Parallelization of Real-time Control Algorithms on Multi-core Architectures using Ant Colony Optimization

Oliver Gerlach, Florian Frick, Armin Lechler and Alexander Verl
Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Seidenstr. 36, 70174 Stuttgart, Germany

Keywords: Ant Colony Optimization, Parallelization, Industrial Control System, Real-time System, Multi-core.

Abstract: The emerging digitalization of production is accelerating the transformation of Industrial Control Systems (ICSs) from simple logic controllers to sophisticated systems utilizing complex algorithms which are running under strict real-time conditions. The required performance has now reached the limitations of single-core processors, making a transition to multi-core systems necessary. The parallelization of the currently monolithic and sequentially designed control algorithms is a complex problem that is further complicated by inherent hardware dependencies and real-time requirements. A fine-grained distribution of the algorithms on multiple cores while maintaining deterministic behavior is required but cannot be achieved with state of the art parallelization and scheduling algorithms. This paper presents a new parallelization approach for mapping and scheduling of model-based designs of control algorithms onto ICSs. Since the mapping and scheduling problem is $\mathcal{NP}$-complete, an ACO algorithm is utilized and the solution is validated by experimental results.

1 INTRODUCTION

Industrial Control Systems (ICSs) used to be robust, though rather simple devices regarding their computational complexity and algorithms. On the one hand, open systems like Programmable Logic Controllers (PLCs) enable users to implement logic operations or control sequences easily. On the other hand, there are highly specialized systems for specific tasks like controllers for motion applications. In both cases, the control algorithm itself is executed cyclically with a fixed frequency. Since ICSs are coupled with real machines and processes, the computation of the control algorithm must be completed within the given time. Violating these hard real-time requirements could have severe consequences, such as degradation of the products, damage to machines or even safety issues for humans.

In the past, specialized embedded systems, including custom processors, were used as hardware platforms for ICSs. Advances in communication technologies and decreasing cost for standard IT hardware led to the development of PC-based control systems that are performing the computation on a PC while using inputs and outputs connected through a field bus to interact with the environment. The aforementioned transition is a key driver of the dominant trend in the field of industrial control: the digitalization of production. Various approaches are currently being researched to add value to the production process through digitalization, including big data, traceability, machine learning and human machine interaction. These approaches can be classified as either add-ons or integral modifications of the control algorithm. Add-ons have no direct feedback in the real-time loop of the control systems. In contrast, integral modifications have a direct feedback in the control algorithms and therefore have to fulfill strict requirements regarding the deterministic real-time behavior.

One example of such an integral modification is an innovative approach that is currently being researched by our institute (Abel et al., 2014). This new method for collision avoidance is motivated by the current trend towards custom manufacturing, i.e. producing batches down to lot size one. Hereby, a key cost factor is the changeover time since the first run of a program after a modification usually needs to be performed slowly and with human supervision in order to avoid collisions. In order to make these test runs obsolete, a new method for online collision avoidance using a multi-domain real-time simulation was developed. It uses a parallel simulation of material removal and collision detection, allowing not only a detection of collisions between machine parts, but also collisions with the material. All potential collision items are protected by boundary boxes allowing for detection

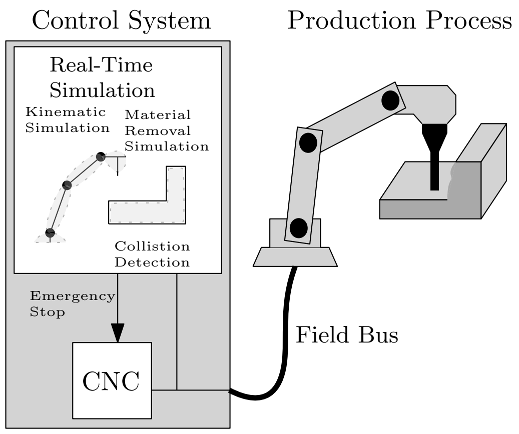

Figure 1: Concept of the real-time collision avoidance.

before a collision occurs. The simulation is tightly coupled to a standard Computer Numerical Control (CNC) which is responsible for the generation of the motion. If an upcoming collision is detected in the simulation, the real system is stopped to avoid damage to the workpiece or the machine. To guarantee this functionality, strict real-time requirements have to be fulfilled during execution. Figure 1 depicts the new approach.

The new collision avoidance method, as well as many other advanced control algorithms, have significantly higher performance requirements than classic control systems, easily exceeding the performance provided by a single-core CPU. Previously, this issue was addressed by constantly increasing the clock frequency. Since this is no longer possible, the performance of standard IT applications is now being improved by the integration of multiple processing core into CPUs. Modern software systems usually consist of a large number of independent tasks that can be executed independently on multiple processing resources and thereby benefit from multiple cores. The algorithms executed on ICSs are different regarding their structure. Usually they are rather sequential and data-dependent, making parallelization more complex. As shown in (Graf, 2014), a straightforward parallelization strategy can easily result in a degradation of the overall system performance rather than an improvement. A further challenge is to maintain the determinism on a multi-core system.

2 SYSTEM ENVIRONMENT

ICSs differ from other system types in various aspects. An efficient parallelization of the control algorithms must consider the specific properties of the algorithms as well as those of the ICSs. ICSs consist of a hardware platform and a software system.

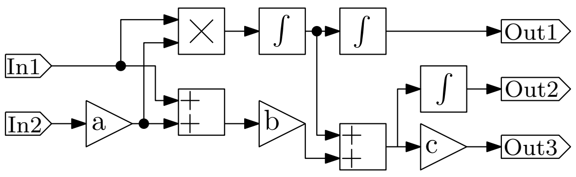

Figure 2: Model-based description of a simple algorithm.

Industrial Control Systems (ICSs) appear to be rather heterogeneous; however, they all share a similar internal structure. They are connected to the physical environment, for example a machine, through inputs and outputs (IO). All types of ICSs run a control software, which cyclically performs three main steps: input sampling, computation of the control algorithm and updating of the outputs. The frequency in which this loop is executed depends on the system and usually ranges in between 100 Hz and 100 kHz.

2.1 Control Algorithm Design

There are different methodologies for the design of control algorithms. Quite common and increasingly popular are model-based design approaches in which the function is defined by a composition of building blocks. New algorithms are composed of existing building blocks.

This design approach is widespread for ICSs and available for various applications. The types of blocks and their capabilities can be very different. They are often organized hierarchically so that the complex blocks are composed of simpler ones. Models consisting of many blocks with simpler functions are often referred to as fine-grained models, while those defined by fewer but more powerful blocks are known as coarse-grained. Analyzing typical control algorithms reveals that many models are locally parallelizable but globally rather sequential. Therefore, an efficient parallelization method must be fine-grained. An example for a fine-grained model is given in Figure 2.

There are two approaches to execute the model-based algorithms on the target ICSs. They are either compiled or executed in a run-time system. Compilation might potentially have a better performance, though it has significant drawbacks regarding usability and real-time behavior. Worst Case Execution Time (WCET) guarantees for compiled algorithms are challenging to provide. A run-time system is configured by instantiating existing blocks. The blocks are tested and their individual WCET is known so that a WCET for the entire algorithm can be guaranteed.

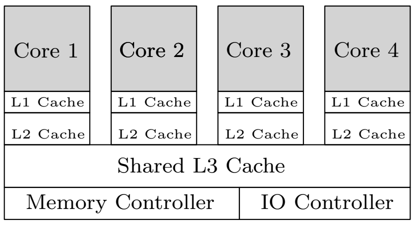

Figure 3: State-of-the-art CPU architecture.

2.2 Hardware

When using a coarse-grained parallelization approach, the CPUs of a multi-core system can be considered as independent units. Fine-grained parallelization is more complex, particularly because the hardware platform has a significant impact.

A processor's performance is limited by its data access rate. Since quickly accessible memory is expensive and limited, current CPUs are equipped with a multi-level cache architecture. Small but fast memories are located close to the processor (registers + L1-Cache). Increasingly larger and slower caches (L2 and L3) are used before accessing the main memory by the memory controller. The access times decrease by magnitudes on each level. On multi-core architectures, the link between the cores is usually the L3-cache. A typical architecture is shown in Figure 3.

As depicted, there is no direct path to switch from one to another processor. In consequence, parallelization can suffer significantly from delays introduced by switching to another core. The effect is more significant when the ratio of switching to sequential computation is higher, as it is the case for fine-grained algorithms.

3 APPROACH FOR MODELING AND PARALLELIZATION

The parallelization approach described in this paper is designed for algorithms using a model-based description. Parallelization of an algorithm for a target platform is an optimization problem. The optimization goal is to map and schedule all function blocks of the algorithm model to resources of the target platform so that the WCET does not exceed the maximum allowed processing time whilst taking dependencies between function blocks of the model into account. This requirement can be solved by optimizing for the lowest processing time and checking if the resulting WCET is below the allowed maximum.

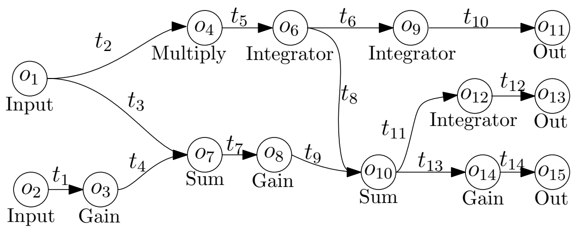

Figure 4: Graph representing the algorithm of Figure 2.

3.1 Model Description

In an algorithmic perspective, the model-based description of the control algorithm can be seen as a graph which, in general, is a directed cyclic graph (DCG). For parallelization, it is required to transform this graph into a directed acyclic graph (DAG). The approach taken in this paper is to ensure that each loop contains at least one delay. This delay is represented as an output and an delayed input in the resulting DAG. Requiring delays in cyclic graphs is a constraint generally accepted by practitioners (Benveniste et al., 2003).

This DAG is the basis for our model of the algorithm and is referred to as G. This type of graph is a common subject of optimization (Kwok and Ahmad, 1998; Bautista and Pereira, 2007). The vertices of G are called operations o_i with $O = \{o_i | 1 \leq i \leq |O|\}$. A function block is represented by one or more operations. Operations are selected from the set of all possible operations Ω. The edges of the DAG are called transitions t_i with $T = \{t_i | 1 \leq i \leq |T|\}$. Therefore, the graph is described by $G = (O, T)$. G represents the algorithm to be scheduled. A graph for the algorithm shown in Figure 2 is shown in Figure 4.

The target system for the algorithm has to be modeled as well. It is called platform P and consists of resources r_i and channels c_i. P is fully described by $P = (R, C)$ with $R = \{r_i | 1 \leq i \leq |R|\}$ and $C = \{c_i | 1 \leq i \leq |C|\}$. Resources are processor cores or other devices that can process operations. A resource can either be able to process all possible operations or only a subset $\omega_{r_i} \subseteq \Omega$. Resources can process one operation at a time. After completion, resources are available for other operations, i.e. they are renewable. Multi-Core-Processors are homogenous. Every task can run on every real-time core at equal costs. However, it is quite common for ICSs to feature dedicated hardware for certain tasks. This specialized hardware can process a subset ω_{r_i} only. An optimization model suitable for industrial control applications must be able to handle such heterogeneous operation environments. Therefore, a relation $\gamma : R \times \Omega \rightarrow \mathbb{N}$ is defined, specifying the execution time for each operation on this resource, which is called cost in our model. If an op-

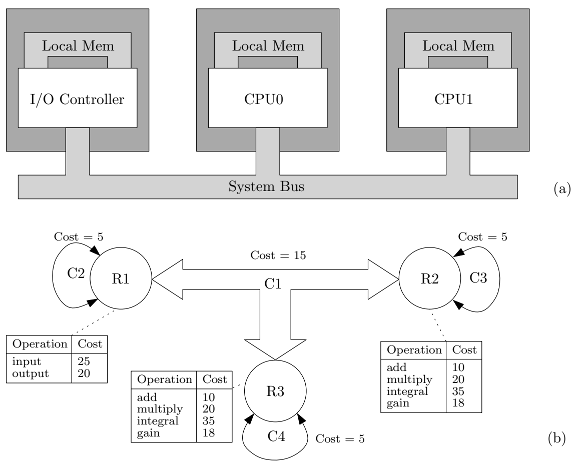

Figure 5: Example of a basic platform and its modeling.

eration cannot be processed by a resource, the cost is ∞.

Resources are connected by communication channels, which have to be modeled as well. Channels can be bus systems, shared memory regions or a local cache. Modeling local cache as a channel allows the optimizer to assign every transition to a channel. In our model, a channel is specified by the connected resources and the costs for using this channel. Similar to a resource, a channel is renewable. Since it is common for a run-time system to connect many resources to a single channel, channels are a bottleneck for optimization.

A model of a basic platform is depicted in Figure 5. It consists of two generic processor cores and a specialized IO controller. While the cores can process all except IO operations, the IO controller can only host IO operations. All resources in the example are connected via a bus system which is modeled as a channel, as is the local memory of each resource.

3.2 Related Approaches

There are similar approaches in literature that also focus on optimizing a DAG. The Assembly Line Balancing Problem (ALBP) in particular has much in common with the problem described above (Boysen et al., 2008). A given set of tasks has to be performed on a pool of workstations. Workstations are similar to resources in our model described above. In ALBP, the tasks have to be processed in a specific order given by a precedence graph, which is a DAG (Boysen et al., 2008). If there are parallel processing routes, an assembly task will unite these routes, resulting in a DAG with a single root. This is different from the problem described in this paper, which can contain multiple inputs as well as multiple outputs. The most common optimization criterion for ALBP is not the processing time, but rather the production rate and the amount of workstations used. Unlike the problem described in this paper, the workstations in ALB can be reused for the next product to be manufactured while the previous product is at another workstation. In our model, resources will be reused for different operations of the same algorithm. Other problems with similar properties like the ALBP exists, like Job Shop Scheduling (Cheng et al., 1996).

An approach much closer to the problem described in this paper was proposed by Ferrandi et al (Ferrandi et al., 2010; Ferrandi et al., 2013). Their work also deals with the problem of assigning operations (or tasks) of a DAG to different components of a heterogeneous embedded system. Unlike the definition above, they consider processing elements to be different from resources. For them, resources are memory regions or Field-Programmable Gate Array (FPGA) logic units which are assigned to the operation to be executed on the associated processing element. As such, they might be renewable or non-renewable. The system described in chapter 2 combined with a model-based approach eliminates the requirement to consider this type of resource as each processing element already has a non-changeable memory region assigned to it. The approach proposed by Ferrandi et al. assumes that the amount of data transferred between tasks varies leading to task- and channel-dependent costs for channels. The run-time model that is the target of our optimization has a fixed data package to be transferred between operations, thereby making channel costs idependent from tasks. Ferrandi et al. consider the implementation of an application on a generic system, while our problem targets a run-time system with preexisting elements for processing operations and transitions.

Although this work focuses on multi-core architectures which can be considered homogenous with regard to the operations they can process, the embedded nature of control systems requires a heterogeneous approach. If cores are distributed over multiple processors, certain IOs might be restricted to certain cores. Therefore, resources have constraints on which tasks or operations can be performed. An advantage of the optimization model described in this paper is that its operations are directly derived from the function block diagram of the algorithm. Unlike in other algorithms, tasks do not have to be identified by the user or a solver. This advantage stems directly from the model-based approach.

4 PARALLELIZATION ALGORITHM

As noted by others, scheduling and mapping a DAG to parallel processing units is an $\mathcal{NP}$-complete problem (Bernstein et al., 1989). Thus, exact solutions cannot be determined within a reasonable amount of time. For problems with large graphs, bio-inspired meta-heuristics like Simulated Annealing (SA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) have been used successfully (Scholl and Becker, 2006; Wang et al., 1997; Zheng et al., 2006; Clerc, 2004). GA has been used to solve mapping and scheduling problems for heterogeneous systems with considerable success (Wang et al., 1997). However, ACO has been found superior to other approaches with respect to computation time and quality of the solution (Ferrandi et al., 2010). Because of its superior performance in solving similar problems, ACO is the chosen algorithm to solve the optimization problem of mapping and scheduling graph G to platform P, namely the ACO variant Ant System (AS) (Dorigo et al., 1996).

4.1 Ant System Description

AS is an optimization algorithm based on a set of independent agents (called ants) cooperating through indirect communication. Given m agents, each agent k traverses the solution graph S to the given problem and generates a solution, which is called path ξ_k. A path is an ordered list of the vertices s_i of S with $S = \{s_i, s_{i,j} | 1 \le i \le |S|, 1 \le j \le |S|\}$ and $s_{i,j}$ being the edges of S. The performance of ξ_k is evaluated and is used to modify the so called pheromone values which are assigned to each edge of S. The update function for the pheromone values is given by

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{1}$$

with

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{Q}{L_k} & \text{if ant } k \text{ moved from } s_i \text{ to } s_j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Here $\tau_{i,j}$ is the pheromone value on edge $s_{i,j}$, ρ is the pheromone decay value, $\Delta\tau_{i,j}^{k}$ the amount of pheromone added by ant k, L_k is the performance of ξ_k and Q is a weighting factor for L_k (Dorigo et al., 1996).

The pheromone value $\tau_{i,j}$ is the communication mechanism for the agents. Each agent evaluates the pheromone values when building its solution ξ_k. Starting from a vertex s_i of S, it selects any of the vertices of S that are connected with s_i via an edge $s_{i,j}$ using

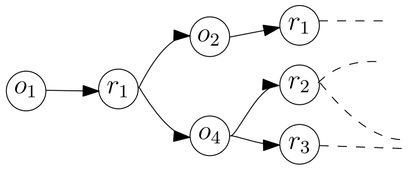

Figure 6: Beginning of the solution graph for Figure 4.

$$p_{i,j}^{k} = \begin{cases} \frac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum\limits_{s_a \in \mathbf{N_{Rk}}} \tau_{i,a}^{\alpha} \cdot \eta_{i,a}^{\beta}}, & \text{if } s_j \in \mathbf{N_{Rk}} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

with

$$\eta_{i,j} = \frac{1}{d_{i,j}} \tag{4}$$

$p_{i,j}^{k}$ is the probability for agent k to choose $s_{i,j}$ for the next step, $\eta_{i,j}$ is the heuristic value for $s_{i,j}$, α and β are weighting factors for the pheromone value and the heuristic value respectively, $\mathbf{N_{Rk}}$ is the set of valid edges to choose from, and $d_{i,j}$ is the cost value for edge $s_{i,j}$ (Dorigo et al., 1996). Valid edges are all edges connected with the current vertex excluding those that lead to vertices visited before. The last condition is fulfilled for all edges when S is a DAG.

4.2 Creation of the Solution Graph

To apply ACO to our problem, a solution graph S has to be constructed from G and P that can be traversed by the ants. To find a solution to the problem given in chapter 3, traversing S must provide a valid scheduling of G as well as a mapping of all elements of G to P while preserving the precedence of G. It also assigns each element of O to an element of R and each element of T to an element of C. One possible solution graph starts with an element o_i of O without predecessors and adds it to S as s_1. It then adds vertices for each element of R that o_i can be assigned to and creates edges $s_{1,j}$ for each connection between each of the resource vertices and s_1. Another element of O is then selected from a list of operations without unprocessed predecessors and the process repeats until all elements of O are added as vertices into S and followed by at least one vertex representing an element of R. An example is given in Figure 6.

So far, transitions have not been considered when building S. Transitions can be represented similarly in S to the operations. While this approach allows for a very simple and straightforward construction of S, it also increases the risk of creating an invalid solution: An agent might select a channel for a transition that could lead to the selection of a task for which no resource is reachable from the selected channel. While this can be detected, it results in many unsuccessful iterations to create a solution. Additionally, this approach further increases the solution space. To avoid these problems, neither C nor T is represented in S. Instead, for every t_i, the fastest channel and the time for the data transfer is used to calculate the performance of ξ_k. As a result, no pheromone value is assigned to the selection of channels. They influence the pheromone values of $s_{i,j}$ by their impact on the performance of the solution.

To simplify the creation as well as the traversing of S, two additional operations are added to O, which are called Nest and Food. Transitions from the Nest are added to T for all operations o_i of O that do not have a predecessor. Similarly, transitions are added to T for all operations o_i of O without a successor (Chiang et al., 2006). To build S, an additional resource r_i is defined that can process these new operations. The resulting DAG now has a single start vertex and a single end vertex.

The heuristic values $\eta_{i,j}$ of all edges of S have not yet been considered. For problems like the Traveling Salesman Problem (TSP), these values are constant and do not change when building a solution (Dorigo et al., 1996). However, for the problem described in this paper, the heuristic values for each $s_{i,j}$ of S largely depend on the path ξ. Since the heuristic values are the costs for traversing a specific edge $s_{i,j}$, they directly correlate with the time between the end of processing s_i and the beginning of processing s_j. Ideally, s_j is processed one execution step after s_i has been processed. However, the channel execution time has to be considered as well, if the s_j is an operation and one or more transitions are required for s_j to be processable. In this case, the heuristic values not only depend on the static channel cost, but also on the availability of the resource used for s_j and the availability of the channel used. Both might be occupied by parallel running operations or transitions, impacting the overall performance of the solution. As a result, edge $s_{i,j}$, which led to a good performance for a previously found solution, might worsen another solution, depending on ξ. This requires a reasonable balance between the heuristic value and the pheromone value.

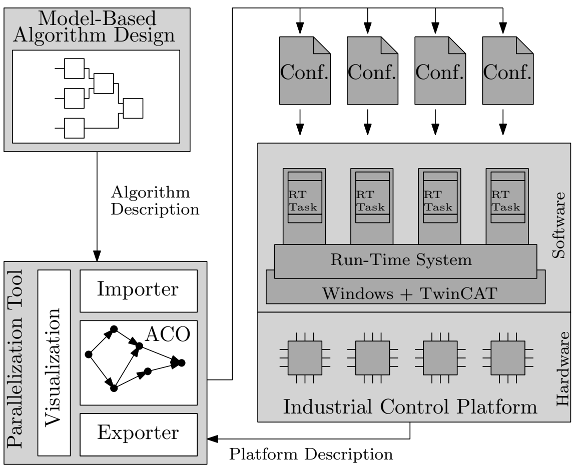

Figure 7: Overview of the sample implementation.

5 EXPERIMENTAL IMPLEMENTATION

Testing and benchmarking the developed parallelization algorithm under real conditions requires additional functions besides the implementation of the algorithm itself. The overall structure is depicted in Figure 7.

A model-based description of the algorithm created in Matlab Simulink and a description of the target platform is the input for the parallelization algorithm. A dedicated parallelization tool translates the imported model to a DAG as described in chapter 3 and then performs the optimization according to chapter 4. It generates a configuration for the run-time system. The run-time system is located on a control platform and executes the algorithm in real-time.

5.1 ICS Platform

The selected target platform for the test setup is an IPC equipped with an Intel i5 quad-core processor. Inputs and outputs are connected through a field bus interface. For the experimental setup, Windows and the real-time control system TwinCAT are used. The run-time system is implemented as a module of the RTOS used and performs the actual computation of the control algorithm. It is composed of run-time tasks, one per core. Each run-time task must be configured, defining which operation is executed in which order. In addition, there are channels between all of the tasks, allowing for communication between the cores. Ports are available to the RTOS to couple the run-time system with the existing IOs and other modules, like PLCs or CNCs.

Table 1: Performance Gains for Example Algorithm in Figure 2.

Platform	Costs [%]				Coefficient of variation [%]			
	$\beta = 0.8$	$\beta = 1.1$	$\beta = 1.2$	$\beta = 1.6$	$\beta = 1.2$	$\beta = 0.8$	$\beta = 1.1$	$\beta = 1.6$
Single-Core	100.0	100.0	100.0	100.0	0.0	0.0	0.0	0.0
Dual-Core	80.6	72.1	70.6	68.6	4.6	5.3	4.5	4.6
Quad-Core	60.4	60.8	58.2	59.2	6.7	5.3	5.9	6.2

Table 2: Performance Evaluation for Collision Avoidance Algorithm in Figure 8.

Platform	Costs [%]	CPU Time [us]					CPU Time [%]	Max f [Hz]
		Core 1	Core 2	Core 3	Core 4	Total		
Single-Core	100	3100	-	-	-	3100	100	322
Dual-Core	51.7	1650	1600	-	-	3250	55	606
Quad-Core	27.6	850	870	790	930	3440	31	1075

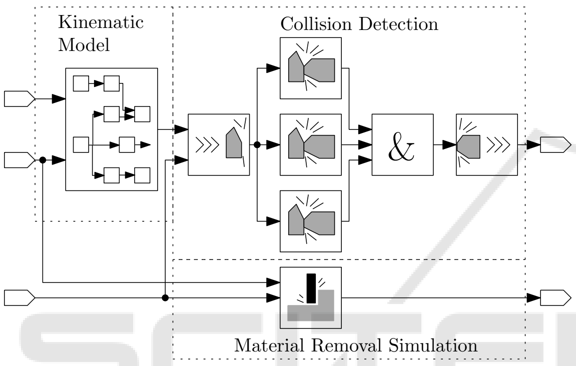

Figure 8: Description of the collision avoidance algorithm.

5.2 Reference Algorithms

Various control algorithms were implemented and parallelized. The algorithm shown in Figure 2 was used as a simple test case for an highly parallelizable algorithm. Additionally, small example systems were used to test the optimizer. Finally, the collision avoidance method described in the introduction was parallelized. The model-based description of the algorithm is depicted in Figure 8.

There are three sub-models included in the algorithm. A kinematic model is used to trace the position of all machine parts. Based on the actual position of the tool and initial information about the work piece, a material removal simulation is performed. The information about the machine parts as well as the actual shape of the work piece are used as inputs to the collision detection algorithm. The algorithm consists of multiple steps: First, possible collision pairs are identified. These pairs are then distributed over multiple collision detection tasks. Finally, information about potential collisions is collected.

5.3 Experimental Results

First, the quality of the parallelization of the algorithm in Figure 2 was evaluated. The pheromone value scaling factor α was set to 1.0 while the heuristic value scaling factor β was varied. The algorithm was parallelized to run on 1, 2 or 4 cores. For each solution the highest costs of all cores were taken as the WCET. Since the costs for single core execution are constant, they are taken as reference and each other solution is given in comparison to the single core solution.

The algorithm is dividable in two parallel paths and therefore an improvement of approximately 45 % can theoretically be achieved using two cores. However, adding additional cores should not further improve the performance due to the algorithms structure (theoretic value for 4 cores: 47 %). Table 1 shows the resulting relative costs for the algorithm in Figure 2, also including the coefficient of variation based on a representative number of test runs. Depending on the value of β, performance improvements for two cores vary between 19 % and 31 %. While not optimal, a significant performance gain can be achieved by automatic parallelization. Solutions for four cores are much closer to the theoretical optimum, because the larger solution space makes it more likely to not converge into a local minimum. The necessity of avoiding local minima also influenced the final choice of β. While higher values of β produce better results for the example algorithm, they put a higher weight on the heuristic value, making it more likely for the ants to choose a locally optimal path, which might not lead to a globally optimal path. Choosing $\beta = 1.2$ proved to be a reasonable compromise for the algorithm evaluated in Table 1 as well as other algorithms. This value was therefore chosen for the parallelization of the collision avoidance algorithm given in Figure 8.

Again, the algorithm was parallelized for 1, 2 or 4 cores. The run-time system was configured to run the same number of task. Thereby, the system is behaving as if single, dual or quad-core systems were used. The remaining cores were not used by the run-time system. The execution time on every core was measured. The results are shown in Table 2. The overall performance defined by the maximum possible control frequency increases with every additional core. Looking at the total execution time reveals that a certain overhead is generated by the parallelization. This was expected due to the necessary communication between the cores. Comparing the highest CPU Time of all cores with the estimated costs reveals that the cost estimate is only slightly optimistic. Most importantly, the generated solution allows real-time execution of the collision avoidance algorithm for complex production processes.

6 CONCLUSION

In this paper, an approach for fine-grain parallelization of control algorithms using ACO-based optimization is presented. The goal of this parallelization is to enable ICSs to benefit from increasing hardware performance to be used for new algorithms. A collision-avoidance-algorithm was used as a test case for parallelization of machine tooling algorithms. It was shown that the approach successfully parallelizes the algorithm and enables ICSs to benefit from multi-core CPUs.

While the presented research focuses on proving the applicability of ACO for the parallelization problem, optimizing ACO for the specific problem will be the focus in the future. Furthermore, the approach will be extended for systems utilizing dedicated hardware accelerators based on FPGAs. It is intended to implement the parallelized collision avoidance algorithm into an industrial real-time hardware-in-the-loop simulation environment for ICSs development.

REFERENCES

Abel, M., Eger, U., Frick, F., Hoher, S., and Lechler, A. (2014). Systemkonzept für eine echtzeitfähige Kollisionsüberwachung von Werkzeugmaschinen unter Nutzung von Multicore-Architekturen. In *Proceedings of SPS IPC Drives 2014*, pages S. 441–445–. Apprimus Verlag.

Bautista, J. and Pereira, J. (2007). Ant algorithms for a time and space constrained assembly line balancing problem. *European Journal of Operational Research*, 177(3):2016–2032.

Benveniste, A., Caspi, P., Edwards, S. A., Halbwachs, N., Le Guernic, P., and de Simone, R. (2003). The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1):64–83.

Bernstein, D., Rodeh, M., and Gertner, I. (1989). On the complexity of scheduling problems for parallel/pipelined machines. *IEEE Transactions on Computers*, 38(9):1308–1313.

Boysen, N., Fliedner, M., and Scholl, A. (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics*, 111(2):509–528.

Cheng, R., Gen, M., and Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms – I. representation. *Computers & industrial engineering*, 30(4):983–997.

Chiang, C.-W., Lee, Y.-C., Lee, C.-N., and Chou, T.-Y. (2006). Ant colony optimisation for task matching and scheduling. *IEE Proceedings - Computers and Digital Techniques*, 153(6):373.

Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering*, pages 219–239. Springer.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

Ferrandi, F., Lanzi, P. L., Pilato, C., Sciuto, D., and Tumeo, A. (2010). Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(6):911–924.

Ferrandi, F., Lanzi, P. L., Pilato, C., Sciuto, D., and Tumeo, A. (2013). Ant colony optimization for mapping, scheduling and placing in reconfigurable systems. In *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, pages 47–54. IEEE.

Graf, R. (2014). Chancen und Risiken der neuen Prozessor-Architekturen. *Computer-Automation*.

Kwok, Y.-K. and Ahmad, I. (1998). Benchmarking the task graph scheduling algorithms. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 531–537. IEEE Comput. Soc.

Scholl, A. and Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693.

Wang, L., Siegel, H. J., Roychowdhury, V. P., and Maciejewski, A. A. (1997). Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of parallel and distributed computing*, 47(1):8–22.

Zheng, S., Shu, W., and Gao, L. (2006). Task scheduling using parallel genetic simulated annealing algorithm. In *Service Operations and Logistics, and Informatics, 2006. SOLI'06. IEEE International Conference on*, pages 46–50. IEEE.