encoding of the elements in a finite set all operations involving the set and its subsets can be represented as Boolean functions. In order to represent the design space symbolically, the elements of the design space have to be encoded as binary vectors. The choice of encoding scheme has a strong impact on the scalability of the symbolic manipulation algorithms. An encoding scheme has been developed after a careful analysis of the problem domain, taking into consideration the hierarchical structure of the solution space [19]. The design space captures feasible configurations for implementing the system functionality, and is represented as a hierarchical data flow graph with alternatives, as described earlier. This representation can modularly define a very large space. The encoding scheme assigns encoding values to each node in the hierarchy such that each configuration receives a unique encoding value. Additionally, the encoding scheme must also encode the resource assignments of components along with performance attributes such as latency, throughput, power, etc. The performance attributes take numeric values from a continuous finite domain. However, for the purpose of encoding the domains of the attributes are discretized. The total number of binary variables required to encode the operational space is primarily dependent upon on the domain size and the quantization levels in the domain. With this encoding the design space is symbolically composed as a Boolean function from the symbolic Boolean representation of components. After deciding the variable ordering this Boolean representation is mapped to an OBDD representation in a straightforward manner.

4.2.2 Symbolic Representation of Constraints

The DESERT tool primarily addresses three basic categories of constraints. Symbolic representation of each of these categories of constraints is summarized below.

- *Compositional constraints* - Compositional constraints express logical relations between processing blocks in the hierarchical data flow representation. Symbolically the constraint can be represented as a logical relation over the OBDD representation of the processing blocks trivially.

- *Resource constraints* - Resource constraints relate processing blocks to resources. Symbolic representation of resource constraints is accomplished by expressing the relation over the OBDD representation of the processing block and resource.

- *Performance constraints* - Performance constraints are more challenging to solve symbolically than the previously specified categories of constraints. There are two primary drivers of the complexity: 1) A system level property has to be composed from component-level properties in a large design space, and 2) The property being composed is numeric, and may admit a potentially very large domain. Representing a large numeric domain symbolically as a Boolean function and performing arithmetic operations symbolically is a challenging problem with serious scalability concerns. In general different performance attributes compose differently. An approach for expressing constraints over additive attribute symbolically has been detailed in [19].

In addition to these basic categories of constraints, complex constraints may be expressed by combining one or more of these constraints with first order logic connectives. The symbolic representation of the complex constraints can be accomplished by composing the symbolic representation of the basic constraints.

The prominent features of the design space exploration tool include the ability to interactively and iteratively apply constraints. The effect of various constraints upon the design space can be visualized in this tool. The tool maintains multiple contexts and it is possible to revert to a previous context. Whenever constraints are applied and the design space is pruned a new context is created. The subsequent pruning is performed in this new context. To "undo" an applied constraint one can simply revert back to the previous context. The depth of the context stack is user programmable.

The scalability of the OBDD based design space exploration can be investigated from two different perspectives: a) the scalability of the symbolic representation, and b) the scalability of symbolic constraint application. Experimental

results indicated that the symbolic representation, characterized with the number of nodes in the OBDD representation of the design space is approximately O(*log*(design-space size)), which is highly scalable. This scalability is attributed to the density of the design space. While many encoding variables are required to represent the design space symbolically, owing to the density of the design space most are 'don't cares', resulting in a smaller compact representation. The direct implication of this high scalability is that the symbolic application of logical and relational constraints remains highly scalable, while the design space is under-constrained. The application of arithmetic, performance constraints on the other hand depends on the structure of the design space. Experimental results indicate that the symbolic application of arithmetic constraints scales linearly with the size of individual designs in the design space, however, it has near exponential growth with the number of designs in the design space. The scalability of the approach can be improved by carefully determining the order of constraint application. In practice good heuristics can be developed for choosing the order of constraint application. Further details of scalability studies can be found in [20].

The design space exploration tool has a multi-pane graphical front-end. The first pane is a checklist box that is filled up with all the constraints present in the model. There is a check box in front of every constraint in the list. The user can check the box to select the constraints to apply. More than one constraint can be selected. The second pane of the user interface shows the structural space as a tree. Different icons are used to distinguish between a compound (AND) node, a template (OR) node, and a primitive (LEAF) node. A box at the bottom of the pane displays the size of the structure space composed in the tree hierarchy. The third pane of the user interface shows the behavioral (mode) space also as a tree. The last pane of the user interface shows the resources in the model. The menu of the user interface has options for applying a selected set of constraint, applying all constraints, or reverting to a previous context. Further details about the DESERT tool can be found in [20].

4.3 Design Space Exploration using HiPerE

The designs that satisfy various constraints during the *Phase I* exploration are further subjected to evaluation based on system wide performance estimation using HiPerE. At this stage the designer can also exploit different low-power states of the processing elements and the storage devices to optimize energy dissipation. Details of HiPerE architecture and optimization techniques based on HiPerE can be found in [17].

4.3.1 Estimation Using HiPerE

HiPerE implements coarse-level trace-driven simulation. Trace-driven simulation is a popular technique used in several research efforts [8, 21]. The information required for trace-driven simulation are a trace-file, a mapping, resource details, and estimated performance values.

Mapping is provided as part of each design given to HiPerE. The details of the resource are obtained from the resource models. Estimated performance values refer to the energy and time estimates associated with each task mapped onto a processing element. Data access and transfer cost per unit data for each storage and routing element are also provided to HiPerE.

Trace file contains an ordered list of communication and computation operations. There are several techniques to generate a trace file. We use source code annotation at task level to generate the trace file. By task level, we mean that we capture only the information that a task is executed and certain amount of data is transfered from one task to another. A task is represented by a node at the lowest level of hierarchy in an application model. We annotate the source code (C, MATLAB, or Java) using three functions:

- *execute*: outputs the name of the task into the trace file,
- *read*: outputs the amount data transferred from the source task or storage element to destination task, and
- *write*: outputs the amount data transferred from the source task to destination task or storage element.

For example, let *matrix_mult* be a task corresponding to a 100×100 matrix multiplication using 32 bit integer values. Also, let the data sources be A and B and the data sink be C. The annotations associated with this task will output four lines to the trace file; a) read 400 bytes from A, b) read 400 bytes from B, c) execute *matrix_mult*, and d) store 400 bytes to C.

The trace-file is generated based on functional simulation using the annotated source files. While functional simulation is needed for application represented using asynchronous data flow models, for synchronous application, evaluation of partial order based on the application task graph is employed as a short-cut to compute the application schedule.

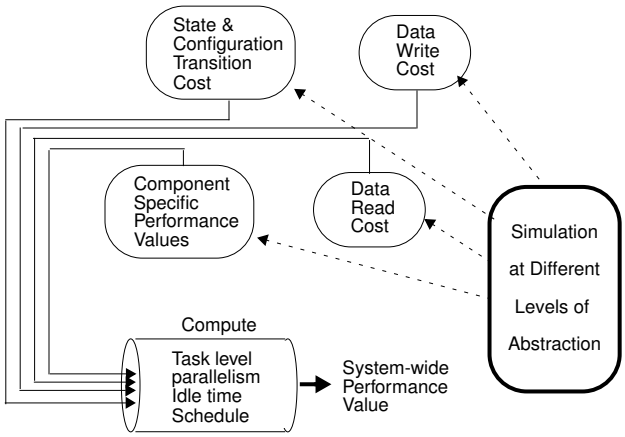

Figure 4: HiPerE Architecture

The estimation technique in HiPerE has two major stages; (a) component specific performance evaluation, and (b) system wide performance evaluation. The component specific performance evaluation is further categorized into three types of estimations: computation, storage, and communication cost estimation. Several techniques can be used to perform component level estimations. We primarily use existing analytical models or simulators for this process.

HiPerE evaluates the system wide performance values for energy and time based on the component specific values. It also takes into account the available parallelism among the

tasks, idle state energy dissipation, state transition and reconfiguration overheads, etc. during the evaluation process (Figure 4).

Figure 5: Activity Report

The system wide evaluation in HiPerE is based on the communication and computation details captured in the trace file. Based on the mapping information and performance estimations, HiPerE generates an activity list for each hardware element. An activity list provides the activity schedule associated with each hardware. For processing elements, it shows the duration of execution of different tasks mapped onto it. For storage elements, it captures the schedule and duration of each data access and for routing elements, it captures the data transfer details. Such activities are shown in shaded regions in Figure 5. The white areas in this figure indicate idle cycles. System-wide energy estimation is fairly straight forward. It is the sum of energy consumed by each resource elements. However, the factor that has to be added is the idle-time energy dissipation (white areas in Figure 5). This information is derived based on the activity report for each processing element. Detailed description of the technique employed by HiPerE to evaluate time and energy can be found in [17].

4.3.2 DSE Using HiPerE

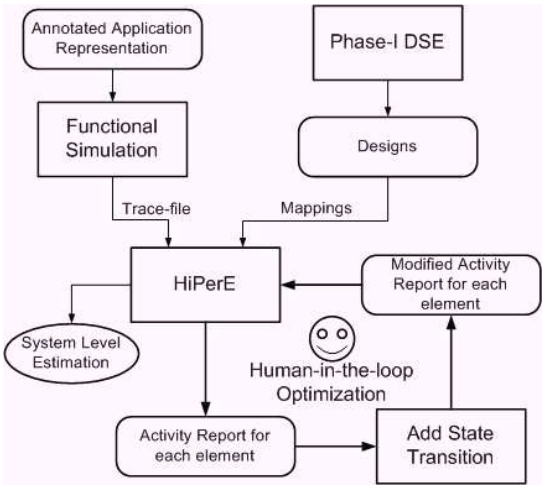

Figure 6: DSE and optimization using HiPerE

Figure 6 illustrates the design space exploration and optimization using HiPerE. The DSE process involves estimation of system wide energy and time performance estimates and use of these values to identify a set of designs as the candidates for the next phase. However, the designer can also perform optimizations by exploiting the information provided in the activity report for each element. This optimization process is a human-in-the-loop process. For example, if there is slackness in a processing element after execution of certain task and there is no data dependency, the designer can consider execution at a lower frequency. However, such modifications alter the activity report for each element. Therefore, once the modifications are performed, HiPerE is executed once again to generate an updated activity report. Finally, using a pre-determined terminating condition (e.g. number of iterations) system wide energy and latency estimations are provided to the designer. This process is repeated for all the designs provided to HiPerE to identify a set of candidate designs for low-level simulations.

4.4 An Illustrative Example

In order to illustrate our methodology we consider a signal processing application that evaluates the azimuth and elevation of an object within an image with respect to a reference coordinate (Figure 7). This application is part of a large radar-based application. The complete system is being modeled and explored using MILAN. At the time of this writing, we have modeled a subsystem shown in Figure 7. Also, in the simplified version considered here, timing constraints have been ignored. As shown in the figure the application consists of two parallel activities "Azimuth" and "Elevation". Azimuth is further divided into 6 tasks. Elevation also has a similar structure. This example also demonstrate the hierarchical nature of our application modeling paradigm.

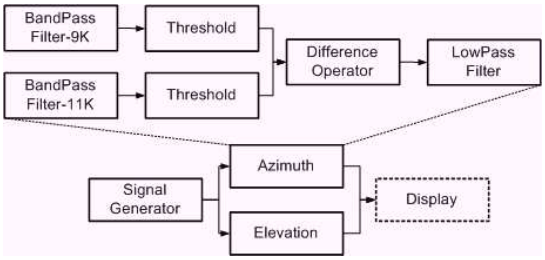

Figure 7: Azimuth elevation application task graph

We conducted experiments with three different architectures, a MIPS processor operating at a frequency of 600 MHz, a StrongARM processor operating at 206 MHz, and a heterogeneous dual processor architecture where both the processors are present. For this dual processor architecture we assumed a constant energy and time cost for data transfer between processors (independent of the source and destination task) through an interconnect.

The application and the resources were modeled using the MILAN modeling paradigms. Each application task was associated with two alternatives and each alternative was mapped to one of the processors. The rough estimates of the time and energy performance of each mapping were captured in the mapping model. We applied DESERT to identify three different designs; a) lowest in terms of time, b) lowest in terms of total energy dissipation, and c) minimum interconnect energy with an (user specified) upper bound on computation energy.

There are 13 different tasks in the application described above. Further, any of the tasks can be implemented on any of the processors. Therefore, based on the analysis described in Section 1, the total number of possible solutions is 8196.

The three designs identified by DESERT were evaluated based on system wide performance estimates provided by

HiPerE. We also demonstrate the accuracy of HiPerE estimates by comparing HiPerE results with the results from low-level simulation of the complete application.

HiPerE was given two sets of values. The "Rough Estimates" are obtained through back-of-the-envelope calculations of energy and time performance of each module after performing sample simulations using the low-level simulators. These estimates are the ones used by DESERT as well. The values in "Component Spec Sim" columns are obtained through isolated simulation using low-level simulators for each application task. These two sets of estimates demonstrate the multi-granular nature of our design space exploration. "Rough Estimates" are the result of estimations with lower accuracy and does not require simulation of all the tasks. For example, we only simulated one version of band pass filter as all the four band pass filters in the example are of similar complexity. On the other hand, for the values in "Component Spec Sim" column we performed component specific simulations for all the tasks.

The energy and latency estimation estimated using HiPerE is compared against the complete application simulation using low-level simulators. For low-level simulation we used SimpleScalar [22] and Wattch respectively for time and energy simulation for MIPS processor and JouleTrack [24] for simulation of the SA-1100 (Strong ARM) processor.

Table 1: Performance results using HiPerE and low-level simulations

Performance Metrics	Complete Low-level Simulations	Using Rough Estimates	Error %	Using Component Spec Sim	Error %
MIPS Processor @ 600 MHz					
Time (μs)	412	520	26	424	2.9
Energy (μj)	17820	22408	25	18354	3
StrongARM Processor @ 206 MHz					
Time (μs)	18228	20247	11	19350	6.1
Energy (μj)	6432	6807	5.8	6868	6.7
Dual Processor Architecture					
Time (μs)	6586	8247	25	6741	2.3
Energy (μj)	10039	11253	12	10270	2.3

Table 1 shows the performance when the application is executed on all the three architectures. As expected, the estimation using the component specific simulation results in higher accuracy (average error $< 3.9\%$). On the other hand the average error is high ($> 19\%$) when the estimates based on rough calculations are used.

If we compare various designs for energy, latency, and cost based on low-level simulations the MIPS architecture is the most attractive with respect to latency. SA-1100 is preferable if energy is the comparison metric and the dual processor architecture provides tradeoff between energy and time.

5. THE MILAN DESIGN FRAMEWORK

MILAN is built using Model Integrated Computing (MIC) technology previously developed at Vanderbilt University [25]. MIC allows designers to create domain-specific models of systems, validate these models, and perform various computational transformations on the models. These formalized models capture various aspects of the system's desired behavior. Model interpreters are used to translate these models for use in the system's execution environment. When changes in the overall system require new application programs, the models are updated to reflect these changes and the applications are regenerated automatically from the models.

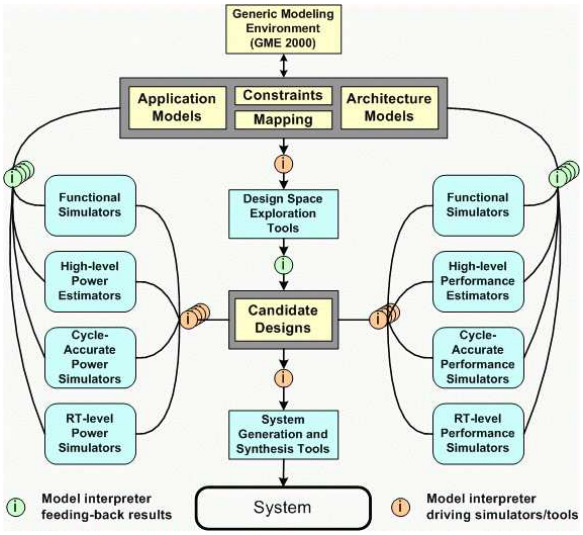

Figure 8: MILAN Architecture

The MILAN architecture is shown in Figure 8. Using the modeled application, resources, and constraints, MILAN is able to perform several activities included design space exploration, system generation, simulator configuration. MILAN currently supports different classes of simulators. Functional simulators, such as MATLAB or SystemC, verify the functionality of the application. The integrated high-level simulator provides a rapid, reasonably accurate estimate of different performance criteria of the system. Lower-level power and performance simulators, such as SimpleScalar or SimplePower, are also supported. While they can be very accurate, their slow speed may prevent the simulation of the whole system. MILAN supports the simulation of user selectable parts of the system.

Using MIC allows MILAN to evolve as new simulators and representation techniques become available. Using metamodels to represent the design language allows the representation methodology to be extended and evolved to support new advances in system representation [14]. Model interpreters can be written to interface new simulators into the existing framework. For example, DESERT and HiPerE are integrated into the MILAN framework using model interpreters. This also illustrates the plug-n-play nature of the MILAN framework. We can replace either of the tools by any tools performing similar functionality.

6. CONCLUDING REMARKS

This paper discussed design space exploration based on constraint satisfaction and high-level performance estimation in the context of application design on heterogeneous embedded architectures. The integration of a design space exploration tool (DESERT) and high level system wide la-

tency and energy estimation tool (HiPerE) makes the MILAN framework suitable for rapid system design. Simulator integration using the MILAN framework facilitates multi-granular simulation and updation of performance estimates. We are currently extending the MILAN framework for energy efficient data path synthesis using FPGAs and time and energy performance analysis of wireless sensor networks [4, 18, 23].

Acknowledgments

We would like to thank Akos Ledeczi (ISIS), Amol Bakshi (USC), Vaibhav Mathur (USC), and Aditya Agrawal (ISIS) for their valuable inputs and feedback concerning the MILAN architecture and design space exploration.

7. REFERENCES

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

[2] M. Auguin, L. Capella, F. Cuesta, and E. Gresset, "CODEF: A System Level Design Space Exploration Tool," Intl. Conf. on Acoustics, Speech, and Signal Processing, 2001.

[3] A. Baghdadi, N-E. Zergainoh, W. Cesario, T. Roudier, and A. Jerraya, "Design Space Exploration for Hardware/Software Codesign of Multiprocessor Systems," Intl. Workshop on Rapid System Prototyping, 2000.

[4] A. Bakshi, J. Ou, and V. K. Prasanna "Power-Aware Embedded System Design Using the MILAN Framework," Workshop on Integrated Management of Power Aware Communications, Computing, and Networking, May, 2002.

[5] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A, Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, "Hardware-Software Codesign of Embedded Systems: The POLIS Approach," Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

[6] T. Bapty, S. Neema, J. Scott, J. Sztipanovits, S. Asaad, "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems," Technical Report, ISIS, Vanderbilt University, 2000.

[7] T. Bapty and B. Abbott "Portable Kernel for High-Level Synthesis of Complex DSP-Systems," Intl. Conf. on Signal Processing Applications and Technology, 1995.

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

[9] R. Bryant, "Symbolic Manipulation with Ordered Binary Decision Diagrams," School of Computer Science, Carnegie Mellon University, Technical Report CMU-CS-92-160, July 1992.

[10] H. J. Eikerling, W. Hardt, J. Gerlach, and W. Rosenstiel, "A Methodology for Rapid Analysis and Optimization of Embedded Systems," Symposium on Engineering of Computer Based Systems, 1996.

[11] Generic Modeling Environment, http://www.isis.vanderbilt.edu/projects/gme/default.html.

[12] J. Henkel and R. Ernest, "High-Level Estimation Techniques for Usage in Hardware/Software Co-Design," Asia South Pacific Design Automation Conference, 1998.

[13] H. Hsien, F. Balarin, L. Lavagno, and A. S. Vincentelli, "Efficient methods for embedded system design space exploration," Design Automation Conference, 2000.

[14] Ledeczi A., Maroti M., A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The Generic Modeling Environment," Workshop on Intelligent Signal Processing, 2001.

[15] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems," Workshop on Signal Processing Systems, 1999.

[16] V. Mathur and V. K. Prasanna, "A Hierarchical Simulation Framework for Application Development on System-on-Chip Architectures," IEEE Intl. ASIC/SOC Conference, 2001.

[17] S. Mohanty and V. K. Prasanna, "Rapid System-Level Performance Analysis and Optimization for Heterogeneous SoC Architectures," submitted to 15th IEEE Intl. ASIC/SOC Conference, 2002.

[18] S. Mohanty, S. Choi, J. Jang, and V. K. Prasanna, "A Model-based Methodology for Application Specific Energy Efficient Datapath Design using FPGAs," to appear in Application-specific Systems Architectures and Processors, 2002.

[19] S. Neema, "Design Space Representation and Management for Model-Based Embedded System Synthesis," Technical Report ISIS-01-203, February 2001.

[20] S. Neema, "System Level Synthesis of Adaptive Computing Systems", Ph.D. Dissertation, Vanderbilt University, Department of Electrical and Computer Engineering, May 2001.

[21] A. Pimentel, L. Hertzbetger, P. Lieverse, P. van der Wolf, and E. Deprettere, "Exploring Embedded-systems Architectures with Artemis," IEEE Computer, November 2001.

[22] SimpleScalar Tool Set, http://www.simplescalar.org/.

[23] M. Singh and V. K. Prasanna "System Level Energy Tradeoffs for Collaborative Computation in Wireless Networks," IEEE Workshop on Integrated Management of Power Aware Communications, Computing, and Networking, May, 2002.

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

[25] J. Sztipanovits and G. Karsai, "Model-Integrated Computing," IEEE Computer, April 1997.