

Energy-Aware System Synthesis for Reconfigurable Chip Multiprocessors

Xiaofang Wang

Electrical and Computer Engineering
Villanova University
Villanova, PA 19085

Sotirios G. Ziavras and Jie Hu

Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102

Abstract - *Even though state-of-the-art FPGAs present new opportunities in exploring low-cost high-performance architectures for floating-point scientific applications, they also pose serious challenges. Multiprocessors-on-a-Programmable-Chip (MPoPCs), which integrate both software-programmability and hardware-reconfigurability, provide substantial flexibility that can result in programming ease and high performance. This paper presents an application-oriented system design methodology for HERA (HEterogeneous Reconfigurable Architecture), an in-house developed MPoPC that targets applications involving large matrices. HERA is a mixed computing-mode multiprocessor supporting simultaneous execution in SIMD (Single-Instruction, Multiple-Data), MIMD (Multiple-Instruction, Multiple-Data), and multiple-SIMD for chosen groups of processors. Given an application with specific energy-performance objectives, our design methodology aims to customize HERA to match the diverse computation and communication characteristics of tasks in the application. Appropriate analysis of the application guides system synthesis that employs a parameterized hardware component library (PHCL) of function units with predictable performance and power dissipation. Extensive experimental results for parallel power-flow analysis are presented to demonstrate the effectiveness of our design methodology.*

Keywords: FPGA, Multiprocessor-on-a-Programmable Chip, design methodology, floating-point unit, mixed-mode parallel processing.

1 Introduction

FPGAs present new opportunities in exploring low-cost high-performance reconfigurable architectures for FP (floating-point) applications [1-6]; they also pose serious challenges to system design methodologies. Reconfigurable systems are most often application-specific programmable circuits (ASPCs) tailored to particular algorithms, thus replacing ASIC designs that have prohibitively high costs. “Cast-in-stone” ASPCs are not end-user programmable; their design follows ASIC-like methodologies that require extensive hardware expertise. The time-consuming hardware design and implementation procedure has to be repeated every time a change is made to the application algorithm. If the application requires a system size that exceeds the given FPGA capacity, the application has to be partitioned for resource time-sharing. Full-FPGA runtime reconfiguration (FRTR) may be required, which usually takes tens to hundreds of milliseconds. In an application shown for the Dynamic Instruction Set Computer (DISC),

the reconfiguration overhead represents more than 25% of the total execution time [7].

In contrast to their ever-increasing operating frequency, the reconfiguration overhead of SRAM FPGAs becomes more serious with increases in the chip area since the volume of configuration data is proportional to the number of on-chip resources. For example, the reconfiguration time of the device we use, the Xilinx XC2V6000-5, is at least 50 ms [8]. In comparison, we can multiply two 1000 x 1000 matrices in about 5 ms and perform LU factorization on a 1000 x 1000 matrix in about 50 ms on this FPGA [6]. Also, this device requires at least 2.97 W of power for reconfiguration [8], which results in 148.5 mJ of energy consumption. Energy issues are critical in embedded systems.

Our research targets FP data-parallel applications involving large input matrices. Intolerable reconfiguration overheads are normally required for these complex applications with the ASPC approach. Encouraged by continuous advances in FPGA technologies, we explore reconfigurable computing from a different perspective: user-programmable MPoPCs. MPoPCs are programmable-processor based systems embedded in reconfigurable logic. FRTR can be avoided by combining runtime software programming and partial reconfiguration. Software programmability is also essential in improving resource reusability, and reducing the system design and verification times. Related research efforts [5, 9] in this direction have focused on integrating multiple soft IP (Intellectual Property) configurable processor cores, such as Microblaze from Xilinx and Nios from Altera, on FPGAs. The configurability of these cores is limited and, hence, the benefits are often marginal.

Compared to chip multiprocessors implemented with fixed logic, our MPoPCs carry the advantage of customizing the architecture at both static and run times to better match the diverse computation and communication characteristics of tasks. Given the large MPoPC design space, a VHDL-based manual design should require extensive hardware expertise, resulting in error-prone and very time-consuming processes. Hence, an efficient system-level design methodology is needed to enable the average user to explore architectural choices involving performance-energy tradeoffs. Performance refers to execution time in this paper. This paper presents such a design methodology for HERA [6]. HERA is a reconfigurable MPoPC that we have designed and

implemented on Xilinx Virtex II FPGAs. HERA can be reprogrammed by instructions at runtime so that different groups of processors can work simultaneously in any of the SIMD, MIMD and Multiple-SIMD parallel computing modes. Mixed-mode parallel computing is an efficient approach to accommodate application idiosyncrasies. It was first employed in an image understanding system [10]; its benefits were also demonstrated with a more diverse system for matrix-based applications [6].

Given an FPGA with limited resources and an application with specific energy-performance objectives, our design methodology aims to customize an MPoPC. Since FP function units (FUs) are very resource-consuming in FPGAs, we employ different PE (Processing Element) configurations for different tasks based on their requirements for FP operations. This can also maximize the number of PEs for each task in the application for better performance. An in-house developed parameterized hardware component library (PHCL) is used during system synthesis. FRTR and/or Partial Runtime Reconfiguration (PRTR) are applied when needed. PRTR allows overlapping reconfiguration with computation. We employ FRTR with MPoPCs only when its benefits are larger than the associated overhead. For evaluation purposes, an in-house developed parallel power-flow analysis code for Newton’s method [11] is employed. Real-time power-flow analysis is of paramount importance for reliable operations in the Power Grid.

2 Architecture Template

To make the problem tractable, our synthesis procedure starts with a predesigned HERA template, as shown in Fig. 1. The architecture template is chosen based on our target applications, which are discussed in Section 3. PEs are interconnected via a 2-D mesh supported by a hierarchical bus system. We also implement fast, direct NEWS (North, East, West, and South) connections between nearest neighbors. The computing fabric is controlled by the Global Control Unit (GCU) that communicates with a host via the PCI bus for data I/O. The GCU fetches instructions from the global program memory (GPM) for PEs operating in SIMD. A PE of HERA is an in-house designed pipelined RISC processor that consists of an integer FU, one or more pipelined FP FUs (from PHCL), other custom function blocks (CFBs), and dual-port local data memory (LDM) and local program memory (LPM). All PEs are semi-customized and generated during architecture synthesis for the target application. A novel feature of HERA’s memory hierarchy is that one port of every PE’s LDM is shared with its neighbors to the south and east. This way, a PE can directly write to or read from the LDMs of its west and north neighbors via the shared port. This feature can be used to eliminate large block data transfers between nearest neighbors in omnipresent block-based matrix computations (e.g., LU factorization). Although PEs may contain different FP FUs, all of them support the same ISA to facilitate ease of software

development. HERA’s complete ISA contains about 30 instructions classified in six major groups: integer arithmetic, FP arithmetic, memory access, jump and branch, NEWS communications, and system control. Every PE includes a small amount of control logic, which interacts with the GCU and can realize both the MIMD and SIMD modes of execution. The operating mode of each PE can be configured dynamically by the GCU as needed through its *Operation Mode Register* that can be modified by the Configure instruction. As a result, HERA can be partitioned at run time into several PE clusters, each comprising a group of PEs running in SIMD or MIMD, resulting in a mixed-mode computing system capable of supporting simultaneously SIMD, MIMD, and multiple-SIMD.

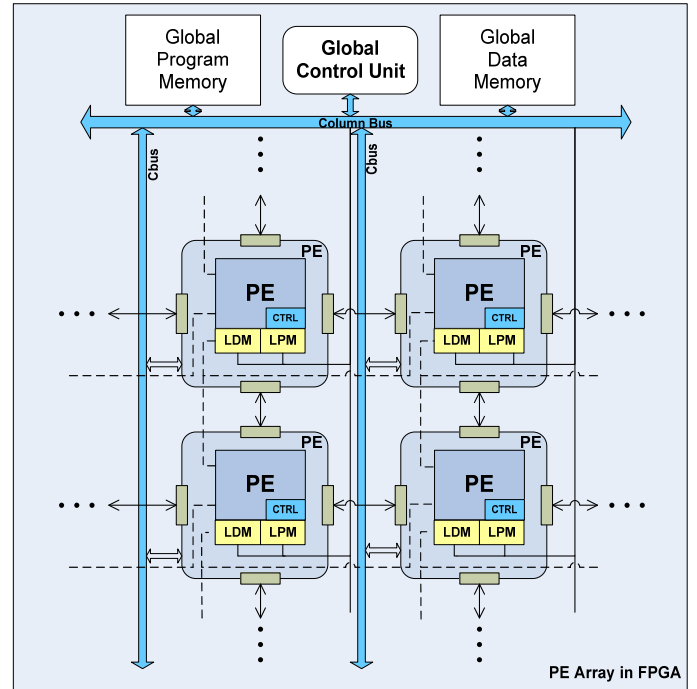


Fig. 1. Architecture synthesis template.

3 Application Model

A typical data-parallel FP application in engineering and science consists of many blocks of inter-dependent nested loops that consume substantial execution time; these loops are controlled by conditional statements. In our design methodology, the behavioral description of the application is first analyzed to construct a mixed-mode Task Flow Graph (*mTFG*). Fig. 2 shows a typical *mTFG*. Each node in this graph represents a task $S_i \in S$, where $i \in [1, s]$ is inclusive of all the tasks in the application. Associated with each task S_i are its computing mode (SIMD represented by a circle or MIMD represented by an octagon), used FP operation types (+, -, *, /, $\sqrt{\quad}$, ...), and the total number of each type of FP operations. An SIMD task in our study is typically a data-parallel block (e.g., a nested loop) which can benefit from SIMD execution, whereas an MIMD task is more of the control-flow style.

Both SIMD and MIMD tasks can be scheduled simultaneously on multiple PEs.

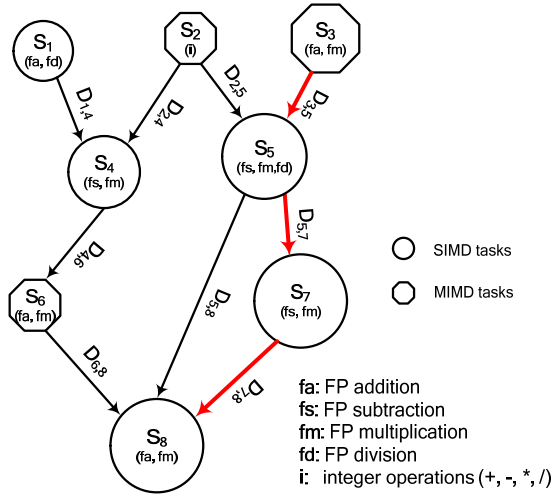


Fig. 2. A typical mixed-mode task flow graph.

4 Synthesis Methodology

A complex data-parallel application may contain diverse and time-varying tasks with different resource requirements in terms of FP FUs. Most often, not all FP operation types are needed all the time by all the tasks in an application. For example, FP division is less frequent than multiplication and addition in many data-parallel applications. FP FUs are very resource expensive and we have limited resources in FPGAs. Based on our implementation results, a single-precision IEEE-754 FP divider consumes about two or six times more resources than an FP adder or multiplier, respectively, of the same precision (shown in Fig. 3). These differences are even larger for double-precision FP FUs. By removing unneeded FP FUs in the PEs to run a task, we can potentially increase the number of PEs assigned to the task and thus improve performance. It is then beneficial to employ a dynamic HERA architecture throughout application realization, where the FP functionality of individual PEs and their number are modified as needed throughout the lifetime of the application. Also, different application-system pairs may have different performance and energy objectives. For example, a mobile embedded system may require maximizing the use of its battery even if the target application may have real-time requirements. A flexible design methodology is therefore preferred to take advantage of reconfigurable logic for a good application-system match under given performance-energy objectives. As these problems are NP-complete, good solutions at reasonable cost must be pursued.

4.1 Problem Definition

The starting point for our design exploration is a matrix-based data-parallel FP application represented by an m TFG and an FPGA chip that supports run-time reconfiguration. The latter has the following available

resource populations: F (logic resources expressed in logic cells), X (on-chip memory blocks), and Ψ (embedded DSP blocks). Logic cells are the basic building blocks consisting of one or more look-up tables (LUTs) and storage elements. The memory blocks are dedicated on-chip memory resources; e.g., BlockRAM or TriMatrix memory for Xilinx or Altera FPGAs, respectively. Our goal is to provide a system-level architecture exploration tool that can automatically select an efficient dynamic HERA configuration for a given application based on any of three performance-energy optimization objectives: (1) optimize performance without energy constraints; (2) optimize performance with energy constraints; and (3) reduce the energy cost for a given performance loss. The methodology facilitates an early design decision without actual VHDL-based time-consuming implementation. In order to speed up the synthesis procedure and provide predictable performance for the initial HERA machine configuration, we use the PHCL library in our methodology.

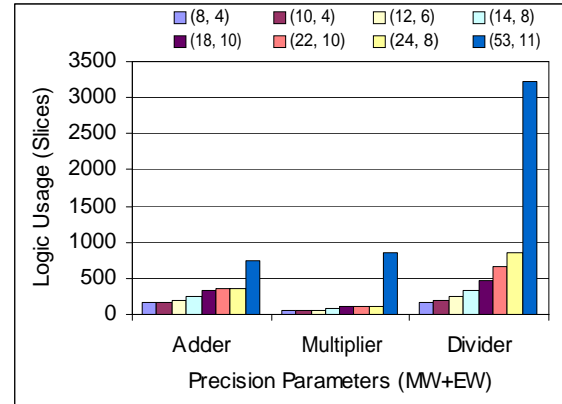


Fig. 3. XC2V6000 slices consumed by the FP FUs of various precisions as a function of (MW, EW). MW: mantissa width; EW: exponent width.

4.2 Parameterized Hardware Component Library

The speed and resource requirements of the FUs in the library represent many design choices. The components are designed in VHDL, and placed and routed for the target FPGA device. Based on our experiments with Virtex II FPGAs, we distinguish among four power states for an HERA FU: *active*, *idle*, *standby*, and *sleep*; each FU is an indivisible block in our design methodology. An FU consumes both static and dynamic power in the *active* and *idle* states, and only static power in the *standby* state. All consumptions are eliminated by shutting down the power supply to an FU, which puts it into the *sleep* state. An FU enters the *idle* state when no instruction accesses it and its consumption is due to clock activities. An FU is put into *standby* by disabling its clock signal. The major parameterized components for our matrix-based applications are variant-precision pipelined FP FUs (including IEEE-754 single- and double-precision implementations). The cores are parameterized by mantissa and exponent sizes. Different choices for the mantissa and exponent lead to different data ranges and resource

requirements. Fig. 3 shows the XC2V6000 slices used by FUs for various precisions. For each operation type (+, -, *, /, $\sqrt{\quad}$, ...) and a given precision, there are several choices in terms of latency, resource requirements, frequency, and power consumption. P^{active} , P^{idle} , and P^{stdby} represent the dynamic power consumption of the FU in the *active*, *idle*, and *standby* states, respectively, at the given frequency. The FU frequency can be set and changed at run-time.

4.3 Application-Driven System Synthesis

To decide on appropriate PE configurations, we must first derive equations for the estimation of the resource, performance, and energy consumption. Let $FU_{j,k}$ denote the k^{th} realization in PHCL of an FP FU capable of the operation type j , where $j \in \{+, -, *, /, \sqrt{\quad}, \dots\}$. The resource requirements and energy consumption per cycle E_i of a PE for a task in the critical path (TiCP) S_i are:

$$L_i = \sum_j \sum_k L_{j,k}^{FU} * \gamma_{i,j} * \gamma_{j,k} + \sum_m L_m^{CFB} * \gamma_{i,m} \quad (1)$$

$$D_i = \sum_j \sum_k D_{j,k}^{FU} * \gamma_{i,j} * \gamma_{j,k} + \sum_m D_m^{CFB} * \gamma_{i,m} \quad (2)$$

$$M_i = \sum_j \sum_k M_{j,k}^{FU} * \gamma_{i,j} * \gamma_{j,k} + \sum_m M_m^{CFB} * \gamma_{i,m} \quad (3)$$

$$E_i = \sum_j P_{j,k}^x * \frac{1}{F_i} * \gamma_{i,j} * \gamma_{j,k} + \sum_m P_{CFBm}^x * \frac{1}{F_i} * \gamma_{i,m} \quad (4)$$

$$\gamma_{i,j} = \begin{cases} 0 & \text{if the PE does not support the FP operation type } j \\ 1 & \text{if the PE supports the FP operation type } j \end{cases}$$

$$\gamma_{j,k} = \begin{cases} 0 & \text{if the PE does not include an } FU_{j,k} \\ 1 & \text{if the PE includes an } FU_{j,k} \end{cases}$$

$$\gamma_{i,m} = \begin{cases} 0 & \text{if the PE does not include a } CFB_m, \text{ i.e., the } m^{th} \text{ type of CFB} \\ 1 & \text{if the PE includes a } CFB_m \end{cases}$$

where $L_{j,k}^{FU}$, $D_{j,k}^{FU}$, and $M_{j,k}^{FU}$ are the usage of slices, DSP blocks, and on-chip memory blocks, respectively, by $FU_{j,k}$. $P_{j,k}^x$ and P_{CFBm}^x represent the power of $FU_{j,k}$ and the m^{th} type of CFB, respectively, in the x power state, where $x \in \{active, idle, stdby\}$. L_m^{CFB} , D_m^{CFB} , and M_m^{CFB} are the usage of slices, DSP blocks, and on-chip memory blocks, respectively, of the m^{th} type of CFB. F_i is the system frequency for task S_i . Up to one instance of an FU for each type of FP operations is included in each PE. Hence, $\sum_k \gamma_{i,k} = 0$ or 1. For example, PEs for S_8 in Fig. 2 include

FUs for addition and multiplication, but not for subtraction, division, and square-root realization.

Let p_i be the maximum number of PEs that could be assigned to task S_i . Let N_{conf} be the total number of reconfigurations during the entire execution of the application. If a required FP operation is not supported before scheduling a task, reconfiguration will be applied. For each PRTR, we count the percentage of reconfiguration bits over the total configuration bits for the target device, which is represented by C_{Conf} . Let B be the configuration

word width per cycle. For example, Xilinx Virtex FPGAs support both the parallel ($B = 8$ or 32) and serial ($B = 1$) configuration modes. Let F_c be the configuration frequency; it is usually lower than the system frequency F . The maximum F_c for Virtex II FPGAs is 50MHz (serial mode) or 66MHz (SelectMAP mode). The total execution time of the application is dominated by the TiCPs and can be approximated by:

$$T_\Sigma = \sum_{i=1}^c \frac{C(O_i, p_i)}{F_i} + \frac{N_{conf} * C_{conf}}{B * F_c} \quad (5)$$

where c is the total number of TiCPs. $C(O_i, p_i)$ is the minimum clock cycles needed for task S_i when p_i PEs are available; for simplicity, we will represent $O(S_i)$ for task S_i by O_i . It is possible that the optimal PE number for the minimum cycles is not p_i . Since the PE power varies with the state, the optimal number does not necessarily correspond to minimum energy consumption. Optimality involving both energy and performance depends on the task characteristics as well as the architecture. $C(O_i, p_i)$ is obtained with symbolic simulation for HERA before synthesis. An estimation table is created during this procedure for the execution time and energy consumption assuming various numbers of PEs.

To estimate the energy consumption E_Σ of the application, we sum up the average energy consumption of all the tasks and the total reconfiguration energy overhead:

$$E_\Sigma = \sum_{i=1}^s \left\{ \frac{1}{F_i} * \sum_j [O_j(S_i) * C_{j,k} * P_{j,k}^{active} * \gamma_{i,j} * \gamma_{j,k}] \right. \quad (6)$$

$$\left. + \sum_m [O_{CFBm}(S_i) * P_{CFBm}^{active} * \gamma_{i,m}] \right\} + T_\Sigma * (P_{sys} + P_{mem}) + \frac{N_{conf} * C_{conf}}{B * F_c} * P_{conf}$$

where s is the total number of tasks, $O_j(S_i)$ is the number of the j^{th} type operations in S_i , and $C_{j,k}$ is the latency (clock cycles) of $FU_{j,k}$. P_{conf} is the average configuration power for the entire chip. The average active power of $FU_{j,k}$ is used. P_{sys} and P_{mem} are the average power of the system template and BlockRAM memory, respectively. For the sake of simplicity, we are primarily interested in first-order factors here and neglect some runtime effects. However, this is sufficient to serve our purpose since we can refine the performance and energy estimates during the runtime allocation phase. The following three scenarios are considered to satisfy various performance-energy optimization objectives.

- *Case-1: Optimize performance without energy constraints.*

Assume capacities of $M_c(i)$ and $M_d(i)$ for the instruction and data memories, respectively, of each PE_i . Let the template requirements in slices and memory bits be L_{sys} and M_{sys} , respectively. Our objective is to minimize T_Σ (Eq. 5) subject to resource constraints imposed by the FPGA device:

$$L_{sys} + \sum_{i=1}^{p_i} L_i \leq \Phi \quad (7)$$

$$M_{\text{sys}} + \sum_{i=1}^p \{[M_c(i) + M_d(i)] + M_i\} \leq X \quad (8)$$

$$\sum_{i=1}^p D_i \leq \Psi \quad (9)$$

- *Case-2: Optimize performance under energy constraints.*

Let E_B be the upper bound on the energy. The objective is to minimize T_Σ subject to the following constraint, in addition to (7)-(9):

$$E_\Sigma \leq E_B \quad (10)$$

- *Case-3: Optimize the energy cost for a permissible performance loss.*

The base execution time T_B is as given in Case-1. Let β be the percentage of permissible performance loss. Our objective is to minimize the total energy cost E_Σ (Eq. 6) subject to the following constraint, in addition to Eqs. (7)-(9):

$$T_\Sigma \leq (1 + \beta)T_B \quad (11)$$

For any of these cases, the following procedure is applied to find an optimal configuration for each TiCP:

- Step 1.* Choose an appropriate FP precision for the given application.
- Step 2.* Select a system template in PHCL assuming the basic PE interconnect and interface to the sequencer. The PE datapath (functionality and width), total number of PEs, and PE layout are customizable.
- Step 3.* Select CFBs in PHCL.
- Step 4.* Identify the required FP operation types (i.e., +, -, *, /, $\sqrt{\quad}$, ...) in the application's tasks.
- Step 5.* Select appropriate FP FUs for the PEs assigned to each TiCP S_i based on its requirements. Formulate the objective equations based on the given performance-energy objective and the corresponding constraints. Apply an Integer Linear Programming (ILP) technique to obtain a near-optimal solution. If no solution is found for Case-2, then either increase the total energy budget or go back to *Step 1* by first lowering the arithmetic precision, if only allowed.

Since many parameters dwell in the relations, such as those associated with the FUs, the number of reconfigurations, and the system frequency, the exploration space is unlimited; and it is impossible to investigate all possible configurations in a short time. Hence, we have to focus on a near-optimal and practical solution with respect to: (1) limiting the number of reconfigurations, (2) using the fastest FUs, or (3) using the same frequency as a starting point for all the tasks.

5 Implementation Results

Newton's method is a robust solution to power-flow analysis, especially for large power networks consisting of thousands of buses. It solves iteratively the following linear

equations until the mismatches $\Delta\delta$ and ΔV are smaller than a pre-specified tolerance:

$$\begin{bmatrix} J^{11} & J^{12} \\ J^{21} & J^{22} \end{bmatrix} \begin{bmatrix} \Delta\delta \\ \Delta V \\ \frac{\Delta V}{|V|} \end{bmatrix} = \begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \quad (12)$$

The Jacobian matrix $J = \{J^{11}, J^{12}, J^{21}, J^{22}\}$ is reevaluated in each iteration. Eq. (12) is solved by LU factorization to produce the lower- and upper-triangular matrices L and U of the Jacobian [5]. The relevant tasks are shown in Table I.

The resources inside the XC2V6000-5 device on the FPGA board used in our experiments are: F (slices) = 33,792, X (on-chip memory blocks) = 144, and Ψ (embedded DSP blocks) = 144. Each memory block contains 18K bits and is configured as a 512 x 36-bit unit in HERA.

We first performed experiments with the first performance-energy objective (Case-1). The first set of experiments was to test the impact of FU precision on the execution time and energy consumption. The results for the 7917-bus power network are shown in Fig. 4. The picture clearly shows how important the chosen precision is.

Single-precision FP FUs were chosen for the following experiments based on the two benchmark networks in Table II. A fixed 125MHz system frequency was used in the experiments. From the task table, we can see that tasks $S_5^i(k_i)$, $S_7^i(k_i)$, $S_{10}^i(k_i)$, and $S_{11}^i(k_i)$ consume most of the execution time, especially for large matrices. The assigned number of PEs to these tasks has a large impact on performance. We compared the performance without energy considerations for three scenarios: *no device reconfiguration* (NR), FRTR, and optimal solution obtained from our synthesis methodology. We have not seen real implementations on ASPCs targeting similar matrix sizes. Table III shows the numbers of PEs for each task in the two benchmarks for various synthesis approaches. The execution time and energy consumption are shown in Table IV. Since our FPGA board and development environment do not support run-time partial reconfiguration, the results are based on full-scale cycle-to-cycle VHDL simulation with the benchmark data targeting the same device. We assume a device reconfiguration time of 50 ms [8].

The optimal solution of our synthesis depends on the matrix size. The 7917-bus benchmark requires a larger number of reconfigurations than the 1648-bus benchmark for the LU factorization and multiplication tasks because of more matrix blocks. With NR, the smallest number of PEs is used by all the tasks. However, we must consider the cost of these schemes. Although we have a small number of PEs in NR, there is no reconfiguration overhead. The overhead becomes significant with FRTR. For this reason, the performance and energy consumption of FRTR for the 1648-bus benchmark is worse than for the other cases. FRTR cannot overlap tasks, and substantial time is required to save and restore data. Also, some resources are wasted waiting for reconfiguration. With the optimal configurations, both FRTR and PRTR are employed only

when their benefits exceed the penalties; partial reconfiguration overlaps computations as much as possible. For the 7917-bus benchmark, the reconfiguration overhead becomes much less significant as compared to the computation time. Hence, FRTR shows a better performance than NR. In all the cases, the optimal configuration used 19.5% and 15.8% less time, and 19.2% and 14.6% less energy for the 1648- and 7917-bus benchmarks, respectively, as compared to NR. These numbers clearly show the benefits of MPoPCs compared to fixed chip multiprocessors (i.e., NR). With matrix size increases the amount of computations is boosted and the benefits of resource customization and reconfiguration become more pertinent.

Finally, Table V shows performance-energy trade-offs through architecture exploration. In Scenario-II, we reduce the total energy consumption by about 10% without major performance penalty in both benchmarks. A performance penalty of 9.9% and 7.64% for the 1648- and 7917-bus benchmarks, respectively, is observed when reducing the energy consumption by 20% (shown in Scenario-III). In Scenario-IV and -V, we relax the performance by about 15% and 20%, respectively. This reduces the energy

consumption by 25.4% and 32.3%, respectively, for the 1648-bus case. The savings are 26.2% and 40.8%, respectively, for the 7917-bus case. Generally, we achieved better performance for the 7919-bus case because of more matrix blocks that can be manipulated in performance-energy tradeoffs.

6 CONCLUSIONS

HERA combines the flexibility of reconfigurable logic and the benefits of parallel processing for large-scale floating-point applications. It delivers high performance at very low cost. This semi-customized MPoPC for data-parallel applications offers general-purpose instructions that reduce drastically the need for hardware reprogramming during task execution. MPoPCs present us with a very large design space. Our proposed system synthesis and resource management methodology can better match application idiosyncrasies and satisfy various energy-performance objectives. Our design tools hide the detailed hardware implementation from the average user but still result in predictable performance. Benchmarking demonstrates the effectiveness of our approach.

TABLE I. TASK INFORMATION IN PARALLEL POWER-FLOW ANALYSIS

Tasks	Description	Mode	FP Op.
$S_1(1), \dots, S_1(n)$	Evaluate Eqs (14) and (15) and calculate ΔP and ΔQ based on Eqs. (14) and (15).	SIMD	+, -, * cos, sin
$S_2(1), \dots, S_2(n)$	Check individual convergence	SIMD	None
S_3	Check global convergence	MIMD	None
$S_4(1), \dots, S_4(n)$	Construct 3-block groups $\{J_{ii}, J_{in}, \text{ and } J_{ni}\}$	SIMD	+, -, * cos, sin
$S_5^i(1), \dots, S_5^i(k_i)$, $i = 1, \dots, m$	LU factorization of 3-block groups of approximately similar sizes	SIMD	+, -, *, /
$S_6(1), \dots, S_6(n)$	LU factorization of 3-block groups of diverse sizes	MIMD	+, -, *, /
$S_7^i(1), \dots, S_7^i(k_i)$, $i = 1, \dots, m$	Multiplication of factored border blocks in S_5	SIMD	+, *
$S_8(1), \dots, S_8(n)$	Multiplication of factored border blocks in S_6	MIMD	+, *
S_9	Factor the last block J_{nn}	SIMD	+, -, *, /
$S_{10}(1), \dots, S_{10}(n)$	Forward reduction	SIMD	+, -, *
$S_{11}(1), \dots, S_{11}(n)$	Backward substitutions	MIMD	+, -, *

n : number of 3-block groups in matrix J .

m : number of task pools with approximately the same computing cost; group i contains k_i 3-block groups.

q : number of 3-block groups of diverse computing cost. $n = q + \sum_{i=k_1}^{k_m} m$

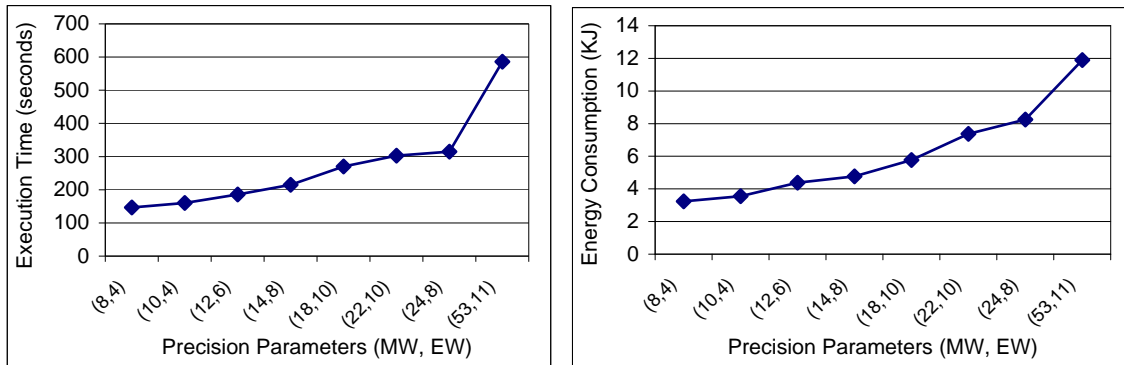


Fig. 4. Execution time and energy consumption of various-precision HERA systems for the 7917-bus benchmark (MW: mantissa width; EW: exponent width).

TABLE II. OPTIMAL PARTITIONING OF Y_{bus} MATRICES FOR TWO BENCHMARK SYSTEMS

Benchmark System	1648-bus	7917-bus
Dimensionality of admittance matrix (Y_{bus})	1648	7917
Dimensionality of Jacobian matrix (J)	2982	14508
Number of independent diagonal blocks	18	67
Minimum dimensionality of independent diagonal blocks	33	15
Maximum dimensionality of independent diagonal blocks	120	150
Dimensionality of the last block	134	541
Size distribution of independent diagonal blocks in Y_{bus} matrix	120, 109, 99, 3(90) , 5(85)*, 79, 5(75), 33	7(150),17(130), 10(120), 13(100), 19(90), 1(15)

*5(85) stands for 5 blocks of approximate size 85 x 85.

TABLE III. THE PARALLELISM PROFILE (NUMBERS OF PEs) DURING EXECUTION

Task	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
NR	24	24	24	24	24	24	24	24	24	24	24
FRTR	48	64	64	48	24	24	48	48	24	48	48
Optimal (1648)	48	48	48	24	24	24	48	48	32	32	32
Optimal (7917)	48	48	48	48	24	24	48	48	39	48	48

NR: no reconfiguration is allowed.

FRTR: Full Run-Time Reconfiguration is used to get the maximum number of PEs.

Optimal (1648): Optimal solution for the 1648-bus system with our synthesis procedure.

Optimal (7917): Optimal solution for the 7917-bus system with our synthesis procedure.

TABLE IV. EXECUTION TIME AND ENERGY CONSUMPTION FOR THE BENCHMARK MATRICES

Case	NR	FRTR	Optimal	
1648-bus	Time (s)	12.0	12.3	10.1
	Energy (J)	271.3	288.4	231.5
7917-bus	Time (s)	391.5	369.3	315.1
	Energy (J)	10214.3	9630.1	8252.0

TABLE V. PERFORMANCE-ENERGY OPTIMIZATION FOR THE TWO BENCHMARKS

Scenario	Objective	Benchmark	Constraints	Energy (J)	Execution Time (s)
I	Minimize T	1648-bus	None	231.5	10.1
		7917-bus	None	8249	314
II	Minimize T	1648-bus	$E < 208.4$	208.1	10.3
		7917-bus	$E < 7424$	7419	316
III	Minimize T	1648-bus	$E < 185.2$	185.0	11.1
		7917-bus	$E < 6600$	6598	338
IV	Minimize E	1648-bus	$T < 11.62$	172.7	11.6
		7917-bus	$T < 361$	6087	360
V	Minimize E	1648-bus	$T < 12.12$	156.7	12.1
		7917-bus	$T < 377$	4883	375

7 References

- [1] D. Bouldin, "Enabling killer applications of reconfigurable systems," *Engineering Reconfig. Systems Algorithms (ERSA)*, June 2005.
- [2] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proc. Computers Digital Techniques*, vol. 152, no. 2, pp. 193-207, March 2005.
- [3] K. Underwood, "FPGAs vs. CPUs: trends in peak floating-point performance," *12th ACM/SIGDA Intern. Symp. Field-Program. Gate Arrays*, pp.171-180, 2004.
- [4] G. Govindu, L. Zhuo, S. Choi, and V. K. Prasanna, "Analysis of high-performance floating-point arithmetic on FPGAs," *11th Reconfigurable Architecture Workshop (joint with IPDPS 2004)*, April 2004.
- [5] X. Wang and S. G. Ziavras, "Parallel LU factorization of sparse matrices on FPGA-based configurable computing engines," *Concurrency Computation: Pract. Exper.*, vol. 16, no. 4, pp. 319-343, 2004.
- [6] X. Wang and S. G. Ziavras, "Exploiting mixed-mode parallelism for matrix operations on the HERA architecture through reconfiguration," *IEE Proc. Computers Digital Techniques*, Vol. 153, No. 4, July 2006, pp. 249-260.
- [7] M. J. Wirthlin and B. L. Hutchings, "Sequencing run-time reconfigured hardware with software," *ACM/SIGDA Intern. Symp. Field-Program. Gate Arrays*, pp.122-128, 1996.
- [8] Virtex II FPGA datasheet, <http://direct.xilinx.com/bvdocs/publications/ds031.pdf>.
- [9] Y. Jin, N. Satish, K. Ravindran, and K. Keutzer, "An automated exploration framework for FPGA-based soft multiprocessor systems," *IEEE Intern. Conf. Hardware/Software Codesign System Synthesis*, Sept. 2005.
- [10] H. J. Siegel, T. D. Braun, H. G. Dietz, M. B. Kulaczewski, M. Maheswaran, P. H. Pero, J. M. Siegel, J. J. E. So, M. Tan, M. D. Theys, and L. Wang, "The PASM project: a study of reconfigurable parallel computing," *Intern. Symp. Parallel Architectures, Algorithms, and Networks*, pp. 529-536, 1996.
- [11] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Trans. Power Apparatus Systems*, vol. PAS-86, no. 3, pp. 1146-1152, 1967.