FPGA-Based Deep-Pipelined Architecture for FDTD Acceleration Using OpenCL

Hasitha Muthumala Waidyasooriya and Masanori Hariyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi,980-8579, Japan
Email: {hasitha, hariyama}@tohoku.ac.jp

***Abstract*—Acceleration of the FDTD (finite-difference time-domain) computation is very important for the electromagnetic simulations. Conventional FDTD acceleration methods using multicore CPUs and GPUs have the common problem of memory-bandwidth limitation due to a large amount of parallel data access. Although FPGAs have the potential to solve this problem, very long design, testing and debugging time is required to implement an architecture successfully. To solve this problem, we propose an FPGA architecture designed using C-like programming language called OpenCL (open computing language). Therefore, the design time is very small and extensive knowledge about hardware-design is not required. We implemented the proposed architecture on an FPGA and achieved over 114 GFLOPS of processing power. We also achieved more than 13 times and 4 times speed-up compared to CPU and GPU implementations respectively.**

***Index Terms*—OpenCL for FPGA, FDTD, stencil computation, accelerator.**

I. Introduction

The finite difference time domain (FDTD) method [1] is widely used in electromagnetic simulations. However, the computation requires a large amount of processing time. On the other hand, FDTD computation is a massively parallel application. Therefore, parallel processing hardware such as multicore processors, GPUs and FPGAs are already used to accelerate the FDTD computation. Fig.1 shows the flow-chart of the FDTD computation. It is an iterative computation method where external memory is accessed in every iteration. Therefore, the computation speed is usually decided by the external memory bandwidth. Even many acceleration techniques such as tiling [2], spatial blocking, temporal blocking [3] have been proposed, the performance of the multicore CPU and GPU accelerations are eventually restricted by the memory bandwidth bottleneck. Compared to CPUs and GPUs, FPGAs contain a large amount of internal memory and registers. Therefore, it is possible to utilize those resources to reduce the external memory access. In fact, works [4], [5] done on other stencil computations such as Jacobi-method report better performance compared to GPUs even the external memory bandwidth of the FPGAs is very low.

There are many FPGA implementations proposed to accelerate 2-D and 3-D FDTD computations. The work in [6] uses parallel processing similar to GPUs to accelerate the FDTD computation. The works in [7] and [8] propose a pipelined FPGA architecture to increase the data sharing. Fixed-point computation is used in [9] and [10] to increase the amount of parallel processing in FPGAs. However, recent FPGAs contain enough DSP units (to do floating-point multiplication) so that floating-point computation is not a problem.

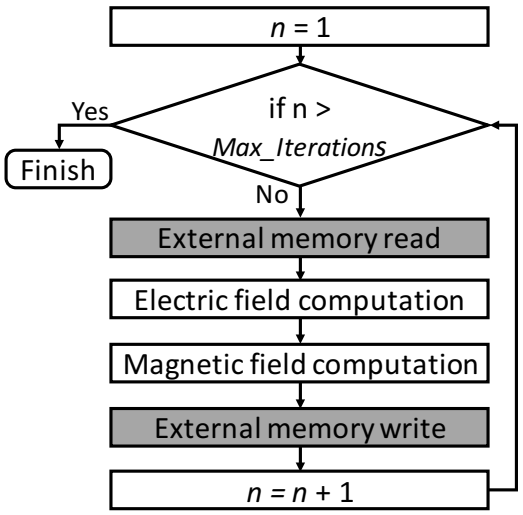


Fig. 1. Flow-chart of the FDTD computation.

Although the results are promising, FPGA accelerators are designed using hardware description languages such as Verilog HDL or VHDL. As a result, the designing, testing and debugging time is very large. Extensive knowledge about the hardware design is also required to use FPGAs. Moreover, we have to re-design the FPGA architecture when an algorithm change or an update occurs. To overcome these problems, OpenCL-based FPGA design has been introduced [11]. It is a complete framework that includes firmware, software and devise drivers to connect, control and transfer data to and from the FPGA. It supports different FPGA boards by the means of a BSP (board support package) that contains pre-designed hardware controllers and interconnects. Recently, some works such as [12] and [13] propose OpenCL-based FPGA accelerators. However, none of those works are about FDTD computation. One rare attempt in [14] to accelerate FDTD computation using OpenCL-based design was not that successful due to very low performance.

In this paper, we propose an OpenCL-based FPGA accelera-

978-1-5090-0806-3/16/$31.00 copyright 2016 IEEE
ICIS 2016, June 26-29, 2016, Okayama, Japan

tor for FDTD computation. The design concept is quite similar to [4] and [5], where deep-pipelines with few ten thousands of stages are used to transfer and re-use the computed results internally without accessing the external memory. Unlike previous works, both the accelerator and host program are designed entirely using software. Therefore, the design time has been reduced to just few hours. Any algorithmic change could be easily implemented by just changing the software code. Moreover, the same code can be re-used in OpenCL capable board so that the proposed method support any future hardware or software update. In fact, we used almost the same software code in two different FPGA boards to generate FDTD accelerators. According to the experimental results, we easily achieved over 13 times better performance compared to multicore CPUs and 4 times better performance compared to GPUs. We also achieved over 58% of the theoretical maximum performance provided by the FPGA.

II. OpenCL-based FPGA architecture for FDTD computation

A. OpenCL-based design environment

OpenCL is a framework to write programs to execute across heterogeneous parallel platforms [15]. Such systems consist of a host CPU and OpenCL capable devices such as multicore CPUs, GPUs, FPGAs, etc. Kernels are the functions that are executed on an OpenCL device. The unit of the concurrent execution of a kernel is called a work item. OpenCL for FPGA classifies kernels in to two types, NDrange kernels and single work item kernels. The processing in the NDrange kernels are similar to the SIMD processing used by GPUs. This is very effective when the work items are completely independent. However, when there are data dependencies, users have to explicitly insert barriers at different stages to synchronize. This synchronization mechanism costs a lot of hardware resources and decreases the performance. Therefore, it is recommended to use single work item kernels, so that the barriers are not required. The parallelism is achieved by designing deep-pipelines with many thousands of stages. The "#pragma unroll" directive is used to unroll the inner-loops to avoid pipeline stalls. Conditional branches are processed in parallel by assigning separate processing elements for all conditions. This is a major difference compared to CPU or GPU based implementations.

B. Data dependencies and parallelism in the FDTD computation

FDTD belongs to the class of stencil computation where the computation is done in multiple iterations using a grid. To compute one iteration, the data of its previous iteration are required. Therefore, the data dependencies exist among multiple iterations. The electric and magnetic fields computations of 2-D FDTD are given by Eqs.(1), (2) and (3). As shown in Eqs.(2) and (3), not only the magnetic field values of the previous iteration, but also the electric field values of the same iteration are required for the magnetic field computation. Therefore, even in the same iteration, magnetic field computation depends on the data of the electric field computation.

$$Ez^{t}_{i,j} = Ez^{t-1}_{i,j} - C1_{i,j}.\left(Hx^{t-\frac{1}{2}}_{i,j+\frac{1}{2}} - Hx^{t-\frac{1}{2}}_{i,j-\frac{1}{2}}\right) + C2_{i,j}.\left(Hy^{t-\frac{1}{2}}_{i+\frac{1}{2},j} - Hy^{t-\frac{1}{2}}_{i-\frac{1}{2},j}\right) \quad (1)$$

$$Hx^{t+\frac{1}{2}}_{i,j+\frac{1}{2}} = Hx^{t-\frac{1}{2}}_{i,j+\frac{1}{2}} - C3_{i,j}.\left(Ez^{t}_{i,j+1} - Ez^{t}_{i,j}\right) \quad (2)$$

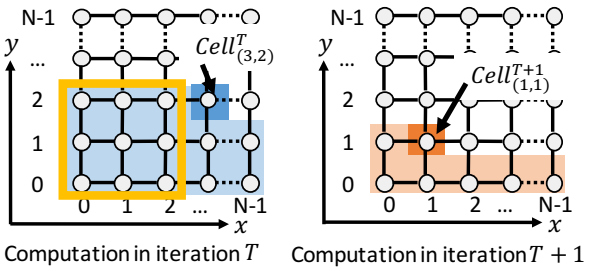
$$Hy^{t+\frac{1}{2}}_{i+\frac{1}{2},j} = Hy^{t-\frac{1}{2}}_{i+\frac{1}{2},j} - C4_{i,j}.\left(Ez^{t}_{i+1,j} - Ez^{t}_{i,j}\right) \quad (3)$$

If we consider one iteration and either electric or magnetic field, the computations of different cells (grid-points) are completely independent. Those computations can be done in parallel and we call it "cell-parallel" computation. There is another parallelism exists among multiple iterations and we explain it using Fig.2. For the simplicity, let us consider a square-shaped 3×3 stencil. As shown in Fig.2(a), to compute $Cell^{T+1}_{1,1}$ in iteration $T+1$, data of its surrounding cells belongs to iteration T are required. When the computation of $Cell^{T}_{3,2}$ is in progress in iteration T, all the data required for the computation of $Cell^{T+1}_{1,1}$ are available. Therefore, the computations of $Cell^{T}_{3,2}$ and $Cell^{T+1}_{1,1}$ are done in parallel. In the next step in Fig.2(b), computations of $Cell^{T}_{4,2}$ and $Cell^{T+1}_{2,1}$ are done in parallel. This type of computation is called "cell-serial iteration-parallel" computation. In order to successfully implement this method, we have to store the computed data temperately until those are accessed in the next iteration. The lifetime of these data are usually not long and we can reuse the same storage space for different data. Therefore, we need to store only a small portion of the data belong to an iteration. However, to compute more iterations in parallel, more storage space is required.

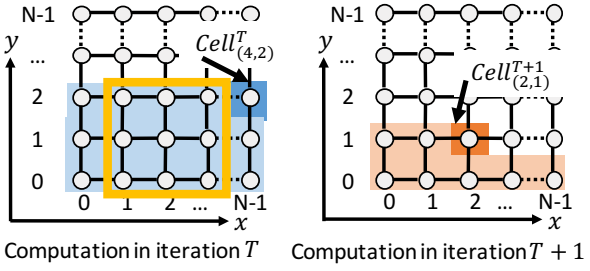
C. FDTD accelerator architecture using deep-pipelines

Previous works such as [4] and [5] have proposed FPGA accelerators that use iteration level parallelism for stencil computations such as Jacobi method. Those accelerators are designed using hardware description languages such as Verilog HDL. Therefore, a complete re-design is required for complex stencil computations such as FDTD, where a complicated data dependencies and boundary conditions exist among electric and magnetic field computations. Additional pipeline registers are required to solve the data dependencies and conditional branches are required to implement various boundary conditions. Since FDTD computation parameters, boundary conditions, required observation points, etc may vary for different simulations, re-designing the whole accelerator is required very often. However, this takes a large design, testing, debugging and compilation time and not practical. In this paper, we discuss how to design an FDTD accelerator entirely using software and still achieve good performance.

Fig.3 shows the architecture of the FDTD computation. It consists of multiple PCMs (pipelined computation modules) for parallel computation. The computations of one whole iteration are done in one PCM. The number of PCMs equals



(a) Computations of $Cell^{T}_{3,2}$ and $Cell^{T+1}_{1,1}$ are done in parallel.



(b) Computations of $Cell^{T}_{4,2}$ and $Cell^{T+1}_{2,1}$ are done in parallel.

Fig. 2. Cell-serial iteration-parallel computation.

to the number of parallel iterations and it is constrained by the FPGA resources. A PCM consists of shift-register arrays and multiple PEs (processing elements). The computation of a cell is done in a PE. Multiple PEs are used to compute electric and magnetic fields in parallel for different cells in the same iteration. All PEs are fully pipelined, so that an output is produced in every clock cycle after the pipeline is filled. Usually, we can implement over 40 PCMs in an FPGA in a pipeline, and we call such pipelines as deep-pipelines. Two shift-register arrays are used per an iteration. One array is used to carry the electric filed data for magnetic field computation in the same iteration. Another array is used to carry the results of one iteration to the next iteration.

We explain the lifetime of the temporary storage on registers in Fig.4 using an example of 3×3 stencil. Fig.4(a) shows the scan order of the grid and Fig.4(b) shows the order of the data are stored. In each clock cycle, new data are pushed-in while the oldest data are popped-out from the shift-register array. In this computation, one data value should remain $2 \times N + 4$ clock cycles in the pipeline. That means, we need a shift-register array of $2 \times N + 4$ stages long. The lifetime of the data depends on the grid size and the shape of the stencil. For the FDTD computation, the lifetime is $N + 3$ cycles. PEs can access any position of the shift registers in parallel. Therefore, the data are shared among the computations of the cells in the same iteration.

Fig.5 shows the FDTD computation flow. Initially, the host computer transfers data to the external memory of the FPGA. Then, FPGA computes d iterations in parallel and write the output data to the external memory. The FPGA computation continues for $Max\_Iterations/d$ times until all the iterations are finished. Then the output data are transferred to the host. In this method, the external memory is accessed only twice for every d iterations. Compared to that, the conventional method in Fig.1 accesses the external memory twice in every iteration. Therefore, the external memory access can be reduced greatly in the proposed architecture by increasing d.

III. Evaluation

For the evaluation, we used two FPGA boards; DE5 and 395-D8, four GPUs and two CPUs. FPGAs are configured using Quartus 15.0 with OpenCL SDK. GPUs are programmed using CUDA 7.5. CUDA 6.5 is used for the older GTX285 GPU. The operating system is CentOS 6.7. The FDTD computation example uses a 1024×1024 grid and runs for 15360 iterations. All computations are single precision.

Table I shows the processing time comparison for FPGAs, GPUs and multicore CPUs. The most straight-forward "cell-parallel" computation is used in CPUs and GPUs. The processing time of the CPUs and GPUs are decided by the memory bandwidth. GTX760 GPU that has the largest bandwidth gives the minimum processing time. The processing time of both FPGA boards are smaller than that of CPUs and GPUs. The DE5 FPGA gives 13.19 times and 4.12 times large processing speed compared to the fastest CPU and GPU implementations respectively. We achieved such performance using DE5, even having a 7.5 times smaller memory bandwidth, 11.5 times smaller peak performance and 3.6 times smaller clock frequency compared to GTX760. Note that, the FDTD accelerator generated on 395-D8 FPGA board uses two parallel data streams to double the computations. Since it has a larger memory bandwidth compared to that of DE5, such parallel data access is possible.

Fig.6 shows the performance of different devices measured in GFLOPS. We achieved over 114 GFLOPS and 87 GFLOPS for DE5 and 395-D8 FPGA boards. Compared to this, we achieved only 27.7 GFLOPS using GTX760. The reason for the low performance in GPUs and CPUs is the memory bandwidth limitation. The performance of 7 years old GTX285 GPU is similar to the performance of 1 year old GTX960 GPU due to the superior memory bandwidth of the older one. The FPGAs we used are also 6 years old Stratix V devices. However, the performance are many times better compared to recently launched GPUs. If we use the very recent Stratix 10 FPGAs that have more than 10 times of resources, we could expect over 10 times better performance compared to that of older Stratix V FPGAs.

Fig.7 shows the "effective to peak performance ratio (EPR)" given by Eq.(4). Although GPUs provide over 2200 GFLOPS of peak performance, only 2% of those are utilized by the FDTD computation. Compared to that, FPGA in the DE5 board uses over 58% of its peak performance. This is because, more parallel computations are implemented on the FPGA using cell-serial iteration-parallel method. Since the cells in the same iteration are processed in serial, data are also accessed in serial from the external memory. This serial data

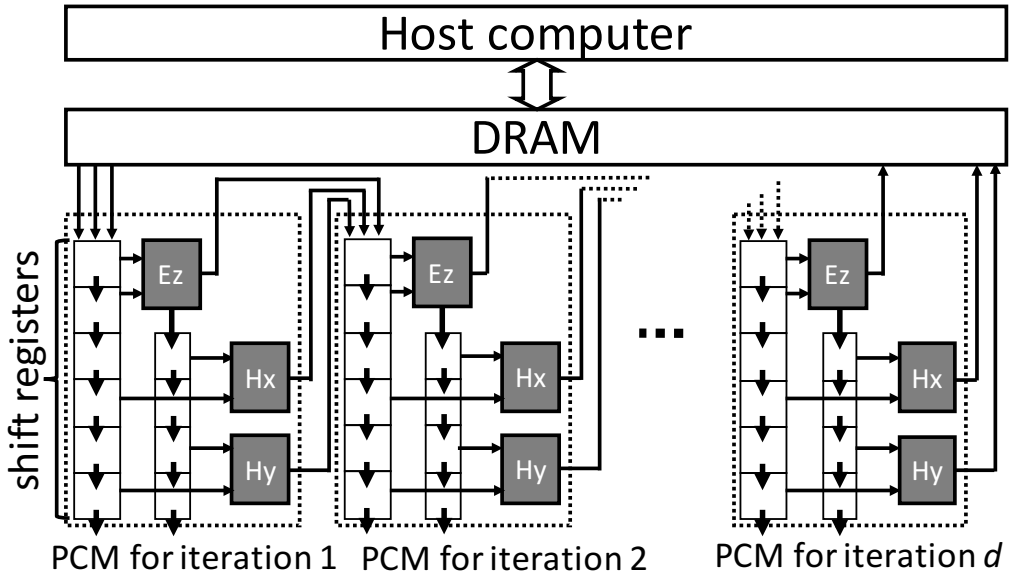


Fig. 3. The architecture of the FDTD accelerator that computes d iterations in parallel. PCM is a pipelined computation module.

TABLE I
PROCESSING TIME COMPARISON FOR FPGAS, GPUS AND MULTICORE CPUS.

	FPGA		GPU				CPU	
	DE5	**395-D8**	**C2075**	**GTX285**	**GTX760**	**GTX960**	**i7-4960x**	**E5-1650 v3**
Number of cores	44 PCMs	24×2 PCMs	448	240	1152	1024	6 (12 threads)	6 (12 threads)
Memory bandwidth (GB/s)	25.6	34.1	150.3	159	192.2	112.1	59.7	68
Peak performance (GFLOPS)	196	1503	1054	1062	2257	2308	47	307
Processing time (s)	1.69	2.20	9.20	10.6	6.97	9.90	26.14	22.30

access reduces the memory bandwidth. The computed results are stored in the shift-registers and carried to the next PCM for the computation of the next iteration. Only the data of the final PCM are written to the external memory. As a result, the memory bandwidth is not a bottleneck.

$$EPR = \frac{\text{Actual performance}}{\text{Peak performance}} \times 100\% \qquad (4)$$

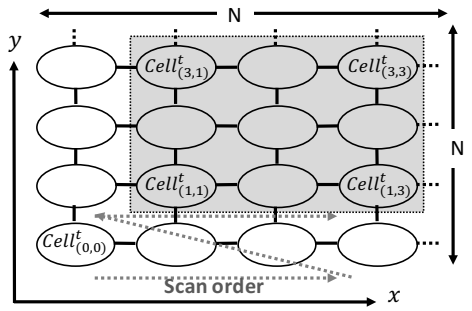
Table II shows the comparison between our proposed accelerator and the one in [10] for 2-D FDTD computation. The accelerator in [10] is implemented on a similar Stratix V FPGA using fixed-point arithmetic. According to the results, the processing speeds are very similar. However, [10] uses 32-bit fixed-point computation that requires less resources compared to our single-precision floating-point computation. Moreover, the proposal in [10] is a custom accelerator designed using hardware description language. Compared to that, our work is entirely software designed. This shows that the proposed OpenCL-based FPGA accelerator is very efficient and almost as good as a custom designed one. Compared to the other benefits such as extremely small design time, software based debugging and software-based updates for algorithm changes, boundary condition changes, etc, the proposed FDTD accelerator has a definite advantage.

TABLE II
COMPARISON WITH THE WORK IN [10].

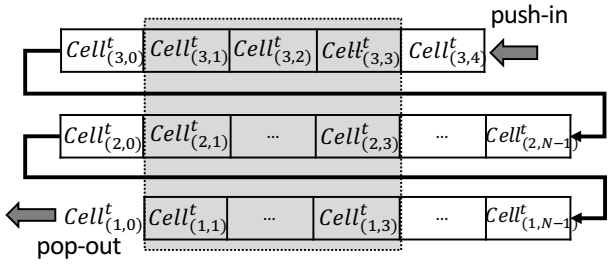
	FPGA	**Performance**
Work in [10]	Stratix V 5SGSMD5K2F40C2N	119.9 Giga fixed-point operations per second
This paper	Stratix V 5SGXEA7N2F45C2	114.2 GFLOPS

IV. CONCLUSION

We proposed an FPGA-based architecture for FDTD computation using OpenCL. The proposed architecture is designed to utilize the iteration level parallelism instead of the cell level parallelism to minimize the external memory access. The architecture contains deep-pipelines to fully utilize the computation results between iterations without accessing the external memory. We achieved over 13 times and 4 times larger processing speed compared to CPU and GPU implementations respectively. The proposed implementation gives almost the same performance compared to a custom accelerator designed using HDL. The proposed architecture is completely designed



(a) $N \times N$ grid and 3×3 stencil. Computations are done by scanning cells from left-to-right and down-to-up.



(b) Values of the cells are stored in shift registers. The oldest data are dumped to store new data in every clock cycle

Fig. 4. Lifetime of the values in the shift-register array.

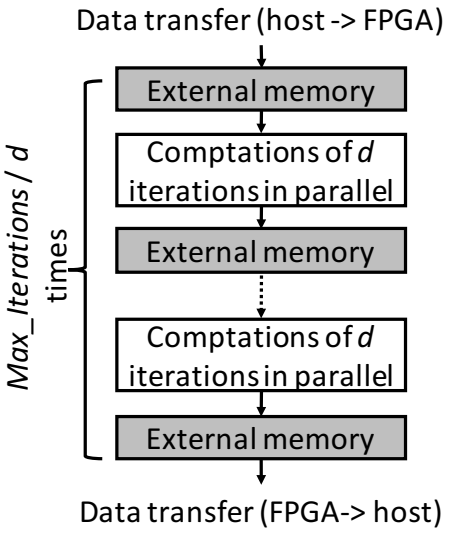


Fig. 5. Flow-chart of the proposed FDTD acceleration using FPGA.

by software using OpenCL. Therefore, the design time is extremely small compared to that of a custom accelerator. The same program code can be reused by recompiling it for any OpenCL capable FPGA board, irrespective of the FPGA type or I/O resources such as different external memory specifications, etc.

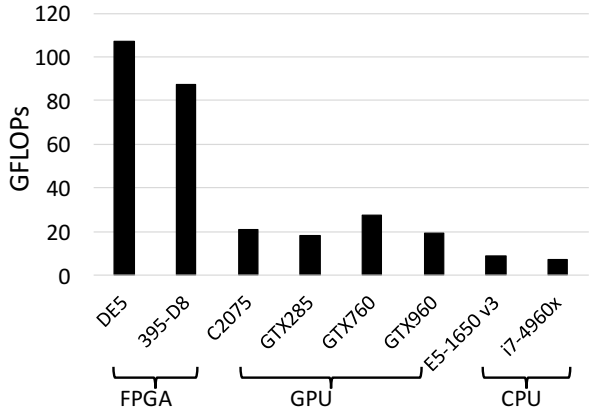


Fig. 6. Comparison of the performance against CPUs and GPUs.

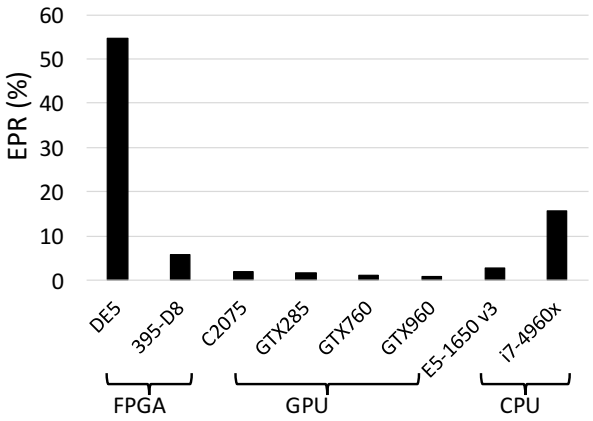


Fig. 7. Comparison of the effective to peak performance ratio (EPR).

ACKNOWLEDGMENT

This work is supported by MEXT KAKENHI Grant Number 15K15958.

REFERENCES

[1] K. S. Yee *et al.*, "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," *IEEE Trans. Antennas Propag*, vol. 14, no. 3, pp. 302–307, 1966.

[2] Z. Li and Y. Song, "Automatic tiling of iterative stencil loops," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 26, no. 6, pp. 975–1028, 2004.

[3] G. Wellein, G. Hager, T. Zeiser, M. Wittmann, and H. Fehske, "Efficient temporal blocking for stencil computations by multicore-aware wavefront parallelization," in *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, vol. 1, pp. 579–586, 2009.

[4] W. Luzhou, K. Sano, and S. Yamamoto, "Domain-specific language and compiler for stencil computation on FPGA-based systolic computational-memory array," in *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 26–39, Springer, 2012.

[5] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth," *IEEE transctions on parallel and distributed systems*, vol. 25, no. 3, pp. 695–705, 2014.

[6] J. Durbano and F. Ortiz, "FPGA-based acceleration of the 3D finite-difference time-domain method," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, pp. 156–163, 2004.

[7] H. Kawaguchi and S.-S. Matsuoka, "Conceptual design of 3-D FDTD dedicated computer with dataflow architecture for high performance microwave simulation," *Magnetics, IEEE Transactions on*, vol. 51, no. 3, pp. 1–4, 2015.

[8] Y. Ishigaki, Y. Tomioka, T. Shibata, and H. Kitazawa, "An FPGA implementation of 3D numerical simulations on a 2D SIMD array processor," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pp. 938–941, 2015.

[9] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, "An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm," in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, FPGA '04, pp. 213–222, 2004.

[10] R. Takasu, Y. Tomioka, Y. Ishigaki, L. Ning, T. Shibata, M. Nakanishi, and H. Kitazawa, "An FPGA implementation of the two-dimensional FDTD method and its performance comparison with GPGPU," *IEICE Transactions on Electronics*, vol. 97, no. 7, pp. 697–706, 2014.

[11] "Altera SDK for OpenCL." https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html, 2016.

[12] S. Tatsumi, M. Hariyama, M. Miura, K. Ito, and T. Aoki, "OpenCL-based design of an FPGA accelerator for phase-based correspondence matching," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, p. 90, 2015.

[13] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pp. 16–25, 2016.

[14] Y. Takei, H. M. Waidyasooriya, M. Hariyama, and M. Kameyama, "FPGA-oriented design of an FDTD accelerator based on overlapped tiling," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, p. 72, 2015.

[15] "The open standard for parallel programming of heterogeneous systems." https://www.khronos.org/opencl/, 2015.