ReCoTOS: A Platform for Resource-sparing Computing Task Optimization

Jānis Kampars[1][a], Guntis Mosāns[1][b], Jānis Zuters[2][c], Aldis Gulbis[3] and Rasa Gulbe[3]
[1]*Institute of Information Technology, Riga Technical University, 10 Zunda krastmala, Riga, Latvia*
[2]*Faculty of Computing, University of Latvia, 19 Raiņa bulvāris, Riga, Latvia*
[3]*Dati Group, 80 A Balasta dambis, Riga, Latvia*

Keywords: Software Optimization, Vectorization, SIMD.

Abstract: To cope with the growing volume of data and complexity of the associated processing logic, modern CPU capabilities such as vector registers and SIMD (Single Instruction Multiple Data) instructions need to be taken advantage of. Although from a technical point of view, usage of SIMD instructions is not complicated, building computing tasks with good SIMD capabilities has always been a challenging task. Modern compilers assist developers to some extent with solutions like Compiler Automatic Vectorization, which is not always sufficient, and several researchers demonstrate that manual code optimization is still necessary. The paper gives an overview of the existing computing task optimization approaches, designs and describes development of a cloud-based software optimization platform and demonstrates its usage by optimizing a software correlator.

1 INTRODUCTION

The data volume and complexity of computing tasks are constantly increasing, requiring more power, resources, and energy (Rong, Zhang, Xiao, Li, & Hu, 2016). It is also driven by the IoT phenomena, due to which the number of connected devices and potential data sources has grown rapidly. Some sources suggest that the number of connected IoT devices will reach 50 billion in the near future (Marjani et al., 2017). One of the possible solutions is the horizontal and vertical scaling of the computing infrastructure, however, the approach is associated with several shortcomings. There are certain limits to vertical scalability, and not all computing tasks are built in a way that they can be scaled horizontally. It may not always be technologically possible, especially in the case of edge computing (Ren, Guo, Xu, & Zhang, 2017). It may also be unfeasible from an economic perspective, especially due to the recent increase in the cost of electricity.

Another approach to solving the problem of computing capacity deficiency is related to the computing task itself. It involves code refactoring or optimization activities allowing developers to get better performance with less resources. The inability of computing tasks to take complete advantage of the CPU has become particularly topical since the beginning of the 21st century, when CPU manufacturers have shifted their focus from growth of clock speed towards more sophisticated solutions like increasing the number of cores, adding larger vector registers, and Single Instruction Multiple Data (SIMD) capabilities. Although modern compilers assist developers to some extent with solutions like Compiler Automatic Vectorization (CAV), this is not always sufficient. Several researchers conclude that manual code optimization is superior to CAV (Amiri & Shahbahrami, 2020; Watanabe & Nakagawa, 2019). Some of the reasons for CAV to fail are conservative dependency analysis and behaviour for handling of boundary cases, data layouts that do not allow the contiguous memory accesses needed for SIMD (Holewinski et al., 2012).

In order to address the challenges associated with computing task optimization, we have previously

[a] https://orcid.org/0000-0003-0045-5593
[b] https://orcid.org/0000-0001-9373-4000
[c] https://orcid.org/0000-0002-3194-9142

Kampars, J., Mosāns, G., Zuters, J., Gulbis, A. and Gulbe, R.
ReCoTOS: A Platform for Resource-sparing Computing Task Optimization.
DOI: 10.5220/0010977100003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 251-258
ISBN: 978-989-758-568-5; ISSN: 2184-4895

proposed a methodology and demonstrated its usage (Kampars et al., 2020). The goal of this paper is to define a cloud-based platform that facilitates the optimization of computing tasks according to the aforementioned methodology.

The paper is structured as follows. Section 2 provides an overview of the computing task optimization approaches while focusing on the vectorization and SIMD instructions. Section 3 gives a brief overview of the ReCoTOS optimization methodology. Section 4 designs the platform according to the previously defined methodology. Section 5 provides a computing task optimization use case that demonstrates the functionality of the platform. Section 6 concludes with final remarks and directions for future research.

2 COMPUTING TASK OPTIMIZATION APPROACHES

Although from a technical point of view, usage of SIMD instructions is not complicated, building computing tasks with good SIMD capabilities has always been a challenging task (Amiri & Shahbahrami, 2020). Some of the applicable optimization approaches are summarized in the following subsections.

2.1 Assembly Level Programming

The resource-intensive sections of the computing task requiring optimization are implemented in the assembly language in the form of modules (Amiri & Shahbahrami, 2020; Cockshott & Renfrew, 2004). Variables are moved between the assembler and a high-level programming language during run-time. Good knowledge of both the assembly and high-level programming language is required. Due to the low level of abstraction, the maintainability of the code will suffer and moving between different CPU architectures will be complex.

2.2 Intrinsic Programming Model

Intrinsic programming model (IPM) relies on assembly-type instructions, which are added in the source code of a high-level programming language. The IPM instructions correspond to a specific SIMD instruction that the compiler will further use to vectorize the code. The code is more readable; no in-depth assembly knowledge is required, but the IPM instructions are specific to the processor microarchitecture, which is why portability of the code suffers. The use of IPM is also specific to the compiler version (Hughes, 2015). The programming model provides the most significant increase in performance compared to other approaches according to some researchers (Nuzman & Zaks, 2008).

2.3 Pragma Syntax

Pragma syntax allows one to insert instructions for the compiler in the source code of a high-level programming language. Such instructions could be inserted before the start of a loop or other logical block, instructing the compiler that vectorization should occur. The compiler may choose to ignore these instructions, which is why performance gains are not guaranteed (Hughes, 2015).

2.4 Compiler Automatic Vectorization

CAV provides indirect code vectorization, without the need to make any changes in the code. This allows the developers to concentrate on the business logic and to worry less about specific microprocessor architectures and SIMD instruction sets. CAV avoids manual code optimization, provides easier-to-read and maintain code portability; however, CAV is not able to successfully vectorize all computing tasks (Watanabe and Nakagawa, 2019).

2.5 External Libraries and Parallel Programming Standards

Programmers can opt for the integration of specialized high-performance libraries into their computing tasks. If the standardized operations such as matrix multiplication or Fourier transformations are required, it is possible to find a high-performance library which provides this operation in highly optimized manner.

Frameworks like OpenCL (Stone, Gohara, & Shi, 2010) allow one to write portable and parallelizable computing tasks relying on standardized parallelization operations. These operations are implemented in such a way that they can take advantage of specific processor microarchitecture while requiring no changes to the source code of the original computing task. OpenCL programs can also be executed on Graphical Processing Units (GPUs). The advantages of the approach are portability and performance at a higher level of abstraction, while the disadvantage is that not all standardized parallelization operations are executed equally fast on all technical platforms due to different architectural constraints and their specific instruction mappings (Lai, Luo, & Xie, 2019).

3 OPTIMIZATION METHODOLOGY OVERVIEW

Availability of various optimization methods presents a difficult choice for the developers. Typical development environments and code management systems do not provide support to evaluate and to select the most suitable approach and lack methodological support.

To address this a platform is developed on the basis of a methodology for vectorization-based resource-saving computing task optimization (Kampars et al., 2020). The methodology consists of concepts, technologies, and optimization process thus facilitating more efficient use of modern CPU vectorization capabilities and gaining the associated performance improvements.

The methodology guides the optimization of computing tasks (application or a component of an application that can be logically isolated and executed separately). Initially, valid results of the computing task are recorded, and a performance baseline is established. The necessary performance improvement is defined, and the optimization starts with performing computing task level optimizations that do not require any changes to the source code. An example of such optimization is to choose the best compiler version and its flags.

Further methodology reviews computing task kernels (certain functionality returning a result or change of kernel's internal state according to the received input) and identifies hot kernels (the ones consuming a significant proportion of resources). Static and dynamic analysis of the computing task is used for this purpose. Each of the identified hot kernels can be optimized as part of an optimization iteration.

Optimization iterations are started by the possible introduction of external libraries or replacement of existing libraries with ones providing better performance. This is proposed as the first step since this method does not affect the portability of the code, can have great impact on the performance, and is simpler to implement compared with other strategies.

The next method is concerned with introducing structural changes in the source code (modifying the source of a computing task or its hot kernel to achieve better performance) and lastly IPM is applied. IPM is the least preferred method since it is harder to implement and negatively affects portability and maintainability of the codebase.

After each of the optimizations, the results of the computing task and kernel being optimized are recorded and checked against the previously recorded ones. Performance also needs to be measured to see if the necessary performance improvement has been reached. If the performance improvement after an optimization is negative or if it is associated with maintainability degradation that outweighs the performance benefits, the changes are reverted, and another iteration is started. While in case of the positive results, the changes are saved, and it is determined if further optimization is required and what would be the most appropriate actions for that.

4 DESIGN OF ReCoTOS PLATFORM

The architecture of the platform is shown in Figure 1 and its components are detailed in the following subsection.

4.1 Platform Functional Components

The ReCoTOS core is used as a mediator in the execution of various asynchronous processes related to the optimization of computational tasks, as well as to ensure the creation of appropriate containers by interacting with the Kubernetes orchestration system.

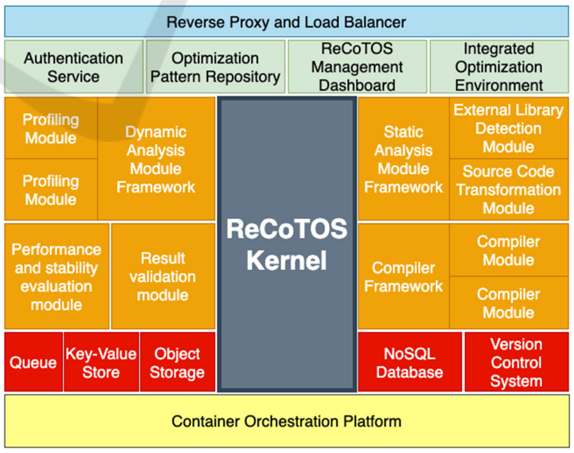

Reverse Proxy and Load Balancer
Authentication Service
Optimization Pattern Repository
ReCoTOS Management Dashboard
Integrated Optimization Environment
Profiling Module
Profiling Module
Dynamic Analysis Module Framework
ReCoTOS Kernel
Static Analysis Module Framework
External Library Detection Module
Source Code Transformation Module
Performance and stability evaluation module
Result validation module
Compiler Framework
Compiler Module
Compiler Module
Queue
Key-Value Store
Object Storage
NoSQL Database
Version Control System
Container Orchestration Platform

Figure 1: ReCoTOS platform architecture.

The ReCoTOS core allows for creation of loosely coupled components by acting as a mediator between them. Multiple instances of the ReCoTOS core can be run simultaneously to ensure horizontal scalability and fault tolerance. It is implemented in the NodeJS, which is also used in the implementation of several other ReCoTOS components.

The purpose of the reverse proxy component is to redirect the request received from the browser to the appropriate platform component, hiding the implementation details from the platform user and providing a homogeneous and integrated environment. It is also used to implement user authentication and access control for components that do not include such functionality. The component includes load balancer functionality, which allows to distribute requests between multiple service instances, thus contributing to scalability and fault tolerance. It is integrated with the Redis key-value store to ensure its reconfiguration during runtime. While reviewing options such as Nginx, Traefik and OpenResty, the latter one was found to be the most suitable. OpenResty is a web platform that integrates an enhanced core of the open-source version of Nginx, a Lua language programming module and compiler, as well as Nginx modules developed by third parties. Lua programming language support allows creation of dynamic configurations which can be integrated with a key-value stores such as Redis.

The platform needs to provide identity and access management, which is achieved by using KeyCloak. The module is integrated with the Reverse proxy server and load balancer verifying that the client has obtained a valid JSON Web Token (JWT) object and has access to the appropriate resource. The module secures user-facing microservices such as Optimization pattern repository, ReCoTOS management dashboard, and Integrated optimization environment. Session information is being stored in the Redis key-value store.

The Optimization pattern repository is designed to store reusable best practices for optimizing computing tasks according to the previously defined methodology (Kampars et al., 2020). A pattern will consist of a description of the context and the corresponding solution, including versioned code samples and measured performance improvements. To store the original code with possibly multiple rounds of edits leading to the final optimization, the module is integrated with the Version control system, while pattern related metadata is stored in the NoSQL database. To make the patterns runnable and easily reusable, integration with the Integrated optimization environment is also provided. The Optimization pattern repository is implemented by using the NodeJS Express and VueJS frameworks.

The ReCoTOS management panel is used for user authorization, management of optimization projects, and navigating between the platform's user-facing modules. It is built based on the NodeJS Express and VueJS frameworks and stores optimization project-related data in the NoSQL database.

The Dynamic analysis module framework is extendable to support multiple profilers. It stores the profiling results in the NoSQL database and makes them available in the Integrated optimization environment. Initially, two dynamic analysis modules are included in the framework supporting Google pprof and GNU gprof. The extendibility of the component allows the addition of more profilers in the future. Profiling tasks are automated using CMake and CTest, while the framework adds the necessary C and C++ flags needed for the profiler to the CMakeLists.txt file. The dynamic analysis process is implemented in an asynchronous manner according to the event-driven architecture, as this architecture is best suited for resource-intensive tasks, provides horizontal scaling capabilities, and does not freeze the user interface during the process.

The result validation module stores the result of a computational task and ensures that it is not corrupted after the optimization activities have been performed. The tests are run from the Integrated optimization environment, while the results are written to the objects storage or NoSQL database depending on the use case. The process is implemented asynchronously according to the event-driven architecture and relies on the use of CMake and CTest.

The performance and stability evaluation module sets the performance baseline and records the possible improvements after optimization activities. In the current version only the processor time is estimated; however, the module can be extended with additional functionality. Performance and stability evaluation is initialized through the Integrated optimization environment and relies on CMake and CTest. The results are logged in the NoSQL database and can be reviewed in the Integrated optimization environment.

The performance and stability evaluation process is implemented asynchronously and uses queue for this purpose.

The purpose of the static analysis module framework is to provide transformations aimed towards performance improvement. The framework contains an external library detection module that allows one to detect used external libraries and their possible alternatives. It also integrates with the ROSE compiler (Quinlan & Liao, 2011), which provides automated transformation of source code allowing to improve its efficiency and performance. The transformation process starts from the integrated optimization environment, and it is implemented asynchronously by using queue for passing tasks to the appropriate containerized workers. Object storage serves the purpose of storing temporary files needed for exchanging data between the Integrated optimization environment and worker containers.

The compiler framework provides unified infrastructure services for assessing the suitability of several compilers and their versions for a specific computing task and determining the optimal values of their configuration parameters in a semiautomated way. Currently three latest major versions of GCC are supported; however, it is possible to extend the component with additional compilers. To determine the optimal compiler and its configuration, it is also necessary to validate the results of computational tasks and evaluate their performance. The interaction with these modules is orchestrated by the ReCoTOS core, and the process is started from the Integrated optimization environment. The results are stored in the NoSQL database. Once the optimal settings have been determined, the Integrated optimization environment is reconfigured accordingly and restarted. The process is implemented asynchronously to avoid freezing of the user interface.

4.2 Integrated Optimization Environment

Special attention was paid to choosing technology for the Integrated optimization environment, since it is a user-facing component and most of the time the users of the platform would spend interacting with this module.

Eclipse Theia, Atheos, Gitpod, Eclipse Che, Cloud9 were reviewed as the potential candidates for the integrated optimization environment. The following tools were dismissed from further evaluation:

- Cloud9 – the free version does not include the needed functionality.
- The Gitpod – AGPL licence requires one to opensource any modifications that are made to the solution.
- Atheos - too few built-in functions for working with C and C++ projects.

From the remaining tools, Eclipse Che seemed to be the most feature-rich based on documentation, however, our practical experiments showed that it is often unstable and lacks sufficient documentation. Due to this reason, Theia was chosen as the most suitable environment.

4.3 Infrastructure Services

The ReCoTOS platform is based on the use of the following infrastructure services:

- Containerization facilities – Kubernetes is used for containerization of the platform components.
- Queue – RabbitMQ is used as the queue backend to enable asynchronous communication between components.
- NoSQL database – MongoDB is used.
- Key-value store – Redis is used to store user sessions.
- Authentication provider – KeyCloak is used for access and identity management.
- Object storage – Minio is used as object storage due to its minimalistic approach and compatibility with the S3 interface.
- Version control system – Gitea is used for the source management and branching of the optimization projects.

5 USE CASE

The functionality of the platform is demonstrated by optimizing C based software correlator for earth observation data processing (KANA), which was developed by Ventspils International Radio Astronomy Centre of the Ventspils University of Applied Sciences Institute of Engineering (IE VIRAC) in 2012 as part of the project "Earth's near-field radio-astronomical research". Optimization is carried out following the previously defined methodology (Kampars et al., 2020). Experiments are performed in the CloudStack (Train release) cloud computing platform. The cloud environment is equipped with Intel(R) Xeon(R) Gold 5218R CPU processors.

5.1 Project Initialization

Optimization starts by creating a new project through the management dashboard. It is necessary to specify the source code archive, performance goal, the external libraries, and custom installation scripts, if any.

After the new project has been saved, a new Gitea repository is created, and Theia container is started in the background. The git branches used during the optimization are shown in Figure 2. The original code from the project archive is added to the git branch 'Original' and a new branch named 'Optimized' is created and linked to the Theia container.

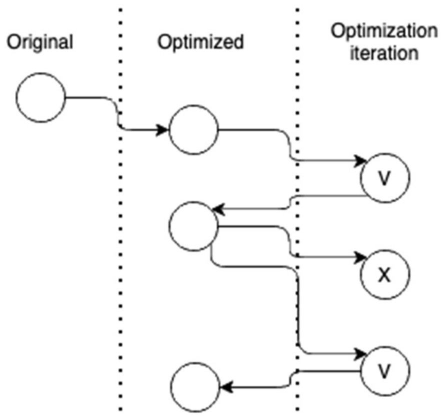
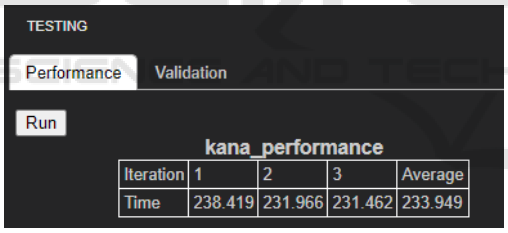


Figure 2: Git branches for optimization.

The initially committed version to the 'Optimized' branch is used for establishing the performance baseline, identifying the most appropriate compiler version and configuration, and for identifying the most resource-consuming sections of the computing task (hot kernels).

To identify the performance baseline (see Figure 3) of the computing task and to confirm its validity Performance and stability evaluation module is used through the Testing widget. As can be seen, the initial time of Kana computing task execution is averaging 234 seconds.



kana_performance

Iteration	1	2	3	Average
Time	238.419	231.966	231.462	233.949

Figure 3: Setting the performance baseline.

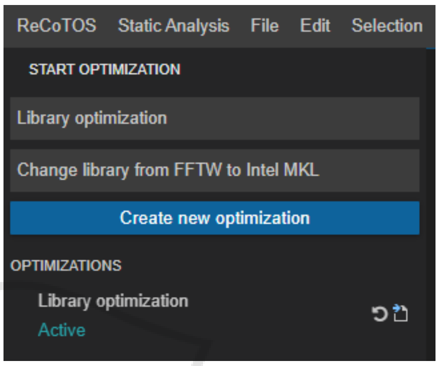
5.2 Compiler Settings Optimization

The interaction with the Compiler framework is done from a corresponding Theia plugin. It is possible to switch to the configuration, which would rebuild the Theia container with the appropriate compiler version and make the according changes to the CMakeList.txt file to set the compiler flags. The tuning of compiler version and its flags have allowed to reduce the execution time by approximately 58 seconds or by 24.8%.

5.3 Profiling and External Library Optimization

Google pprof and GNU gprof profilers are used to identify hot kernels through the corresponding Theia plugin. Both profilers provide tabular and graph representations of the profiling results. The GNU gprof compiler shows a better overview of internal functions or kernels while Google pprof also shows the functions that are executed within the linked external library.

Once the optimal compiler configuration has been set, the performance baseline and the hot kernel to be optimized have been identified, the user can start a new optimization iteration (see Figure 4) for which a corresponding git branch and a new instance of Theia container will be created.



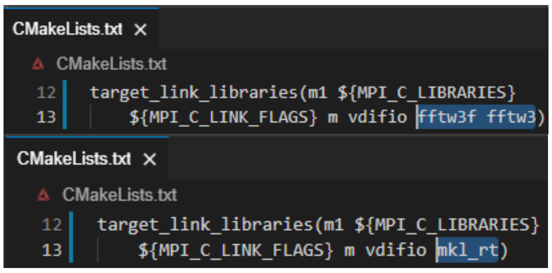
Library optimization

Change library from FFTW to Intel MKL

Library optimization
Active

Figure 4: Starting a new optimization iteration.

According to the optimization methodology (Kampars et al., 2020) the first optimization step is concerned with choosing the optimal external library. As the Google pprof results show, a significant amount of time is spent on Fourier transformations by FFTW. The Intel Math Kernel Library (MKL) has been identified as a potential alternative and will be further evaluated as part of the optimization iteration. Switching from FFTW to the Intel MKL library requires minor changes in CMakeLists.txt (see the file before and after the changes in Figure 5).

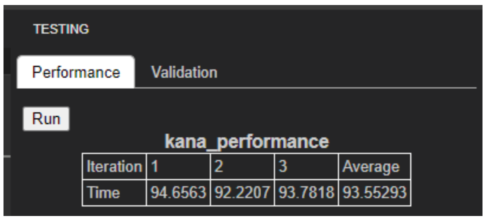


```
target_link_libraries(m1 ${MPI_C_LIBRARIES}
    ${MPI_C_LINK_FLAGS} m vdifio fftw3f fftw3)
```

```
target_link_libraries(m1 ${MPI_C_LIBRARIES}
    ${MPI_C_LINK_FLAGS} m vdifio mkl_rt)
```

Figure 5: Switching from FFTW to Intel MKL.

To evaluate the performance effect of changing external library, a test needs to be run using the Testing widget (see Figure 6) and the validity of computing task needs to be re-evaluated.



kana_performance

Iteration	1	2	3	Average
Time	94.6563	92.2207	93.7818	93.55293

Figure 6: Performance improvement after the first iteration.

The changes introduced did not have a negative effect on validity, while the execution time was reduced by approximately 82 seconds or 35.0% from the baseline originally measured. The optimization iteration ends with the saving of results and merging of the changes in the 'Optimization' branch (see Figure 7).

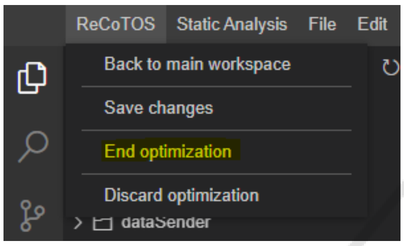


Figure 7: End a successful optimization iteration.

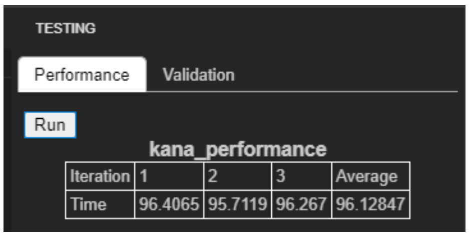
Afterwards, the user is redirected to the Theia container linked to the 'Optimization' branch.

5.4 Introduction of Structural Changes

According to the applied methodology (Kampars et al., 2020) structural changes to the computing task are the next most suitable optimization strategy after optimizing use of external libraries.

A new optimization iteration is started, and the computing task is profiled once again. It is decided that since all major internal functions are in the file mode1.c it could be transformed by the Rose compiler, which is part of the Static analysis module framework. The file is selected, and the corresponding Rose submenu is clicked under the Static Analysis menu item.

We can conclude that the maintainability of the code has been decreased, and it is necessary to evaluate if there is sufficient performance improvement to compensate for that. It is concluded that the computing task is still valid; however, there is no performance improvement (see Figure 8), in fact the execution time has increased by approximately 2 seconds.



kana_performance

Iteration	1	2	3	Average
Time	96.4065	95.7119	96.267	96.12847

Figure 8: Performance degradation after the second iteration.

It is decided to discard the changes made and return to the result of the previous successful optimization (see menu item 'Discard optimization' in Figure 7). The optimization ends with gaining execution time reduction by approximately 140 seconds or 59.9%, which is almost equal to the initially defined optimization target.

6 CONCLUSION AND FUTURE WORK

The ReCoTOS platform has been designed, implemented, and its functionality has been demonstrated using a C-based software correlator. Initially, it targets C and C++ programming languages, and the applied optimization methodology is mostly concerned with improving the vectorization of the computing tasks.

Our experience while developing the methodology and the platform shows that software optimization is a complicated process which would benefit from availability of a supporting methodology, cloud-based platform and best practices expressed as patterns.

We have formalized 7 patterns in the pattern repository with the measured execution time reduction as follows:

- 2D array processing pattern (structural changes in the code) - 47%.
- External library usage pattern (external library replacement) - 43%.
- Array sort pattern variant #1 (structural changes in the code) - 66%.
- Array sort pattern variant #2 (structural changes in the code) - 78%.
- Matrix multiplication pattern #1 (structural changes in the code) - 78%.
- Matrix multiplication pattern #2 (structural changes in code) - 91%.
- Square matrix transposition pattern (structural changes in code) - 28%.

While working on the patterns we have discovered that some of the introduced structural changes can have very different impact on various processor architectures – resulting in significant execution time reduction on one processor, while causing execution time increase on another one. We have also acknowledged that even optimizations with positive performance impact can have negative effects on the stability of the computing task, maintainability, and portability of the codebase, which might prove them not feasible. Software developers are not always familiar with existing optimization practices, their efficiency on certain CPU versions and efficient usage of modern CPU capabilities, which is why it is particularly important to increase the number of best practice patterns stored in the pattern repository. This would also assist us in further improvement of the methodology and the platform.

It is necessary to extend the platform with support for testing computing tasks on different CPU architectures, which would require some changes on the hardware level and platform itself.

Another possible direction for evolving the ReCoTOS platform is to add support for optimizing computing tasks designed for edge computing. This would require extending the cloud computing platform with several edge nodes, as well as changes in the container orchestration and a number of modules.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the project "Competence Centre of Information and Communication Technologies" of EU Structural funds, contract No. 1.2.1.1/18/A/003 signed between IT Competence Centre and Central Finance and Contracting Agency, Research No. 1.13" Resource-saving Computing Task Optimization Solutions".

REFERENCES

Amiri, H., & Shahbahrami, A. (2020). SIMD programming using Intel vector extensions. *Journal of Parallel and Distributed Computing*, *135*, 83–100. https://doi.org/10.1016/j.jpdc.2019.09.012

Cockshott, P., & Renfrew, K. (2004). SIMD Programming in Assembler and C BT - SIMD Programming Manual for Linux and Windows. In P. Cockshott & K. Renfrew (Eds.) (pp. 23–46). London: Springer London. https://doi.org/10.1007/978-1-4471-3862-4_3

Holewinski, J., Ramamurthi, R., Ravishankar, M., Fauzia, N., Pouchet, L. N., Rountev, A., & Sadayappan, P. (2012). Dynamic trace-based analysis of vectorization potential of applications. *ACM SIGPLAN Notices*, *47*(6), 371–382. https://doi.org/10.1145/2345156.2254108

Hughes, C. J. (2015). Single-instruction multiple-data execution. *Synthesis Lectures on Computer Architecture*, *32*, 1–121. https://doi.org/10.2200/S00647ED1V01Y201505CAC032

Kampars, J., Irbe, J., Kalnins, G., Mosans, G., Gulbe, R., & Pinka, K. (2020). ReCoTOS: A Methodology for Vectorization-based Resource-saving Computing Task Optimization. In *2020 61st International Scientific Conference on Information Technology and Management Science of Riga Technical University, ITMS 2020 - Proceedings*. https://doi.org/10.1109/ITMS51158.2020.9259289

Lai, Z., Luo, Q., & Xie, X. (2019). Efficient data-parallel primitives on heterogeneous systems. *ACM International Conference Proceeding Series*, (Mic). https://doi.org/10.1145/3337821.3337920

Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A., & Yaqoob, I. (2017). Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges. *IEEE Access*, *5*, 5247–5261. https://doi.org/10.1109/ACCESS.2017.2689040

Nuzman, D., & Zaks, A. (2008). Outer-loop vectorization - revisited for short SIMD architectures. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, 2–11. https://doi.org/10.1145/1454115.1454119

Quinlan, D., & Liao, C. (2011). The ROSE Source-to-Source Compiler Infrastructure. *International Journal*, 1–3.

Ren, J., Guo, H., Xu, C., & Zhang, Y. (2017). Serving at the Edge: A Scalable IoT Architecture Based on Transparent Computing. *IEEE Network*, *31*(5), 96–105. https://doi.org/10.1109/MNET.2017.1700030

Rong, H., Zhang, H., Xiao, S., Li, C., & Hu, C. (2016). Optimizing energy consumption for data centers. *Renewable and Sustainable Energy Reviews*, *58*, 674–691. https://doi.org/10.1016/j.rser.2015.12.283

Stone, J. E., Gohara, D., & Shi, G. (2010). OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in Science and Engineering*, *12*(3), 66–72. https://doi.org/10.1109/MCSE.2010.69

Watanabe, H., & Nakagawa, K. M. (2019). SIMD vectorization for the Lennard-Jones potential with AVX2 and AVX-512 instructions. *Computer Physics Communications*, *237*, 1–7. https://doi.org/10.1016/j.cpc.2018.10.028