More or Less? A Discussion of the Abstraction Level of Future Operating Systems

Hendrik Borghorst
hendrik.borghorst@udo.edu
Department of Computer Science, TU Dortmund
Dortmund, Germany

Michael Müller
michael.mueller@uos.de
Institute of Computer Science, Osnabrück University
Osnabrück, Germany

Olaf Spinczyk
olaf.spinczyk@uos.de
Institute of Computer Science, Osnabrück University
Osnabrück, Germany

ABSTRACT

The demand on today's computing systems is rapidly increasing, for example, with cyber-physical systems capturing real-time images of the real world. The computational power necessary to fulfill such tasks is extremely high and therefore new ways of thinking are necessary to tackle the challenges that come with ever increasing hardware and software complexity for mission critical systems. Many-core processors will introduce memory systems with a much higher complexity than current systems. Data structures are increasing in size but need more fine-granular locking mechanisms to fulfill the computational tasks in parallel. Heterogeneous systems are built to solve complex tasks like artificial intelligence within more traditional applications and are therefore calling for an tight integration within the system. This papers will give some ideas on what challenges lay ahead to provide better suited operating systems for future generations of cyber-physical systems.

CCS CONCEPTS

• **Computer systems organization → Embedded software**; *Real-time operating systems*; Dependable and fault-tolerant systems and networks; • **Software and its engineering → Operating systems**.

KEYWORDS

operating systems, cyber-physical systems, real-time, many-core

1 INTRODUCTION

The necessity for powerful computers in cyber-physical systems is constantly increasing. With more complex systems being in the design phase right now it is more important than ever to have an operating system that is able to aid system designers with the task of handling complex computing architectures, such as heterogeneous systems with special accelerator units or systems that feature a very complex memory architecture, e.g. modern NUMA many-core processors. Currently popular operating systems tend to keep the interfaces between the operating system and applications running on top of it as simple as possible. Consequentially the operating system usually is not aware of a variety of non-functional aspects of an application, like memory-constraints, timing guarantees or the possibility to run code fragments on special dedicated hardware.

With this paper we will give an opinion on matters that we think are important for modern operating systems for cyber-physical systems. We will describe the details of some challenges and give some inspirations on how to address these challenges.

NGOSCPS 2019, April 15, 2019, Montreal Canada

2 FUTURE CHALLENGES AND REQUIREMENTS

Future cyber-physical systems most likely will use some kind of modern System on Chip architectures that feature highly complex hardware designs, exhibiting problems due to a lot of shared resources. One shortcoming of many current operating system interfaces is, that applications request resources only when they need them on demand. This requires, that optimizations the OS is trying to apply, always need to be based on heuristics that predict application's resource usage. It would be better if the OS is able to plan resource usage ahead based on application's information.

2.1 Timing Requirements

Almost all cyber-physical systems are used for tasks that have hard or soft real-time constraints. It is inevitable that the operating system is able to guarantee that tasks are executed within these timing constraints. The scheduling algorithm with the highest efficiency at runtime is one that is calculated offline. Therefore the operating system should provide a interface for tasks that have static timing constraints, thus enabling the OS to plan most of the schedule ahead. Non-static task-sets could still be integrated with modern online scheduling algorithms.

Current popular programming languages lack the support for the definition of timing constraints and therefore need to be either extended or replaced by more sophisticated ones with abilities to provide more metadata about requirements for code-snippets. Research was done a long time ago on integration of real-time constraints within programming languages. RTC++ is an object-oriented programming languages which allows to define timing-constraints for programms [6].

2.2 Memory System

Almost every processor that features more than one processor core has a memory subsystem with some parts of it shared between processor cores. Future operating systems need to be aware of the whole memory hierarchy. This means that the system needs to have detailed knowledge about how memory is connected to which processor core, which parts of the cache are shared and even if there is a non-uniform cache architecture (NUCA) present. It would be optimal to have some knowledge about cache interconnects, e.g. Infinity Fabric on AMD systems, as well. The most comfortable solution for application developers and system designers, would be

to hide the complete level of detail. However, the past has shown that this approach does not work very well and results in subpar performance under memory intense workloads [4]. For NUMA processors the memory access time is highly dependent on the memory region that is accessed, because an access can occur over shared buses and memory controllers. An alternative would be an interface that allows for applications to specify how much memory it requires in advance. This in combination on how much parallel hardware-based threads it will need would enable the OS to optimize the placement of both the application to the hardware threads and to the memory regions as the demand is bounded and predictable.

Shared caches and caches in general are also completely hidden by the hardware for the OS, making it hard and cumbersome for application developers to optimize their code for it. Usually an application needs optimizations that only work for one specific hardware architecture. Our research system **CyPhOS** [2][1] tries to improve on that with a component-based structure. Each component is well-suited for placement in a region of the shared cache [2]. The components are preloaded and then locked in the shared cache, eliminating the problem of unwanted cache eviction. We are also working on a memory allocator, that allows for applications to specify the caching requirements for each memory requests. The system provides three classes of memory allocations. The first one allows uncached requests, which means that all the memory allocated with this class is never polluting the cache and each access is directly on the main memory. The second class is always cached, that means that the whole memory block is preloaded to the cache on software component activation and stays locked within the cache until deactivation of the component. The last class is on-demand caching with memory regions, that are getting preloaded (as a whole) to the cache on first access (either read or write).

2.3 Heterogeneous Hardware

In the foreseeable future new cyber-physical systems are increasingly interacting with the physical world. To do this in a safe manner modern image recognition systems will become more and more ubiquitous. The demand for the integration of high performance GPUs for GPGPU tasks will ramp up and therefore operating systems need standardized interfaces for the development of applications using them. For many years researchers and developers have been struggling to find a good way to interact with GPUs within normal applications. So far OpenCL [15] prevailed as an open industry standard, attracting many developers. However until today only specialized tasks with special programming languages are executed on GPUs or other accelerator units. The management of GPU resources is mostly done by user-space libraries with no control by the OS. New operating systems should be able to transparently enable developers to use accelerator units within their applications, without specifically selecting code snippets to be executed on external devices. Integration of support for GPUs within mainstream compilers like LLVM[2] and GCC[3] has been done to support integration of GPU code within normal C/C++ applications. PTasks is an approach to try to integrate GPU workloads within more traditional tasks [14]. Further research on the integration of FPGAs in existing operating systems was done with the ReconOS research system, that integrates FPGA-based hardware threads within traditional software operating system threads [8].

2.4 Programming Models

Future many-core processors need applications that spawn over a large number of hardware threads. The current threading model of current operating systems might not be a good fit for processors with more than 100 cores because thread creation is comparably slow and resource-intensive. A better solution would be lightweight tasks that can be created in a very short time frame and are light in resource usage [13]. Research was done in optimizing DBMS workloads with tasks running on worker threads to improve the efficiency of a system [12]. Preemption would not be useful for such tasks as a preemption would bring no benefit and would only increase the complexity. A hybrid approach with lightweight tasks and normal threads would allow flexible systems do be designed, that can run long running preemptible tasks in conjunction with lightweight tasks. A model like this would also enable tasks to be run optionally on heterogeneous accelerator devices if it would provide a benefit. The operating system could provide a binary form of the code for all computing devices a system includes. This would require a metadata description language to precisely describe the interface of a task. For example what input and output data a task requires. Real-time systems would require an additional description of timing requirements. This extra information could be used by an operating system to decrease resource usage and therefore increasing the overall system efficiency by selecting the computing device best fit for a task [14].

2.5 Data Synchronization

With more complex tasks the necessary data structures are getting harder to synchronize as more hardware threads are competing for data access. A more fine-granular synchronization is inevitable for many-core systems, otherwise no performance gain can be reached [3]. A possible solution could be to divide data structures in smaller parts and bind hardware threads to parts of the data structure. With this approach only one processor can access certain parts of data structures and all operations are running on the same processor thus serializing all accesses. This can be combined with a lightweight task model [13]. This would ensure that data access is fast enough and not bottlenecked by a slow thread creation. A major advantage of this approach would be that data would stay local, meaning that a lot of the data can stay inside the cache and therefore also reducing the contention of shared buses.

2.6 Portability

Modern hardware architectures are fast developing meaning that it can be challenging for operating systems designers to keep up with ever changing hardware architectures. New ways to adopt existing operating systems to new systems should be examined. Model-driven software development can be used to generate operating system code from abstract code in combination with detailed

[1] https://github.com/ESS-Group/CyPhOS
[2] https://llvm.org/docs/AMDGPUUsage.html
[3] https://repo.or.cz/official-gcc.git/commit/f8ada740d4349df85fadcbd1354ee095ae4fe18a

hardware models [1]. A metadata model to specify hardware architecture details to applications running on an OS needs to be developed. They have to be able to specific complex system-on-chip architectures as well as modern many-core server architectures with multiple NUMA nodes and complex cache hierarchies.

The Barrelfish operating system project takes modeling of hardware even further with the integration of its own DSL (domain specific language) to describe hardware interfaces for driver development. The DSL is especially designed to model memory-mapped registers, hardware data structures and everything related to driver development [11].

2.7 Reliability

With future cyber-physical systems in mind that interact with an increasing count of humans it is imperative that these systems are reliable. Future operating systems need to provide a base for reliable systems in a way that is not too complex for system designers to use. Possible solutions for high availability systems are virtualization-based approaches like Remus [5]. To guarantee timing requirements and still provide a fail-safe solution, a real-time extension of Remus called CPS-Remus can be used [7]. The system allows virtualized operating systems to be run on multiple physical hosts to guarantee that the virtual machine continues running even if one (or multiple) machine fails.

2.8 Energy Management

Energy awareness needs to be tightly integrated within future many-core operating systems, as it is very likely that a majority of cyber-physical systems are mobile. With more hardware subsystems available on nearly every hardware platform the operating system needs to be able to control each hardware subsystem with a fine granularity. Applications should be able to specify which interfaces to which device they will be using during execution. This would make energy-aware scheduling for the OS much easier than it is at the moment where everything is only scheduled online. Furthermore the operating system needs detailed information about the hardware platform it is running on. For example a many-core processor might require that the operating system tries to minimize hotspots on the chip itself [9] and therefore tasks should be scheduled far away from each other. On the other hand this might also cause problems for data movement as more bus traffic might be necessary. As with other requirements it could be beneficial if applications can provide more information about their computational behavior to the operating system. One example is DVFS [10]. With more information the OS could better decide if a boost to a higher frequency state is worthwhile or if it is better to stay in its current power state.

3 OUTLOOK

In the past operating systems presented applications the illusion of infinite resources like memory or computing power. However this approach only shifts the real problem of resource juggling to the operating system that has to try to hide the real world details of limited resources as best as possible with methods like swapping for memory shortage. After all, we think it would be better if operating systems provide a detailed overview of both hardware architectural details and the available resources. This could enable applications to better coordinate resource usage with the OS and better overall system efficiency. A simple operating system interface is easy to maintain for developers of operating systems but is not optimal for good resource usage. Therefore a more complex interface should be available for developers to optimize future resource usage. A model for task metadata should be designed to describe all requirements an application has. Interoperability and portability requirements call for open standards of at least a common base of interfaces.

ACKNOWLEDGMENTS

This work is supported by the German Research Foundation as part of the priority program 2037 "Scalable Data Management for Future Hardware" under Grant No.: SP 968/9-1.

REFERENCES

[1] Hendrik Borghorst, Karen Bieling, and Olaf Spinczyk. 2016. Towards versatile Models for Contemporary Hardware Platforms. In *Proceedings of the 12th Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT '16)*. RheinMain University of Applied Sciences, 7–9.

[2] Hendrik Borghorst and Olaf Spinczyk. 2019. CyPhOS – A Component-based Cache-Aware Multi-Core Operating System. In *Proceedings of the 32th International Conference on Architecture of Computing Systems (ARCS '19)*. to appear.

[3] Silas Boyd-Wickizer, M Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. 2012. Non-scalable locks are dangerous. In *Proceedings of the Linux Symposium*. 119–130.

[4] P. Caheny, M. Casas, M. Moretó, H. Gloaguen, M. Saintes, E. Ayguadé, J. Labarta, and M. Valero. 2016. Reducing cache coherence traffic with hierarchical directory cache and NUMA-aware runtime scheduling. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*. 275–286. https://doi.org/10.1145/2967938.2967962

[5] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. 2008. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. San Francisco, 161–174.

[6] Y. Ishikawa, H. Tokuda, and C. W. Mercer. 1992. An object-oriented real-time programming language. *Computer* 25, 10 (Oct. 1992), 66–73. https://doi.org/10.1109/2.161281

[7] B. Jablkowski, M. Mueller, and O. Spinczyk. 2018. High Availability in Cyber-physical Systems by Self-determined Virtual Machine Replication. In *Proceedings of the 13th IEEE International Symposium on Industrial Embedded Systems (SIES 2018)*.

[8] E. Lubbers and M. Platzner. 2007. ReconOS: An RTOS Supporting Hard-and Software Threads. In *Proc. Int. Conf. Field Programmable Logic and Applications*. 441–446. https://doi.org/10.1109/FPL.2007.4380686

[9] Andreas Merkel and Frank Bellosa. 2006. Balancing Power Consumption in Multiprocessor Systems. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys '06)*. ACM, New York, NY, USA, 403–414. https://doi.org/10.1145/1217935.1217974

[10] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Hierarchical Power Management For Asymmetric Multi-core In Dark Silicon Era. In *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*. ACM, New York, NY, USA, Article 174, 9 pages. https://doi.org/10.1145/2463209.2488949

[11] Barrelfish project. 2013. Mackerel User Guide - Barrelfish Technical Note 2. http://www.barrelfish.org/publications/TN-002-Mackerel.pdf

[12] Iraklis Psaroudakis, Tobias Scheuer, Norman May, and Anastasia Ailamaki. 2013. Task Scheduling for Highly Concurrent Analytical and Transactional Main-Memory Workloads.. In *Proceedings of the 4th International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*. 36–45.

[13] James Reinders. 2007. *Intel Threading Building Blocks* (first ed.). O'Reilly & Associates, Inc., Sebastopol, CA, USA.

[14] Christopher J. Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. 2011. PTask: Operating System Abstractions to Manage GPUs As Compute Devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 233–248. https://doi.org/10.1145/2043556.2043579

[15] John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 3 (2010), 66–73.