*Progress In Electromagnetics Research C, Vol. 105, 161–174, 2020*

# Parallel Hardware Architecture of the 3D FDTD Algorithm with Convolutional Perfectly Matched Layer Boundary Condition

**Chang Kong and Tao Su***

**Abstract**—The finite-difference time-domain (FDTD) algorithm is a numerical stencil computation method, which is widely used in solving electromagnetic simulation problems. However, this algorithm is both computing and storage intensive, so the simulation efficiency is usually restricted in software implementation on CPUs. Recently, hardware accelerators have proved to be effective in improving the performance of various stencil computations. In this paper, we propose a hardware architecture of the 3D FDTD algorithm along with a practical convolutional perfectly matched layer (CPML) boundary condition and implement it on a field programmable gate array (FPGA). By applying the chain processing elements array and temporal parallel strategy, the proposed accelerator can achieve a maximum of 608 mega cells per second (Mcells/s), which is approximately 6 times higher than that of other reported designs on FPGAs. Moreover, the accelerator can maintain the speed above 467 Mcells/s for different grid sizes and CPML layers without modifying the hardware design, which demonstrates the performance stability and flexibility of the architecture under various applications.

## 1. INTRODUCTION

Finite-difference time-domain (FDTD) algorithm is a commonly used numerical method for analyzing antennas, designing microelectronic devices, performing electromagnetic compatibility analysis, and resolving other electromagnetic field problems [1–6]. The FDTD method belongs to a type of stencil computation, which updates the electric and magnetic field components of each Yee cell in the time domain following a regular calculation pattern. In realizing the FDTD algorithm, the high data and operational intensity of this algorithm consume many on-chip-memory spaces and hardware logic resources. This feature results in a significant time cost in the software implementation of the algorithm on CPUs. Besides, the regularity of the stencil expression determines its suitability for parallel processing. Thus, hardware acceleration has become an indispensable and promising method to improve the efficiency of simulations to expand the practical applications of FDTD algorithms.

Hardware processing solutions based on the graphics processing unit (GPU) platform have been widely reported [7–12]. They achieved a considerable acceleration effect owing to the larger global cache and more abundant logic resources on GPUs than CPUs. Multiple cores and cluster structures accompanied with compute unified device architecture (CUDA) models can also make effective use of parallel properties in FDTD. However, GPU-based accelerations suffer from server memory bandwidth bottleneck. The fixed underlying hardware components also limit the space for optimization of dataflow and pipelines to improve parallelism. These factors lead to difficulties to further improve the GPU performance.

Another effective hardware approach to accelerate the algorithm is using a field programmable gate array (FPGA) based accelerator. FPGA-based accelerators have shown their potential for accelerating

*Received 28 July 2020, Accepted 4 September 2020, Scheduled 17 September 2020*
* Corresponding author: Tao Su (sutao@mail.sysu.edu.cn).
The authors are with the School of Electronics and Information Technology, Sun Yat-Sen University, No. 132 Wai Huan Dong Lu, Panyu District, Guangzhou, Guangdong, P. R. China.

kinds of computation intense algorithms, such as fast Fourier transforms, compression algorithms, convolution neural network, and so on [13–18]. Recently, several FPGA accelerators for 2D and 3D FDTD algorithms have been successively reported [19–24]. Since FPGAs allow designers to customize nearly every detail of the data path and memory access structure of the accelerator, these works apply deep pipeline architecture and dedicated data movement to increase the throughput. Despite some achievements, most accelerators in these works cannot be efficiently used in practical applications because of certain deficiencies. Lack of usability under different simulation scenes is the first reason. For example, some researchers set aside the data exchange between FPGA and off-chip memory to achieve better performance speedup [22, 23]. However, this approach restricts the size of the simulation space to the available space of the on-chip memory on FPGA, so it is not applicable in larger simulation models. In terms of boundary conditions, some works only apply simple boundary conditions like the absorbing boundary condition (ABC) and perfectly matched layer (PML), or even consider no boundary conditions [22–24]. In practical applications, a state-of-art boundary condition algorithm is essential for obtaining an accurate simulation result in a finite-space model. Compared to ABC or PML, convolutional PML (CPML) is currently more advanced and widely used due to its better wave absorption properties in the boundaries and less reflection to the domain when simulating open space [25]. However, this algorithm introduces extra calculation operations and auxiliary terms to the FDTD scheme and causes greater pressure on data transfer between the FPGA and the external memory. Therefore, implementing this boundary condition is more challenging than others. The work in [26] implements the CPML algorithm on four FPGAs and additionally uses a CPU to execute the extra operations, but the performance is not satisfactory. Moreover, there are few discussions on the performance variation of the reported accelerators under different sizes of simulation spaces, which is an important feature of performance stability and essential for practical use. Additionally, the simulation efficiency of FPGA-based accelerators needs further improvement by optimizing the memory bandwidth usage and the pipeline to make accelerators available for larger space applications.

In this paper, we propose a hardware accelerator architecture dedicated to the 3D FDTD algorithm and CPML boundary condition and deploy it on an FPGA. In this hardware architecture, the processing element (PE) is designed as a six-stage computation pipeline to integrate the function of processing two algorithms simultaneously. It can also preserve the regularity of the dataflow to a small cost of hardware logic resources. Multiple PEs are connected in a one-dimensional topology with a dedicated memory mapping method, called chain PEs array, to improve the usability and performance under different grid sizes. In addition, we also apply a temporal parallel strategy, which enables the accelerator to process several iterations on different time steps. This method further increases the running speed and reduces the data transactions required at each time step calculation to save memory bandwidth. The experimental results indicate that the peak performance of the proposed accelerator can reach 608 mega cells per second (Mcells/s), which is approximately 6 times faster than the work in [26]. In addition, the performance can maintain above 467 Mcells/s under different grid sizes and CPML layers. For a certain main frequency or the memory bandwidth, the result of the performance comparison to other GPU and FPGA accelerators is more prominent. These evaluations reveal the potential and value of this hardware architecture to be deployed on application-specific integrated circuit (ASIC) platforms in the future.

## 2. ACCELERATOR DESIGN FOR THE 3D FDTD AND CPML BOUNDARY CONDITIONS

### 2.1. Overview of the FDTD Accelerator Schematic

In this section, we introduce the schematic of processing FDTD and CPML algorithms on a hardware accelerator and some details of their formulas. Fig. 1 shows the flowchart of a complete electromagnetic simulation process using FDTD. The process starts with the accelerator initialization. The host PC sends various information and configuration parameters relevant to simulation geometrical features to the accelerator, including the size of the input grid, total time step, material spatial distribution, material coefficients, and so on. Then follows the computation of electric field components, including $E_x$, $E_y$, and $E_z$, at each Yee cell ("cell" hereinafter) at a time step $t$. In the source injection step, we define all sources as hard sources, which means the desired values of the source are directly written to

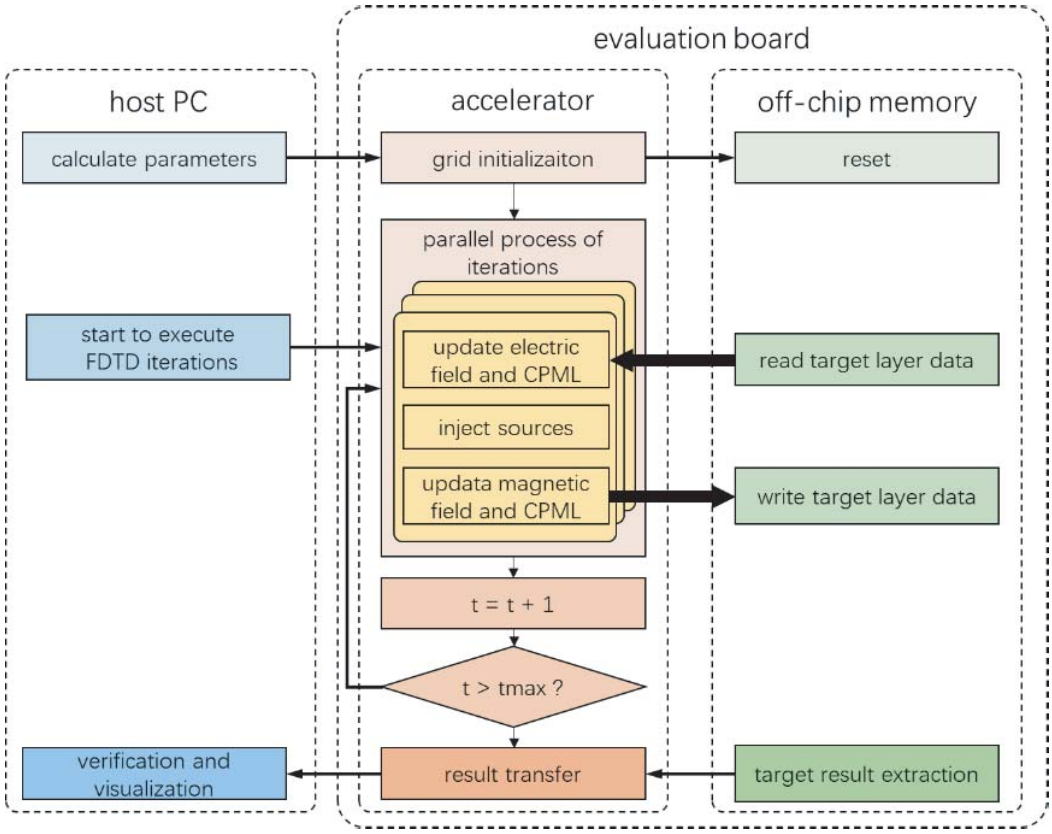



**Figure 1.** Flowchart of the FDTD computation process. The accelerator can process three iterations simultaneously. The host PC only takes charge of the initialization and result post-treatment but does not participate in any computation operation in the FDTD scheme.

the specific electric field component of a single or multiple target cells. Since calculating the source value at every time step is a one-time computation, this process is scheduled to be executed by the host PC according to the analytical equation of the source, and the computation results are sent to the accelerator in the initialization step. The last step is to calculate the magnetic field components, including $H_x$, $H_y$, and $H_z$ to complete an iteration. Several iterations are in process simultaneously by using the temporal parallel strategy, which is detailed in the Section 2.3.

In a large-scale or a high-resolution and fine-grained simulation model, the on-chip memory is usually insufficient for storing all electromagnetic data of the entire model. Therefore, in this architecture all data are stored in an off-chip memory, and the accelerator only caches partial data of the grid being calculated at the current time step. The data transfers between the accelerator and off-chip memory are interleaved in the whole computation process and parallel with the computation, so that the memory access time can be hidden in the computation time. This hardware architecture is equipped with two peripherals, double data rate 4 (DDR4) and universal asynchronous receiver transmitter (UART). DDR4 serves as the off-chip memory to store data. UART enables the host to transfer initialization information to the accelerator and receive the simulation results from it for verification and visualization.

## 2.2. 3D FDTD and CPML Boundary Condition Algorithms

The 3D FDTD algorithm consists of six formulas for updating the six electromagnetic field components, and the cells in the boundary layers require additional CPML computation of certain components. For example, the formulas for updating $E_x$ are given as:

$$E_x\Big|_{i,j,k}^{n+1} = C_a\Big|_{i,j,k} \cdot E_x\Big|_{i,j,k}^{n} + C_{b1}\Big|_{i,j,k} \cdot \left( H_z\Big|_{i,j+\frac{1}{2},k}^{n+\frac{1}{2}} - H_z\Big|_{i,j-\frac{1}{2},k}^{n+\frac{1}{2}} \right) + C_{b2}\Big|_{i,j,k} \cdot \left( H_y\Big|_{i,j,k+\frac{1}{2}}^{n+\frac{1}{2}} - H_y\Big|_{i,j,k-\frac{1}{2}}^{n+\frac{1}{2}} \right)$$

$$+C_{psi1}\Big|_{i,j,k} \cdot \Psi_{exz}\Big|_{i,j,k}^{n+\frac{1}{2}} + C_{psi2}\Big|_{i,j,k} \cdot \Psi_{exy}\Big|_{i,j,k}^{n+\frac{1}{2}} \tag{1}$$

where $C_a$, $C_{b1}$, $C_{b2}$, $C_{psi1}$, and $C_{psi2}$ denote the update coefficients determined by the material, time-step size, and space step length of the grid. $\Psi_{exz}$ and $\Psi_{exy}$ are CPML auxiliary terms, and they are updated as follows:

$$\Psi_{exz}\Big|_{i,j,k}^{n+\frac{1}{2}} = b_{ez} \cdot \Psi_{exz}\Big|_{i,j,k}^{n-\frac{1}{2}} + a_{ez} \cdot \left( H_y\Big|_{i,j,k+\frac{1}{2}}^{n+\frac{1}{2}} - H_y\Big|_{i,j,k-\frac{1}{2}}^{n+\frac{1}{2}} \right) \tag{2}$$

$$\Psi_{exy}\Big|_{i,j,k}^{n+\frac{1}{2}} = b_{ey} \cdot \Psi_{exy}\Big|_{i,j,k}^{n-\frac{1}{2}} + a_{ey} \cdot \left( H_z\Big|_{i,j+\frac{1}{2},k}^{n+\frac{1}{2}} - H_y\Big|_{i,j-\frac{1}{2},k}^{n+\frac{1}{2}} \right) \tag{3}$$

where $b_{ey}$, $b_{ez}$, $a_{ey}$, and $a_{ez}$ denote coefficients that depend on the position of the cells in the CPML boundaries. It is noticed that in the non-boundary domain, $\Psi_{exz}$ and $\Psi_{exy}$ are both zero. When updating the cells in the boundary layers in the $z$-direction, the value of $\Psi_{exz}$ is non-zero, and Equation (2) needs to be calculated. Similarly, Equation (3) is only valid on the boundary layers in the $y$-direction. Specially, in the overlay area of the $y$-direction and $z$-direction boundaries, two terms need to be calculated and added to $E_x$. The stencil for other electromagnetic field components operates in a similar way and must be controlled accordingly. This algorithm feature leads to extra calculation operations resulting to a larger processing time of the cells at the boundary area compared to that of other algorithms, and the accelerator needs to store an additional twelve CPML auxiliary terms compared to accelerators using other simple boundary conditions.

### 2.3. Processing Element Design Based on FDTD and CPML

Processing elements are basic hardware logic modules deployed to execute arithmetic operations. The goal of our PE design is to ensure that each PE integrates the function of processing FDTD and CPML algorithm so that every PE is capable of handling cells at any position. At the same time, we need to maintain the regularity of dataflow and the uniformity of total clock cycles when calculating the boundary conditions at low resources overhead. These two design principles can help reduce hardware complexity to integrate more PEs for higher parallelism. Here, we take the PE component module specialized for updating $E_x$ as an example. Such component modules are compatible with other electric and magnetic field components by simply changing the input data to the corresponding ones. Fig. 2 shows a schematic representation. In this component module, all cells in the simulation space require three clock cycles to calculate the main domain value, and the result is buffered. This part uses one adder, three subtractors, and three multipliers. When processing the cells at the $y$-direction or $z$-directions boundaries, the module enables the calculation of $\Psi_{exz}$ and $\Psi_{exy}$. This process is parallel to the computation of the main domain and also consumes three clock cycles. In stages 4–6, the updated CPML auxiliary terms are selectively accumulated to the $E_x$ according to the CPML mode signal, which is generated by the controller to denote the active terms. The $E_x$ value and auxiliary terms are eventually forwarded to the output after stage 6.

The PE component module operates in a particular data movement mode. The input data are set as a group and are validated every three clock cycles, as shown in Fig. 3. In the accelerator architecture, spatially adjacent data of each electromagnetic component are stored in the same buffer. These buffers are configured as true dual-port random access memory (RAM), which means two ports are available for reading and writing. Considering three electric field components together, the dependent data for their update include three datasets for each of the three magnetic field components and one dataset for each of their previous iteration value. Therefore, three clock cycles are required to fetch all dependent data in order to make full use of the bandwidth of magnetic buffer port A.

This data movement mode brings benefits. Since the $E_x$ buffer is only occupied for data fetch in two-thirds of the clock cycles, the updated results can be written back to the corresponding buffer in the certain idle cycle through the same port. In this way, port B can be released from the computation process and used for continuously loading data from the external memory to maximize the external memory bandwidth usage. In addition, this mode ensures that stages 1–3 of the pipeline are idle for two cycles after processing a group of valid data. The computing logic resources in the idle stages can be scheduled for reuse from stage 1 to stage 3 through reconfigurable interconnection logic. For

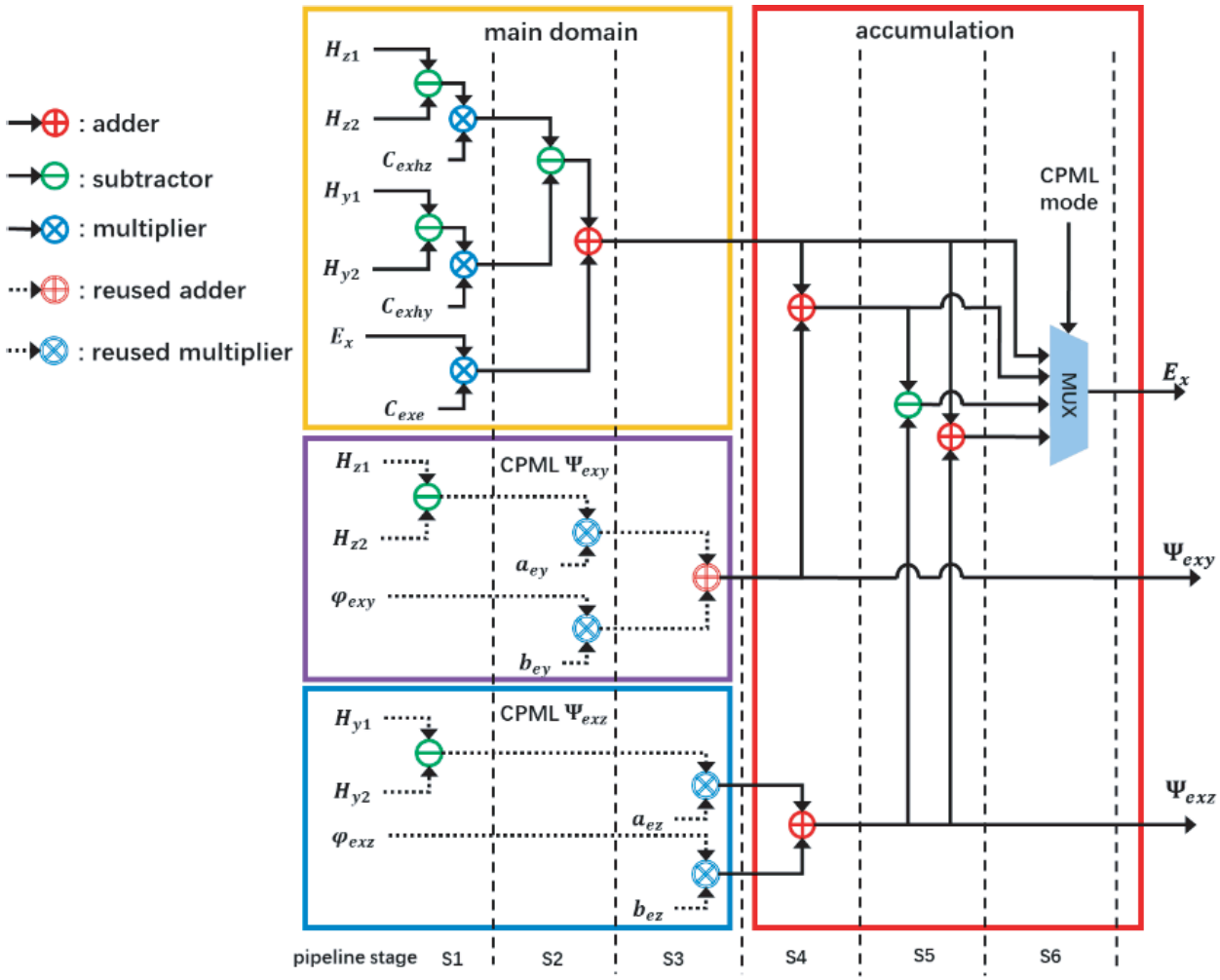



**Figure 2.** Schematic of the PE component module for processing $E_x$. The pipeline consists of six stages. The CPML mode signal in stage 6 is a 2-bit signal to denote four types of cell position occasions.

example, the multipliers of stage 1 and the adder of stage 2 in the main domain are reused in computing the CPML terms of other stages (dotted lines in Fig. 3). This technique does not require additional multipliers after implementing the CPML algorithm in the architecture. This advantage is embodied in the absence of extra digital signal processing (DSP) consumption during FPGA implementation.

The update coefficients for PE computation are grouped according to the material types in the simulation space. They are generated by the host and sent to the accelerator during initialization. These coefficient data are arranged as look-up tables and then stored in RAM. During fetching electromagnetic data from buffers, PE component modules simultaneously read the dependent coefficients according to the material type of the current cell. In most simulation applications, the total number of material types is limited. Hence, we can simply encode all material types and then read the coding information from the external memory instead of a large amount of repeated material coefficients to reduce the bandwidth stress. For example, if the model consists of four material types, we only require a 2-bit signal to encode every material. Compared to reading nine 32-bit coefficients of every material type from the external memory, the time cost of reading a 2-bit single can be nearly neglected. In addition, one material coefficient look-up table can be accessed by multiple PEs simultaneously by using a proper fan-out synthesis strategy in the electronic design automation tool. This method can also reduce memory usage.

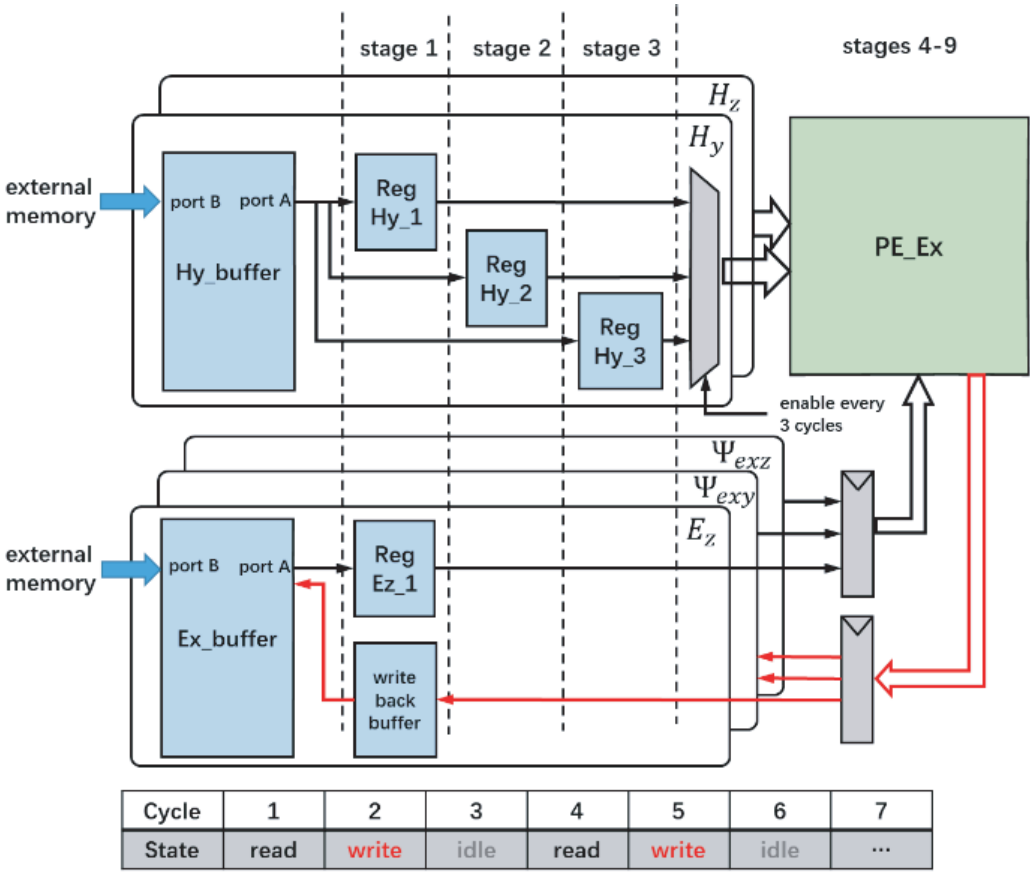



**Figure 3.** Schematic of the data input mode in the electric field update process. The data input modes of the electric field components and CPML terms are consistent with each other. In the magnetic field update process, the input modes of the electric and magnetic fields are interchangeable, but those of the CPML terms remain the same.

### 2.4. Chain PEs Array with Buffer Banks

Hardware accelerators usually integrate multiple PEs to improve the computation parallelism. The architecture of PEs array directly affects the operation load of each PE and the hardware complexity, which eventually affects the overall performance. In this design, we propose a PEs arrangement structure, called chain PEs array, to balance the performance and hardware complexity. The features of the chain PEs array are reflected in the interconnection between PEs, data path, and memory mapping of the dependent grid data. Fig. 4 shows the framework of the chain PEs array, which emphasizes the interconnection between adjacent PEs. In this architecture, the array integrates sixteen PEs, and each of them consists of three PE component modules for updating the electric or magnetic fields. Each PE is assigned a unique connected buffer bank, which contains a set of RAM-based buffers for storing electromagnetic data and CPML auxiliary terms. Every PE only exchanges data with its two adjacent PEs and is isolated from others. The electric field data in the $n$th element, $\mathrm{PE}_n$, are transferred to $\mathrm{PE}_{n-1}$ through a group of right shifters, and the magnetic field data in $\mathrm{PE}_n$ are transferred to $\mathrm{PE}_{n+1}$ through left shifters. $\mathrm{PE}_0$ and $\mathrm{PE}_{15}$ are connected end-to-end following the data exchange pattern above.

This data path design is inspired by the spatial data dependence embedded in the FDTD algorithm. The six formulas in the FDTD indicate that each cell requires some specific data in six adjacent cells in the $x$, $y$, and $z$-directions to finish computation of all six electromagnetic components. In other words, one cell has data correlation with others in three dimensions. In this regard, our accelerator adopts a unique data rearrangement method to try to decompose this correlation in order to reduce the data

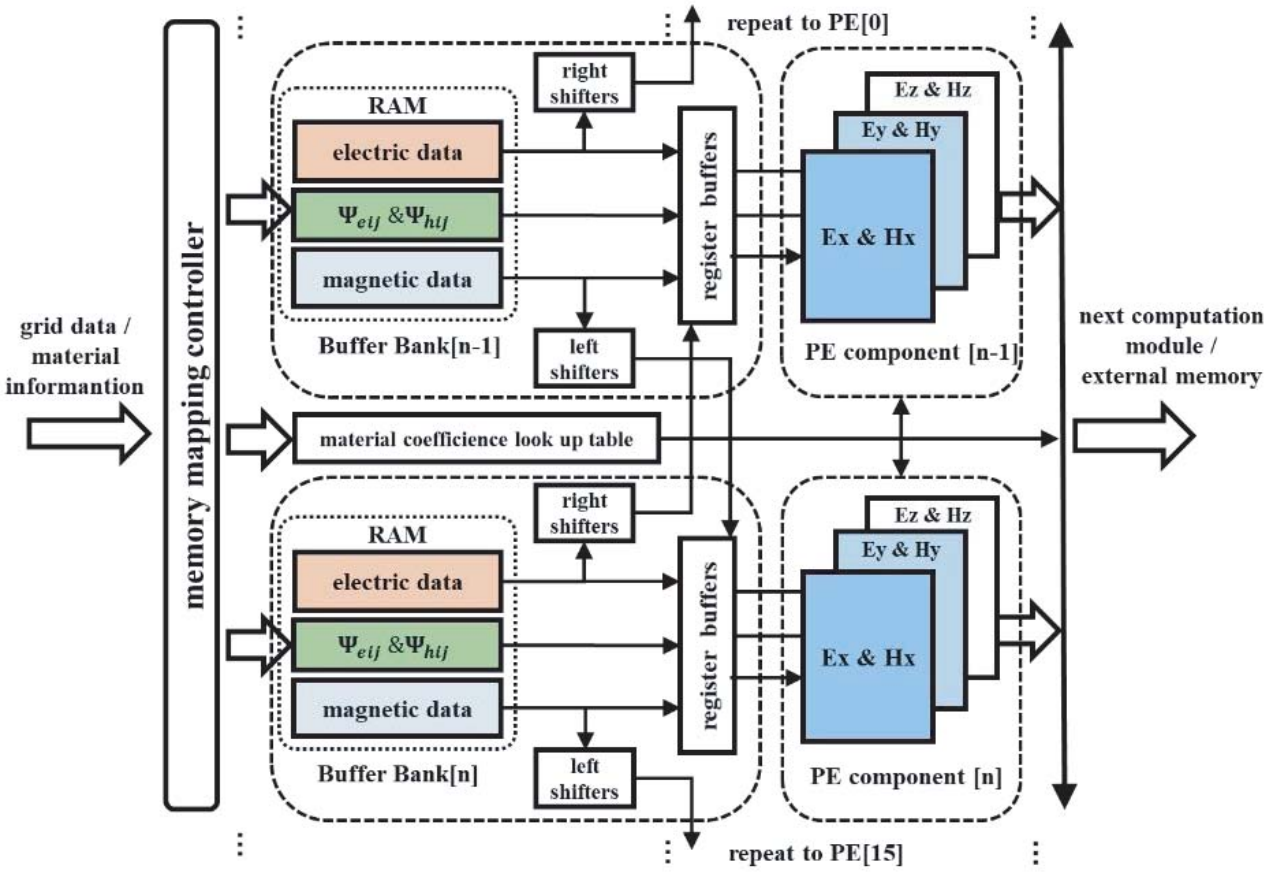



**Figure 4.** Framework of the chain PEs array. It consists of sixteen PEs. Shifters refer to several shift registers, which require one clock cycle to send a group with 32-bit data to other PEs.

exchange between PEs. Fig. 5(a) illustrates this method taking a four-PE chain array as an example. First, the whole simulation grid is sliced along the $z$-direction into many two-dimensional layers. The accelerator reads data from the external memory in layers. Then, planar cells in the layer are expanded along the $x$-direction into a one-dimensional chain-like grid stream and stored in buffer banks 0–3 in the order of increasing $x$ coordinates. When the current cell reaches the $x$-direction boundary, the next cell switches to the next $y$-line, and its data is stored from buffer bank 0 until all cells in the layer have been buffered. Using this pattern, the dependent data in the $y$ and $z$-directions of the current cell are stored in the same buffer bank, so they can be fetched along with the current cell data. Only the dependent data in the $x$-direction are stored in the adjacent two PEs and need an additional data path to be accessed. Such a pattern can save considerable routing resources compared to conventional 3D PEs arrays, which require interconnection with six PEs adjacent in three directions.

Figure 5(b) reveals the cell scanning mode by taking a four-PE chain array scanning a $6 \times 6$ layer as an example. The array scans along the $x$-direction of the grid and updates four cells every six clock cycles. When the $x$-direction boundary is encountered by any PE, the PEs beyond the boundary are disabled temporarily. After the boundary computation is completed, the array switches to the next $y$-line and repeats this pattern until the whole layer is updated. Using this scanning pattern, the continuity of the PEs array operation is not affected by any grid size in the $y$ or $z$-directions. Partial PEs are only paused intermittently when the $x$-direction grid size is not divisible by the number of PEs in the implemented array. Moreover, when the grid size in the $x$-direction increases, the boundary computation becomes a small part of the whole update process. Thus, the proportion of the paused time of disable PEs gradually decreases and the overall performance is improved.

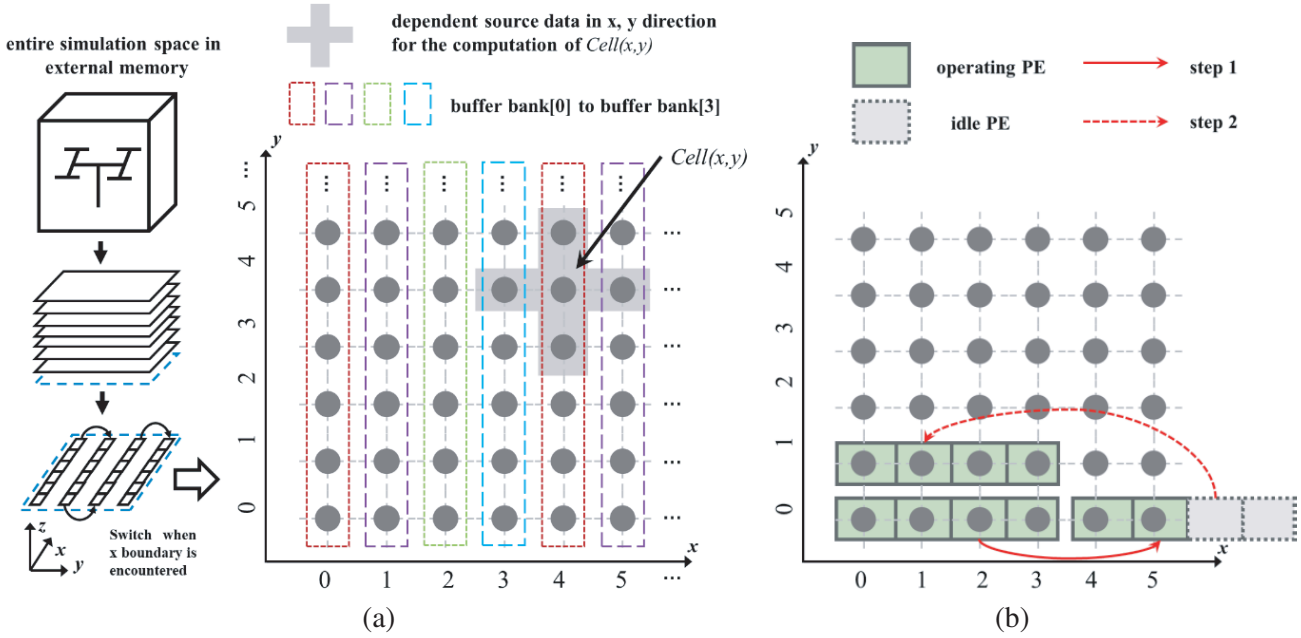


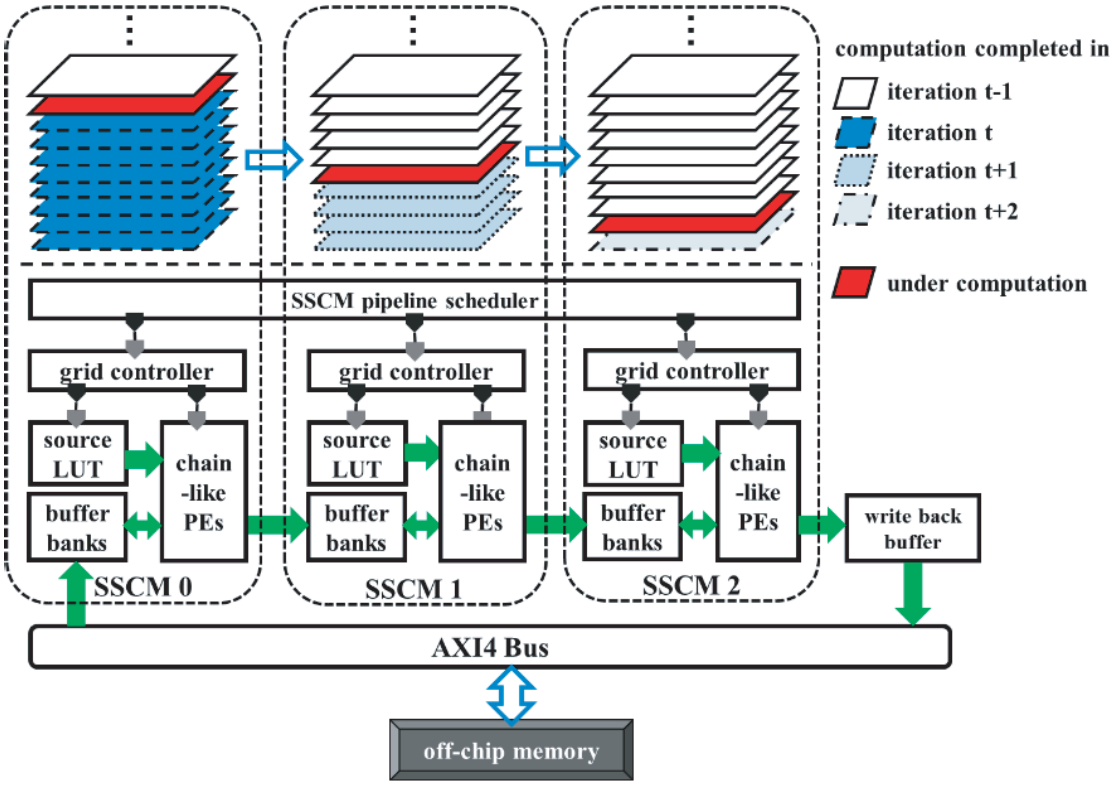

**Figure 5.** Memory mapping and corresponding computation mode of the chain PEs array taking a four-PE array as an example. (a) Data rearrangement method to transform a 3D grid to a 1D grid stream. (b) Cell scanning mode. The size of the layer is $6 \times 6$.

The memory mapping pattern and scanning mode proposed can be actually summarized by reconstructing a 3D grid into a 1D grid stream, and this method is applicable to 3D grids of any size. Therefore, we only need to configure some parameters related to the size of the grid in the $x$-direction during the initialization step; the accelerator can handle any 3D electromagnetic model without modifying the hardware architecture.

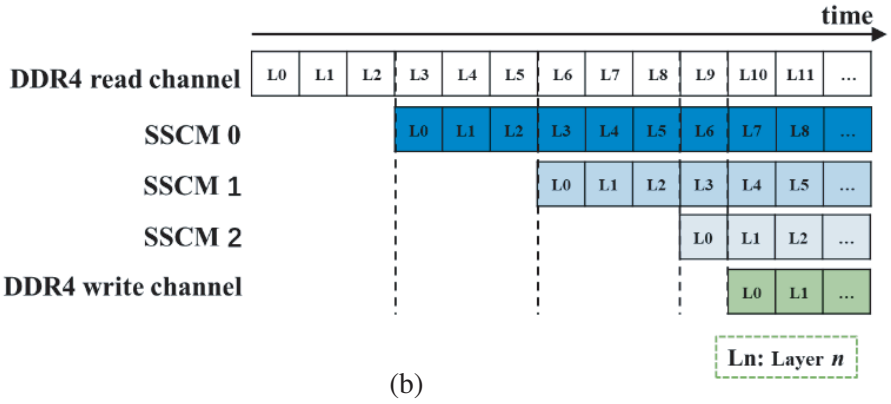
### 2.5. Temporal Parallelism Strategy

The PEs array exploits the spatial parallelism property of the algorithms, and the temporal parallelism strategy can further utilize the parallelism potential between iterations. According to the formulas, as long as some adjacent layers have completed their update in iteration $t$, the cells inside these layers are immediately available for their next computation in iteration $t+1$, even if the update of other points in the current time step is not yet completed. Therefore, if we can ensure that the grid domains in each iteration do not overlap with each other, it is possible to simultaneously process several iterations at different time steps. In the proposed hardware architecture, several modules that can independently complete the FDTD and CPML scheme are packed up as a single step computation module (SSCM). The SSCM includes PEs array, buffer banks, and other modules for control. Several SSCMs are connected in series and pipelined. Fig. 6(a) shows a schematic representation of the three-stage SSCMs pipeline used in this study. It should be noted that the data capacity of each buffer bank is set to simultaneously store four layers. This is because the update of each cell uses the data of three layers, including the upper, lower, and current layers. Moreover, the fourth layer serves as the read-write buffer to enable the next layer data to be pre-read from the external memory when the data of the other three layers are being fetched by the PEs for computation.

During the FDTD process, after $\text{SSCM}_n$ finishes the update of a layer in iteration $t$, the updated data are written to the buffer bank in the current SSCM stage and the buffer bank in $\text{SSCM}_{n+1}$. The source injection and material distribution information are also streamed to $\text{SSCM}_{n+1}$ to avoid repeated reading from the external memory. When three adjacent layers, layer $k-1$, $k$, and $k+1$, have been updated in $\text{SSCM}_0$, $\text{SSCM}_1$ accumulates all dependent data for updating layer $k$ in iteration $t$+1 and starts the computation in parallel to $\text{SSCM}_0$. Following this principle, when six adjacent layers have




(a)




(b)

**Figure 6.** Schematic of the three-stage SSCM pipeline. (a) Dataflow of layers between SSCM stages. (b) Sequence diagram of data transfer among each SSCM stage and DDR4.

been updated in $SSCM_0$, the three-stage SSCM pipeline operates at full capacity to process the cells in three iterations until the simulation ends.

Additionally, this strategy brings two benefits in terms of memory access. $SSCM_1$ and $SSCM_2$ can also serve as buffers for the computation results acquired from $SSCM_0$ before they are streamed to the DDR4 write channel, as shown in Fig. 6(b). These buffers have sufficient capacity and ensure the data in the write channel are independent of the data in the current read channel. Therefore, the write channel can operate continuously without waiting for the data from $SSCM_0$. Another benefit is that the original data in iteration $t$ fetched from the external memory go through three updates in iteration $t+3$ before being written. Hence, considering a simulation that requires $T$ iterations in total, only $T/3$ data transfers between the accelerator and external memory are required.

Since the hardware structure of each stage in the SSCM pipeline is identical, according to the stencil, the stage number of the SSCM pipeline can be modified by simply duplicating the SSCM based on resources of the target hardware platform, and not extra input or output ports are required. Moreover, the size of the grid model in each SSCM pipeline stage is identical, so the latency in each stage is the same and causes no pipeline stall. Therefore, increasing the number of SSCM stages can linearly improve the overall performance and reduce the memory access in inverse proportion. The only cost of having too many SSCM stages is the slight increase in the complexity of the SSCM pipeline scheduler. However, the design scale of the pipeline scheduler is far smaller than that of the SSCM, so the cost is negligible. Owing to these advantages, this temporal parallel strategy is quite suitable for implementing on multiple FPGAs or many-core processing platforms. Possessing the advantages of abundant of logic resource, on-chip memory storage, and sufficient port number for memory access, the proposed architecture can achieve good simulation speedup.

## 3. EXPERIMENTAL RESULTS AND DISCUSSION

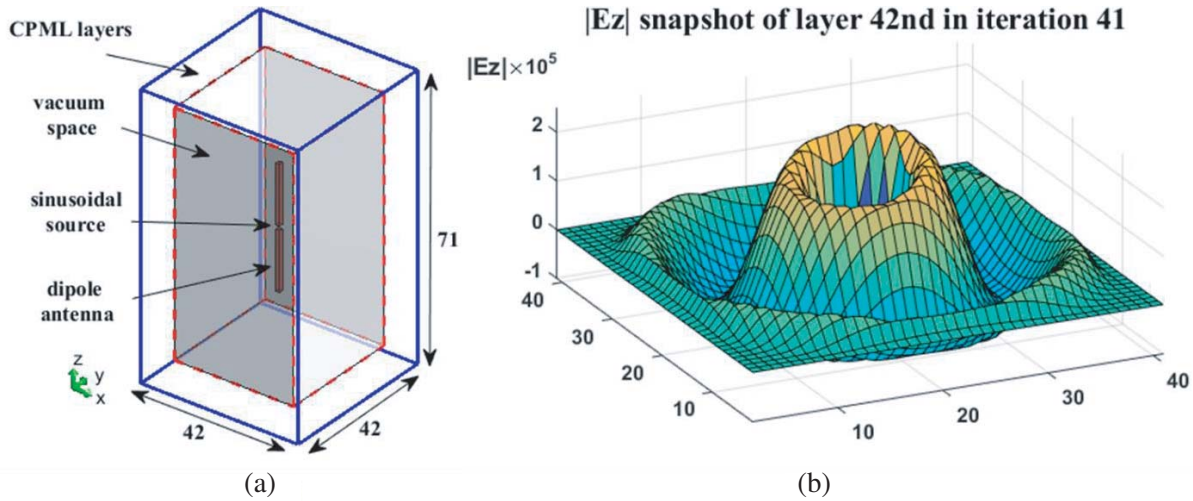
To quantitatively evaluate the functionality and performance of the proposed architecture, we completed our accelerator and deployed it on the Xilinx KCU105 evaluation board. The evaluation board was equipped with an xcku040-2ffva1156e device and 64-bit DDR4, providing a maximum memory bandwidth of 19.2 GB/s. The accelerator was designed using Verilog to accurately control the underlying hardware. We used Xilinx Vivado 2018.1 for synthesis and implementation on the FPGA. Table 1 describes the resource utilization of our accelerator in this FPGA device. It indicates that albeit the lookup table (LUT) and block RAM (BRAM) utilization reaches 80% or even higher, our accelerator can run in the clock frequency of 100 mega Hz (MHz) with a sufficient setup time margin. This occurs owing to the fine optimization of interconnection and data path inside and between modules. It significantly reduces the difficulty of placing and routing, resulting in lower fan-out, shorter path delay, and higher clock frequency.

**Table 1.** Resources utilization of the accelerator in the target FPGA device.

| Device | Xilinx Kintex UltraScale xcku040-2ffva1156e |
|---|---|
| LUTs | 193944/242400 (80%) |
| Registers | 200307/484800 (41%) |
| BRAMs | 557.5/583 (95%) |
| DSP Slices | 898/1920 (46%) |

To verify the functionality of the architecture, a typical dipole antenna application was simulated in our accelerator. The simulation model is shown in Fig. 7(a). The simulation space is in vacuum with $42 \times 42 \times 42$ cells in the $x$, $y$, and $z$-directions, respectively. The dipole antenna consists of two arms, each of them is modeled with $2 \times 2 \times 30$ cells and considered as perfect electric conductor. A sinusoidal source is assigned in the gap between two bars, and the size of the gap is $2 \times 2 \times 2$. We set 8-cell-thick CPMLs at all six boundaries of the simulation space. Fig. 7(b) shows the snapshot of the $E_z$ distribution in layer 42 after 41 iterations. The sine wave can be effectively absorbed by the CPMLs and hardly cause reflections. The application ran 1000 time steps in total on the accelerator and took 0.22 s to solve this problem. This performance is approximately 33 times higher than that of i7-8700 CPU implementation, which took 7.24 s to complete the entire simulation. All the electromagnetic data computed in the accelerator are sent back to the host PC through UART. These data are compared to the computation results obtained by the reference FDTD algorithm running on MATLAB to ensure the simulation result of every cell is correct and accurate.

In the performance test, we first evaluated the impact of various grid sizes in three dimensions on the performance of the accelerator. We scanned the number of the cells from 32 to 80 in one direction while fixing two other cell numbers to 32. The results depicted in Fig. 8(a) show that the peak performance can reach 608 Mcells/s, and it is hardly affected by the size of the grid in the $y$ or $z$-directions. This speedup is about 6 times higher than that of the FPGA-based accelerator in [26]. The size variation in

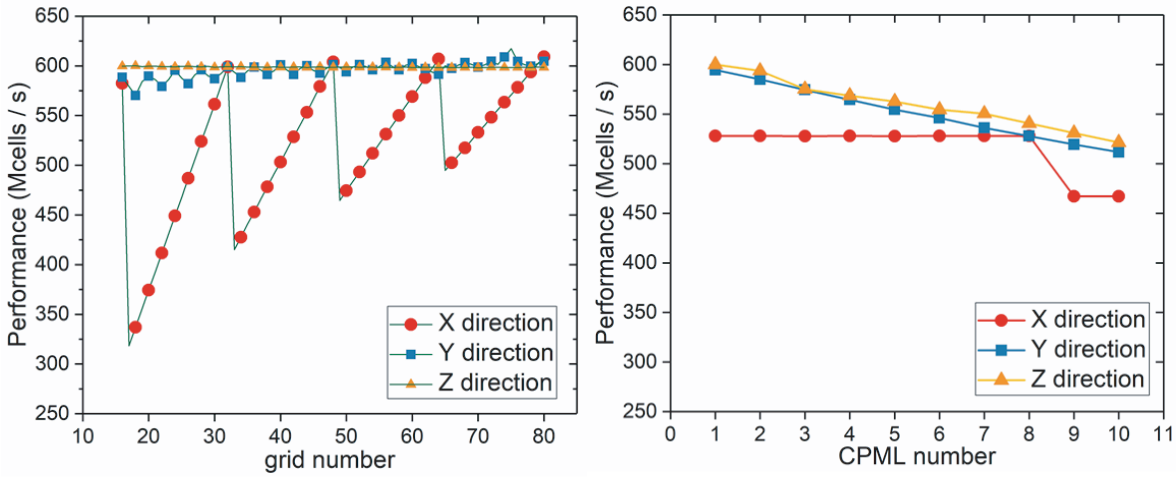



(a) (b)

**Figure 7.** A simulation application of a dipole antenna. (a) Geometry details, CPML condition, and source injection of the model. (b) $E_z$ snapshot of the simulation result.

the $x$-direction causes performance fluctuation. When the grid size in the $x$-direction is divisible by 16, the performance reaches its maximum, and the fluctuation gradually attenuates when the grid size in the $x$-direction increases. This fluctuation pattern is consistent with the chain PEs array feature stated in Section 2.4. In addition, we tested the effect of different numbers of CPMLs on the performance. In a simulation model with $32 \times 32 \times 32$ cells, we fixed the number of CPMLs in two directions to 0 and scanned the number of CPMLs in the other direction from 0 to 10, which is a commonly used range in practical applications. The test results shown in Fig. 8(b) indicate that the performance degrades when the CPML layers increases. This is because there are three times more data in the boundary cell than the non-boundary domain bottleneck rather than the computation efficiency of the PEs array. Luckily, this negative effect can also be reduced in large simulation spaces, where the boundary comprises a small part of the whole grid. Nevertheless, in general, the performance of the accelerator can still be maintained above 467 Mcells/s.

Table 2 provides further performance comparisons with two previous works in [26, 27]. The two




(a) (b)

**Figure 8.** Performance test results for different grid sizes and CPML numbers in three directions. (a) Test results for different grid sizes. (b) Test results for different CPMLs numbers.

accelerators reported in these works also implement the same CPML boundary condition algorithm in the 3D FDTD scheme. In the digital circuit, the clock frequency determines one owing to the extra CPML auxiliary term. This results in a data-transfer hot spot between the accelerator and external memory and costs more time. Further, the memory bandwidth becomes the how many clock cycles occur in a certain time, and the memory bandwidth determines the maximum data transfer in units of time between the circuit and the memory chip. These two properties are essential to the overall performance of accelerators, but they are usually determined by the manufacture process, chip area, bus protocol, and other factors unrelated to the hardware architecture. Therefore, performance comparison under the constant clock frequency and memory bandwidth is more reasonable to reveal the advantages of the proposed architecture. Generally, GPUs integrate much more logic, memory resources, and higher bandwidth than FPGAs. Compared to GPU-based acceleration in [27], we can still achieve a 40% peak performance speedup, and our performance under the same bandwidth or clock frequency is further superior. It shows that our chain PEs array design and temporal parallel strategy greatly improve bandwidth utilization. Table 2 also indicates that dedicated computation unit and data path design in out FPGA-based accelerators can effectively improve the computing density in a clock cycle. The comparison results show that our architecture is suitable for future implementation on ASIC devices. After being deployed on the customized circuit, the accelerator can be applied to scientific research and the industry.

**Table 2.** Detailed performance comparison between the proposed architecture and that of two related works.

| Method | Work in [27] | Work in [26] | Proposed |
|---|---|---|---|
| Boundary condition | CPML | CPML | CPML |
| Hardware platform | GPU | FPGA | FPGA |
| Device | NVIDIA Tesla C1060 | Xilinx Virtex-6 XC6VSX475T $\times 4$ | Xilinx Kintex UltraScale xcku040-2ffva1156e |
| Clock Frequency (MHz) | 1300 | 100 | 100 |
| Maximum Bandwidth (GB/s) | 102 | 38.6 | 19.2 |
| Peak Performance (Mcells/s) | 446 | 100 | 608 |
| Performance/Bandwidth (Mcells/GB) | 4.4 | 2.6 | 31.7 |
| Performance/Clock frequency (Mcells/s $\cdot$ MHz) | 0.3 | 1.0 | 6.1 |

## 4. CONCLUSION

In this paper, we proposed a hardware architecture that implemented the 3D FDTD and CPML algorithms on an FPGA. We designed a dedicated PE module compatible with the two algorithms and obtained a regular dataflow in a low logic resource cost. A novel chain PEs array structure was proposed to integrate multiple PEs to improve parallelism. It also reduced the complexity of the data path and data transfer pattern between PEs to simplify the wiring. We also applied a temporal parallel strategy to build an FDTD computation pipeline, which can simultaneously process multiple time steps to further improve performance. In the conducted experiment, the proposed architecture achieved a peak performance of 608 Mcells/s on the Xilinx KCU105 platform, which is 6 times and 40% higher than the performance of the reported FPGA-based and GPU-based accelerators, respectively. The performance can still be maintained above 467 Mcells/s for different grid sizes and CPMLs, which proves the performance stability of the architecture in various simulation applications. Further performance comparison revealed that the hardware architecture could be implemented on the ASIC platform to achieve higher simulation efficiency and make the accelerator more practical for real-world applications.

## ACKNOWLEDGMENT

This work was supported in part by the Science and Technology Program of Guangdong Province under grant 2017B090909005 and 2019B010140002.

## REFERENCES

1. Yee, K. S., "Numerical solution of initial boundary value problems involving Maxwell's Equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, Vol. 14, 302–307, 1966.
2. Jensen, M. A. and Y. Rahmat-Samii, "Performance analysis of antennas for hand-held transceivers using FDTD," *IEEE Transactions on Antennas and Propagation*, Vol. 42, No. 8, 1106–1113, 1994.
3. Orjubin, G., F. Petit, E. Richalot, S. Mengue, and O. Picon, "Cavity losses modeling using lossless FDTD method," *IEEE Transactions on Electromagnetic Compatibility*, Vol. 48, No. 2, 429–431, 2006.
4. Ziolkowski, R. W., "The incorporation of microscopic material models into the FDTD approach for ultrafast optical pulse simulations," *IEEE Transactions on Antennas and Propagation*, Vol. 45, No. 3, 375–391, 1997.
5. Wang, X., W. Yin, Y. Yu, Z. Chen, J. Wang, and Y. Guo, "A Convolutional Perfect Matched Layer (CPML) for one-step leapfrog ADI-FDTD method and its applications to EMC problems," *IEEE Transactions on Electromagnetic Compatibility*, Vol. 54, No. 5, 1066–1076, 2012.
6. Mukherjee, B. and D. K. Vishwakarma, "Application of finite difference time domain to calculate the transmission coefficient of an electromagnetic wave impinging perpendicularly on a dielectric interface with modified MUR-I ABC," *Defence Science Journal*, Vol. 62, 228–235, 2012.
7. Sypek, P., A. Dziekonski, and M. Mrozowski, "How to render FDTD computations more effective using a graphics accelerator," *IEEE Transactions on Magnetics*, Vol. 45, No. 3, 1324–1327, 2009.
8. Zygiridis, T. T., "High-order error-optimized FDTD algorithm with GPU implementation," *IEEE Transactions on Magnetics*, Vol. 49, No. 5, 1809–1812, 2013.
9. Cicuttin, M., L. Codecasa, B. Kapidani, R. Specogna, and F. Trevisan, "GPU accelerated time-domain discrete geometric approach method for Maxwell's Equations on tetrahedral grids," *IEEE Transactions on Magnetics*, Vol. 54, No. 3, 1–4, 2018.
10. Livesey, M., J. F. Stack, F. Costen, T. Nanri, N. Nakashima, and S. Fujino, "Development of a CUDA implementation of the 3D FDTD method," *IEEE Antennas and Propagation Magazine*, Vol. 54, No. 5, 186–195, 2012.
11. Jia, C., L. Guo, and P. Yang, "EM scattering from a target above a 1-D randomly rough sea surface using GPU-based parallel FDTD," *IEEE Antennas and Wireless Propagation Letters*, Vol. 14, 217–220, 2015.
12. Lee, K. H., I. Ahmed, R. S. M. Goh, E. H. Khoo, E. P. Li, and T. G. G. Hung, "Implementation of the FDTD method based on Lorentz-Drude dispersive model on GPU for plasmonics applications," *Progress In Electromagnetics Research*, Vol. 116, 441–456, 2011.
13. Ghouwayel, A. A. and Y. Louet, "FPGA implementation of a re-configurable FFT for multi-standard systems in software radio context," *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 2, 950–958, 2009.
14. Ingemarsson, C., P. Källström, F. Qureshi, and O. Gustafsson, "Efficient FPGA mapping of pipeline SDF FFT cores," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 25, No. 9, 2486–2497, 2017.
15. Choi, S., et al., "Design of FPGA-based LZ77 compressor with runtime configurable compression ratio and throughput," *IEEE Access*, Vol. 7, 149583–149594, 2019.
16. Li, B., L. Zhang, Z. Shang, and Q. Dong, "Implementation of LZMA compression algorithm on FPGA," *Electronics Letters*, Vol. 50, No. 21, 1522–1524, 2014.
17. Nguyen, D. T., T. N. Nguyen, H. Kim, and H. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 27, No. 8, 1861–1873, 2019.

18. Guo, K., et al., "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 37, No. 1, 35–47, 2018.
19. Fujita, Y. and H. Kawaguchi, "Full-custom PCB implementation of the FDTD/FIT dedicated computer," *IEEE Transactions on Magnetics*, Vol. 45, No. 3, 1100–1103, 2009.
20. Okina, K., R. Soejima, K. Fukumoto, Y. Shibata, and K. Oguri, "Power performance profiling of 3-D stencil computation on an FPGA accelerator for efficient pipeline optimization," *SIGARCH Comput. Archit. News*, Vol. 43, No. 4, 9–14, 2015.
21. Sano, K., Y. Hatsuda, and S. Yamamoto, "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 3, 695–705, 2014.
22. Kawaguchi, H. and S. Matsuoka, "Conceptual design of 3-D FDTD dedicated computer with dataflow architecture for high performance microwave simulation," *IEEE Transactions on Magnetics*, Vol. 51, No. 3, Art No. 7202404, 2015.
23. Kawaguchi, H., "Improved architecture of FDTD dataflow machine for higher performance electromagnetic wave simulation," *IEEE Transactions on Magnetics*, Vol. 52, No. 3, Art No. 7206604, 2016.
24. Waidyasooriya, H. M., Y. Takei, S. Tatsumi, and M. Hariyama, "Open CL-based FPGA-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 28, No. 5, 1390–1402, 2017.
25. Roden, J. A. and S. D. Gedney, "Convolution PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media," *Microw. Opt. Technol. Lett.*, Vol. 27, 334–339, 2000.
26. Giefers, H., C. Plessl, and J. Förstner, "Accelerating finite difference time domain simulations with reconfigurable dataflow computers," *SIGARCH Comput. Archit. News*, Vol. 41, No. 5, 65–70, 2014.
27. Toivanen, I., T. P. Stefanski, N. Kuster, and N. Chavanne, "Comparison of CPML implementations for the GPU-accelerated FDTD solver," *Progress In Electromagnetics Research B*, Vol. 19, 61–75, 2011.