A Framework for Evaluating High-Level Design Methodologies for High-Performance Reconfigurable Computers

Esam El-Araby, *Member IEEE*, Saumil G. Merchant, *Member IEEE*, and Tarek El-Ghazawi, *Senior Member, IEEE*

Abstract— High-performance reconfigurable computers have potential to provide substantial performance improvements over traditional supercomputers. Their acceptance, however, has been hindered by productivity challenges arising from increased design complexity, a wide array of custom design languages and tools, and often overblown sales literature. This paper presents a review and taxonomy of High-Level Languages (HLLs) and a framework for the comparative analysis of their features. It also introduces new metrics and a model based on computational effort. The proposed concepts are inspired by Netwon's equations of motion and the notion of work and power in an abstract multi-dimensional space of design specifications. The metrics are devised to highlight two aspects of the design process; the total time-to-solution and the efficient utilization of user and computing resources at discrete time steps along the development path. The study involves analytical and experimental evaluations demonstrating the applicability of the proposed model.

Index Terms— High-level language productivity, performance evaluation, productivity, reconfigurable computing.

1 Introduction

PROGRAMMABILITY challenges with high-performance reconfigurable computers (HPRCs) have hindered their wide spread acceptance amongst the supercomputing community. Application development on these systems typically requires software and hardware programming expertise for which design paradigms and tools have been traditionally separate. The standard way of describing software is using high-level languages (HLLs), such as C, C++, or Fortran, whereas, hardware is typically designed using hardware description languages (HDLs), such as VHDL and Verilog. Fragmented design flow and the need for expertise in parallel software and hardware design are major productivity hurdles facing the high-performance reconfigurable computers. To bridge the productivity gap several high-level language tools such as Xilinx Forge, Celoxica Handel-C, Impulse-C, and Mitrion-C have been proposed which attempt to abstract underlying hardware design details and streamline the disparate design flows. These tools often tradeoff performance for programmability. Dataflow design tools, based on the graphical user interface, e.g. DSPLogic, seem to offer an interesting compromise between HLLs and HDLs. These languages offer a trade-off between a shorter development time and a performance overhead imposed by high level languages.

Streamlining hardware description using HLLs typically used in software programming, or at least using dataflow languages, is a major and distinctive feature of HPRCs that potentially allows domain scientists to develop entire applications without relying on hardware designers. However, an HLL compiler for HPRCs must combine the capabilities of tools for traditional microprocessor compilation and tools for computer-aided design with FPGAs. It must also extend these two separate set of tools with capabilities for mutual synchronization and data transfer between microprocessors and reconfigurable processor sub-systems [1, 2]. The problems are further escalated by lack of standard interfaces and architectural diversity in reconfigurable computing sub-systems. Moreover, the range of tool choices and puffed up sales literature make it hard to comprehend real differences.

This paper aims to present a framework and a mathematical model to compare and contrast different HLL languages and their features. For this a detailed review and a taxonomy of the existing design languages for HPRCs is provided. The model and the metrics to evaluate HLLs are inspired from the principles of Newton's equations of motion and the notions of work and power in an abstract multi-dimensional space of design specifications. They highlight two distinct aspects of the development process, (i) the total time-to-solution, and (ii) the efficient utilization of user and computing resources. We believe that this enables a comprehensive evaluation of the design languages. The experimental study presented includes HLLs from the imperative and the dataflow programming paradigms, showcasing the wide applicability of our methodology. In brief, the major contributions of this work are as follows: (i) A detailed review and taxonomy of HLL languages; (ii) New evaluation metrics to emphasize the total time to solution

Manuscript received October 1, 2009.
Esam El-Araby, Saumil Merchant, and Tarek El-Ghazawi are with the NSF Center for High-performance Reconfigurable Computing (CHREC), ECE Department, The George Washington University, Washington, DC 20052. E-mail: esam@gwmail.gwu.edu, {smerchan, tarek}@gwu.edu.

as well as the resource usage efficiency of an HLL tool/language along the development path; and (iii) An analytical framework for comparative analyses of HLL languages.

Remainder of this manuscript is organized as follows. Section 2 presents a detailed review and taxonomy of HLL design tools for HPRCs. Section 3 presents the related work. Section 4 introduces the model and the framework to evaluate and compare HLLs. Section 5 presents an experimental study that uses the proposed framework to evaluate HLLs from imperative and dataflow programming styles. Finally, section 6 concludes the paper.

2 HLL Review and Taxonomy

To understand and evaluate the different language attributes, taxonomy is imperative. This section provides a thorough review and taxonomy of available HLL tools in research and commercial literature. Table 1 shows a list of HLLs reviewed. Some of the listed languages are text-based, either C-, Fortran-, Java-, or Matlab-like, others are Graphical-based.

Our review revealed that various vendors provide not only a high-level language, but also a complete development environment that may integrate with the tools of the basic development flow, see Fig. 1. Users can develop their applications using either the standard HDL flow or a suite of higher-level languages such as C and C++ or the Xilinx System Generator for DSP package.

Impulse Accelerated Technologies, for example, provides a C-based development kit, Impulse-C, that allows the users to program their applications in standard C with the aid of a library of functions for describing

Table 1. Reviewed HLLs

1	Impulse-C	14	System-Studio	27	CoreFire
2	Handel-C	15	ConvergenSC	28	DSS
3	Mitrion-C	16	Transmogrifier-C	29	Forge
4	Dime-C	17	SPARK	30	CASH
5	System-C	18	Brass	31	C2Verilog
6	Catapult-C	19	DeFacto	32	Bach C
7	Carte-C	20	MATCH	33	SpecC
8	Streams-C	21	JHDL	34	Ocapi
9	AccelChip	22	Galadriel & Nenya	35	HardwareC
10	NAPA-C	23	Viva	36	Cones
11	SA-C	24	Ptolemy II	37	BDL
12	Trident Compiler	25	SysGen	38	Cocentric
13	CHiMPS	26	RC Toolbox	39	C2H

parallel processes, partitioning the application into software and hardware parts, and simulating and instrumenting the application. The Impulse-C programming model provides stream-based communication between processes. The kit provides a compiler that generates HDL code for synthesis from the hardware parts of the application targeting different HPRC platforms such as Cray-XD1 and/or SGI Altix/RASC [3, 4]. Impulse-C represents a class of imperative languages with syntax based strongly on ANSI C [5]. The language is extended to address specific hardware concepts such as communicating sequential processes (CSP) and streams. Existing VHDL designs may also be incorporated and called from the Impulse-C code as external functions.

Celoxica provides a C-based hardware design language, Handel-C, and the DK Design Suite that can be customized for specific HPRC platforms such as the Cray-XD1 and/or SGI Altix/RASC systems [3, 4]. DK Design Suite unifies system verification, hardware/software codesign, and Handel-C synthesis in a GUI-based

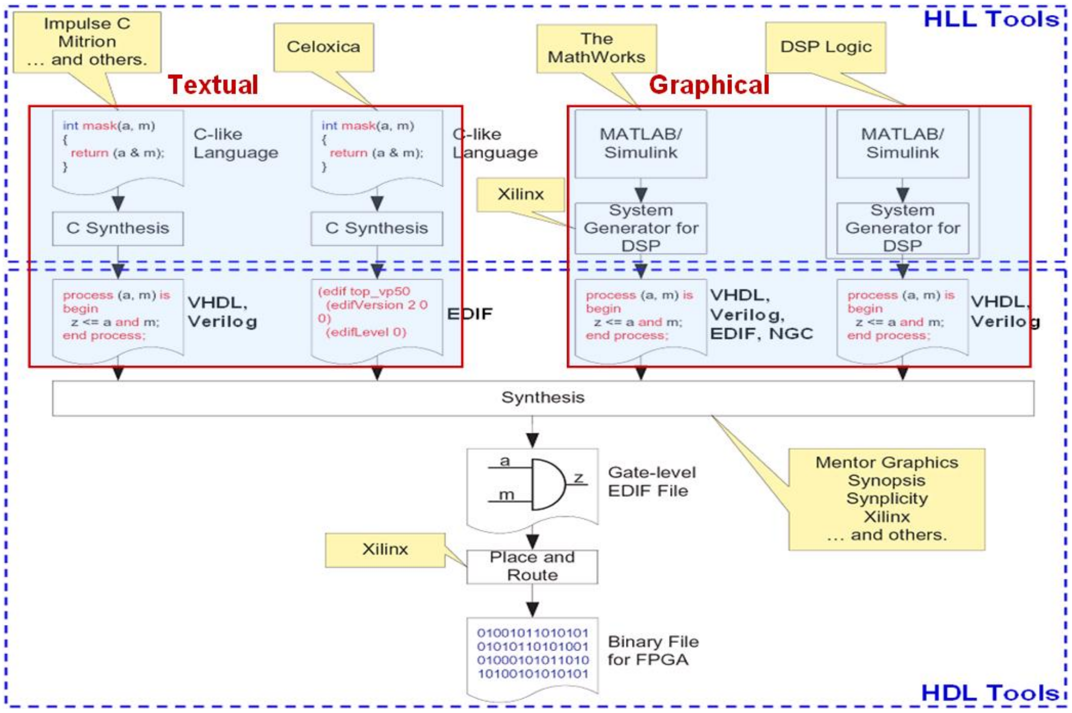


Fig. 1. Development flow of high-level tools [3].

development environment. The customization includes a Platform Support Library (PSL), which includes Handel-C interfaces to the underlying HPRC platform [6].

Mitrionics provides a C-based hardware design language, Mitrion-C, and an abstract machine, the Mitrion Virtual Processor (MVP), which allows the users to develop portable high-level code for FPGA applications. The Mitrion architecture uses a data-driven representation of the program, which the tools map onto programmable logic. Mitrion-C is an ANSI C-based functional language. Mitrion-C programming language is an implicitly parallel programming language with syntax similar to C. The language centers on parallelism and data-dependencies. In contrast, traditional languages are sequential and center on order-of-execution. In Mitrion-C there is no order-of-execution; any operation may be executed as soon as its data-dependencies are fulfilled. Mitrion-C is a Single-Assignment language (variables may only be assigned once in a scope) in order to prevent variables from having different values within the same scope. Software written in the Mitrion-C programming language is compiled into a configuration of the MVP. The Mitrion Virtual Processor is a fine-grain, massively parallel, reconfigurable soft-core processor [7].

The Xilinx System Generator (SysGen) for DSP tool uses a somewhat different approach for designing digital signal processing (DSP) blocks. It integrates with the MATLAB and Simulink packages from MathWorks to allow users to design the algorithmic block of their applications in the MATLAB GUI environment [8].

DSPlogic provides the Rapid Reconfigurable Computing Development Kit (RC Toolbox), which integrates with MATLAB and Simulink from MathWorks and with Xilinx tools. It allows users to design, verify, and build the FPGA logic for DSP applications entirely within the MATLAB/Simulink environment. The tool also includes the Reconfigurable Computing I/O (RCIO) library, which provides a portable application programming interface for communications between the software application that runs on the host processor(s) and the attached FPGAs. DSPLogic RC Toolbox is a combined graphical and text-based programming environment for HPRC application development. Blocks from the DSPlogic RC blockset and Xilinx System Generator are used to create a data flow diagram. Existing VHDL designs may also be incorporated using System Generator's HDL co-simulation capabilities [9].

The Carte-C (Carte-Fortran) development environment is somewhat different from the above mentioned flows in the sense that it is tightly integrated with SRC systems. It is a C-based (Fortran- based) environment that allows the users to program their applications in standard C (Fortran) with the aid of a library of pre-synthesized hardware functions. There are two types of application source files to be compiled, one that targets the microprocessor and another that targets the reconfigurable processor. Since users often wish to extend the built-in set of operators, the compiler allows users to integrate their own VHDL/Verilog macros. The environment also provides a means for debugging and verification [10].

2.1 HLL Tool Taxonomy

After reviewing the literature of HLLs, we recognized the need for a taxonomy of their programming models that would provide a useful means for the characterization of the differences among them. Fig. 2 shows our taxonomy of the programming models of the different HLLs. The programming model can be defined as the hardware abstract view presented to the programmer by the programming tool. Thus, a programming model defines which parts of the hardware architecture will become visible to the programmer and be under his/her direct control.

In general, HLLs can be categorized as either imperative or dataflow programming paradigms. This is mainly dependent on how parallelism is expressed and/or extracted. In imperative paradigms parallelism is non-native and expressed in an explicit manner which is solely the user's responsibility. In other words, in imperative languages, everything is sequential unless

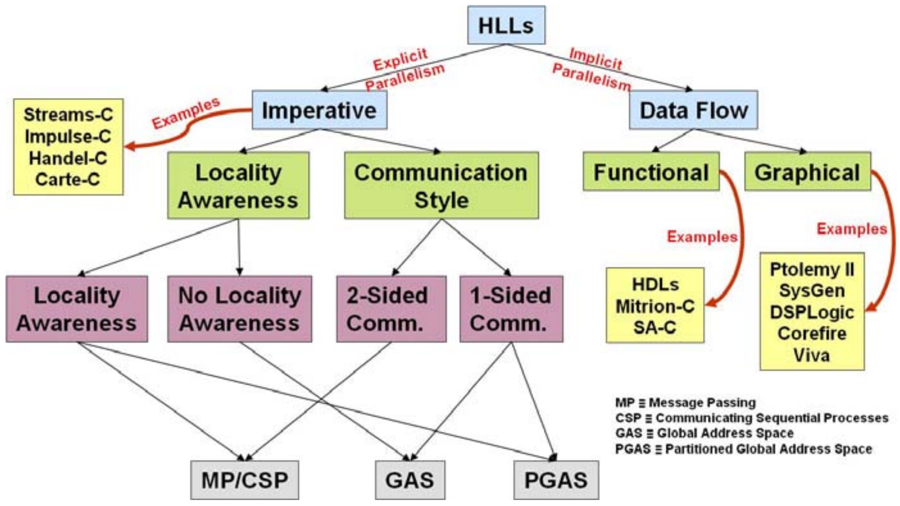


Fig. 2. Taxonomy of HLL programming models

otherwise stated. On the other hand, in dataflow paradigms parallelism is native and expressed in an implicit manner which is hidden from the user. In other words, in dataflow languages, everything is parallel unless otherwise stated.

Dataflow paradigms, as shown in Fig. 2, can be further divided into two subcategories, namely functional and graphical paradigms. Functional languages are basically the text-based versions of dataflow paradigms as opposed to the graphical version. In general, functional HLLs are characterized as being single-assignment languages. In light of this, HDLs, i.e. VHDL and/or Verilog, fall under this category. Examples of this family are Mitrion-C, SA-C, etc. Examples of graphical dataflow languages include SysGen, DSPLogic, CoreFire, Viva, etc.

Imperative paradigms, on the other hand, include languages such as Streams-C, Impulse-C, Handel-C, Carte-C, etc. However, because parallelism is nonnative, locality awareness and communication style, from an HPC perspective, become issues that are difficult to express in imperative paradigms. These issues are resolved by either the introduction of new extensions to the language, as in the case of Handel-C, or by the insertion of compiler directives/pragmas, as in the case of Impulse-C and Carte-C. In addition, concepts such as Message Passing (MP), Communication Sequential Processes (CSP), Global Address Space (GAS), and Partitioned Global Address Space (PGAS), are commonly found among the plethora of languages that fall under imperative paradigms. Fig. 2 shows how those languages deal with communication issues as well as with locality awareness. For example, languages such as Streams-C and Impulse-C provide a 2-sided, i.e. send-and-receive, communication style and also show an awareness of data locality through the MP/CSP model. The MP/CSP model is also supported by Carte-C and Handel-C. In addition, the GAS and/or PGAS models are also implicitly enabled by Carte-C and other languages, but not explicitly supported.

3 Related Work

The objective of this work is to formalize a statistical framework to evaluate various HLL features/attributes to characterize and compare different high-level design languages/methodologies. To achieve this objective our approach is based on leveraging previous work and concepts that were introduced, and proved useful to us, in similar investigations. For example, Holland [11] reviewed some C-based HLLs and highlighted some of the differences among them. Similarly, Edwards [12] discussed the challenges of synthesizing hardware from C-like languages. Finally, our previous work [13, 14] provided a formal and empirical comparative analysis of HLLs along with experimental work conducted on Cray-XD1. The work presented here significantly extends the model in [13, 14] and leverages some of its terminology and concepts. It enables evaluation of language features as well as experimental metrics, both of which are termed as attributes in the model. Furthermore, in our investigation the elimination of biasing effects has been formalized based on statistical analysis and validation.

In our study we considered productivity as one of the evaluation metrics. The definition and model of productivity have been widely discussed in literature. For example, Sterling [15] in his "Special Theory of Productivity" defines it as utility divided by time, where utility being the useful work, e.g. operations. While Snir and Bader [16] defined productivity of a system as the time dependent utility of the answers it produces divided by the total lifetime cost. Kennedy et. al. [17] definition is tailored towards tool expressiveness and efficiency. Abstract expressiveness determines the programming tool development power and computational efficiency is the programming tool execution efficiency. Kepner [18, 19] combines all of the above ideas into a single synthesis formulation. In addition, Numrich [20] presents his generic performance metric based on computational action. He examines work as it evolves in time and computes computational action as the integral of the work function over time.

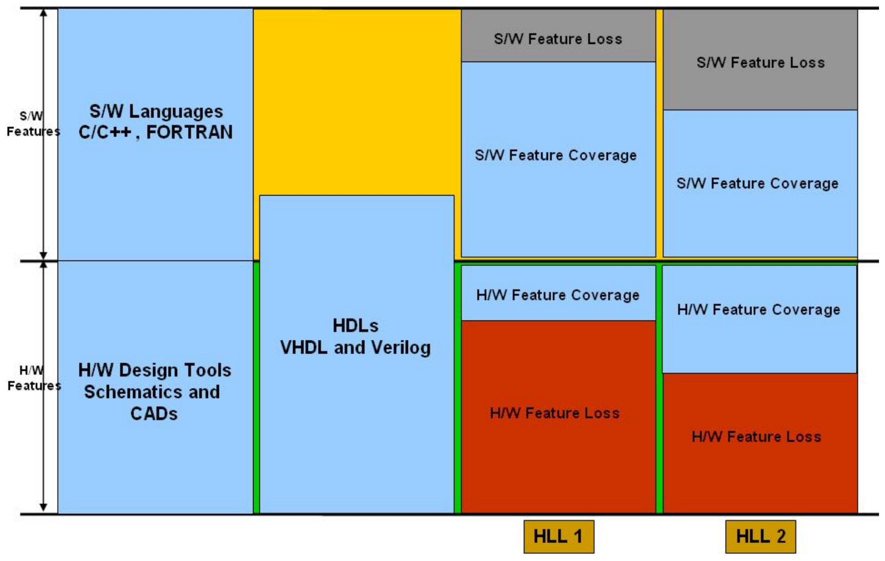


Fig. 3. HLLs as observations in the feature space.

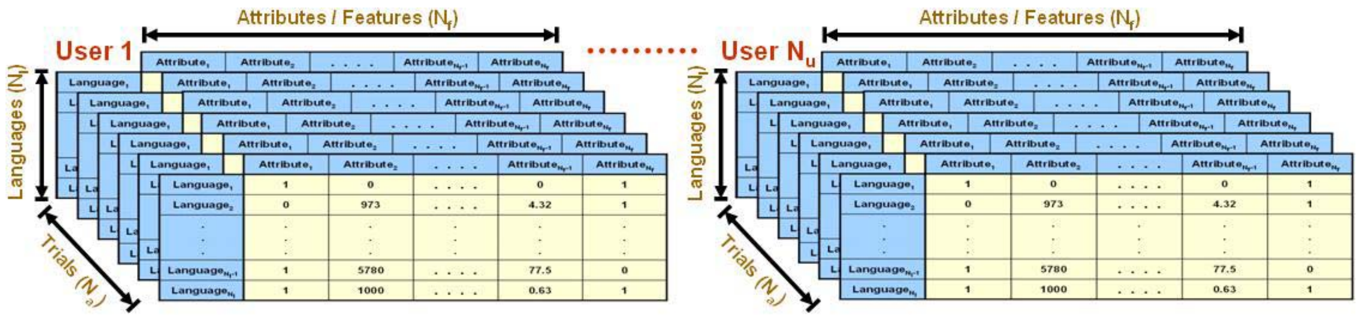
Our work leverages and builds off concepts from Numrich's research on performance and productivity metrics based on the principle of computational least action [20-24]. The metrics proposed here emphasize the rate of this computational work/effort. We call this rate as the work progress rate. Computational work or effort is the work done by the user-tool combination in traversing an abstract specifications space. As it will be shown later, the productivity metric emphasizing the total time-to-solution is a special case of this new metric.

4 HLL Evaluation Framework

4.1 Formalizing the Framework

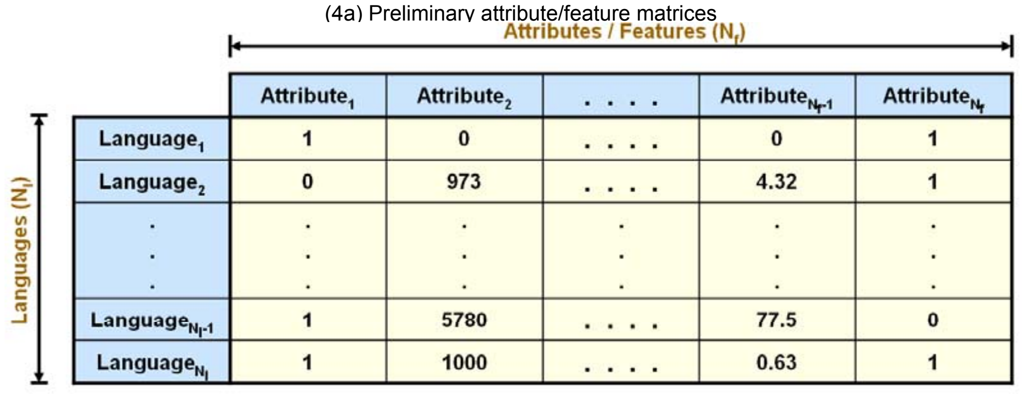
We start our formalization by visualizing the different HLLs as being observations of a space of attributes. These attributes can be either qualitative language features such as support for pointers and debugging capability, or experimental metrics such as development time and productivity. Therefore, in this multi-dimensional space of attributes each language can be considered as a single point or as an observation of that space. For example, Fig. 3 shows a case where this space has been hypothetically reduced to a two-dimensional space of, collectively, two orthogonal sets of attributes, namely software features and hardware features. Therefore, the evaluation of the different HLLs can be simply performed by comparing the different components along the dimensions of that space, e.g. feature coverage and/or feature loss. Fig. 3 shows a pictorial comparison of two hypothetical HLLs with different degrees of feature coverage and/or loss.

It is worth mentioning that in developing our framework we needed to minimize certain biasing effects associated with small sample sizes. For example, biasing effects may include previous user experience and knowledge of a specific language and/or design methodology. Therefore, we formalized our framework by instrumenting a normalization mechanism through which each user is required to provide three different observations of the same trial (application). In other words, each user develops the same application in three different implementations (languages). The first implementation is performed using the language under



1) Collect data points for all user trials through different applications
2) Normalize to minimize user's experience effects
3) Calculate the average space observation for each user

(4a) Preliminary attribute/feature matrices


Attributes / Features (N_f)
Languages (N_l)

	Attribute$_1$	Attribute$_2$	. . . .	Attribute$_{N_f-1}$	Attribute$_{N_f}$
Language$_1$	1	0	. . . .	0	1
Language$_2$	0	973	. . . .	4.32	1
⋮	⋮	⋮	⋮	⋮	⋮
Language$_{N_l-1}$	1	5780	. . . .	77.5	0
Language$_{N_l}$	1	1000	. . . .	0.63	1

4) Calculate the average space observation for all users

$$s_{l,f} = \frac{1}{N_u}\sum_{u=1}^{N_u} s_{u,l,f} \Rightarrow s_{l,f} = \frac{1}{N_u N_a}\sum_{u=1}^{N_u}\sum_{a=1}^{N_a}\left(\frac{v_{a,u,l,f}-v^{\min}_{a,u,l_0,f}}{v^{\max}_{a,u,l_1,f}-v^{\min}_{a,u,l_0,f}}\right)$$

$$where\ \left\{\ a\in[1,N_a]\ ,\ u\in[1,N_u]\ ,\ l\in[1,N_l]\ ,\ f\in[1,N_f]\ \right\},\ and$$

$$\left\{\ v^{\min}_{a,u,l_0,f}=\min_{l=1}^{N_l}\left(v_{a,u,l,f}\right)\ ,\ v^{\max}_{a,u,l_1,f}=\max_{l=1}^{N_l}\left(v_{a,u,l,f}\right)\ \right\}$$

(4b) Final attribute/feature matrix
Fig. 4. Formalizing the scoring mechanism

consideration, while the other two are performed using both pure software, e.g. in C/C++, and pure hardware, e.g. in VHDL/Verilog. These two other observations (implementations and/or languages) serve as extreme reference points that can minimize the biasing effects of user experience and knowledge of a given language. In other words, each user observation for a certain trial is normalized to his/her own experience making the observations more language-specific rather than being user-specific measurements. Fig. 4 summarizes the steps involved in our formal methodology for conducting the experimental work. Fig. 4(a) shows the preliminary formulation of attribute matrices leading to the final attribute matrix shown in Fig. 4(b). Each row in the final attributes matrix represents an observation (language) projected in the multi-dimensional space of attributes/features.

Based on the above discussion, we introduce the following notations in order to quantify our concepts:

- N_f is the total number of attributes/features, i.e. the dimension of the attribute space
- N_l is the total number of languages
- N_u is the total number of independent users involved in the experiments
- N_a is the total number of applications developed by each user, i.e. the number of trials of the attribute space for each user
- $v_{a,u,l,f}$ is the value of attribute f for language l as observed by user u when developing application a
- $v^{min}_{a,u,l,f}$ is the minimum value of attribute f for reference language l_0 as observed by user u when developing application a

$$v^{\min}_{a,u,l_0,f} = \min_{l=1}^{N_l}\left(v_{a,u,l,f}\right) \tag{1}$$

- $v^{max}_{a,u,l,f}$ is the maximum value of attribute f for reference language l_1 as observed by user u when developing application a

$$v^{\max}_{a,u,l_1,f} = \max_{l=1}^{N_l}\left(v_{a,u,l,f}\right) \tag{2}$$

- $s_{a,u,l,f}$ is the normalized value of attribute f for language l, with respect to reference language l_0 and language l_1, as observed by user u when developing application a

$$s_{a,u,l,f} = \frac{v_{a,u,l,f} - v^{\min}_{a,u,l_0,f}}{v^{\max}_{a,u,l_1,f} - v^{\min}_{a,u,l_0,f}} \tag{3}$$

- $s_{u,l,f}$ is the average normalized value of attribute f for language l, with respect to reference language l_0 and language l_1, as observed by user u across all trials (applications), i.e. the average space observation for each user

$$s_{u,l,f} = \frac{1}{N_a}\sum_{a=1}^{N_a} s_{a,u,l,f} = \frac{1}{N_a}\sum_{a=1}^{N_a}\left(\frac{v_{a,u,l,f} - v^{\min}_{a,u,l_0,f}}{v^{\max}_{a,u,l_1,f} - v^{\min}_{a,u,l_0,f}}\right) \tag{4}$$

- $s_{l,f}$ is the average normalized value of attribute f for language l, with respect to reference language l_0 and language l_1, as observed by all users across all trials (applications), i.e. the average space observation for all users

$$s_{l,f} = \frac{1}{N_u}\sum_{u=1}^{N_u} s_{u,l,f}$$

$$\Rightarrow s_{l,f} = \frac{1}{N_u N_a}\sum_{u=1}^{N_u}\sum_{a=1}^{N_a}\left(\frac{v_{a,u,l,f} - v^{\min}_{a,u,l_0,f}}{v^{\max}_{a,u,l_1,f} - v^{\min}_{a,u,l_0,f}}\right) \tag{5}$$

$$\textit{where}\quad \left\{\ a \in [1, N_a]\ ,\ u \in [1, N_u]\ ,\ l \in [1, N_l]\ ,\ f \in [1, N_f]\ \right\}$$

4.2 Validating the Framework

We statistically validated the fairness of our framework by applying our formulation to the evaluation metrics (attributes), i.e. ease-of-use and efficiency, as proposed and defined in [13]. This validation was performed by the replacement of all entries in the preliminary attribute matrices, i.e. attribute value $v_{a,u,l,f}$, with independent and identically distributed random variables. The random variables were uniformly distributed. After applying our framework through equations (1-5), we compared the theoretical expected values of the metrics of evaluation with the observed values. We found those to be almost identical. Furthermore, the variance of the data points was measured to be minimum and consistent with the theoretical expectations, see Fig. 5. In other words, the different hypothetical languages (observations) in the attribute space were closely clustered with minimum relative dispersion around the expected value. This proved to us the fairness of the proposed framework as well as the minimization of biasing effects towards certain languages over others. Fig. 5 shows our findings with this respect. One can also note the relative placement of the reference languages as two extremes.

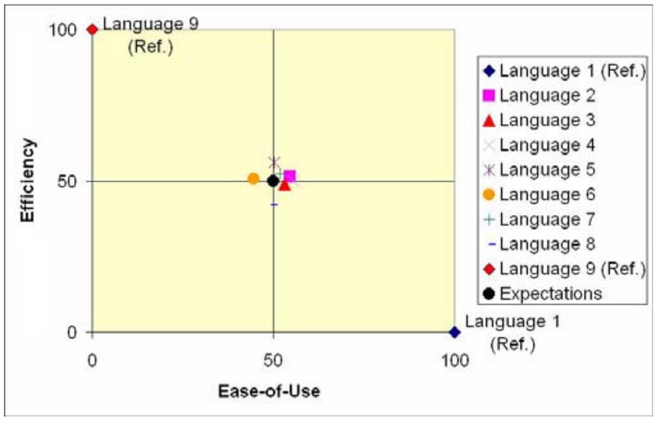


Fig. 5. Validation of the formal framework.

4.3 Metrics of Evaluation

Having established those top-level guidelines for the framework, different design methodologies, as mentioned earlier, can be evaluated by comparing their components along the dimensions of the attribute space. In order to calculate each attribute (metric) value for a particular methodology, the design process needs to be analyzed in more details. Adopting a black box approach, design methodologies and their

Inputs
(Weighted Specifications)
Design Methodology
Outputs
(Solutions Close to Input Specifications)

Fig. 6. Black box representation of design methodologies.

corresponding tools/languages need a quantitative representation of their inputs and the corresponding outcome. Inputs can be represented as a set of requirements/specifications, and a corresponding set of preferences/weights adjustable along the design process. Outputs can be represented as a set of solutions where the target solution would be the preferred input specifications, see Fig. 6. In other words, given a set of specifications, S = *{power, resources, speed, etc.}*, with their allowable ranges, S_{min} = *{minimum power, minimum resources, minimum speed, etc.}* and S_{max} = *{maximum power, maximum resources, maximum speed, etc.}*, and also given their corresponding weights or preference W = *{power weight, resources weight, speed weight, etc.}*, the goal is to achieve a target set of specifications, S_{target} = *{target power, target resource utilization, target frequency, etc.}*. More formally, this can be represented as a multi-dimensional space of specifications whose basis can be described by the following vector representation:

$$\vec{S} = \begin{bmatrix} s_1 \\ s_2 \\ \cdots \\ s_N \end{bmatrix}, \ \overrightarrow{S_{\min}} = \begin{bmatrix} s_{1_{\min}} \\ s_{2_{\min}} \\ \cdots \\ s_{N_{\min}} \end{bmatrix}, \ \overrightarrow{S_{\max}} = \begin{bmatrix} s_{1_{\max}} \\ s_{2_{\max}} \\ \cdots \\ s_{N_{\max}} \end{bmatrix}, \ \overrightarrow{S_{target}} = \begin{bmatrix} s_{1_{target}} \\ s_{2_{target}} \\ \cdots \\ s_{N_{target}} \end{bmatrix},$$

$$\vec{W} = \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_N \end{bmatrix} \tag{6}$$

where

$N \equiv$ Dimensionality of the specification (solution) space

Due to the different scales and units for each component of the specification vector, a normalized and unitless representation of the space is desirable. Therefore, a mapping function is needed to establish the correspondence relation between the original space and the normalized space. The mapping is done such that more desirable solutions always have higher coordinates in the normalized space, see equation (7). The design problem becomes now a search process for the target solution. For specifications that need to be maximized, e.g. frequency, search in the normalized space moves in the same direction as in the original space. For specifications that need to be minimized, e.g. area and/or power, search in the normalized space moves in the opposite direction to that in the original space, see Fig. 7. It can be seen in Fig. 7 that the most desirable solution, i.e. optimal solution with respect to the given range of specifications, is located at the positive extreme of the normalized space.

Based on this representation, design methodologies/tools can be evaluated by analyzing the

$$x_i = \left\{ \begin{array}{ll} w_i \times \dfrac{s_i - s_{i_{\min}}}{s_{i_{\max}} - s_{i_{\min}}}, & s_i \text{ to be maximized} \\ w_i \times \dfrac{s_{i_{\max}} - s_i}{s_{i_{\max}} - s_{i_{\min}}}, & s_i \text{ to be minimized} \end{array} \right\}, \ i = 1,2,....,N$$

$$\Delta x_i = \left\{ \begin{array}{ll} w_i \times \dfrac{\Delta s_i}{s_{i_{\max}} - s_{i_{\min}}}, & s_i \text{ to be maximized} \\ w_i \times \dfrac{-\Delta s_i}{s_{i_{\max}} - s_{i_{\min}}}, & s_i \text{ to be minimized} \end{array} \right\}, \ i = 1,2,....,N \tag{7}$$

$$\vec{X} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_N \end{bmatrix}, \ \overrightarrow{X_{optimal}} = \begin{bmatrix} x_{1_{optimal}} \\ x_{2_{optimal}} \\ \cdots \\ x_{N_{optimal}} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_N \end{bmatrix} = \vec{W}, \ \overrightarrow{X_{target}} = \begin{bmatrix} x_{1_{target}} \\ x_{2_{target}} \\ \cdots \\ x_{N_{target}} \end{bmatrix}$$

where

$\vec{X} \equiv$ Solution position vector in the normalized space

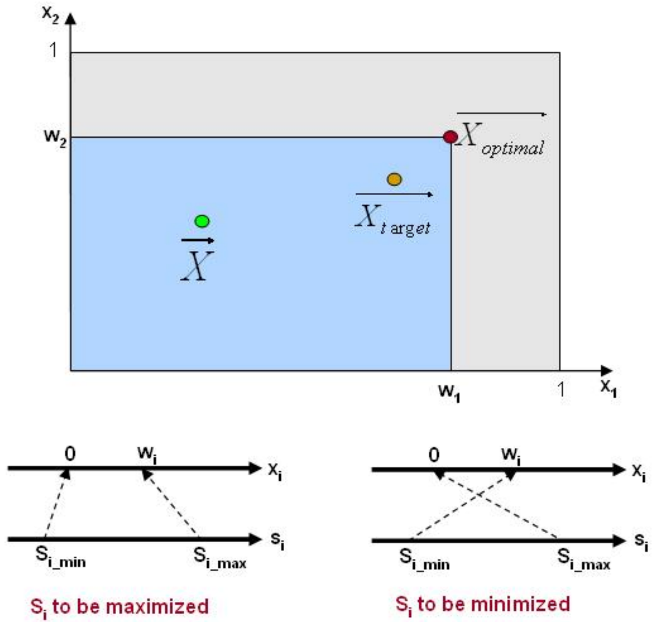


Fig. 7. Space for the design problem (solution search-process)

time evolution of the search path towards the target solution. Different methodologies differ in the selection of the search path, i.e. location of candidate solutions in the search space. They also differ in the manner through which the search path is being time sampled, i.e. total number of iterations and/or the candidate solutions, M, along the search path. In other words, any given design methodology can be characterized by the instantaneous progress of the search path as a function of time. Under this representation, several useful quantities can be defined and used in studying the design process. More specifically, the target vector, $\boldsymbol{T_k}$, at time sample (iteration) k, can be defined as the distance from a given candidate solution, $\boldsymbol{X_k}$, to the target solution, $\boldsymbol{X_{target}}$. $\boldsymbol{T_k}$ measures the closeness of a given candidate solution to the final target solution. Also, displacement vector, $\boldsymbol{D_k}$, at time sample (iteration) k, which is the shift between consecutive candidates in the solution space measures the distance traversed along the search path from a candidate solution to the following one. Moreover, sensitivity, λ_k, at time sample (iteration) k, defined as the shift between consecutive candidate solutions in the direction of the target solution, i.e. the projection of $\boldsymbol{D_k}$ onto $\boldsymbol{T_k}$, represents a measure for the convergence towards the target solution. φ_k is the angle of projection or the angle of deviation of the tool away from the target. Additionally, the velocity vector, $\boldsymbol{\upsilon_k}$, at time sample (iteration) k, represents the instantaneous traversal speed along the search path from a candidate solution to the following one. The definitions of these quantities are given in equation (8) and shown in Fig. 8.

$$\left\{\begin{matrix} \overrightarrow{X_k} = \overrightarrow{X(t_k^{'})} \ , \ \overrightarrow{D_k} = \overrightarrow{D(t_k^{'})} \\ \overrightarrow{T_k} = \overrightarrow{T(t_k^{'})} \ , \ \overrightarrow{\upsilon_k} = \overrightarrow{\upsilon(t_k^{'})} \ , \ \lambda_k = \lambda(t_k^{'}) \end{matrix}\right\}$$

$$\left.\begin{matrix} \overrightarrow{T_k} = \overrightarrow{X_{target}} - \overrightarrow{X_k} \\ \overrightarrow{D_k} = \overrightarrow{X_{k+1}} - \overrightarrow{X_k} \\ \\ \lambda_k = \dfrac{\overrightarrow{D_k} \cdot \overrightarrow{T_k}}{\left\|\overrightarrow{T_k}\right\|} = \dfrac{\overrightarrow{D_k} \cdot \overrightarrow{T_k}}{T_k} = D_k \cos(\varphi_k) \end{matrix}\right\} \begin{matrix} , \ 0 \le k < M \\ \\ , \ and \ \overrightarrow{X_0} = \vec{0} \end{matrix} \qquad (8)$$

where

$\vec{A} \cdot \vec{B} \equiv$ Dot product of vectors $\vec{A}$ and $\vec{B}$

$\left\|\vec{A}\right\| \equiv A \equiv$ Length of vector $\vec{A}$

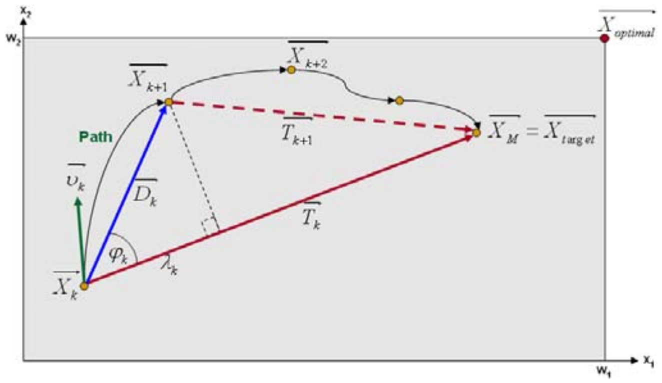


Fig. 8. Time evolution of the search path

Finally, it is essential to consider two activities that are typically associated with the design process, namely the user activity and the tool activity. These two activities are disjoint and mutually exclusive in time (consecutive activities), however, they are dependent. The design process typically starts with a user activity, i.e. initial design given to the tool, and ends with a tool activity after which the target solution at minimum is reached, see Fig. 8. The design process alternates (iterates) between the two exclusive activities, see Fig. 9. The user activity can be viewed as corrective to the tool deviations away from the target solution. Therefore, if we assume that t_M is the total development time, t_k represents the time spent after k iterations, Δt_k is the time period between two consecutive iterations, the expression given by equation (9) can be used to describe the timeline of the user and tool activities. It can be seen in Fig. 9 that Δt_k is the summation of the time spent by the user Δt_k^{user} and the time spent by the tool Δt_k^{tool} to generate the outputs of iteration k. Note that the shaded and un-shaded regions in Fig. 9 represent respectively the active and idle time periods for either the user or the tool.

$$t_{k+1} = t_k + \Delta t_k = t_k + \left(\Delta t_k^{user} + \Delta t_k^{tool}\right) = t_k^{'} + \Delta t_k^{tool} \ , \ 0 \le k < M \qquad (9)$$

where

$$t_k^{'} = t_k + \Delta t_k^{user}$$

4.3.1 Productivity

Productivity, Ψ, is usually defined as utility, U, per cost, C as shown in equation (10) [15-22].

$$\Psi = \frac{U}{C} \qquad (10)$$

Utility is typically a function of achieved design objectives such as performance, power, and area, etc. Cost is the development cost expressed in either development time or proportional time equivalents such as man-hours, dollars etc. When the total development time, t_M, is considered as the cost, productivity gives a measure of how fast a desired solution was obtained which can be expressed as follows:

$$\Psi = \frac{U}{C} = \frac{T_0}{t_M} = \frac{X_{target}}{t_M} \equiv \upsilon_{target} \qquad (11)$$

Relative productivity of two methodologies is the ratio of individual productivities. Thus for two methodologies attaining the same design objectives the relative productivity is the inverse ratio of their development costs, see equation (12).

$$\Psi_{1/2} = \frac{\Psi_1}{\Psi_2} = \frac{C_2}{C_1} = \frac{t_{M_2}}{t_{M_1}} \qquad (12)$$

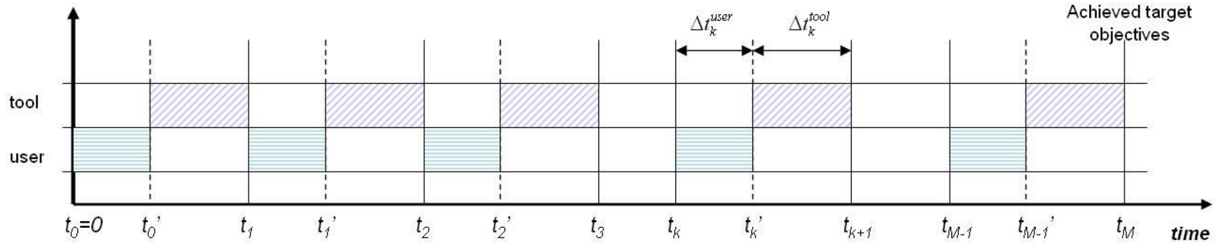


Fig. 9. Activities of the design process

Also, based on this metric two methodologies achieving the same design objectives with same development costs have the same productivity. Although this is a logical inference, the metric has no notion of the user effort, tool maturity, or resource utilization efficiency during the development process. Two tools could have taken two completely different paths in the objective search space to reach the target in the same amount of time, and have different levels of user involvement, design iterations, tool algorithm complexities, and compute resource requirements. The productivity metric based on time-to-solution cannot capture these facets of development process. Hence evaluating methodologies based on solely the productivity metric is incomplete at best. To address this we present a new metric, i.e. *work progress rate*, as defined below.

4.3.2 Work Progress Rate

The purpose of work progress rate metric is to capture the efficiency of resource utilization, by the user and the tool combined, at discrete time steps along the search path progressing towards the target specifications. The metric draws from the principles of classical mechanics based on Newton's laws of motion. It evaluates the computational effort exerted by the user and the tool in moving towards the target specifications along the search path in the specification space. Based on the fact that the user and the tool computational efforts are mutually exclusive activities as discussed earlier, the instantaneous computational effort, $\boldsymbol{E_k}$, in any given time period Δt_k of iteration k, can be expressed as a vector with two components. The two components are the user effort, E_k^{user}, and the tool effort, E_k^{tool}, as shown in equation (13).

$$\overrightarrow{E_k} = \begin{bmatrix} E_k^{user} \\ E_k^{tool} \end{bmatrix} \tag{13}$$

The instantaneous work progress rate, $\boldsymbol{\Gamma_k}$, in any given time period Δt_k is the rate of exerting computational effort in order to move along the search path in that time period. It is also a vector with two components, the user work progress rate, Γ_k^{user}, and the tool work progress rate, Γ_k^{tool}, as shown in equations (14a) and (14b).

$$\overrightarrow{\Gamma_k} \equiv \begin{bmatrix} \Gamma_k^{user} \\ \Gamma_k^{tool} \end{bmatrix} \equiv \frac{d\overrightarrow{E_k}}{dt_k} \cong \frac{\Delta\overrightarrow{E_k}}{\Delta t_k} \tag{14a}$$

$$\Delta\overrightarrow{E_k} \equiv \frac{\partial\overrightarrow{E_k}}{\partial t_k^{user}}\Delta t_k^{user} + \frac{\partial\overrightarrow{E_k}}{\partial t_k^{tool}}\Delta t_k^{tool}$$

$$\Delta\overrightarrow{E_k} = \frac{\partial}{\partial t_k^{user}}\begin{bmatrix} E_k^{user} \\ E_k^{tool} \end{bmatrix}\Delta t_k^{user} + \frac{\partial}{\partial t_k^{tool}}\begin{bmatrix} E_k^{user} \\ E_k^{tool} \end{bmatrix}\Delta t_k^{tool}$$

$$\Delta\overrightarrow{E_k} = \begin{bmatrix} \dfrac{dE_k^{user}}{dt_k^{user}} \\ 0 \end{bmatrix}\Delta t_k^{user} + \begin{bmatrix} 0 \\ \dfrac{dE_k^{tool}}{dt_k^{tool}} \end{bmatrix}\Delta t_k^{tool} = \begin{bmatrix} \dfrac{dE_k^{user}}{dt_k^{user}}\Delta t_k^{user} \\ \dfrac{dE_k^{tool}}{dt_k^{tool}}\Delta t_k^{tool} \end{bmatrix}$$

$$\Rightarrow \Gamma_k^{user} = \frac{dE_k^{user}}{dt_k^{user}} \cong \frac{E_k^{user}}{\Delta t_k^{user}}, \ \ and\ \Gamma_k^{tool} = \frac{dE_k^{tool}}{dt_k^{tool}} \cong \frac{E_k^{tool}}{\Delta t_k^{tool}} \tag{14b}$$

The overall work progress rate, $\boldsymbol{\Gamma}$, can be defined as the statistical average of the instantaneous rates across all iterations along the search path in the specifications space. This can be described as shown in equation (15).

$$\vec{\Gamma} \equiv \frac{1}{M}\sum_{k=0}^{M-1}\overrightarrow{\Gamma_k} = \frac{1}{M}\sum_{k=0}^{M-1}\left(\begin{bmatrix} \dfrac{E_k^{user}}{\Delta t_k^{user}} \\ \dfrac{E_k^{tool}}{\Delta t_k^{tool}} \end{bmatrix}\right)$$

$$\Rightarrow \vec{\Gamma} \equiv \begin{bmatrix} \Gamma^{user} \\ \Gamma^{tool} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{M}\sum\limits_{k=0}^{M-1}\left(\dfrac{E_k^{user}}{\Delta t_k^{user}}\right) \\ \dfrac{1}{M}\sum\limits_{k=0}^{M-1}\left(\dfrac{E_k^{tool}}{\Delta t_k^{tool}}\right) \end{bmatrix} \tag{15}$$

Based on equation (15), two methodologies achieving the same design objectives, in equal amount of time, along different paths and across different number of iterations can have different work progress rates even though their productivities as given by equation (11), will be the same. This depends on the user and tool computational efforts along the search path. Thus, the work progress rate metric allows us to compare two methodologies not only based on how fast the target objectives were achieved, but how efficiently the resources were used along the way to achieve the target objectives.

It is necessary at this point to analyze and model the computational efforts exerted by both the user and the tool. In our model we leverage the concept of computational force as introduced and defined by Numrich [23, 24]. We will also define the computational effort as the work, as defined in classical mechanics, done by a force field to move an object between two positions in a given space.

In our model, the user, at any given time period Δt_k of iteration k, expends effort at the beginning of the iteration to maximize the displacement in the specifications space towards the target solution. Hence the computational effort expended by the user can be modeled by the work done by the user to push the tool to move from a given position (candidate solution), $\boldsymbol{X_k}$, in the specifications space to the next position, $\boldsymbol{X_{k+1}}$. Because the target solution is always known to the user at any point of time, the user's computational force is assumed to be pointing towards the target position. In other words, the computational force, $\boldsymbol{F_k^{user}}$, applied by the user is an impact force used to give the initial push to the tool to move in the specifications space and is aligned with the direction of the current target vector, $\boldsymbol{T_k}$. This is shown in Fig. 10 and expressed in equation (16). Similarly, the computational effort expended by the tool is the work done by the tool to cause the displacement $\boldsymbol{D_k}$ in the specifications space, see Fig. 10 and equation (17). The reader is reminded of the fact that the user and tool computational efforts, although being mutually dependent, are time exclusive activities and hence their computational forces cannot be used simultaneously to describe the dynamics of motion in the solution space, see equations (16) and (17). More specifically, the user, during the time interval Δt_k^{user}, exerts an effort only through his/her own computational force, ***F****k**user*, in the direction of the target vector, ***T****k*. After that time interval the tool receives an initial impact velocity, ***υ****0k**tool*, from the user and starts searching the space for a duration of *Δtk**tool*. It is assumed that the user starts his activity from rest, i.e. zero velocity, and continues his activity until he reaches a final velocity, ***υ****k**user*, after a time interval of *Δtk**user*. It is also assumed the user final velocity is transferred to the tool as an initial velocity, ***υ****0k**tool*, in a perfectly inelastic collision/impact. These dynamics are described through the equations of motion (18) and (19).

$$E_k^{user} = \overrightarrow{F_k^{user}} \cdot \overrightarrow{T_k} = F_k^{user} T_k \tag{16}$$

$$E_k^{tool} = \overrightarrow{F_k^{tool}} \cdot \overrightarrow{D_k} = F_k^{tool} D_k \cos\omega_k \tag{17}$$

$$\overrightarrow{F_k^{user}} \equiv m^{user} \frac{d\overrightarrow{\upsilon_k^{user}}}{dt_k^{user}} \cong m^{user} \frac{\Delta\overrightarrow{\upsilon_k^{user}}}{\Delta t_k^{user}} = m^{user} \frac{\overrightarrow{\upsilon_k^{user}} - \overrightarrow{\upsilon_{0k}^{user}}}{\Delta t_k^{user}}$$

$$\overrightarrow{F_k^{user}} = m^{user} \frac{\overrightarrow{\upsilon_k^{user}}}{\Delta t_k^{user}}$$

$$\Rightarrow \overrightarrow{\upsilon_k^{user}} = \frac{\overrightarrow{F_k^{user}}}{m^{user}} \Delta t_k^{user} \equiv \frac{\overrightarrow{T_k}}{\Delta t_k^{user}} \tag{18}$$

where

$m^{user} \equiv$ The user's inertial resistance

$$\overrightarrow{F_k^{tool}} \equiv m^{tool} \frac{d\overrightarrow{\upsilon_k^{tool}}}{dt_k^{tool}} \cong m^{tool} \frac{\Delta\overrightarrow{\upsilon_k^{tool}}}{\Delta t_k^{tool}} = m^{tool} \frac{\overrightarrow{\upsilon_k^{tool}} - \overrightarrow{\upsilon_{0k}^{tool}}}{\Delta t_k^{tool}}$$

$$\Rightarrow \overrightarrow{\upsilon_k^{tool}} = \overrightarrow{\upsilon_{0k}^{tool}} + \frac{\overrightarrow{F_k^{tool}}}{m^{tool}} \Delta t_k^{tool} \tag{19}$$

where

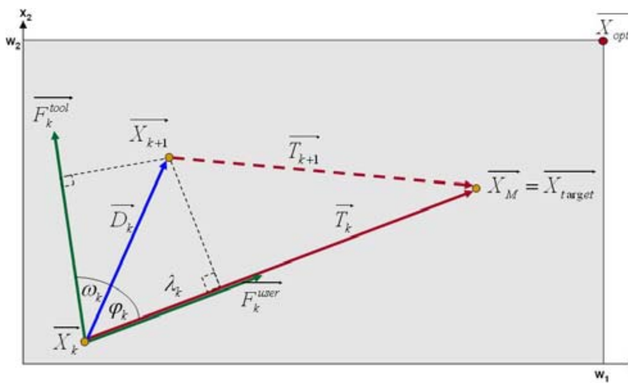
$$\overrightarrow{\upsilon_{0k}^{tool}} = \overrightarrow{\upsilon_k^{user}} = \frac{\overrightarrow{T_k}}{\Delta t_k^{user}} \equiv \left\{ \begin{array}{l} \text{The initial tool velocity in} \\ \text{the direction of the target} \\ \text{vector } \overrightarrow{T_k} \text{ due to computational} \\ \text{effort expended by the user} \end{array} \right.$$

$m^{tool} \equiv$ The tool's inertial resistance

The instantaneous work progress rate can now be given as:

$$\overrightarrow{\Gamma_k} \equiv \begin{bmatrix} \Gamma_k^{user} \\ \Gamma_k^{tool} \end{bmatrix} = \begin{bmatrix} \dfrac{E_k^{user}}{\Delta t_k^{user}} \\ \dfrac{E_k^{tool}}{\Delta t_k^{tool}} \end{bmatrix} = \begin{bmatrix} \dfrac{\overrightarrow{F_k^{user}} \cdot \overrightarrow{T_k}}{\Delta t_k^{user}} \\ \dfrac{\overrightarrow{F_k^{tool}} \cdot \overrightarrow{D_k}}{\Delta t_k^{tool}} \end{bmatrix} = \begin{bmatrix} \overrightarrow{F_k^{user}} \cdot \overrightarrow{\upsilon_k^{user}} \\ \overrightarrow{F_k^{tool}} \cdot \overrightarrow{\upsilon_k^{tool}} \end{bmatrix} \tag{20}$$

It is worth mentioning that the user final velocity, ***υ****k**user*, and hence the initial tool velocity, ***υ****0k**tool*, are proportional to the user's experience. For a given iteration target, ***T****k*, advanced users spend less time, *Δtk**user*, to debug and redevelop/modify their designs than novice users before they start their tools, see equations (18) and (19). On the other hand, least experienced users spend long periods of times, i.e. $\Delta t_k^{user} \to \infty$, with almost no guidance or corrective efforts to the tool, i.e. $\boldsymbol{\upsilon}_{0k}^{tool} = \boldsymbol{\upsilon}_k^{user} = 0$ and $\Gamma_k^{user}=0$, see equations (19) and (20).



x₂
w₂
$\overrightarrow{X}_{opt}$
$\overrightarrow{F_k^{tool}}$
$\overrightarrow{X}_{k+1}$
$\overrightarrow{T}_{k+1}$
$\overrightarrow{X_M} = \overrightarrow{X}_{target}$
$\overrightarrow{D_k}$
$\overrightarrow{T_k}$
ω_k
λ_k
φ_k
$\overrightarrow{F_k^{user}}$
$\overrightarrow{X_k}$
w₁

Fig. 10. User and tool computational forces.

As mentioned earlier, the user effort is always corrective to the tool effort. Therefore, it is desirable at this point to investigate the divergent behavior of the tool from that of the user's away from the target solution. In other words, it is essential to calculate the initial direction, *ωk*, of the tool computational force with respect to the final displacement vector, ***D****k*, see Fig. 10. *ωk* represents the angle with which the tool starts searching the space. Using equations (16)-(19) and the geometrical properties shown in Fig. 10, the following expression can be derived for calculating *ωk*:

$$\cos(\omega_k) = \frac{1 - \left(\dfrac{\lambda_k}{D_k}\right)^2 \left(\dfrac{\upsilon_{0k}^{tool} \Delta t_k^{tool}}{\lambda_k}\right)}{\sqrt{1 + \left(\dfrac{\lambda_k}{D_k}\right)^2 \left(\dfrac{\upsilon_{0k}^{tool} \Delta t_k^{tool}}{\lambda_k}\right)\left(\dfrac{\upsilon_{0k}^{tool} \Delta t_k^{tool}}{\lambda_k} - 2\right)}} \tag{21}$$

where

$$\frac{\lambda_k}{D_k} = \cos(\varphi_k)$$

Based on the proposed model, important characteristics of the development process can be understood by considering some special cases of equation (21). For example, at any design iteration *k*, when there is no initial push to the tool by the user, i.e. ***υ****0k**tool* =0 and $\Gamma_k^{user}=0$, due to his/her lack of experience putting together a good initial design/development, the tool is left on its own searching the design space with $\omega_k=0$, see equation (22a) and Fig. 10. This results in a random displacement, ***D****k*, the magnitude and direction of which are purely dependent on the computational force of the tool and on how efficient the tool is. On the other extreme, advanced users tend to give the tool the maximum guidance and/or corrective efforts, i.e. $\boldsymbol{\upsilon}_{0k}^{tool} \to \infty$, resulting in a pure resistive behavior of the tool opposing the user effort, i.e. $\omega_k=\pi - \varphi_k$, see equation (22b) and Fig. 10. Additionally, if the sensitivity λ_k, i.e. the distance traveled by the tool towards the target solution, is solely due to the user's first push, i.e. ($\boldsymbol{\upsilon}_{0k}^{tool}\Delta t_k^{tool}$), then the tool has not performed any useful work and its computational force has been orthogonal, i.e. $\omega_k=\pi/2 - \varphi_k$, to the target vector, see equation (22c) and Fig. 10.

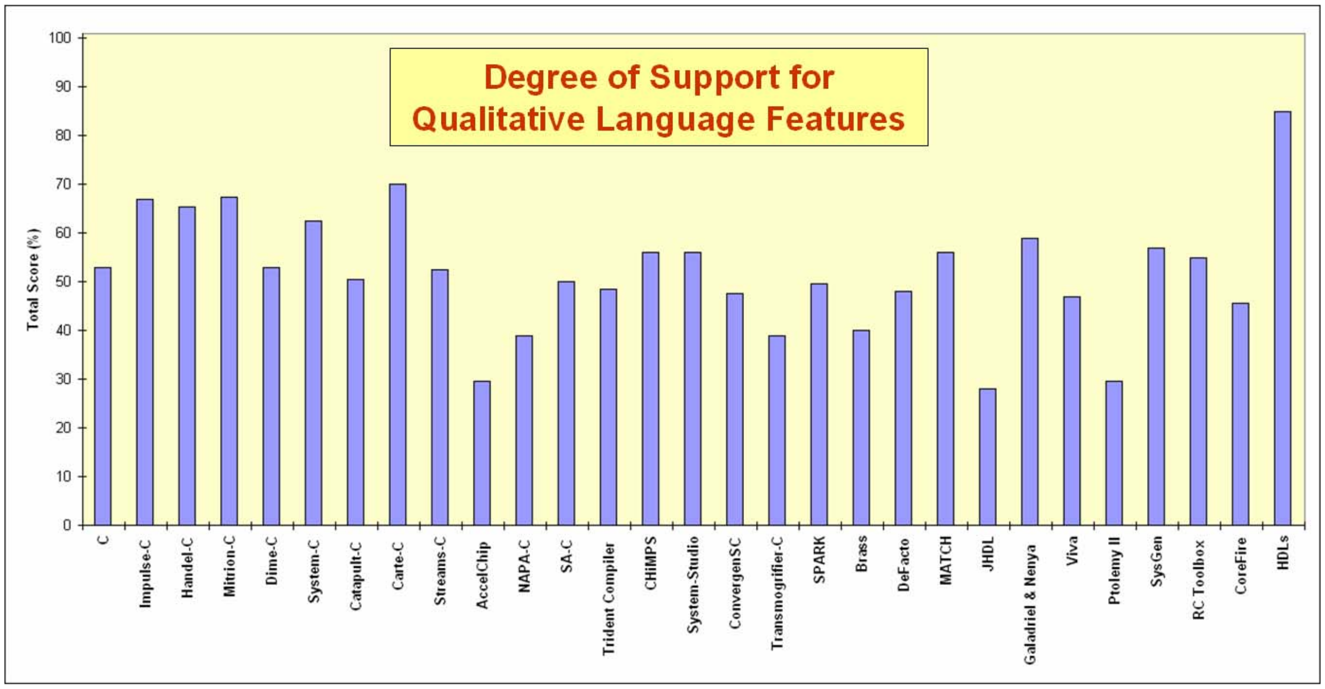


Fig. 11. Degree of support for qualitative language features

$$\lim_{\upsilon_{0k}^{tool}\to 0}(\omega_k)=0 \tag{22a}$$

$$\lim_{\upsilon_{0k}^{tool}\to \infty}(\omega_k)=\pi-\varphi_k \tag{22b}$$

$$\lim_{\frac{\upsilon_{0k}^{tool}\Delta t_k^{tool}}{\lambda_k}\to 1}(\omega_k)=\frac{\pi}{2}-\varphi_k \tag{22c}$$

5 Experimental Evaluation

The framework was applied to evaluate the support for qualitative language features of a subset of the HLLs from our review list. A number of useful language features are considered including behavioral hierarchy, structural hierarchy, supported architectures, degree of parallelism, locality exploration, portability, and dynamic memory allocation. The quantifying process is a simple scoring system that ranks the degree of support of each HLL tool for the specific qualitative language feature. The scoring mechanism allows weighting, see equation (23), to rate specific features higher according to evaluator preferences. In our case we used equal weights since we consider each of those features equally with no particular preference. Fig. 11 plots the final scores for the languages evaluated.

$$Score_l=100\times\left(\frac{\sum_{f=1}^{N_f}s_{l,f}\cdot w_f}{\sum_{f=1}^{N_f}s_f^{\max}\cdot w_f}\right),\ \ where\ l\in[1,N_l], \tag{23}$$

$$f\in[1,N_f],\ \ s_f^{\max}=\max_{l=1}^{N_l}(s_{l,f}),\ \ and\ s_{l,f}\in[0,s_f^{\max}]$$

For our experimental evaluation we selected the HLL tools that scored the highest in this scoring mechanism. We also considered representative high-level tools that were selected to represent imperative, functional, and graphical programming. The availability of these tools for experimentation was also a factor in our selection. The HLL tools selected for our experimental evaluation are listed in Table 2.

Table 2. Experimental Results

	Tool	Development Time (Hours)	Frequency (MHz)	Area (% Utilization)	Normalized Frequency (X1)	Normalized Area (X2)
1	C	3	0	100	0	0
2	Impulse-C	6	125	21.25	0.625	0.7875
3	Handel-C	7.5	200	20.25	1	0.7975
4	Carte-C	6.5	100	19.5	0.5	0.805
5	Mitrion-C	10.75	100	22.75	0.5	0.7725
6	SysGen	9	200	19	1	0.81
7	RC Toolbox	9	200	19	1	0.81
8	HDLs	15.25	200	16.75	1	0.8325

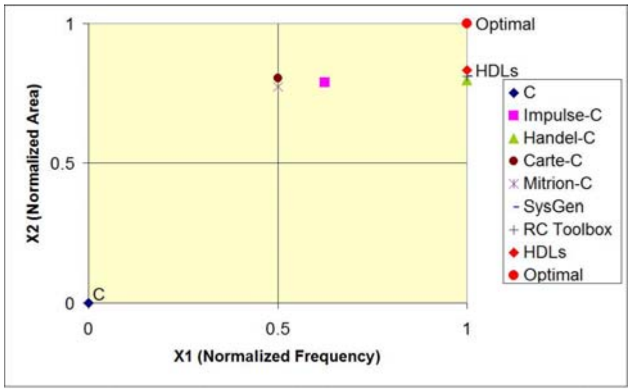


Fig. 12. HLL results plotted onto the specification space

Four workloads were selected for implementation using the selected HLL tools. The first workload is a simple pass-through implementation that reads input from the µP and sends it back unmodified. The purpose of this simple application is to measure the overhead caused by each tool on the FPGA with respect to the area utilization and also to measure the maximum clocking rates reached by each tool in the simplest of applications. This will give an initial and basic idea of the performance with each tool. The second application implemented is a discrete wavelet transform (DWT), [25, 26]. The third and fourth applications implemented are the data encryption standard (DES) and the DES breaking algorithms [27]. DWT and DES were selected as representative workloads of communication-intensive applications while DES breaker is a computational-intensive workload.

In conducting our experiments, users, mainly students and faculty members, with different levels of experience and backgrounds in computer science, computer engineering, and electrical engineering were selected. As mentioned earlier, we developed our framework such that certain biasing effects associated with small sample sizes are minimized. For example, biasing effects may include previous user experience and knowledge of a specific language and/or design methodology. Therefore, we formalized our framework by instrumenting a normalization mechanism through which each user is required to provide three different observations of the same trial (application). In other words, each user develops the same application in three different design paradigms (languages). The first of which is performed using the language under consideration, while the other two are performed using a pure software approach, e.g. in C/C++, and a pure hardware approach, e.g. in VHDL/Verilog. These two other observations serve as extreme reference points that can minimize the biasing effects of previous user programming experience which can range from software-centric programming experience to hardware-centric programming experience. In other words, in our experiments each user observation for a certain trial is normalized to his/her own experience making the observations more language-specific rather than being user-specific measurements.

5.1 Results

Table 2 shows the final attribute matrix for the four workloads implemented using the selected HLL tools. The reference implementations in HDLs and C are also included. The specifications space is assumed to be 2-dimensional with area and frequency as the specifications. Fig. 12 shows the frequency and area utilization achieved by each tool plotted on the normalized specification space. On the assumption that each tool generated the result in a single iteration, the work progress rate for each tool is equal to the productivity, see equations (11) and (15). Fig. 13 presents the results projected onto the attribute space while Fig. 14 plots the productivity of each tool.

We may observe from the experimental results that

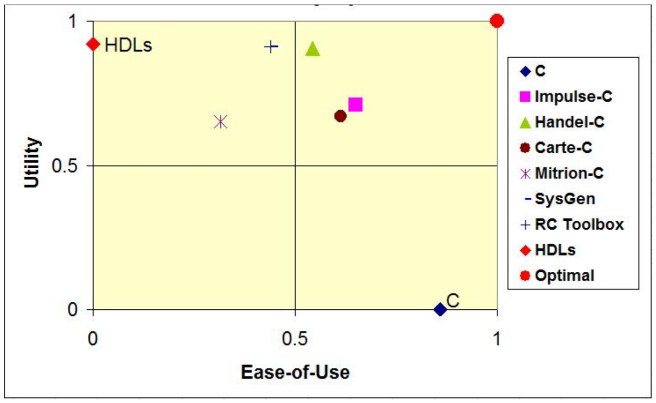


Fig. 13. HLL results plotted onto the attribute space.

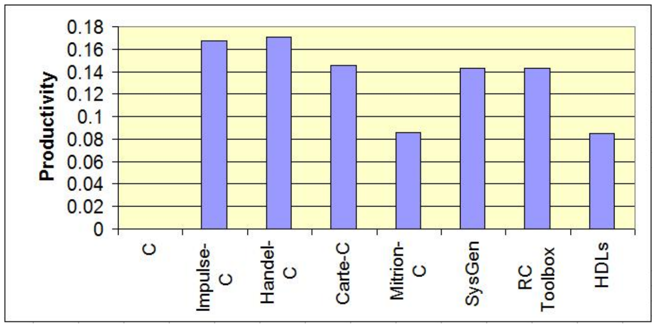


Fig. 14. Productivity of evaluated HLL tools.

imperative approaches proved to be the easiest to use while performing reasonably and comparably with standard HDL approaches. On the other hand, dataflow approaches, both functional and graphical, proved to achieve the high utility but were not as easy to use as imperative counterparts. Pure functional approaches proved to be the most difficult to use amongst the three approaches. Moreover, HDL approaches achieve the highest utility (close to optimal with this respect) but at the expense of being the most difficult to use for application developers. These observations are captured in Figs. 12 and 13.

6 Conclusions

The work reported in this manuscript presents a comprehensive review and taxonomy of HLL languages for high-performance reconfigurable computers. It also presents new metrics and a framework for comparative evaluation of the HLLs. The concepts and methodology are inspired from the principles of Newtonian mechanics, which are applied notionally to the movement of user and tool in an abstract specifications space as they progress towards the target solution. The performance of this user and tool combination is evaluated based on two principle criteria, (i) total time to solution, and (ii) incremental progress rate encapsulating the combined user and tool resource usage efficiency at discrete time steps along the development path. The metrics that focus on each criterion are the productivity and the work progress rate respectively. The productivity metric characterizes how fast the solution is obtained whereas the work progress rate captures how efficiently a solution is obtained. These metrics are used as attributes in the overall framework. The HLL evaluation framework presented provides a structure that enables evaluation of qualitative language features such as support for pointers and debugging capability, as well as quantitative metrics such as productivity and work progress rate. Our review and experimental results showcase the applicability of our methodology to a wide-array of languages from imperative to dataflow programming models.

ACKNOWLEDGMENT

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. IIP-0706352.

REFERENCES

[1] W. Luk, N. Shirazi, and P.Y.K. Cheung, "Compilation Tools for Run-time Reconfigurable Designs", *IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM 1997, pp. 56–65.
[2] K. Compton, and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software", *ACM Computing Surveys* 34 (2) (2002) 171-210.
[3] Cray Inc., "Cray XD1TM FPGA Development (S-6400-14)", 2006.
[4] Silicon Graphics, Inc., "Reconfigurable Application-Specific Computing User's Guide (007-4718-005)", January 2007.
[5] Impulse C – "Impulse Accelerated Technologies" web site available at http://www.impulsec.com/
[6] Celoxica, Inc., web site available at http://www.celoxica.com
[7] Mitrionics web site available at http://www.mitrion.com/index.shtml
[8] Xilinx Inc., web site available at http://www.xilinx.com/ise/optional_prod/system_generator.htm
[9] DSPLogic web site available at http://www.dsplogic.com
[10] SRC Computers, Inc., "SRC CarteTM C Programming Environment v2.2 Guide (SRC-007-18)", August 2006.
[11] B. Holland, M. Vacas, V. Aggarwal, R. DeVille, I. Troxel, and A.D. George, "Survey of C-based Application Mapping Tools for Reconfigurable Computing", *2005 MAPLD International Conference*, Washington, DC, USA, September, 2005.
[12] S.A. Edwards, "The Challenges of Synthesizing Hardware from C-Like Languages", *IEEE Design & Test of Computers*, Vol. 23, Issue 5, pp. 375 - 386, September 2006.
[13] E. El-Araby, M. Taher, M. Abouellail, T. El-Ghazawi, and G. B. Newby, "Comparative Analysis of High Level Programming for Reconfigurable Computers: Methodology and Empirical Study", *III Southern Conference on Programmable Logic (SPL2007)*, Mar del Plata, Argentina, February, 2007.
[14] E. El-Araby, P. Nosum, and T. El-Ghazawi, "Productivity of High-Level Languages on Reconfigurable Computers: An HPC Perspective", *IEEE International Conference on Field-Programmable Technology (FPT 2007)*, Japan, December, 2007.
[15] T. Sterling, "Productivity Metrics and Models for High Performance Computing", *International Journal of High Performance Computing Applications*, vol. 18, pp. 433-440, 2004.
[16] M. Snir and D. A. Bader, "A Framework for Measuring Supercomputer Productivity", *International Journal of High Performance Computing Applications*, vol. 18, pp. 417-432, 2004.
[17] K. Kennedy, C. Koelbel and R. Schreiber, "Defining and Measuring the Productivity of Programming Languages", *International Journal of High Performance Computing Applications*, vol. 18, pp. 441-448, 2004.
[18] J. Kepner, "HPC Productivity: An Overarching View", *International Journal of High Performance Computing Applications*, vol. 18, pp. 393-397, 2004.
[19] J. Kepner, "High Performance Computing Productivity Model Synthesis", *International Journal of High Performance Computing Applications*, vol. 18, pp. 505-516, 2004.
[20] R.W. Numrich, "Performance Metrics Based on Computational Action", *International Journal of High Performance Computing Applications*, Vol. 18, No. 4, pp. 449 - 458, November 2004.
[21] R.W. Numrich, "A Metric Space for Computer Programs and The Principle of Computational Least Action", *The Journal of Supercomputing*, Vol. 43, No. 3, pp. 281-298, March 2008.
[22] R.W. Numrich, L. Hochstein, V. Basili, "A Metric Space for Productivity Measurement in Software Development", *Proceedings of the Second International Workshop on Software Engineering for High Performance Computing System Applications (SE-HPCS'05)*, St. Louis, Missouri, 15 May 2005.
[23] R.W. Numrich, "Computational Force: A Unifying Concept for Scalability Analysis", *Advances in Parallel Computing*, Volume 15, ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.
[24] R.W. Numrich, "Computational Force, Mass and Energy", *International Journal of Modern Physics C*, Vol. 8, No. 3, pp. 437-457, 1997.
[25] E. El-Araby, M. Taher, T. El-Ghazawi, and J. Le Moigne, "Remote Sensing and High Performance Reconfigurable Computing Systems", in *High Performance Computing in Remote Sensing*, Editors A. J. Plaza, C.I. Chang, Volume 16, New York, Chapman & Hall/CRC Computer & Information Science Series, 2007, pps. 496. ISBN: 9781584886624, ISBN 10: 1584886625.
[26] E. El-Araby, T. El-Ghazawi, J. Le Moigne, K. Gaj, "Wavelet Spectral Dimension Reduction of Hyperspectral Imagery on a Reconfigurable Computer", *IEEE FPT 2004*, Brisbane, Australia, December, 2004.
[27] O.D. Fidanci, H. Diab, T. El-Ghazawi, K. Gaj, and N. Alexandridis, "Implementation Trade-offs of Triple DES in the SRC-6E Reconfigurable Computing Environment", *Proc. MAPLD 2002*.

Esam El-Araby received his B.Sc. degree in Electronics and Telecommunication Engineering from Assiut University, Egypt, in 1991, his Higher Diploma degree in Automatic Control and Computer Engineering from Assiut University, Egypt, in 1997. He received his M.Sc. in Computer Engineering from the George Washington University (GWU), USA, in 2004. He is currently pursuing his Ph.D. degree in Computer Engineering in the Department of Electrical and Computer Engineering at the George Washington University (GWU), USA. He is also a research assistant at the High Performance Computing Lab (HPCL) at GWU as well as the NSF Center for High-Performance Reconfigurable Computing (CHREC) at GWU. Being a member of the HPCL lab and a researcher with CHREC, he participated in several research projects involving organizations such as DoD, DARPA, NSF, and NASA. Through these projects, he published a number of research work in international gatherings organized by organizations such IEEE, ACM, and NASA. His publication record includes a book chapter on HPRCs and remote sensing, 6 journal papers, and 33 conference papers. His research interests include reconfigurable computing, hybrid architectures, evolvable hardware, performance evaluation, digital signal/image processing, and hyperspectral remote sensing.

Saumil Merchant received the B.E. degree in Electronics from Mumbai University, India in 1999, and the M.S. and PhD. degrees in Computer Engineering from University of Tennessee, Knoxville in 2003 and 2007 respectively. He is currently a research scientist in the department of Electrical and Computer Engineering at George Washington University. His research interests include reconfigurable computing, high-performance computing, embedded computing, and machine intelligence. He is a member of IEEE and ACM.

Tarek El-Ghazawi is a Professor in the Department of Electrical and Computer Engineering at The George Washington University, where he also directs the High Performance Computing Laboratory (HPCL). He is a fellow of the Arctic Region Supercomputing Center and a Visiting Scientist at NASA GSFC. He has received his Ph.D. degree in Electrical and Computer Engineering from New Mexico State University in 1988. His research interests include high-performance computing and architectures, reconfigurable computing, parallel I/O, and performance evaluations. He has published over 100 refereed research papers and book chapters in these areas and his research has been supported by DoD/DARPA, NASA, NSF, and industry including IBM and SGI. He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE), a member of the Association for Computing Machinery (ACM), Phi Kappa Phi National Honor Society.