Modeling and Simulation Speed-Up of Plasma Actuators Implementing Reconfigurable Hardware

Abbas Ebrahimi and Mohammad Zandsalimy
Sharif University of Technology, Tehran 11155-1639, Iran

The objective of the present study is to investigate the capability of field-programmable gate array hardware in numerical simulation of a model of a dielectric barrier discharge plasma actuator to accelerate the calculations. The reconfigurable hardware is designed such that it is possible to reprogram its architecture after manufacturing. This provides the capability to design and implement various architectures for several applications. Two reconfigurable chips are used in the present study, one of which consists of a programmable logic unit and a typical microprocessor. This hybrid architecture makes the high performance of the reconfigurable hardware in custom computing and the efficiency of the microprocessor in data flow control accessible. An automated design procedure is used for the design of the reconfigurable hardware. Further, a finite difference representation of a phenomenological model of a plasma actuator is derived and implemented on the field-programmable gate array hardware. The results are validated against other numerical data, and the computational time is compared to different conventional processors. Using the reconfigurable hardware results in up to 96% computational time reduction compared to a recent Core i7 processor.

I. Introduction

ONE of the most challenging problems in science and engineering is the task of obtaining accurate low-latency solutions to the governing equations of fluid mechanics [1]. Computational fluid dynamics (CFD) has been known as a technique with massive floating-point operations in most of these problems, requiring a fine high-quality computational mesh as well as a sufficient number of iterations to yield an accurate solution. As a result, even by employing extremely advanced modern digital computers, the solution time of these problems will still be significantly high [2]. Although nowadays CFD is inexpensive relative to physical experiments, there is a great demand for solution speed-up.

Modeling and simulation of dielectric barrier discharge (DBD) plasma actuators using proper computational methodologies are an example of such numerical calculations with high solution cost. DBD plasma actuators are devices known for generating controllable wall-bounded jet flow [3–5]. Having an accurate and efficient computational model for simulating the actuator effect on the flowfield is crucial. There are a number of such models developed for simulating the underlying physics of ionization process and calculating the actuator performance [6–8]. These computational models include simplified models and first-principles-based models. Simplified models are less complex representations of the plasma physics and are often based on a phenomenological approach [9]. The first-principles-based models are derived directly from the level of established laws of physics and do not contain any assumptions such as empirical models [10].

Several methods have been suggested and implemented for reducing the computational time of PDEs that govern the fluid dynamics problems (software and hardware methods). Optimization of computer programs and the use of new and more advanced numerical algorithms are used as software-centered methods of computational time reduction. On the other side, the use of more powerful processors or employing high-performance computing systems are examples of hardware methods [11,12].

There has been a dramatic increase in employing new and more powerful hardware for computational purposes in the past two decades. One of the most studied hardware in this field is the graphics processing unit (GPU). Phillips et al. [13] employed GPU clusters for compressible flow computations using a finite-volume method. Further, Ma et al. [14] implemented a GPU for the solution of compressible flow problems using a meshless method and reached over an order of magnitude speed-up. Stone and Davis [15] investigated the performance and accuracy of solving stiff chemical kinetics equations on a GPU and achieved over 20x speed-up compared to traditional CPUs. The solution of the two-dimensional (2-D) Boltzmann transport equation using the CUDA language and an NVidia GPU was carried out by Priimak [16]. Xu et al. [17] implemented high-order compact finite difference schemes for complex grids on GPUs and attained 60% more efficiency in computations. Thomas et al. [18] used a hybrid combination of CPU and GPU and implemented a free-vortex method together with an unsteady Reynolds-averaged Navier–Stokes solver for the simulation of two-phase flow beneath a hovering rotor. In another work, Liu et al. [19] proposed a method for the solution of CFD problems on CPU + GPU platforms and evaluated their method with the lid-driven cavity flow. Chan et al. [20] presented the algorithm for efficient implementation of a high-order discontinuous Galerkin method for the wave equation on a GPU. Moreover, Remacle et al. [21] executed a spectral finite element scheme for the solution of elliptic problems on unstructured grids on a GPU and demonstrated solution speed-up. Tredak et al. [22] implemented a molecular dynamics algorithm using reactive bond order empirical potential on a single GPU and reached over 12 times the calculation speed of a CPU. Most recently, Liang et al. [23] implemented the canonical Monte Carlo simulation of Coulomb many-body systems on a GPU and reached 440 times faster calculations. Despite numerous studies on GPU acceleration of numerical computations, there still exists an increasing interest for more powerful computational systems in the scientific community.

Studies performed within the last decade demonstrate the promising future of using field-programmable gate array (FPGA) hardware as an alternative to conventional processors. An FPGA is an integrated circuit which contains an array of logic blocks. The architecture and internal wiring of this hardware can be reconfigured many times [24]. Complex circuits for various applications can be designed and implemented using this electronic hardware. As a result, computers with an FPGA-based processing unit are in the spotlight for numerical simulations and CFD problems. This is due to the high efficiency of FPGA-based processors in parallelizing at the hardware level for

There are several studies in the literature reporting on different digital applications [26–29] and computational purposes of FPGAs [30–43]. By the significant increment of clock frequency as well as logic block density, FPGAs can now be implemented as highly flexible standalone computational processors [39,44]. In the following, some of the most recent investigations on the latter application of FPGAs are reviewed.

Andrés et al. [37] presented a brief study on the feasibility of using FPGAs to accelerate CFD simulations. Sanchez-Roman et al. [38] exhibited an FPGA-based accelerator to implement a cell-vertex finite volume algorithm for solving the Euler equations. Further, Sano et al. [39] evaluated the performance of an FPGA-based computing system including multiple FPGAs implementing iterative linear equation solvers and conducted a 2-D Laplace problem as a benchmark on this platform. Later, they introduced a performance model of a tightly coupled FPGA cluster to accelerate a lattice Boltzmann method solver [45]. Liu et al. [40] published a framework based on reconfigurable logic to implement a one-dimensional (1-D) CFD model of a diesel fuel system.

Gan et al. [46] proposed a hybrid computing system (including both FPGA and CPU) that made use of single and multiple FPGAs to compute an upwind scheme for the Saint-Venant equations. Using mixed precision calculations, they were able to build a fully pipelined circuit using a high-level hardware design procedure. Also, they reported a 100 times computational speed-up over a six-core CPU. Dohi et al. [47] evaluated three-dimensional stencil computing on a stream-based FPGA accelerator and implemented a heat conduction problem as a benchmark to test the results. They achieved a computational speed-up of six times faster than a multithreaded conventional CPU.

Zhang et al. [48] indicated that existing approaches for FPGA architecture design are inefficient due to underutilization of memory bandwidth and logic resource on the reconfigurable hardware. To conquer this problem, they proposed an analytical design procedure using the roofline model [49] and achieved a peak performance of 61.62 GFLOPS while performing at 100 MHz. Recently, Nagasu et al. [50] proposed a high-performance computing system for simulating tsunami with an FPGA hardware, using a highly pipelined architecture performing in a limited bandwidth. Also, in a recent work by the authors of the present paper, the feasibility of the FPGA hardware for accelerating the numerical solution of Laplace problem as well as the 1-D Euler equation is studied, and a 20x speed-up is achieved [51].

The main objective of this paper is the numerical modeling of the plasma actuator and computational speed-up, using a reconfigurable hardware with a high-level description language for the design process. To the best of the authors' knowledge, no report has been found thus far modeling and simulating the plasma actuator using this method. In the present research, the hardware structure and the configuration methods of an FPGA are presented. Then, the feasibility of this hardware for numerical computation applications is studied. Moreover, the equations of a phenomenological model of the DBD plasma actuator are solved using this hardware. The most important factors in performance increment of numerical computations are memory bandwidth and low latency of arithmetic operations. Hence, various directives are used to reach the best possible performing architecture and lowest computational latency. Moreover, the numerical results of calculations are validated, and the solution times of two FPGA chips are compared to the results of four different conventional CPUs.

II. Reconfigurable Computing

There are two general hardware configurations to perform digital computations. First is to use hardware that is designed for a specific purpose and cannot be reconfigured after manufacturing such as application-specific integrated circuits (ASICs). Every single ASIC is designed to perform a specific and predetermined task; as a result, it is very fast and reliable at executing its own assignment. The second method involves using a software-programmed microprocessor, which is a far more flexible method than the previous one. Hardware application in this method is controlled by a software program. Changing the software results in an application alteration. The most obvious drawback of this flexibility is the lower performance of hardware in computations than the first method.

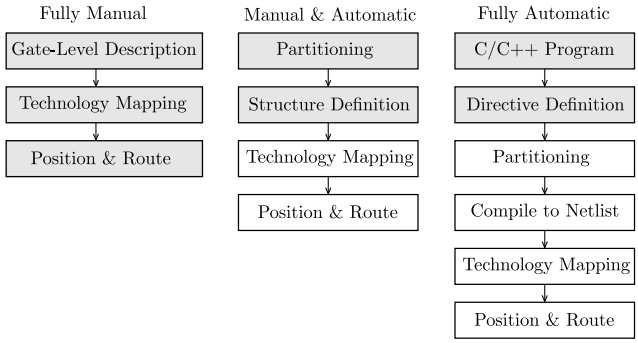

Fig. 1 Different possible design flows for the configuration of FPGAs (user has to perform the shaded levels).

The main objective of reconfigurable computing is to bridge the gap between software and hardware, to be able to get the flexibility of software as well as the performance of hardware at the same time [52]. Reconfigurable hardware (such as FPGA) contains an array of computational elements whose applications can be altered and configured through a number of configuration bits. These elements, which are sometimes called logic blocks, are connected together with some reconfigurable connection wires and switches. As a result, it is possible to construct various custom digital circuits by changing each logic block's application and the connection baseline. As mentioned in Sec. I, the performances of some applications have been highly elevated using reconfigurable computing.

Hardware configuration tools can exist in the form of manual languages for the design process in which the user has to manually adjust and introduce every single element and routing to the system. It can also exist in the form of automatic software that interprets and compiles a high-level representation of the design architecture to the hardware-level language automatically. The latter is known as high-level languages for hardware design. Using an automated design procedure requires less effort from the user, but manually adjusting the framework can result in a more optimized architecture for the application [53]. Accordingly, different possible design flows for the configuration of FPGAs are depicted in Fig. 1. Herein, an automatic design flow is used for the FPGA configuration with manual directives for a more optimized design.

A. Employed Hardware

In the present study, two different FPGA chips are employed for reconfigurable computing and solution speed-up. Zynq-7020 from the Zynq-7000 lineup [54] by Xilinx Co. is the first FPGA hardware that contains both reprogrammable logic (FPGA hardware) and processing system (microprocessor hardware) on the same chip. The Zynq-7000 family products integrate a feature-rich dual-core or single-core ARM Cortex-A9-based processing system and 28 nm Xilinx programmable logic in a single device. The Zynq-7020 chip is optimized for massive computations with low power consumption. This FPGA is implemented on a board called "z-turn board" manufactured by MYiR Co. The next FPGA is XC7VX690T from the Virtex-7 lineup [55] by Xilinx Co. This chip only contains reprogrammable logic and can be coupled with an external CPU through high-speed PCIe connections for a hybrid setup. The VC709 evaluation board provides a hardware environment for developing and evaluating designs targeting this chip. This board provides features common to many embedded processing systems, including a dual DDR3 memory module, an eight-lane PCIe interface, general purpose I/O, and a UART interface. The latter FPGA is much more powerful than the former, benefiting from 8x the logic cells and 9x the DSP slices. Figures 2 and 3 show the top view of these boards indicating some of the components.

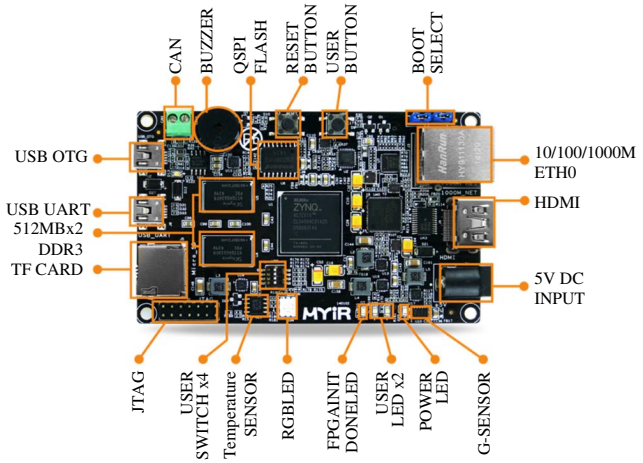

Fig. 2 z-turn board by MYiR Co.

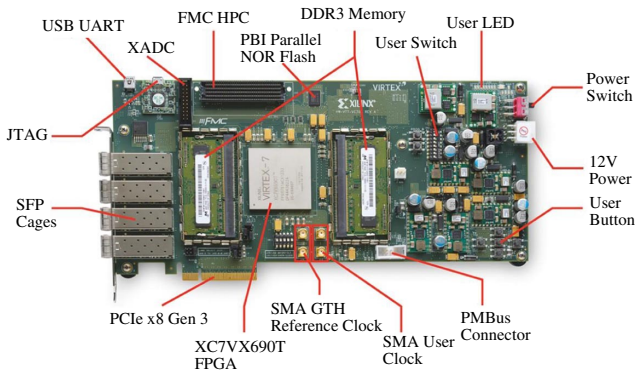

Fig. 3 VC709 board by Xilinx Co.

Four different CPUs by Intel, an old-generation Core i7-740QM [56], a Core i7-3770 K [57], a Core i7-4790 K [58], and a new-generation Core i7-6950X [59], are used to compare the solution times of various tests with the results obtained from the FPGAs. These chips have multiple processing cores; thus, for the sake of comparison, single- and multicore (without vectorization) as well as fully vectorized performances are reported and compared to the results of the reconfigurable hardware. The Core i7-740QM is a mobile processor with four physical processing cores running at 1.73 GHz and has the lowest computational power of the four, with only 23.44 GFLOPS. The Core i7-6950X is the most powerful CPU here, with 10 processing cores, each running at 3.00 GHz generating 70 GFLOPS of computational power. Further, the single-core performance of each chip is much lower than multicore performance. These CPUs run at their maximum turbo frequency, which means a higher performance per core than what is expected. The i7-740QM runs at about 2.93 GHz, the i7-3770 K at 3.9 GHz, the i7-4790 K at 4.4 GHz, and the i7-6950X at 3.5 GHz with maximum turbo frequency. Further, the latest chip has a high amount of cache (25 MB to be exact), which makes it perfect for numerical calculation purposes, not to mention its 20 processing threads. Table 1 presents a comparison of maximum frequency and processing speed between the FPGAs and CPUs used in the present study.

Table 1 Specifications of the processors in use

Model	Maximum frequency, GHz	Processing speed, GFLOPS
Core i7-740QM CPU	2.93	23.44
Core i7-3770 K CPU	3.90	31.20
Core i7-4790 K CPU	4.40	35.20
Core i7-6950X CPU	3.50	70.00
Zynq-7020 FPGA	0.25	Architecture-dependent
XC7VX690T FPGA	0.81	Architecture-dependent

B. Configuration Method

An automated design process recommended by Xilinx Co. is used for reconfigurable hardware design in the present study. Three different software modules are used in this method, all of them optimized for the task, including Vivado, Vivado HLS, and Xilinx SDK. Using Vivado HLS and starting with a code in C++, some intellectual property (IP) blocks are produced and then packaged. An IP block is a logical hardware description layout that includes a number of inputs and outputs. These IP blocks are then transferred to Vivado and connected together to form a fully functional electronic circuit. An example of hardware design in Vivado for solving the Laplace equation on the Zynq-7020 FPGA and the used IPs are demonstrated in Fig. 4. Each one of the blocks has a special role and function in the main architecture.

Hls_accel block is the solver of the Laplace equation that has been developed via Vivado HLS and from a program written in C++. This IP receives the initial conditions for the Laplace equation and then, after completing the solution, sends out the results through the output port. The AXI Timer block is an IP provided by Xilinx Co. to calculate the exact clock rate of the chip during calculations of a program. The main purpose of this block is to calculate the correct solution time for each problem. Also, the ZYNQ7 Processing System block can initiate and control the ARM processors available in the Zynq-7020 chip. By means of this IP, it is possible to communicate with the Hls_accel block through the high-bandwidth AXI Interconnect connection block and control the input/output dataflow of the reconfigurable unit. A full discussion about IP blocks can be found in [60]. The final circuit design is then applied to the actual hardware and debugged using Xilinx SDK.

III. Governing Equations for the Plasma Actuator

The main purpose of this study is to reduce the computation time required for solving governing equations of the DBD plasma actuator (a phenomenological model), using different types of reconfigurable hardware. DBD plasma actuator is composed of two electrodes separated by a dielectric material, usually glass, quartz, Kapton, or ceramics. One of the electrodes is exposed to airflow connected to a high-voltage power supply. The other electrode is enclosed inside the dielectric layer and often grounded. A schematic of this actuator is presented in Fig. 5. The DBD plasma actuator is often driven by a high sinusoidal voltage 1–20 kV, 0.05–20 kHz. When the driving voltage reaches a breakdown value, the actuator causes a weak ionization of the air molecules. Charged particles in the vicinity of the exposed electrode propagate along the dielectric surface (due to the electric potential field) and transfer momentum to the surrounding neutral particles of the air. The induced airflow will move from the exposed electrode in the direction of the enclosed electrode [10].

However, a number of models to directly simulate the ionization process do exist, and a numerical model of the plasma actuator can effectively provide the data for a simplified model of the problem. According to [61], the S-H model [62] has acceptable accuracy, despite being a simplified model of the physics in question. The S-H model allows for a control over the charge density distribution over the enclosed electrode. In the following, the governing equations of the original S-H model are presented.

Because the gas particles are weakly ionized in the plasma generation process, Suzen et al. [62] assumed that the electric potential is the superposition of the potential due to the external electric field and the potential due to the net charge density. They also assumed that the Debye length and the charge density above the enclosed electrode are small. With further assumptions, Suzen and Huang simplified Maxwell equations and expressed this problem with two independent PDEs for the distributions of the electric potential and the net charge density inside the plasma region as follows:

$$\nabla \cdot (\epsilon_r \nabla \phi) = 0 \tag{1}$$

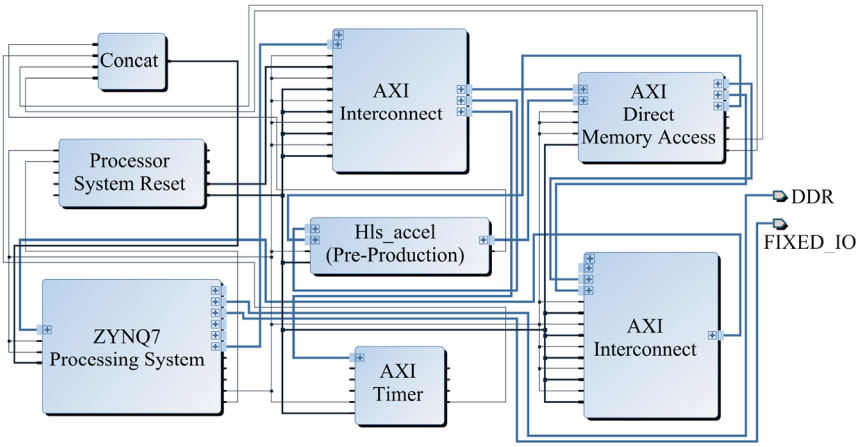

Fig. 4 Block diagram of hardware designed to solve the Laplace equation on Zynq-7020.

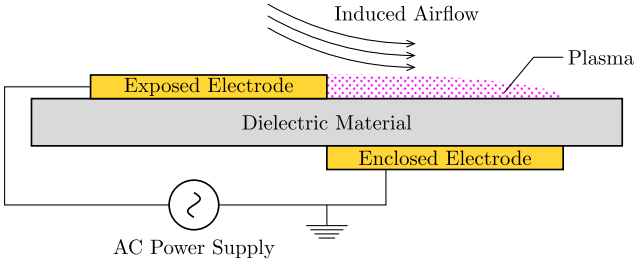

Fig. 5 Schematic of a DBD plasma actuator.

$$\nabla \cdot (\epsilon_r \nabla \rho_c) = \frac{\rho_c}{\lambda_d^2} \tag{2}$$

where ϕ is the electric potential, λ_d is the Debye length, ρ_c is the net charge density, and ϵ_r is the relative permittivity of the medium.

A. Electric Potential Equation Discretization

A Cartesian nonuniform computational grid (Fig. 6) has been implemented for the numerical solution of Eq. (1). This equation can be expanded as

$$\left(\frac{\partial}{\partial x}\hat{i} + \frac{\partial}{\partial y}\hat{j}\right) \cdot \left(\epsilon_r \left(\frac{\partial \phi}{\partial x}\hat{i} + \frac{\partial \phi}{\partial y}\hat{j}\right)\right) = 0 \tag{3}$$

The nonuniform grid (x, y) is conformally mapped to a uniform grid (ζ, η) using relations (4) and (5):

$$\frac{\partial}{\partial x} = \frac{\partial \zeta}{\partial x}\frac{\partial}{\partial \zeta} + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta} \tag{4}$$

$$\frac{\partial}{\partial y} = \frac{\partial \zeta}{\partial y}\frac{\partial}{\partial \zeta} + \frac{\partial \eta}{\partial y}\frac{\partial}{\partial \eta} \tag{5}$$

According to Fig. 6, considering $\partial\eta/\partial x = 0$ and $\partial\zeta/\partial y = 0$ and using Eqs. (4) and (5), Eq. (3) can be rewritten as

$$\begin{aligned}\frac{\partial \zeta}{\partial x}\frac{\partial \epsilon_r}{\partial \zeta}\frac{\partial \zeta}{\partial x}\frac{\partial \phi}{\partial \zeta} + \frac{\partial \zeta}{\partial x}\epsilon_r \frac{\partial^2 \zeta}{\partial \zeta \partial x}\frac{\partial \phi}{\partial \zeta} + \left(\frac{\partial \zeta}{\partial x}\right)^2 \epsilon_r \frac{\partial^2 \phi}{\partial \zeta^2} \\ + \frac{\partial \eta}{\partial y}\frac{\partial \epsilon_r}{\partial \eta}\frac{\partial \eta}{\partial y}\frac{\partial \phi}{\partial \eta} + \frac{\partial \eta}{\partial y}\epsilon_r \frac{\partial^2 \eta}{\partial \eta \partial y}\frac{\partial \phi}{\partial \eta} + \left(\frac{\partial \eta}{\partial y}\right)^2 \epsilon_r \frac{\partial^2 \phi}{\partial \eta^2} = 0\end{aligned} \tag{6}$$

Further simplification of Eq. (6) results in

$$A_\zeta \frac{\partial \phi}{\partial \zeta} + A_\eta \frac{\partial \phi}{\partial \eta} + \epsilon_r \left(\left(\frac{\partial \zeta}{\partial x}\right)^2 \frac{\partial^2 \phi}{\partial \zeta^2} + \left(\frac{\partial \eta}{\partial y}\right)^2 \frac{\partial^2 \phi}{\partial \eta^2}\right) = 0 \tag{7}$$

in which

$$A_\zeta = \left(\frac{\partial \zeta}{\partial x}\right)^2 \frac{\partial \epsilon_r}{\partial \zeta} + \frac{\partial \zeta}{\partial x}\epsilon_r \frac{\partial^2 \zeta}{\partial \zeta \partial x}$$

and

$$A_\eta = \left(\frac{\partial \eta}{\partial y}\right)^2 \frac{\partial \epsilon_r}{\partial \eta} + \frac{\partial \eta}{\partial y}\epsilon_r \frac{\partial^2 \eta}{\partial \eta \partial y}$$

An explicit scheme with first-order-accurate finite difference discretization is used for the solution of Eq. (7); thus,

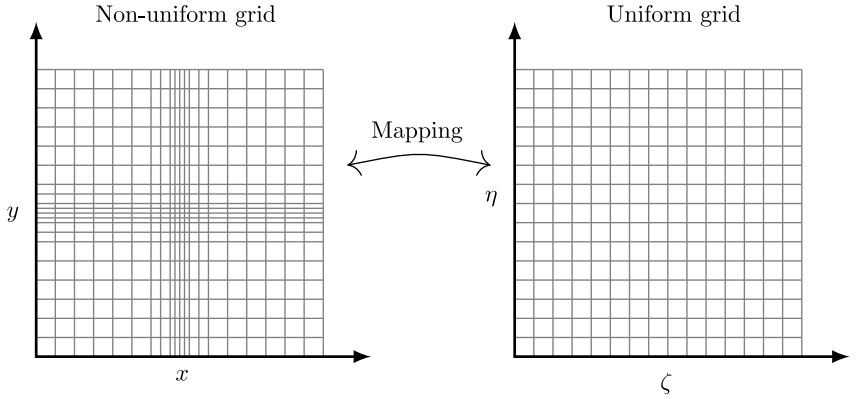

Fig. 6 Conformal mapping used for the numerical calculation of the plasma model.

$$A_\zeta \frac{\phi^k_{i+1,j} - \phi^{k+1}_{i,j}}{\Delta\zeta} + A_\eta \frac{\phi^k_{i,j+1} - \phi^{k+1}_{i,j}}{\Delta\eta} + \epsilon_r \left(\left(\frac{\partial\zeta}{\partial x}\right)^2 \frac{\phi^k_{i+1,j} - 2\phi^{k+1}_{i,j} + \phi^k_{i-1,j}}{\Delta\zeta^2} + \left(\frac{\partial\eta}{\partial y}\right)^2 \frac{\phi^k_{i,j+1} - 2\phi^{k+1}_{i,j} + \phi^k_{i,j-1}}{\Delta\eta^2} \right) = 0 \quad (8)$$

Rewriting Eq. (8) yields

$$B\phi^{k+1}_{i,j} = \frac{A_\zeta}{\Delta\zeta}\phi^k_{i+1,j} + \frac{A_\eta}{\Delta\eta}\phi^k_{i,j+1} + \epsilon_r \left(\left(\frac{\partial\zeta}{\partial x}\right)^2 \frac{\phi^k_{i+1,j} + \phi^k_{i-1,j}}{\Delta\zeta^2} + \left(\frac{\partial\eta}{\partial y}\right)^2 \frac{\phi^k_{i,j+1} + \phi^k_{i,j-1}}{\Delta\eta^2} \right) \quad (9)$$

in which

$$B = \frac{A_\zeta}{\Delta\zeta} + \frac{A_\eta}{\Delta\eta} + 2\epsilon_r \left(\left(\frac{\partial\zeta}{\partial x}\right)^2 \frac{1}{\Delta\zeta^2} + \left(\frac{\partial\eta}{\partial y}\right)^2 \frac{1}{\Delta\eta^2} \right)$$

Derivatives inside A_ζ, A_η, and B are discretized using the same first-order-accurate finite difference method. These parameters are always constant on every node of the computational grid. Thus, it is acceptable to calculate them at the beginning of the numerical solution and then continue the rest.

Numerical solution of Eq. (9) using the boundary conditions of Fig. 7 yields the electric potential ϕ distribution in the domain. Kapton is used as the dielectric material which its electrical permittivity relative to air is considered to be 2.7. Obviously, for the air side, we should use $\epsilon_r = 1$. Conservation of the electric field requires using a harmonic mean of ϵ_{r_1} and ϵ_{r_2} on the dielectric–air interface. According to Suzen et al. [62], this harmonic mean value can be computed using

$$\epsilon_{r_m} = \frac{\epsilon_{r_1}\epsilon_{r_2}}{\epsilon_{r_1}(\Delta x_{m2}/\Delta x_{12}) + \epsilon_{r_2}(\Delta x_{1\,m}/\Delta x_{12})} \quad (10)$$

Parameters of this relation are shown in Fig. 8. The enclosed electrode is grounded with $\phi = 0$. Also, the electric potential on the exposed electrode is an ac voltage with $\phi(t) = \phi_{\max} f(t)$. In this relation, $f(t)$ is considered to be a sin wave function as follows:

$$f(t) = \sin(2\pi\omega t) \quad (11)$$

where ω is the frequency, and $\phi_{\max}$ is the amplitude of the ac voltage supply.

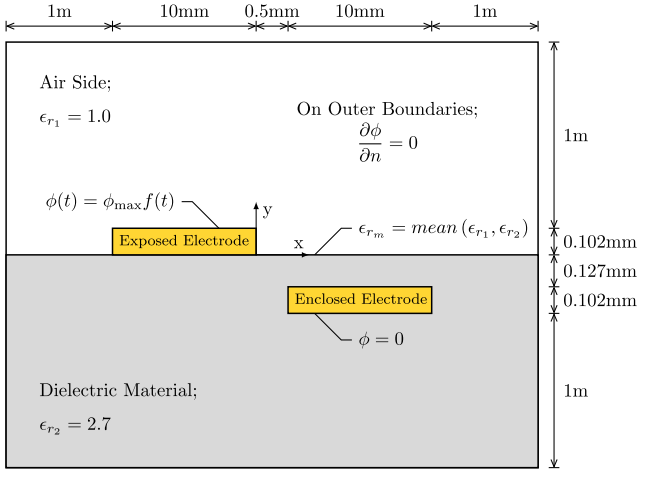

Fig. 7 Computational domain and boundary conditions for Eq. (1) (dimension scales are not correct).

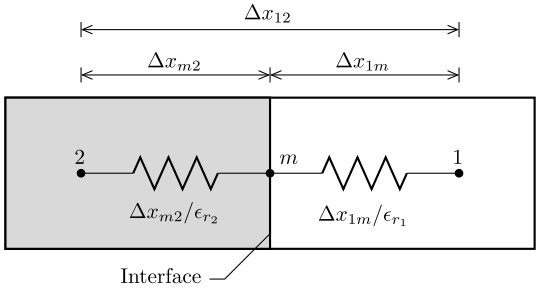

Fig. 8 Permittivity on the interface.

B. Net Charge Density Equation Discretization

The same Cartesian nonuniform computational grid is used for the solution of Eq. (2) as before. Using the same procedure as for the potential equation (see Sec. III.A), Eq. (2) can be discretized as

$$B\rho^{k+1}_{c_{i,j}} = \frac{A_\zeta}{\Delta\zeta}\rho^k_{c_{i+1,j}} + \frac{A_\eta}{\Delta\eta}\rho^k_{c_{i,j+1}} + \epsilon_r \left(\left(\frac{\partial\zeta}{\partial x}\right)^2 \frac{\rho^k_{c_{i+1,j}} + \rho^k_{c_{i-1,j}}}{\Delta\zeta^2} + \left(\frac{\partial\eta}{\partial y}\right)^2 \frac{\rho^k_{c_{i,j+1}} + \rho^k_{c_{i,j-1}}}{\Delta\eta^2} \right) \quad (12)$$

in which A_ζ and A_η are also the same as before, but

$$B = \frac{A_\zeta}{\Delta\zeta} + \frac{A_\eta}{\Delta\eta} + 2\epsilon_r \left(\left(\frac{\partial\zeta}{\partial x}\right)^2 \frac{1}{\Delta\zeta^2} + \left(\frac{\partial\eta}{\partial y}\right)^2 \frac{1}{\Delta\eta^2} \right) + \frac{1}{\lambda_d^2}$$

has a new term.

Numerical solution of Eq. (12) using the boundary conditions of Fig. 9 yields the net charge density ρ_c distribution in the domain. This time around, only the air side of the domain needs to be solved. The normal gradient of ρ_c over the bottom wall is set to zero except for the region above the enclosed electrode. On the outer boundaries, it is assumed that $\rho_c = 0$. Being a simplified model of the plasma actuator, ρ_c on the wall over the enclosed electrode is considered to be in synchronization with the time variations of $\phi(t)$ (applied voltage) [62], which means that

$$\rho_{cw}(x,t) = \rho_{c\,\max} G(x) f(t) \quad (13)$$

in this equation, $\rho_{c\,\max}$ is the maximum allowable charge density in the domain, $f(t)$ is a sin function [Eq. (11)], and $G(x)$ is a function to describe the distribution of the plasma on the wall. A half-Gaussian distribution of plasma on the wall over the enclosed electrode is proposed as [3,63];

$$G(x) = \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right) \quad (14)$$

in which μ is the location of the maximum value, and σ determines the rate of decay of G. The values of constant parameters required for the

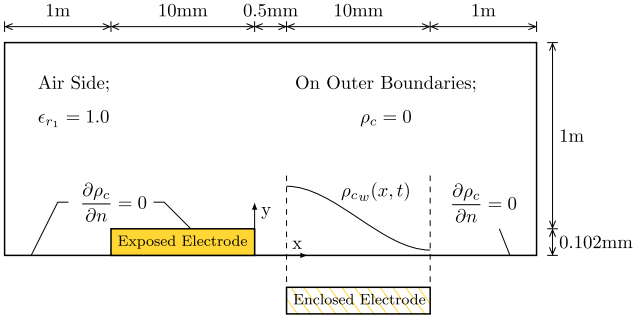

Fig. 9 Computational domain and the boundary conditions for Eq. (2) (dimension scales are not correct).

numerical solution are assumed to be $\sigma = 0.3$, $\lambda_d = 1$ mm, and $\rho_{c\max} = 8 \times 10^{-4}$ (C/m^3).

IV. Results and Discussion

The governing equations of the plasma model are solved numerically using different reconfigurable chips and CPUs. The numerical solution is performed through an iterative procedure, starting from an initial guess. Also, the results are validated with [62], and the high-precision computational time is calculated.

A. Electric Potential

Numerical solution of Eq. (1) is conducted with the use of boundary conditions of Fig. 7. The solution result for the contour of the nondimensionalized ϕ in a 164×117 computational grid for single-precision arithmetic is presented in Fig. 10. This solution was obtained with 10,000 iterations of Eq. (9). The results achieved from FPGAs and CPUs are exactly equal. In Fig. 11, the comparison of the results of ϕ from [62] and the present study, over three separate horizontal lines, are presented. According to this plot, there is a good agreement between results of the electric potential from both works. At its worst, the results from these two studies have a 5.36% difference.

Figure 12 illustrates the plot of computational time of the problem versus mesh size, using all four CPUs with different data precision. The computational time is calculated under three performance conditions, i.e., single-core (without any optimization), multicore (multithreaded with manual optimization), and fully vectorized (with automatic optimization). Four different grids including G_1 (44×31), G_2 (87×61), G_3 (123×88), and G_4 (164×117) are used for the numerical solution. As seen in Fig. 12, computational time increases with grid size. Further, in all cases, the best and worst performances are related to the Core i7-6950X and i7-740QM, respectively. Also, the multicore performance of each CPU is higher than the single-core as well as the fully vectorized results. The computational time in the multicore case has been reduced up to 80% relative to the single-core and up to 50% relative to the fully vectorized cases, for the Core i7-6950X CPU. A spreadsheet of computational time results is available as supplementary material for this paper.

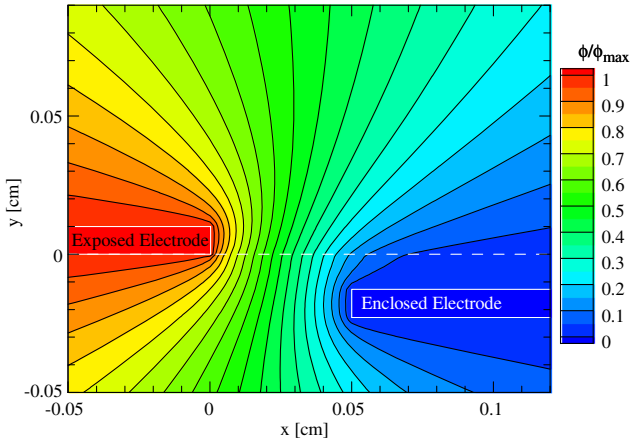

Fig. 10 Contour of ϕ in the numerical solution of Eq. (9).

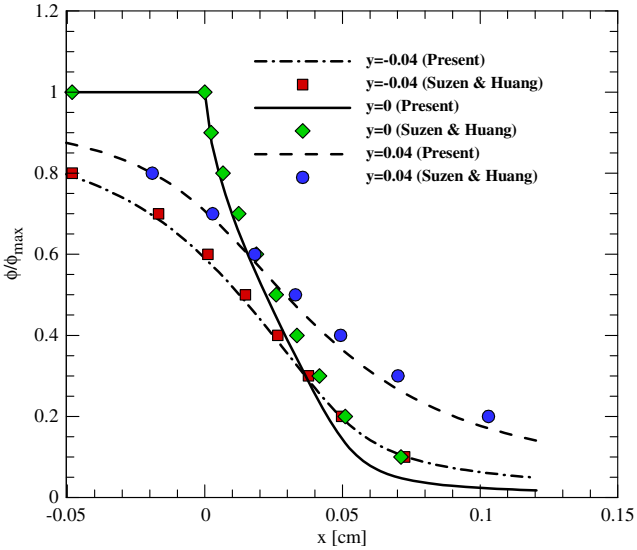

Fig. 11 Comparison of ϕ from [62] and the present study, on three separate lines in the domain.

Two compilers are used for the compilation process of the codes, i.e., the GNU Compiler Collection (GCC) version 6.3.0 and the Microsoft Visual C++ compiler (MSVC) version 19.10.25019 for x64 architecture in Windows 10 operating system. Fully vectorized results are conducted using the autovectorizer of both compilers (i.e., −O3 and /O2 compiler flags for GCC and MSVC, respectively, which result in the highest vectorization ratio possible). The assembly listing file has been extracted using both compilers and checked for the SIMD registers in use. In the case of automatic vectorization, the machine code compiled on all CPUs (except for the Core i7-740QM) contains Advanced Vector Extensions (AVX) registers (YMM0-YMM15) without switching over to SSE registers (XMM0-XMM15). This indicates successful vectorization with AVX architecture by the compiler. The listing file generated without automatic vectorization does not contain AVX registers. A short copy of the assembly listing file with automatic vectorization is available in the Appendix A.

Further, the multicore performance of each CPU is performed using OpenMP application programming interface (API), which enables us to explicitly conduct calculations on multiple threads at the same time. The code is altered in a way to be able to run on multiple threads, which in turn results in a highly parallel and fast executing code. In this case, the raw multithreaded performance of the CPUs (without autovectorization) is taken into account. In conclusion, with the use of OpenMP API, the codes are manually parallelized in contrast to autovectorization, in which the compiler automatically vectorizes the code according to appropriate compiler flags.

Herein, the solution times for single- and double-precision data are approximately the same. This is due to fact that, in current CPU architecture, any operation is executed within a certain amount of clock cycles regardless of the precision of the number [64,65], although this is only true for the single-core results without any optimization. In the case that all the loops within the code are vectorized, AVX operations are capable of operating twice the number of single-precision data as double precision in the same time [66]. The small difference between solution time of single- and double-precision data in the presented results indicates that not all of the loops are vectorized. However, the speed-ups achieved and the AVX registers employed (without switching over to SSE) in the assembly listing of automatically generated vectorized codes reveal that automatic vectorization has been successful to a certain extent. Manually programming multithreaded codes with the highest possible degree of parallelism results in almost the same latency as the automatic vectorization. This indicates that the compiler has done a genuine job of automatically vectorizing the code. Also, further optimization of the codes in question is not possible due to the natural loop data dependency of the used solution algorithm.

Figure 13 demonstrates the plot of computational time of the problem versus mesh size, using the i7-6950X CPU and both FPGAs with double-precision arithmetic. This figure depicts the performance of FPGA hardware in computational speed-up. As seen, XC7VX690T and Zynq-7020 FPGAs are the best and worst performing, respectively. XC7VX690T FPGA is the most powerful hardware for numerical calculations here and has reduced the computational time up to 94, 90, 70, and 43% compared to Zynq-7020 and i7-6950X single-core, fully vectorized, and multicore cases, respectively. Although Zynq-7020 has 30% more latency than the Core i7-6950X single-core case, it is currently 94 and 98% cheaper than the i7-6950X CPU and VC709 board, respectively [54,55,59].

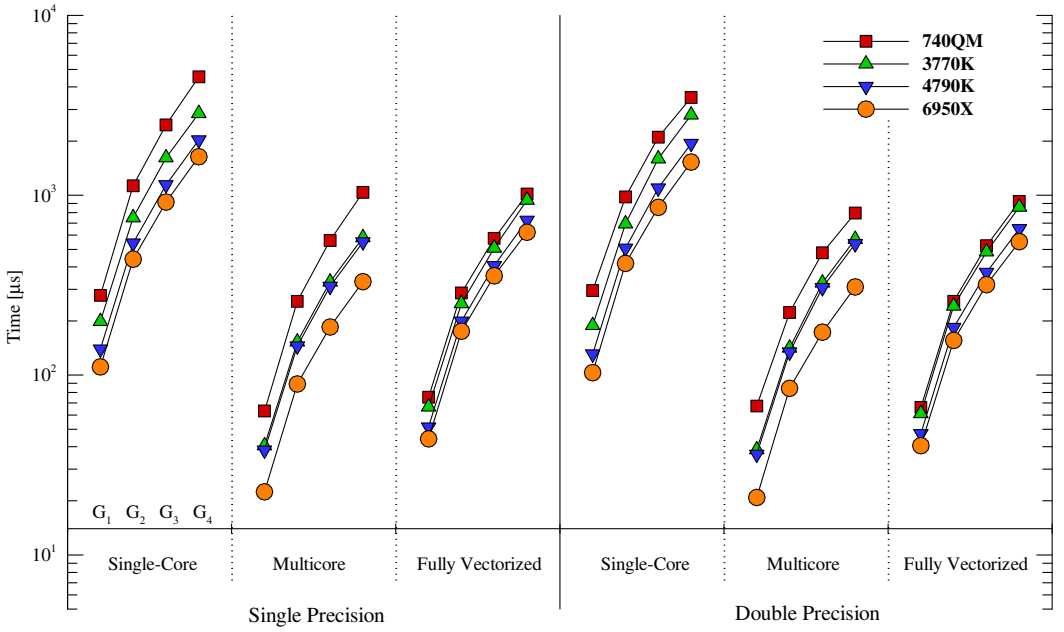

Fig. 12 Computational time of Eq. (9) vs grid size using different CPUs.

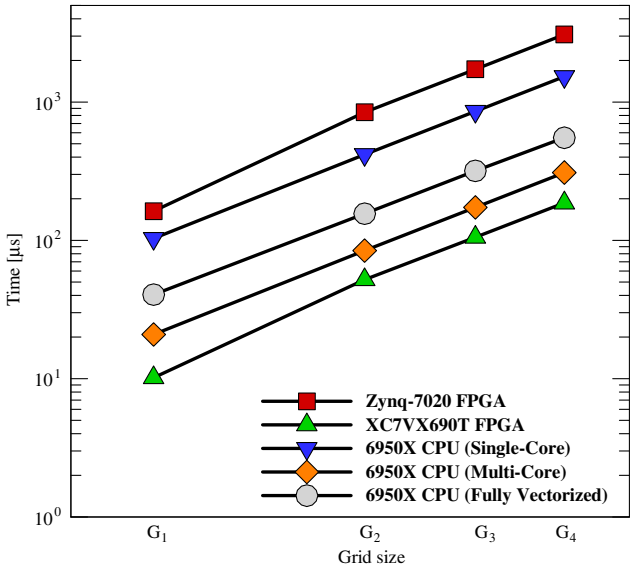

Fig. 13 Computational time of Eq. (9) vs grid size for double-precision arithmetic.

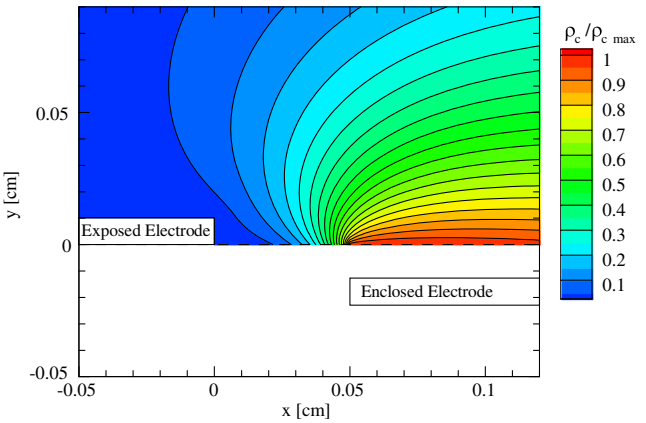

Fig. 14 Contour of ρ_c in the numerical solution of Eq. (12).

B. Net Charge Density

Numerical solution of Eq. (2) is performed with the use of boundary conditions of Fig. 9. The contour of the nondimensionalized ρ_c in a 164×53 computational grid for single-precision arithmetic is presented in Fig. 14. This solution is obtained with 10,000 iterations of Eq. (12) (the results achieved from FPGAs and CPUs are exactly equal). In Fig. 15, the comparison of the results of ρ_c from [62] and the present study, on three separate horizontal lines, is presented. According to this plot, there is a good agreement between the results of charge density from both works. At its worst, the results from these two studies have a 9.14% difference.

Figure 16 illustrates the computational time of the problem versus mesh size, using all four CPUs with different data precision. The computational time is calculated under three performance conditions, i.e., single-core, multicore (multithreaded), and fully vectorized. Four different grids including G_1 (44×14), G_2 (87×27), G_3 (123×40), and G_4 (164×53) are used for the numerical solution. As seen in Fig. 16, computational time increases with grid size. Further, in all cases, the best and worst performances are related to the

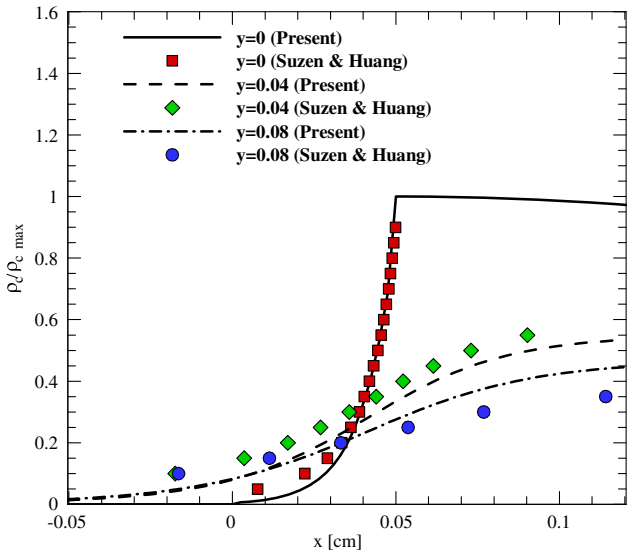

Fig. 15 Comparison of ρ_c from [62] and the present study, on three separate lines in the domain.

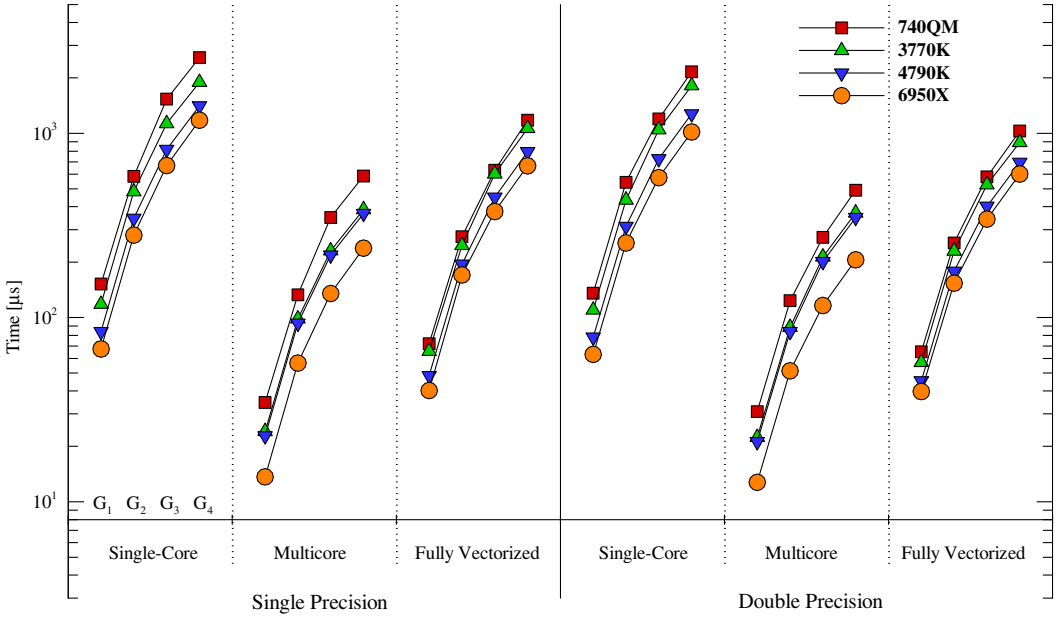

Fig. 16 Computational time of Eq. (12) vs grid size using different CPUs.

Core i7-6950X and i7-740QM, respectively. Also, the multicore performance of each CPU is higher than the single-core as well as the fully vectorized results. The computational time in the multicore case has been reduced up to 80% relative to the single-core and up to 67% relative to the fully vectorized cases for the Core i7-6950X CPU. A spreadsheet of computational time results is available as supplementary material for this paper.

Figure 17 demonstrates the plot of computational time of the problem versus mesh size, using the i7-6950X CPU and both FPGAs with double-precision arithmetic. This figure depicts the performance of FPGA hardware in computational speed-up. As seen, XC7VX690T FPGA and i7-6950X CPU (single-core case) are the best and worst performing, respectively. XC7VX690T FPGA has reduced the computational time up to 95, 96, 93, and 82% compared to Zynq-7020 and i7-6950X single-core, fully vectorized, and multicore cases, respectively. In this test, despite being the cheapest hardware, the computational time result of Zynq-7020 is similar and slightly better than the single-core performance of the i7-6950X CPU.

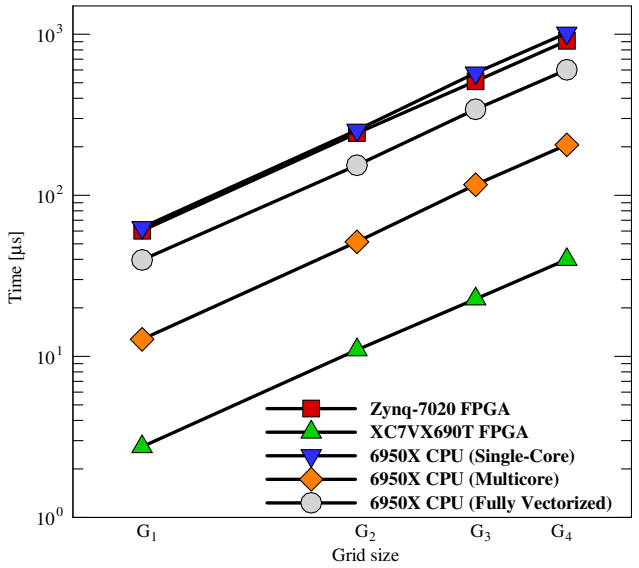

Fig. 17 Computational time of Eq. (12) vs grid size for double-precision arithmetic.

V. Scalability

According to the presented results, the use of FPGA can help speed-up numerical calculations. The scalability of the presented methodology using more powerful reconfigurable hardware for complex CFD problems is studied in this section.

In case of the hybrid system containing both FPGA and CPU, the program is partitioned into sections to be executed on each hardware separately. Specific functions of software are used to control the hardware application. Functions that are supposed to be implemented in the reconfigurable array occupy different hardware area and number of logical components, based on the configuration method. Also, a higher number of configurable resources and more onboard memory mean more area for the optimization and speed-up of numerical calculations. Further, implementing the entire function in the FPGA and minimizing delay in the circuit will increase the performance of the system substantially. Considering these factors, the target problems of the present paper are completely implemented in the reconfigurable hardware, and the microprocessor only controls the input/output of the FPGA.

There are 3600 DSP48Es, 866,400 flip-flops (FFs), and 433,200 total lookup tables (LUTs) available on the XC7VX690T chip. Full implementation of the electric potential problem with double-precision arithmetic on the XC7VX690T FPGA [Eq. (9)] with a 164×117 grid results in 38% DSP48E slice, 19% FF, and 94% LUT usage. Also, full implementation of the net charge density problem [Eq. (12)] on a 164×53 grid occupies 20% of DSP48Es, 7% of FFs, and 35% of the total LUTs available. More complex problems (possibly with larger domain size) will need more logic blocks and reconfigurable area on the chip for a full implementation of the solver. Take the electric potential problem with double-precision arithmetic on a half-million node grid as an example. Complete implementation of this problem (with a speed-up goal in mind) needs about 30,000 DSP48Es, 4 million FFs, and 10 million LUTs. Another trouble is the allowable level of complexity of the governing equations for full implementation on a certain FPGA. This depends on numerical methods as well as hardware implementation procedure. One can calculate the number of logical operations required for the solution of the problem in question. Then, by using currently achieved clock latency of these logical operations, the hardware requirement can be estimated. In the following, three different methods are presented for optimization of the solution of complex problems using the reconfigurable hardware.

A. Partial Evaluation

Partial evaluation [67] is a process that is used to reduce the required hardware resources through optimization based on the known static inputs. An example of such cases is the constant coefficients in the function. If an input to a multiplication function is a constant number, the general multiplication can be replaced with a series of summation operations with static length shifts corresponding to the placement of ones in the binary form of that static number [68]. This method of optimization can help reduce the hardware footprint required for the circuit definition through reducing the logical gates in use. Partial evaluation of DES encryption circuits [69] and the partial evaluation of constant multipliers and fixed polynomial division circuits [70] are other examples of the application of this method.

B. Memory Allocation

There are two main types of memory available for a reconfigurable processor: memory blocks integrated inside the chip and separate memory units outside the chip. The integrated memory can be accessed through very high-speed/throughput connection lines, and it can reduce calculation latency substantially, although the size of this storage is limited by manufacturing technology and physical chip area. If there is not enough embedded memory for the application, we have to use external storage units, which in turn can exist in a large size but have lower speed/throughput. If there are multiple external storage units available to the reconfigurable hardware, it is best to store parallel variables on multiple units, such that they can be accessed at the same time [71]. If embedded memory with the FPGA is used, it is better to occupy each memory block with the variables that are related to the nearest calculation unit on the FPGA. This is done for the sake of reducing read and write operation latency. Embedded storage can also be used as a logical operator as a bonus.

C. Parallelization

When a manual hardware definition language is used to specify the circuit, the user defines the entire architecture and timings. As a result, operations can be explicitly defined to be executed in parallel. In the case of an automatic design tool, the user can help optimize the circuit using compiler directives. In the former method, the user has to mark the areas of the algorithm to be parallelized and assign each part to a parallel calculation thread. A signal/wait method can be used to synchronize calculation threads of this architecture [72]. Automatic parallelization of the inner loops is another method to optimize the calculations and is an attempt to use as most of the chip area as possible. In this method, the compiler selects the most inner logical loop for unrolling and parallel execution. This results in a highly pipelined structure. This kind of optimization using automatic design tools can be accessed through compiler directives. If inputs to the current iteration depend on outputs of the last iteration, the outer loops cannot be unrolled. However, there exist compilers that focus on unrolling all of the loops [73].

VI. Conclusions

The main motivation of the present study is reducing the computational time of a DBD plasma actuator model, using a reconfigurable hardware. This model includes two equations, one for the electric potential and the other for the net charge density in the solution domain. The reconfigurable hardware used in the present study includes a Zynq-7020 and an XC7VX690T manufactured by Xilinx Co. A finite difference representation of the problem is derived to be executed on the computing machine. An automated design procedure is used for the design of the reconfigurable hardware. The computational times using FPGAs are calculated and compared with different CPUs. Using the reconfigurable hardware results in up to 96% computational time reduction compared to a Core i7-6950X CPU. The scalability of the presented methodology is explored for more complex problems.

Appendix A: Assembly Listing File Samples

Sample sections of the assembly listing file to indicate the utilization of AVX registers for the electric potential problem for the fully vectorized case (with highest optimization ratio) using both of the compilers are presented here.

A.1. Microsoft Visual C++ Compiler

```
_TEXT	SEGMENT
out$ = 160
A$ = 168
B$ = 176
count$ = 184
?run_AVX_8@@YAXPEATMat44@@PEBT1@1H@Z PROC
mov	rax, rsp
push	rbp
sub	rsp, 144
vmovaps XMMWORD PTR [rax − 24], xmm6
vmovaps XMMWORD PTR [rax − 40], xmm7
vmovaps XMMWORD PTR [rax − 56], xmm8
vmovaps XMMWORD PTR [rax − 72], xmm9
vmovaps XMMWORD PTR [rax − 88], xmm10
vmovaps XMMWORD PTR [rax − 104], xmm11
vmovaps XMMWORD PTR [rax − 120], xmm12
lea	rbp, QWORD PTR [rax − 120]
and	rbp, −32
xor	r11d, r11d
test	r9d, r9d
jle	$LN1@run_AVX_8
npad	14
$LL3@run_AVX_8:
mov	eax, DWORD PTR ?the_mask@@3HA
and	eax, r11d
inc	r11d
movsxd	r10, eax
shl	r10, 6
vmovups	ymm10, YMMWORD PTR [r10 + rdx]
vbroadcastf128	ymm0, XMMWORD PTR [r10 + r8]
vmovups	ymm12, YMMWORD PTR [r10 + rdx + 32]
vbroadcastf128	ymm6, XMMWORD PTR [r10 + r8 + 32]
vbroadcastf128	ymm5, XMMWORD PTR [r10 + r8 + 48]
vshufps	ymm1, ymm10, ymm10, 85
vmovups	ymm4, ymm0
vmovups	ymm11, ymm0
vbroadcastf128	ymm0, XMMWORD PTR [r10 + r8 + 16]
vmovups	ymm7, ymm0
vmulps	ymm3, ymm1, ymm0
vshufps	ymm1, ymm10, ymm10, 170
vshufps	ymm0, ymm10, ymm10, 0
vmulps	ymm2, ymm0, ymm4
vmulps	ymm0, ymm1, ymm6
vaddps	ymm4, ymm3, ymm2
vaddps	ymm3, ymm4, ymm0
vshufps	ymm2, ymm10, ymm10, 255
vmulps	ymm1, ymm2, ymm5
vaddps	ymm0, ymm3, ymm1
vshufps	ymm1, ymm12, ymm12, 0
vmulps	ymm2, ymm1, ymm11
vmovups	YMMWORD PTR [r10 + rcx], ymm0
vshufps	ymm0, ymm12, ymm12, 85
vmulps	ymm3, ymm0, ymm7
vshufps	ymm0, ymm12, ymm12, 170
vaddps	ymm4, ymm3, ymm2
vmulps	ymm1, ymm0, ymm6
vshufps	ymm2, ymm12, ymm12, 255
vaddps	ymm3, ymm4, ymm1
vmulps	ymm0, ymm2, ymm5
vaddps	ymm1, ymm3, ymm0
vmovups	YMMWORD PTR [r10 + rcx + 32], ymm1
cmp	r11d, r9d
jl	$LL3@run_AVX_8
$LN1@run_AVX_8:
Vzeroupper
lea	r11, QWORD PTR [rsp + 144]
vmovaps	xmm6, XMMWORD PTR [r11−16]
vmovaps	xmm7, XMMWORD PTR [rsp + 112]
```

Appendix (*Continued.*)

```
vmovaps   xmm8,   XMMWORD PTR [r11–48]
vmovaps   xmm9,   XMMWORD PTR [r11–64]
vmovaps   xmm10,  XMMWORD PTR [r11–80]
vmovaps   xmm11,  XMMWORD PTR [r11–96]
vmovaps   xmm12,  XMMWORD PTR [r11–112]
mov       rsp,    r11
pop       rbp
ret       0
?run_AVX_8@@YAXPEATMat44@@PEBT1@1H@Z ENDP
_TEXT   ENDS
```

A.2. GNU Compiler Collection

```
.L21:
movl      the_mask(%rip),   %eax
andl      %r11d,   %eax
Cltq
salq      $6,   %rax
leaq      (%r8,%rax),   %r10
leaq      (%rdx,%rax),   %rbx
addq      %rcx,   %rax
Vzeroupper
vmovups (%rbx),   %ymm0
addl      $1,   %r11d
vmovups 32(%rbx),   %ymm4
cmpl      %r11d,      %r9d
vshufps   $0,   %ymm0,   %ymm0,   %ymm5
vbroadcastf128   (%r10),   %ymm3
vshufps   $85,   %ymm0,   %ymm0,   %ymm12
vbroadcastf128   16(%r10),   %ymm9
vshufps   $0,   %ymm4,   %ymm4,   %ymm10
vbroadcastf128 32   (%r10),   %ymm8
vshufps   $85,   %ymm4,   %ymm4,   %ymm2
vbroadcastf128   48(%r10),   %ymm7
vshufps   $170,   %ymm0,   %ymm0,   %ymm11
vshufps   $255,   %ymm0,   %ymm0,   %ymm6
vshufps   $170,   %ymm4,   %ymm4,   %ymm1
vshufps   $255,   %ymm4,   %ymm4,   %ymm0
vmulps   %ymm2,   %ymm9,   %ymm2
vmulps   %ymm5,   %ymm3,   %ymm4
vmulps   %ymm12,   %ymm9,   %ymm5
vmulps   %ymm10,   %ymm3,   %ymm3
vmulps   %ymm1,   %ymm8,   %ymm1
vaddps   %ymm5,   %ymm4,   %ymm5
vmulps   %ymm11,   %ymm8,   %ymm4
vaddps   %ymm2,   %ymm3,   %ymm2
vmulps   %ymm6,   %ymm7,   %ymm6
vmulps   %ymm0,   %ymm7,   %ymm0
vaddps   %ymm4,   %ymm5,   %ymm5
vaddps   %ymm1,   %ymm2,   %ymm1
vaddps   %ymm6,   %ymm5,   %ymm4
vaddps   %ymm0,   %ymm1,   %ymm0
vmovups %ymm4,   (%rax)
vmovups %ymm0,   32(%rax)
jne.      L21
Vzeroupper
```

References

[1] Hoffmann, K. A., and Chiang, S. T., *Computational Fluid Dynamics*, Vol. 1, Computational Fluid Dynamics, Engineering Education System, Wichita, KS, 2000, pp. 1–2.

[2] Caughey, D., and Hafez, M., *Frontiers of Computational Fluid Dynamics 2006*, Computational Fluid Dynamics Series, World Scientific, Singapore, 2005, pp. 1–10.

[3] Enloe, C., McLaughlin, T. E., Van Dyken, R. D., Kachner, K., Jumper, E. J., and Corke, T. C., "Mechanisms and Responses of a Single Dielectric Barrier Plasma Actuator: Plasma Morphology," *AIAA Journal*, Vol. 42, No. 3, 2004, pp. 589–594. doi:10.2514/1.2305

[4] Likhanskii, A. V., Shneider, M. N., Macheret, S. O., and Miles, R. B., "Modeling of Dielectric Barrier Discharge Plasma Actuator in Air," *Journal of Applied Physics*, Vol. 103, No. 5, 2008, Paper 053305. doi:10.1063/1.2837890

[5] Yoon, J.-S., and Han, J.-H., "One-Equation Modeling and Validation of Dielectric Barrier Discharge Plasma Actuator Thrust," *Journal of Physics D: Applied Physics*, Vol. 47, No. 40, 2014, Paper 405202. doi:10.1088/0022-3727/47/40/405202

[6] Orlov, D., Corke, T., and Patel, M., "Electric Circuit Model for Aerodynamic Plasma Actuator," *44th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2006-1206, Jan. 2006.

[7] Hall, K., Jumper, E., Corke, T., and McLaughlin, T., "Potential Flow Model of a Plasma Actuator as a Lift Enhancement Device," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2005-0783, Jan. 2005.

[8] Thomas, F. O., Corke, T. C., Iqbal, M., Kozlov, A., and Schatzman, D., "Optimization of Dielectric Barrier Discharge Plasma Actuators for Active Aerodynamic Flow Control," *AIAA Journal*, Vol. 47, No. 9, 2009, pp. 2169–2178. doi:10.2514/1.41588

[9] Suzen, Y., Huang, G., and Ashpis, D., "Numerical Simulations of Flow Separation Control in Low-Pressure Turbines Using Plasma Actuators," *45th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2007-0937, Jan. 2007.

[10] Singh, K. P., and Roy, S., "Force Approximation for a Plasma Actuator Operating in Atmospheric Air," *Journal of Applied Physics*, Vol. 103, No. 1, 2008, Paper 013305.

[11] Corrigan, A., Camelli, F. F., Lhner, R., and Wallin, J., "Running Unstructured Grid-Based CFD Solvers on Modern Graphics Hardware," *International Journal for Numerical Methods in Fluids*, Vol. 66, No. 2, 2011, pp. 221–229. doi:10.1002/fld.v66.2

[12] Dong, S., and Karniadakis, G. E., "Dual-Level Parallelism for High-Order CFD Methods," *Parallel Computing*, Vol. 30, No. 1, 2004, pp. 1–20. doi:10.1016/j.parco.2003.05.020

[13] Phillips, E. H., Zhang, Y., Davis, R. L., and Owens, J. D., "Acceleration of 2-D Compressible Flow Solvers with Graphics Processing Unit Clusters," *Journal of Aerospace Computing, Information, and Communication*, Vol. 8, No. 8, 2011, pp. 237–249. doi:10.2514/1.44909

[14] Ma, Z., Wang, H., and Pu, S., "GPU Computing of Compressible Flow Problems by a Meshless Method with Space-Filling Curves," *Journal of Computational Physics*, Vol. 263, 2014, pp. 113–135. doi:10.1016/j.jcp.2014.01.023

[15] Stone, C. P., and Davis, R. L., "Techniques for Solving Stiff Chemical Kinetics on Graphical Processing Units," *Journal of Propulsion and Power*, Vol. 29, No. 4, 2013, pp. 764–773. doi:10.2514/1.B34874

[16] Priimak, D., "Finite Difference Numerical Method for the Superlattice Boltzmann Transport Equation and Case Comparison of CPU(C) and GPU(CUDA) Implementations," *Journal of Computational Physics*, Vol. 278, 2014, pp. 182–192. doi:10.1016/j.jcp.2014.08.028

[17] Xu, C., Deng, X., Zhang, L., Fang, J., Wang, G., Jiang, Y., Cao, W., Che, Y., Wang, Y., Wang, Z., Liu, W., and Cheng, X., "Collaborating CPU and GPU for Large-Scale High-Order CFD Simulations with Complex Grids on the TianHe-1A Supercomputer," *Journal of Computational Physics*, Vol. 278, 2014, pp. 275–297. doi:10.1016/j.jcp.2014.08.024

[18] Thomas, S., Amiraux, M., and Baeder, J. D., "Modeling the Two-Phase Flowfield Beneath a Hovering Rotor on Graphics Processing Units," *AIAA Journal*, Vol. 53, No. 8, 2015, pp. 2300–2320. doi:10.2514/1.J053661

[19] Liu, X., Zhong, Z., and Xu, K., "A Hybrid Solution Method for CFD Applications on GPU-Accelerated Hybrid HPC Platforms," *Future Generation Computer Systems*, Vol. 56, 2016, pp. 759–765. doi:10.1016/j.future.2015.08.002

[20] Chan, J., Wang, Z., Modave, A., Remacle, J.-F., and Warburton, T., "GPU-Accelerated Discontinuous Galerkin Methods on Hybrid Meshes," *Journal of Computational Physics*, Vol. 318, 2016, pp. 142–168. doi:10.1016/j.jcp.2016.04.003

[21] Remacle, J.-F., Gandham, R., and Warburton, T., "GPU Accelerated Spectral Finite Elements on All-Hex Meshes," *Journal of Computational Physics*, Vol. 324, 2016, pp. 246–257. doi:10.1016/j.jcp.2016.08.005

[22] Tredak, P., Rudnicki, W. R., and Majewski, J. A., "Efficient Implementation of the Many-Body Reactive Bond Order (REBO) Potential on GPU," *Journal of Computational Physics*, Vol. 321, 2016, pp. 556–570. doi:10.1016/j.jcp.2016.05.061

[23] Liang, Y., Xing, X., and Li, Y., "A GPU-Based Large-Scale Monte Carlo Simulation Method for Systems with Long-Range Interactions,"

Journal of Computational Physics, Vol. 338, 2017, pp. 252–268. doi:10.1016/j.jcp.2017.02.069

[24] Kilts, S., *Advanced FPGA Design: Architecture, Implementation, and Optimization*, Wiley, Hoboken, NJ, 2007, pp. 11–16.

[25] Smith, W. D., and Schnore, A. R., "Towards an RCC-Based Accelerator for Computational Fluid Dynamics Applications," *Journal of Supercomputing*, Vol. 30, No. 3, 2004, pp. 239–261. doi:10.1023/B:SUPE.0000045211.07895.cb

[26] Huang, W., Saxena, N., and McCluskey, E. J., "A Reliable LZ Data Compressor on Reconfigurable Coprocessors," *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE Publ., Piscataway, NJ, 2000, pp. 249–258.

[27] Crookes, D., Benkrid, K., Bouridane, A., Alotaibi, K., and Benkrid, A., "Design and Implementation of a High Level Programming Environment for FPGA-Based Image Processing," *IEE Proceedings—Vision, Image and Signal Processing*, Vol. 147, No. 4, 2000, pp. 377–384. doi:10.1049/ip-vis:20000579

[28] Dick, C., and Harris, F., "FPGA Signal Processing Using Sigma-Delta Modulation," *IEEE Signal Processing Magazine*, Vol. 17, No. 1, 2000, pp. 20–35. doi:10.1109/79.814644

[29] Asano, S., Maruyama, T., and Yamaguchi, Y., "Performance Comparison of FPGA, GPU and CPU in Image Processing," *Proceedings of the 2009 International Conference on Field Programmable Logic and Applications*, IEEE Publ., Piscataway, NJ, 2009, pp. 126–131. doi:10.1109/FPL.2009.5272532

[30] Hauser, T., "A Flow Solver for a Reconfigurable FPGA-Based Hypercomputer," *43rd AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2005-1382, Jan. 2005. doi:10.2514/6.2005-1382

[31] Nunez, R., Gonzalez, J., and Camberos, J., "Large-Scale Numerical Solution of Partial Differential Equations with Reconfigurable Computing," *18th AIAA Computational Fluid Dynamics Conference*, AIAA Paper 2007-4085, June 2007. doi:10.2514/6.2007-4085

[32] Sano, K., Iizuka, T., and Yamamoto, S., "Systolic Architecture for Computational Fluid Dynamics on FPGAs," *Proceedings of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE Publ., Piscataway, NJ, 2007, pp. 107–116. doi:10.1109/FCCM.2007.20

[33] Morishita, H., Osana, Y., Fujita, N., and Amano, H., "Exploiting Memory Hierarchy for a Computational Fluid Dynamics Accelerator on FPGAS," *Proceedings of the 2008 International Conference on ICECE Technology*, IEEE Publ., Piscataway, NJ, 2008, pp. 193–200. doi:10.1109/FPT.2008.4762383

[34] Sun, J., Peterson, G. D., and Storaasli, O. O., "High-Performance Mixed-Precision Linear Solver for FPGAs," *IEEE Transactions on Computers*, Vol. 57, No. 12, 2008, pp. 1614–1623. doi:10.1109/TC.2008.89

[35] Andrés, E., Carreras, C., Caffarena, G., Molina, M. D. C., Nieto-Taladriz, O., and Palacios, F., "A Methodology for CFD Acceleration Through Reconfigurable Hardware," *46th AIAA Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2008-0481, Jan. 2008. doi:10.2514/6.2008-481

[36] Inakagata, K., Morishita, H., Osana, Y., Fujita, N., and Amano, H., "Modularizing Flux Limiter Functions for a Computational Fluid Dynamics Accelerator on FPGAs," *Proceedings of the 2009 International Conference on Field Programmable Logic and Applications*, IEEE Publ., Piscataway, NJ, 2009, pp. 654–657. doi:10.1109/FPL.2009.5272347

[37] Andrés, E., Widhalm, M., and Caloto, A., "Achieving High Speed CFD Simulations: Optimization, Parallelization, and FPGA Acceleration for the Unstructured DLR TAU Code," *47th AIAA Aerospace Sciences Meeting*, AIAA Paper 2009-0759, Jan. 2009. doi:10.2514/6.2009-759

[38] Sanchez-Roman, D., Sutter, G., Lopez-Buedo, S., Gonzalez, I., Gomez-Arribas, F. J., and Aracil, J., "An Euler Solver Accelerator in FPGA for Computational Fluid Dynamics Applications," *Proceedings of the 2011 Southern Conference on Programmable Logic*, IEEE Publ., Piscataway, NJ, 2011, pp. 149–154. doi:10.1109/SPL.2011.5782640

[39] Sano, K., Hatsuda, Y., and Yamamoto, S., "Performance Evaluation of FPGA-Based Custom Accelerators for Iterative Linear-Equation Solvers," *20th AIAA Computational Fluid Dynamics Conference*, AIAA Paper 2011-3223, June 2011. doi:10.2514/6.2011-3223

[40] Liu, I., Lee, E. A., Viele, M., Wang, G., and Andrade, H., "A Heterogeneous Architecture for Evaluating Real-Time One-Dimensional Computational Fluid Dynamics on FPGAs," *Proceedings of the 2012 IEEE 20th Annual International Symposium on Field-Programmable Custom Computing Machines*, IEEE Publ., Piscataway, NJ, 2012, pp. 125–132. doi:10.1109/FCCM.2012.31

[41] Constantinides, G., Kinsman, A., and Nicolici, N., "Numerical Data Representations for FPGA-Based Scientific Computing," *IEEE Design & Test of Computers*, Vol. 28, No. 4, 2011, pp. 8–17. doi:10.1109/MDT.2011.48

[42] Bleris, L. G., Vouzis, P. D., Arnold, M. G., and Kothare, M. V., "A Co-Processor FPGA Platform for the Implementation of Real-Time Model Predictive Control," *Proceedings of the American Control Conference*, 2006, IEEE Publ., Piscataway, NJ, p. 6.

[43] Sano, K., Hatsuda, Y., and Yamamoto, S., "Multi-FPGA Accelerator for Scalable Stencil Computation with Constant Memory Bandwidth," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 3, 2014, pp. 695–705. doi:10.1109/TPDS.2013.51

[44] Lin, Y., Wang, F., Zheng, X., Gao, H., and Zhang, L., "Monte Carlo Simulation of the Ising Model on FPGA," *Journal of Computational Physics*, Vol. 237, March 2013, pp. 224–234. doi:10.1016/j.jcp.2012.12.005

[45] Kono, Y., Sano, K., and Yamamoto, S., "Scalability Analysis of Tightly-Coupled FPGA-Cluster for Lattice Boltzmann Computation," *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications*, IEEE Publ., Piscataway, NJ, 2012, pp. 120–127.

[46] Gan, L., Fu, H., Luk, W., Yang, C., Xue, W., Huang, X., Zhang, Y., and Yang, G., "Accelerating Solvers for Global Atmospheric Equations Through Mixed-Precision Data Flow Engine," *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications*, IEEE Publ., Piscataway, NJ, 2013, pp. 1–6.

[47] Dohi, K., Fukumoto, K., Shibata, Y., and Oguri, K., "Performance Modeling and Optimization of 3-D Stencil Computation on a Stream-Based FPGA Accelerator," *Proceedings of the 2013 International Conference on Reconfigurable Computing and FPGAs*, IEEE Publ., Piscataway, NJ, 2013, pp. 1–6.

[48] Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., and Cong, J., "Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks," *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Assoc. for Computing Machinery, New York, 2015, pp. 161–170.

[49] Williams, S., Waterman, A., and Patterson, D., "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, Vol. 52, No. 4, 2009, pp. 65–76. doi:10.1145/1498765

[50] Nagasu, K., Sano, K., Kono, F., and Nakasato, N., "FPGA-Based Tsunami Simulation: Performance Comparison with GPUS, and Roofline Model for Scalability Analysis," *Journal of Parallel and Distributed Computing*, Vol. 106, Supplement C, 2017, pp. 153–169. doi:10.1016/j.jpdc.2016.12.015

[51] Ebrahimi, A., and Zandsalimy, M., "Evaluation of FPGA Hardware as a New Approach for Accelerating the Numerical Solution of CFD Problems," *IEEE Access*, Vol. 5, May 2017, pp. 9717–9727. doi:10.1109/ACCESS.2017.2705434

[52] Hartenstein, R., "A Decade of Reconfigurable Computing: A Visionary Retrospective," *Proceedings of the Conference on Design, Automation and Test in Europe*, IEEE Publ., Piscataway, NJ, 2001, pp. 642–649.

[53] Tessier, R., Pocek, K., and DeHon, A., "Reconfigurable Computing Architectures," *Proceedings of the IEEE*, Vol. 103, No. 3, 2015, pp. 332–354. doi:10.1109/JPROC.2014.2386883

[54] "Zynq-7000 FPGA Family," Xilinx, Inc., San Jose, CA, https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html [retrieved 1 May 2017].

[55] "Virtex-7 FPGA Family," Xilinx, Inc., San Jose, CA, https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html [retrieved 1 May 2017].

[56] "Intel Core i7-740QM Processor (6M Cache, 1.73 GHz) Product Specifications," Intel Corporation, San Jose, CA, https://ark.intel.com/products/49024/Intel-Core-i7-740QM-Processor-6M-cache-1_73-GHz [retrieved 1 May 2017].

[57] "Intel Core i7-3770 K Processor (8M Cache, up to 3.90 GHz) Product Specifications," Intel Corporation, San Jose, CA, https://ark.intel.com/products/65523/Intel-Core-i7-3770K-Processor-8M-Cache-up-to-3_90-GHz [retrieved 1 May 2017].

[58] "Intel Core i7-4790 K Processor (8M Cache, up to 4.40 GHz) Product Specifications," Intel Corporation, San Jose, CA, https://ark.intel.com/products/80807/Intel-Core-i7-4790K-Processor-8M-Cache-up-to-4_40-GHz [retrieved 1 May 2017].

[59] "Intel Core i7-6950X Processor Extreme Edition (25M Cache, up to 3.50 GHz) Product Specifications," Intel Corporation, San Jose, CA, https://ark.intel.com/products/94456/Intel-Core-i7-6950X-Processor-Extreme-Edition-25M-Cache-up-to-3_50-GHz [retrieved 1 May 2017].

[60] "Intellectual Property," Xilinx, Inc., San Jose, CA, https://www.xilinx.com/products/intellectual-property.html [retrieved 1 May 2017].

[61] Laten, J. B., and LeBeau, R. P., "Improving the Performance of a Plasma Actuator Model for DBD and Multi-Encapsulated Electrode Actuators," *55th AIAA Aerospace Sciences Meeting*, AIAA Paper 2017-1808, 2017.

[62] Suzen, Y., Huang, G., Jacob, J., and Ashpis, D., "Numerical Simulations of Plasma Based Flow Control Applications," *35th AIAA Fluid Dynamics Conference and Exhibit*, AIAA Paper 2005-4633, June 2005.

[63] Enloe, C. L., McLaughlin, T., Font, G. I., and Baughn, J. W., "Parameterization of Temporal Structure in the Single-Dielectric-Barrier Aerodynamic Plasma Actuator," *AIAA Journal*, Vol. 44, No. 6, 2006, pp. 1127–1136.
doi:10.2514/1.16297

[64] Hennessy, J., and Patterson, D., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Series in Computer Architecture and Design, Elsevier Science, Burlington, MA, 2011, pp. 520–575.

[65] Silc, J., Robic, B., and Ungerer, T., *Processor Architecture: From Dataflow to Superscalar and Beyond*, Springer, Berlin, 2012, pp. 206–230.

[66] Lomont, C., *Introduction to Intel Advanced Vector Extensions*, Intel Corporation, Santa Clara, CA, June 2011, https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions [1 May 2017].

[67] Prakash, A. R., and Kirubaveni, S., "Performance Evaluation of FFT Processor Using Conventional and Vedic Algorithm," *Proceedings of the 2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology*, IEEE Publ., Piscataway, NJ, 2013, pp. 89–94.
doi:10.1109/ICE-CCN.2013.6528470

[68] Nedjah, N., and de Macedo Mourelle, L., "Reconfigurable Hardware Implementation of Montgomery Modular Multiplication and Parallel Binary Exponentiation," *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, IEEE Publ., Piscataway, NJ, 2002, pp. 226–233.
doi:10.1109/DSD.2002.1115373

[69] Leonard, J., and Mangione-Smith, W. H., *A Case Study of Partially Evaluated Hardware Circuits: Key-Specific DES*, Springer, Berlin, 1997, pp. 151–160.
doi:10.1007/3-540-63465-7_220

[70] Payne, R., *Run-Time Parameterised Circuits for the Xilinx XC6200*, Springer, Berlin, 1997, pp. 161–172.
doi:10.1007/3-540-63465-7_221

[71] Gokhale, M. B., and Stone, J. M., "Automatic Allocation of Arrays to Memories in FPGA Processors with Multiple Memory Banks," *Proceedings of the 7th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, IEEE Publ., Piscataway, NJ, 1999, pp. 63–69.
doi:10.1109/FPGA.1999.803668

[72] Cronquist, D. C., Franklin, P., Berg, S. G., and Ebeling, C., "Specifying and Compiling Applications for Rapid," *Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines*, IEEE Publ., Piscataway, NJ, 1998, pp. 116–125.
doi:10.1109/FPGA.1998.707889

[73] Goldstein, S. C., Schmit, H., Budiu, M., Cadambi, S., Moe, M., and Taylor, R. R., "Piperench: A Reconfigurable Architecture and Compiler," *Computer*, Vol. 33, No. 4, 2000, pp. 70–77.
doi:10.1109/2.839324