# Exploiting Multicore Processors in PLCs using Libraries for IEC 61131-3

Felix Specht, Holger Flatt, Jens Eickmeyer, and Oliver Niggemann
Fraunhofer IOSB-INA, Application Center Industrial Automation
Langenbruch 6, 32657 Lemgo, Germany
{felix.specht, holger.flatt, jens.eickmeyer, oliver.niggemann}@iosb-ina.fraunhofer.de

***Abstract*—This paper presents an approach for exploiting multicore hardware architectures on coding level for the IEC 61131-3. An interface between the IEC 61131-3 code and software of a different programming language outsources the actual parallel workload. For validation purpose, an embedded multicore hardware is used as a controlling device, which executes software for the use case of model based condition monitoring. The results show an explicit benefit of the multicore exploiting software in comparison to its singlecore counterpart, which is reflected with a faster processing of up to a factor of 3. Overall, this approach can be used for developing high performance applications or for accelerating existing applications in industry.**

## I. Introduction

Cyber Physical Systems (CPSs) are a holistic view onto both the physical systems and the computers controlling them. An example for CPSs are production plants, where the increasing complexity of plants leads to increasing requirements for the corresponding automation systems, i.e. real-time requirements, processing power demands or safety regulations [1].

The automation systems are evolving from pure controllers towards complex CPSs, which leads towards an increasing interconnection with algorithms and computer scientific technology. Model-based condition monitoring [2], image interpretation algorithmic for camera sensors [3] or simulation-based controlling [4] are use cases of such technology, which require high processing performance. The performance is directly connected with the efficiency of production facilities and consequently has a big economical impact. In order to keep the promises of CPSs, a sufficient processing power is a key requirement, a requirement which can only be met by leveraging technologies like multicore CPUs.

Physical devices are controlled by programmable logic controllers (PLCs), which are specialized hardware systems for the industrial environment. Nowadays, the majority of PLCs in operation use singlecore architectures, while PLC manufacturers already provide multicore solutions and will expand this trend[5]. Besides the advantage in terms of processing power, multicore architectures outperform singlecore architectures in respect of energy efficiency [6].

The software counterpart for programming PLCs is the IEC 61131-3 standard, which supports the utilization in various domains [7]. Considering IEC 61131-3 in the context of multicore architectures, parallelism is supported at program and task level, but not at coding level. However, code parallelism is a considerable objective for a successful accomplishment of the prior mentioned use cases, otherwise their potential for high-performance computing can not be fully utilized.

Recent approaches consider the general use case, in an attempt to generate parallelism from sequential IEC 61131-3 code [8][9]. In contrast, this work focuses only on particular uses cases, which heavily benefit from parallelism on code level. The basic idea is the deployment of a library, containing corresponding parallel solutions. This is achieved by connecting the sequential 61131-3 code with this library, which is based on a different programming language. Interfaces enable the PLC programmer to invoke the parallel functionality inside his code, while the algorithmics and parallel execution is hidden. Interface realizations are part of the IEC standard, thus the complexity level remains the same for the user.

This paper is organized as follows: Section II reviews the related work, considering both hardware and software issues. In section III, different approaches to exploit the multicore architecture are compared. Section IV issues the library approach and the mapping onto multiple CPUs. The case study is explained in section V, introducing the implementation, test scenario and corresponding results. Finally, section VI concludes this paper.

## II. State of the Art

Considering the single core architecture as origin, various approaches for hardware, software and mixed solutions are being researched and developed to increase the processing performance.

### A. Hardware Approaches

A combination of a single processor with FPGA extension is demonstrated in [10]. This solution greatly increases the performance of an implementation in the FPGA part, while keeping the capability of general purpose computation. Due to the limited bus connection, difficulties occur with tasks requiring fine grained communication between the CPU and the FPGA.

Coupling CPU and FPGA into a single system-on-chip overcomes this connection issue by exploiting a very short communication distance [11], though the CPU part does not reach the performance of a standalone version. As in the case of all FPGA solutions, the above mentioned are accompanied with high development effort and relatively expensive hardware.

A more sophisticated optimization for a specific problem can be achieved with an application specific integration circuit (ASIC) architecture. This is a highly integrated circuit, designed to solve one specific problem, but lacks in terms of development efforts and costs [12].

The industrial trend evolves towards shorter assembly cycles and individual products. The controlling systems need to be both flexible and powerful, thus research focuses on corresponding solutions. Since the performance of single core processors reached physical limits, the trend evolves towards multi-processor architectures [13]. Embedded multicore hardware has the potential to supply better performance at lower energy cost [14], hence modern PLCs already exploit these architectures [5].

### *B. Software Approaches*

The IEC 61131-3 software standard for PLCs is formed by five distinct graphical and textual based programming languages to achieve flexibility for different domains [15].

The usual IEC 61131-3 program consists of several tasks, which are cyclically executed. In every cycle, a task reads input values, processes controlling code and returns output values to the environment [7]. A task is build of program organization units (POUs), the basic software components. A POU is either a function, a function block or a program. Besides the standard functionality IEC 61131-3 supports implementing customized POUs.

The IEC 61131-3 runtime enables parallel processing of different programs and tasks onto multiple cores [5][16]. This provides a benefit in comparison to singlecore PLCs. The programming capabilities only support sequential programming, thus IEC 61131-3 is limited to coarse-grained parallelism. The granularity of task parallelism depends on the application, but it is not sufficient as the only source of parallelism [17]. Figure 1 exemplarily demonstrates the difference in execution behavior for a program of four tasks with varying computational costs. In *a)* the four tasks are successively processed onto a single core, while in *b)* each task uses a separate core, with which the whole program is significantly accelerated. Figure 1 *c)* illustrates the additional benefit of fine-grained parallelism that introduces the missing component required for shorter execution potential in shape of fine-grained.

When taking a closer look at fine-grained parallelism, industrial image processing is a suitable example, as it is a key technology in automated manufacturing [18]. Here, specific processing techniques perform algorithmic operations to every pixel of an image. An operation without dependencies between the pixels provides the opportunity of nearly unlimited data parallelism [17], which is a specific fine-grained type.

Figure 2 illustrates an example of a simplified pipeline consisting of different tasks as part of an entire image procedure [19]. After the first filter is applied to the input image, the pipeline splits into two traces. Here, task parallelism can simultaneously handle *Transformation 1* and *Transformation 2* for instance, but not *Transformation 2 and Filter 2*.

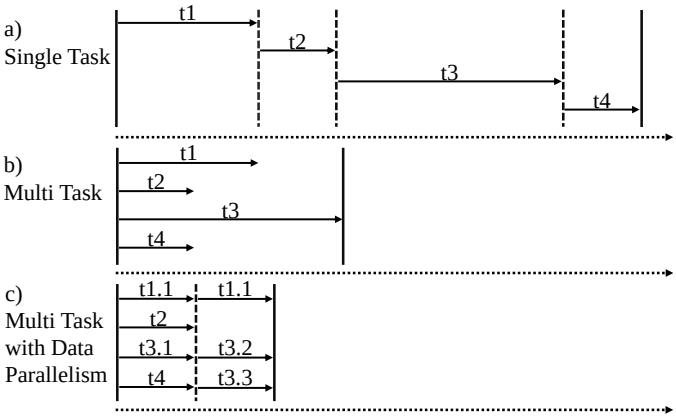


Fig. 1. Parallel execution behavior over time in comparison of approaches with different granularity.

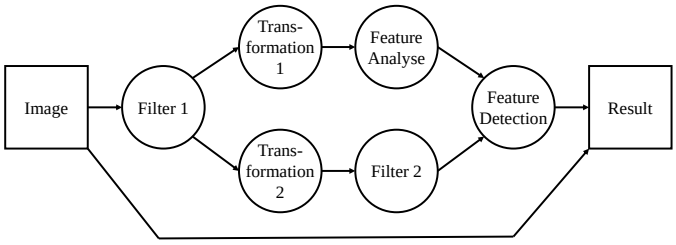


Fig. 2. Image pipeline of different tasks with dependencies.

The input of a task depends on the results of the former one, thus task parallelism has no effect at this point. Inside a single task, the same operation is executed many times onto the input data, providing a high speed-up potential through data parallelism and pipelining.

As a simple approach, an image processing task can manually be split into multiple tasks, which the IEC 61131-3 runtime can execute. For instance, an initial task splits the input image into four pieces, distributes them to four other tasks for processing and merges the partial results afterwards. However, this would involve a significant additional effort and requires a specific knowledge about the operating PLC, since the number of used tasks has to be set statically in advance of the operation. Considering the scalability of a system, this is not a practicable approach. Manual task handling for a big number of CPU cores, like 16, 32, 64 cores, would be very complex, if even possible. At this point, self-handled software parallelism is a more sensible way.

Besides of image processing, additional data intensive procedures are in the focus of research and development for industrial usage, like *Data-driven Condition Monitoring* or *Big Data*. In these domains, data parallelism also yields a significant potential for improving the processing performance.

Enabling data parallelism within IEC 61131-3 is a primary objective to successfully exploit multicore hardware for industrial CPSs.

## III. Parallel Extension to IEC 61131-3

Computer science offers established concepts to exploit multicore architectures with fine-grained parallelism. Different approaches to integrate those concepts into IEC 61131-3 will be introduced and compared in the following subsection. The subsequent subsection presents the *Libary Extension* to IEC 61131-3, which is the actual approach in this paper.

### A. Comparing Approaches

Existing programming languages provide techniques to enable parallelism at coding level, like the *POSIX Threads (Pthreads)* library [20]. Assume an extension to IEC 61131-3, which reproduces *Pthreads*, that supports programmers to manually implement an arbitrary parallel application. This method provides huge capabilities for high-performance, but also comes with disadvantages. It requires a considerable know-how in software development, while consuming huge amounts of time and costs [21]. Engineering tools for parallel programming are not yet in a state for a fast and reliable development process [22]. Furthermore this extension would require many changes to the IEC standard.

A different approach is the usage of software constructs like *OpenMP* [23] within IEC 61131-3. *OpenMP* works on a higher abstraction level [22], resulting in less complex requirements for the developer. Here, parts of the programming code are marked with pragmas, which instruct the compiler to translate the sequential code to a parallel executing program. Code constructs, like loops, suit well for this approach and yield good results. Multiple iterations of a loop are executed on multiple cores simultaneously. *OpenMP* can be classified as semi-automatic parallelism, which abstracts from thread handling or data consistencies.

The downside is an additional need of know-how about assigning pragmas in the correct way at meaningful code locations. Furthermore, *OpenMP* is limited to shared memory systems and symmetric multiprocessing. In comparison to *Pthreads*, the development risks are lower due to outsourced thread handling, but at the costs of flexibility. Porting *OpenMP* to IEC 61131-3 would require a medium amount of extension to the actual syntax, but significant changes to the respective compiler.

In terms of abstraction, fully generated parallelism is the next step. The authors in [24] introduce different approaches based on *OpenMP*, which analyze sequential code to find data dependencies and code sections that suit parallel execution. Depending on the tool, code coverage for parallel generation and the correlating performance differs for distinct use cases (e.g. Matrix Multiplication or Fourier Transformation).

The work [8] introduces a tool that generates parallelism from sequential IEC 61131-3 code, based on a previous transformation to C code. Generated parallelism promises good usability and good results, but it turned out that industrial applications in practice oftentimes poorly fit for parallel generation [9]. This approach shifts the complexity and development risk to the generator, thus the output is hardly comprehensible for the user.

Table I summarizes the properties of the different approaches. In addition to the introduced approaches, this paper is classified according to the same categories. The primary objectives are high performance and low user complexity, however there is a trade-off between performance and versatility.

TABLE I
Overview: The introduced Approaches in Comparison

| | **Manual** *(Pthreads)* [20] | **Semi-automatic** *(OpenMP)* [23] | **Full-automatic** *(Generator)* [24][8] | **This Paper** |
|---|---|---|---|---|
| **Performance** | highest | medium | low - medium | high |
| **Complexity** | highest | low - medium | low | low |
| **Risk** | highest | medium | high | medium |
| **Versatility** | highest | medium | high | low |
| **Changes to IEC** | highest | medium | low | low |

### B. Library Extension

The previously introduced approaches implement parallelism through different sorts of source code modifications. In contrast, this work does not directly aim towards the creation of parallel code, but towards a connection between the sequential IEC 61131-3 code and parallel constructs in another programming language.

By including essential software libraries into IEC 61131-3, established solutions from other domains are reused. As explained previously, the IEC 61131-3 does not support parallelism at coding level. However, by implementing a custom POU, which is a standardized procedure, an interface towards a different programming language is established.

The actual IEC 61131-3 code remains sequential, while the invoked library conducts parallel computing. The corresponding development effort and complexity is outsourced, e.g. handling of threads and data dependencies.

Figure 3 illustrates the approach with the example of a controlling application. Here, the IEC 61131-3 controlling task runs cyclically on one of the four CPU cores. In addition to standard POU calls, the custom POU interface invokes an external library in a different programming language, which is capable of code parallelism. At this point, the invoking task can hand over arbitrary parameters, which can serve as settings or data input. Then this library can create threads with corresponding functionality, which are distributed to the different CPU cores. The thread scheduling is assumed by the operating system that allocates the processing time for a certain thread.

The workload over time of multiple CPU cores is demonstrated in Figure 4. Already during execution of the IEC 61131-3 task, the library threads are started concurrently on all the cores. The library threads fully utilize the CPU capacity in between two task intervals, but complete and stop the execution before the next interval. While this demonstrated behavior is exemplary, an arbitrary combination of library threads and IEC 61131-3 task are possible, each with various execution and interval time.

## IV. Case Study

This section applies the presented approach as an example to data driven condition monitoring (CM).

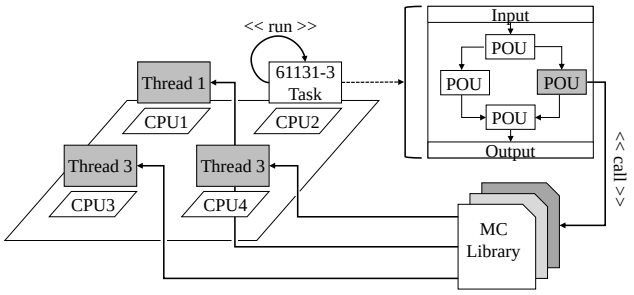


Fig. 3. Concept of task and library distribution to multiple cores.

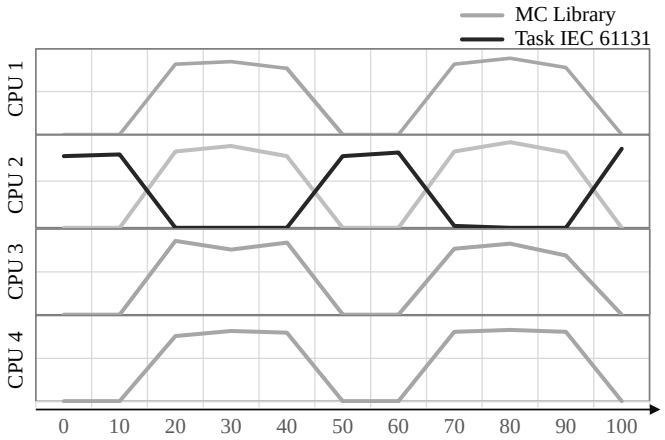


Fig. 4. Concept of workload distribution onto multiple cores.

Maintaining a smooth production flow by a quick reaction to malfunctions of machines is an important nontrivial task. With modern CM systems being object of current research and development, this scenario represents a meaningful use case in automation.

At first, this section introduces a specific approach of data driven CM, as a foundation for the case study. Then the used hardware platform and its corresponding attributes with respect to the study are described. Afterwards, the CM software implementation is introduced, taking a closer look at the applied tools and libraries. The section closes with the measurement results of the example application.

### A. Data driven Condition Monitoring

Condition Monitoring Systems (CMS) support maintenance tasks of complex machines. In CMS, common solutions rely on expert knowledge as part of the PLC [25]. While these solutions are specialized for a specific machine, they require time extensive development.

Machine learning algorithms, as a key component of CMSs, automatically generate complex models. This procedure is based on historical data that represent a normal work flow of the monitored machine. As representation of the learned normal machine behavior, the model is compared with the actual device state to identify system deviations [26]. Especially, data driven modeling overcomes the need for expert knowledge and enable a reduction of development costs. In complex systems, one requirement is the processing of huge amount of sensor values in a short time period. Parallel computing techniques and high performance hardware help to utilize complex CM algorithms as part of the PLC. This leads to a significantly reduced data traffic, as there is no need to transfer sensor data to a central service, which enables the option to skip additional techniques for data acquisition. Furthermore, the timestamps directly generated within the PLC increase the precision of these used algorithms.

Complex facility processes and their corresponding models require an appropriate visualization in order to support humans to understand the information content. One approach to achieve this is the principal component analysis (PCA) [27], which transforms a high dimensional data set $M_D$ into a low dimensional space $M_d$. With $d, D = (R)$, where $d \leq D$, the low dimensional space $M_d$ is spanned by the *eigenvectors* of the covariance matrix. The *eigenvectors* are then sorted decreasingly according to their *eigenvalues*. The PCA returns a rotation matrix $R$, which is used to transform new sensor values into the low dimensional principal component space $M_d$. Time-critical production processes come with challenging requirements for a data based CMS. Computing tasks, like $M_d = R \cdot M_D$, execute a matrix-vector operation for each incoming sensor value and therefore need fast processing. Parallel computing is able to challenge these requirements.

### B. Hardware

The integration of the CM concept directly into a PLC requires a powerful hardware, which is capable of handling both controlling and CM in parallel. Therefore, the *Odroid U3* development board was exemplary chosen, which provides a multicore processor onboard of an embedded platform.

This hand-sized device comprises an ARM v7 based Cortex-A9 processor with 4 CPU cores at 1.7 GHz. Each core includes 32 KB of first level cache, while 1 MB second level cache is shared. Furthermore, the board provides 2 GB of internal memory, as well as a single Ethernet port for communication. Academic interest in this processor architecture is growing, especially for the high performance computing and high energy physics domains [28]. The ARM architecture, in general, is part of market available PLC products [5].

### C. Application

PLC manufacturers usually provide specific IDEs to support the integration of their own devices. Modifications are not commonplace, thus a connection to a foreign device is hardly realizable. Therefore the application development requires a modifiable editor and a converter to compile IEC 61131-3 into the C programming language. Furthermore, the option to describe hardware registers and to directly access them in IEC 61131-3 is essential for the communication with industrial devices. The development tool *GebAutomation* [29] was chosen for implementation, since it meets these requirements.

The implementation integrates a device specific cross-compiler and enables register usage of an industrial I/O device. Due to its low complexity, the *modbus* protocol is used

for communication. By applying new IEC 61131-3 function blocks into the *GebAutomation* environment, the C library *libmodbus* enables a communication between the I/O device and the *Odroid*.

The CM application requires a mathematical foundation, which is made available through the hierarchical libraries shown in Figure 5. The top level applies the interface between the IEC 61131-3 FBs and C, using a library for IEC 61131-3-to-C transformation. The transformed sources access the high-level library *libpca* [30], which is an extension to the linear algebra library *Armadillo*. It provides a good trade-off between processing speed and ease of use, and therefore can be used for fast prototyping and computationally intensive experiments [31]. *Armadillo* itself builds upon a variation of *basic linear algebra subprograms* (BLAS) [32], which holds as the de facto standard for linear algebra operations [33]. The variant used in this approach is *OpenBLAS*, which ranks as one of the top implementations of BLAS [34][35]. The essential processing performance of this library comes with its highly adapted operations in *Fortran* and *Assembler*.

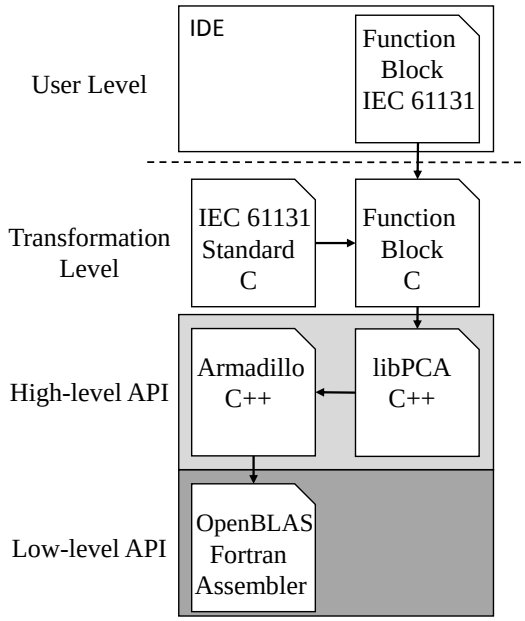


Fig. 5. Library Architecture

Building upon the new functionality, the following software modules are implemented in IEC 61131-3:

1) An IEC 61131-3 controlling program for a modbus I/O device
2) The data-driven CM application, triggering the PCA and processing input data for monitoring
3) Communication between the controlling program and the CM application
4) Data value acquisition as input to the CM application
5) Providing feedback to the user and the controlling program for adjustments

The realization is done with FBs on IEC 61131-3 side, which interfaces to the corresponding C/C++ implementation. Consequently, an FB enables the CM setup by performing an initial PCA to compute the rotation matrix and scaling values. During the operation process, a specific FB transforms incoming data vectors based on the PCA results, e.g. process data received via modbus connection. Another FB performs the evaluation by comparing the transformed values against the initially learned model. A corresponding feedback is send to the controlling software, e.g. for performing adjustments or for visualizing the CM results for the user.

### *D. Results*

As a reference, the evaluation runs the previously introduced CM application onto one and four CPU cores. The test system consists of the *Odroid* board with a non-real time variant of the Linux OS. Historical process data obtained by a real wind power facility is used as input. The performance is measured by reference to the cyclic execution time of a CM task.

Table II shows the average cyclic execution time of a task and the corresponding data vector. In each execution cycle, a vector of size 20 - 1000 is processed, measuring the required time. The used historical data provides different vectors with numerical entries of length 3 - 7. One million test iterations were performed with these different input vectors. Due to the mathematical background of vector/matrix multiplication, the values show an exponential growth in cycle time. For vectors of size 60 - 300, the quadcore solution spends significantly less time than the singlecore.

Table II directly compares the single- and quadcore timings, which show very similar results for small vector sizes. With a vector size of 20, singlecore even comes with slightly better results. This owes to the overhead of thread handling while using multiple cores for small tasks. Furthermore, this causes a better standard deviation for singlecore, due to less interference of the operating system.

With an increasing vector size, the number of operations increases, while the effort for thread handling remains unchanged. As a consequence, the quadcore solution outperforms singlecore. This progression is displayed in Figure 6, showing the ratio between quad- and singlecore timings. The ratio is synonymous to the performance gain of the quadcore towards the singlecore solution. The ratio increases significantly up to factor 3 for vectors with a size between 60 and 300. At this point, the performance of quadcore is more than three times better. Considering the vector range 20 - 300, the ratio evolution corresponds to the typical relationship between a single- and multicore solution, described by *Amdahl's Law* [36].

The quadcore progression massively drops for even bigger vectors, with timings being reduces to nearly the same level as singlecore. This effect occurs beacuse of hardware limits, in particular the limited CPU cache. This is a common observation in the context of the BLAS library for matrix/vector multiplications [33]. With the matrix reaching a certain size, the CPU cache is not sufficient to store it and therefore needs to fetch the matrix from main memory in every processing step. After crossing this border, the processing bottleneck

TABLE II
AVERAGE EXECUTION TIME OF SINGLE/QUADCORE FOR DIFFERENT VECTOR SIZES.

| **Singlecore** | **20** | **40** | **60** | **80** | **100** | **200** | **300** | **400** | **800** | **1000** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Avg.** ($\mu$s) | 4.20 | 8.26 | 15.16 | 25.69 | 36.13 | 118.79 | 253.20 | 548.00 | 3.157.00 | 4.936.52 |
| **Std. Dev.** ($\mu$s) | 0.79 | 1.01 | 0.82 | 1.05 | 1.13 | 2.36 | 5.50 | 13.02 | 25.12 | 53.97 |
| **Quadcore** | **20** | **40** | **60** | **80** | **100** | **200** | **300** | **400** | **800** | **1000** |
| **Avg.** ($\mu$s) | 5.01 | 6.23 | 9.40 | 11.09 | 14.20 | 41.86 | 82.64 | 450.66 | 2.918.04 | 4.742.31 |
| **Std. Dev.** ($\mu$s) | 2.35 | 1.82 | 3.11 | 2.25 | 2.11 | 3.21 | 12.25 | 20.94 | 57.98 | 65.98 |

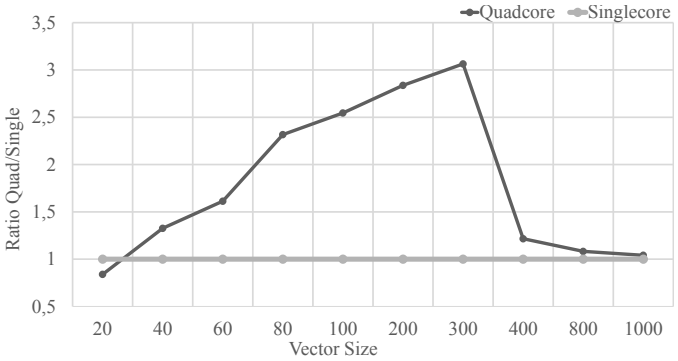


Fig. 6. Ratio Single-/Quadcore for different vector sizes.

shifts from the CPU power towards the bridge, connecting CPU and the main memory. The CPU cores are in an idle state most of the time, hence the quadcore advantage disappears.

Assuming the cache limit is not an issue, the multicore performance tends towards even better results.

### *E. Approaches to solve the caching problem*

Two strategies are conceivable to approach the caching limitation, as well as a combination of those. On the one hand hardware based approaches promise better results. Due to the direct dependency between L2 cache and matrix size, a brute force approach is the usage of a bigger L2 or an additional L3 cache. Another approach could focus on the bus connection between main memory and CPU. While an improved bus cannot nullify the need of cache, it can increase the performance of a system, which has already reached its cache limit. A third approach is the utilization of Scratchpad memory, which is a high-speed internal memory holding small data for rapid retrieval with a corresponding instruction and data prefetching technique. On the other hand, software offers an approach to a solution. During CM calculations, the transformation matrix is applied to every input, thus the matrix can be split and partly be used [37]. After applying all matrix parts to an input, it is necessary to assemble the result. Although this approach requires additional processing efforts, it promises a better workload of the multicore architecture.

## V. CONCLUSION

In this paper a concept for parallelism on coding level for IEC 61131-3 was presented, which is based on an interconnection towards parallel libraries in different programming languages. The evaluation was performed by the example of parallel processing for data driven conditon monitoring. Here, exploiting the multicore hardware with a corresponding parallel software showed a significant benefit in execution time, in comparison to sequential software, as long as the hardware resources are not exhausted.

The use of the introduced approach yields the following advantages: It is an extension to IEC 61131-3, which gets along without affecting the IEC standard. By encapsulating the IEC 61131-3 code from the parallel library, it is possible to exchange the code without changing the library and vice versa. Furthermore, this enables the same program to run on a platform with one or multiple CPU cores. The basic idea can be adapted to any library or a combination of libraries with arbitrary functionality. By itself, a library is a modular component and therefore suits well for a certification process. Using a reliable library delivers predictable results, while reducing the development risk and complexity.

While this approach is providing several advantages, this work helped in revealing the following disadvantages: The functionality of a library is limited to a certain area of application, yet an additional effort for induction is required. Implementing an extension or fixing errors is expensive and may need to engage the library developer. Making use of an untested library can involve unpredictable risks and consequently lead to corresponding problems.

Future work will engage the problem of a dropping performance when hitting the cache limit, though this is an application specific issue and occurs by using the *BLAS* library. The usage of a CM system on a multicore device, is not only intended for demonstration purpose, but also in real industrial operations. This porting procedure will come up with further challenges, like the integration of a real-time OS or hypervisor technologies. Parallelism in PLCs in general and corresponding technologies are of further interest, like the *Embedded Multicore Building Blocks* library or the Grand Central Dispatch approach.

In addition, it is a desirable objective to create a collection of parallel solutions for typical automation problems, for the status quo as well as for upcoming problems.

## REFERENCES

[1] G. Mustapic, A. Wall, C. Norstroem, I. Crnkovic, K. Sandstrom, J. Froberg, and J. Andersson, "Real world influences on software architecture - interviews with industrial system experts", in *Software Ar-*

*chitecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*, 2004, pp. 101–111.

[2] S. Ding, P. Zhang, E. Ding, P. Engel, and W. Gui, "A survey of the application of basic data-driven and model-based methods in process monitoring and fault diagnosis", in *Proceedings of the 18th IFAC World Congress*, 2011.

[3] S. Chen and D. Perng, "Automatic optical inspection system for ic molding surface", *Journal of Intelligent Manufacturing*, pp. 1–12, 2014.

[4] A. Canedo, G. Münzel, G. Lo, and T. Grünewald, "Cyber-physical programmable logic controller", *atp edition*, pp. 58–64, 04 2013.

[5] Beckhoff, *Multi-core processors for controllers in the medium performance range*, http://www.beckhoff.com, 11 2013.

[6] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors", *Parallel and Distributed Systems, IEEE Transactions on*, vol. 19, no. 11, pp. 1540–1552, 2008.

[7] *IEC 61131-3 ed3.0, Programmable controllers - Part 3: Programming languages*, http://www.iec.ch/.

[8] A. Canedo and M.A. Al-Faruque, "Towards parallel execution of IEC 61131 industrial cyber-physical systems applications", pp. 554–557, 2012.

[9] A. Canedo, H. Ludwig, and M.A. Al Faruque, "High communication throughput and low scan cycle time with multi/many-core programmable logic controllers", *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 21–24, 2014.

[10] H. Flatt, J. Jasperneite, D. Dennstedt, and Tran Dinh Hung, "Mapping of prp/hsr redundancy protocols onto a configurable FPGA/CPU based architecture", in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, 2013, pp. 121–128.

[11] S. Korf, G. Sievers, J. Ax, D. Cozzi, T. Jungeblut, J. Hagemeyer, M. Porrmann, and U. Rückert, "Dynamic reconfiguration of real-time ethernet standards with hard real-time requirements (in german)", *Proceedings Wissenschaftsforum 2013 Intelligente Technische Systeme*, 2014.

[12] K. Ian and J. Rose, "Measuring the gap between FPGAs and ASICs", vol. 26, no. 2, pp. 203–215, 2007.

[13] J.A. Darringer, "Multi-core design automation challenges", pp. 760–764, 2007.

[14] D. Abdurachmanov, P. Elmer, G. Eulisse, and S. Muzaffar, "Initial explorations of arm processors for scientific computing", in *Journal of Physics: Conference Series*. IOP Publishing, 2014, vol. 523, p. 012009.

[15] M. Tiegelkamp and K.H. John, *IEC 61131-3: Programming Industrial Automation Systems*, 2006.

[16] F.J. Bartos, "Computing power: Multi-core processors help industrial automation", http://www.controleng.com/, 02 2011.

[17] M.I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs", in *ACM SIGOPS Operating Systems Review*. ACM, 2006, vol. 40, pp. 151–162.

[18] C. Demant, B. Streicher-Abel, and C. Garnica, *Industrial image processing*, Springer, 2013.

[19] H. Flatt, H. Blume, and P. Pirsch, "Mapping of a real-time object detection application onto a configurable RISC/coprocessor architecture at full HD resolution", in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, 2010, pp. 452–457.

[20] "Ieee standards interpretations for ieee std 1003.1-1995 ieee standard for information technology – portable operating system interface (posix) - system application program interface (api) amendment 2: Threads extension (c language)".

[21] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era", *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1369–1386, 2012.

[22] R. Strebelow, "Heavy. debugging of embedded multicore system.", *IX*, , no. 2, pp. 77–79, 2014, (in German).

[23] *OpenMP - API specification for parallel programming*, http://openmp.org/wp/openmp-specifications/.

[24] E. Kallel, Y. Aoudni, and M. Abid, "Openmp automatic parallelization tools: An empirical comparative evaluation", *IJCSI International Journal of Computer Science Issues, Vol. 10*, 2013.

[25] M.G. Ioannides, "Design and implementation of PLC-based monitoring control system for induction motor", *IEEE Transactions onEnergy Conversion*, vol. 19, no. 3, pp. 469–476, 2004.

[26] O. Niggemann and B. Kroll, "On the applicability of model based software development to cyber physical production systems", in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, 2014, pp. 1–4.

[27] K. Pearson, "Principal components analysis", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 6, no. 2, pp. 559, 1901.

[28] D. Abdurachmanov et al., "Explorations of the viability of arm and xeon phi for physics processing", *Journal of Physics: Conference Series*, vol. 513, no. 5, 2014.

[29] *Geb Automation*, http://gebautomation.com/.

[30] "libpca c++ library", http://sourceforge.net/projects/libpca/.

[31] C. Sanderson, "Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments", *NICTA*, 09 2010.

[32] C.L. Lawson et al., "Basic linear algebra subprograms for fortran usage", *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 3, pp. 308–323, 1979.

[33] M.I. Soliman, "Performance evaluation of multi-core intel xeon processors on basic linear algebra subprograms", in *Computer Engineering & Systems, 2008. ICCES 2008. International Conference on*, 2008, pp. 3–9.

[34] D. Eddelbuettel, "Benchmarking single-and multi-core blas implementations and gpus for use with r", Mathematica, 2010.

[35] X. Zhang, "Openblas", https://github.com/xianyi/OpenBLAS/wiki, 2012.

[36] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", in *Proceedings of the April 18-20, 1967, spring joint computer conference*. ACM, 1967, pp. 483–485.

[37] A. Asaduzzaman, F.N. Sibai, and H. El-Sayed, "Performance and power comparisons of MPI vs pthread implementations on multicore systems", in *Innovations in Information Technology (IIT), 2013 9th International Conference on*, 2013, pp. 1–6.