Analysis of CNC Software Modules Regarding Parallelization Capability

Matthias Keinert*, Benjamin Kaiser, Armin Lechler, and Alexander Verl

Institute for Control Engineering of Machine Tools and Manufacturing Units
University of Stuttgart
Stuttgart, 70174, Germany

ABSTRACT

The software design of current CNC systems is of limited suitability for real parallel execution on multicore systems. Even though CNC systems are rudimentarily designed in a modular way, a good load balancing and therefore an efficient use of multiple processor cores is not satisfactorily to be accomplished, as the modularization is not sophisticated enough. A parallelization of CNC functions and algorithms would remedy this deficit. This paper presents an analysis on which parts of a CNC system show the capability for parallelization. Furthermore an approach is presented on how the parallelization of a specific function, namely the look-ahead function, can be accomplished.

1. INTRODUCTION

Computer Numerical Controls (CNC) are used in the area of machine tools and machining centers. Their main task is the trajectory generation to realize a relative movement between tool and workpiece for the manufacturing process. There are several requirements a CNC has to fulfill regarding this task:

First of all, the trajectory generation has to show a deterministic time behavior as the machine tool's drives that realize the tool path, demand new setpoints in real-time and within a defined cycle time. That implicates that the data throughput within the CNC has to be maintained by all means to provide each cycle new setpoints to the drives avoiding violations of the real-time constraints. In ordinary manufacturing processes this requirement does not constitute a problem. However, in manufacturing processes that are realized at high feedrate (HSC – High Speed Cutting) and with short path segment lengths (e.g. when manufacturing freeform surfaces) the data throughput and the real-time requirement represents a challenge [1].

Furthermore, the trajectory generation is heavily responsible for the quality and the productivity of the manufacturing process. Quality and productivity are increasing when smooth-curvature trajectories are generated that are executable at high feedrate without exceeding the machine tool's dynamic ability. These characteristics lead to short production cycles, high surface quality and low machine strain [2].

Last but not least the trajectory generation has to be capable to generate the setpoints for different levels of machine kinematic complexity. CNCs are used to control machines beginning at ordinary three axes kinematics up to machining centers with multiple spindles and a huge amount of axes.

All these requirements reveal that the trajectory generation is a rather complex issue. Nowadays, the functionality of CNC systems is mostly realized in software that is executed on a PC-based system platform. That can be an embedded PC, a particular industrial PC or even a standard desktop PC. In either case, the CNC software, respectively the complex software functions that realize the trajectory generation, have to be executed on system platforms that provide enough processing power to fulfill said requirements. Looking at available system platforms it can be stated that multicore system platforms would provide sufficient processing power for CNC systems [3]. However, the change from single-core to multicore processors in the area of CNCs and industrial automation systems in general is just at its very beginning. That is because the change from single-core to multicore constitutes a challenge especially in the area of real-time systems. Due to the hardware architecture and its characteristics in executing software in parallel, it can

* Corresponding author: Tel.: (+49 711) 685-84625; Fax: (+49 711) 685-82808; E-mail: matthias.keinert@isw.uni-stuttgart.de

happen that the theoretically unlimited performance increase provided by multicore systems in praxis turns into the opposite and leads to longer cycle times [4]. That is mainly the case when the software is not prepared for real parallel execution but appears as monolithic structure that is designed for serial execution. In the area of CNC systems this kind of software structure can be discovered. Even though there is some level of modularization within the CNC architecture accruing from the beginnings of the CNC era and the multiprocessor control system (MPST) design, the algorithms that are responsible for the trajectory generation are not.

This paper presents an approach on how CNC software can be designed to be parallel executable on multicore system platforms. The paper is organized as follows: In section 2 the software architecture of CNC systems is described. Thereof derived section 3 describes the parallelization capabilities of dedicated software functions. Section 4 demonstrates at the example of the look-ahead function the possibilities of software parallelization. The paper closes with a summary and an outlook.

2. CNC SOFTWARE ARCHITECTURE

As the CNC software architecture is primarily vendor specific, literature depicts different kinds of software design. Nevertheless the main components of a CNC system do not seriously differ. A CNC system is first of all composed by a user interface, denominated as HMI (Human Machine Interface) or MMI (Man Machine Interface) that allows user interaction with the CNC, by a logic control, known as the PLC (Programmable Logic Control) handling all IO operations, and by a motion control, known as the numerical control kernel (NCK) that cares about trajectory generation [5] [6]. PLC and NCK are responsible for controlling the machine, respectively the drives and actors, and for capturing the machine status, respectively all sensors. NCK and PLC are therefore executed in the real-time space of the CNC system, whereas the HMI does not have severe requirements concerning deterministic time behavior and can be executed in the user space of the CNC system. As the focus of this paper is situated in the motion functionality, PLC and HMI are not further considered.

The architectural design of the NCK differs widely in theory and praxis. From a functional point of view, however, the NCK is primarily a composition of the following functions [5]:

- Interpreter: The interpreter is responsible for decoding the NC-program holding technological and geometrical information about the manufacturing process. The interpreter handles data migration from the NC-program buffer and decoding of the NC-program in consideration of subprograms and NC cycles. The decoded information is stored in some sort of data structure so that it can be passed to the subsequent CNC functions caring about trajectory generation.
- Tool compensation: NC-programs are mostly generated independent from tool radius and length and from the workpiece clamping position. The tool compensation accommodates the geometrical data by considering offsets and geometrical tool data. It calculates an equidistant tool path.
- Smoothing: Smoothing is the first part of the velocity planning functionality of a CNC. This function adjusts the given tool path in a way that the physical limitations of the machine tool are not exceeded. Smooth path segments are used to replace segments that would lead to discontinuities in the first, second up to the third derivative, e.g. sharp corners. The smoothing functionality must be explicitly activated.
- Look-ahead: The look-ahead function is the second part of the velocity planning functionality. It calculates an anticipatory feedrate profile representing feedrate limitations. That allows applying the maximum permissible feedrate for the pre-calculated path segments considering the path characteristics and yet permits an exact stop in appropriate cases, e.g. at sharp corners.
- Slope: Finally, the slope function is the third and last part of the velocity planning functionality. It is responsible for generating smooth velocity profiles avoiding acceleration steps and infinite jerk values. The acceleration is increased by using slopes.
- Interpolation: Within the interpolation axis command values are generated based on a specific algorithm. Mostly used is a linear and circular interpolation method, however there are more elevated methods like spline interpolation.
- Transformation: The tool path trajectory planning has been executed without consideration of the machine tool's kinematic so far. The axis based command values are subjected to an appropriate transformation.

Regarding the architectural design literature describes different arrangements of the software functions within the CNC system. This paper assumes an architecture as described in [5]. The dedicated software functions are grouped in respect to their required time behavior in synchronous and asynchronous functions. Synchronous functions are meant to be executed once each control cycle whereas asynchronous functions can be interrupted and possibly be executed over several control cycles. Interpolation and transformation are part of the synchronous functions, whereas NC code interpretation, tool compensation and velocity planning belong to the asynchronous functions. The data exchange between the considered software functions is realized by shared memory or when it comes to the data exchange between asynchronous and synchronous software functions and as well between the functions of the asynchronous function group, by some sort of buffer, e.g. FIFO (Fist In First Out). Figure 1 illustrates the CNC software architecture.

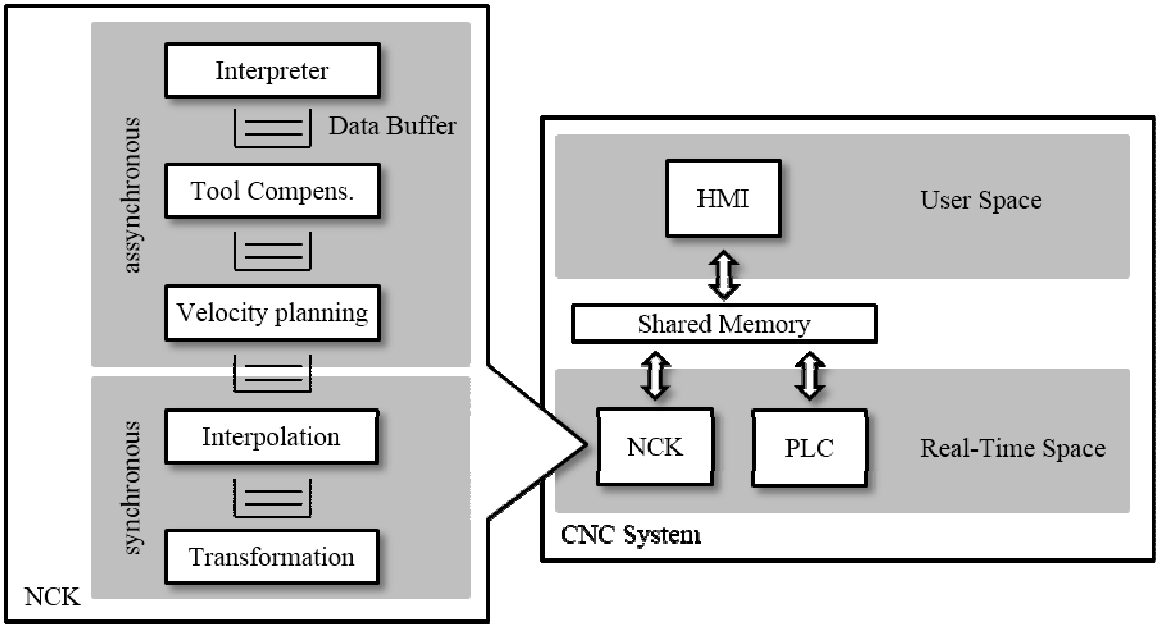

Figure 1. CNC software architecture with focus on the numerical control kernel (NCK).

The arrangement of the software functions into synchronous and asynchronous functions is more or less the task affiliation at the same time, describing on how the operating system executes these functions. Each task is assigned a priority that defines the invocation order, and a cycle time that defines the invocation frequency. The execution of the NCK as a whole as well as of each task is controlled by some sort of task control. The task control decides on which task or within the task on which function has to be executed next to maintain the CNC system operation.

Considering this kind of CNC system architecture it can be stated that in the best case the NCK consists of two tasks that are executable on different cores. Within each task, however, complex operations are executed. Further fragmentations of these tasks are desirable to modify the load balancing when it comes to the task assignment to the different cores. Many small task portions, executable in parallel, would lead to a high flexibility when it comes to the use of multicore systems with different number of cores and the CNC system as a whole would benefit from a multicore performance increase.

3. PARALLELIZATION CAPABILITIES

Based on the CNC system architectural design there are three levels of distributing the CNC software to multiple cores of a system platform: First of all there is the distribution on the main component level, secondly on the task (synchronous and asynchronous task) level and finally on an algorithmic level.

The distribution on the main component level is the most obvious possibility to exploit a multicore system. Some CNC systems currently available are already offering that possibility [7].

Regarding the NCK component a fragmentation of this component into two tasks corresponding to the distinction into synchronous and asynchronous functions is present and constitutes a possibility for distribution on the task level.

However, it has to be secured that the data flow between the tasks is maintained as the synchronous task requires a time deterministic execution. Any delay in providing data to the synchronous functions must not occur.

However, the distribution of the CNC software on the first levels is not optimal as computationally intensive algorithms of the CNC system are still executed sequentially. A further fragmentation of tasks and functions into parallel executable parts could lead to an efficient use of multicore systems. There are already approaches to sort out single functionalities, e.g. the interpreter, executing them in a single thread [1]. Beyond that we suggest a parallelization on the algorithmic level, as well. Even though the CNC system is first of all a sequential system there is especially in the area of the asynchronous functions the possibility to work in parallel at a specific task. This applies on the interpreter, tool compensation, smoothing and the look-ahead. The approach is to execute a specific function several times in an own thread or any other parallel executable construct. Each of these instances gets a part of the available data from the previous function. Each data item is tagged with a unique identifier so that a composition of the output data of each instance is possible. In the case that data items cannot be considered isolated an overlapping area is suggested so that each instance can access information that lies beyond its area of responsibility. Regarding the execution of the parallelized software functions it has to be assured that the priorities of the threads, executing parts of any specific function, do not exceed the priority of the superior task. Thereby threads do not restrain other threads, affiliated to tasks with higher priorities, from execution. Above all the task control has to make sure that the execution of threads within a task is proceeded correctly.

The following section depicts at the example of the look-ahead function an implementation of a parallel executable asynchronous CNC function.

4. LOOK-AHEAD FUNCTION FOR REAL PARALLEL EXECUTION

The look-ahead function calculates the maximum permissible feedrate for given interpolation points in advance. This allows applying a feedrate that does not exceed the dynamic abilities of the machine tool but stays close at the programmed feedrate. Hence results a good surface quality, high productivity and small machine strain for the given tool path. The further description of the look-ahead function is based on [6].

The look-ahead function works on the data that is provided by the interpreter, maybe already processed by the tool compensation. This data represents the decoded motion commands from the NC-program and mainly consists next to some auxiliary information that is currently not important for the look-ahead function, of start and end position of each path segment and programmed feedrate. The look-ahead function stores depending on its configuration a defined number of data elements into its own data buffer. If there are less than the configured data elements available the look-ahead function only stores the available data elements, but at least two. Figure 2 gives an overview on important elements and definitions relevant for the look-ahead function.

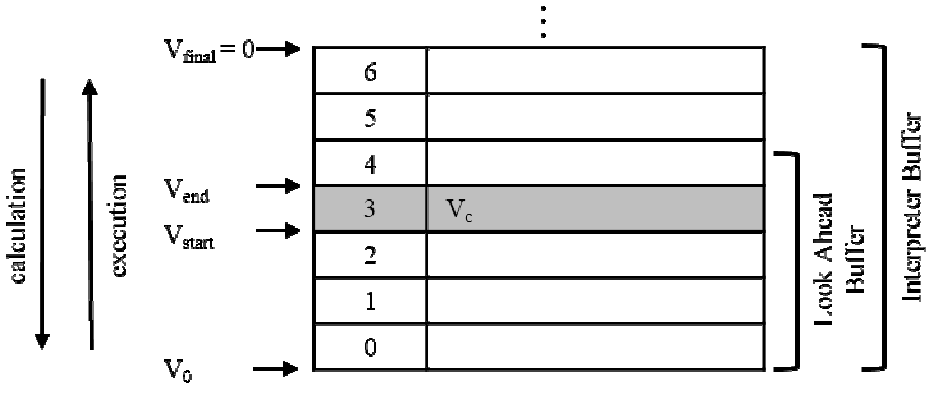

Figure 2. Elements and definitions of the look-ahead function.

Based on this data selection the maximum permissible feedrate profile is calculated beginning from the last entry up to the current entry in the look-ahead buffer. The look-ahead function calculates for each two data elements the feasible corner speed V_1 based on the angle between the path segments considering the machine tool's dynamic abilities which are the maximum path acceleration and the sampling time for position control. Additionally the look-ahead function calculates a path segment's start velocity V_2 that can be achieved by applying the maximum acceleration or

deceleration on the whole path segment length. The current segment's end velocity V_{end} is assumed as the start velocity of the previous, already calculated segment. Finally the look-ahead function calculates for each path segment in the look-ahead buffer the maximum possible start velocity V_{start} as the minimum of V_1, V_2 and the programmed feedrate V_C. There are two special cases to be considered: First of all the end velocity for the last data element in the look-ahead buffer, that is the element which is considered first due to the inverse calculation, is assumed as zero. Secondly the end velocity of the first data element, considered last in the look-ahead calculation, is calculated based on the maximum acceleration on the whole path segment length, limited by the programmed feedrate. Output of the look-ahead function is a data structure holding mainly start velocity V_{start} and end velocity V_{end} for each path segment.

A look-ahead function modified for parallel execution differs thereby that the available data from the interpreter buffer is distributed to a configurable number of look-ahead data buffers. This goes along with a fragmentation of the look-ahead function in parallel executable threads. To consider the above mentioned special cases the thread needs to get the information on which part of the data fragmentation it is working. The distribution of the data has to follow a specific logic: The look-ahead function as described above follows a sequential software design. The calculation for the maximum permissible velocity for each path segment is based on the velocity of the previous path segment. In a parallel design each look-ahead thread is working on an excerpt of the common data buffer and therefore misses appropriate information about the velocity at the border of each buffer. For this reason an overlapping area is defined that covers that much data elements that the realized path length within these data elements is sufficient for accelerating or decelerating the machine up to the maximum permissible feedrate (that is the programmed feedrate). Each thread is informed about the number of overlapping data elements so that this area is not considered twice but can be discarded in one thread's output buffer. After the common data buffer fragmentation each look-ahead thread works on its own data buffer including the overlapping area. If the overlapping area is not defined big enough due to a configurable overlapping limitation the maximum permissible feedrate cannot be reached. As a consequence it comes to a drop in the look-ahead velocity profile. That does not imply a drop in the executed velocity profile as the look-ahead velocity profile only represents a velocity limitation. After having processed each look-ahead data buffer the overlapping area in the output data buffer of each look-ahead thread is discarded based on the area responsibility. The common look-ahead buffer is terminated by assembling the common look-ahead output data buffer based on a stored unique block number. Figure 3 illustrates the concept of a look-ahead function for parallel execution.

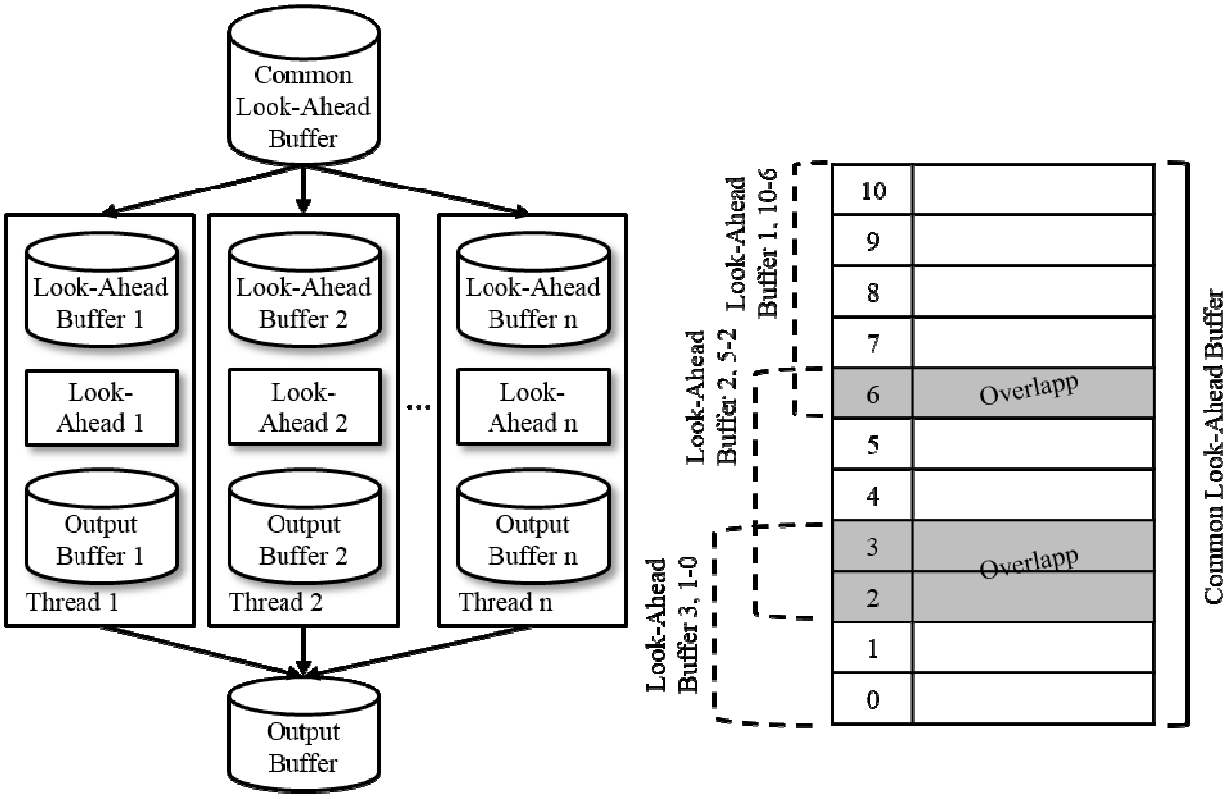

Figure 3. Design of a look-ahead function for parallel execution.

5. CASE STUDY

In the current development state the look-ahead function for parallel execution is implemented in a non-real-time environment using Microsoft Windows 7 and OpenMP for distributing the single threads of the look-ahead function, in the nomenclature of OpenMP called sections, dynamically to the available cores. The hardware is an Intel Core i5 Quad Core processor with 3.2GHz. The look-ahead function is considered isolated and not within a complete NCK software distribution. The interpreter data buffer is filled from a NC-program before the look-ahead function starts working on the available data. The number of look-ahead data elements was set to 300, the minimum overlapping area to 5, the maximum overlapping area to 50 data elements. Both, the number of look-ahead data and the size of the overlapping area are freely configurable. The analysis of the look-ahead performance is based on a test workpiece illustrated in figure 4.

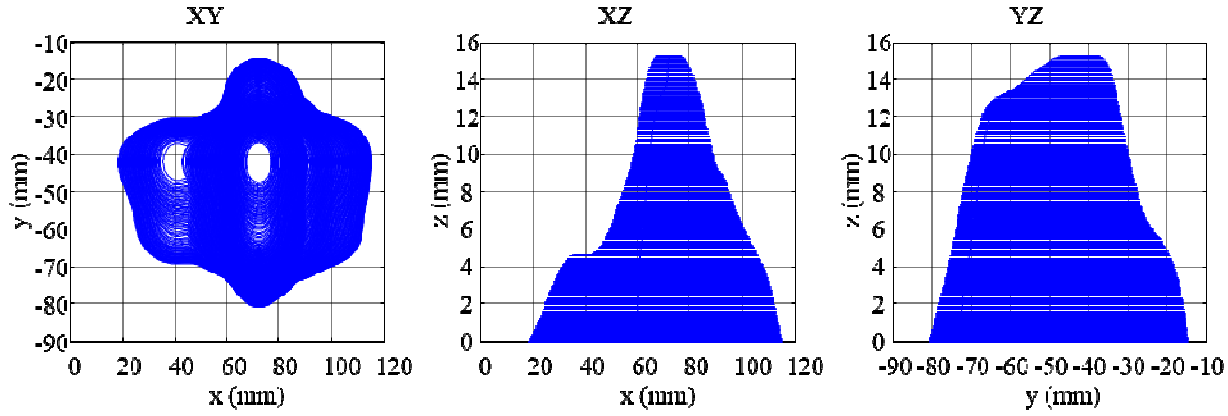

Figure 4. Test workpiece for the analysis of the parallel executable look-ahead function.

The performance increase of a real parallel execution of the look-ahead is achieved not before a dedicated time of execution or rather a dedicated number of data elements as illustrated in figure 5. That is arising from the multicore architecture and its characteristics concerning thread initialization, context changes, cache misses or similar, as well as from the necessity to work on some additional tasks like filling each thread's buffer, working on the overlapping area and assembling the common look-ahead buffer. In other words, a parallel execution of CNC functions is only be worth it if each thread, working on a part of the function, achieves continuously full capacity utilization. If not, the organizational overhead due to parallel processing is proportionally large. Even though it can be assumed that, at least regarding the asynchronous functions, this overhead do not have severe consequences on the system reliability a strategy has to be developed leading to the ideal level of parallelization.

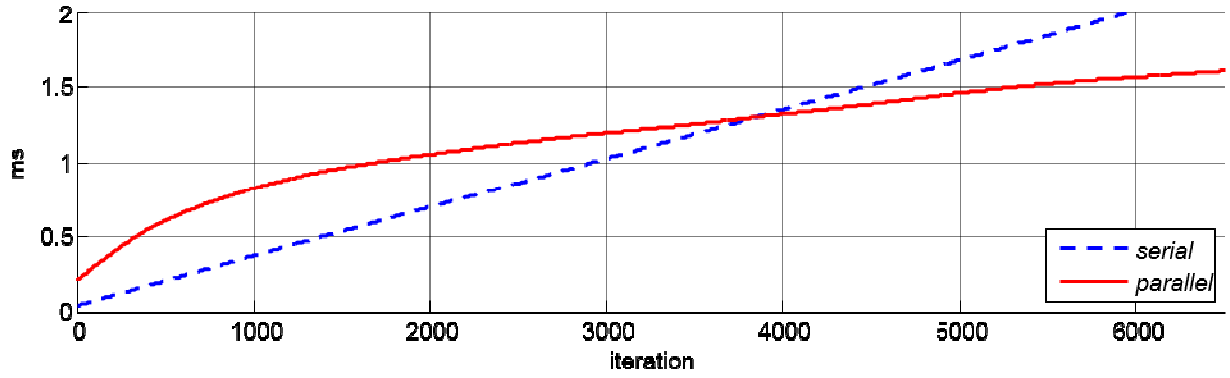

Figure 5. Comparision of serial and parallel look-ahead efficiency depending on execution time.

Comparing the results of the sequential and the parallel look-ahead as illustrated in figure 6 it can be proved that the look-ahead functionality remains equal as the outputs do not differ, at least if the overlapping area of the parallel

look-ahead is sufficiently rated. However, as illustrated in figure 7, the used overlapping area remains in a reasonable area.

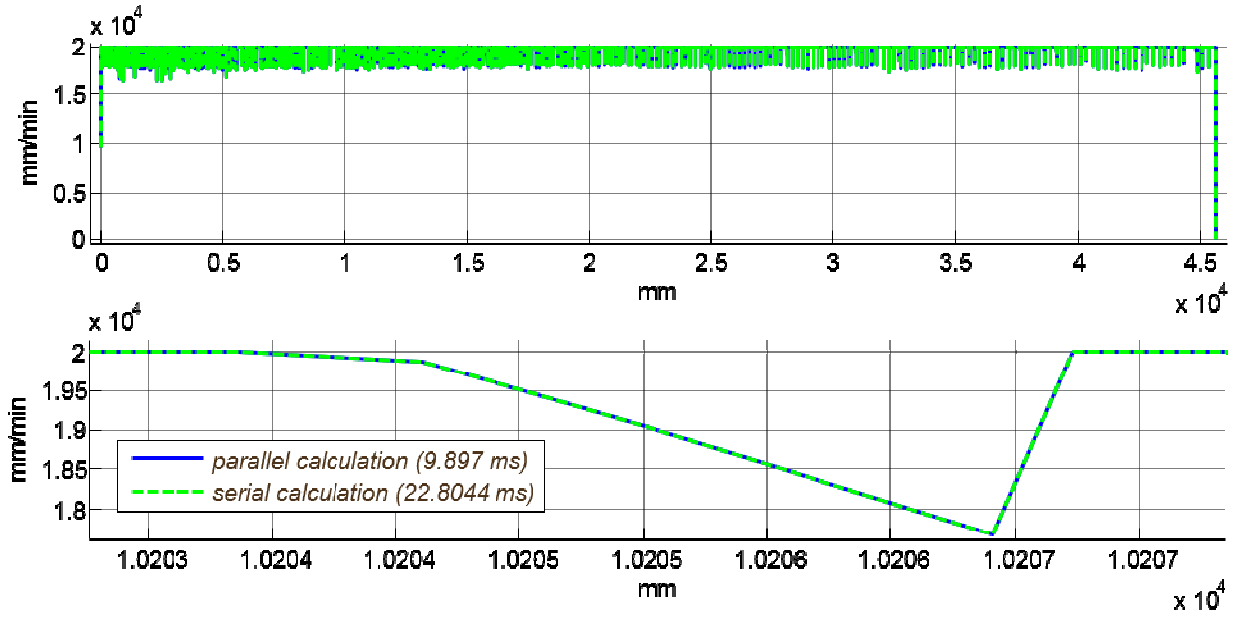

Figure 6. Comparision of serial and parallel look-ahead feedrate profiles.

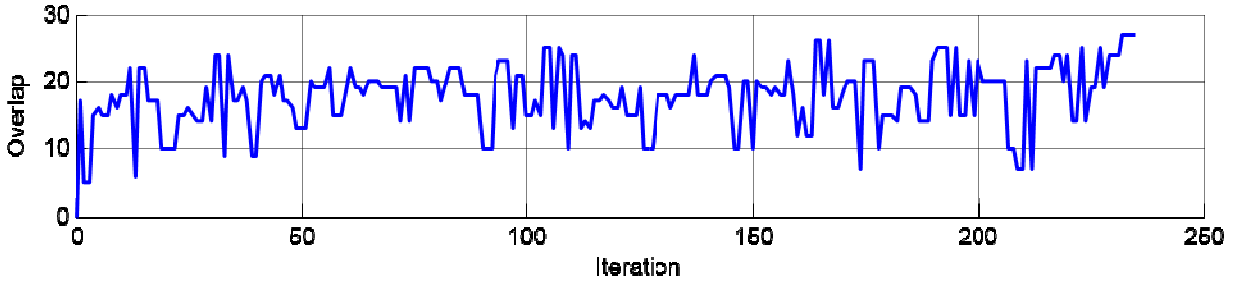

Figure 7. Parallel look-ahead overlap buffer size.

Finally figure 8 illustrates the real parallel execution of the look-ahead buffer. For each task a time stamp was recorded on entry and on exit of the task. While the graph on the left hand side in figure 8 illustrates the thread execution for the complete interpreter buffer, the graph on the right hand side illustrates the thread execution while the last data items in the interpreter buffer are processed.

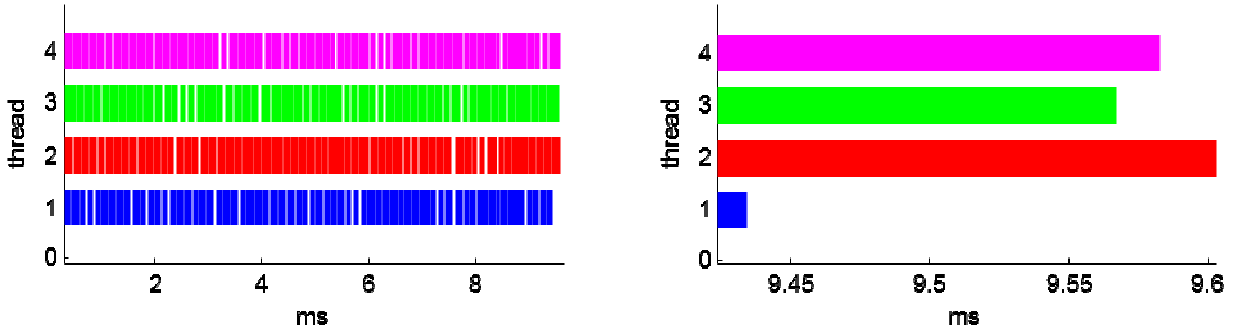

Figure 8. Real parallel thread execution of the parallel look-ahead function.

6. Conclusion and Future Work

Current CNC systems do not exploit the available processing power from multicore systems as their software design does not meet the requirements of these systems, namely a high grade of modularization that allows scheduling small pieces of the CNC application dynamically the available cores. This paper demonstrates that it is possible to benefit from the processing power of multicore systems by parallelizing CNC functions up to the algorithmic level.

At the example of the look-ahead function it was shown how dedicated CNC functions can be executed beneficially on such a multicore system platform. However, the presented look-ahead function is analyzed decoupled from the system where it has to be applied later on. The objective and future work is, next to parallelizing further functions, to embed this approach into the overall context of a CNC system. That allows analyzing additionally effects arising from CNC system characteristics and from the fact that multiple processes compete under real-time constraints for the available hardware resources.

Acknowledgements

The work presented in this paper was funded by the German Research Foundation (DFG) in the project “PANAMA”.

References

[1] H. Hong, D. Yu, X. Zhang and L. Chen: “Research on the Data Hungry Problem in CNC System Based on the Architecture of Real-time Multitask”, 3rd International Conference on Computer Research and Development, pp. 103-108, Shanghai, 2011.

[2] K. Erkorkmaz and Y. Altintas: “High speed CNC system design. Part I: jerk limited trajectory generation and quantic spline interpolation”, International Journal of Machine Tools & Manufacture, Vol. 41, pp. 1323-1345, 2001.

[3] M. Keinert and A. Verl: “System Platform Requirements for High Performance CNCs”, Proceedings of 22nd International Conference on Flexible Automation and Intelligent Manufacturing, Helsinki/Stockholm, 2012.

[4] P. McKenney: “When Do Real Time Systems Need Multiple CPUs?”, Proceedings of the 12th Real-Time Linux Workshop, Nairobi, Kenya, 2010.

[5] G. Pritschow: “Introduction to control engineering (Einführung in die Steuerungstechnik)”, Carl Hanser Verlag, Munich, 2006.

[6] S.-H. Suh, S.-K. Kang, D.-H. Chung and I. Stroud: “Theory and Design of CNC Systems“, Springer, 2008.

[7] Beckhoff White Paper: “Simplifying Multi-Core Migration in Automation Applications”, Intel Corporation, 2008.