Parallel Task Processing on a Multicore Platform in a PC-based Control System for Parallel Kinematics

Yannick Dadji[1], Jochen Maass[2], Harald Michalik[1]
[1]Institute of Computer and communication Network Engineering, Braunschweig, Germany
[2]Institute of Machine Tools and Production Technology, Braunschweig, Germany

{y.dadji-foyet, j.maass, michalik } @tu-bs.de

ABSTRACT

Multicore platforms are such that have one physical processor chip with multiple cores interconnected via a chip level bus. Because they deliver a greater computing power through concurrency, offer greater system density multicore platforms provide best qualifications to address the performance bottleneck encountered in PC-based control systems for parallel kinematic robots with heavy CPU-load. Heavy load control tasks are generated by new control approaches that include features like singularity prediction, structure control algorithms, vision data integration and similar tasks. In this paper we introduce the parallel task scheduling extension of a communication architecture specially tailored for the development of PC-based control of parallel kinematics. The Scheduling is specially designed for the processing on a multicore platform. It breaks down the serial task processing of the robot control cycle and extends it with parallel task processing paths in order to enhance the overall control performance.

Keywords: Multicore, parallel processing, PC-based control, parallel kinematics

1. INTRODUCTION

Due to their structure, parallel kinematics enable the achievement of high velocities and accelerations (up to 10 g) and provide structural stiffness and repeating accuracy. Generally, their structure consists of a closed kinematic chain with more than one link connected over joint elements to the manipulator on the one hand and to a stationary based platform on the other hand. Serial kinematics however consist of an opened chain of links and joints connecting the base platform to the manipulator. Compared to serial kinematics, all the active joints in a parallel kinematic structure are fixed to the stationary platform, so that the mass of the drives has no significant influence to the energy balance of the manipulator movements and thus enable high dynamic movements. Some examples of parallel kinematics are depicted in Figure 1. The major drawback of parallel kinematics is the relative limited workspace, so applications with high dynamic requirements in a restricted environment will be more appropriate for this environment.

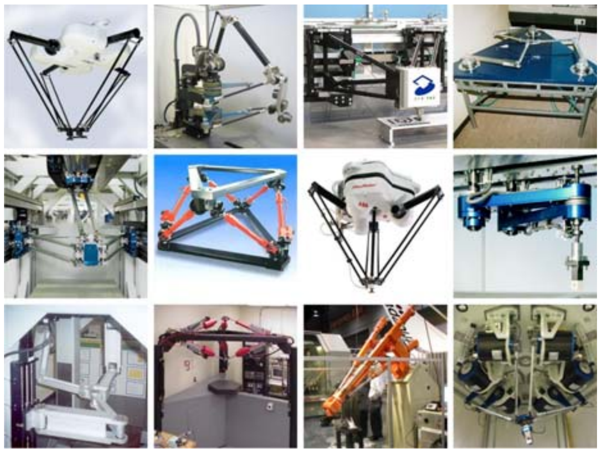


Figure 1: Structural variety and application diversity of parallel robots

The development of fundamental concepts to handle parallel kinematics is one of the focuses of the Collaborative Research Center 562 (SFB562) [1] at the technical university Braunschweig. In this context, we have designed a unified robot control system [2] [3] that easily adapts to the specific robot and application. The control system is built on the top of a communication architecture [3] [4] that supports the

development chain of the control application from the PC-software up to the drive controllers.
In the first section of this paper, we will give a brief description of the communication architecture. We will particularly consider the task scheduling component of the communication middleware MiRPA-X. The second section of the paper will describe the structure of the robot control software and discuss performance limitation related to the implementation on a single core PC. We will insist on the realized serial control cycle. In the third section we introduce the parallelization strategy of the task processing in the control cycle, specially tailored for the deployment on multicore platform.

2. ROBOT CONTROL SOFTWARE ARCHITECTURE

The design of the control system relies on a PC-based approach [2] [5]. The advantages are obvious: The choice of a PC as control hardware guarantees that the robot control system will always be based on the state-of-the-art computer architecture, i.e., the robot control system will keep pace with the development of the PC technology. The latest achievements in the multicore processor technology open the way to more robust and complex control applications by providing the necessary processing power.

Communication Architecture

Figure 2 depicts the PC-based communication architecture [3] [4] we deployed for the control of parallel kinematics. It consists of control software running on a single PC under the real-time operating system QNX. The control software is connected to the external robot communication nodes (sensor/actuator) via the IEEE1394 bus. In order to support developers in designing modular control software the application of a communication middleware (MiRPA-X, Figure 2) is a key feature. It supports the designer by straightforward implementation of communication issues within the system to concentrate on the algorithmic part of the software. To guarantee a fixed control cycle and assure a deterministic communication between the control software and the external robot node, the dedicated protocol IAP (Industrial Automation Protocol) is applied. The protocol is implemented on both the control PC and the external communication nodes of the robot.

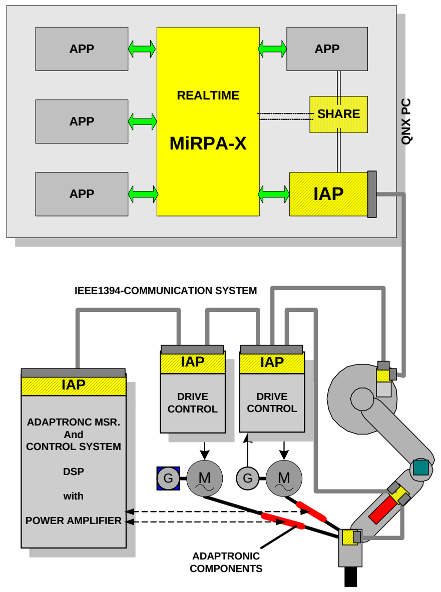
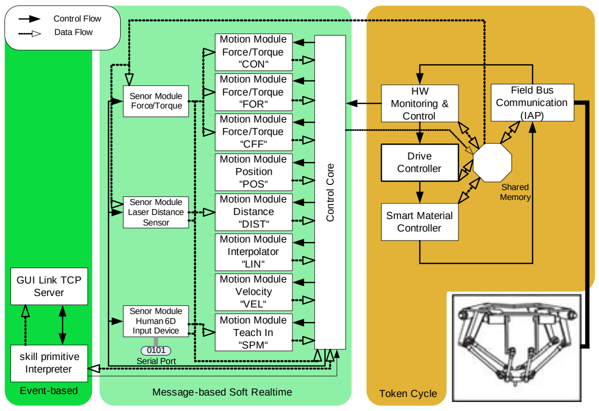


Figure 2: Software and communication architecture of the parallel robot control system.

Control Software

In this section, the robot control architecture and its design principles are described. The key technology to facilitate the development of a control framework is the usage of the flexible and highly efficient communication and synchronization middleware MiRPA-X [3] [4]. It uses QNX [6] internal message passing as the basic mechanism for putting synchronous and asynchronous communication services into reality. According to this, the control architecture can be ported to any realtime operating system that supports this kind of mechanism. Application processes providing control level services are regarded as servers, while service requesters are regarded as clients. According to these roles, servers block on the reception of specific queries and instructions. Alongside the message-based communication, MiRPA-X provides a communication mechanism based on shared memory usage. As for messages, the shared memory mechanism uses the MiRPA-X name service. The usage of shared memory facilitates a high-speed, non-blocking data transfer between application processes, without the object server being involved in the actual communication task. After registration, application processes directly read and write inter-process data using memory pointers provided by the middleware. If multiple application processes access the same shared memory region, data integrity has to be ensured at any time. As shared memory access is neither blocking nor synchronized by itself, MiRPA-X provides a token-passing scheduler. These features allow the control architecture to be designed in a runtime-evolvable layered structure, as described in the following subsection.



Control Flow
Data Flow
Motion Module Force/Torque "CON"
Motion Module Force/Torque "FOR"
Motion Module Force/Torque "CFF"
Motion Module Position "POS"
Motion Module Distance "DIST"
Motion Module Interpolator "LIN"
Motion Module Velocity "VEL"
Motion Module Teach In "SPM"
Sensor Module Force/Torque
Senor Module Laser Distance Sensor
Senor Module Human 6D Input Device
0101
Serial Port
Control Core
HW Monitoring & Control
Field Bus Communication (IAP)
Drive Controller
Shared Memory
Smart Material Controller
GUI Link TCP Server
skill primitive Interpreter
Event-based
Message-based Soft Realtime
Token Cycle

Figure 3: Information flow in the layered control architecture

Layered design

An overview of the control implementation is given in Figure 3. The architecture consists of three layers, the very right section representing the hard real-time layer of application processes. They operate in a token-passing context in a strictly serial and deterministic manner within a high-speed cycle frequency. Hard-real-time processes always run at high priority without becoming re-scheduled. This leads to extremely low jitter [7] and the subordinated drive controller benefits significantly thereof, compared to pure soft real-time approaches. The token cycle features a field bus communication process, which is the master synchronization source. It transmits the sensor data from the robot via the IAP [3] communication protocol to a shared memory area. After the process has released its token, it is received by the Hardware Monitoring and Control process. This process is responsible for activation and shutdown sequences of the robot next to monitoring and surveillance functionality. Moreover, it activates the control core in the adjacent layer according to an adjustable ratio of cycles. The token is passed subsequently to two processes that encapsulate robot specific controllers: one for control of the drives and one for smart structure control, which is applied to parallel kinematic machines to reduce vibrations induced by high-speed motion. Then the token returns to the Field Bus Communication process and the output values are transmitted to the actuators of the robot. The cycle described above starts each time the Field Bus Communication process is triggered by a hardware clock generated interrupt. In the remaining CPU time, the middle layer of the architecture is executed, which is described in the following. The middle layer is responsible for generating a Cartesian space trajectory with respect to the actual manipulation primitive that is executed. A Cartesian trajectory is required, since it provides abstraction from a particular robot by defining the motion of the end-effector in distinction from defining the motion of the drives. The layer is operated in a message-based soft-realtime environment. When a synchronization event from the Token Cycle layer occurs, the sensor module processes are notified by a multicast message in order to read data from the shared memory where the sensor data is stored. As the sensor modules encapsulate signal processing algorithms, such as filtering or coordinate transformations into the task frame, they pass the processed information to the motion module processes. The motion modules encapsulate the trajectory generation algorithms and are explained in detail in the next section. They are activated by the control core by a point-to-point message using the highly efficient name-service of the middleware only if required for the execution of the actual manipulation primitive. Detail information on control engineering aspects is provided in [7]. After a configurable number of token cycles has passed, the Hardware Monitoring and Control process synchronizes the control core again and the data from the motion modules is fusioned to a valid set of Cartesian trajectory data. This information is passed to the drive controller via the shared memory area located in the right layer by a mutex-synchronized mechanism. The upper layer features relaxed reaction-time requirements. As a result, the robot program interpreter and the GUI link server are located in an event-driven message-passing environment located on the left side of Figure 3.

3. PERFORMANCE ENHANCEMENT THROUGH MULTICORE PLATFORM

Although the control software has been successfully deployed for the control of parallel kinematics, the current implementation based on a single core platform comes to its performance limits, when new and complex control paradigms are realized. Indeed, state-of-the-art and future control algorithms for parallel kinematics are quite demanding in terms of computational efforts. Representative examples are a) singularity prediction [8] in order to safely operate parallel kinematics machines, b) incorporation of vision data [9] in general and c) the latest approaches using force-torque maps for the execution of assembly sequences [10]. Multicore systems offer a suitable platform to address this performance bottleneck. Since the software architecture obeys a modular design and both motion and sensor modules are autarkic components that can be run concurrently, they are offhand distributed on a multicore platform for parallel processing. Although the performance is gained through parallel processing, one important design factor of the control limits the achievable performance on a multicore architecture: the token scheduler that activates the cyclic processing of the software components involved in the token cycle (Figure 3). In order to synchronize the inter-process communication over shared memory regions, the token scheduler was originally designed to automatically realize a serial scheduling, i.e. to schedule the processes successively and according to an adjustable order. The serial scheduling represents a performance limitation on a multicore platform, because it does not support parallel processing. In the next section, we present the extension of the scheduler for the execution that supports parallel task execution.

Extension of the Token scheduler

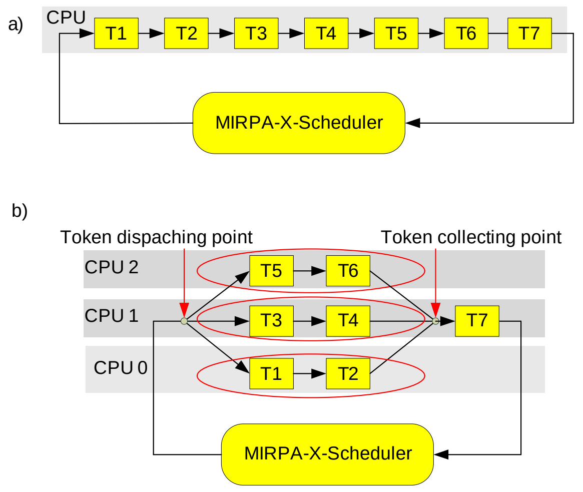


Figure 4: Extension of the token scheduler for parallel task execution in a token cycle

To get additional performance from the multicore platform in the MiRPA-X environment, we extended the functionality of the MiRPA-X scheduler from a sequential (serial) to a parallel scheduling. In Figure 4a an abstract view of the sequential scheduling on the single core processor is depicted. The tasks are successively scheduled and no additional synchronisation mechanism is necessary. At the beginning, all the tasks are in a wait-for-token state, waiting to receive a token and to start processing. To schedule a task, the scheduler sends a token (based on blocking message passing) and waits in a blocked state for reply. This way, two tasks cannot be scheduled in parallel and the shared memory data exchange between two tasks is secure.

On multicore systems, the idea is to classify tasks with no shared memory data dependency in modules which will be scheduled on different CPUs (Figure 4b, encircled tasks). In this context, the scheduler activates the different modules by dispatching multiple tokens, one token for each module, and thus executing the parallel processing of the modules. The token passing may not be based on a blocking mechanism; otherwise the token dispatching feature will not succeed. For this reason it has been implemented with a non blocking pulse message. To preserve synchronisation, a token collecting point is provided. Subsequent tasks (i.e. Figure 4b, task T7) can only be scheduled after all parallel processing modules complete. The parallel scheduling of the modules running on different CPUs reduces the processing time in the token cycle und thus improve the performance of the control system. The processing time of the token cycle can theoretically be reduced up to n times (n number of CPU on the system).

In Figure 5 a trace of the system events on a dual core platform is depicted. The trace recording is a feature of the QNX instrumented kernel. The trace shows the activation and the processing of two tasks running on different CPUs. In the figure, the token scheduler is displayed as the third thread (Thread 3) of the object server process (observ). The scheduler first activates the IAP module. After completion of the IAP, It activates the Modules Hardware_Control and Drive_Control simultaneously. The latter are beforehand configured to run each on a different CPU. The scheduler latency for the activation of the parallel processing is about 6µs.The extended functionality of the MIRPA-X scheduler is operational on systems with up to 32 cores under the QNX Neutrino operating system. On a dual core system a performance enhancement of 45% was measured for the token cycle.

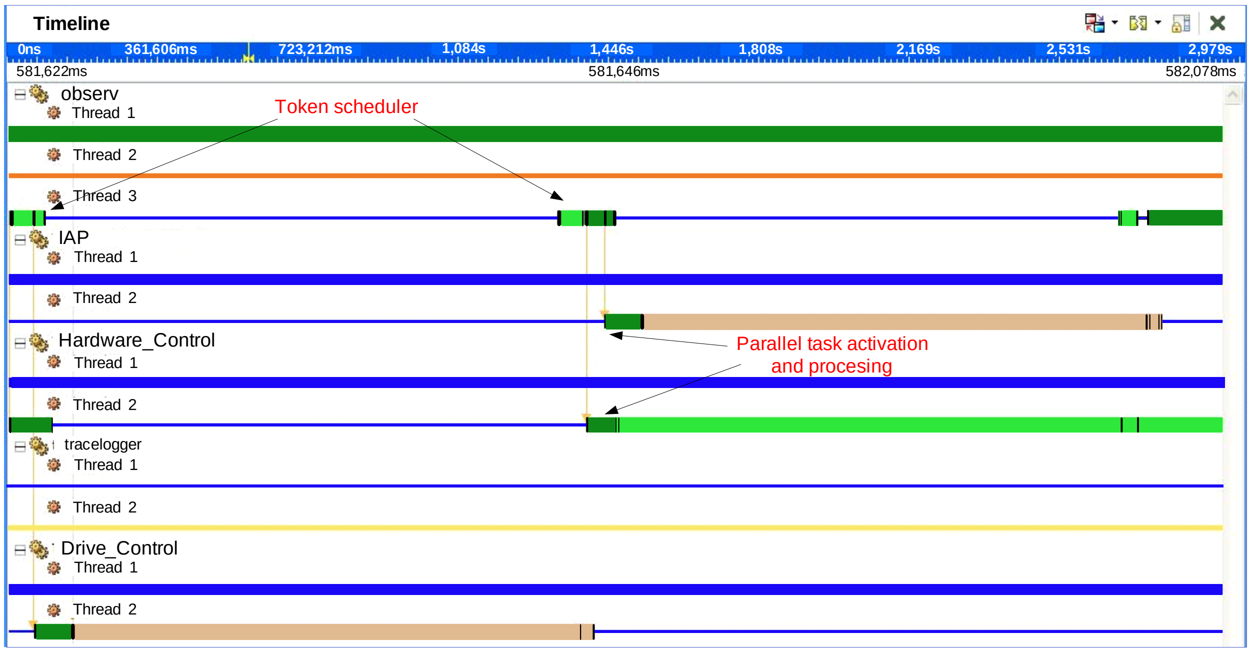


Figure 5: trace event of the parallel scheduling of two tasks on a multicore platform

Configuration and design restriction

To simplify the use of this feature, we introduce a central configuration space which parameters are depicted in Table 1. It enables a static mapping of tasks on the available CPUs, the configuration of the modules that will be processed in parallel and the task processing order. The column "token name" defines the token cycle. It contains a symbolic reference to the tasks which will be executed within the token cycle. According to the token cycle depicted in Figure 3, the symbolic references IAP, HW_Ctrl, DRIVE_Ctrl, and SM_Ctrl are respectively set for tasks "Field Bus Communication", "HW Monitoring & Control", "Drive Controller" and "Smart Material Controller". The order of the settings also defines the task execution sequence. The token cycle is limited by the special key words _START and _REPEAT. To specify which tasks should be scheduled in parallel we defined a parallelization index that is associated to every task. Tasks that should not be parallelized get the index 0. All successive tasks with an associated index 1 will be processed in parallel. In this case, the tasks should be mapped on different CPU cores for parallel processing. This CPU mapping is set in the column "Cpu mask". The settings in Table 1 consider a platform with at least 2 CPU cores and correspond to a parallel execution of the tasks "HW_Ctrl" and "DRIVE_Ctrl" on the CPU-Cores 1 and 2, respectively.

Token name	Parallelization index	Cpu mask
_START		
IAP	0	1
HW _Ctrl	1	1
DRIVE_Ctrl	1	2
SM_Ctrl	0	2
_REPEAT		

Table 1: static configuration of the parallel task scheduling

The only design restriction of the parallel scheduling extension is that the parallel processed modules are not allowed to have any shared memory data dependency, since data inconsistency could occur. But this restriction is not relevant, compared to the overall performance gained.

4. CONCLUSION

In this paper we presented a PC-based control system for parallel kinematics. We described the layered robot control software based on the top of a real time communication architecture. To solve the performance bottleneck occurring on single core platform we introduced the multicore platform approach. Then, we discussed the design related performance limitation, due to the serial task scheduling in the token cycle. After this, we introduced the parallel task scheduling approach specially tailored for multicore platforms. It consists in classifying tasks with no shared memory data dependency in modules which will be scheduled on different CPUs. In this context, the token scheduler uses a non blocking communication mechanism to activate the different modules by dispatching multiple tokens and such enabling a parallel task processing. For synchronization purposes, additional dispatching and collecting points are introduced. Finally, we gave some performance data and introduced some software design restrictions. Up to 45% of performance enhancement on a dual core platform could be achieved for a token cycle. The parallel scheduling and the additional performance gained open the door to the realization of more complex and robust control approaches.

5. ACKNOWLEDGMENT

The authors highly appreciate the support given by the German Research Foundation (DFG) within SFB562 and by QNX Software Systems providing free software licenses.

6. REFERENCES

[1] P. Last, C. Budde, F. M. Wahl, **proceedings of the second international colloquium of the collaborative research center 562**, Braunschweig, Germany, may 2005.

[2] J. Maaß, N. Kohn and J. Hesselbach, "Open modular robot control architecture for assembly using the task frame formalism", **International Journal of Advanced Robotic Systems**, vol 3-1, pp. 001-010, 2006.

[3] N. Kohn, J.-U. Varchmin, J. Steiner, U. Golz, "Universal communication architecture for high-dynamic robot systems using QNX", **8th International Conference on Control, Automation Robotics and Vision**, Kunming, China, pp. 205- 210, 2004.

[4] Y. Dadji, H. Michalik, T. Moeglich, J. Steiner, " Performance optimized Communication system for high-dynamic and real-time Robot Control Systems, **CD-ROM proceedings of the 16th. International Workshop on Robotic** in Alpe-Adria-Danube Region, 7-9 June 2007, Ljubljana, Slovenia.

[5] G. Pritschow, T.L Tran, "Parallel kinematics and PC-based control system for machine and tools", **proceedings of the 37th IEEE Conference on Control,** Tampa, Florida USA, pp. 2605-2610, 1998.

[6] The QNX operating system, http://www.qnx.com, April 2008

[7] J. Maaß, J. Hesselbach, N. Kohn, "Open modular robot control architecture for assembly using the task frame formalism", **International Journal of advanced robotic systems,** 3, pp. 001-010

[8] J. Hesselbach, J. Maaß and C. Bier, "Singularity prediction for parallel robots for improvement of sensor integrated assembly", **Annals of the CIRP,** pp. 349-352, 2005

[9] S. Hutchinson, G. Hager, P. Corke, "A tutorial on visual servo control", **Transactions on Robotics and Automation,** vol. 12, pp. 651-670, 1996

[10] S. R. Chhatpar and M. S. Branicky, "Localization for Robotic Assemblies Using Probing and Particle Filtering", **IEEE/ASME International Conference on Advanced Intelligent Mechatronics,** pp. 1379-1384, 2005