Hardware of computer, Telecommunications and Control Systems

Аппаратное обеспечение вычислительных, телекоммуникационных и управляющих систем

Research article
DOI: https://doi.org/10.18721/JCSTCS.17104
UDC 004.312.44

RESEARCH AND COMPARATIVE ANALYSIS OF THE EFFECTIVENESS OF SOFTWARE AND HARDWARE IMPLEMENTATIONS OF TRANSPOSED MATRIX MULTIPLICATION

A.P. Antonov ✉ , *D.S. Besedin*, *A.S. Filippov*

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

✉ antonov@eda-lab.ftk.spbstu.ru

Abstract. The article is devoted to the study and comparative analysis of the software and hardware implementation of the transposed matrix multiplication operation and its modified version, the matrix multiplication transpose. A feature of this study is the use of high-level synthesis tools to obtain and optimize hardware implementations of these operations. The relevance of this study is due to the widespread use of matrix operations, such as transposition and multiplication, to solve various applied problems, the power-law asymptotic complexity of matrix calculations and the lack of data on the effectiveness of using high-level synthesis tools in the tasks of creating hardware devices for matrix calculations. A step-by-step method for synthesizing and optimizing the hardware implementation of these operations is proposed. A comparative study of the software and hardware implementations of these two operations was carried out. It is shown that the gain in performance of hardware implementations is achieved by increasing the degree of parallelism of matrix calculations. Additionally, studies were conducted on the required resources while increasing productivity through parallelization.

Keywords: hardware implementation, performance, hardware costs, FPGA, parallel computing, pipelining

Acknowledgements: State Assignment of SPBSTU FSEG-2022-0001

Citation: 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

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.17104
УДК 004.312.44

ИССЛЕДОВАНИЕ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЭФФЕКТИВНОСТИ ПРОГРАММНЫХ И АППАРАТНЫХ РЕАЛИЗАЦИЙ ТРАНСПОНИРОВАННОГО МАТРИЧНОГО УМНОЖЕНИЯ

А.П. Антонов ✉ *, Д.С. Беседин, А.С. Филиппов*

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

✉ antonov@eda-lab.ftk.spbstu.ru

Аннотация. Статья посвящена исследованию и сравнительному анализу программной и аппаратной реализации операции транспонированного матричного умножения и ее модифицированной версии – операции транспонирования матричного умножения. Особенностью данного исследования является использование высокоуровневых средств синтеза для получения и оптимизации аппаратных реализаций указанных операций. Актуальность данного исследования обусловлена широким использованием матричных операций, таких как транспонирование и умножение, для решения различных прикладных задач, степенной асимптотической сложностью матричных вычислений и отсутствием данных об эффективности использования высокоуровневых средств синтеза в задачах создания аппаратных устройств для матричных вычислений. Предложен пошаговый метод синтеза и оптимизации аппаратной реализации указанных операций. Проведено сравнительное исследование программной и аппаратной реализаций двух указанных операций. Показано, что выигрыш в производительности аппаратных реализаций достигается за счет увеличения степени параллелизма матричных вычислений. Дополнительно были проведены исследования требуемых ресурсов при повышении производительности за счет распараллеливания.

Ключевые слова: аппаратная реализация, производительность, затраты на оборудование, ПЛИС, параллельные вычисления, конвейерная обработка

Финансирование: Государственное Задание СПБПУ FSEG-2022-0001

Для цитирования: 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. Т. 17, № 1. С. 44–53. DOI: 10.18721/JCSTCS.17104

Introduction

Matrix calculations are widely used to solve computational problems of various classes. The asymptotic complexity of matrix calculations has a power dependence on the number of rows/columns of the processed matrix. There are, for example, $O(n^3)$ for matrix multiplication and $O(n^2)$ for matrix addition. Therefore, the task of increasing the performance of computing systems for performing matrix operations is relevant [1, 2].

Matrix calculations are reduced to operations with matrix elements. Such operations are independent of each other. The elements of the resulting matrices are also independent. As a result, spatial parallelization can be implemented for matrix calculations. In addition, independence of resulting matrix elements also allows the use of time parallelization or pipelining. Thus, operations of reading data, arithmetic operations on them and writing the result can be performed simultaneously at different stages of the pipeline processing.

Spatial parallelization of calculations can increase performance by a magnitude of the level of parallelization. Accordingly, pipelining can give an increase in performance that is a magnitude of the pipelining stages. However, such theoretical estimates are overly optimistic. They do not take into account the limitations of the data read and write subsystems bandwidth. They also do not consider the limit of available hardware logical and memory resources [3].

Software approaches used to improve the performance of matrix calculations are well known. There are multi-threaded, multi-core and multi-processor spatial parallelization. These techniques can be implemented on universal processing units that have N physical cores and allow up to N*2 threads. Also, there are graphic cards operating in computing mode. They are called General Purpose Graphic Processing Unit (GPGPU). They have a systolic Single Instruction Multiple Data (SIMD) architecture. Finally, there are Field Programmable Gate Arrays (FPGAs). These are ultra-large integrated hardware reconfigurable microcircuits [4–7].

The possibilities for increasing performance for fixed architecture computers (multi-core/multithreaded processors and GPGPU) are limited by the characteristics of a particular computer. They have a fixed number of computational units and unchangeable connections between them. Also, it has limited and constant amount of local/distributed memory. These restrictions are fixed for a specific computer with a fixed architecture and determine the relationship between the performance of the computer and the type of problem solved on it.

Computers with reconfigurable internal hardware architecture could be adapted to the solving task. They are traditionally implemented on FPGAs. Such computers are largely free from the disadvantages of fixed architectures mentioned above. However, there is one disadvantage of using FPGA accelerators. It is the complexity of the device development procedure. Traditional development methods are based on circuit input description or on the use of hardware description languages (HDL), for example, VHDL, Verilog HDL, SystemVerilog [8–11].

The process of creating an optimized hardware implementation of the algorithm being solved is rather complicated even by using HDL. It requires significant programmer's time for development, debugging and optimization [5]. This often does not allow for research and comparative analysis of different options for hardware implementations of the algorithm for the problem being solved. This leads to hardware solutions that are close in performance to solutions based on computers with fixed architectures [12, 13].

There are a lot of articles demonstrating mentioned above features and disadvantages.

Article [3] describes an attempt to use FPGA to perform operations with matrices and vectors. Analysis and comparison of efficiency with implementations based on Digital Signal Processors (DSP), GPGPU and Application Specific Integrated Circuits (ASIC) are also performed. The conclusion is made about the applicability of FPGA for solving such computing problems. The main disadvantage of using FPGAs is the difficulty of developing a device using hardware description languages (HDL). At the same time, the work did not conduct optimization studies of hardware implementations.

The article [4] compares the performance of FPGA and GPGPU when solving the problem of sparse matrix-vector multiplication. A solution based on CUDA technology is used for GPGPU. The work provides a comparative table of the performance of solving the given problem on GPGPU and on FPGA. This shows that the proposed hardware implementation of the algorithm is slightly more productive than the GPGPU implementation.

The article [5] presents a comparison of the performance of GPGPU, multi-core systems and hardware implementation on FPGA when solving the problem of matrix multiplication. It is shown that maximum performance is achieved for solutions based on GPGPU and FPGA. At the same time, FPGAs showed the best result in terms of energy efficiency.

Article [6] considers two implementations of the same matrix multiplication algorithm on FPGA. Both are created using a traditional design approach. The resulting hardware implementations are comparable in performance to implementations of similar algorithms on multi-core systems.

The paper [7] proposed a pipelined matrix multiplication architecture on FPGA. This device created using circuit input. Particular attention is paid to assessing the performance of data transfer between device submodules. The performance evaluation was compared to a performance evaluation based on MATLAB simulations. The result shows that the created FPGA implementation has similar performance.

Using the high-level synthesis tools is a modern approach to creating a hardware implementation of the algorithm for the problem being solved. High-level synthesis of hardware solutions allows you to write solutions to problems in high-level programming languages, usually C and C++. Leading FPGA manufacturers such as Xilinx and Intel PSG provide such software. Companies that create electronic design tools, such as Mentor Graphics, also already have similar tools. Such tools allow not only to create a basic hardware solution based on an existing algorithm description in C (or C++), but also to conduct research and comparative analysis of various options for hardware solutions. You can easily change the level of parallelism, the number of pipeline stages, memory interfaces, etc. with the help of high-level synthesis tools capabilities [14–16].

The efficiency, productivity and hardware costs of the final result of the hardware implementation of the algorithm for the problem being solved depend on the selected algorithm for solving the problem and on the selected parameters of the high-level synthesis procedure. The procedure for obtaining the optimal final result is not formalized. It is heuristic and requires research using simulation modeling and comparative analysis [17–19].

Object, subject, methods and purpose of the research

The object of research is a method for increasing the performance of matrix calculations. The results of this object of research are presented in this article.

The operation of the multiplication of transposed two-dimensional matrices of size N columns and M rows is the subject of research:

$$C_{N\times M} = B^{T}_{V\times N} * A^{T}_{M\times V},$$

where C is a two-dimensional output matrix of size N×M; A is a two-dimensional input matrix of size M×V; B is a two-dimensional input matrix of size V×N.

The algorithm of the subject of research can be modified with the help of using the known properties of the matrix transposition operation:

$$C_{N\times M} = \left(A_{M\times V} * B_{V\times N}\right)^{T},$$

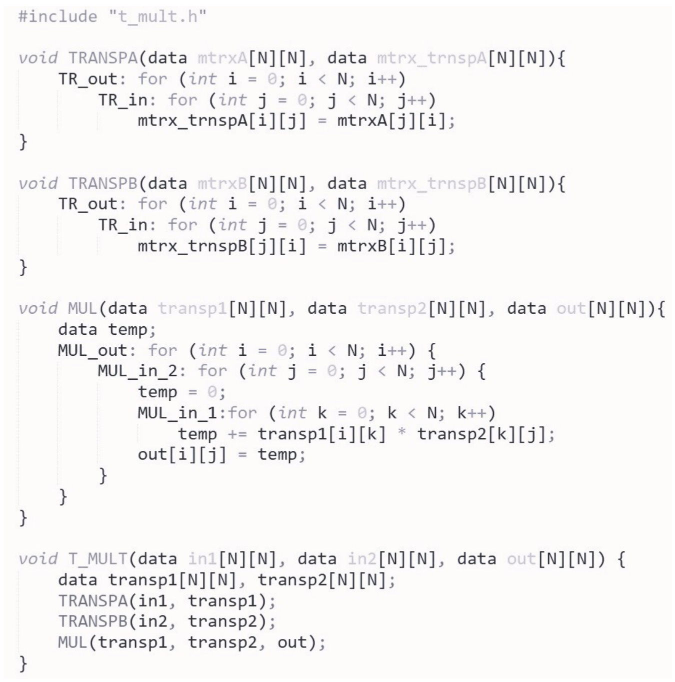
where C is a two-dimensional output matrix of size N×M; A is a two-dimensional input matrix of size M×V; B is a two-dimensional input matrix of size V×N.

A description of the algorithm of the subject of study is shown in Fig. 1, where in1 and in2 are two-dimensional input matrices; out – two-dimensional output matrix. For convenience, the code is written for square matrices. Algorithm has written in C programming language.

The TRANSPA and TRANSPB functions transpose the input matrix. The MUL function calculates the multiplication of two matrices. The T_MUL function solves the problem by calling TRANSPA, TRANSPB and MUL functions sequentially.

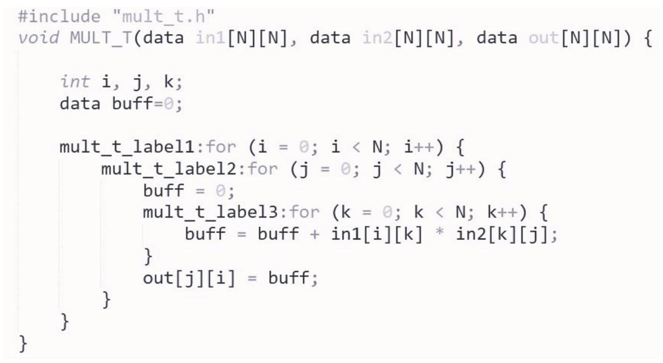
A description of the modified algorithm of the subject of research is shown in Fig. 2, where in1 and in2 are two-dimensional input matrices; out is a two-dimensional output matrix. Modified algorithm has written in C programming language too.

The modified algorithm has different order of performing matrix operations. Transposition is performed when writing the results of the product of two matrices by changing the order of the indices. Therefore, there is no need for buffer arrays used to store intermediate results in this implementation of the algorithm.



```
#include "t_mult.h"

void TRANSPA(data mtrxA[N][N], data mtrx_trnspA[N][N]){
    TR_out: for (int i = 0; i < N; i++)
        TR_in: for (int j = 0; j < N; j++)
            mtrx_trnspA[i][j] = mtrxA[j][i];
}

void TRANSPB(data mtrxB[N][N], data mtrx_trnspB[N][N]){
    TR_out: for (int i = 0; i < N; i++)
        TR_in: for (int j = 0; j < N; j++)
            mtrx_trnspB[j][i] = mtrxB[i][j];
}

void MUL(data transp1[N][N], data transp2[N][N], data out[N][N]){
    data temp;
    MUL_out: for (int i = 0; i < N; i++) {
        MUL_in_2: for (int j = 0; j < N; j++) {
            temp = 0;
            MUL_in_1:for (int k = 0; k < N; k++)
                temp += transp1[i][k] * transp2[k][j];
            out[i][j] = temp;
        }
    }
}

void T_MULT(data in1[N][N], data in2[N][N], data out[N][N]) {
    data transp1[N][N], transp2[N][N];
    TRANSPA(in1, transp1);
    TRANSPB(in2, transp2);
    MUL(transp1, transp2, out);
}
```

Fig. 1. Original form of the transposed matrix multiplication algorithm



```
#include "mult_t.h"
void MULT_T(data in1[N][N], data in2[N][N], data out[N][N]) {

    int i, j, k;
    data buff=0;

    mult_t_label1:for (i = 0; i < N; i++) {
        mult_t_label2:for (j = 0; j < N; j++) {
            buff = 0;
            mult_t_label3:for (k = 0; k < N; k++) {
                buff = buff + in1[i][k] * in2[k][j];
            }
            out[j][i] = buff;
        }
    }
}
```

Fig. 2. Modified form of the transposed matrix multiplication algorithm

Research methods:

• simulation modeling of software and hardware implementations of the matrix operation of the multiplication of two transposed two-dimensional matrices and its modified;

• comparative analysis of device performance and hardware costs;

The research methodology includes the following stages:

• performance assessment of software implementation of the multiplication of two transposed two-dimensional matrices;

• performance assessment of hardware implementation of the multiplication of two transposed two-dimensional matrices;

• performance assessment of software implementation of transposing the multiplication of two two-dimensional matrices;

• performance assessment of hardware implementation of transposing the multiplication of two two-dimensional matrices;

• comparative analysis of estimates obtained during the study;

The purpose of the study is to find the optimal way to implement the matrix operation of the multiplication of two transposed two-dimensional matrices. Choice based on two criteria: performance and hardware costs.

The measure of performance is the time interval after which a new task of the multiplication of two transposed matrices can be launched to compute.

The number of calculating blocks (DSP), the sum of the number of flip-flops (FF) and the number of logic gates (Lookup Tables or LUT) are measures of hardware costs.

The research was carried out for square matrices of size NxN. N was selected from the set: 256, 512, 1024, 2048, 3072. The number of row/column elements was chosen equal to the power of two because this is the maximum number of elements for the corresponding row/column address bit size. The results obtained are practically independent of setting the number of row/column elements not equal to a power of two. The elements of the matrix are signed integers.

A personal computer was chosen as a tool for simulation modeling during software implementation. It has the following characteristics: processor – AMD Ryzen7 4800H 2.9 GHz; RAM – 16 GB, DDR5. Development shell and a compiler for building executable files of simulation software models – Microsoft Visual Studio 2022 Community.

A FPGA unit xcku115-flva1517-2-e by Xilinx was chosen as for hardware implementation. This defines limits on available hardware resources and delays in executing hardware functions.

The tool for synthesizing hardware implementations based on the description of the algorithm in C language was the Vitis HLS 2022.2 software. It allows you to synthesize a device, obtain estimates of the time to solve a problem, and the clock frequency of the device. It also allows you to run simulations to obtain performance estimates.

Procedure and results of the research

The programs described in Fig. 1 and 2 were written in C. They solve the problem of multiplication of transposed matrices by original and modified form. It used to create both hardware and software solutions.

Simulation model was created to conduct a research of software implementation. It does the following for each matrices size:

• It generates two-dimensional square matrices in1 and in2, and fills them with random data;

• It repeatedly runs the functions T_MUL and MUL_T, the algorithms of which are shown in Fig. 1, 2 respectively;

• It evaluates the execution time of the MUL_T and T_MUL functions each time it is run and writes the estimates to the corresponding arrays;

• It finds the median, maximum and minimum time estimates for completing the task being solved;

• It checks the correctness of the T_MUL and MUL_T functions by comparing the results obtained by these functions;

• It displays the minimum, maximum and median execution time estimates for functions T_MUL and MUL_T;

Several solutions have been created for hardware implementations of each of the two variants of the algorithm for the product of two transposed matrices within the framework of the Vitis HLS 2022.2 package. They differ in the level of parallelism, pipelining coefficient and structural organization of data storage memory. The research was carried out and as a result, the period of the clock signal was selected to synchronize the elements of the hardware implementation. Estimates of the performance and hardware costs of the created hardware solution options were obtained.

A comparative analysis of the research results was carried out and the optimal algorithm for implementing the multiplication of two transposed two-dimensional matrices and the method of its implementation were selected, according to the performance/hardware costs criteria.

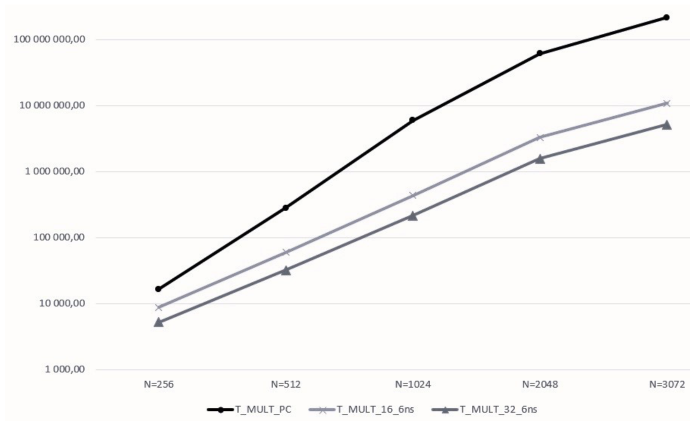


Fig. 3. Performance for the original form of the transposed two-dimensional matrix multiplication algorithm

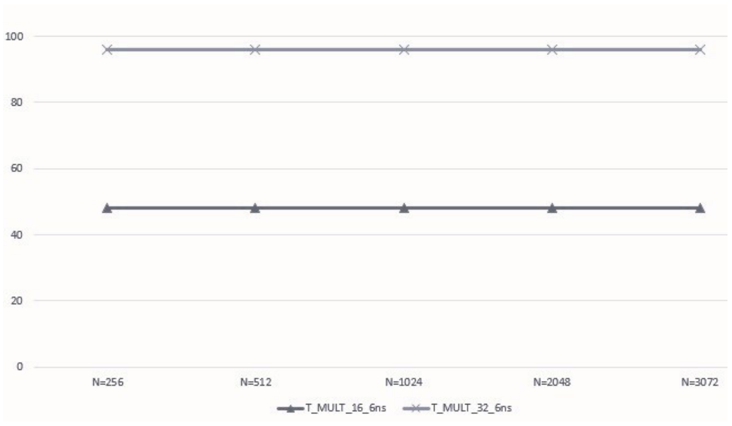


Fig. 4. Hardware costs for the original form of the transposed two-dimensional matrix multiplication algorith

The following Vitis HLS software directives were used while creating a set of hardware solutions:

• PLIS_16_6ns: the level of parallelism of the inner loop is 16, the pipelining coefficient of the inner loop is 3, the input data arrays are structurally transformed into 16 arrays at the second index, the output data arrays are structurally transformed into 16 arrays at the first index. The clock period is set to 6 ns;

• PLIS_32_6ns: the level of parallelism of the inner loop is 32, the pipelining coefficient of the inner loop is 3, the input data arrays are structurally transformed into 32 arrays at the second index, the output data arrays are structurally transformed into 32 arrays at the first index. The clock period is set to 6 ns;

The results of the research of software and hardware implementations of the original algorithm for the multiplication of two transposed two-dimensional matrices are shown in Fig. 3.

The abscissa axis shows the number of columns and rows of the matrices used in the study. The y-axis shows the execution time of the algorithm in microseconds. Y axis was presented in logarithmic scale.

Data analysis showed:

• The solution on PC turned out to be the slowest.
• The PLIS_32_6ns solution turned out to be the fastest.
• Both PLIS solutions much faster than PC solution.

An estimate of hardware costs for the above hardware solutions is shown in Fig. 4.

The y-axis shows the sum of the used logic gates and flip-flops for the corresponding hardware implementation. DSP blocks hardware cost does not depend on matrices size but depends on unrolling level. There are 48 DSP blocks using by T_MULT_16_6ns solution and 96 by T_NULT_32_6ns.

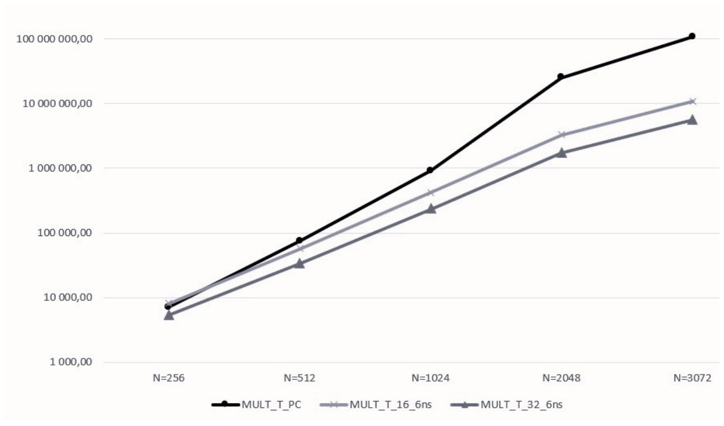


Fig. 5. Performance for a modified form of the transposed 2D matrix multiplication algorithm

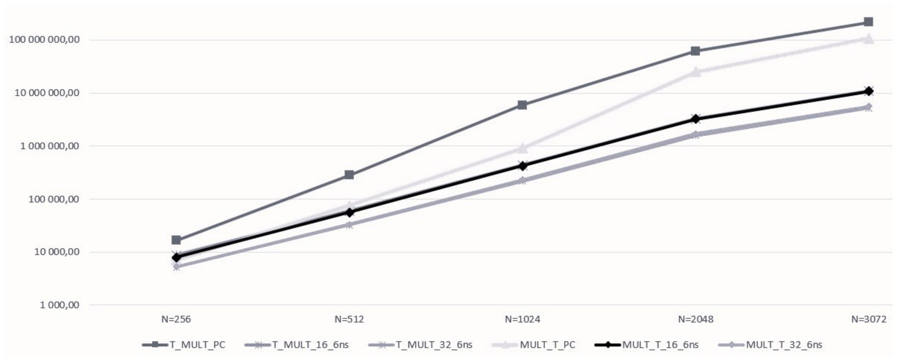


Fig. 6. Result summary

The results of the research of software and hardware implementations of the modified algorithm for the multiplication of two transposed two-dimensional matrices, presented in the form of graphs, are shown in Fig. 5.

The abscissa axis shows the number of columns and rows of the matrices used in the study. The y-axis shows the execution time of the algorithm in microseconds. Y axis was presented in logarithmic scale

Results are similar to previous algorithm. PC solution is the slowest and MULT_T_32_6ns is the fastest.

Estimates of hardware resource costs show a similar figure. 4 the nature of the dependence on the number of rows/columns of the processed matrices, but with smaller amount of it.

A summary of the results obtained as a result of the research is shown in Fig. 6.

The abscissa axis shows the number of columns and rows of the matrices used in the study. The y-axis shows the execution time of the algorithm in microseconds. Y axis in logarithmic scale.

Analysis of the obtained generalized results shows:

- All hardware solutions faster than software solutions.
- PC solution of modified algorithm better than original one.
- Higher unrolling level shows higher performance.
- FPGA solutions of original algorithm a little bit faster than modified ones.

Conclusion

The goal of the research was achieved. We determined a method for implementing the multiplication of two transposed two-dimensional matrices, which provides maximum performance for the given restrictions, and created the corresponding hardware solution T_MUL_32_6ns.

It is shown that the algorithm for implementing the operation of the multiplication of two transposed two-dimensional matrices has a significant impact on the performance of the software solution, but an insignificant impact on the hardware solution.

As further work in this direction, it is advisable to conduct similar studies of other commonly used matrix operations.

REFERENCES

1. **Antonov A.P., Besedin D.S., Filippov A.S.** Research and comparative analysis of the effectiveness of software and hardware implementations of the operation of summing transposed matrices // Computing, Telecommunications and Control. 2022. Vol. 15, no. 4. pp. 51–63. DOI: 10.18721/JCSTCS.15404

2. **Kobayashi R., Oobata Y., Fujita N., Yamaguchi Y., Boku T.** OpenCL-ready high speed FPGA network for reconfigurable high performance computing. Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, 2018, pp. 192–201. DOI: 10.1145/3149457.3149479

3. **Qasim S.M., Telba A.A., Al Mazroo A.** FPGA design and implementation of matrix multiplier architectures for image and signal processing applications. International Journal of Computer Science and Network Security, 2010, no. 10, pp. 168–176.

4. **Zhang Y., Shalabi Y.H., Jain R., Nagar K.K., Bakos J.D.** FPGA vs. GPU for sparse matrix vector multiply. International Conference on Field-Programmable Technology, 2009, pp. 255–262. DOI: 10.1109/FPT.2009.5377620

5. **Tan Y., Imamura T., Mukunoki D.** Design of an FPGA-based matrix multiplier with task parallelism. Parallel Computing: Technology Trends, 2020, vol. 36, pp. 241–250. DOI: 10.3233/APC200047

6. **Kumar V., Joshi S., Patkar S., Narayanan H.** FPGA based high performance double-precision matrix multiplication. International Journal of Parallel Programming, 2010, no. 38, pp. 322–338. DOI: 10.1007/s10766-010-0131-8

7. **Jiang J., Mirian V., Tang K., Chow P., Xing Z.** Matrix multiplication based on scalable macro-pipelined FPGA accelerator architecture. International Conference on Reconfigurable Computing and FPGAs, 2009, pp. 48–53. DOI: 10.1109/ReConFig.2009.30

8. **Abbaszadeh A., Iakymchuk T., Bataller-Mompeán M., Frances-Villora J.V., Rosado A.** Anscalable matrix computing unit architecture for FPGA and SCUMO user design interface. Electronics, 2019, vol. 8. pp. 94. DOI: 10.3390/electronics8010094

9. **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. Smart Innovation, Systems and Technologies. Springer, 2021, vol. 220. DOI: 10.1007/978-981-33-6632-9_22

10. Antonov A., Besedin D., Filippov A. Research of the efficiency of high-level synthesis tool for FPGA based hardware implementation of some basic algorithms for the big data analysis and management tasks. 2020 26th Conference of Open Innovations Association (FRUCT), Yaroslavl, Russia, 2020, pp. 1–7. DOI: 10.23919/FRUCT48808.2020.9087355

11. **Kalyaev I., Antonov A., Zaborovskij V.** 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, pp. 2–11. DOI: 10.21681/2311-3456-2019-5-02-11

12. **Asch M., Moore T., Badia R., et al.** Big data and extreme-scale computing: Pathways to Convergence-Toward a shaping strategy for a future software and data ecosystem for scientific inquiry. The International Journal of High Performance Computing Applications, 2018, vol. 4, No. 32, pp. 435–479. DOI: 10.1177/1094342018778123

13. **Haidar A.** Investigating power capping toward energy-efficient scientific applications. Concurrency and Computation Practice and Experience, 2018, pp. 1—14. DOI: 10.1002/cpe.4485

14. **Antonov A., Zaborovsky V., Polyanskiy V.** Neural computations in control problems: Aspects of computability and spatial-time characterization of cognitive functions. Journal of Physics: Conference Series, 2021, vol. 1864, No. 1. DOI: 10.1088/1742-6596/1864/1/012104

15. **Dongarra J.** Race to exascale. Computing in Science and Engineering, 2019, Vol. 21, no. 1, pp. 4—5. DOI: 10.1109/MCSE.2018.2882574

16. **Harris A.** Exascale models of stellar explosions: Quintessential multi-physics simulation. The International Journal of High Performance Computing Applications, 2021. DOI: 10.1177/10943420211027937

17. **Le Fèvre V.** 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

18. **Usman A., Fathy A., Aiiad A., Abdullah A.** Performance and power efficient massive parallel computational model for HPC heterogeneous exascale systems. IEEE Access, 2018, no. 6, pp. 23095—23107. DOI: 10.1109/ACCESS.2018.2823299

19. **Mantovani F., Calore E.** Performance and power analysis of HPC workloads on heterogeneous multi-node clusters. Low Power Electron, 2018, Vol. 2, No. 8, pp. 1—14. DOI: 10.3390/jlpea8020013

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Antonov Alexander P.
Антонов Александр Петрович
E-mail: antonov@eda-lab.ftk.spbstu.ru

Besedin Denis S.
Беседин Денис Сергеевич
E-mail: nero1310@yandex.ru

Filippov Alexey S.
Филиппов Алексей Семенович
E-mail: alexey.s.filippov@gmail.com

Submitted: 28.02.2024; Approved: 03.04.2024; Accepted: 12.04.2024.
Поступила: 28.02.2024; Одобрена: 03.04.2024; Принята: 12.04.2024.