Strategic Challenges for Application Development Productivity in Reconfigurable Computing

Saumil G. Merchant, Brian M. Holland, Casey Reardon, Alan D. George, Herman Lam, Greg Stitt
NSF Center for High-Performance Reconfigurable Computing (CHREC), University of Florida
{merchant, holland, reardon, george, lam, stitt}@chrec.org

Melissa C. Smith, Nahid Alam
Clemson University
{smithmc, nalam}@clemson.edu

Ivan Gonzalez, Esam El-Araby, Proshanta Saha, Tarek El-Ghazawi, Harald Simmler
NSF Center for High-Performance Reconfigurable Computing (CHREC), The George Washington University
{ivangm, esam, sahap, tarek, simmler}@gwu.edu

***Abstract*—Performance and versatility requirements arising from escalating fabrication costs and design complexities are making reconfigurable computing technologies increasingly advantageous on the roadmap towards many-core technologies. This reformation in device architectures is necessitating a critical reformation in application design methods to bridge the widening semantic gap between design productivity and execution efficiency. This paper explores the strategic challenges in FPGA design methodologies and evaluates potential solutions and their impact on future DoD applications and users. A new research initiative, Strategic Infrastructure for Reconfigurable Computing Applications (SIRCA), has also been proposed as a potential new DARPA program to address the FPGA productivity problem.**

I. Architecture Reformation

Unsustainable thermal and power overheads have saturated achievable clock frequencies in modern processors, ending the performance growth relying on instruction-level parallelism and higher clock frequencies. The continued quest for higher performance has started a new trend focusing on thread- and task-level parallelism, leading to the emergence of homogeneous and heterogeneous multi-/many-core computing structures. As the number of cores on a chip increases, relative performance benefits from thread-level parallelism will diminish as per Amdahl's Law, necessitating performance gains through other types of parallelism that can be achieved mainly by heterogeneous, many-core computing structures. Furthermore, increasingly high fabrication costs will encourage use of reconfigurable computing (RC) structures ranging from gate-level, reconfigurable devices such as FPGAs [1] to interconnect-level, reconfigurable devices such as Tile-64 processor [2] to increase versatility and therefore market size. Thus, diverse design paradigms and architectures from parallel and custom computing systems will eventually infuse into mainstream computing systems resulting in a technological convergence and reformation. Figure 1 shows the landscape of evolving device architectures. In this paper we do not distinguish between multi-core and many-core devices and use the notation MC to refer to them collectively. The two primary classes of MC architectures technology are *fixed MC* (FMC) devices and *reconfigurable MC* (RMC) devices. FMC devices have fixed hardware structure that cannot be changed after fabrication, whereas RMC devices can change hardware structure post-fabrication to adapt to specific problem requirements. Depending on the types of cores in FMC and RMC devices these can be further divided in homogeneous and heterogeneous devices. Early indicators of this convergence trend can be seen in vendor roadmaps and upcoming device architectures such as the Intel tera-scale research chip (homogeneous FMC) [3], Cell processor (heterogeneous FMC) [4], Tile-64 processor (homogeneous RMC) [2], Monarch (heterogeneous RMC) [5], and FPGA (heterogeneous RMC) [1].

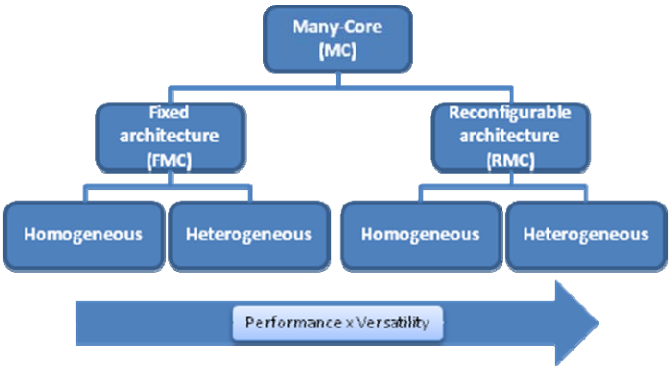


Figure 1. Landscape of future computing devices

This material is based on research sponsored by DARPA under agreement number FA8650-07-1-7742. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

II. APPLICATION REFORMATION

This reformation in computing architecture necessitates rethinking of application software layers to fully exploit multiple types and layers of parallelism. The widening semantic gap between application design productivity and execution efficiency poses significant design challenges to fully exploit the potential of heterogeneous, reconfigurable, and many-core architectures. Design challenges previously faced by a small community of designers for parallel and custom computing systems are now in the forefront affecting a large user base. As these systems enter the mainstream, there is an urgent need for new concepts, methods, and tools to bridge the widening semantic gap and boost design productivity.

This paper explores limitations and challenges of the current FPGA-based design methodologies and proposes potential research innovations to address them. The proposed innovations are qualitatively and quantitatively analyzed for their projected impacts on overall design productivity via a representative case study. Due to widespread interest in tapping performance, power, and versatility advantages of FPGA-based reconfigurable computing technology for DoD systems, proposed innovations can play a critical role in reforming application design methodologies. Concepts and methodologies developed based on the proposed innovations may also significantly impact performance and productivity of many forms of future many-core computing systems and applications. The identified research innovations are proposed for a potential new DARPA research initiative, **S**trategic **I**nfrastructure for **R**econfigurable **C**omputing **A**pplications (SIRCA), to address the FPGA productivity problem and invigorate widespread use of FPGA-based RC technology among DoD community.

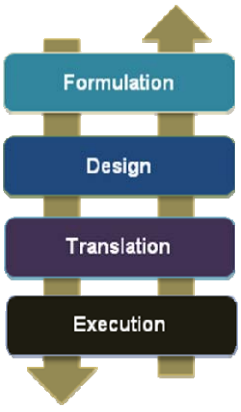


Figure 2. FPGA application development phases

III. FPGA TOOL LIMITATIONS & CHALLENGES

Despite significant performance and flexibility advantages, the widespread success of reconfigurable computing has been limited by inadequate design tools and methodologies that require hardware design expertise – a skill not common among the large potential user base of software programmers and domain scientists. This problem is worsened by the widening semantic gap due to increasing device and design complexities and poses significant challenges to the widespread acceptance of these technologies. To address this problem, new methodologies and concepts based on four application development phases, *Formulation (F)*, *Design (D)*, *Translation (T)*, and *Execution (E)*, are critical and can potentially revolutionize design tool productivity. These phases encompass the overall FPGA-design process from problem formulation to execution and are illustrated in Figure 2. During Formulation, requirements are expressed using abstract modeling, enabling strategic exploration of the algorithm and architecture of the application. During the Design phase, details regarding the behavior and structure of the algorithm(s) and circuit architecture are specified. High-level synthesis and mapping from the behavioral and structural representation provided during the Design phase to actual resources on the target hardware producing a hardware/software solution for the platform occurs in Translation. Finally, during Execution, run-time services are integrated to support executing, monitoring, and debugging of the full application. These phases apply at the device level (i.e. design issues within the FPGA) as well as at the system level (i.e. design issues outside the FPGA for single- and multi-device scenarios). Although these phases may apply to all existing and emerging classes of FPGA use cases in DoD (from real-time SoC systems to high-end HPC systems), the objectives, priorities, and details vary by case.

Current FPGA application development tools are fragmented across the FDTE application development spectrum and at best partially address requirements of a few FDTE phases. Significant limitations exist in current tools such as a lack of integrated, end-to-end toolsets and interoperability among tools from different vendors. These limitations and challenges are discussed next, grouped by the four application development phases.

A. Formulation: Limitations and Challenges

Formulation has been largely overlooked by existing tools and methodologies and is either skipped completely or conducted in an ad-hoc manner in the Design Phase. Critical strategic design decisions are not made through algorithm exploration, architecture exploration and mapping, and numerical analysis (e.g. dynamic range vs. precision) but rather ad-hoc in the Design, Translation, and Execution phases [6-7]. As a result, this method often leads to wasteful and costly design iterations caused by poor initial design decisions. The few tools that exist for Formulation are largely inadequate and often specific to an application domain. Furthermore, they are not integrated with other key elements of the tool flow [8]. Without a bridge to the Design phase, any analysis and algorithm models created using such Formulation tools would have to be discarded, necessitating a fresh start in the Design phase.

B. Design: Limitations and Challenges

While there are many Design tools available, ranging from hardware description languages (HDLs) to high-level languages (HLLs), the tools still require a significant amount of hardware knowledge to use effectively [9]. HDLs require extensive hardware expertise, thus excluding a potential broader user base of domain scientists and programmers. Conversely, HLLs lack the ability to adequately specify the

concurrency required to fully exploit the features of FPGAs systems and sub-systems. The lack of integrated, system-level, hardware/software co-design tools and languages necessitates manual design partitioning and in some cases requires separate software and hardware tool flows [10]. Support for interoperability and portability among tools and designs is either non-existent or limited to a small group of supported tools and platform configurations [11]. Hence, even though new HLL tools offer limited productivity improvements, more often the designs must be completely re-written to execute on different platforms, severely limiting design/core reusability and portability. Further, the scalability of designs to or across multiple FPGAs (or nodes) is largely left to the user, as there is no formal system-level support.

C. *Translation: Limitations and Challenges*

Translation time, in particular place-and-route (PAR) time, is generally considered a major impediment to FPGA development productivity. Translation algorithms are hampered by the unpredictable and often excessively long place-and-route (PAR) times that can further vary depending on the tool chain [12]. As FPGAs become denser and more complex, these issues will continue to worsen unless new approaches are taken in the Translation phase. However, the proprietary and closed-source nature of these tools makes academic and research innovations difficult [13].

D. *Execution: Limitations and Challenges*

In the Execution phase, tools and methodologies for run-time support are lagging, especially for verification, analysis, debugging, and optimization at the system-level [14]. Run-time services to support mission-specific requirements such as job scheduling and completion, checkpoint and heartbeat services for fault tolerance (FT), and configuration management for run-time full/partial reconfigurations (RTR/PR) are also largely lacking. The few available run-time services are limited in functionality and are platform-specific. These problems are worsened by lack of standards for FPGA-systems and subsystem architectures within which to integrate run-time tools [11].

IV. Proposed Research Innovations

With FPGAs gaining prominence as computing platforms of prime interest in DoD community due to advantages of performance, power, and versatility, it is critical to revolutionize design productivity by addressing the current limitations in application development tools. We have identified twelve potential research thrusts to address the challenges and close the semantic gap between productivity and execution efficiency. These are listed in Table 1 and described below, organized across the four application development phases. Although some research thrusts span across multiple phases as indicated by × marks in Table 1, they have been grouped under their primary application development phases in our discussion below. The proposed innovations are research directions for development of high-productivity, end-to-end, integrated design environment for FPGA application development as part of the proposed new SIRCA program for DARPA. The program goals are:

1. Develop innovative end-to-end concepts, methods, and integrated toolsets to dramatically improve the development productivity of FPGA-based RC applications.
2. Revolutionize implementation of critical DoD applications by broadening the use of FPGA-based RC among domain scientists and system designers for both high-performance computing (HPC) and embedded (HPEC) scenarios.

Table 1. Proposed research thrusts

Research Thrusts	F	D	T	E
1. Strategic exploration	×			
2. High-level prediction	×			
3. Numerical analysis	×			
4. Bridging design automation	×	×		
5. System-level parallel languages		×	×	
6. HW/SW co-design methods		×	×	
7. Reusable & portable design	×	×	×	
8. Translation algorithms			×	
9. Translation target architectures			×	
10. Run-time debug & verification		×		×
11. Performance analysis		×		×
12. Run-time services				×

It should be noted that SIRCA is distinct from but complementary and synergistic with two existing DARPA programs, HPCS (High-Productivity Computing Systems and AACE (Architecture-Aware Compiler Environment). HPCS mainly focuses on development of high-performance computing systems and methodologies built using FMC technologies, whereas AACE primarily focuses on compiler optimizations using feedback from run-time analysis tools.

A. *Research Thrusts: Formulation*

The Formulation phase is the strategic design "playground" for managing increasing complexity of an FPGA-based RC application. It is the phase for critical strategic design decisions and tradeoffs and involves abstract application and architecture modeling for design space exploration. The Following research thrusts were identified for the Formulation phase.

1. Strategic Exploration

This consists of graphical, textual, or hybrid modeling techniques for parallel algorithm and architecture exploration with constructs for expressing deep and wide parallelism. Focus on pattern-based models for algorithm exploration and abstract target platform models for architecture mappings can facilitate reuse increasing productivity [15]. Constructs for iterative algorithm and architectural exploration and incremental refinements can assist with strategic design decisions and tradeoffs to explore viable algorithm and architecture combinations.

2. High-level Prediction

High-level performance and resource prediction techniques and methods can facilitate rapid algorithm and architecture exploration. These methods can be analytical [16-17], simulative [18], or a combination thereof.

3. Numerical Analysis

Research effort for concepts, methods, and tools enabling numerical analysis are essential. As arithmetic precision and FPGA resource utilization are inextricably linked, performance, resources utilization, and functionality may depend on choice of data precision.

4. Bridging Design Automation

A bridge from the Formulation to the Design phase is critical for productivity and widespread adoption of Formulation methods among users. It forms the basis for design automation by auto-generating code templates and cores from core and pattern libraries, significantly innovating overall design productivity.

Qualitative Impact. Innovations in Formulation, with methods and tools to bridge to the Design phase, will result in a major reduction in DTE (Design, Translation, Execution) development costs, where DTE cost is a product of time spent per DTE iteration and DTE frequency (i.e. number of DTE iterations). Coding would be minimized and Design phase would be semi-automated through patterns and code templates reducing Design time. Also, DTE frequency would be reduced since better strategic choices in Formulation mean less frequent design and re-design. Abstract algorithm modeling and automation would result in significant utility gain (and thus productivity improvement) for non-experts. Intuitive methods and tools amenable to domain scientists and system designers will result in shorter learning curves, eliminating the need for user hardware expertise and invigorating widespread use of FPGA-based RC technology.

B. Research Thrusts: Design

While strategic design is performed in the Formulation phase, tactical design and coding are performed in the Design phase. System-level design languages, hardware/software co-design, and other design tools are necessary for design exploration, refinement and verification. Issues to be addressed include design reusability, interoperability, and portability. Ideally, coding is minimized through the use of design patterns and code templates resulting from the Formulation phase. The research thrusts identified for the Design phase are as follows.

5. System-level Parallel Languages

System-level parallel design languages for implementation coding will significantly impact user productivity. Constructs for system-level coding, deep and wide parallelism, and hierarchical design abstractions will significantly reduce Design time facilitating incremental and iterative design refinements for crucial debug and optimizations [19].

6. HW/SW Co-design Methods

Hardware/software co-design methodologies for partitioning and mapping on target architecture will increase user productivity providing integrated, system-level design environment. Automated constructs for design partitioning can provide efficient system-level mechanisms targeting both fixed and reconfigurable computing structures facilitating abstraction of underlying hardware details from the user [20].

7. Reusable and Portable Design

Built-in constructs facilitating reuse and design portability have potential to significantly innovate user productivity. Use of pattern-based design methodologies [15] and core libraries [21] not only facilitate automation but maximize reusability and portability. Standards for FPGA system and sub-system interfaces are essential for achieving design portability across platforms from different vendors [21-24].

Qualitative Impact. Innovations in Design phase result in reduction in Design time due to intuitive and high-level development methods and tools. Mechanisms for reuse and portability significantly improve design productivity. Better Design tools mean less frequent design and re-design, reducing Design time and DTE frequency.

C. Research Thrusts: Translation

Translation cost is a product of Translation time and Translation frequency (i.e. how often one must translate). Translation time, in particular place-and-route (PAR) time, is considered one of the major impediments to FPGA development productivity. Translation time can be improved through innovations in Translation algorithms or innovations in architecture of the target devices. Translation frequency can be reduced through innovations in other phases. The research thrusts identified for the Translation phase are as follows.

8. Translation Algorithms

Innovations in Translation algorithms may significantly reduce Translation times increasing overall productivity. Potential research directions for Translation algorithms include region-based PAR methods to reduce routing complexity and improved PAR techniques to tradeoff routing effort and times [25-26]. Parallelizing PAR tools to exploit newer multi-core microprocessor technologies can also significantly expedite Translation [27].

9. Translation Target Architectures

Innovations in device architectures to be better targets for PAR tools may significantly impact Translation times [28]. Hierarchical routing techniques with tool emphasis on locality and pipelining, and coarser device architectures to reduce PAR complexity, may significantly reduce Translation time but may tradeoff circuit performance.

Qualitative Impact. Although Translation time is generally considered a major bottleneck in development productivity, reductions in iterations through Translation phase could potentially also have a large impact on overall development productivity. Innovation in Translation algorithms and device architectures will result in reduction in Translation time, but not Translation frequency. Reductions in Translation frequency can be achieved via innovations in Formulation, Design, and Execution phases. Therefore, overall development productivity impact of innovations in Translation phase may be modest as compared to impact of Formulation and Design innovations.

D. Research Thrusts: Execution

Methods and tools are necessary to provide run-time services in the Execution phase to support debug, verification, analysis, and optimization of FPGA-based RC applications. The research thrusts identified for the Execution phase are as follows.

10. Run-time Debug and Verification

Run-time, in-circuit debug methods and tools can significantly reduce debug and verification times. System-level debug concepts and methods are vitally important for verifying complete systems as current tools are device-level and platform-specific. Interface standards for FPGA systems and sub-systems with in-built debug support are essential for reductions in verification times and increasing overall productivity.

11. Performance Analysis

Run-time, post-mortem analysis methods and tools can help quickly identify and locate bottlenecks for critical performance and resource optimizations [29]. Analysis data feedback from Execution phase to Formulation, Design, and Translation phases is critical to locate bottlenecks and provide run-time information for algorithm optimizations. Major productivity improvements can be realized due to rapid optimizations reducing required DTE iterations and potential increase in utility.

12. Run-time Services

Run-time services to support varied mission scenarios such as fault-tolerant (FT), real-time (RT), partial-reconfiguration (PR), and system-on-a-chip (SoC) applications are essential for wide applicability of FPGA-based computing systems. Example services include checkpoint and heartbeat services [30], PR configuration management [31], load balancing and job scheduling for multi-node systems [32], and secure configuration mechanisms for sensitive missions.

Qualitative Impacts. Superior Execution tools can offer significant development cost savings, and reduction in debug time and DTE frequency. Critical optimizations and bottleneck elimination can also result in critical utility gains, thus increasing productivity.

V. Quantitative Impact Analysis

Proposed innovations based-on FDTE-based design methodologies can significantly reduce development costs and improve performance of RC applications resulting in significant impact on the overall design productivity. Not only can it improve productivity of RC experts, but it can enable the use of this technology for a large community of domain scientists in high-performance computing (HPC) and system designers in high-performance embedded computing (HPEC) who can significantly benefit from the advantages of performance, power, cost, size, and versatility of RC systems. The previous section presented the qualitative reasoning of impact on development productivity. This section presents a quantitative impact analysis of proposed FDTE via a case study.

A. RC Productivity Model

The analysis is based on the following *RC Productivity Model*. Productivity, as defined by DARPA HPCS program [33], is the ratio of utility over development cost:

$$\psi = \frac{U}{C} \tag{1}$$

Utility U is a function of set of weighted attributes a_i used to measure the usefulness of a design. Examples of utility attributes include speed, throughput, power savings, and area savings. The magnitude of utility can be measured as the length of the utility vector in a multidimensional utility space:

$$U = length\left(\overline{U}\right) \tag{2}$$

Figure 3 shows an example of a 2-dimensional utility space, using utility attributes *weighted speedup S* and *power savings P*. For example, the utility for Design 1 is:

$$U_1 = length\left(\overline{U_1}\right) = \left|\overline{U_1}\right| = \sqrt{S_1^2 + P_1^2} \tag{3}$$

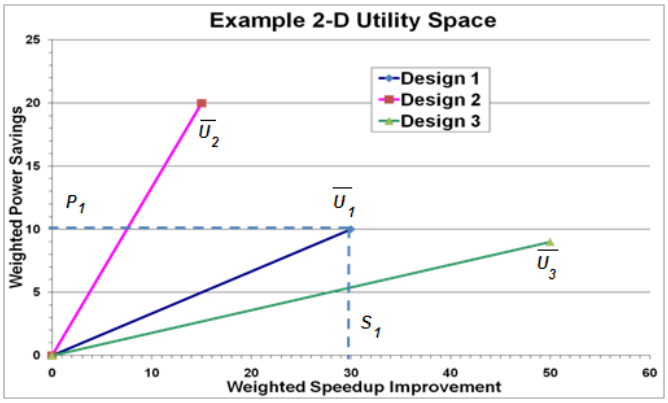


Figure 3. Example 2-dimensional utility space

The development cost C is the work required to achieve that utility and is typically measured as development time (e.g. man-hours) or dollars. The cost for an RC application prototype is modeled as the summation of the costs for the four phases of the FDTE model: Formulation (C_F), Design (C_D), Translation (C_T), and Execution (C_E). Each application may require the development of multiple (k) prototypes. Each prototype is represented as a design iteration i.

$$C = \sum_{i=1}^{k} \left(C_F + C_D + C_T + C_E\right)_i \tag{4}$$

The productivity number is often more meaningful when compared with a reference. Hence, our analysis will mainly calculate relative productivity gains given by the ratio of two productivities:

$$Productivity\ Gain = \frac{\psi_1}{\psi_2} = \frac{U_1}{C_1} \div \frac{U_2}{C_2} \qquad (5)$$

B. Impact Projections of Proposed FDTE Innovations

Before presenting the details of the analysis, we first describe our general approach and define the key assumptions. This information provides the background for the projections to be estimated.

Approach. Productivity impact is determined by many factors, interrelated in a complex manner. To keep our analyses tractable and gain a clear understanding of the analyses, our approach is to break the complex problem into its component factors in order to isolate and focus on the impact of each factor. Our setup consists of five analyses as described below:

- For the first four, analyze the productivity effect with focus on improvement in cost *C,* assuming minimal change in utility *U*. These four analyses are as follows:
  1) Assume that there are innovations ***only*** in the ***Formulation*** phase, with no innovation in the other three phases (holding them constant for simplicity and clarity), and determine the end-to-end development cost and the impact on productivity.
  2) Similar to #1, but with innovations ***only*** in the ***Design*** phase, determine productivity impact.
  3) Similar to #1, but with innovations ***only*** in the ***Translation*** phase, determine productivity impact.
  4) Similar to #1, but with innovations ***only*** in the ***Execution*** phase, determine productivity impact.
- For the fifth analysis, using a different scenario, analyze the productivity effect based on the change in utility *U,* assuming minimal change in cost *C*.
- For all five analyses:
  - Cost of each phase is the product of time spent in that phase and the number of iterations through that phase (i.e. frequency). For example, the Translation cost is $C_T = f_T \times t_T$
  - Two projections were made to compare to a defined baseline: *conservative* and *optimistic*. The conservative projection represents expected improvements that should readily result from the proposed innovations. Optimistic estimates represent ambitious but attainable productivity gains.

Baseline assumptions. The five analyses assume a baseline development cost as a reference for relative productivity gains. The assumptions for the baseline costs are as follows:

- The percent breakdown for the baseline development time per phase was inferred from reported percentages of five HPC and HPEC DoD/NASA software projects; Hawk, Saturn V, SAGE, NTDS, and Gemini [34-35]. Note that the percentages were adjusted to reflect the development of an FPGA-based RC application (e.g. longer Translation time):

  Formulation = 5%, Design = 50%,
  Translation = 20%, and Execution = 25%

- Total baseline development time is assumed to be 1000 hours.
- Analyses assume the user is a domain scientist knowledgeable only in sequential programming.
- Baseline code is assumed to be developed in HDL, with a learning curve cost of 400 hours (approximate time for an intensive one-semester university course (s/c)).

Table 2 shows the resulting numbers for the baseline development costs (in units of hours and weeks).

1. Quantitative Impact of Formulation Innovations

For this analysis, the assumption is that we are considering innovations ***only*** in the ***Formulation*** phase, and there are no innovations in the other three phases. Also, we assume that there is no increase in utility *U*, but there will be a decrease in cost *C*. This approach represents the scenario of a user who is already capable of getting the most utility out of a design. Thus, new innovations in Formulation tools will not help obtain more utility (e.g., speedup). However, it can help produce a design faster (decrease in cost *C*).

Table 2. Baseline development costs

Development Phase	Development Cost (hrs)	
	Baseline	***Rationale***
Learning Curve	400 (s/c)	*HDL learning curve*
Formulation	40 (1w)	*Current Formulation methods*
Design	480 (12w)	*Estimated from literature*
Translation	200 (5w)	*Estimated from experience*
Execution	280 (7w)	*Estimated from literature*
Total	400 (10w) + 1000 (25w) = 1400 (35w)	

Detailed results are shown in Table 4. Cost improvements due to innovations in each FDTE phase are estimated based on empirical knowledge. Note that the rationale for cost reduction for each row of the table is also given. As is evident, if we spend more time and effort in the Formulation phase (80 hours or 2 weeks for both the conservative and optimistic cases), it results in significant reduction of cost in the other three phases. Innovations in Formulation, with methods and tools to bridge to the Design phase, will result in a reduction in the learning curve and Design time. Coding would be minimized and Design phase would be semi-automated through design patterns and code templates from Formulation. Also, the number of prototype designs required and DTE frequency would be reduced, since better strategic choices in Formulation means less frequent design and re-design.

2. Quantitative Impact of Design Innovations

Similar to the previous subsection, the assumption here is that we are considering innovations ***only*** in the ***Design*** phase, and there are no innovations in the other three phases. Also, we assume that there is no increase in utility, but there will be a decrease in cost. The detailed results are shown in Table 5. The rationale for the cost reduction for each row of the data is also given.

In general, innovations in the Design phase result in many of the same benefits as Formulation. Improved Design

methodologies lead to considerable DTE cost reduction. Intuitive, high-level design tools help reduce Design time. DTE frequency is reduced (less debugging and optimization iterations) as a result of better concepts and tools. However, poor initial design decisions made ad-hoc in Design due to the lack of adequate Formulation methods may not reduce wasteful prototype iterations.

3. Quantitative Impact of Translation Innovations

As before, we are considering innovations ***only*** in the ***Translation*** phase, thus we assume that there are no innovations in the other three phases. We also assume that there will be a modest increase in utility and a decrease in cost. The detailed results are shown in Table 6.

Although Translation time (T time) is often considered one of the major impediments to FPGA-based development productivity, it is shown here that the impact of innovations in reducing T time may not affect overall productivity significantly. The reason is that innovations in Translation do not affect F, D, or E costs, nor the T frequency, which is more critical than just reducing T time. As shown in the earlier analyses, T frequency reductions can be achieved via innovations in the F, D, and E phases. Note that unlike the previous analyses, we also assume a 10% (conservative) and a 30% (optimistic) gain in utility, assuming better Translation algorithms and devices will improve utility. Even with this assumption, the productivity gain is modest.

It should be noted that our analyses does not account for any intangible benefits of reductions in Translation times. Benefits of faster Translation may impact debugging iterations and user interactions with the design and could result in higher productivity gains. Although reductions in Translation frequency may have higher impact on overall productivity, Translation time improvements are vitally important and innovations here must not be ignored.

4. Quantitative Impact of Execution Innovations

As before, we are considering innovations ***only*** in the ***Execution*** phase, thus we assume that there are no innovations in the other three phases. We also assume that there will be a slight increase in utility and a decrease in cost. The detailed results are shown in Table 7. The rationale for the cost reduction for each row of the data is also given.

Innovations in Execution methods and tools will reduce debug, verification, and optimization costs in the DTE phases. They will also reduce the DTE frequency to minimize the number of debugging and optimization cycles. The learning curve is reduced since run-time analysis tools will become more intuitive to the user. As in the case of Translation, this analysis assumes a 10% (conservative) and a 30% (optimistic) gain in utility because innovations in Execution methods and tools will help identify bottlenecks and critical design optimizations that may increase utility (e.g. speedup).

5. Summary of Cost Analyses

Table 3 shows the projected productivity impact numbers for the four cost analyses. Figure 4 plots the projected impact for each phase, using the baseline development cost as a reference. Based on the projected impact numbers following observations can be made for impact of innovations in each of the four phases.

Table 3. Productivity impacts for analyses 1 through 4

	Formulation	Design	Translation	Execution
Conserv.	2.7	2.1	1.2	1.5
Optim.	9.5	6.5	1.5	2.4

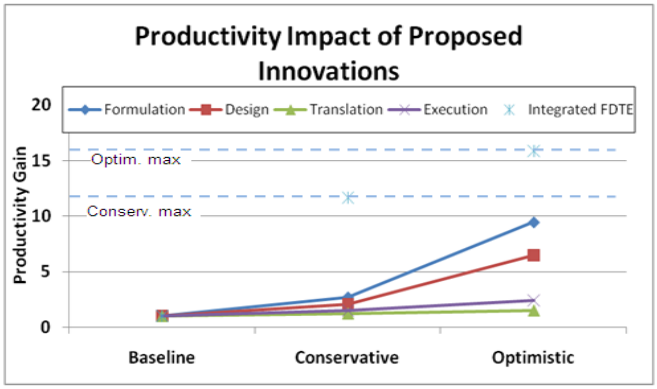


Figure 4. Productivity impact of proposed innovations

a) Formulation innovations have the highest potential for impact on productivity. Also, Formulation innovations have the potential to dramatically increase the user base for FPGAs, cascading a multiplicative effect for productivity. Finally, although strategic exploration concepts are central to RC, they are also potentially applicable to the FMC world.

b) In general, innovations in the Design phase result in many of the same benefits as Formulation. Design innovations by themselves increase overall productivity. However, Formulation and Design innovations together will likely generate maximum impact.

c) Execution innovations are important for providing critical run-time information. They enable the reduction in optimization, debug, and verification costs.

d) Improvement in Translation time is beneficial and important, but less impactful on overall productivity.

Our analyses have focused on the isolation of the impact for innovations in each development phase. It should be clear that the cumulative impact for all four phases will yield maximum benefit. Unfortunately, since the cumulative productivity impact is a complex function of the individual impacts, we cannot simply add them together. However, we can make some meaningful observations. Recall that innovations in all four phases can combine to drive DTE down, whereas none of the other innovations will drive F lower. Thus, theoretically the maximum achievable productivity, considering development time alone, will be limited by the Formulation cost and learning curve. Thus, as DTE costs approach zero, the development cost approaches the learning curve cost plus the Formulation cost. As shown by the dashed horizontal lines in Figure 4, for the

conservative case, the gain approaches 1400/120 = 12, and for the optimistic case 1400/88 = 16. Note that these theoretical maximums are based on the numbers in Table 4 and will vary depending on the particular case study. Productivity is unlike speedup; a 12× reduction in cost (*C*) would be truly outstanding (e.g. man-year effort reduced to man-month).

Finally, all the analyses thus far considered only the improvement in the cost part of the productivity equation. We have held the utility part of the equation constant. It should be clear that the productivity benefits would be even more pronounced when utility improvement from using FPGA-based RC is included, as will be shown in the next analysis.

6. Projected Impact Based on Utility

This analysis evaluates the productivity impact resulting from the utility of FPGA-based reconfigurable computing as compared to fixed computing. With systems featuring FMC devices, many of the development problems faced by RC are now also faced for FMC, although more severe in RC due to benefits of variable architecture as opposed to fixed with FMC devices. Thus, this analysis assumes comparable development costs, and evaluates the impact based on the utility of RC. Although we do not expect RC development cost to be equal to fixed-computing costs, we believe that with proposed innovations in design methodologies we can get close in the future, especially as development in the FMC world gets harder as parallelism expands. Additional assumptions for this analysis are follows:

- FMC device baseline is modern microprocessors
- Conservative: 10× speedup, 10× power savings, 2× development cost with RC vs. fixed-computing
- Optimistic: 50× speedup, 20× power savings, 1.2× development cost with RC vs. fixed-computing.

(Note: this range of utility gain is common in today's literature [36-39]).

Based on these assumptions, a two-dimensional utility space (in terms of power savings P and speedup S) is shown in Figure 5a, comparing the utility of the three cases. Recall from Section IV-A that the utility for a Design 1 is:

$$U_1 = length\left(\overline{U_1}\right) = \left|\overline{U_1}\right| = \sqrt{S_1^2 + P_1^2} \qquad (6)$$

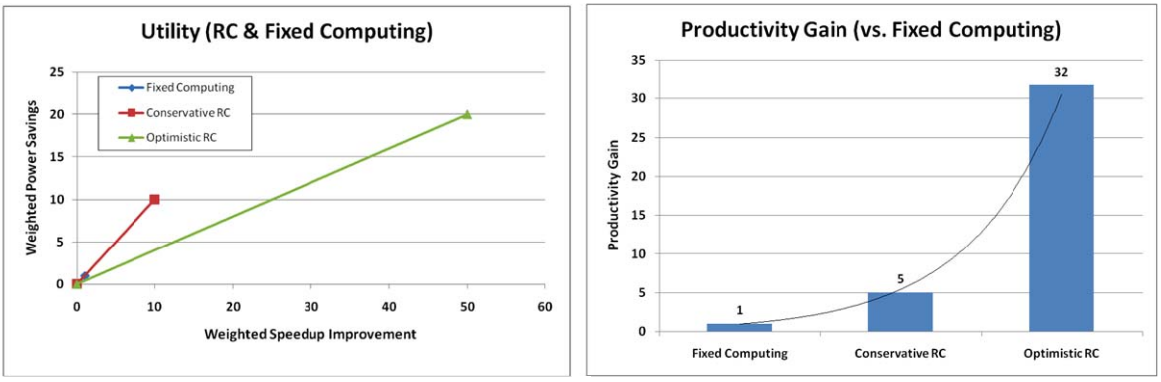
Shown in Figure 5b is the productivity gain as compared to the baseline fixed-computing system with FMC microprocessors.

In summary, it has been shown in this analysis that the productivity benefit is even more pronounced when utility improvement from using FPGA-based RC is included. Furthermore, FDTE innovations will make RC tools amenable to domain scientists. System-level, hierarchical tools will abstract the underlying hardware details. Automation will facilitate targeting FMC and RMC devices with significantly less effort, making performance, power, versatility, and cost advantages of FPGAs available to a far larger DoD user base.

VI. Conclusions

It is clear that FPGA-based reconfigurable computing is rapidly becoming a critical enabling technology for DoD, supporting future missions from satellites to supercomputers, and featuring advantages in speed, low-power, versatility, size, etc. However, the chief roadblock with these technologies is productivity in application development. To address these challenges, 12 research thrusts were identified that can revolutionize design tool productivity. Impact projections for proposed research directions have shown the potential for a dramatic decrease in application development cost, up to 12× by conservative estimates and 16× by optimistic ones. This achievement would imply a reduction in development effort from one man-year to one man-month or less, which is a dramatic increase in productivity for DoD. When factoring in utility as well, where productivity is the ratio of utility versus development cost, the results are even more dramatic. Estimates suggest a gain in overall productivity of 30× and more can be achieved versus conventional HPC and HPEC systems that are based upon fixed-architecture, multi-core technologies.

Based on these proposed innovations, a new research initiative, SIRCA, has been proposed for a high-productivity



(a) (b)

Figure 5. (a) Utility comparison, (b) Productivity gain comparison

integrated development environment (IDE) for FPGA-based applications. The goal of the program is to develop innovative, end-to-end concepts, methods, and integrated toolsets to dramatically improve the development productivity. The proposed SIRCA program will set the foundation to meet these critical needs for DoD and more. Successful outcomes from this program could invigorate widespread exploitation of RC acceleration throughout the DoD community.

REFERENCES

[1] K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys (CSUR)*, Vol. 34, No. 2, pp. 171-210, Jun. 2002.

[2] A. Agarwal, "The Tile processor: A 64-core multicore for embedded processing," *Proc. High-Performance Embedded Computing Workshop (HPEC)*, MIT Lincoln Lab, Lexington, MA, Sep. 18-20, 2007.

[3] J. Held, et al., "From a few cores to many: A tera-scale computing research overview," *White paper*, Intel Corporation, 2006.

[4] D. Pham et al., "Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor," *IEEE Journal of Solid-state Circuits,* Vol. 41, No. 1, pp. 179-196, Jan. 2006

[5] L. Lewins et. al, "World's first polymorphic computer-MONARCH," *Proc. High-Performance Embedded Computing Workshop (HPEC)*, MIT Lincoln Lab, Lexington, MA, Sep. 18-20, 2007.

[6] W. Fornaciari, D. Sciuto, C. Silvano, V. Zaccaria, A sensitivity-based design space exploration methodology for embedded systems, *Design Automation for Embedded Systems*, Kluwer Academic Publishers 7 (1-2), pp. 7-33, 2002.

[7] M. Gries, "Methods for Evaluating and Covering the Design Space during Early Design Development", *Technical report, Electronics Research Lab, University of California at Berkeley*, UCB/ERL M03/32pp. 189-194, Aug. 2003.

[8] D. Densmore, A. Sangiovanni-Vincentelli, and R. Passerone, "A Platform-Based Taxonomy for ESL Design", *IEEE Design & Test of Computers*, Volume 23, Issue 5, pp. 359-374, May 2006.

[9] 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.

[10] A. Antola, "A Novel Hardware/Software Codesign Methodology Based on Dynamic Reconfiguration with Impulse C and Codeveloper," *Programmable Logic, SPL '07, 3rd Southern Conference on,* pp. 221-224, 2007.

[11] Miaoqing Huang, Ivan Gonzalez, and Tarek El-Ghazawi, "A Portable Memory Access Framework for High-Performance Reconfigurable Computers", *Proc. IEEE International Conference on Field-Programmable Technology (ICFPT'07)*, Kokurakita, Kitakyushu, Japan, Dec. 12-14, 2007.

[12] Kannan, P.; Bhatia, D., "Interconnect estimation for FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* , vol.25, no.8, pp. 1523-1534, Aug. 2006.

[13] Xingzheng Li, Haigang Yang, Hua Zhong; "Use of VPR in Design of FPGA Architecture", *8th International Conference on Solid-State and Integrated Circuit Technology, ICSICT '06)*, pp.1880 – 1882, 2006.

[14] J. G. Tong, "A Comparison of Profiling Tools for FPGA-Based Embedded Systems", *Electrical and Computer Engineering, CCECE 2007, Canadian Conference on*, pp. 1687-1690, 2007.

[15] A. DeHon, et al., "Design patterns for reconfigurable computing," *Proc. Twelfth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 13-23, Napa Valley, CA, Apr. 20-23, 2004.

[16] M.C. Smith, "Analytical modeling of high performance reconfigurable computers: prediction and analysis of system performance," *Ph.D. Dissertation, ECE Dept. University of Tennessee*, pp. 208, Dec. 2003.

[17] B. Holland, K. Nagarajan, C. Conger, A. Jacobs, and A. George, "RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs," *Proc. High-Performance Reconfigurable Computing Technologies and Applications Workshop (HPRCTA)* at SC'07, Reno, NV, Nov. 11, 2007.

[18] E. Grobelny et al., "Simulation Framework for Performance Prediction in the Engineering of RC Systems and Applications," *Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, Jun. 25-28, 2007

[19] A. Sangiovanni-Vincentelli, "Quo Vadis, SLD? Reasoning About the Trends and Challenges of System-Level Design," *Proc. IEEE,* Vol. 95, pp. 467-506, Mar. 2007.

[20] P. Saha, "Automatic Software Hardware Co-Design for Reconfigurable Computing Systems," *Proc. Seventeenth International Conference on Field-Programmable Logic and Applications (FPL)*, Amsterdam, Netherlands, Aug. 27-29, 2007.

[21] M. Wirthlin et al., "OpenFPGA CoreLib Core Library Interoperability Effort," *Proc. Reconfigurable Systems Summer Institute (RSSI)*, Urbana, IL, Jul. 17-20, 2007.

[22] R. Merritt, "IBM calls for modeling standard," *EETimes*, Jan. 31, 2008.

[23] T-GENAPI: Technical working group on general FPGA functionality APIs, *OpenFPGA*, http://openfpga.org/.

[24] T-APPLIB: Technical working group on application specific FPGA libraries, *OpenFPGA*, http://openfpga.org/.

[25] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," *Proc. ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, pp. 29-36, Monterey, CA, 2001.

[26] R. Lysecky et al., "Warp Processors," *ACM Trans. Design Automation Electronic Systems (TODAES)*, Vol. 11, pp. 659-681, Jul. 2006.

[27] C. Fobel et al., "Using Hardware Acceleration to Reduce FPGA Placement Times," *Proc. Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 647-650, Vancouver, Canada, Apr. 22-26, 2007.

[28] G. Stitt and F. Vahid, "Thread warping: a framework for dynamic synthesis of thread accelerators," *Proc. International Conf. on Hardware/Software Co-design and System Synthesis (CODES/ISSS)*, pp. 93-98, Salzburg, Austria, Sept. 30-Oct. 5, 2007.

[29] S. Koehler, J. Curreri, and A. George, " Performance Analysis Challenges and Framework for High-Performance Reconfigurable Computing," *Parallel computing journal (special issue on HPRC)*, Vol. 34, No. 4, pp. 217-230, May 2008.

[30] I. Troxel et al., "Reliable Management Services for COTS-based Space Systems and Applications," *Proc. International Conference on Embedded Systems and Applications (ESA)*, Las Vegas, NV, Jun. 26-29, 2006.

[31] B. Holland, J. Greco, I. Troxel, G. Barfield, V. Aggarwal, and A. George, "Compile- and Run-time Services for Distributed Heterogeneous Reconfigurable Computing," *Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, Jun. 26-29, 2006.

[32] V. Kindratenko, R. Brunner, A. Myers, "Dynamic load-balancing on multi-FPGA systems: a case study," *Proc. Reconfigurable Systems Summer Institute (RSSI)*, Urbana, IL, Jul. 17-20, 2007.

[33] J. Kepner, "HPC Productivity: An Overarching View," *Int. Journal of High Performance Computing Applications*, Vol 18, No. 4, pp. 393-397, 2004.

[34] J. Kepner, "High Performance Computing Productivity Model Synthesis," *Int. Journal of High Performance Computing Applications,* Vol 18, No. 4, pp. 505-516, 2004.

[35] R. Kendall et al., "Case Study of the Hawk Code Project," *LLANL technical report* # LA-UR-05-9011, 2005.

[36] B. Shackleford et al., "A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine," *Genetic Programming and Evolvable Machines,* Vol. 2, pp. 33-60, 2001.

[37] S. Masuno et al., "Multiple Sequence Alignment Based on Dynamic Programming Using FPGA," *IEICE Trans. Information and Systems*, Vol. E90-D, No. 12, Dec 2007.

[38] S. Merchant, "Intrinsically Evolvable Artificial Neural Networks," *Ph.D. dissertation, ECE Dept. University of Tennessee*, Aug 2007.

[39] J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh, "Fixed and reconfigurable multi-core device characterization for HPEC," to appear *Proc. High-Performance Embedded Computing Workshop (HPEC),* MIT Lincoln Lab, Lexington, MA, Sep. 23-25, 2008.

Table 4. Impact of Formulation innovations

Development Phase	Development Cost (hrs)			
	Baseline	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	40 (1w)	8 (1d)	*Reduction in time reqd to learn high-level tools & methodologies*
Formulation	40 (1w)	80 (2w)	80 (2w)	*New FDTE will spend more time in F to save time in DTE*
Design	480 (12w)	160 (4w)	20 (½ w)	*Reduction in D time (templates) & freq (strategic exploration)*
Translation	200 (5w)	80 (2w)	20 (½ w)	*Reduction in T frequency, optimistically approaching one step*
Execution	280 (7w)	160 (4w)	20 (½ w)	*Lower freq (strategic exploration) & less optimization & bugs*
Total Dev Cost	1400	520	148	
Productivity Gain	Conservative = 1400/520 = **2.7**		Optimal = 1400/148 = **9.5**	*(from 35 weeks to only 3.7 weeks)*

Table 5. Impact of Design innovations

Development Phase	Development Cost (hrs)			
	Baseline	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	80 (2w)	16 (2d)	*Reduction in time reqd to learn high-level tools & methodologies*
Formulation	40 (1w)	40 (1w)	40 (1w)	*No impact*
Design	480 (12w)	240 (6w)	80 (2w)	*Reduction in D time due to intuitive languages & methodologies*
Translation	200 (5w)	120 (3w)	40 (1w)	*Fewer design flaws lead to reduction in Translation frequency*
Execution	280 (7w)	200 (5w)	40 (1w)	*Fewer bugs lead to reduction in debug*
Total Dev Cost	1400	680	216	
Productivity Gain	Conservative = 1400/680 = **2.1**		Optimal = 1400/216 = **6.5**	*(from 35 weeks to only 5.4 weeks)*

Table 6. Impact of Translation innovations

Development Phase	Development Cost (hrs)			
	Baseline	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	400 (s/c)	400 (s/c)	*No impact*
Formulation	40 (1w)	40 (1w)	40 (1w)	*No impact*
Design	480 (12w)	480 (12w)	480 (12w)	*No impact*
Translation	200 (5w)	40 (1w)	8 (1d)	*Significant reductions in Translation time, not frequency*
Execution	280 (7w)	280 (7w)	280 (7w)	*No impact*
Total Dev Cost	1400	1240	1208	
Utility Gain		10%	30%	*Better resource utilization and clk freq due to innovated tools*
Productivity Gain	Conservative = 1.1× (1400/1240) = **1.2**		Optimal 1.3 × (1400/1208) = **1.5**	

Table 7. Impact of Execution innovations

Development Phase	Development Cost (hrs)			
	Baseline	*Conserv.*	*Optim.*	*Rationale*
Learning Curve	400 (s/c)	360 (9w)	320 (8w)	*Easier learning curve for run-time analysis & optimization tools*
Formulation	40 (1w)	40 (1w)	40 (1w)	*No impact*
Design	480 (12w)	360 (9w)	280 (7w)	*Reduction in Design frequency and debugging time*
Translation	200 (5w)	120 (3w)	80 (2w)	*Fewer debugging cycles leads to reduction in T frequency*
Execution	280 (7w)	160 (4w)	40 (1w)	*Better tools quickly identify bottlenecks and bugs*
Total Dev Cost	1400	1040	760	
Utility Gain		10%	30%	*Critical design optimization and bottleneck identification*
Productivity Gain	Conservative = 1.1× (1400/1040) = **1.5**		Optimal 1.3 × (1400/760) = **2.4**	