Integrating Application Specification and Performance Prediction for Strategic Design-Space Exploration

Brian Holland, Alan D. George, and Herman Lam
NSF Center for High-Performance Reconfigurable Computing (CHREC)
University of Florida
Email: [holland, george, lam]@chrec.org

***Abstract*—Modeling environments and performance prediction boost application productivity, but often lack integration into an efficient and comprehensive approach to strategic design-space exploration for reconfigurable computing (RC) systems. This paper proposes a framework allowing simple yet extensible bridging of modeling environments for high-level application specification with the authors' RC Amenability Test (RAT), an analytical model for performance prediction prior to expensive implementation. Two case studies, a modified Needleman-Wunsch application for bioinformatics and a task graph of mean-value analysis, demonstrate the efficiency of the proposed integrated framework for rapid and reasonably accurate analysis of application designs.**

I. Introduction

Widespread adoption of reconfigurable computing (RC) systems is increasingly limited by application design productivity. Particularly for field-programmable gate arrays (FPGAs), prototyping and revising applications based on execution-time performance analysis of hardware implementations is impractical due to lengthy synthesis, placement, and routing phases. Existing design flows for RC systems lack structured application specification and analysis at an abstract level. Strategic design-space exploration (DSE), the evaluation and revision of algorithm and architecture design choices based on the application's performance requirements *prior* to costly implementation, receives insufficient attention. Illustrated in Figure 1, a key productivity challenge is the gap in the abstraction pyramid for system design [1] between "Back of the Envelope" estimations and "Abstract Executable Models." Existing modeling environments and performance prediction techniques provide important yet commonly isolated methodologies and tools for application specification (i.e., design entry) and analysis, respectively.

One solution to the RC productivity challenge is an extensible methodology and tools for bridging modeling environments with performance prediction. Modeling environments provide more customizable, human-productive design entry than lists or spreadsheets of quantitative performance information. Performance prediction techniques can provide relatively rapid and reasonably accurate estimations, albeit with potentially manual data input, error checking, and revision. Without an integrated approach, strategic DSE using isolated abstract specification and analysis tools can be tedious, disconnected from subsequent implementation tasks, and ultimately counterproductive.

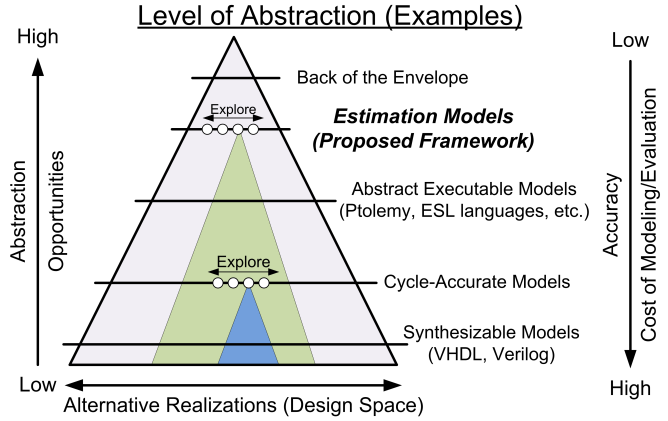


Fig. 1. Abstraction pyramid comparing levels of modeling for hardware applications [1]

This paper proposes a *methodology* for a framework allowing integration of modeling environments with the authors' RC Amenability Test (RAT) [2], an analytical model for FPGA performance prediction prior to implementation. The RAT methodology (and corresponding analysis tool) provides models describing the behavior of the individual computation and communication operations and estimates the total application performance based on their agglomeration. RAT has demonstrated reasonably accurate performance prediction, but its efficiency is limited by currently manual interpretation of application specifications for the necessary inputs to the analysis.

A number of different tools for strategic DSE (hereafter referred to as "DSE tools") can be constructed based on the proposed framework depending on the capabilities and constraints of the included modeling environment. One such DSE tool is presented with case studies in Section IV. Using the framework methodology as the basis for DSE tools allows greater interchangeability of modeling environments and reuse of components connecting the underlying specification and analysis tools. Consequently, the proposed framework defines a "translation" component that distills the required prediction inputs for RAT from the (supported) model of computation (MoC) of the application specification. An abstraction layer insulates the translation functionality from the tool-dependent details of the particular modeling environment. Additionally, the framework defines an "orchestration"

component, which performs RAT prediction on an initial application design and potential revisions to the underlying algorithm and/or platform architecture. The functionality of a DSE tool includes the connectivity and usage of existing tools for application specification and RAT prediction by the newly constructed components for translation and orchestration.

The remainder of this paper is structured as follows. Section II presents background and related research on RAT performance prediction and several modeling environments amenable to integration with RAT, which ultimately provide the analysis and specification capabilities for the framework. Section III provides an overview of the requirements for the translation and orchestration components of the framework methodology that enable integration of modeling environments and performance prediction. Section IV discusses two case studies, a modified Needleman-Wunsch (MNW) application for bioinformatics and a task graph of mean-value analysis (hereafter referred to as "MVA graph"), demonstrating productivity benefits with a DSE tool. Conclusions are presented in Section V.

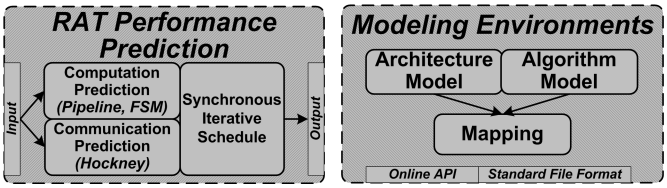
II. Background and Related Research

The proposed framework leverages existing research in RAT performance prediction and modeling environments to facilitate application specification and analysis for strategic DSE for RC. RAT uses separate computation and communication models, based on underlying assumptions about the application structure and behavior, which agglomerate into complete predictions of application. Several modeling environments provide methods and tools with similar approaches for abstracting algorithm and platform architecture specifications, albeit with differing levels of implementation detail.

A. RAT Performance Prediction

Assuming known model inputs, the rapidity of analytical modeling is advantageous for *strategic* application analysis as compared to direct measurement based on a hardware implementation or simulation, which often requires lengthy analysis times. The accuracy (i.e., correctness) of the analytical models such as RAT [2] is based on the validity of the simplifying assumptions that keep the analysis tractable. RAT assumes a synchronous iterative (i.e., multiphase) performance model, a subset of fork-join models with each hardware resource (e.g., microprocessor or FPGA) performing an independent portion of the application computation each iteration with synchronizing communication separating every iteration from preceding and proceeding iterations [3], [4]. The model assumes application execution time is defined by the summation of the slowest computation and communication each iteration.

The underlying computation and communication models for RAT describe the potentially complex and typically data-oriented behaviors within each iteration using a few key quantitative attributes. Computation is defined by total number of application-specific operations and their rate of execution based on usage of the algorithm's deep or wide parallelism by the hardware resources. Alternatively, RC computation can often be described by the number of data elements to be processed and the rate of completion (i.e., cycles per element). RAT uses an extension of the Hockney model [5] to describe communication. Figure 2a outlines the general structure of RAT.



(a) Performance prediction with RAT (b) Y-chart approach to app. specification in modeling environments [1]

Fig. 2. Abstract structure of RAT performance prediction and modeling environments

A tool constructed from the RAT methodology includes an API to provide the necessary prediction input and gather the resulting performance estimation. The general methodology of separate computation and communication modeling is common in prediction techniques, though research directed towards strategic RC analysis is not as expansive as compared to modeling environments. RAT is included within the framework due to its fairly unique focus of strategic prediction prior to implementation. Encapsulation of RAT for replacement with other synchronous iterative performance models is possible, but outside the scope of this paper.

B. Modeling Environments

Modeling environments provide abstract yet reasonably precise descriptions of application structure and behavior. An application specification consists of models (i.e., descriptions) of the underlying algorithm and RC platform architecture along with their respective mapping. Algorithm and architecture models are often specified separately using the Y-chart approach [1], as illustrated in Figure 2b. These application models (particularly the algorithm model) describe the behavior of an application in terms of a model of computation (MoC). A MoC defines a set of allowable "operations" (i.e., basic and often technology-dependent computational events), communication between operations (i.e. data movement), their relative costs (e.g., clock cycles), and the total system behavior based on the operations composing the application. Each modeling environment uses graphical and/or textual elements to denote precise syntactic and semantic meanings for an application specification based on the MoC.

The case studies for this paper, MNW and MVA graph, are specified by asynchronous message-passing (AMP) and synchronous dataflow (SDF) MoCs, respectively, which represent common models for FPGA systems. AMP denotes the use of one or more queues to describe communication between groups of operations. Only messages within the same queue are strictly ordered with unspecified timing between different queues. SDF represents a special case of AMP with groups of operations evaluated as soon as the necessary messages are available from the communication channels, which are unidirectional. Data enters the application model at a constant

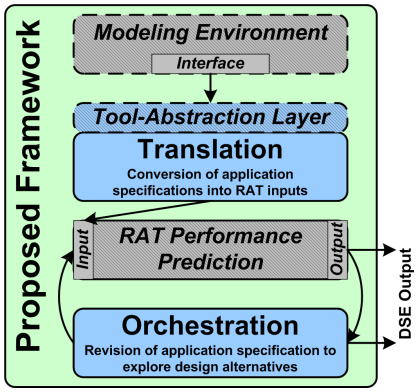


Fig. 3. Framework bridging modeling environments and performance prediction, and orchestrating DSE

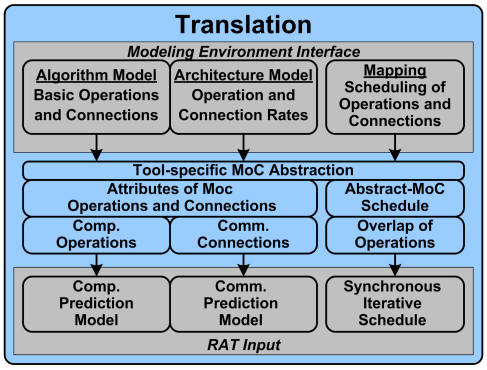


Fig. 4. Translation of application specification information for RAT prediction

rate, which eventually induces a steady-state evaluation rate for each group of operations with total performance defined by the slowest group. AMP suitably describes the straightforward DMA communication between the microprocessor and FPGAs for MNW. SDF provides mechanisms for describing the pipeline network of the MVA graph.

The proposed DSE tool requires a modeling environment capable of effectively representing abstract algorithm, architecture, and subsequent application mapping models based on AMP and SDF MoCs. Ptolemy [6] is an environment specifically for simulating and prototyping systems involving heterogeneous MoCs, including AMP and SDF. Metropolis [7] defines "metamodels" that use formal execution semantics to define the application function, platform architecture, and mapping of the system based on a new or existing MoCs. Artemis [8] and Sesame [9] use a hierarchical Kahn Process Network (KPN), a specialized AMP MoC, to describe the system concurrency and individual component behavior. The RC Modeling Language (RCML) [10] provides hierarchical models (typically involving AMP and SDF MoCs) for the algorithm, architecture, and total application mapping with specialized constructs to express parallelism, communication patterns, and other common aspects in RC. RCML is intended to allow users to quickly model systems before lengthy coding of a functional implementation, using abstract constructs and quantitative attributes to define behavior. With suitable abstraction, any of these modeling environments could provide effective application specification for RAT performance prediction. The proposed DSE tool in Section IV uses RCML because of the strategic, RC-specific focus for application specification.

III. Integrated Framework

This section describes the general methodology of the framework for connecting RAT performance prediction with modeling environments for increased productivity during strategic DSE. (Section IV discusses the DSE tool bridging RAT with the RCML modeling environment.) Figure 3 provides an overview of the framework structure. The proposed methodology includes *translation* of specification information from the modeling environment to RAT and *orchestration* of DSE based on revisions to the specification information. The modeling environment and RAT performance prediction components are the existing methods and tools from Section II, as indicated by the shaded boxes. The dashed border of the modeling environment and tool-abstraction layer indicates the interchangeability of the specification tool. Section III-A describes the general procedure for translation of algorithm-based MoCs into the quantitative performance attributes and application scheduling necessary to direct the synchronous, iterative performance model of RAT. Section III-B provides a high-level description of the mechanism for orchestration of strategic DSE, specifically the directed revision of an initial application specification to examine and compare the performance potential design alternatives.

A. Translation

Although individual modeling environments and performance prediction techniques sometimes include methods for direct connectivity to other tools, an explicit intermediary between specification and analysis is advantageous. The proposed framework provides translation between the algorithm MoCs and the RAT performance prediction, facilitating the transfer of the required quantitative attributes and scheduling information to the corresponding computation and communication model. Potential issues during translation include differences in the data structures (e.g., format, representation, or precision), abstraction levels, and semantic mean along with other dilemmas such a missing, redundant, or inconsistent data. Resolving these issues can require acute awareness of the low-level details of the data formats, syntax, and semantics of the tools with extra functionality to identify and request additional information from the user as necessary. The need for unique bridges between every desired modeling tool and RAT is greatly reduced by an abstraction layer, which allows the framework to perform the majority of the translation based on a generic format for algorithm MoCs derived from the specific modeling environment tool.

As illustrated in Figure 4, the algorithm and architecture attributes for the basic operations of the MoC of the application specification must be reorganized and formatted

based on their contribution to the RAT computation and/or communication estimation. The framework constructs RAT computation models for every hardware resource based on the groups of operations mapped to that resource and RAT communication models based on data movement between hardware resources. A generic schedule that describes the parallelism and overlap between the computation and communication is constructed from the semantics of the MoC. Conversion between algorithmic MoCs and RAT performance models is possible as the important structure and behavior of the application specification, properly formatted, correspond directly to available computation and communication models. The basic computation operations within an MoC are often generic abstractions that require additional quantitative attributes from the application designer, specifically formatted for the assumed technology (e.g. FPGAs), for translation to the RAT model.

For the DSE tool used in Section IV (and by extension, any future tool connecting modeling environments and RAT), the underlying translation step must be tuned for the MoCs of interest, specifically AMP and SDF for the case studies. For FPGA systems, these MoCs can be abstractly represented as a number of "tasks" (i.e., generic encapsulations of groups of operations with detailed specification left to the application designer) with the data movement through algorithm "connections." Tasks often represent either pipelines or state machines, which imply structured execution at a deterministic (or statistically observed) rate. Classification of these basic operations is straightforward because tasks perform only computation and connections facilitate only communication. The quantitative attributes for the computation tasks include the amount of data to be processed and the cost of processing each data element, which are contained within the particular task specification. Similarly, quantitative attributes for algorithm connections define the amount of data and segmentation for transfers between tasks, which are contained within the connection specification. Computation and communication models for tasks and connections, respectively, are provided with corresponding architectural information (e.g., FPGA clock frequency or interconnect bandwidth) by the framework translation based on the application mapping. For scheduling, the key difference between the two MoCs is the specificity of the overlap of task execution. For AMP, task execution is dependent only upon the order of communication message from its predecessor tasks (i.e., those prior tasks which provide data to the current task). In contrast, SDF models assumes simultaneous, fine-grain operation of all tasks and connections, typically as a pipeline operating on individual data elements within one or more streams of data. In practice, AMP is sufficient for serializing communication between microprocessors and FPGA application accelerators (e.g., MNW) whereas SDF is useful for describing multiple directly connected pipelines (e.g., MVA graph).

B. Orchestration

Strategic DSE involves evaluating a range of application designs to determine the most desirable configuration. Design alternatives may differ in multiple facets including the algorithm requirements (e.g., problem size) and architectural capabilities (e.g., clock frequency). The framework supports strategic DSE by repetitively revising an application specification and evaluating the resulting performance against other design alternatives. RAT is provided different sets of quantitative performance features, which typically represent several permutations of one or more attributes. (DSE based on major revisions to the application mapping is outside the scope of this paper.) Predictions are revised not once but potentially hundreds or thousands of times depending on the breadth of the design space and complexity of the algorithm.

Strategic DSE begins with identification of performance features for revisions. The application designer may choose to annotate a parameter with one or more alternative values denoting possible changes to the application design. The goal is to propose revisions to specific features and compare the range of performance values against the performance requirements of the designer. For example, several different pipeline rates or clock frequencies may be evaluated. Also, scalability can be analyzed using revisions that define progressively larger problem sizes and hardware resources. Alternatively, different schedules can be evaluated by adjusting the ordering (i.e., priority) of messages to outgoing communication channels. As illustrated by the case studies in Section IV, rapid exploration of large design spaces can greatly aid design productivity. However, a designer using the framework must ensure that the design space under investigation is realistic with respect to the architectural constraints (e.g., maximum circuit size or clock frequency).

IV. Case Studies

This section describes two case studies, MNW and MVA graph, which demonstrate the capabilities of the integrated framework for efficient (i.e., rapid and reasonably accurate) strategic DSE. The experimental setup, including the construction of the DSE tool bridging the RCML modeling environment with RAT performance prediction, is discussed in Section IV-A. The MNW case study in Section IV-B is a bioinformatics application with an AMP MoC. This case study demonstrates accurate prediction, as compared to subsequent hardware implementations, and rapid DSE. The MVA graph in Section IV-C contains a more complex network of pipelines with performance defined by the SDF MoC. This case study maintains rapid DSE, even for very large numbers of revisions to a complex algorithm structure.

A. Experimental Setup

For validation of the proposed methodology, a DSE tool provides functionality for gathering the application specification, performing translation, and orchestrating DSE using performance prediction. This functionality includes interaction with a modeling environment tool, RCML, to collect the necessary information from an application specification and usage of a prediction tool, RAT, for performance analysis. RCML provides an RC-specific abstraction environment with semantic constructs amenable to the RAT prediction. The

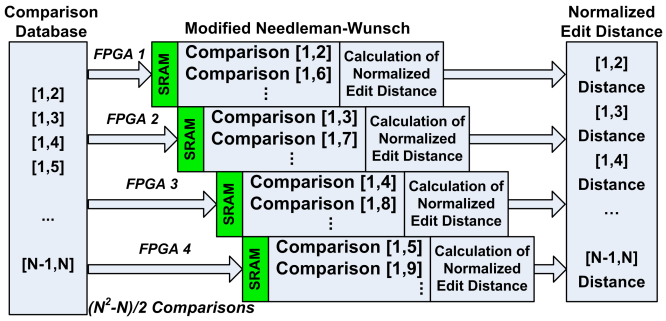
Fig. 5. Architecture specification of FPGA platform

DSE tool is a hierarchical composite of existing tools for RCML and RAT and newly constructed components providing translation and orchestration. These translation and orchestration components are implemented as a Java-based Eclipse plug-in to help minimize the customized interfacing necessary for connecting to the RCML and RAT tools. The specific implementation details of the DSE tool are outside the scope of this paper. Briefly, the translation component constructs an environment-independent graph representation of the application behavior based on the MoC for use in RAT performance prediction. The orchestration component adjusts the graph representation and provides these revisions to the RAT tool.

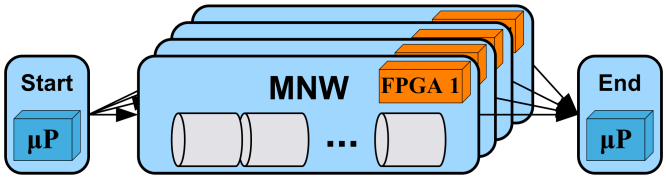
The two application case studies for this paper are mapped onto a Linux server containing a GiDEL PROCStar-III FPGA card connected by a PCIe $\times 8$ bus to a Xeon E5520 (i.e., 2.26GHz Quad-core Nehalem) microprocessor. The GiDEL FPGA card contains four Altera Stratix-III E260 FPGAs, which have interconnects to adjacent FPGAs and support DMA transfers to and from the microprocessor. Figure 5 outlines the general architecture model for the FPGA-augmented platform. This FPGA system can be used as a prototype for an RC-augmented embedded platform or represent a single node in a multi-node RC supercomputer.

B. Modified Needleman-Wunsch (MNW)

The MNW case study is an FPGA-optimized application for calculating the normalized edit distance between two DNA sequences within the composite ESPRIT application for metagenomics [11]. The normalized edit distance provides concise quantitative insight about the similarity of two DNA sequences based on the length of the sequences, the number of gaps in the global sequence alignment, and the number of edits required to transform one sequence string into the other. The MNW application pipelines the standard Needleman-Wunsch [12] calculations for individual alignment scores and resulting global alignment with the ESPRIT calculation of the normalized edit distance. The pipeline concurrently computes the alignment scores with the normalized edit distance rather than computing the edit distance from the character representation of the alignment as is done in software. Computing the edit distance in this way eliminates the need to store a score matrix, significantly reducing the memory requirements for the FPGA system. This case study is referred to as *modified* because the typical outputs of Needleman-Wunsch, the score matrix and global alignment, are unnecessary after the calculation of the normalized edit distance and are never retained. However, as with traditional Needleman-Wunsch,



(a) Calculation of normalized edit distances on multiple FPGAs for MNW



(b) MNW algorithm specification and mapping

Fig. 6. Overview of MNW case study

MNW is often useful for comparing many pairs of sequences of similar length as a batch.

Figure 6a provides a general overview of the algorithm structure. A database of comparisons is built from the sequences and divided, round-robin, among the specified number of FPGAs. A total of N sequences requires a database of $\frac{N^2-N}{2}$ comparisons since sequences are not compared against themselves and comparisons such as $[2,1]$ are equivalent to $[1,2]$. The initial configuration of this case study involves 1500 sequences, each 105 characters in length. The resulting $\frac{N^2-N}{2}$ normalized edit distances are collected by microprocessor after computation is complete.

The framework queries the modeling environment for the quantitative performance information necessary for RAT prediction. The communication of the DNA sequence database and the resulting values for the normalized edit distance are described using the AMP MoC. From the algorithm specification (Figure 6b), each of the computation tasks (Start, MNW, and End) and two communication connections requires a separate analytical model. The performance of the software Start and End tasks are defined by an execution-time attribute. The number of characters in the database of sequence comparisons determines the amount of input communication (between Start and MNW) and the amount of computation for MNW. The output communication (between MNW and End) is defined by the number of sequence comparisons. The architecture model (Figure 5) contains the parameters outlining the communication capabilities of the PCIe interconnect. The FPGA clock frequency (architecture) and pipeline depth (algorithm) parameters define the computation rate.

Table I summarizes the predicted and experimental execution times for MNW using 1500 DNA sequences (i.e., $\frac{1500^2-1500}{2}$ comparisons) divided across 1, 2, and 4 FPGAs. The predicted execution times were generated by RAT based on the quantitative performance information provided by the

TABLE I
PREDICTED AND EXPERIMENTAL RESULTS FOR MNW

	Predicted Time (s)	Experimental Time (s)	Error
1 FPGA	9.44E-1	9.58E-1	1.5%
2 FPGAs	4.72E-1	4.83E-1	2.3%
4 FPGAs	2.36E-1	2.46E-1	4.1%

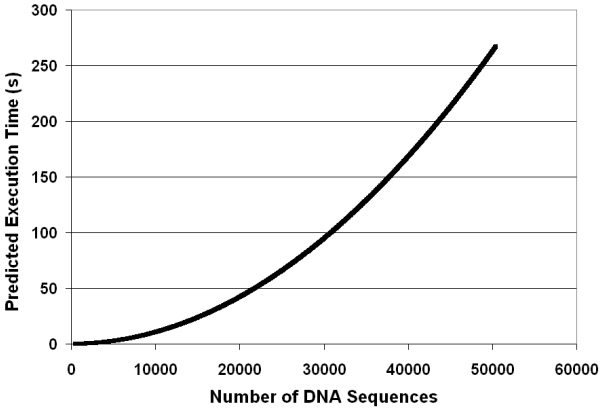


Fig. 7. Predicted execution times of MNW on four FPGAs based on revisions to the number of DNA sequences for comparison

framework. The experimental execution times were measured from subsequent hardware implementations that correspond to the application specification. Based on the 1% to 4% error rate, the integrated framework was able to maintain reasonable accuracy during the abstract application specification, collection of quantitative performance parameters, and resulting performance prediction. Generating the abstract specification took only a few minutes and the subsequent analysis, as directed by the framework, took approximately 2.3ms. The productivity gained by using the framework is significant because the actual hardware implementation for MNW required approximately 200 man-hours to code, place and route, debug, and evaluate.

Beyond the initial prediction, evaluating the performance impact of alternative MNW designs can provide insight about the desirability of possible structural or behavior revisions. The DSE tool can explore different architectural optimizations (e.g., faster pipelines), but these analyses can be trivial for this computation-bound application due to the direct correspondence between the rate of execution and the overall application performance. Instead, Figure 7 illustrates the predicted execution time of MNW based on 1000 design revisions that represent different problem sizes (i.e., comparisons of 500 to 50450 DNA sequences) divided among four FPGAs. These revisions expand the initial four-FPGA design of 1500 DNA sequences. The execution time of MNW increases exponentially with the number of DNA sequence comparisons. This DSE can help evaluate the suitability of the MNW application for meeting the broad performance requirements of a designer, particularly when the size of the sequence database is expected to increase at a potentially unknown rate after implementation. The DSE tool took only 6.1ms to analyze this significantly larger design space. Table II

TABLE II
ANALYSIS TIMES FOR DESIGN SPACES OF MNW

Number of Revisions	Analysis Time (ms)
0 (initial design)	2.3
1000	3.0
10000	15
100000	140

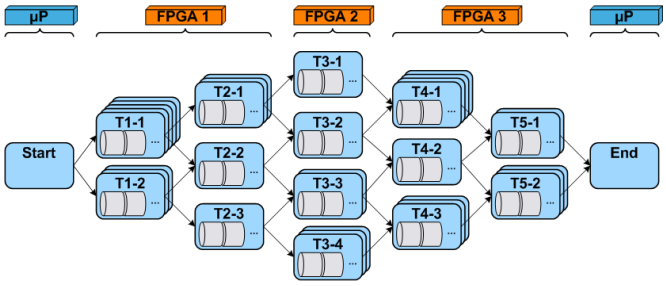


Fig. 8. MVA graph specification and mapping

summarizes analysis times for the initial design, the 1000 revisions, and two other large DSEs. The analysis times grow linearly with the size of the design space and allows very large numbers of revisions to be explored in significantly less than one second.

C. Task Graph of Mean-Value Analysis (MVA Graph)

An MVA graph is a defined structure for tasks and dependences that commonly describes the parallel decomposition of a recursive algorithm. As illustrated in Figure 8, the general algorithm structure widens until the particular "base" case is achieved and subsequent contracts to agglomerate the individual values. This task graph is commonly associated with mean-value analysis [13], though other algorithms such as quicksort have comparable structure. For this case study, the tasks represent a network of pipelined computations conforming to the SDF MoC. Random volumes of data and execution rates (i.e., pipeline rates and parallel decomposition) are assigned to each task. Communication is based on the multi-FPGA mapping. The structure of the MVA graph is often used as a benchmark for hardware/software scheduling algorithms. For this case study, the randomly populated MVA graph is used as a synthetic application to demonstrate the capabilities of the DSE tool for rapidly analyzing large design spaces of complex applications.

Although the algorithm structure is a SDF MoC, collecting quantitative parameters from the individual computation tasks and communication operations remains very similar to the MNW case study. The algorithm complexity is manifested in the number of tasks, their dependencies, and their scheduling. The predicted execution time of the initial design for the MVA graph is 0.17s, which is dominated by the slowest pipeline, T2-1 (Figure 8), due to its highest assigned workload. Strategic DSE can provide useful insight about minimum execution rates for the *other* pipelines and allows a designer to construct the slowest (and least resource-intensive) pipeline possible without increasing the overall execution time for the

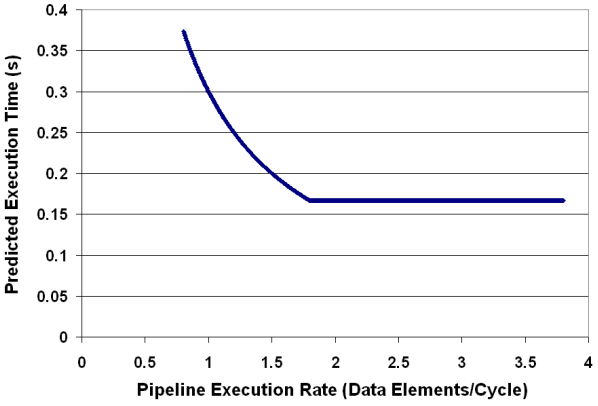


Fig. 9. Predicted execution times of the MVA graph based on revisions to the execution rate of the T4-2 pipeline

TABLE III
ANALYSIS TIMES FOR DESIGN SPACES OF MVA GRAPH

Number of Revisions	Analysis Time (ms)
0 (initial design)	10
1000	12
10000	33
100000	340

application. For example, Figure 9 illustrates the predicted execution time of the MVA graph based on 1000 design revisions that represent different execution rates for the T4-2 pipeline. Pipeline rates for T4-2 above a certain threshold have no impact on the total performance of the MVA graph because the execution time remains dominated by the slower T2-1 pipeline. However, sufficiently slow rates for the T4-2 pipeline increase the execution time of the MVA graph. Rapid determination of this threshold is difficult without the DSE tool.

Despite the large number of revisions, DSE using the integrated framework remained tractable. Table III summarizes the framework analysis times, including the transfer of the performance information and RAT prediction, for various numbers of revisions to the design of the MVA graph. The analysis time grows approximately linearly with the size of the design space. The longest analysis of 100,000 revisions took 340ms, which is nearly indistinguishable by the user from the duration of a single analysis of a simple application (e.g., 2.3ms for MNW). The primary limitation of broad DSE (aside from Java memory requirements) is the ability of the user to efficiently digest the generated prediction values. Additional analysis tools could be constructed to identify the highest performing design(s) based on criteria such as fewest revisions from the original application specification, but such features are outside the scope of this paper.

V. CONCLUSIONS

Widespread adoption of FPGA systems is limited by application development productivity. Modeling environments and performance prediction help reduce the costly development process by facilitating iterative application refinement prior to hardware implementation. However, efficient design-space exploration requires a comprehensive approach to application specification, analysis, and ultimately revision, if necessary.

The proposed framework introduces a methodology to integrate modeling environments with RAT performance prediction for strategic DSE. Tools based on the framework facilitate translation of application specifications into performance characterizations and orchestration of the required RAT predictions. The DSE tool described in this paper demonstrated accurate performance analysis with under 5% error for the MNW case study. Strategic DSE with the tool was efficient and rapid, requiring only 140ms and 340ms for analysis of 100,000 revisions to the MNW application and the MVA graph, respectively. Future work includes more detailed focus on low-level issues for translation and orchestration components, and broader support for performance prediction of scalable FPGA systems.

ACKNOWLEDGMENTS

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge vendor equipment and tools provided by Altera.

REFERENCES

[1] B. Kienhuis, E. F. Deprettere, P. van der Wolf, and K. Vissers, *Embedded Processor Design Challenges*. Springer, 2002, ch. A Methodology to Design Programmable Embedded Systems: The Y-Chart Approach, pp. 18–37.

[2] B. Holland, K. Nagarajan, and A. D. George, "Rat: Rc amenability test for rapid performance prediction," *ACM Trans. on Reconfigurable Technology and Systems (TRETS)*, vol. 1, no. 4, pp. 22:1–22:31, 2009.

[3] G. D. Peterson and R. D. Chamberlain, "Beyond execution time: Expanding the use of performance models," *IEEE Parallel Distrib. Technol.*, vol. 2, no. 2, pp. 37–49, 1994.

[4] M. Smith and G. Peterson, "Parallel application performance on shared high performance reconfigurable computing resources," *Perform. Eval.*, vol. 60, pp. 107–125, May 2005.

[5] R. W. Hockney, "The communication challenge for mpp: Intel paragon and meiko cs-2," *Parallel Comput.*, vol. 20, no. 3, pp. 389–398, 1994.

[6] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation*, vol. 4, pp. 152–184, April 1994.

[7] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment," *Computer*, vol. 36, no. 4, pp. 45–52, April 2003.

[8] A. D. Pimentel, L. O. Hertzbetger, P. Lieverse, P. van der Wolf, and E. F. Deprettere, "Exploring embedded-systems architectures with artemis," *Computer*, vol. 34, no. 11, pp. 57–63, November 2001.

[9] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. on Computers*, vol. 55, no. 2, pp. 99–112, February 2006.

[10] C. Reardon, B. Holland, A. George, G. Stitt, and H. Lam, "RCML: An environment for estimation modeling of reconfigurable computing systems," *ACM Transactions on Embedded Computing Systems (TECS)*, to appear.

[11] Y. Sun, Y. Cai, L. Liu, F. Yu, M. L. Farrell, W. McKendree, and W. Farmerie, "Esprit: estimating species richness using large collections of 16s rrna pyrosequences," *Nucleic Acids Res.*, vol. 37, no. 10, p. e76.

[12] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[13] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queuing networks," *J. ACM*, vol. 27, no. 2, pp. 313–322, 1980.