Designing a Graphics Accelerator with Heterogeneous Architecture

Ilya Tarasov(✉), Dmitry Mirzoyan, and Peter Sovietov

MIREA – Russian Technological University, Vernadsky Avenue 78, 119454 Moscow, Russia
tarasov_i@mirea.ru

Abstract. The article discusses the architecture of a graphics accelerator, based on a combination of general-purpose processor cores and pipeline accelerators for performing operations with matrices and transcendental operations. The article proposes a general architecture for GPUs of this type and suggests the main options for computing nodes designed to implement target group algorithms. In order to reduce technical and organizational risks, it is planned to simplify the hardware component of Very Large Scale Integrated Circuits (VLSI) and transfer the functions of managing calculations to embedded software, for which control processor cores have been introduced into VLSI. The VLSI project involves the development of a GPGPU-class computing accelerator, in which the ability to work with three-dimensional graphics is an additional feature. This allows you to take advantage of an architecture based on a large number of simple computational cores, using such VLSI in conjunction with a general-purpose processor.

Keywords: VLSI · Heterogeneous Architecture · Graphical Accelerator · GPU

1 Introduction

Currently, the relevance of creating an element base for high-performance computing is increasing. It is known that specialized computing devices are more efficient than general-purpose computers, but Very Large Scale Integrated Circuits (VLSI) with a limited scope of application have a smaller market and therefore turn out to be prohibitively expensive for small production runs. It is also important that, along with the development of technological standards of 3 nm and less technology node, the production of 28-7 nm standard VLSI and in some cases even larger standards (90–45), for applications with low performance requirements, but sensitive to development costs.

A widely known approach is based on the use of the GPU as a hardware accelerator working in combination with the CPU. Technologies such as Nvidia CUDA and OpenCL provide a layer of hardware abstraction which allows the use of a high-level C-like programming language. This expands the scope of use of CPU+GPU class computing systems. At the same time, GPUs can be considered as a type of architecture based on the use of a large number of simple computing cores grouped into clusters. This architecture can use combined connections at different levels of the hierarchy: tree, fat tree, ring, grid/mesh, etc.

V. Jordan et al. (Eds.): HPCST 2023, CCIS 1986, pp. 29–40, 2024.
https://doi.org/10.1007/978-3-031-51057-1_3

Based on the significant complexity of designing VLSI, comparable in characteristics to solutions of the world's leading manufacturers (such as Nvidia and AMD), we can consider an alternative approach based primarily on the implementation of a computation accelerator, which additional function would be to work as a graphics coprocessor. It is worth noting that a number of GPU families are also focused on use as part of workstations with high performance in general-purpose tasks. Changing priorities will allow us to distance ourselves from performance assessments based on 3D graphics algorithms, since this is not the main purpose of such VLSI. At the same time, it becomes possible to implement architectural and circuit solutions that further enhance the capabilities of VLSI in target subclasses of tasks, where existing GPGPU VLSI are forced to also support data processing algorithms for 3D graphics. Studying the specifics of individual subclasses of algorithms and clarifying the current list of tasks that require hardware support is planned as part of the research work within the framework of VLSI design.

The following areas of application of specialized VLSI are considered:

- digital signal processing accelerators;
- software-defined radio;
- measuring instruments;
- medical equipment;
- accelerators for image processing;
- CCTV;
- industrial robots;
- machine learning;
- VR/AR.

2 Architecture of a Specialized Graphics Accelerator

According to Hennessey and Patterson [1], the dominant trend in performance improvement is Domain-Specific Architectures, DSA. At the same time, the need to place control components, and especially program memory, within the processor node increases the relative hardware costs for implementing one operation. Therefore, along with programmable computing nodes, non-programmable computing nodes designed to implement frequently used transformations can also be used as part of a specialized computing system. Non-programmable pipeline structures can be modified to be able to switch between separate operations at each stage, or to combine data movement along the pipeline with cyclic repetition of calculations at the same stage.

The combination of hardware and software methods for implementing calculations within the GPU was used both in Intel Larrabee projects [2] and in research projects based on the RISC-V core to implement general-purpose computing [3] or directly 3D graphics [4]. This gives reason to consider a similar approach using newly developed processor cores specialized for certain types of calculations as part of VLSI.

To design components that perform calculations as part of specialized VLSI chips, the following system-level implementation options can be considered:

1. Making changes to the data processing path of a specialized VLSI with a wide command word in order to provide support for operations typical for crypto conversions.

2. Connecting a pipelined data processing path to the processor core with a wide command word as an auxiliary arithmetic-logical device.
3. Connecting a pipelined data processing path to the processor core or system bus with a wide command word as a stand-alone configurable device.

The listed options can be considered as candidate architectures with clarification of their characteristics at the system level.

At the computing device architecture level, the following options can be considered:

1. Programmable computing node (processor).
2. Non-programmable (configurable) pipeline path for processing streaming data.
3. Combination of conveyor paths and programmable nodes.

The considered architectures of computing nodes are shown in Fig. 1.

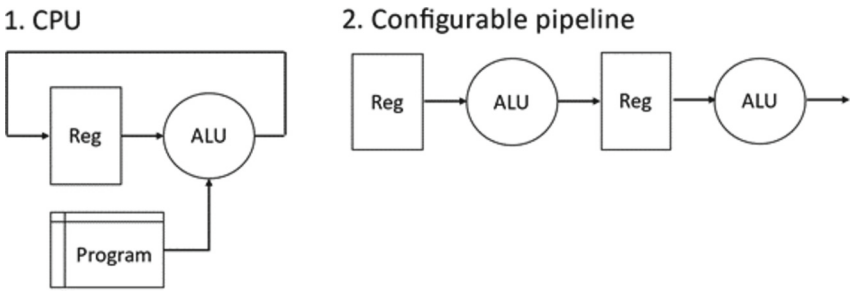

Fig. 1. Architectures of VLSI graphics accelerator computing nodes.

The architectures shown above correspond to mutually complementary approaches to organizing computing – CPU-based, i.e. distributed in execution time, and pipeline-based, distributed in the space of the VLSI chip. The current trend of using synchronous pipelined computers makes them preferable, but the functionality of a pipeline is determined by the order in which its stages are connected, while for a processor the order of calculations is determined by the program code and can be changed at runtime. At the same time, the hardware redundancy of the processor is determined by the need to add program memory in such a size that would ensure the execution of all algorithms of the target group.

In Fig. 2 the layout of a VLSI cluster that combines the functions of a graphics controller and a general-purpose computing accelerator is shown.

The architecture of the processor that implements the computing node is the subject of research. Compared to a general-purpose core (such as RISC-V), it is possible to further specialize the instruction set architecture while maintaining a simple microarchitecture. Reducing the redundancy of a single core will have a positive impact on the performance of VLSI chips that contain many such cores.

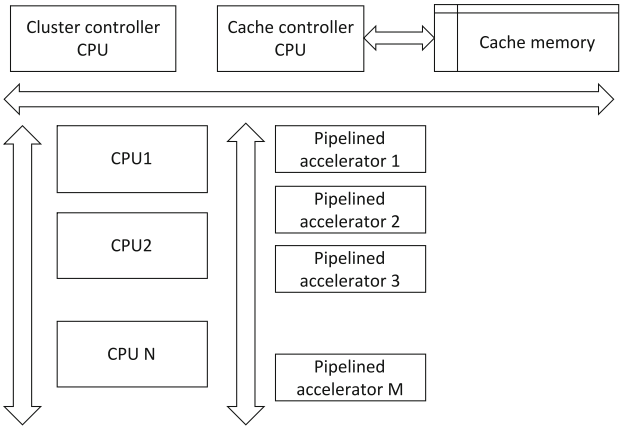

Fig. 2. Architecture of a graphics accelerator cluster aimed at general-purpose computing.

3 Architectural Solutions for Graphics Accelerator

The following architectural solutions are being considered for a promising graphics accelerator.

3.1 Programmable Task Distribution

GPU-specific computing tasks combine simple RISC-like operations and pipelined computing, as discussed in the previous section. A reduction in the complexity of the hardware component of the GPU control system can be achieved by transferring the task distribution functions entirely to the software component of the system. To perform this, a specialized task management processor is added to the VLSI cluster, which has access to the system bus and is controlled by both system drivers of the central processor and embedded software. The control processor runs cluster-local pipeline accelerators, if possible, or implements operations in software. To do this, it is necessary to provide access to the program memory of auxiliary processors based on dual-port memory.

3.2 Software Memory Management

The combination of on-chip static memory and external dynamic memory requires the implementation of a controller that, among other things, performs caching using certain algorithms. Depending on the scenarios for working with data, different caching algorithms may be optimal, which significantly complicates the design of the controller in the absence of a database of experimental data collected on implemented GPUs. Therefore, for a VLSI prototype project, software memory management is assumed with the allocation of address spaces and copying of data between memory of different types under the control of a dedicated processor.

3.3 Distribution of Tasks Depending on Data

If there are heterogeneous computational blocks, it becomes possible to assign a block not only in accordance with the type of task, but also, in some cases, depending on the specific values of the data being processed. For example, in a rotation matrix for angles that are multiples of 90 degrees, all coefficients are equal to 0, 1, or −1, which greatly simplifies multiplication by such matrices. Support for such operations is possible by introducing special flags that provide quick return of the result if one of the operands is 0, 1 or −1.

3.4 Redistribution of Resources within the Computing Cluster

Differences in resource requirements imposed by various algorithms of the target group under consideration necessitate the addition of memory and functional nodes, which will be redundant for a certain subclass of computing. Therefore, an approach based on placing a switched matrix of functional nodes, such as processor devices, memory and configurable pipelines, within one cluster is being considered. The ability to dynamically switch connections at medium and large levels of the hierarchy will allow memory to be redistributed between computing nodes, adapting VLSI to the requirements of the corresponding memory-intensive algorithms.

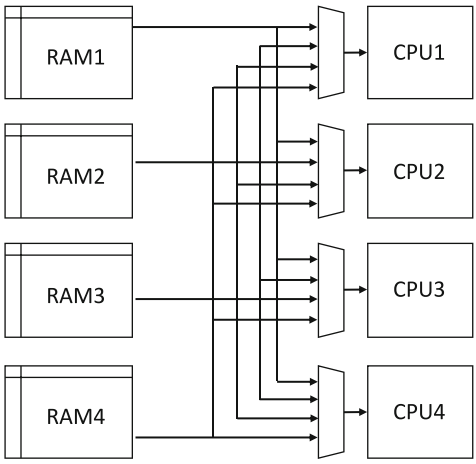

Fig. 3. An example of programmable distribution of memory blocks between processor cores.

In Fig. 3, N memory blocks are connected to M processor cores using full switches. In the mentioned scheme, it is possible both to distribute blocks in pairs across the corresponding processor cores, and to transfer all memory to one or more processor cores. This mode may be required to implement algorithms that require a large amount of memory. In this case, the overall performance of a group of processors will be reduced, since some of the cores are idle due to a lack of free memory blocks, but the ability to execute the algorithm remains.

In some cases, generalizing memory to form a larger block will not cause processors to turn off if the algorithm being executed is SIMD (Single Instruction, Multiple Data) class compliant.

3.5 Thread Management According to the SIMD Approach

Reducing the amount of required memory is possible by using the SIMD approach, in which the same program is used to control multiple compute nodes. This solution is suitable for a number of problems in three-dimensional graphics, digital signal processing (for example, multi-channel filtering) and mathematical modeling of processes using the finite element method.

3.6 Combining Raster and Cache Memory for General-Purpose Computing

For high resolution images (FullHD and 4K2K), storing pixels in a format that complies with the TrueColor standard requires a minimum of 24 bits per pixel, i.e. 48 Mbit for FullHD resolution and 192 Mbit for 4K2K. Additionally, we can use an alpha channel, as well as a hardware depth buffer with a resolution of at least 16 bits per pixel. This increases the total image buffer size to 96 or 256 Mbits of memory. If we consider the use of static memory, which with such a volume will occupy a large area, we should consider using this memory not only in the form of a screen buffer, but also as a general cache memory of the computation accelerator.

To represent general-purpose programs, one can use the SPIR-V language [5], which is used for intermediate representation of OpenCL programs. Since the language is designed as an intermediate language, its implementations are considered both for general-purpose processors [6] and for hardware accelerators designed on the basis of FPGAs. For example, [7] addresses accelerator integration, and other works explore the use of OpenCL for specialized areas. For example, digital filters [8], convolutional neural networks [9], and image motion prediction systems [10] use limited subsets of OpenCL, so they can be implemented more efficiently by taking this factor into account.

However, GPUs with OpenCL capability require an implementation of the full SPIR-V specification. This task is complicated by the heterogeneous nature of the operations supported in the language specification. For example, simple bitwise logical and arithmetic operations on integer arguments are easily implemented both as part of the arithmetic-logical unit of the processor core and as part of a pipeline, although pipelining such simple operations is inappropriate in most cases. Integer and floating point multiplication operations require pipelining, but can be implemented as hardware extensions to the processor core. Finally, transcendental functions (primarily trigonometric) implemented using the CORDIC algorithm require pipelined or cyclic implementation. Taking into account the features of subsets of SPIR-V commands, we can assume the joint use of the described approaches with the corresponding distribution of operations by type of implementing device, as shown in Fig. 4.

The integration of heterogeneous components within the processor subsystem was considered in [11]. The design route involves conducting pre-RTL modeling of the system to clarify its characteristics, followed by the implementation of parameterized

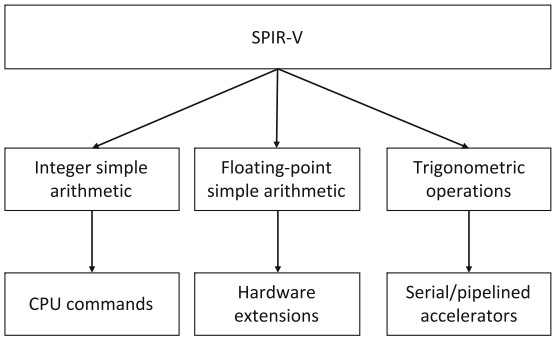

Fig. 4. Distributing SPIR-V intermediate language operations between cluster components.

components to perform individual operations. It appears promising to implement a configurable pipeline to perform operations based on the CORDIC algorithm, which form a large subset of SPIR-V instructions.

The general format of the command can be represented as follows:

$$<Dest> = <Operand1> \text{ op } <Operand2>$$

where *Dest* is the destination device (register) for storing the result of the operation; *Operand1* – first operand; *Op* – type of operation; *Operand2* – is the second operand.

For a command system, the concept of addressing is used, which reflects the number of registers described in the command code. Increasing addressability generally increases the capabilities of the tool software, but also increases the bit width of the command word, and therefore the amount of memory required to store the program. In this case, the absence of an indication of a particular type of resource means that it implicitly follows from the type of operation being performed or coincides with the specified resources. For example, in the instruction set of x86 processors, the destination register is the same as the first operand, in accumulator architectures the destination register and the first operand is always the accumulator, and in a stack architecture, the operands are always located on top of the data stack, and the result is also placed there.

In [12], a unified description of a computing node by four parameters (I, O, D, S) is considered, where I – number of instructions executed per clock cycle; O – number of operations determined by the instruction; D – number of operands (pairs of operands) related to operations; S – degree of conveyorization.

Based on this unified description, an ALU with a set of operations and combinations of operands is selected for the processor node, which:

1. Sufficient for implementing target group algorithms.
2. Optimal according to the selected criteria.

Optimality criteria include the number of clock cycles for executing algorithms, the area of VLSI (or the number of FPGA cells), the amount of program and data memory, and power consumption. The design route established for this project does not include early determination of the optimality criterion, since this limits the design space.

For a processor node, the register file model is not specified in SPIR-V and can be selected during the design process. This makes the number of registers and read/write ports parameters of the optimization process.

In Fig. 5 the transition to partial generalization of the resources of the pipeline stages is shown. If the pipeline generalizes only a register group, but the functional devices at individual stages have a similar structure and use the same subblocks, it may be possible to partially generalize such subblocks and implement multiplexers not between data paths, but between subblocks that are not identical in different versions of the data path.

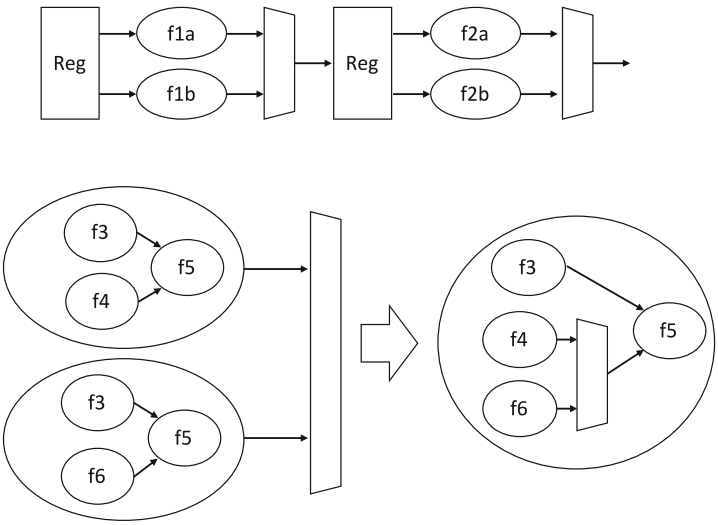

Fig. 5. Transformation of individual stages of a pipeline computer for the purpose of partial generalization of resources.

The implementation option of a conveyor with partial generalization of the resources of individual stages provides better component density, but at the same time complicates heat removal when using technological standards susceptible to the "dark silicon" effect. Therefore, the possibility of parameterized synthesis of pipeline stages should be maintained throughout the early stages of the project, right up to clarifying the characteristics of the topology library.

As part of preliminary research, the characteristics of a conveyor at stage 32, combining the performance of two types of operations, were assessed. The original RTL description of the module uses only one LUT layer using switching based on additional resources of logical cells - F7MUX, F8MUX. The pipeline is synthesized using 2247 LUTs and 2821 FFs based on the Xilinx Kria module. The trace results for a single pipeline are shown in Fig. 6.

An analysis of the placement of a pipeline with a set clock period of 1.5 ns shows that even in the absence of area constraints for the Xilinx Kria FPGA, the specified frequency is achieved due to the dense layout of the pipeline stages. An additional positive effect is the ability to shift the phase of the clock signal for individual registers (time borrow) for

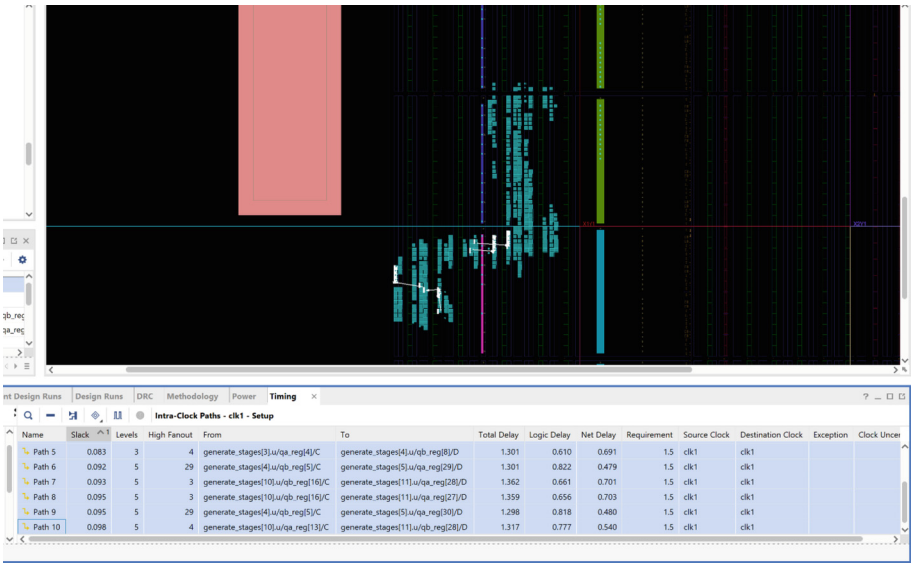

Fig. 6. Results of tracing a switched pipeline project without using area constraints

FPGAs made using 16 nm FinFET technology. This provides balancing of delays within the pipeline, implemented by CAD without direct involvement of the developer. Thus, pipeline structures are of interest as specialized devices for accelerating calculations, while providing compact placement of nodes and low latency due to local connections between individual stages of the pipeline.

4 Automation of Design of Specialized Nodes

Since the designed processor elements and pipeline computers are not standard, their development process is the subject to two opposing trends. On one hand, the improvement in functionality is determined by the complication of the ALU, an increase in the amount of processor resources and the complication of functions at individual stages of the pipeline. On the other hand, these actions lead to increased signal delays, die area, and power consumption. In the early stages of design, it is too difficult to create an accurate model of the target tasks, so priority should be given to the development of a VLSI system model that would allow the performance to be assessed for a certain combination of specified component parameters.

High performance of a hardware accelerator can be achieved by specializing its structure for those narrow classes of tasks where computational operations predominate over control operations and access to general data. At the same time, various forms of static parallelism are implemented in hardware form:

– task parallelism;
– data parallelism;
– pipelining.

High energy efficiency of specialized accelerators is achieved, in many cases, due to the irregularity of their structure, reflecting the specifics of a narrow class of problems, as well as through the use of local, direct connections between computing elements [13].

In this regard, first of all, specialization of the accelerator data path is of interest. It can be implemented using code analysis of target algorithms at various levels [14, 15]:

– an expression consisting of a small number of operations;
– linear section;
– cycle nest;
– hammock or function;
– call graph.

In many cases, a dedicated accelerator is used in conjunction with a control processor in one of the following configurations:

– part of the general data path as part of the control processor;
– a separate hardware unit connected to the control processor via some external interface.

In the case of using a common data path, the specialization no higher than the linear section level is the most preferable, since in this scheme a specialized accelerator competes with the control processor for shared resources, such as command fields, register file, memory.

Achieving the highest performance should be expected with hardware acceleration of calculations of complex software structures that include loop nests. Moreover, if the accelerator is implemented as a separate hardware unit, then the control processor is free to perform other tasks in parallel with specialized calculations.

To automate individual design tasks, a specialized CAD system is being developed, intended for the subclass of systems described in this article. The system uses descriptions of target algorithms in a high-level language [16] to analyze their features and distribute tasks between processor devices and pipelines. In addition to analyzing text representations, graphs are also currently used for this purpose [17], however, this method seems more labor-intensive if the volume of analyzed algorithms increases. Text analyzers are also used for this purpose [18–20].

The main tasks of CAD are:

– formation of a structural description of the upper level of VLSI;
– setting the parameters of VLSI components and formally checking their admissibility;
– integration with the compiler.

The technical specifications for CAD development are clarified as information is received about VLSI architectures, component parameters, assembly scenarios at the top level of description and other elements of the project, and design work that makes up the development process.

For the development of CAD, in accordance with the identified trends, a modular architecture is assumed in combination with a common project database. CAD elements can include both software applications developed in high-level languages and scripting languages, CAD scripts in these languages, as well as software interfaces of third-party tools such as compilers and applications for modeling domain processes.

5 Conclusions

The materials presented in the article represent the results of preliminary studies of the GPU architecture, intended to work as a computation accelerator as part of high-performance computing systems.

Acknowledgement. This work is supported by the Ministry of Science and Education of RF (Project No. FSFZ-2022-0004).

References

1. Hennessy, J.L., Patterson, D.A.: Computer Architecture. 6th edn. A Quantitative Approach (The Morgan Kaufmann Series in Computer Architecture and Design) (2017)
2. Seiler, L., et al.: Larrabee: a many-core x86 architecture for visual computing. IEEE Micro **29**(1), 10–21 (2009)
3. Elsabbagh, F., et al.: Vortex: OpenCL Compatible RISC-V GPGPU (2020). https://doi.org/10.48550/arXiv.2002.12151. Accessed 27 Feb 2020
4. Tine, B., Elsabbagh, F., Yalamarthy, K., Kim, H.: Vortex: extending the RISC-V ISA for GPGPU and 3D-graphicsresearch. In: Proceedings of MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, October 2021, pp. 754–766 (2021). https://doi.org/10.1145/3466752.3480128
5. Khronos Group: Khronos SPIR-V Registry (2021). https://registry.khronos.org/SPIR-V/. Accessed 10 Oct 2023
6. He, W., et al.: Streamline ahead-of-time SYCL CPU device implementation through bypassing SPIR-V. In: Proceedings of the 2023 International Workshop on OpenCL, April 2023, Article no. 28 (2023). https://doi.org/10.1145/3585341.3585381
7. Leppänen, T., Lotvonen, A., Mousouliotis, P., Multanen, J., Keramidas, G., Jääskeläinen, P.: Efficient OpenCL system integration of non-blocking FPGA accelerators. Microprocess. Microsyst. **97**, 104772 (2023). https://doi.org/10.1016/j.micpro.2023.104772
8. Firmansyah, I., Yamaguchi, Y.: Real-time FPGA implementation of FIR filter using OpenCL design. J. Signal Process. Syst. **94**, 1–13 (2022). https://doi.org/10.1007/s11265-021-01723-6
9. Wu, Y., Zhu, H., Zhang, L., Hou, B., Jiao, L.: Accelerating deep convolutional neural network inference based on OpenCL. In: Shi, Z., Jin, Y., Zhang, X. (eds.) ICIS 2022. IFIPAICT, vol. 659, pp. 98–108. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14903-0_11
10. de Castro, M., Osorio, R., Vilariño, D., Gonzalez-Escribano, A., Llanos, D.: Implementation of a motion estimation algorithm for Intel FPGAs using OpenCL. J. Supercomput. **79**(5), 1–23 (2023). https://doi.org/10.1007/s11227-023-05051-3
11. Tarasov, I.E., Potekhin, D.S., Platonova, O.V.: Prospects for the use of soft processors in systems on a chip based on programmable logic integrated circuits. Russ. Technol. J. **10**(3), 24–33 (2022). https://doi.org/10.32362/2500-316X-2022-10-3-24-33
12. Sima, D., Fountain, T., Kacsuk, P.: Advanced Computer Architectures: A Design Space Approach. Addison-Wesley (1997)
13. Trilla, D., Wellman, J.-D., Buyuktosunoglu, A., Bose, P.: Novia: a framework for discovering non-conventional inline accelerators. In: Proceedings of 54th Annual IEEE/ACM International Symposium on Microarchitecture, October 2021, pp. 507–521 (2021). https://doi.org/10.1145/3466752.3480094

14. Zacharopoulos, G., Ferretti, L., Ansaloni, G., Di Guglielmo, G., Carloni, L., Pozzi, L.: Compiler-assisted selection of hardware acceleration candidates from application source code. In: Proceedings of 2019 IEEE 37th International Conference on Computer Design (ICCD), pp. 129–137. IEEE (2019)
15. Brumar, I., Zacharopoulos, G., Yao, Y., Rama, S., Wei, G.-Y., Brooks, D.: Early DSE and automatic generation of coarse-grained merged accelerators. ACM Trans. Embed. Comput. Syst. **22**(2), 32 (2021). https://doi.org/10.1145/3546070
16. Dave, S., Shrivastava, A.: Design space description language for automated and comprehensive exploration of next-gen hardware accelerators. In: Proceedings of Workshop on Languages, Tools, and Techniques for Accelerator Design (LATTE 2022) (2022)
17. Ferretti, L., Cini, A., Zacharopoulos, G., Alippi, C., Pozzi, L.: Graph neural networks for high-level synthesis design space exploration. ACM Trans. Des. Autom. Electron. Syst. **28**(2), 25 (2022). https://doi.org/10.1145/3570925
18. Agostini, N.B., et al.: An MLIR-based compiler flow for system-level design and hardware acceleration. In: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, pp. 1–9 (2022). https://doi.org/10.1145/3508352.3549424
19. Venkataramani, G., Budiu, M., Chelcea, T., Goldstein, S.C.: C to asynchronous dataflow circuits: an end-to-end toolflow. Carnegie Mellon University, J. Contrib. (2018). https://doi.org/10.1184/R1/6603986.v1
20. Jordan, H., Scholz, B., Subotić, P.: Soufflé: on synthesis of program analyzers. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9780, pp. 422–430. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_23