Optimizing FPGA Performance, Power, and Dependability with Linear Programming

NICHOLAS WULF, ALAN D. GEORGE, and ANN GORDON-ROSS, NSF Center for High-Performance Reconfigurable Computing (CHREC), University of Florida

Field-programmable gate arrays (FPGA) are an increasingly attractive alternative to traditional microprocessor-based computing architectures in extreme-computing domains, such as aerospace and supercomputing. FPGAs offer several resource types that offer different tradeoffs between speed, power, and area, which make FPGAs highly flexible for varying application computational requirements. However, since an application's computational operations can map to different resource types, a major challenge in leveraging resource-diverse FPGAs is determining the optimal distribution of these operations across the device's available resources for varying FPGA devices, resulting in an extremely large design space. In order to facilitate fast design-space exploration, this article presents a method based on linear programming (LP) that determines the optimal operation distribution for a particular device and application with respect to performance, power, or dependability metrics. Our LP method is an effective tool for exploring early designs by quickly analyzing thousands of FPGAs to determine the best FPGA devices and operation distributions, which significantly reduces design time. We demonstrate our LP method's effectiveness with two case studies involving dot-product and distance-calculation kernels on a range of Virtex-5 FPGAs. Results show that our LP method selects optimal distributions of operations to within an average of 4% of actual values.

CCS Concepts: • **Hardware** → **Modeling and parameter extraction**; *Chip-level power issues*; • **Computer systems organization** → *Availability;*

Additional Key Words and Phrases: Dependability, design methodologies, FPGA, linear programming, optimization, power

ACM Reference Format:
Nicholas Wulf, Alan D. George, and Ann Gordon-Ross. 2017. Optimizing FPGA performance, power, and dependability with linear programming. ACM Trans. Reconfigurable Technol. Syst. 10, 3, Article 23 (June 2017), 23 pages.
DOI: http://dx.doi.org/10.1145/3079756

1. INTRODUCTION

Increasing demand for higher-performing, more power-efficient computing systems further compounds current high-performance computing challenges, such as memory bottlenecks [Mahapatra and Venkatrao 1999], the frequency wall [Flynn and Hung 2005], the power wall [Horowitz et al. 2005], and now impending feature-size limits [Iwai 2015]. Even though traditional microprocessor-based computing architectures

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. IIP-1161022. The authors gratefully acknowledge vendor equipment and tools provided by Xilinx that helped make this work possible.
Authors' addresses: N. Wulf, 2400 NE Palm Bay Road Bldg., HTC/C5434, Palm Bay, FL 32905; email: nwulf@harris.com; A. D. George, ECE Department, University of Pittsburgh, 1238D Benedum Hall, 3700 O'Hara Street, Pittsburgh, PA 15261; email: alan.george@pitt.edu; A. Gordon-Ross, 216 Larsen Hall, Gainesville, FL 32611; email: ann@ece.ufl.edu.

DOI: http://dx.doi.org/10.1145/3079756

continue to improve incrementally, reconfigurable computing devices such as field -programmable gate arrays (FPGA) can offer one to two orders of magnitude improvement over traditional architectures in computational performance and power efficiency [Guo et al. 2004; Underwood 2004]. Therefore, FPGAs are an increasingly attractive alternative to traditional computing, especially in extreme-computing domains, such as aerospace and supercomputing, where designers often attempt to push computing systems to their furthest limits to meet design goals such as computational performance and power efficiency. Many commercial, government, and academic entities have incorporated FPGAs in their designs, including NASA's SpaceCube family of reconfigurable on-board processing systems [Petrick et al. 2014], the university-developed CHREC Space Processor [Rudolph et al. 2014], the Ryft ONE commercial data analytics platform, the FPGA-based supercomputer Novo-G [George et al. 2011], and Microsoft's FPGA-accelerated Bing web search engine prototype [Putnam et al. 2014].

FPGAs afford significant speedups and power efficiency as compared to traditional computing using different hardware resource types, such as flip-flops (FF), look-up tables (LUT), and digital signal processing (DSP) units, which make FPGAs highly flexible for the varying computational operations among different applications. This flexibility enables designers to create different variants of the same operations using different FPGA resources (e.g., multiply operations with and without using DSP units). These different variants offer different tradeoffs between speed, power, and resource usage, resulting in a very large design space. Designers must determine the best operation distribution (e.g., which variants of operations to use and how many) that will optimally use the FPGA's resources to meet design goals and maximize FPGA performance and/or minimize power consumption. In addition to performance and power-consumption goals, designers must also work to meet a dependability goal for certain extreme-computing domains, such as aerospace or supercomputing, which involve harsh radiation environments and large numbers of processors, respectively. Finding the best operation distribution that most closely adheres to these three design goals simultaneously can be a daunting task for a designer.

The wide variety of available FPGA devices further complicates FPGA-based system design. Even if a designer narrows their choice down to a single FPGA vendor, there may still be hundreds of different FPGAs available for selection. FPGA vendors are constantly releasing newer generations of their technology to take advantage of shrinking feature sizes, and within each generation there may be multiple families that each specialize in a different area, such as low power and high dependability. Within each family, there are multiple resource specializations that make different tradeoffs between available FPGA resources, such as FPGAs with more DSPs, block memory units, or standard logic (FFs and LUTs). The number of available resources can also vary up to an order of magnitude between the largest and smallest FPGAs in a family. Finally, for each size and specialization, in addition to different packaging options for varying amounts of input/output bandwidth, there can be up to three different speed grades that can significantly increase the performance and price of the FPGA. With so many options available, it can be difficult for designers to select the best device to meet their design goals.

To help assist system designers in choosing the best device to meet their design goals, device vendors may report quantitative metrics for various device properties, including performance, power consumption, and dependability. For FPGAs, however, these properties are highly application-dependent, thus FPGA vendors either do not report these metrics or report metrics that represent the theoretical device limits. To accurately measure the properties of a particular FPGA while executing a specific application, a designer must write the application in a hardware design language (HDL) and use vendor tools to synthesize, place, and route the design for each considered device. Even

once the HDL is written, the process of placing and routing may require on the order of tens of hours to complete and would increase linearly for every additional operation distribution considered by the designer. Given these long testing times coupled with the plethora of available devices and potential operation distributions, designers must have a method to determine the best devices and operation distributions early in the design process to dramatically speed up design times.

To address this need, we present a method based on linear programming (LP) that, depending on an application's specific computational operations and a device's available resources, quickly determines the optimal operation distribution with respect to performance, power, or dependability and calculates quantitative metrics for design comparison purposes. Our LP method is more accurate than application-agnostic, vendor-supplied, first-order estimations and is capable of making relative comparisons between different FPGAs and operation distributions. Furthermore, designers can perform our LP method with a minimal description of the application, prior to HDL implementation, and without lengthy synthesis times. The speed of our LP method makes this method an effective tool for exploring early designs by quickly analyzing thousands of FPGAs to determine the best FPGA devices and operation distributions, which significantly reduces design time. We demonstrate our LP method's effectiveness by comparing the calculated metrics from our LP method to the experimental results of an optimized and synthesized FPGA design. Since lengthy synthesis times prohibit the rapid testing of many designs and devices, we focus our analysis on two case studies involving dot-product and distance-calculation kernels on a range of Virtex-5 FPGAs. Results show our LP method selects the optimal distribution of operations and predicts when increasing performance can decrease power efficiency and dependability.

The remainder of this article is organized as follows. Section 2 discusses the background that provides the foundation of our LP method and related work. In Section 3, we show the methodology behind the performance optimization of our LP method. Section 4 discusses how this performance optimization methodology can be modified to optimize for power or dependability instead. In Section 5, we show the results of two case studies involving dot-product and distance-calculation kernels on a range of Virtex-5 FPGAs. Finally, in Section 6, we discuss our conclusions and suggest a course for future research.

2. BACKGROUND AND RELATED WORK

LP is a powerful method for determining optimal solutions to problems that can be stated in terms of linear relationships. Since such a wide range of problems can fit into this format (e.g., maximizing profits while dealing with limited resources), LP is frequently used in a diverse set of domains, such as business, economics, industry, military, and engineering. Designers also use LP in many areas of circuit design to produce layouts that minimize latency, power, or error rates, while satisfying many simultaneous constraints. Landis et al. [1990] used LP techniques to improve fault detection and recovery while meeting multiple constraints, such as number of gates, latency, and power. Agrawal et al. [1999] used LP to minimize the imbalance in the gate input delays that cause transient energy consumption during a gate transition. By representing gate and buffer delays in combinatorial logic circuits with linear equations, they were able to reduce power by up to 47% in some instances compared to original circuits and find the optimal tradeoff between latency and power. Srinivasan et al. [2006] used LP to create power-optimized, network-on-chip architectures for application-specific, system-on-chip designs. Srinivasan et al. minimized total system power by creating a floor plan such that networking routes between processing cores were minimized while also ensuring that all cores and routing fit within the area of a bounding rectangle and that certain performance constraints were met.

In this article, we use LP to generalize a portion of our methodology using prior work by Williams et al. [2010], which established the computational density (CD) metric. CD measures a device's computational performance and is measured in operations per second (typically in giga operations per second (GOPS)). A device's CD depends on the precision and functions of the operations being analyzed (e.g., 8-bit integer add or double floating-point multiply). CD is useful for comparing performance between a wide range of processing devices, including CPUs, DSPs, FPGAs, and GPUs. To compute an FPGA's CD, Williams et al. considered four operations (combinations of an add or multiply function type with DSP and logic-only variants) and instantiated each operation one at a time on the FPGA to determine the resources consumed per operation. A simple analytical method then used this data to project how to optimally use the FPGA's resources to fit the most simultaneous operations on the FPGA. Williams et al. then calculated the FPGA's CD as the maximum number of simultaneous operations that could fit on the FPGA multiplied by the limiting frequency of the slowest operation. In this article, we introduce an LP method that generalizes Williams et al.'s methodology by considering any number of function types and operation variants when optimizing an FPGA for performance. We also discuss an improved method for handling the limiting frequency imposed by the slowest operation, which in some cases yields further performance improvements. Finally, we demonstrate how the flexibility of our LP method allows us to make small modifications to optimize an FPGA design for power or dependability instead of performance.

Other works have demonstrated non-LP methodologies for predicting the optimal performance of an application design on an FPGA. The RC Amenability Test (RAT) [Holland et al. 2009] is an analytical methodology that uses three tests for throughput performance, numerical precision, and resource utilization to determine the viability of an algorithm design on an FPGA prior to the use of any HDL. RAT measures throughput performance with both communication time for transferring data on and off the FPGA and computation time for processing the data according to an application design, which relies on a user-supplied frequency estimation for the FPGA. Enzler et al. [2000] described a similar high-level estimation methodology for characterizing the area and performance of an application on an FPGA. Using *a priori* information about the FPGA's architecture, the methodology created a set of equations to describe area, frequency, throughput, latency, and I/O pin count, enabling the user to quickly test the tradeoffs involved in decomposing parts of the design, replicating those parts, or adding registers for pipelining. Finally, Meswani et al. [2012] showed how to model and predict the performance of high-performance computing applications on systems that use GPU or FPGA hardware accelerators. Their model evaluated the application's code to find sections that could be easily accelerated and used simple benchmarks to predict accelerator speedup. In contrast to these methodologies, our LP method uses the results of single instantiated operations to predict the frequency and performance of an application on an FPGA without requiring detailed *a priori* information about the FPGA supplied by the user. Additionally, our LP method can consider multiple operation variants for any function type and can also optimize for power and dependability.

3. OPTIMIZING PERFORMANCE

Our LP method's primary purpose is to generalize the original CD methodology proposed by Williams et al. [2010] for FPGAs to determine the maximum performance for any number of function types and operation variants. The original CD methodology constrains analysis to only two function types (add and multiply) and two operation variants for each function type (a total of four operation variants), so the methodology only relies on a few simple calculations. Adding support for additional function types rather than just add and multiply enables analysis on a much broader range

of applications, and adding support for more operation variants gives designers more options to test when determining an FPGA's maximum performance. However, generalizing the methodology to consider additional function types and operation variants quickly increases the problem size beyond the capabilities of a few simple calculations, requiring the power and robustness of LP. The remainder of this section is organized as follows. Section 3.1 provides a brief discussion of how our LP method uses LP, Section 3.2 shows the necessary equations to describe the optimization problem and create the initial tableau for LP, and Section 3.3 describes how to extract results from the LP's final tableau.

3.1. Linear Programming (LP)

LP is a methodology for determining the values of a set of decision variables in a linear system that lead to an optimal result for an objective function. LP operates on a set of input linear constraint equations written in terms of the decision variables. For our LP method, the decision variables represent the quantity of each operation variant used in the FPGA's operation distribution. Every operation variant requires a unique combination of FPGA resource types, and each resource type is limited based on the FPGA device under consideration. Our LP method describes these resource limits with constraint equations that use the operation variant quantities as inputs. Additionally, the application running on the FPGA requires a specific ratio of various function types, which our LP method again represents with constraint equations using the operation variant quantities. When optimizing for performance, our LP method creates an equation for the objective function that defines the FPGA's performance in terms of the operation variants. Finally, our LP method combines the contents of these equations to create an initial tableau. Our LP method then uses the simplex algorithm [Vanderbei 2001] and Bland's rule [Bland 1977] to iteratively perform pivot operations on the initial tableau and create the final tableau. From the final tableau, our LP method can extract any necessary information to determine the optimal operation distribution and predict the optimal performance.

Our LP method does not capture information about or analyze routing distances between operations for several reasons. Our LP method is concerned with estimating the theoretical maximum performance for some combination of device and application, whereas routing inefficiencies are difficult to assess absolutely, often depending on routing methodology, routing effort, and/or programmer skill. Additionally, it might not only be difficult for a user to capture routing information about the operations, but it would be infeasible to integrate this information into the existing LP-based analysis, since routing constraints are not naturally linear and routing algorithms are very different from LP.

The simplex algorithm requires the optimization problem to be represented in standard form, which means that all decision variables must be nonnegative and all constraints must be represented with equations and not inequalities. It is impossible to use a negative number of operation variants, so the decision variables are necessarily nonnegative. Not only is the nonnegative variable requirement of the simplex algorithm automatically satisfied, but this requirement also prevents our LP method from needing additional equations to define the lower bounds of the decision variables. Unfortunately, many of the constraints are naturally represented with inequalities, which violate the equation requirement of the simplex algorithm, so these constraints must first be transformed into equations before the initial tableau is formed. An example of a simple constraint inequality is shown in Equation (1), where x is a decision variable with an upper limit of 100:

$$x \leq 100. \tag{1}$$

Table I. Virtex-5 LX20T Resources

FFs		LUTs		
Actual	**Usable**	**Actual**	**Usable**	**DSPs**
12,480	10,608	12,480	10,608	24

Table II. Properties of Virtex-5 LX20T Operation Variants

Function	Variant	FFs	LUTs	DSPs	Freq. (MHz)
Add	Small	64	64	0	362
Add	Large	170	210	0	401
Multiply	Logic	1,093	1,133	0	354
Multiply	Mixed	734	711	1	328
Multiply	DSP	81	32	4	500

By adding a new nonnegative slack variable s to the lesser side of the inequality and switching the inequality sign with an equal sign, the constraint inequality in Equation (1) is transformed into the equivalent constraint equation:

$$s + x = 100. \tag{2}$$

Slack variables are similar to decision variables in that slack variables are nonnegative and are used to define constraint equations, but slack variables do not represent the quantity of any operation variant. Since s is assumed to be nonnegative, Equation (2) requires that x can be no larger than 100, and so the constraint inequality of Equation (1) is preserved.

3.2. Creating the Initial Tableau

In order to demonstrate how our LP method forms the initial tableau required for LP, we discuss a base example using a specific FPGA device, the Virtex-5 LX20T-FF323-2, a dot-product kernel, and a set of operation variants. We chose to focus our example on the Virtex-5 LX20T, because it is the smallest device in the Virtex-5 family, which is suitable for a base example, and because Virtex-5 devices have a lower DSP-to-logic ratio than the more modern Virtex families, thereby enabling our LP method to make more interesting decisions. Dot product is an important kernel that is widely used within other basic kernels and applications (e.g., matrix multiplication, convolution). Before our LP method can begin creating the constraint equations, the user must define various properties for the device, application, and operation variants. Table I shows the necessary device properties for defining the Virtex-5 LX20T device. Our LP method defines the usable resource amounts for FFs and LUTs as 85% of the corresponding actual resource amounts, an adjustment identical to that of the CD methodology to account for a 15% logic resource overhead for steering logic and memory or I/O interfacing [Williams et al. 2010].

Our particular dot-product kernel receives two vectors consisting of multiple 32-bit integer values and outputs a single 64-bit integer, requiring 32-bit integer multiply operations and 64-bit integer add operations. Table II shows the data that is required for defining the set of operation variants used in our example. Each operation variant has a function, a variant name, the number of resources (i.e., FFs, LUTs, and DSPs) required to instantiate each instance of the operation variant, and the maximum achievable frequency at which the operation variant can operate correctly. For our example, we consider two add variants: a larger, fully pipelined add variant that can operate at a high frequency; and a smaller add variant that uses less resources but cannot operate as quickly. We also consider three multiply variants: a logic variant that uses no DSPs; a mixed variant that uses a mixture of DSPs and basic logic resources; and a DSP variant that uses almost no basic logic resources. To determine the properties of each

operation variant, a single instance of each operation variant was generated with the Xilinx CORE Generator System and instantiated on a Virtex-5 LX20T with Xilinx Integrated Synthesis Environment (ISE), the results of which provided the data in Table II.

With the data from Tables I and II, our LP method can create the resource-limiting equations (RLE), which are constraint equations that ensure the operation distribution does not lead to a design that uses more resources than are available on the considered device. One RLE is needed for each resource type, and each RLE describes how many of the corresponding resource are available on the device and how many each instance of an operation variant consumes. In our example, our LP method uses the inequalities shown in Equations (3), (5), and (7) to form the RLEs in Equations (4), (6), and (8). The decision variables x_{As}, x_{Al}, x_{Ml}, x_{Mm}, and x_{Md} represent the quantity of the small-add, large-add, logic-multiply, mixed-multiply, and DSP-multiply variants, respectively. The slack variables s_{FF}, s_{LUT}, and s_{DSP} represent the amount of unused FFs, LUTs, and DSPs, respectively,

$$64x_{\mathrm{As}} + 170x_{\mathrm{Al}} + 1093x_{\mathrm{Ml}} + 734x_{\mathrm{Mm}} + 81x_{\mathrm{Md}} \leq 10{,}608, \tag{3}$$

$$s_{\mathrm{FF}} + 64x_{\mathrm{As}} + 170x_{\mathrm{Al}} + 1093x_{\mathrm{Ml}} + 734x_{\mathrm{Mm}} + 81x_{\mathrm{Md}} = 10{,}608, \tag{4}$$

$$64x_{\mathrm{As}} + 210x_{\mathrm{Al}} + 1133x_{\mathrm{Ml}} + 711x_{\mathrm{Mm}} + 32x_{\mathrm{Md}} \leq 10{,}608, \tag{5}$$

$$s_{\mathrm{LUT}} + 64x_{\mathrm{As}} + 210x_{\mathrm{Al}} + 1133x_{\mathrm{Ml}} + 711x_{\mathrm{Mm}} + 32x_{\mathrm{Md}} = 10{,}608, \tag{6}$$

$$x_{\mathrm{Mm}} + 4x_{\mathrm{Md}} \leq 24, \tag{7}$$

$$s_{\mathrm{DSP}} + x_{\mathrm{Mm}} + 4x_{\mathrm{Md}} = 24. \tag{8}$$

Next, our LP method creates the function ratio equations (FRE) that enable us to target the specific application running on the device. Our LP method characterizes an application by the function types that an application comprises (e.g., add, multiply, divide, square root) and the ratio between those function types. For a dot-product kernel, the function ratio is approximately one add to one multiply, but for a fast Fourier transform kernel, the function ratio is approximately three adds to two multiplies. For an application with n different function types, our LP method requires an FRE for all but one of the n function types, resulting in a total of $n - 1$ FREs. To create an FRE for a particular function type, our LP method uses the generalized FRE shown in Equation (9), where x_i is the ith decision variable that represents an operation variant with the corresponding function, y_i is the ith decision variable that represents an operation variant with a different function, and α equals the fraction of the application's operations that correspond to the function type:

$$(1 - \alpha)\left(\sum x_i\right) - \alpha\left(\sum y_i\right) = 0. \tag{9}$$

For our example, there are only two function types, so our LP method only creates one FRE using the add function type, which is shown in Equation (10):

$$(1 - 0.5)(x_{\mathrm{As}} + x_{\mathrm{Al}}) - 0.5(x_{\mathrm{Ml}} + x_{\mathrm{Mm}} + x_{\mathrm{Md}}) = 0. \tag{10}$$

Finally, our LP method creates the objective function that relates the decision variables to a new objective variable, for which the intention is to maximize. Since our LP method is currently trying to maximize performance, the objective variable represents the performance of the device in terms of millions of operations per second (MOPS). Our computational model assumes that all operations operate in the same clock region, so the performance of the device can be calculated as the sum of all the operations used in the operation distribution scaled by the limiting frequency, which is the minimum achievable frequency amongst the operation variants being considered. For our example, the mixed-multiply variant sets the limiting frequency with the lowest

Table III. Summary of Equations

Type	Equation	Ref. Number
Objective Function	$z = 328(x_{\text{As}} + x_{\text{Al}} + x_{\text{Ml}} + x_{\text{Mm}} + x_{\text{Md}})$	12
FRE	$(1 - 0.5)(x_{\text{As}} + x_{\text{Al}}) - 0.5(x_{\text{Ml}} + x_{\text{Mm}} + x_{\text{Md}}) = 0$	10
FF RLE	$s_{\text{FF}} + 64x_{\text{As}} + 170x_{\text{Al}} + 1093x_{\text{Ml}} + 734x_{\text{Mm}} + 81x_{\text{Md}} = 10,608$	4
LUT RLE	$s_{\text{LUT}} + 64x_{\text{As}} + 210x_{\text{Al}} + 1133x_{\text{Ml}} + 711x_{\text{Mm}} + 32x_{\text{Md}} = 10,608$	6
DSP RLE	$s_{\text{DSP}} + x_{\text{Mm}} + 4x_{\text{Md}} = 24$	8

Table IV. Initial Tableau for Performance Optimization

	obj.	slack			decision					
	z	s_{FF}	s_{LUT}	s_{DSP}	x_{As}	x_{Al}	x_{Ml}	x_{Mm}	x_{Md}	
Obj. Func.	1	0	0	0	-328	-328	-328	-328	-328	0
FRE	0	0	0	0	0.5	0.5	-0.5	-0.5	-0.5	0
FF RLE	0	1	0	0	64	170	1093	734	81	10608
LUT RLE	0	0	1	0	64	210	1133	711	32	10608
DSP RLE	0	0	0	1	0	0	0	1	4	24

achievable frequency of 328MHz. Equation (11) shows the general objective function for performance optimization, where the objective variable z is the device's performance in MOPS, x_i is the ith decision variable, and f is the limiting frequency. Equation (12) shows the objective function for our example. Note that although Equation (11) assumes that every operation variant produces an output every cycle, Equation (11) can be easily modified to allow for operation variants that do not satisfy this assumption by scaling the associated decision variables:

$$z = f \sum x_i, \tag{11}$$

$$z = 328(x_{\text{As}} + x_{\text{Al}} + x_{\text{Ml}} + x_{\text{Mm}} + x_{\text{Md}}). \tag{12}$$

After creating the necessary RLEs, FREs, and objective function, our LP method creates the initial tableau. Table III summarizes and reformats these equations to clarify how these equations correspond to the rows of the initial tableau.

Table IV shows the initial tableau for our example. Our LP method constructs the tableau such that each row represents one of the equations from Table III (the only restriction is that the objective function must go into the topmost row to be easily handled by the simplex algorithm). Every column (except for the rightmost column) corresponds to either an objective variable, slack variable, or decision variable. Each element in the tableau shows the coefficient of the variable corresponding to the element's column in the equation corresponding to the element's row. The elements of the rightmost column represent the constant terms within the corresponding equations on the side opposite of the variables.

After creating the initial tableau, our LP method can apply the simplex algorithm to perform pivot operations on the tableau until the final tableau is produced. Due to the inclusion of the FRE, the tableau is not in canonical form (i.e., there is no subset of the tableau's columns that can be rearranged to create an identity matrix equal in height to the tableau), so our example requires a two-phase simplex algorithm, where Phase I transforms the tableau into canonical form, and Phase II produces the final tableau. Phase I requires the addition of a new artificial variable for each FRE, a single new artificial objective function, and a single new artificial objective variable. These additions transform the initial tableau into canonical form, and the rest of Phase I can begin to zero out the artificial variables using the artificial objective function. Assuming that at least one operation distribution exists that satisfies the FREs and RLEs (which is always true when optimizing for performance), Phase I successfully completes by

Table V. Final Tableau for Performance Optimization

obj.	slack			decision						
z	s_{FF}	s_{LUT}	s_{DSP}	x_{As}	x_{Al}	x_{Ml}	x_{Mm}	x_{Md}		
1	0.6	0	140.5	0	68.4	91.2	0	0	**10217.8**	$= z$
0	0.0	0	0.2	**1**	1.1	0.1	0	0	15.6	$= x_{\text{As}}$
0	0.0	0	-0.0	0	0.1	1.5	**1**	0	12.8	$= x_{\text{Mm}}$
0	-1.0	**1**	11.7	0	41.5	56.3	0	0	431.4	$= s_{\text{LUT}}$
0	-0.0	0	0.3	0	-0.0	-0.4	0	**1**	2.8	$= x_{\text{Md}}$

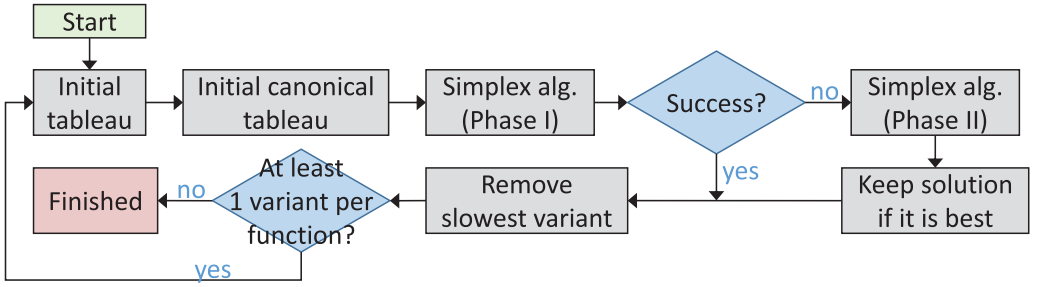

Fig. 1. Iterative process for testing multiple limiting frequencies.

producing a canonical tableau from which the added artificial terms can be dropped. Phase II then begins and completes by producing the final tableau, which is recognized by the absence of negative values in the top row.

3.3. Final Tableau and Results

Table V shows the final tableau for our example. Since the tableau must be canonical, there must exist a subset of five columns that can be rearranged to form an identity matrix. The variables associated with these columns are called basic variables, while the remaining variables are called nonbasic variables. Our LP method can quickly find the value for each basic variable in the rightmost column by assuming a value of zero for the nonbasic variables. Reading out the values in this manner shows that a maximum performance of about 10.2 GOPS is achieved when the operation distribution uses about 16 small-add operations, 13 mixed-multiply operations, and 3 DSP-multiply operations. The operation distribution does not use the large-add and logic-multiply variants to achieve maximum performance. The slack variables also show that the operation distribution uses all of the FFs and DSPs and almost all of the LUTs.

Although our example determines the maximum performance when considering all five operation variants, it may be possible to find a higher performance by considering only a subset of the operation variants. Although limiting the use of any operation variants can only reduce or have no effect on the total number of operations used by the operation distribution, the performance could still improve if the set of remaining operation variants has an improved limiting frequency. Unfortunately, the method for determining the limiting frequency of a set of operation variants is not linear, so our LP method must instead iteratively perform the LP steps described above multiple times to test alternative subsets of the operation variants and find the optimal limiting frequency. Figure 1 shows a flow diagram for this iterative process.

After the first iteration shown in our example, successive iterations remove the operation variant with the lowest achievable frequency, thereby enabling our LP method to test higher limiting frequencies with the remaining operation variants. This iterative process continues until any function type has lost all associated variants (a situation

Table VI. Iteration Results for Performance Optimization

Operation is Available? (# if used)							
Small Add	Large Add	Logic Mult.	Mixed Mult.	DSP Mult.	Limiting Freq. (MHz)	Simultaneous Operations	Performance (GOPS)
✓15.6	✓	✓	✓12.8	✓2.8	328	31.2	10.22
✓14.4	✓	✓8.4		✓6	354	28.8	10.17
✓6	✓			✓6	362	12.0	4.35
	✓6			✓6	401	12.0	4.81

in which the FREs would be impossible to satisfy), after which our LP method outputs the operation distribution with the highest performance. Table VI shows the results of each iteration. After the first iteration, the iteration process removes the mixed-multiply, logic-multiply, and finally small-add variants, leaving only the large-add and DSP-multiply variants for the final iteration. Although removing the large-add variant would further increase the limiting frequency to 500MHz, this removal makes it impossible to achieve the correct function ratio with only the DSP-multiply variant remaining. In our example, the first iteration produced the highest performance, yet our LP method cannot guarantee that this result is optimal without testing other alternative subsets of the operation variants as well. Note that in the case of an application that requires only one function type, there are no FREs, so the initial tableau is already canonical and Phase I of the simplex algorithm can be skipped. Although Phase I is always successful when optimizing for performance, Phase I may not complete successfully when optimizing for other goals, since these other goals require an additional constraint equation for the target performance.

Furthermore, this iterative process is the only divergence of our LP method's performance optimization (not including the power and dependability optimization discussed later) from the CD methodology (aside from the greater flexibility of our LP method to consider any number of function types and operation variants). The CD methodology actually optimizes for the maximum number of simultaneous operations on the device. Then the CD methodology finds the limiting frequency of the operation variants that were actually used and multiplies the number of operations by the limiting frequency to obtain the performance. The difference between the methodologies is subtle and does not always produce different results; however, in some situations, our LP method can find a more optimal operation distribution when considering the same operation variants as the CD methodology.

4. MODIFICATIONS TO OPTIMIZE FOR POWER OR DEPENDABILITY

Generally, there is only one operation distribution that can produce the optimal performance, thus our LP method cannot optimize for another design goal after optimizing for performance, because there would not be any other design options from which to choose. However, a designer may not need to achieve the maximum performance on a device if they already know how much performance their application needs. In cases where a designer is already targeting a specific level of performance for their application, there may be many operation distributions that can satisfy the performance demands, making it possible to optimize for alternative design goals. This section shows how to modify the performance optimization of our LP method to instead optimize an operation distribution for power or dependability for a given target performance.

4.1. Optimizing Power

Power-consumption design goals can be tantamount to performance goals, especially for certain extreme-computing domains. In the aerospace computing domain, power is often a limited resource, and certain situations may place more importance on meeting

Table VII. Estimated Power Consumption for Virtex-5 LX20T Operation Variants

		Dynamic Power Consumption (mW/MHz)				
Function	**Variant**	**FFs**	**LUTs**	**DSPs**	**Clock**	**Total**
Add	Small	0.007	0.010	0.000	0.006	0.023
Add	Large	0.019	0.057	0.000	0.025	0.101
Multiply	Logic	0.146	0.206	0.000	0.113	0.465
Multiply	Mixed	0.094	0.151	0.018	0.084	0.347
Multiply	DSP	0.009	0.014	0.072	0.011	0.106

a minimal power budget than meeting less extreme performance goals. Additionally, although supercomputers may have access to incredible amounts of computational resources, the actual costs required to pay for these resources places a large emphasis on increasing computational power efficiency. Due to the importance of power consumption as a design goal, we show how our LP method can optimize power for a target performance with only two modifications to the initial tableau: replacing the performance-based objective function with one for power and adding a target performance constraint.

In order to create a power-based objective function to replace the performance objective function, our LP method must know how much power is consumed by each operation variant. We use the Xilinx Power Estimator tool to estimate a frequency-normalized power value with units of mW/MHz, which enables our LP method to reuse this supplied data to determine power consumption for any operating frequency. There are two methods for estimating the power consumption of each operation variant. The first method is easiest and only requires information on the power consumption of the three main FPGA resources that compose all operation variants. With these three values, our LP method can estimate the power consumption of any operation variant by summing up the total power consumptions of the operation variant's constituent resources. A more accurate method involves importing data for each operation variant from Xilinx ISE into the power-estimator tool to directly determine a more accurate power estimate for each operation variant. For our work, we use the latter, more accurate method. Table VII shows the power-estimate results for all five operation variants in our example after importing data from Xilinx ISE into the power-estimator tool. Table VII also shows the contributions of the FPGA resources and clock tree to the total power consumption of each operation variant, though this data is not required by our LP method.

With the data from Table VII, our LP method can create the power-based objective function. Equation (13) defines the total dynamic power consumption (static power consumption is added on at the very end of the dynamic power analysis to calculate total power) as the sum of the power contributions of every operation in the operation distribution scaled by the limiting frequency. Since our goal is to maximize the objective variable, if we naively use power as the objective variable, then power is maximized instead of minimized. To circumvent this issue without significant alterations, we define the objective variable as the negative of the power consumption. In this way, as our LP method attempts to maximize the objective variable (negative power), it actually minimizes the device's power consumption (positive power). Equation (14) shows the new objective function for our example, where the objective variable z is now the negative of the total dynamic power consumption in mW:

$$\text{Power (mW)} = f(0.023x_{\text{As}} + 0.101x_{\text{Al}} + 0.465x_{\text{Ml}} + 0.347x_{\text{Mm}} + 0.106x_{\text{Md}}), \quad (13)$$

$$z + 7.6x_{\text{As}} + 33.2x_{\text{Al}} + 152.4x_{\text{Ml}} + 113.7x_{\text{Mm}} + 34.7x_{\text{Md}} = 0. \quad (14)$$

Next, our LP method creates the target performance equation (TPE), a new constraint equation that enables the targeting of a specific performance for the operation

Table VIII. Initial Tableau for Power Optimization

	obj.	slack			decision					
	z	s_{FF}	s_{LUT}	s_{DSP}	x_{As}	x_{Al}	x_{Ml}	x_{Mm}	x_{Md}	
Obj. Func.	1	0	0	0	7.6	33.2	152.4	113.7	34.7	0
TPE	0	0	0	0	328	328	328	328	328	7500
FRE	0	0	0	0	0.5	0.5	-0.5	-0.5	-0.5	0
FF RLE	0	1	0	0	64	170	1093	734	81	10608
LUT RLE	0	0	1	0	64	210	1133	711	32	10608
DSP RLE	0	0	0	1	0	0	0	1	4	24

Table IX. Final Tableau for Power Optimization

obj.	slack			decision					
z	s_{FF}	s_{LUT}	s_{DSP}	x_{As}	x_{Al}	x_{Ml}	x_{Mm}	x_{Md}	
1	0	0	26.3	0	25.7	12.4	0	0	**-1056.8**	$= z$
0	0	**1**	226.3	0	146.0	195.7	0	0	4583.1	$= s_{\text{LUT}}$
0	0	0	-0.0	**1**	1.0	-0.0	0	0	11.4	$= x_{\text{As}}$
0	0	0	-0.3	0	0.0	1.3	**1**	0	7.3	$= x_{\text{Mm}}$
0	**1**	0	217.7	0	106.0	141.3	0	0	4211.1	$= s_{\text{FF}}$
0	0	0	0.3	0	0.0	-0.3	0	**1**	4.2	$= x_{\text{Md}}$

distribution. Without the TPE, our LP method would always achieve a minimum power consumption of 0 Watts by removing all operations from the operation distribution. Equation (15) shows the general TPE, which is nearly identical to Equation (11), the general performance-based objective function. For our example, we target a performance of 7.5 GOPS, and Equation (16) shows the resulting TPE:

$$\text{Target Performance (MOPS)} = f \sum x_i, \tag{15}$$

$$328x_{\text{As}} + 328x_{\text{Al}} + 328x_{\text{Ml}} + 328x_{\text{Mm}} + 328x_{\text{Md}} = 7{,}500. \tag{16}$$

Our LP method creates the initial tableau for our example (Table VIII) in the same manner as before, except that now the TPE is inserted below the objective function. As seen in Table VIII, the TPE is similar to the DFEs and requires the addition of an extra artificial variable, meaning that our LP method can never skip Phase I of the simplex algorithm when targeting a specific performance. Furthermore, the TPE can cause Phase I to fail if the designer sets the target performance above the maximum performance. A failure in Phase I means that no viable solutions exist for the given constraints, which would obviously be true with an unachievable performance constraint. Aside from the fact that Phase I is now necessary and can potentially fail, our method operates on the initial tableau in the same manner as before to create the final tableau.

Table IX shows the final tableau for the first iteration of our example. Taking the negative of the objective variable shows a minimum dynamic power consumption of 1.057W for a dot-product kernel running on the Virtex-5 LX20T-FF323-2 at 7.5 GOPS. Table IX also shows that a significant number of FFs and LUTs are unused on the device, indicating that this design is indeed not optimized for performance.

Our LP method performs the iterative process (Figure 1) just as before to test the benefits of higher limiting frequencies. Table X shows the results of each iteration. Once again, the first iteration produced the best operation distribution. Note that Table VI shows that the final two iterations of the performance optimization that restrict usage of the logic-multiply and mixed-multiply variants have maximum performances below 5 GOPS, so these same subsets of operation variants fail (in Phase I of the simplex algorithm) in the final two iterations of power optimization, because the target performance of 7.5 GOPS is too high.

Table X. Iteration Results for Power Optimization

Operation is Available? (# if used)							
Small Add	Large Add	Logic Mult.	Mixed Mult.	DSP Mult.	Limiting Freq. (MHz)	Simultaneous Operations	Dynamic Power (W)
✓11.4	✓	✓	✓7.3	✓4.2	328	22.9	1.057
✓10.6	✓	✓4.6		✓6	354	21.2	1.069
✓	✓			✓	362	N/A	N/A
	✓			✓	401	N/A	N/A

Table XI. Estimated Error Rates for Virtex-5 LX20T Operation Variants

		Error Rate (errors/year)			
Function	Variant	FFs	LUTs	DSPs	Total
Add	Small	0.20	0.20	0.00	0.40
Add	Large	0.53	0.66	0.00	1.19
Multiply	Logic	3.43	3.56	0.00	6.99
Multiply	Mixed	2.30	2.23	0.10	4.63
Multiply	DSP	0.25	0.10	0.39	0.75

4.2. Optimizing Dependability

Dependability, represented here as mean time between failures (MTBF), is also an important design goal for extreme-computing domains. In aerospace, radiation-induced failures can cause systems with low dependability to suffer from data errors, downtime, and even catastrophic failure. Although devices on Earth do not typically suffer from as many failures, a supercomputer that comprises thousands of processing devices can suffer from a low total dependability if the failure of individual devices leads to system-wide failures. Since dependability is an important design goal in some situations, we show how our LP method can use the initial tableau for performance optimization and make two modifications (similar to the modifications for power optimization) to optimize for dependability instead.

In order to create a dependability-based objective function to replace the performance objective function, our LP method must define a linear relationship between the quantity of the operation variants and the total dependability. Unfortunately, the total MTBF of a system is not linear with respect to the MTBF of the constituent parts, thus our method uses the error rate (the reciprocal of MTBF) to define a linear relationship between the quantity of the operation variants and the total error rate and then converts the error rate to the total dependability. Estimating error rates requires first estimating the rate of upsets induced in the device by the operating environment and then determining the rate of errors caused by the upset rate (upsets occurring in unused areas of an FPGA do not cause errors). For our example, we consider an FPGA device operating on the International Space Station and predict an upset rate using CREME96 [Tylka et al. 1997] and Virtex-5 fault-injection data [Quinn et al. 2007; Hiemstra et al. 2010]. As with power estimations, we can estimate error rates either as per resource or per operation variant. The most accurate method measures the error rate for each operation variant by performing fault injection (radiation- or software-based) on a single instance of the operation variant. However, fault injection is beyond the scope of this article, so instead we measure the error rate for each FPGA resource using a worst-case estimation technique that assumes all configuration bits associated with a resource cause an error when upset. Table XI shows the error-rate estimation results for all five operation variants in our example. Table XI also shows the contributions of the FPGA resources to the total error rate of each operation variant to provide greater clarity on the sources of error in an FPGA, though this data is not required by our method.

Table XII. Initial Tableau for Optimizing Dependability

	obj.	slack			decision					
	z	s_{FF}	s_{LUT}	s_{DSP}	x_{As}	x_{Al}	x_{Ml}	x_{Mm}	x_{Md}	
Obj. Func.	1	0	0	0	0.40	1.19	6.99	4.63	0.75	0
TPE	0	0	0	0	328	328	328	328	328	7500
FRE	0	0	0	0	0.5	0.5	-0.5	-0.5	-0.5	0
FF RLE	0	1	0	0	64	170	1093	734	81	10608
LUT RLE	0	0	1	0	64	210	1133	711	32	10608
DSP RLE	0	0	0	1	0	0	0	1	4	24

Because our LP method only requires basic information about the operation variants as a whole (e.g., total resource usage, total power consumption, total error rate), without regard for their internal structure, our LP method can easily analyze any fault-tolerant operations with internal redundancies as well. Although dependability-focused designers would normally include fault-tolerant variants in their design analysis, we do not include them in our case studies for two reasons. First, to effectively demonstrate the effectiveness of our LP method, we must use case studies where the optimal solution is not trivial to find. Because fault-tolerant variants typically use extra resources and/or power for significant dependability improvements, it is almost certain that the dependability-optimized designs would use the fault-tolerant variants exclusively whenever they could, and the power-optimized designs would use the non-fault-tolerant variants. Second, because fault-tolerant variants are not available in the Xilinx CORE Generator System, any fault-tolerant variants that we create might not be optimized similarly to the other variants and would not provide a fair comparison.

With the data from Table XI, our LP method can create the dependability-based objective function. Equation (17) defines the total error rate as the sum of error rates of every operation in the operation distribution. Note that the limiting frequency plays no direct role in the calculation of the total error rate, because we model the error rates of the FPGA resources to be independent of the frequency (though models incorporating single-event transients could be used instead if desired). In order to maximize dependability, our LP method minimizes the error rate by defining the objective variable as the negative of the total error rate, similarly to the power-minimizing objective function from before. Equation (18) shows the new objective function for our example, where the objective variable z is now the negative of the total error rate measured in errors per year:

$$\frac{\text{Errors}}{\text{Year}} = 0.40x_{\text{As}} + 1.19x_{\text{Al}} + 6.99x_{\text{Ml}} + 4.63x_{\text{Mm}} + 0.75x_{\text{Md}}, \tag{17}$$

$$z + 0.40x_{\text{As}} + 1.19x_{\text{Al}} + 6.99x_{\text{Ml}} + 4.63x_{\text{Mm}} + 0.75x_{\text{Md}} = 0. \tag{18}$$

Since our example is still targeting a performance of 7.5 GOPS, we can reuse Equation (16) as the TPE for optimizing dependability. With the new objective function and TPE, our LP method creates the initial tableau (Table XII) similarly to that for power optimization, with the TPE once again inserted underneath the new objective function. As with the power optimization, the inclusion of the TPE means that Phase I of the simplex algorithm is now necessary and may potentially fail. Our LP method operates on the initial tableau similarly as before to create the final tableau.

Table XIII shows the final tableau for the first iteration of our example. Taking the negative of the objective variable shows a minimum error rate of 41.3 errors/year (or a maximum MTBF of 8.83 days) for a dot-product kernel running on the Virtex-5 LX20T-FF232-3 at 7.5 GOPS. Except for the top row, the final tableaus in our example for the first iteration of the power and dependability optimizations are identical, meaning that this operation distribution is simultaneously ideal for power and dependability.

Table XIII. Final Tableau for Dependability Optimization

obj.	slack			decision						
z	s_{FF}	s_{LUT}	s_{DSP}	x_{As}	x_{Al}	x_{Ml}	x_{Mm}	x_{Md}		
1	0	0	1.3	0	0.8	1.1	0	0	**-41.3**	$= z$
0	0	**1**	226.3	0	146.0	195.7	0	0	4583.1	$= s_{\text{LUT}}$
0	0	0	-0.0	**1**	1.0	-0.0	0	0	11.4	$= x_{\text{As}}$
0	0	0	-0.3	0	0.0	1.3	**1**	0	7.3	$= x_{\text{Mm}}$
0	**1**	0	217.7	0	106.0	141.3	0	0	4211.1	$= s_{\text{FF}}$
0	0	0	0.3	0	0.0	-0.3	0	**1**	4.2	$= x_{\text{Md}}$

Table XIV. Iteration Results for Dependability Optimization

Operation is Available? (# if used)							
Small Add	**Large Add**	**Logic Mult.**	**Mixed Mult.**	**DSP Mult.**	**Limiting Freq. (MHz)**	**Simultaneous Operations**	**Dependability (MTBF:days)**
✓11.4	✓	✓	✓7.3	✓4.2	328	22.9	8.833
✓10.6	✓	✓4.6		✓6	354	21.2	8.925
✓	✓			✓	362	N/A	N/A
	✓			✓	401	N/A	N/A

Our LP method performs the iterative process (Figure 1) similarly as before to test the benefits of higher limiting frequencies. Table XIV shows the results of each iteration. Surprisingly, even though each iteration produces the same operation distributions as were produced for the power optimization, the second iteration is actually the optimal choice for dependability optimization with an MTBF of 8.93 days. For a fixed target performance, increasing the limiting frequency requires a proportional decrease in the number of simultaneous operations. Since power is proportional to both frequency and quantity of simultaneous operations, increasing the limiting frequency with a fixed target performance has no direct effect on power consumption. However, since the total error rate is only proportional to the quantity of simultaneous operations and not to the frequency, increasing the limiting frequency directly improves dependability (although limiting the availability of certain operation variants may still produce overall worse results for higher frequencies).

5. RESULTS AND ANALYSIS

This section discusses the setup and results of two case studies, each using a different kernel to test a variety of capabilities for a wide-range evaluation of our LP method's effectiveness. Section 5.1 concludes this article's example of using a base case study with the Virtex-5 LX20T running a dot-product kernel to experimentally determine the minimum achievable power for various performance values using this setup. Section 5.2 introduces a more complex case study involving a wide range of Virtex-5 devices running distance-calculation kernels, which significantly increases the number of operation variants analyzed in our method.

5.1. Base Case Study: Dot Product

Our base case study tests our LP method's predictions of this article's base example involving a Virtex-5 LX20T device running a dot-product kernel. To test these predictions, we create multiple designs of a dot-product kernel on the Virtex-5 LX20T using various combinations of the five operation variants defined in Tables II, VII, and XI. After placing and routing the designs in Xilinx ISE with a high effort level, we obtain the frequency of the design, which we multiply by the design's number of operations to calculate the design's performance in GOPS. We then import the designs into the Xilinx Power Estimator tool to estimate each design's power. Because this power estimator

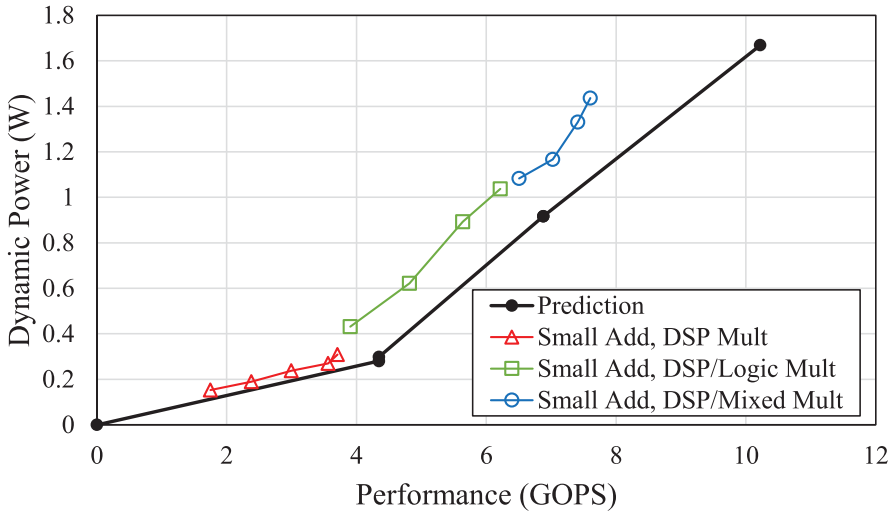

Fig. 2. Power predictions and results for designs using minimum power on Virtex-5 LX20T running dot-product kernel.

only ingests the map report files from our designs, the contribution of FPGA routing to the total power consumption is estimated based on the average fanout of the various FPGA resources. Because fault injection is a complicated process that is beyond the scope of this article, we do not measure the dependability of the designs. By comparing the power and performance of various design sizes and operation distributions, we measure the minimum required dynamic power for a given performance and determine which designs are actually most power-efficient for various ranges of performance:

$$\mathbf{a} \cdot \mathbf{b} = \sum a_i b_i. \tag{19}$$

Equation (19) defines a dot-product operation for vectors $\mathbf{a}$ and $\mathbf{b}$, where a_i and b_i are the ith entries in vectors $\mathbf{a}$ and $\mathbf{b}$, respectively. A single dual-port block RAM supplies each set of corresponding 32-bit integer input entries (i.e., a_i and b_i), and a simple address generator drives each of these block RAMs, striding across the block RAM memory. Because small modifications to these address generators to alter the striding patterns would provide the necessary control logic to target other specific applications (e.g., specific matrix sizes in matrix multiply), the results from this case study are broadly applicable to a wide range of applications that rely heavily on the dot-product kernel. For each block RAM, a single 32-bit integer multiply operation calculates the product of the block RAM's two 32-bit integer values and outputs the 64-bit integer product. Finally, an add-tree consisting of 64-bit integer add operations sums all of these products in parallel and outputs the final answer. The design is fully pipelined, meaning the device performs a full dot-product calculation every cycle. We vary the performance per cycle of the dot-product kernel by varying the length of the input vectors.

Figure 2 shows our LP method's predictions, as well as the most power-efficient designs for various performance values in GOPS. Predictions show that designs should only use the DSP-multiply variant for multiply operations when targeting a performance below 4.35 GOPS. At 4.35 GOPS, all of the DSP units are needed to keep up with input vectors of length six. Above 4.35 GOPS, increasing the length of the input vectors requires additional multiply operations, which requires using the remaining unused logic resources. As we add more power-hungry, logic-multiply operations, dynamic

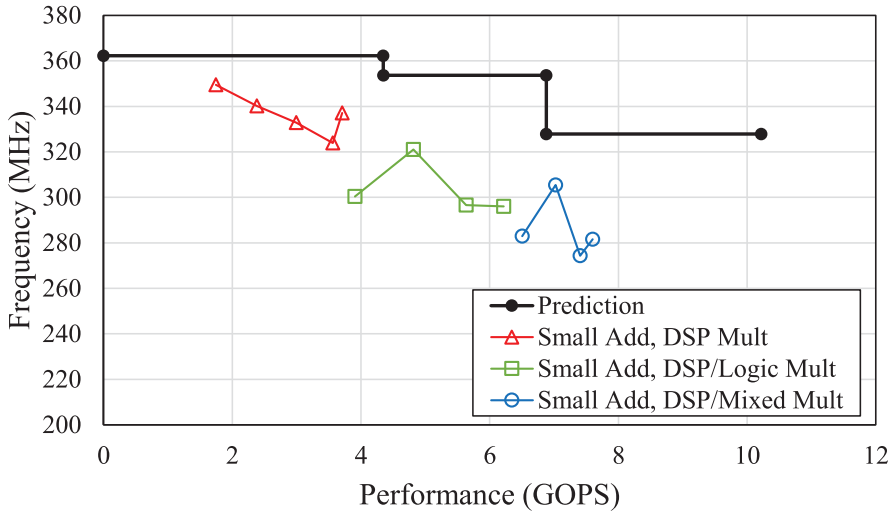

Fig. 3. Frequency predictions and results for designs using minimum power on Virtex-5 LX20T running dot-product kernel.

power consumption rises sharply. For targeted performances of 6.88 GOPS and above, the most power-efficient designs completely avoid the logic-multiply variant. Instead, mixed-multiply operations gradually replace some of the DSP-multiply operations, starting with around 57% of the multiply operations being mixed-multiply variants and progressing to around 82%. Our method predicts that the maximum performance operation distribution consists of 15.6 small-add, 2.8 DSP-multiply, and 12.8 mixed-multiply operations, for a total performance of 10.22 GOPS and dynamic power consumption of 1.669 W. Finally, predictions show that using the large-add variant does not increase maximum performance or power efficiency. The higher frequency of the large-add variant does not improve designs using mixed or logic-multiply variants, which operate at a lower frequency. Even when the mixed and logic-multiply variants are absent in lower-performance designs, the extra logic resource overhead of the large-add variant offsets the power-efficiency of the DSP-multiply variant.

Figure 2 shows the experimental results, which largely confirm our LP method's predictions. For greatest power efficiency, the large-add variant is never beneficial, low-performance designs require only the DSP-multiply variant, mid-performance designs require both logic and DSP-multiply variants, and high-performance designs require mixed and DSP-multiply variants. We achieve the highest performance of 7.60 GOPS using two DSP-multiply and 12 mixed-multiply operations. Furthermore, for the highest possible power efficiency (calculated as performance divided by power consumption), our method suggests to only use small-add and DSP-multiply operations and to avoid filling up the device any further with the other variants. This insight may be important if the cost of power (or lack of power) is the designer's primary concern rather than maximum performance or the cost of hardware. In such a situation, using multiple devices at less-than-maximum capacity may be preferable to using more total power on fewer fully utilized devices.

Several effects are responsible for the differences between our LP method's predictions and the experimental results. The primary effect comes from the difference between the predicted and achievable frequencies, shown in Figure 3. The increased complexity of a full design over just a single instance of an operation variant increases the difficulty of placing and routing the design, which can lead to lower achievable

frequencies. A reduction in frequency results in a proportional reduction in performance and dynamic power consumption, causing the points in Figure 2 to shift toward the origin. For this reason, graphs of experimental results should look like shrunken versions of our predicted results.

Smaller secondary effects result in additional small experimental differences from predictions. The logic overhead required to support the block RAMs and block RAM address generation results in a slight increase in dynamic power consumption for all designs. For designs using logic and mixed-multiply variants, extra registers are needed to efficiently pipeline the designs and meet timing. These extra registers significantly increase the dynamic power consumption of the mid/high-performance designs and limit designs from being able to handle vector lengths of 15 as predicted, resulting in decreases to the maximum achievable performance as well. Finally, the larger than expected drop in frequency when adding the logic-multiply variant results in a larger than expected increase in dynamic power consumption at around 4 GOPS.

5.2. Complex Case Study: Distance Calculation

Our complex case study is similar to the base case study but increases the complexity of our LP method's analysis by focusing on a 32-bit floating-point distance-calculation kernel, which involves a greater number of function types and operation variants. Furthermore, this case study tests our method's performance optimization across the full range of sizes in the Virtex-5 LXT subfamily and tests power optimization on the mid-size Virtex-5 LX85T device.

Distance calculations are common in numerous applications in super computing (e.g., physical simulations and complex-value mathematics) and aerospace (e.g., star tracking using planar triangles [Cole and Crassidis 2006]). The distance-calculation kernel used in this case study involves repeatedly performing the 2D-distance calculation shown in Equation (20) on a series of **a** and **b** vectors, where **a** and **b** are 2D-Cartesian coordinate vectors with 32-bit floating point values for the x and y entries. Similar to the dot-product kernel, dual-port block RAMs driven by address generators supply each of the corresponding entries of the input vectors, requiring one block RAM for the x coordinates and one for the y coordinates. For each block RAM, a subtract operation computes the difference of the block RAM's two 32-bit floating-point values, and the result goes to both inputs of a multiply operation to compute the squared difference between the corresponding coordinates. An addition operation then sums these two results and passes the sum to a square-root operation, which outputs the final 32-bit floating-point answer. The described design represents a single distance-calculation core, which is fully pipelined and therefore performs a distance calculation every cycle, and which consists of two add, two multiply, one subtract, and one square-root operations. We can vary the performance per cycle of the kernel by increasing the number of distance-calculation cores included in the design:

$$d = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}. \tag{20}$$

To demonstrate the capabilities of our LP method, we consider all operation variants available in the floating-point library of the Xilinx CORE Generator System for each function type in our design. For both the add and subtract function types, there is a logic-only variant and a DSP variant that uses two DSP units. Since the subtract variants are so similar to the corresponding add variants, we group the subtract variants with the corresponding add variants for our LP method's analysis to simplify the results without sacrificing any accuracy. For the multiply function type, there is a logic-only multiply variant, a medium-multiply variant that uses one DSP unit, a full-multiply variant that uses two DSP units, and a max-multiply variant that uses three DSP

Table XV. Virtex-5 LX85T Operation Variant Properties

Function	Variant	FFs	LUTs	DSPs	Max Freq. (MHz)	Dyn. Power (mW/MHz)
Add/Sub	Logic	546	416	0	508	0.213
Add/Sub	DSP	327	230	2	504	0.153
Multiply	Logic	681	619	0	454	0.283
Multiply	Medium	368	258	1	493	0.147
Multiply	Full	171	103	2	515	0.0969
Multiply	Max	106	90	3	497	0.0966
Square Root	Logic	765	531	0	503	0.266

units. Only the logic-only variant is available for the square-root function type. For each device in this case study, we measure the resource consumptions, maximum achievable frequency, and dynamic power consumption of each operation variant using Xilinx ISE with a high-effort place and route level and the Xilinx Power Estimator tool. Table XV shows the results of these measurements for only the Virtex-5 LX85T-FF1136-3, but these results are similar across the entire set of studied devices, with the exception of proportional reductions in achievable frequencies for devices of a slower speed grade.

This case study investigates performance optimization on all eight unique sizes of the Virtex-5 LXT subfamily to show how our LP method performs on a wide range of FPGA sizes. We focus the case study on the Virtex-5, because this family of FPGAs has a lower DSP-to-logic ratio than the more modern Virtex-6 and Virtex-7 FPGAs. Using devices with lower DSP-to-logic ratios helps to demonstrate our LP method's decision-making process, since the most power efficient computational resources on the device (i.e., DSP units) are limited and must be used intelligently. We focus the case study on the LXT subfamily of the Virtex-5 family, because this subfamily has the largest range of resource amounts, with the largest member, the Virtex-5 LX330T, containing over sixteen times the number of logic resources as the smallest member, the Virtex-5 LX20T. Since package size and speed grade have negligible and predictable effects, respectively, we only investigate the fastest speed grade on the smallest package for each unique size of the Virtex-5 LXT subfamily.

Table XVI shows our LP method's predictions for the highest-performance designs on each device, as well as the highest-performance designs that we can experimentally achieve. As with the base case study, we test these predictions by creating many designs of the distance-calculation kernel on each device using various combinations of the nine available operation variants. After placing and routing the designs in Xilinx ISE with a high-effort level, we obtain the frequency of the design, which we multiply by the number of operations in the design to calculate the performance of the design in GOPS. We select the highest-performing design for each device and report the features of these designs in Table XVI, including the operation distributions of the design, the percent of the device's total resources that the design uses, and the percent of predicted performance that we can achieve.

Table XVI shows that our LP method accurately predicts the correct operation distribution that produces the optimal-performance design. For the smaller devices that have higher DSP-to-logic ratios, it correctly predicts that some of the add operations should be of the DSP variant and all of the multiply operations should be of the full variant, which uses two DSPs and a small amount of logic. For the larger devices with lower DSP-to-logic ratios, it correctly recommends using more logic-centric variants by using only the logic variant for the add operations and a combination of medium and full variants for the multiply operations.

Average resource utilizations of 85.5% for FFs and 95.8% for DSPs confirm our LP method's assumptions of a 15% overhead for logic resources and 0% overhead for DSPs.

Table XVI. Predictions and Results of Performance Optimization for Distance-Calculation Kernel

Virtex-5 Device	Variants Predicted Add/Sub	Variants Predicted Mult.	Variants Used Add/Sub	Variants Used Mult.	Achieved Resource Utilization % FFs	LUTs	DSPs	Performance (GOPS) Pred.	Result	RU
LX20T	70% Logic 30% DSP	Full	75% Logic 25% DSP	Full	88.3%	58.7%	91.7%	10.96	9.40	85.7%
LX30T	81% Logic 19% DSP	Full	78% Logic 22% DSP	Full	86.6%	57.7%	100.0%	18.26	15.90	87.1%
LX50T	81% Logic 19% DSP	Full	78% Logic 22% DSP	Full	86.6%	57.7%	100.0%	27.47	18.97	69.0%
LX85T	Logic	42% Med. 58% Full	Logic	40% Med. 60% Full	89.0%	60.6%	100.0%	44.80	28.90	64.5%
LX110T	Logic	42% Med. 58% Full	Logic	40% Med. 60% Full	89.0%	60.6%	100.0%	58.36	38.59	66.1%
LX155T	Logic	89% Full 11% Max	Logic	93% Med. 7% Max	83.8%	56.7%	90.6%	89.99	59.26	65.9%
LX220T	Logic	41% Med. 59% Full	Logic	39% Med. 61% Full	81.6%	54.4%	90.6%	102.55	56.95	55.5%
LX330T	Logic	41% Med. 59% Full	Logic	33% Med. 67% Full	79.4%	54.0%	93.8%	156.69	82.40	52.6%

We could not test the overhead for LUTs in this case study, because every operation variant uses more FFs than LUTs, and every tested device contains an equal number of FFs and LUTs, so LUTs can never be a limiting resource. We also note that it is possible to use more than 85% of logic resources in a design, but doing so increases design complexity and reduces achievable frequencies, which ultimately offsets any advantage from an increased number of operations and reduces performance. Therefore, the 15% logic overhead value does not represent a hard limit but is useful when optimizing for maximum performance.

Since our LP method is based on the CD methodology, which predicts the theoretical maximum performance for a device, we expect the experimentally achieved maximum performance for each device to be some proportion of our maximum performance prediction. Richardson et al. [2012] describe a realizable utilization (RU) metric to quantify the difference between theoretical device performance shown by CD and the performance designers can achieve. For smaller devices, the RU score reaches around 86%, but as device size increases, the RU score steadily falls to as low as 52%. The discrepancy between predicted design performance and achievable performance is primarily caused by a discrepancy in design frequencies, which proportionately affects performance and dynamic power consumption. The Xilinx ISE place and route process is able to obtain slightly higher frequencies for the operation variants when measured in isolation than when the operation variants are included in an entire design, and this effect increases for the larger devices as routing complexity increases. Overall, the theoretical maximum performance predicted by our method is a good first-order estimate of achievable performance on a particular device, and any known RU scores for similar devices running similar applications can enhance our predictions even further. Furthermore, even without *a priori* RU scores, our LP method serves as a useful tool in predicting the relative performance between similarly sized devices.

To test our LP method's power optimization on this more complex case study, we investigate the mid-sized Virtex-5 LX85T across a range of performances. With knowledge of the Virtex-5 LX85T RU score, we scale the operating frequency of the device's operation variants by 64.5% to improve the accuracy of our predictions. Figure 4 shows our minimum dynamic-power predictions, as well as the most power-efficient designs

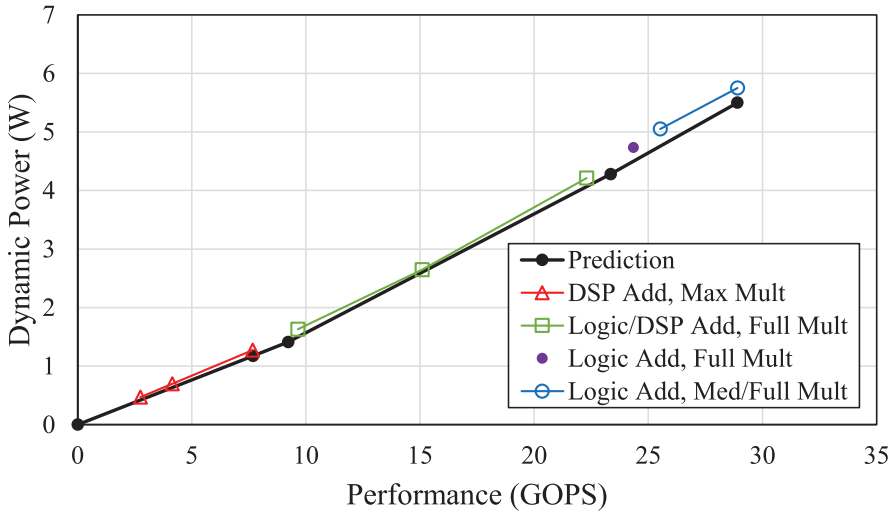

Fig. 4. Power predictions and results for designs using minimum power on Virtex-5 LX85T running distance-calculation kernel.

achievable for various performance values. Predictions show that designs should only use the DSP-add and max-multiply variants when targeting a performance below 7.68 GOPS. At 7.68 GOPS, all of the DSP units are needed to keep up with four simultaneous distance-calculation cores. Just after 7.68 GOPS, our method suggests trading the max-multiply operations for full-multiply operations until 9.22 GOPS, where all multiply operations are of the full variant. After 9.22 GOPS, logic-add operations start replacing DSP-add operations until 23.3 GOPS, where all add operations are of the logic variant. Beyond 23.3 GOPS, predictions recommend replacing full-multiply operations with medium-multiply operations until 28.90 GOPS, where a design consisting of 15.15 distance-calculation cores is using all available FF and DSP resources.

Figure 4 shows experimental results that confirm our LP method's predictions for minimum power after accounting for the device's RU score. In comparison to the base case study, the logic overhead required to support the block RAMs and block RAM address generators is relatively small as compared to the increased number of operations required to operate on the data from the block RAMs, so the extra power increase is smaller than in the previous case study as well. Additionally, unlike with the base case study, the distance-calculation kernel does not require extra power-consuming registers to help with efficient pipelining.

Finally, our LP method suggests that max-multiply operations should progressively replace full-multiply operations between 7.67 and 9.22 GOPS, but this is untrue experimentally. This discrepancy occurs because our method assumes that the quantities of resources, operations, and distance-calculation cores are continuous values rather than integer values. At 7.67 GOPS, the minimum-power design uses only DSP-add and max-multiply operations to create four distance-calculation cores. However, at 9.22 GOPS, our method recommends using only the full variant for multiply operations, which allows for a maximum of 4.8 distance-calculation cores before the design requires all of the DSP units. Since the max-multiply variant is best when using only four distance-calculation cores, and we cannot actually design a fraction of a distance core, designs using full-multiply operations without using logic-add operations are never power-optimal. Fortunately, this issue with using continuous values is minimal

when the size of the device is significantly larger than the size of an application's computational cores, so most of our other predictions are accurate.

6. CONCLUSIONS

In this article, we have introduced our LP method, an effective tool for exploring early designs by quickly determining the optimal operation distribution for a particular device and application with respect to performance, power, or dependability and calculating quantitative metrics for design comparison purposes. Our LP method is a more powerful generalization of the established CD methodology and still requires that designers characterize the resource usage and operating frequency of any operations considered for a design.

To demonstrate our LP method's capabilities, we conducted a base and a complex case study. The base case study analyzed power optimization for a dot-product kernel on the smallest Virtex-5 device. The complex case study analyzed performance optimization on a wide range of Virtex-5 devices and again analyzed power optimization, but on a mid-size Virtex-5 device as opposed to the smallest device in the base case study. The complex case study also increased the complexity of the analysis by investigating a distance-calculation kernel involving a more diverse set of function types and operation variants. Results of the case studies show that our method accurately predicts the operation distribution (within an average of 4% of actual values) that can achieve maximum performance and provides reasonable estimates for the amount of performance a designer can reach. Results also show that our method can accurately recommend operation distributions for creating designs that achieve a given performance with minimum power consumption, although these predictions may need to be supplemented with the device's RU score to give absolute rather than relational information. Overall, the results demonstrate that our method can help designers to compare devices, predict design metrics, and select the optimal operation distribution to best meet their design goals.

Future work includes expanding our LP method to analyze hybrid devices that contain both reconfigurable and fixed logic resources (e.g., the Xilinx Zynq, which contains two embedded ARM processors). This expansion would involve gathering data on the fixed logic resources using the fixed-logic CD methodology and then representing this data as linear equations that could be included with the existing equations describing the reconfigurable logic. Future work in studying the RU scores of other FPGA devices and common kernels would help refine the accuracy of our method for those devices and kernels and may result in a general methodology for predicting decreasing design frequencies without the need for *a priori* RU score measurements.

REFERENCES

V. D. Agrawal, M. L. Bushnell, G. Parthasarathy, and R. Ramadoss. 1999. Digital circuit design for minimum transient energy and a linear programming method. In *Proceedings of the Twelfth International Conference on VLSI Design*. 434–439. DOI:http://dx.doi.org/10.1109/ICVD.1999.745194

R. G. Bland. 1977. New finite pivoting rules for the simplex method. *Math. Operat. Res.* 2, 2 (1977), 103–107. DOI:http://dx.doi.org/10.1287/moor.2.2.103

C. L. Cole and J. L. Crassidis. 2006. Fast star-pattern recognition using planar triangles. *J. Guid. Contr. Dynam.* 29, 64–71. DOI:http://dx.doi.org/10.2514/1.13314

R. Enzler, T. Jeger, D. Cottet, and G. Tröster. 2000. *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing (FPL'00)*. Springer Berlin. 525–534. DOI:http://dx.doi.org/10.1007/3-540-44614-1_57

M. Flynn and P. Hung. 2005. Microprocessor design issues: Thoughts on the road ahead. *IEEE Micro.* 25, 3, 16–31. DOI:http://dx.doi.org/10.1109/MM.2005.56

A. George, H. Lam, and G. Stitt. 2011. Novo-G: At the forefront of scalable reconfigurable supercomputing. *Comput. Sci. Eng.* 13, 1 (2011), 82–86. DOI:http://dx.doi.org/10.1109/MCSE.2011.11

Z. Guo, W. Najjar, F. Vahid, and K. Vissers. 2004. A quantitative analysis of the speedup factors of FPGAs over processors. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA'04)*. ACM, New York. 162–170. DOI:http://dx.doi.org/10.1145/968280.968304

D. M. Hiemstra, G. Battiston, and P. Gill. 2010. Single event upset characterization of the virtex-5 field programmable gate array using proton irradiation. In *Proceedings of the 2010 IEEE Radiation Effects Data Workshop (REDW'10)*. 1–4. DOI:http://dx.doi.org/10.1109/REDW.2010.5619490

B. Holland, K. Nagarajan, and A. D. George. 2009. RAT: RC amenability test for rapid performance prediction. *ACM Trans. Reconfig. Technol. Syst.* 1, 4, Article 22. DOI:http://dx.doi.org/10.1145/1462586.1462591

M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein. 2005. Scaling, power, and the future of CMOS. In *Proceedings of the IEEE International Electron Devices Meeting, 2005. IEDM Technical Digest*. 7–15. DOI:http://dx.doi.org/10.1109/IEDM.2005.1609253

H. Iwai. 2015. Future of nano {CMOS} technology. *Solid-State Electron.* 112 (2015), 56–67. DOI:http://dx.doi.org/10.1016/j.sse.2015.02.005

D. L. Landis, J. R. Samson, and J. H. Aldridge. 1990. *Defect and Fault Tolerance in VLSI Systems: Volume 2*. Springer, Boston, MA. 267–281. DOI:http://dx.doi.org/10.1007/978-1-4757-9957-6_22

N. R. Mahapatra and B. Venkatrao. 1999. The processor-memory bottleneck: Problems and solutions. *Crossroads* 5, 3, Article 2. DOI:http://dx.doi.org/10.1145/357783.331677

M. R. Meswani, L. Carrington, D. Unat, A. Snavely, S. Baden, and S. Poole. 2012. Modeling and predicting performance of high performance computing applications on hardware accelerators. In *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW'12)*. 1828–1837. DOI:http://dx.doi.org/10.1109/IPDPSW.2012.226

D. Petrick, A. Geist, D. Albaijes, M. Davis, P. Sparacino, G. Crum, R. Ripley, J. Boblitt, and T. Flatley. 2014. SpaceCube v2.0 space flight hybrid reconfigurable data processing system. In *Proceedings of the 2014 IEEE Aerospace Conference*. 1–20. DOI:http://dx.doi.org/10.1109/AERO.2014.6836226

A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceedings of the 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. 13–24. DOI:http://dx.doi.org/10.1109/ISCA.2014.6853195

H. Quinn, K. Morgan, P. Graham, J. Krone, and M. Caffrey. 2007. Static proton and heavy ion testing of the Xilinx virtex-5 device. In *Proceedings of the IEEE Radiation Effects Data Workshop*. 177–184. DOI:http://dx.doi.org/10.1109/REDW.2007.4342561

J. W. Richardson, A. D. George, and H. Lam. 2012. Performance analysis of GPU accelerators with realizable utilization of computational density. In *Proceedings of the 2012 Symposium on Application Accelerators in High Performance Computing (SAAHPC'12)*. 137–140. DOI:http://dx.doi.org/10.1109/SAAHPC.2012.13

D. Rudolph, C. Wilson, J. Stewart, P. Gauvin, G. Crum, A. D. George, M. Wirthlin, and H. Lam. 2014. CSP: A multifaceted hybrid system for space computing. In *Proceedings of the 28th Annual AIAA/USU Conference on Small Satellites*. 1–7.

K. Srinivasan, K. S. Chatha, and G. Konjevod. 2006. Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 14, 4, 407–420. DOI:http://dx.doi.org/10.1109/TVLSI.2006.871762

A. J. Tylka, J. H. Adams, P. R. Boberg, B. Brownstein, W. F. Dietrich, E. O. Flueckiger, E. L. Petersen, M. A. Shea, D. F. Smart, and E.C. Smith. 1997. CREME96: A revision of the cosmic ray effects on micro-electronics code. *IEEE Trans. Nucl. Sci.* 44, 6, 2150–2160.

K. Underwood. 2004. FPGAs vs. CPUs: Trends in peak floating-point performance. In *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA'04)*. ACM, New York. 171–180. DOI:http://dx.doi.org/10.1145/968280.968305

R. J. Vanderbei. 2001. *Linear Programming: Foundations and Extensions*. Springer.

J. Williams, A. D. George, J. Richardson, K. Gosrani, C. Massie, and H. Lam. 2010. Characterization of fixed and reconfigurable multi-core devices for application acceleration. *ACM Trans. Reconfig. Technol. Syst.* 3, 4, 19:1–19:29.

Received April 2016; revised December 2016; accepted February 2017