Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2024, Vol. 3, Issue 1, pp. 33–44
https://doi.org/10.5937/jcfs3-49392
Original research paper

Merging Control-flow and Dataflow Architectures on a Single Chip

Nenad Korolija[1*] and Svetlana Štrbac-Savić[1]

[1] School of Electrical Engineering, University of Belgrade, Serbia; nenadko@etf.rs

[2] Academy of Technical and Art Applied Studies Belgrade, Serbia; svetlanas@viser.edu.rs

* Corresponding author: nenadko@etf.rs

Received: February 23, 2024 • Accepted: March 23, 2024 • Published: April 10, 2024.

Abstract: Computing power rises predominantly by increasing the number of cores in modern processors and the number of processors in cluster and cloud architectures. Along with increasing processing power, high-performance computing requirements also rise. The majority of the computing infrastructure includes control-flow processors that are based on the von Neumann paradigm. On the contrary, the principle of dataflow architectures is based on the data flowing through the already configured hardware. Recent research has proposed hybrid architectures, where both control-flow and dataflow hardware would exist on the same chip die. This article proposes a new hybrid control-flow and dataflow architecture where the control-flow hardware resembles modern graphical cards with thousands of cores and each GPU core has a reasonable amount of dataflow hardware. In this way, the advantages of dataflow architecture are exploited, including faster processing of high-performance computing algorithms and lower power consumption, while the conventional problem of communicating between control-flow and dataflow architectures is minimized. The proposed architecture is tested by analyzing the conjugate gradient method executed on both control-flow and dataflow hardware. The execution of the algorithm is divided onto GPU cores, and the execution of repeated instructions on each GPU core is delegated to the assigned dataflow hardware. The results indicate that it is possible to accelerate the execution of algorithms using the proposed architecture.

Keywords: dataflow; control-flow; GPU; high performance computing.

1. INTRODUCTION

The computing power of computer architectures tends to rise. So does the need for computation. Many high-performance computing algorithms include instructions that are repeatedly executed. A relatively small number of machine instructions that present the implementation of an algorithm may be responsible for almost all of the execution time. Many algorithms iterate over matrices, calculating parameters in finite volumes, e.g., in fluid dynamics. At the same time, the frequencies of processors cannot be increased much further without greatly increasing power consumption and cooling requirements. This has governed the development of high-performance computing architectures.

Although dataflow programming has existed for more than six decades, high-performance computing architectures that are predominantly in use are based on control-flow har-

Citation: N. Korolija, S. Štrbac-Savić, "Merging Control-flow and Dataflow Architectures on a Single Chip," *Journal of Computer and Forensic Sciences*, vol. 3, no. 1, pp. 33–44, 2024, https://doi.org/10.5937/jcfs3-49392.

dware, i.e., based on the von Neumann computing model [1]. Some of the major benefits of control-flow architectures include their ability to execute any of the instructions defined by the architecture in any order and at any time. This way, a programmer can direct the architecture instruction execution, helped by the compiler, which can further optimize the execution of the program. The number of cores in processors rises, as it used to be the case with frequencies in the past. Graphical cards can include thousands of processors, overpowering the computational capabilities of central processing units. This has led to utilizing graphical card processing power, not only for computer graphic algorithms but also for executing algorithms that used to be executed solely on central processing units.

Contrary to the control-flow programming model, dataflow programming assumes data flowing through the hardware. There are software and hardware dataflow architectures. In the case of software dataflow architectures, the hardware is usually based on control-flow principles, but it takes input from the input queue as soon as it can and sends results to the queue dedicated for results. Multiple processing elements work in parallel. Hardware dataflow architectures are configured to execute a single algorithm. They are limited in terms of the number of instructions an algorithm may have, as the size of the dataflow hardware is limited. Further, in order for them to be applicable for multiple algorithms, reconfiguration of the hardware is necessary. For this reason, high-performance computing dataflow architectures are predominantly implemented using FPGAs [2]. In some cases, a dataflow architecture may be produced for a single purpose, e.g., coin mining, as is the case with application-specific integrated circuit (ASIC).

Research available in the open literature provides evidence that dataflow architectures can accelerate many high-performance algorithms while reducing power consumption at the same time. Dataflow hardware is suitable for accelerating algorithms that include the uniform processing of relatively large amounts of data. Examples include simulations for nature-oriented civil engineering [3] and big data and machine learning algorithms [4], but also sorting and other methods for accelerating simulations [5–9].

The goal of this paper is to introduce the architecture and the programming model that achieve faster processing per number of transistors on a chip, as well as lower power consumption. The proposed architecture should be based on both control-flow and dataflow hardware. The first one is capable of executing any instruction in any order, while the second one provides faster execution per transistor on the chip as well as lower power consumption. Further, the control-flow processor should be able to execute high-performance code as well and communicate relatively fast with dataflow hardware.

Control-flow algorithms differ from dataflow algorithms. Certain frameworks are designed to enable conventional programmers to work with dataflow hardware without investing a considerable amount of time in learning new programming languages. Along with proposing a new programming model, it is necessary to provide automatic translation of a source code from control-flow to dataflow so that the best of both control-flow and dataflow hardware is combined. Without automatic translation, a programmer would have to care about multiple constrains related to the dataflow hardware.

The following chapter, Materials and Methods, provides a more in-depth view of dataflow architectures potentials, as well as the definition of the proposed hybrid control-flow and dataflow architecture. The chapter Results presents the algorithm for matrix-vector multiplication for control-flow architectures, analyzes bottlenecks if converted directly into

the dataflow paradigm, provides a dataflow version that eliminates the bottlenecks, and provides results of potential acceleration using the hybrid architecture. The chapter Discussion briefly analyzes the acceleration and the reduction in power consumption, which is followed by the chapter Conclusions.

2. MATERIALS AND METHODS

This paper is based on many assumptions that are built on top of the existing control-flow and dataflow infrastructure, as well as available programming models. The first assumption is that the dataflow hardware can run instructions spread over the surface instead of executing them in order, as is the case with control-flow hardware. This brings better performance in terms of the number of instructions that can be executed in parallel, but the dataflow clock cycle is around 10 times slower compared to the control-flow hardware.

One of the most important assumptions that is made is that the surface of a chip die should be comparable to that of any hardware, i.e., the suggested architecture should not increase the number of transistors needed on a chip drastically.

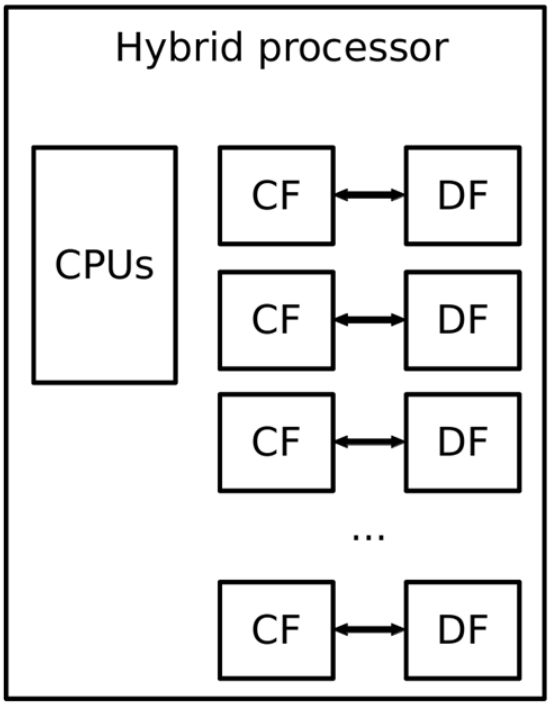

Figure 1. *Proposed computer architecture.*

The dataflow hardware needs to be reconfigured before it is used. The initialization of dataflow hardware is nearly proportional to its size multiplied by the speed of communication between control-flow and dataflow hardware. If the dataflow hardware is split into pieces and spread over the hybrid control-flow and dataflow chip die, the reconfiguring speed can be faster. The speed of reconfiguring each portion of dataflow hardware is estimated to be faster by a factor that is equal to the number of such pieces available on the chip. In cases where there are thousands of pieces of dataflow hardware, we can often neglect dataflow

reconfiguration time, as it is often treated as a background job. This paper assumes a hybrid architecture consisting of many pieces of dataflow hardware available on each control-flow core similar to those from graphics processing units (GPU). This is depicted in Figure 1. Control-flow processors are marked with CF, and dataflow with DF. CPUs represent a number of CPUs that have greater capabilities than those marked with CF and are responsible for synchronizing algorithm execution. These are out of the scope of this paper.

The proposed architecture is judged based on a GPU-executing high-performance computing algorithm with the following characteristics:

1) The processing is performed over the 3D space, divided into finite volumes. The proposed architecture is not limited to this type of processing. The principle is general and applicable to many high-performance computing algorithms.

2) The new state of any elementary volume is calculated based on the previous one. The important aspect is that the new state of any of the elementary volumes depends solely on the previous states of the elementary volume and surrounding elementary volumes.

3) In each iteration over the matrix that presents the simulation volume, the current state is updated to the new one after each elementary volume is processed.

4) The algorithm is run until the change of state is smaller than a given relatively small amount of change between consecutive states or a given number of iterations.

The algorithm execution consists of a processing matrix divided onto GPU cores. Each core is responsible for a portion of the matrix. The processing portion of a matrix requires iterating over a subset of matrix elements, where elements can be processed independently of each other. Each iteration can be executed using the control-flow and dataflow hardware.

High-performance computing requirements direct the development of the underlying hardware [10]. Various algorithms require combining control-flow and dataflow hardware so that the execution time becomes lower than using only a single type of hardware [11]. Although algorithms can be accelerated using the dataflow hardware, programming dataflow hardware requires more effort compared to programming control-flow hardware [12], so automatic translation is needed to enable efficient use of the proposed hardware. Much work has been performed to support programming dataflow hardware. Some of them focus on the availability of dataflow hardware in the cloud as well as providing an integrated development environment for programming [13]. Others tried to provide automatic parallelization of control-flow algorithms onto the dataflow hardware [14, 15]. Many researchers and programmers let others utilize their source code that is left on a common repository [16–18]. When dataflow hardware is separated from the control-flow hardware, the distance between them affects performance [19]. Therefore, it is beneficial to bring these two types of hardware closer to each other in order to utilize both hardware for the execution of a single algorithm. The architecture of cache memory needs to be paid special attention to if the dataflow hardware needs to lie closer to the GPU cores [20, 21]. Cache memories are one of the most promising solutions for the communication between these two types of hardware.

Researchers have been working on defining hybrid architectures that exploit benefits from both control-flow and dataflow architectures [22, 23], where a control-flow architecture is responsible for preparing the data and orchestrating the processing on the dataflow hardware. In some cases, the best architectures of both types are combined on a single chip [24, 25].

The work described in the open literature shows that algorithms can benefit from performance improvements using hybrid architectures [27–28]. However, the acceleration requires special job scheduling algorithms that combine both control-flow and dataflow jobs [29], as the scheduling parameters differ substantially from those of any of the two types of architectures. This is especially important when it comes to cloud processing using hybrid architectures [30]. With the raising possibilities introduced by hybrid architectures, the distance between control-flow and dataflow hardware is reduced, allowing for the wide range of high-performance computing algorithms to be accelerated by splitting each algorithm on relatively small chunks of code, where it would still be justifiable to execute them on different architectures and cover the communication costs [31]. There is a problem with the appearance of recycled integrated circuits on the market. Control-flow, dataflow, and hybrid architectures exhibit the same behavior regarding the aging of the chip, making it possible for any of these types of chips to be detected as recycled using existing methods [32, 33].

The control-flow architecture has the advantage over dataflow architectures in that it can execute any of the instructions defined by the architecture at any moment. On the other hand, dataflow provides faster processing for algorithms with a relatively high amount of repetition in instructions that are executed over and over again, requiring less energy for the computation at the same time. Existing hybrid architectures usually propose integrating control-flow and dataflow hardware by placing them close to each other or utilizing them in clusters or computer clouds.

In this paper, the hybrid architecture is considered, where each relatively small core, similar to a GPU core, is assigned a relatively small amount of dataflow hardware. The goal behind this idea is to enable faster communication between control-flow and dataflow hardware by a factor of a thousand or more. The downside of the proposed approach is that each dataflow hardware can execute a relatively small amount of instructions. The proposed architecture and the programming model can be treated as the Implantation, where a new architecture is invented by implanting a dataflow resource into existing graphics processing units so that the characteristics of the new architecture overcome the characteristics of any of the old architectures [34].

3. RESULTS

The proposed hybrid architecture is compared to the control-flow architecture using the conjugate gradient algorithm [35]. This algorithm contentiously performs certain matrix and vector operations. The most time-consuming one is matrix-vector multiplication. The goal of this research is to demonstrate the ability of dataflow hardware to accelerate the execution of algorithms by placing the most time-consuming portions of the code onto the dataflow hardware. Results are tested for different values of the parameter SIZE from Algorithm 1.

If we analyze the statement that calculates the resulting elements of matrix vector multiplication, we can see only three arithmetic statements: two multiplications and one addition. Theoretically, we can calculate the possibilities of dataflow hardware to accelerate the calculation of a matrix-vector product. Let's assume that around 1000 transistors are needed

on average for implementing a single arithmetic operation and that a processor has one billion transistors and 3300 cores. This means that the dataflow hardware processing should be able to execute 100 cores in parallel. As it is 10 times slower than the control-flow hardware, it would be only 10 times faster. However, practically, dataflow hardware in each clock cycle executes three operations, while control-flow executes only a single one, but 10 times faster. Therefore, dataflow hardware is expected to be slower. This is an important conclusion, supporting the evidence that dataflow programming requires more sophisticated approaches in order to produce the desired acceleration.

```
for(int j = 0; j < SIZE; j++){
    out[j] = 0;
    for(int i = 0; i < SIZE; i++)
            out[j] += alpha * A(j, i) * b(i);
}
```

Algorithm 1. *Matrix-vector multiplication.*

One way to solve the problem of a relatively small number of machine instructions per iteration is to perform loop unwinding. Loop unwinding represents a method to consecutively execute statements of a loop, eliminating the need for a loop while the results of execution remain unchanged. This is possible for loops with a known number of iterations. This process creates more instructions that can be run in parallel.

Another important aspect of dataflow programming is that iterations should be independent from each other in order to avoid stalls in execution. Algorithm 1 calculates in the inner for loop the value of a single element out[j] in each consecutive iteration. This fact can present a problem, as one could have to wait for all three operations to be finished before being able to sum out[j] with the result. More precisely, in this particular case, it would be possible to perform them in parallel in this scenario since out[j] is only written to and in a single dataflow clock cycle. This means that calculating two products can be performed in parallel for successive iterations, and the appropriate result should be stored in the appropriate out[j]. This requires that additional buffers may be needed for the implementation so that two consecutive iterations would not affect each other.

More appropriate source code would be obtained if rearranging of for loops is performed, as shown in Algorithm 2, because dependencies between consecutive statements are eliminated.

```
int j;
for(j = 0; j < SIZE; j++)
    out[j] = 0;
for(int i = 0; i < SIZE; i++){
    for(j = 0; j < SIZE; j++)
            out[j] += alpha * A(j, i) * b(i);
}
```

Algorithm 2. *Rearranged matrix-vector multiplication*

After rearranging the computation, we can perform loop unwinding, as shown in Algorithm 3. The inner for loop has disappeared, as it is replaced with consecutive statements, one for each iteration.

```
int j;
for(j = 0; j < SIZE; j++)
    out[j] = 0;
for(int i = 0; i < SIZE; i++){
    out[0] += alpha * A(0, i) * b(i);
    out[1] += alpha * A(1, i) * b(i);
    out[2] += alpha * A(2, i) * b(i);
    ...
    out[SIZE-1] += alpha * A(SIZE-1, i) * b(i);
}
```

Algorithm 3. *Unwound rearranged matrix-vector multiplication.*

At this stage, we could estimate the acceleration of the dataflow hardware for performing matrix-vector multiplication. Formulas 1–4 show how many cycles are needed for a GPU core and the dataflow hardware assigned to it to execute an inner for loop (unwound in the case of dataflow hardware), where tCF represents the time needed for a GPU core to execute the portion of the algorithm, tCF the time needed for dataflow hardware to execute the same portion of algorithm, tCycleCF and tCycleDF are the time of a clock cycle of dataflow and control-flow hardware, nOperationsCF is the number of statements in the source code if each arithmetic operation is considered a separate statement, and nInstructionsDF is the number of instructions that dataflow hardware has to execute.

$$tCF = tCycleCF * nCyclesCF \quad (1)$$
$$tDF = tCycleDF * nCyclesDF \quad (2)$$
$$nOperationsCF = 5 * SIZE \quad (3)$$
$$nCyclesDF = nInstructionsDF = SIZE \quad (4)$$

However, in the case of control-flow hardware, a value of A(j, i) cannot be fetched in a single cycle. More precisely, a position in a memory can be determined as the sum of the beginning address of matrix A and a multiplication of j and SIZE increased by i. If fetching a single value from a memory takes one cycle, a total of four cycles are needed for fetching A, j, i, and SIZE. Further, an additional three cycles are needed for the arithmetic operations involved, and an additional six for storing and reading the temporary values of multiplying j and SIZE, adding the product to i, multiplying the address offset with the size of a word in the number of bytes, and summing the previous sum with the beginning memory address of A. Fetching b(i) takes at least four cycles. This means that a minimum of 21 cycles is needed for control-flow hardware to execute the C++ statement that involves A(j, i). Similarly, another three clock cycles are needed for increasing i by one, three for checking whether the condition of the inner for loop is met, and two for jumping on the first statement of the for loop if the condition is met. In total, there are at least 29 cycles needed for a single iteration, leading to Formula 5.

$$nCyclesCF = 29 * SIZE \quad (5)$$

From formulas 1 and 2, the acceleration factor of the dataflow hardware is 2.9, keeping in mind that the tCycleCF is approximately 10 times lower than the tCycleDF. Formulas 6–7 show how many cycles are needed for a GPU core and the dataflow hardware assigned to it to execute a matrix-vector product.

$$nCyclesDF = SIZE * (SIZE + 2) \quad (6)$$

$$nCyclesCF = (29 * SIZE + 2) * SIZE \quad (7)$$

The total acceleration factor of the dataflow hardware can be calculated as a factor between nCyclesDF and nCyclesCF, multiplied by 10. Figure 2 depicts the acceleration factor of dataflow hardware for the parameter SIZE in a range from 1 to 100. The acceleration tends to reach the value of 2.9, as the SIZE rises.

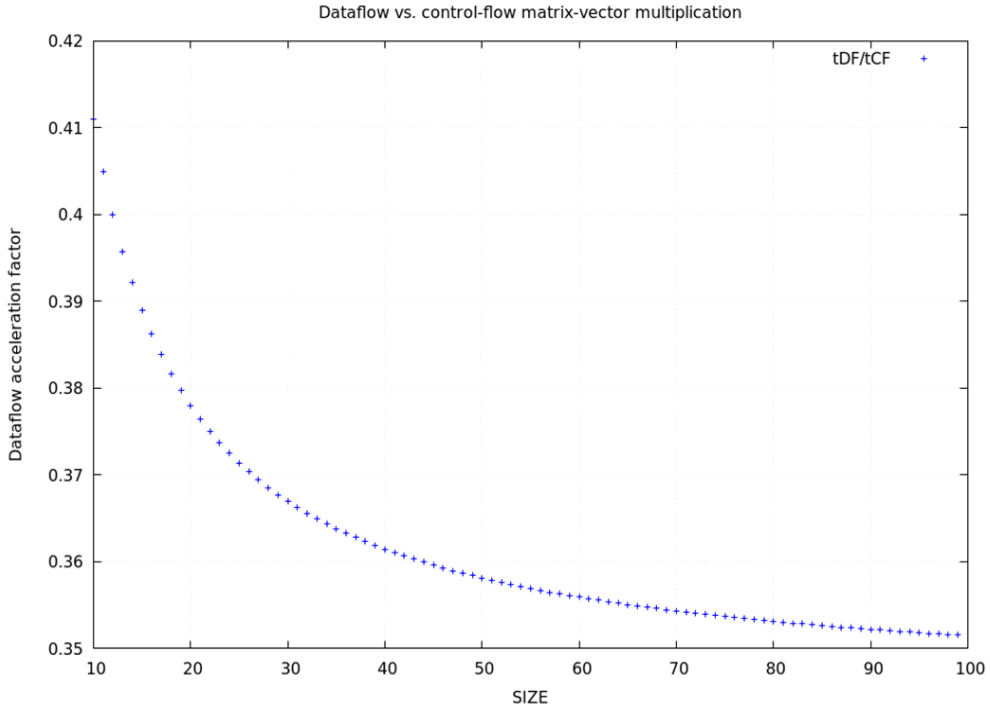

Figure 2. *Acceleration factor of the dataflow hardware for matrix-vector multiplication.*

4. DISCUSSION

As the time needed for the dataflow hardware is lower by a factor higher than two, we can assume that dividing the available transistors on the chip into control-flow and dataflow hardware would result in accelerating the algorithm. At the same time, the power consumption is expected to be lower as the dataflow hardware operates at a lower frequency while the total algorithm execution time is shorter.

Further research is needed for the development of the proposed hybrid control-flow and dataflow hardware. The precise timing constraints should be evaluated based on the im-

plemented architecture. The industry can benefit from merging control-flow and dataflow paradigms, but the problem of converting control-flow to dataflow algorithms remains. As already stated, there are methods to automate this process, but further development of these methods should be directed toward increased use of dataflow architectures.

5. CONCLUSIONS

This paper proposes the use of both control-flow and dataflow hardware on a single chip die by spreading the dataflow hardware over the chip so that each core, similar to GPU cores, can have separate dataflow hardware. The proposed algorithm is tested analytically using the conjugate-gradient method, whose most time-consuming part in total execution time is related to matrix-vector multiplication. The results indicate that the proposed architecture can be effectively used for accelerating the execution of certain high-performance computing algorithms while reducing total power consumption. Future work should show the possibility of utilizing a hybrid processor that combines GPU cores that do not have dataflow hardware assigned to them and GPU cores with dataflow hardware. This requires more sophisticated scheduling algorithms to be developed.

FUNDING
This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT:
Not applicable.

INFORMED CONSENT STATEMENT:
Not applicable.

CONFLICTS OF INTEREST:
The author declares no conflict of interest.

REFERENCES

[1] I. I. Arikpo, F. U. Ogban, and I. E. Eteng: Von Neumann architecture and modern computers. Global Journal of Mathematical Sciences, 6(2), 97–103 (2007).

[2] V. Milutinović, J. Salom, N. Trifunović, and R. Giorgi: Guide to dataflow supercomputing. Springer Nature, 10, 978–3 (2015).

[3] Z. Babović, B. Bajat, V. Đokić, F. Đorđević, D. Drašković, N. Filipović, et al.: Research in computing-intensive simulations for nature-oriented civil-engineering and related scientific fields, using machine learning and big data: an overview of open problems. Journal of Big Data, 10(1), 1–21 (2023).

[4] Z. Babović, B. Bajat, D. Barac, V. Bengin, V. Đokić, F. Đorđević, et al.: Teaching computing for complex problems in civil engineering and geosciences using big data and machine learning: synergizing four different computing paradigms and four different management domains. Journal of Big Data, 10(1), 89 (2023).

[5] N. Korolija, T. Djukic, V. Milutinovic, and N. Filipovic: Accelerating Lattice-Boltzman Method using Maxeler DataFlow Approach. Transactions on Internet Research, 9(2), 5–10 (July 2013).

[6] S. Stojanovic, D. Bojic, and V. Milutinovic: Solving Gross Pitaevskii Equation using Dataflow Paradigm. Transactions on Internet Research, 9(2), (July 2013).

[7] A. Kos, V. Rankovic, and S. Tomazic: Sorting Networks on Maxeler Dataflow Supercomputing Systems. Advances in Computers, 96, 139–186. Amsterdam, Elsevier, Academic Press (2015).

[8] I. Stanojevic, V. Senk, and V. Milutinovic: Application of Maxeler Dataflow Supercomputing to Spherical Code Design. Transactions on Internet Research, 9(2), 1–4 (July 2013).

[9] N. Bezanic, J. Popovic-Bozovic, V. Milutinovic, and I. Popovic: Implementation of the RSA Algorithm on a DataFlow Architecture. Transactions on Internet Research, 9(2), 11–16 (July 2013).

[10] M. J. Flynn, O. Mencer, V. Milutinovic, G. Rakocevic, P. Stenstrom, R. Trobec, and M. Valero: Moving from petaflops to petadata. Communications of the ACM, 56(5), 39–42 (2013).

[11] V. Milutinović, B. Furht, Z. Obradović, and N. Korolija: Advances in high performance computing and related issues. Mathematical problems in engineering (2016).

[12] J. Popovic, D. Bojic, and N. Korolija: Analysis of task effort estimation accuracy based on use case point size. IET Software, 9(6), 166–173 (2015).

[13] N. Korolija and A. Zamuda: On Cloud-Supported Web-Based Integrated Development Environment for Programming DataFlow Architectures. In Exploring the DataFlow Supercomputing Paradigm, New York: Springer Cham, pp. 41–51 (2019).

[14] N. Korolija, J. Popović, M. Cvetanović, and M. Bojović: Dataflow-based parallelization of control-flow algorithms. Advances in computers, 104:73–124, Elsevier (2017).

[15] V. Milutinovic, J. Salom, D. Veljovic, N. Korolija, D. Markovic, and L. Petrovic: Transforming applications from the control flow to the dataflow paradigm. Dataflow Supercomputing Essentials, New York: Springer Cham, 107–129 (2017).

[16] N. Trifunovic, V. Milutinovic, N. Korolija, and G. Gaydadjiev: An AppGallery for dataflow computing. Journal of Big Data, 3(1), 1–30 (2016).

[17] N. Trifunovic, B. Perovic, P. Trifunovic, Z. Babovic, and A. R. Hurson: A novel infrastructure for synergistic dataflow research, development, education, and deployment: the Maxeler AppGallery project. Advances in Computers, Elsevier, 106:167–213 (2017).

[18] V. Milutinovic, J. Salom, D. Veljovic, N. Korolija, D. Markovic, and L. Petrovic: Maxeler AppGallery Revisited. DataFlow Supercomputing Essentials: Research, Development and Education, 3–18 (2017).

[19] R. Trobec et al.: Interconnection networks in petascale computer systems: A survey. ACM Computing Surveys (CSUR), 49(3), 1–24 (2016).

[20] V. Milutinovic, M. Tomasevic, B. Markovic, and M. Tremblay: A new cache architecture concept: the split temporal/spatial cache. Proceedings of 8th Mediterranean electrotechnical conference on industrial applications in power systems, computer science and telecommunications (MELECON 96). Vol. 2. IEEE (1996).

[21] V. Milutinovic, B. Markovic, M. Tomasevic, and M. Tremblay: The split temporal/spatial cache: A complexity analysis, In Proceedings of the SCIzzL, Vol. 6, pp. 89–96 (September 1996).

[22] F. Yazdanpanah, C. Alvarez-Martinez, D. Jimenez-Gonzalez, and Y. Etsion: Hybrid dataflow/von-Neumann architectures. IEEE Transactions on Parallel and Distributed Systems, 25(6), 1489–1509 (2013).

[23] D. Miladinović, M. Bojović, V. Jelisavčić, and N. Korolija: Hybrid Manycore Dataflow Processor, Proceedings, IX International Conference IcETRAN, Novi Pazar, Serbia, June 6–9 (2022).

[24] V. Milutinović et al., The ultimate dataflow for ultimate supercomputers-on-a-chip, for scientific computing, geo physics, complex mathematics, and information processing. 10th Mediterranean Conference on Embedded Computing, IEEE, pp. 1–6 (June 2021).

[25] V. Milutinović, M. Kotlar, I. Ratković, N. Korolija, M. Djordjevic, K. Yoshimoto, and M. Valero: The Ultimate Data Flow for Ultimate Super Computers-on-a-Chip. Handbook of Research on Methodologies and Applications of Supercomputing, IGI Global, 312–318 (2021).

[26] J. Popović, V. Jelisavčić, and N. Korolija: Hybrid Supercomputing Architectures for Artificial Intelligence: Analysis of Potentials. In 1st Serbian International Conference on Applied Artificial Intelligence (SICAAI), Kragujevac, Serbia (2022).

[27] V. Milutinović, N. Trifunović, N. Korolija, J. Popović, and D. Bojić: Accelerating program execution using hybrid control flow and dataflow architectures. In 2017 25th Telecommunication Forum (TELFOR), pp. 1–4, IEEE (November 2017).

[28] L. Egharevba, S. Kumar, H. Amini, M. Adjouadi, and N. Rishe: Detecting and Removing Clouds Affected Regions from Satellite Images Using Deep Learning. IPSI Bgd Transactions on Internet Research, 19(2), 13–23 (July 2023).

[29] N. Korolija, D. Bojić, A. R. Hurson, and V. Milutinovic: A runtime job scheduling algorithm for cluster architectures with dataflow accelerators. Advances in computers, Elsevier, 126 (2022).

[30] K. Milfeld and N. Korolija: Towards hybrid supercomputing architectures. Journal of Computer and Forensic Sciences, 1(1), 47–54 (2022).

[31] M. Popović, N. Korolija, and S. Štrbac-Savić: Hybrid control-flow and dataflow processor: algorithm granularity analysis. In Zbornik 29. konferencije YUINFO (2023).

[32] K. Huang, Y. Liu, N. Korolija, J. M. Carulli, and Y. Makris: Recycled IC detection based on statistical methods. IEEE transactions on computer-aided design of integrated circuits and systems, 34(6), 947–960 (2015).

[33] K. Huang, Y. Liu, N. Korolija, J. M. Carulli, and Y. Makris: Statistical Methods for Detecting Recycled Electronics: From ICs to PCBs and Beyond. IEEE Design & Test. (2023).

[34] V. Blagojević, D. Bojić, M. Bojović, M. Cvetanović, J. Đorđević, Đ. Đurđević, et al.: A systematic approach to generation of new ideas for PhD research in computing. Advances in computers. 104, 1–31. Elsevier (2017).

[35] T. Lebailly, Conjugate gradient algorithm implementation: https://github.com/tileb1/CG-CUDA/blob/master/sequential.c, visited on February 16th, 2024.