Computing, Telecommunication and Control, 2025, Vol. 18, No. 4, Pp. 76–86.
Информатика, телекоммуникации и управление. 2025. Том 18, № 4. С. 76–86.

Software and Hardware of Computer, Network, Telecommunication, Control, and Measurement Systems

Компьютерные сети, вычислительные, телекоммуникационные, управляющие и измерительные системы

Research article
DOI: https://doi.org/10.18721/JCSTCS.18407
UDC 004.312.44

DESIGN AND ANALYSIS OF A RECONFIGURABLE HARDWARE ACCELERATOR FOR SOLVING A SYSTEM OF LINEAR EQUATIONS USING JACOBI METHOD

M.F. Gonzalez, A.P. Antonov ✉

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

✉ antonov_ap@spbstu.ru

Abstract. This work presents the design and analysis of a reconfigurable hardware accelerator for solving a system of linear equations using Jacobi method, implemented on a reconfigurable device, as well as a comparative study of software and hardware implementations. Recent advancements in computing capabilities have been hindered by the so-called “walls”: memory, power consumption and clock frequency limitations imposed by current technology. Solutions to overcome these “walls” include reconfigurable computing and high-level synthesis. The system under development and analysis was described in the C++ language and implemented using a high-level synthesis method, which reduces design time and enables more efficient exploration of different hardware architectures. The comparative analysis showed a performance increment over the original implementation, with energy consumption comparable to that of a modern mid-class microprocessor.

Keywords: supercomputer, reconfigurable hardware accelerator, Jacobi method, field-programmable gate array, high-level synthesis, SystemVerilogHDL, reconfigurable computing

Citation: Gonzalez M.F., Antonov A.P. Design and analysis of a reconfigurable hardware accelerator for solving a system of linear equations using Jacobi method. Computing, Telecommunications and Control, 2025, Vol. 18, No. 4, Pp. 76–86. DOI: 10.18721/JCSTCS.18407

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.18407
УДК 004.312.44

ПРОЕКТИРОВАНИЕ И АНАЛИЗ РЕКОНФИГУРИРУЕМОГО АППАРАТНОГО УСКОРИТЕЛЯ ДЛЯ РЕШЕНИЯ СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ МЕТОДОМ ЯКОБИ

М.Ф. Гонзалез, А.П. Антонов ✉

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

✉ antonov_ap@spbstu.ru

Аннотация. В данной работе представлены разработка и анализ реконфигурируемого аппаратного ускорителя для решения системы линейных уравнений методом Якоби, реализованного на реконфигурируемом устройстве. Проведено сравнительное исследование производительности аппаратной и программной реализаций. В последнее время рост вычислительных возможностей высокопроизводительных вычислительных систем сдерживается такими преградами, как память, энергопотребление, тактовая частота, накладываемыми современными технологиями. Решениями для преодоления указанных преград являются реконфигурируемые вычисления и высокоуровневый синтез. Разрабатываемая и анализируемая система была описана на языке C++ и реализована с использованием метода высокоуровневого синтеза, что позволило сократить время проектирования и эффективнее исследовать различные аппаратные архитектуры. Сравнительный анализ показал увеличение производительности по сравнению с первоначальной реализацией при потреблении энергии, сопоставимом с современным микропроцессором среднего класса.

Ключевые слова: суперкомпьютерный вычислитель, реконфигурируемый аппаратный ускоритель, метод Якоби, сверхбольшие интегральные схемы программируемой логики, высокоуровневый синтез, язык SystemVerilogHDL, реконфигурируемые вычисления

Для цитирования: Gonzalez M.F., Antonov A.P. Design and analysis of a reconfigurable hardware accelerator for solving a system of linear equations using Jacobi method // Computing, Telecommunications and Control. 2025. Т. 18, № 4. С. 76–86. DOI: 10.18721/JCSTCS.18407

Introduction

The computing capabilities of microprocessors increased steadily for decades. However, this trend has recently been significantly disrupted due to the so-called “walls”: memory, power consumption, clock frequency/technology. As an alternative to general-purpose microprocessors for performance-critical tasks, application-specific integrated circuits (ASICs) can be used. However, they involve high complexity, cost and design time on the one hand, and narrow specialization for the solution of a single task or the implementation of one algorithm, on the other hand. The solution of general problems using graphics processing units (GPGPUs), which fall into the single instruction, multiple data (SIMD) category according to Flynn’s taxonomy, is not efficient in terms of performance and power consumption, since GPGPU is optimized for vector and vectorizable problems [1–3].

This trend is evident, for example, in the analysis of the Top 500 list. On the one hand, performance growth can be observed, achieved by increasing the number of cores and power consumption. On the other hand, as showed in [4], using the conjugate gradient method instead of the “classical” Linpack leads to performance degradation of two orders of magnitude.

Thus, there is a relationship between performance and computing architecture: existing architectures, optimized for solving one task, are not efficient for solving similar tasks with different methods, let alone tasks of different classes.

A solution to this problem is the use of hardware reconfigurable accelerators, whose physical structure is adapted to the algorithm of the task to be solved [5, 6].

The base of reconfigurable accelerators is hardware reconfigurable devices, such as field-programmable gate arrays (FPGAs). Widely known FPGAs enable the implementation of hardware solutions for computationally complex tasks with high performance, close to that of ASICs, but with significantly lower time and resource consumption. However, the traditional approach to FPGA design, including the use of hardware description languages, require specialized knowledge and skills, which has essentially limited the practical adoption of FPGAs as reconfigurable accelerators.

A solution to this problem is the use of high-level synthesis methods [7], which enable an abstraction of the complexity of hardware development. Apart from the automatization of the design process, high-level development environments enable the generation and analysis of a wide range of hardware architectures, including parallel and pipelined variants.

The system of linear equations (SLE) models a broad spectrum of scientific problems, such as weather forecasting or finite element methods. SLE is typically represented in the form of matrix-vector multiplication, which is efficiently processed by computers. There are different approaches to solving SLE, mainly direct and iterative methods. This article focuses on the Jacobi iterative method.

Unlike other methods, such as Gauss–Seidel, the Jacobi method computes all equations before updating the values of the unknown, meaning that hard dependencies between variables arise only between computations at different iterations. This characteristic gives the Jacobi method a high degree of parallelization, which increases with the size of the problem [8]. For this reason, a massively parallel architecture may represent an effective platform for solving SLE using the Jacobi iterative method. Thus, reconfigurable devices appear particularly well-suited for this task.

Several authors have presented their contributions to this problem in recent years [9–11]. A general formulation of the problem is presented at [12]. As in most hardware implementations, there are several possible approaches, from completely parallel systems to fully sequential ones, as well as intermediate levels of parallelism. In [12], both parallel and sequential approaches are proposed, while also outlining the principles of pipelined processing systems. Considerations regarding system complexity are also discussed.

In [13], the Jacobi method is applied to fractal calculations. According to the finite element method, each element in a mesh is most closely related to the neighboring nodes. These dependencies are modeled by SLE, where each equation computes a new value for each node according to the values of its four nearest neighbors. A convenient degree of parallelism is achieved by dividing the mesh into sub-frames, which are computed in parallel.

In [14], a similar problem is addressed, related to solving a large SLE using the Jacobi iterative method. In this case, the mathematical problem under investigation is solving Laplace's equations with Dirichlet boundary conditions, where standard (non-boundary) nodes are computed based on their four neighbors. This structure is modeled by a computing system based on computing blocks (nodes) interconnected by communication channels. The connections may be physical, corresponding to channels of spatially parallel structures, or virtual, represented by the exchange of information between iterations.

The computing architecture is also studied from different points of view: as fully parallel (spatial), fully sequential (temporal) or as a combination of both approaches adapted to the computing resources available on the reconfigurable device. The elementary computing block may receive input data in parallel, which in general corresponds to the values of neighboring nodes, or simpler but slower computing blocks may receive this data sequentially, processing it using corresponding accumulators and registers for storing the intermediate calculations between clock cycles. This reduction in the required logical resources simultaneously allows to limit the number of communication channels to a small fraction of the initially needed amount.

The decision on which approach will be implemented depends on the ratio of required computing resources, related to the size of the problem, to the resources available on the physical device. Therefore, elementary blocks, whether based on accumulators or not, may be reproduced several times according to the ratio of required to available resources, taking into account the achieved performance. A computing structure pipelined in several stages allows achieving higher performance without a significant increase of required logical resources by their reuse once intermediate results are no longer needed. The limited number of available communication channels also plays a key role, since the ratio of external communication channels on reconfigurable devices to the increasing area of logical computing elements has been observed to decrease constantly. The authors also studied the performance advantage of using clusters of reconfigurable devices for solving large-scale problems.

It is commonly accepted that the traditional approach based on hardware description language (HDL) design is excessively complex and time-consuming, especially for large systems of equations that required to be adapted to the resources available on the target device. In [11], a design for a Maxeler acceleration card is presented, using MaxCompiler, which allows system to be described in Java language. The key proposal for pipelining the system is to compute as many equations as necessary within the first iteration so that the second and successive iterations may start afterwards in a pipelined fashion, overcoming the inter-iteration dependencies described above.

In [15], the Jacobi iterative method in a high-level language for hybrid reconfigurable platforms is described. In [16], the Jacobi solver on a SCR heterogeneous supercomputer using the HLL-to-HDL compiler is implemented. Mapping the floating-point units was identified as the main difficulty faced by the authors. In [17], the use of Jacobi solvers is proposed, each solving one of the equations and acting in parallel within the global system. Other related architectural aspects, aimed at increasing the performance, are presented in [18], taking into account the use of accumulators for computing the exit condition, as soon as a sufficiently precise result is achieved.

Materials and methods

The object of the research is a method for increasing the performance of solving SLE.

SLE is typically presented in the following way:

$$\begin{gathered} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1; \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2; \\ \cdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n. \end{gathered} \tag{1}$$

An algebraic representation of SLE follows:

$$Ax = p. \tag{2}$$

As mentioned above, SLE may be solved using either direct or iterative methods. Iterative methods are particularly advantageous for computational solutions. A classical iterative method is the Jacobi algorithm, based on the provision of an initial guessed approximate solution x^0, typically expected to be close to the exact solution, and on the computation of the value of each unknown assuming that the remaining unknowns in the current equation have the value from the initial vector x^0. For each equation, the unknowns placed at the diagonal of the matrix A (2) are computed using the following procedure:

$$x_i^{k+1} = \frac{\left(b_i - \sum_{i \neq j}^{n} a_{i,j} x_j^k\right)}{a_{i,j}}. \quad (3)$$

The obtained result is the approximation x^1 to the exact solution. A more accurate solution may be obtained by another computation of the values of each unknown, this time based on the previously computed vector x^1 for the remaining unknowns instead of vector x^0. The process may be repeated iteratively until a solution with an acceptably small error is found. The criteria of an admissible error are usually expressed in relation to the norm of the right-hand vector b. The method converges if, for each row (or column) of matrix A, the absolute value of the diagonal element is greater than that for each individual element in the same row (or column).

Other iterative methods include the Gauss–Seidel method, which takes each newly computed value of the unknowns and immediately uses it for computing the following equation within the same iteration. This typically leads to a faster SLE solving than the Jacobi method. However, from the computational point of view, this approach introduces data dependencies between consecutive equations within each iteration, instead of between consecutive iterations, as in the Jacobi method. Such tight dependencies significantly reduce the degree of parallelism in the algorithm. For this reason, the Jacobi method is often preferred for parallel implementations.

The subject of the research is the Jacobi algorithm for solving SLE. This algorithm was chosen due to its wide use in many applications and its high potential for parallelism.

In this work, the specific case of SLE is examined with the matrix A containing five non-zero diagonals.

$$A = \begin{pmatrix} c_1 & d_1 & \cdots & e_1 & & & & \\ b_2 & c_2 & d_2 & \cdots & e_2 & & & \\ & b_3 & c_3 & d_3 & \cdots & e_3 & & \\ a_4 & \cdots & b_4 & c_4 & d_4 & \cdots & e_4 & \\ & a_5 & \cdots & b_5 & c_5 & d_5 & \cdots & e_5 \\ & & a_6 & \cdots & b_6 & c_6 & d_6 & \cdots \\ & & & a_7 & \cdots & b_7 & c_7 & d_7 \\ & & & & a_8 & \cdots & b_8 & c_8 \end{pmatrix}. \quad (4)$$

The diagonals are arranged symmetrically with outer diagonals located at a distance L from three central diagonals. Thus, SLE may be expressed in the following way:

$$\begin{gathered} c_i x_i + d_i x_{i+1} + e_i x_{i+L} = p_i, \quad i = 1; \\ b_i x_{i-1} + c_i x_i + d_i x_{i+1} + e_i x_{i+L} = p_i, \quad i = 2, \ldots, L; \\ a_i x_{i-L} + b_i x_{i-1} + c_i x_i + d_i x_{i+1} + e_i x_{i+L} = p_i, \quad i = L+1, \ldots, N-L; \\ a_i x_{i-L} + b_i x_{i-1} + c_i x_i + d_i x_{i+1} = p_i, \quad i = N-L+1, \ldots, N-1; \\ a_i x_{i-L} + b_i x_{i-1} + c_i x_i = p_i, \quad i = N. \end{gathered} \quad (5)$$

The approximate solution is found by assigning the initial guessed value of the unknowns and the repeated iteration until the margin of error is acceptable.

Each of five diagonals may be effectively represented computationally as vectors, considered as constant data within the algorithm. The assumed values of the unknowns are variable data, updated gradually through the instructions inside the main inner loop, but are not updated until the end of each iteration.

This provides the possibility of massive parallelism for large systems of equations, due to the absence of read-after-write (RAW) dependencies within the inner loop. Nevertheless, consecutive iterations do present unsolvable RAW dependencies.

The algorithm stops when the solution error is sufficiently small. The following expression presents the condition of acceptable error:

$$\frac{\left\|b - A_x^k\right\|}{\left\|b\right\|} < \varepsilon. \tag{6}$$

The algorithm consists of two basic cyclical steps: the update of the unknowns X and the computation of the error E. Fig. 1 presents the behavior of the algorithm.

This work carries out the analysis of existing implementations, aimed at achieving the best performance, leading to the conclusion that a hardware implementation of the Jacobi iterative method using reconfigurable accelerators is highly relevant.

This work employs the following *research methods*:

- simulation modeling of software and hardware implementations for solving SLE using the Jacobi method;
- comparative analysis of hardware performance and hardware costs.

The goal of the analysis and synthesis of hardware implementation of the Jacobi iterative method is to achieve higher performance mainly through the parallelization of the algorithm.

Fig. 2 shows the parallelism applied to the basic conceptual computing units. For high-level language descriptions, each of the base blocks may be parallelized using the declaration of the type #pragma unroll applied to its loops.

The design may be additionally parallelized using the pipelined processing, i.e., dividing each computationally intensive instruction within the loops into several stages. In this way, intermediate results from each stage are stored in certain registers at each clock cycle, so that the next instruction may immediately start the completion of the stage accomplished by the previous instruction. This concept, similar to the pipelined processing in standard microprocessors, is shown in Fig. 3.

Considering the approximative character of the algorithm, it is possible to define asymmetric workloads, where E is not executed for each newly computed value of X, but only after a given k number of X computations. Consequently, the implementation is more effective when hardware resources are allocated more heavily to computing X than to computing E.

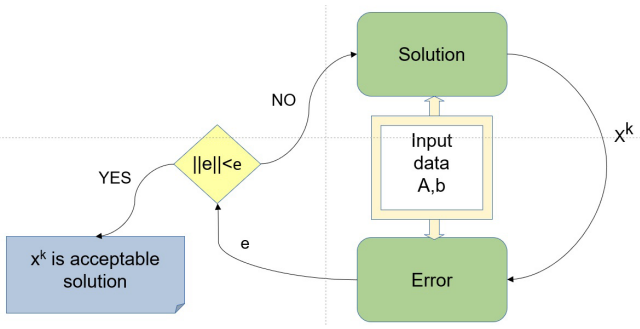

Fig. 1. Behavior of the described algorithm

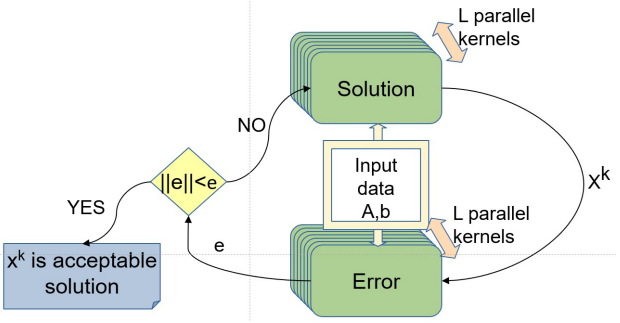

Fig. 2. Parallel approach representation

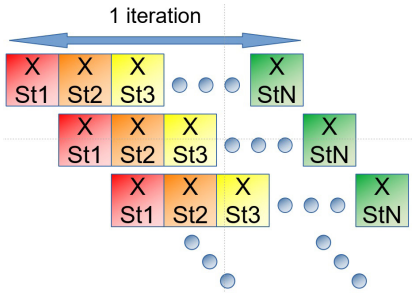

Fig. 3. Pipeline representation

An analysis of several approaches for the implementation of the elementary loops of the algorithm was carried out, in particular, to computing X. Assuming a direct translation of the algorithm to hardware resources, each equation may be represented by a line of code inside the inner loop of the algorithm, each line synthesized into hardware resources performing the required multiplications, subtractions, additions and divisions. Considering that for large problems the size N of the Jacobi matrix is usually much larger than the distance L of the outer diagonal, the central equations (5) dominate in number and thus require more computation. For this reason, as shown in this work, this section benefits most from the maximum achievable parallelism in order to achieve the best possible performance.

This work shows that data organization, the vectors storing the variables and constants of the algorithm in particular, should support higher memory bandwidth in the computing process for the most frequently accessed data. In particular, in order to increase memory bandwidth and the memory replication and adaptation of its structure were used, improving the performance of the implementation.

In order to achieve better performance, several additional optimizations were applied, including the replication of memories storing the most frequently accessed unknowns to alleviate bottlenecks, the parallelization of the accumulator and the preliminary computation of the floating-point divisions and storage of their results into a separate memory.

The goal of the optimization of the hardware solution within this work is the best possible performance considering the limitations imposed by the available resources on the target FPGA device.

The Xilinx Vitis HLS high-level synthesis tool was used for the hardware implementation. The software implementation was executed on a standard computer microprocessor.

Fig. 4 shows a representation of an accumulator. Although its implementation may differ, its functionality is identical to an adder, where the output is fed back to one of its inputs. Thus, its characteristics may be considered identical to those of a standard adder. In particular, for the implementation using the available digital signal processors (DSP), the typical case was considered with the initiation interval

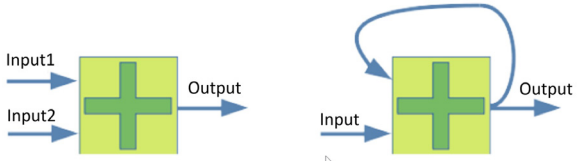

Fig. 4. Logical representation of an adder (left) and an accumulator (right)

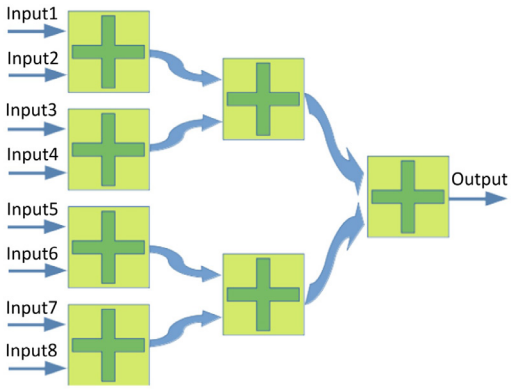

Fig. 5. Completely parallel accumulator

consisting of three cycles. Therefore, for SLE with N variables, the global delay for computing the error norm, required for enabling early completion of the algorithm, is three N cycles.

This computation time may become significantly longer than the actual computation of variables. For this reason, a parallel version of the accumulator is proposed. Fig. 5 shows the necessary structure of an accumulator for fully parallelizing the accumulation of eight values. It can be observed that the structure requires in total $N - 1$ adders and provides a result within $3*|\log_2 N|$ cycles. Therefore, while the performance improvement is considerable, the area requirements also increase substantially.

For this reason, employing a hybrid accumulator is advantageous, as shown in Fig. 6. A pipelined implementation of this approach allows to obtain a result with a speed of $3*(|\log_{2K}| + (N/K - 1))$ cycles, where K is the number of adders (i.e., the area) required by the implementation.

In this work, the accumulator was fully parametrized, and K becomes an arbitrary factor, determined by design choice. In this context, it is advisable to maximize K within the area constraints, determined by the maximum achievable parallelism of the remaining components of the accelerator.

For problems of large size and degrees of parallelism of several tens of computing units, the required memory bandwidth may exceed the capabilities of the platform, which, in our case, is limited to 460 GB/s. For this reason, efficient memory access becomes necessary.

Given the SLE under consideration, each instruction may involve up to five simultaneous accesses to the vector of variables (in the case of the computation of the error, while for the update of the variables up to four read accesses may be required at once).

Regarding the five-diagonal system under study, at each instruction two neighboring variables will be accessed, as well as two distant variables, corresponding to the outer diagonals located at a distance L from the central diagonal.

To the extent allowed by available resources, a window of relevant variables may be stored in internal registers, including neighboring and distant variables for all instructions that may execute in parallel. The variable from the left distant diagonal may be deleted from the registers when it is no longer needed in subsequent computations, releasing resources for new variables from the right distant diagonal.

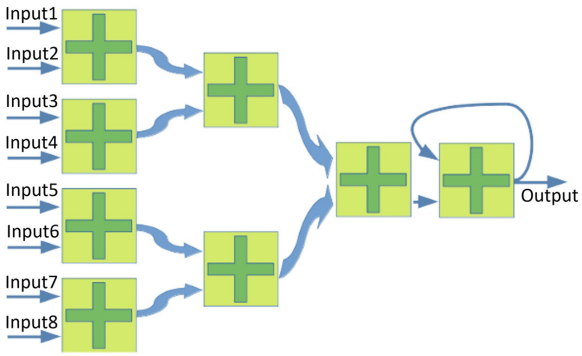

Fig. 6. Hybrid implementation of an accumulator

In this way, the variables may be stored in a cache in a sliding window fashion, so that memory access bandwidth to main memory is reduced: from five (or four) accesses to the memory of variables to one, achieving an approximate 30% reduction in required main memory bandwidth.

In some cases, the problem is so large and the distance L so great, that it's impossible to store all relevant variables in registers. In this case, even the storage of immediate neighbors leads to the reduction of memory bandwidth requirements by roughly 10%. A compromise solution is also possible, achieving an approximate 20% reduction.

Results and discussion

Research on the efficiency of the software and hardware implementations was carried out. The performance of the solution was taken as the criterion of efficiency when comparing results.

Considering that the achievable performance depends on the size of the problem, a number of matrices of different sizes was generated for this research.

Another important factor influencing the performance of the system, considering the iterative nature of the Jacobi method, is the precision required for the approximate solution. The more precise a result should be in order to be considered as acceptable, the more iterations should be executed, with a proportional increase in the processing time. In this work, values of *epsilon* (ε) within the range from 10^{-1} to 10^{-5} were investigated.

The results obtained for the software and hardware implementations show that the hardware implementation generally achieves a performance increase of approximately one order of magnitude compared to the software approach.

The system was described within the Vitis HLS of Xilinx environment[1], and the target device was the Alveo[2] [20] device: xcvu5p-fiva2104-1-e. Experimental results show that the time required to solve large SLE is reduced *by up to two orders of magnitude under given conditions*, compared to the standard software approach, based on a microprocessor.

Conclusion

The hardware implementation of SLE solver using the Jacobi iterative method, developed in this work, targets reconfigurable hardware accelerators and provides better performance than the software-based approach.

Using the resulting solution as a library component allows for increasing the efficiency of computing systems, achieving the best performance, when solving problems requiring the solution of SLE.

[1] Vitis HLS User Guide, Available: https://docs.amd.com/viewer/book-attachment/4lwvWeCi9jb~DWzdfWuVQQ/xRd0KUy2XBYs6mDFnJO6YA-4lwvWeCi9jb~DWzdfWuVQQ (Accessed 13.01.2026)

[2] Alveo U50 Data Center Accelerator Card Data Sheet: Alveo U50 Card Data Sheet (DS965), Available: https://docs.amd.com/r/en-US/ds965-u50 (Accessed 13.01.2026)

In the course of this work, a number of approaches to increasing the degree of parallelism were proposed, including parallelization and pipelining, implemented using high-level synthesis tools.

A further direction of the work is related to the analysis of the efficiency of hardware implementation for SLE solvers using other methods, considering their inherent parallelism.

REFERENCES

1. **Le Fèvre V., Herault T., Robert Y., Bouteiller A., Hori A., Bosilca G., Dongarra J.** Comparing the performance of rigid, moldable and grid-shaped applications on failure-prone HPC platforms. *Parallel Computing*, 2019, Vol. 85, Pp. 1–12. DOI: 10.1016/j.parco.2019.02.002

2. **Ashraf M.U., Alburaei Eassa F., Ahmad Albeshri A., Algarni A.** Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems. *IEEE Access*, 2018, Vol. 6, Pp. 23095–23107. DOI: 10.1109/ACCESS.2018.2823299

3. **Kalyaev I.A., Antonov A.P., Zaborovsky V.S.** Architecture of reconfigurable heterogeneous distributed supercomputer system for solving problems of intelligent data processing in the era of digital transformation of the economy. *Cybersecurity Issues*, 2019, Vol. 33, No. 5, Pp. 2–11. DOI: 10.21681/2311-3456-2019-5-02-11

4. **Dongarra J., Gottlieb S., Kramer W.T.C.** Race to exascale. *Computing in Science & Engineering*, 2019, Vol. 21, No. 1, Pp. 4–5. DOI: 10.1109/MCSE.2018.2882574

5. **Antonov A.P., Besedin D.S., Filippov A.S.** Research and comparative analysis of the effectiveness of software and hardware implementations of transposed matrix multiplication. *Computing, Telecommunications and Control*, 2024, Vol. 17, No. 1, Pp. 44–53. DOI: 10.18721/JCSTCS.17104

6. **Antonov A., Zaborovskij V., Kisilev I.** Developing a new generation of reconfigurable heterogeneous distributed high performance computing system. *Proceedings of International Scientific Conference on Telecommunications, Computing and Control*, 2021, Vol. 220, Pp. 255–265. DOI: 10.1007/978-981-33-6632-9_22

7. **Kastner R., Matai J., Neuendorffer S.** Parallel programming for FPGAs, *arXiv:1805.03648*, 2018. DOI: 10.48550/arXiv.1805.03648

8. **Foertsch J., Johnson J., Nagvajara P.** Jacobi load flow accelerator using FPGA. *Proceedings of the 37th Annual North American Power Symposium*, 2005, Pp. 448–454. DOI: 10.1109/NAPS.2005.1560554

9. **Tamuli M., Debnath S., Ray A., Majumdar S.** Implementation of Jacobi iterative solver in Verilog HDL, *2016 2nd International Conference on Control, Instrumentation, Energy & Communication* (*CIEC*), 2016, Pp. 103–105. DOI: 10.1109/CIEC.2016.7513747

10. **Morris G.R., McGruder R.Y., Abed K.H.** Accelerating a sparse matrix iterative solver using a high performance reconfigurable computer. *2010 DoD High Performance Computing Modernization Program Users Group Conference*, 2010, Pp. 517–523. DOI: 10.1109/HPCMP-UGC.2010.30

11. **Ruan H., Huang X., Fu H., Yang G.** Jacobi solver: A fast FPGA-based engine system for Jacobi method. *Research Journal of Applied Sciences, Engineering and Technology*, 2013, Vol. 23, No. 6, Pp. 4459–4463. DOI: 10.19026/rjaset.6.3452

12. **Pelipets A.V.** Rasparallelivanie iteratsionnykh metodov resheniia sistem lineinykh algebraicheskikh uravnenii na rekonfiguriruemykh vychislitel'nykh sistemakh [Parallelization of iterative methods for solving system linear algebraic methods on reconfigurable computing resources]. *Superkomp'iuternye Tekhnologii* (*SKT-2016*) [*Supercomputer Technologies*], 2016, Vol. 1, Pp. 194–198.

13. **Chekina M.D.** Modification of the implementation of the Jacobi method in simulating superdiffusion of radon on reconfigurable computer systems. *Izvestiya SFedU. Engineering Sciences*, 2021, Vol. 7, Pp. 198–206. DOI: 10.18522/2311-3103-2021-7-198-206

14. **Levin I.I., Dordopulo A.I., Pelipets A.V.** Implementation of iteration methods for solution of linear equation systems in problems of mathematical physics on reconfigurable computer systems. *Bulletin of the*

South Ural State University. Series: Computational Mathematics and Software Engineering, 2016, Vol. 5, No. 4, Pp. 5–18. DOI: 10.14529/cmse160401

15. **Levin I.I., Dordopulo A.I., Pisarenko I.V., Melnikov A.K.** Description of Jacobi algorithm for solution of linear equation system in architecture-independent set@l programming language. *Izvestiya SFedU. Engineering Sciences*, 2018, Vol. 5, Pp. 34–48. DOI: 10.23683/2311-3103-2018-5-34-48

16. **Morris G.R., Abed K.H.** Mapping a Jacobi iterative solver onto a high-performance heterogeneous computer. *IEEE Transactions on Parallel and Distributed Systems*, 2013, Vol. 24, No. 1, Pp. 85–91. DOI: 10.1109/TPDS.2012.121

17. **Pourhaj P., Teng D.H.-Y., Wahid K., Ko S.-B.** System size independent architecture for Jacobi processor. *2008 Canadian Conference on Electrical and Computer Engineering*, 2008, Pp. 002033–002036. DOI: 10.1109/CCECE.2008.4564902

18. **Uguen Y., Dinechin F.** Design-space exploration for the Kulisch accumulator, 2017, Available: https://hal.science/hal-01488916/file/kulisch-acc-2017.pdf (Accessed 23.12.2025)

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Mauricio F. Gonzalez
Гонзалез Мауризио Фаюла
E-mail: fayula.gm@edu.spbstu.ru

Alexander P. Antonov
Антонов Александр Петрович
E-mail: antonov_ap@spbstu.ru
ORCID: https://orcid.org/0000-0002-4107-8950

Submitted: 27.05.2025; Approved: 05.12.2025; Accepted: 25.12.2025.
Поступила: 27.05.2025; Одобрена: 05.12.2025; Принята: 25.12.2025.