Scheduling and Synchronization on Multicores

Michal Riša
1st year, full-time study
Supervisor: Josef Strnadel

Brno University of Technology, Faculty of Information Technology
Božetěchova 2, 612 66 Brno, Czech Republic
irisa@fit.vutbr.cz

***Abstract*—The article presents a motivation, basic terms, principles and problems related to applications created on basis of multicore platforms. Then, state of the art is outlined in brief, followed by a summary of work being done in preceding MSc thesis. Afterwards, ideas behind prepared PhD thesis are outlined such as research directions, goals and roadmap, followed by details to research questions and hypothesis. At the end, methods and instruments to reach the goals are summarized and the paper is concluded.**

***Keywords*—scheduling, synchronization, multicore**

I. Introduction

For the past 50 years, Moore's law accurately predicted that the number of transistors on an integrated circuit would double every two years. To translate these transistors into equivalent levels of system performance, chip designers increased [1]:

- Clock frequencies (requiring deeper instruction pipelines),
- instruction level parallelism (requiring concurrent threads and branch prediction),
- memory performance (requiring larger caches),
- and power consumption (requiring active power management).

Each of these four areas is hitting a wall that impedes further growth [1]:

- Increased processing frequency is slowing due to diminishing improvements in clock rates and poor wire scaling as semiconductor devices shrink,
- instruction-level parallelism is limited by the inherent lack of parallelism in the applications,
- memory performance is limited by the increasing gap between processor and memory speeds,
- power consumption scales with clock frequency; so, at some point, extraordinary means are needed to cool the device.

A. Motivation

Using multiple processor cores on a single chip allows designers to meet performance goals without using the maximum operating frequency. They can select a frequency in the sweet spot of a process technology that results in lower power consumption. Overall performance is achieved with cores having simplified pipeline architectures relative to an equivalent single core solution. Multiple instances of the core in the device result in dramatic increases in the MIPS-per-watt performance [1].

The introduction of multicore processors provides a new challenge for software developers, who must now master the programming techniques necessary to fully exploit multicore processing potential. On a single-core processor, separate *tasks* share the same processor (i.e., its time). On a multicore processor, multiple tasks can run in paralel (i.e. they can be executed at the same time by the corresponding cores), resulting in more efficient execution.

B. Basic Terms, Principles and Problems

One of the first steps in mapping an application to a multicore processor is to identify the task parallelism and select a processing model that fits best.

1) Parallel Processing Models: According to [1], two dominant models exist:

- *Master/Slave*, i.e., centralized control with distributed execution. A master core is responsible for scheduling various executions that can be allocated to any available (slave-)core for processing. It also must deliver any data to a slave. Applications (e.g., multi-user data link layer of a communication protocol stack) that fit this model i) inherently consist of many small independent tasks that fit easily within the processing resources of a single core and ii) often run on a high-level OS like Linux (i.e., the master in charge of the scheduling) and potentially already have multiple execution to be done. Typically, task assignment is achieved by message-passing between the (master and slave) cores – the messages provide the control triggers to begin a task execution and pointers to the task's data.

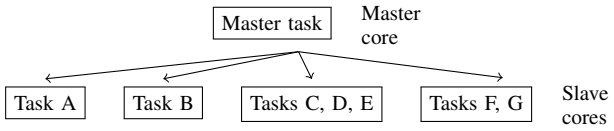

Figure 1. Master/Slave parallel processing model.

- *Data Flow*, i.e., distributed control and execution. Each core processes a block of data using various algorithms; then, the data can be passed to another core for further processing. The initial core is often connected to an input interface supplying the initial data for processing. Scheduling is triggered upon data availability. Applications (such as the physical layer of a communication

protocol stack) that fit the model often contain large and computationally complex components that are dependent on each other and may not fit on a single core. They likely run on a real-time (RT) OS where minimizing latency is critical. The challenge for applications using this model is partitioning the complex components across cores and the high data flow rate through the system. Components often need to be split and mapped to multiple cores to keep the processing pipeline flowing regularly. The high data rate requires good memory bandwidth between cores. The data movement between cores is regular and low latency hand-offs are critical. Synchronization of execution is achieved using message passing between cores. Data is passed between cores using shared memory or DMA transfers.

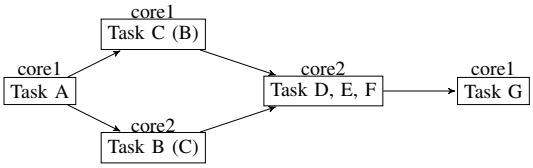

Figure 2. Data Flow parallel processing model.

2) Software Decomposition for Multicores: To design a software to be executed on a multicore platform, the following problems must be solved (typically, the solution must be searched in several cycles to find a (sub)optimal result):

- *Partitioning*, i.e., to identify a large number of small tasks (that are paralelizable, i.e. with low coupling and high cohesion) in order to yield a fine-grained decomposition; in the ideal situation, the tasks belonging to different partitions are intended to execute concurrently,
- *Dependency*, i.e., to identify (partitioned) tasks that are not independent; those must be mutually synchronized (serialized) to guarantee the dependencies, so their concurency is limited. Metrics are typically utilized to evaluate the level of cohesion in order to assist the grouping of tasks for minimizing dependency effects,
- *Combination*, i.e., to reflect information about partitions/dependencies for deciding about grouping of tasks so that they can be efficiently executed on a multicore,
- *Mapping*, i.e., to assign (potentially grouped) tasks to particular cores on basis of the selected parallel processing model (see I-B1 on p. 1). After all the tasks are mapped, the overall loading of each core can be evaluated to indicate areas for additional refactoring to balance the processing load across cores. Alike, further parameters such as message passing latency or worst-case blocking time due to the most pessimistic intra&inter-core synchronization/communication scenario(s) can be evaluated as well.

II. State-of-the-Art

A. Representatives of Existing Works

On basis of problems being solved, existing works can be divided into several groups. Basic analysis/summary of effects w.r.t. synchronization/communication on multicores (such as performance, speedup and scaling from Amdahl's law perspective) can be found in [2].

On top of that, particular works can be found, each dealing with a special area of interest w.r.t. multicores such as [3], dealing with the load balancing, or [4], trying to contribute to the synchronization problem by proposing a hardware lock implementation (so-called the lock arbiter herein) designed to reduce the lock latency while minimizing hardware overheads and maintaining high levels of fairness. In software, mechanisms such as resource access protocols [5], [6] are designed to prevent undesired effects such as priority/deadlock inversion, chained blocking and/or deadlock. Traditionally, the problem of clock synchronization must be solved too [7].

Many works deal with power management and issues [8] and solutions to the scheduling problem for multicores, being utilized to construct applications being critical somehow such as time-critical (or, real-time) applications. It should be noted there that the problem of scheduling (real-time tasks on a multicore processor) is the same as that of scheduling on a multi- processor system, i.e. an NP-hard problem – its solution can be approximated by heuristics. Basically, these heuristics can be divided into two categories, so-called *partitioning schemas* (*policies*) too [5]:

- *partitioned* being constructed to assign tasks to cores so that a task is going to be executed just by the core being assigned to the task,
- *global* allowing a task to be executed by different cores, depending on actual parameters of a system such as the computational load.

After the partitioning of all tasks is completed, tasks in each core can be scheduled using well-known mechanisms [6] or their multicore variants [5]. Due to their simplicity and efficiency, partitioned scheduling algorithms are generally preferred over global scheduling algorithms.

B. Work Done in MSc Thesis

The MSc Thesis [9] is focused on *asymmetric multiprocessing* (*AMP*) on the ARM Cortex–A9 MPcore platform. The *AMP* is an approach to computer system load distribution among heterogeneous software or hardware environments. In the thesis, there are heterogeneous software environments (see the figure below). Two equal processor cores run Linux, but when needed, the second processor core (slave) is run–time given to a bare–metal application by the first processor core (master).

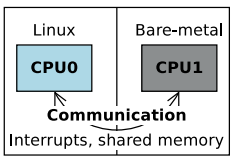

Figure 3. MSc Thesis *AMP* system.

The *AMP* was required to run on an Altera Cyclone V platform. It was discovered that an OpenAMP framework [10] that provides required functionality exists for Xilinx

Zynq platform that also contains the ARM Cortex–A9 MPcore processor. Thus the porting process from Xilinx Zynq to Altera Cyclone V had begun:

- The OpenAMP was running on PetaLinux on the Zynq and was ported to xilinx–2014.4 Linux.
- Later, the OpenAMP was ported to Altera Cyclone V hardware platform. Changes were made to OpenAMP's bare–metal libraries, system memory map was reorganized and missing loadable kernel modules for Altera's linux–socfpga were derived from Xilinx's xilinx–2014.4 Linux.

OpenAMP is now functional on the Altera Cyclone V platform although some work needs to be done in stabilizing the port.

III. PhD Thesis Details

Although many works have already dealt with the topics mentioned in II-A on p. 2, there are still many problems to solve e.g. in the area of studying inter-relationships among various parts of a multicore system. Particularly, it is necessary to study impacts of undesired events (such as a fault, error or failure, performance variations, overheating etc. of a component in the system) to parameters and behavior of the system.

For that purpose, appropriate instruments and techniques must be utilized (for more information, see III-D on p. 4, please). Those must be capable to i) describe behavior as well as attributes of a system and its components, their dynamics and ii) analyze properties of the system. Although it is planned to focus mainly to digital (discrete) systems, it would be advantageous if analog systems would be covered by the instruments too; this gives one an opportunity to model and analyze e.g. mixed/hybrid (i.e., digital/analog) systems – such as *cyber-physical* or *mixed-signal* systems – being widely utilized in practice.

A. Research Directions

1) Directions: There are many directions for the research w.r.t. multicores; we have limited their list to – modeling, analysis and/or design of – the following ones we plan to focus on:

- Cores, interconnections and topology,
- clock, memory and I/O subsystems,
- power and dependability issues,
- synchronization/communication primitives and mechanisms,
- policies for partitioning tasks, scheduling tasks and communications.

B. Research Questions and Hypothesis

On basis of our previous activity, many problems have been identified w.r.t. the multicore area. Since we intend to address some of the problems in our research, we have prepared a (preliminary, work in progress) list of *research questions* we would like to answer:

- Is it possible to build a credible and valid model of a generalized multicore platform ?
- Can the (above-mentioned) model be utilized to facilitate analysis, design and/or portability of platform-dependent routines for construction of a preemptive OS ?
- Can the (above-mentioned) model be utilized to facilitate analysis, design and/or portability of (power-, time-, safety- etc.) constrained, OS-controlled applications ?

On basis of the questions, our *research hypothesis* can be formulated as follows:

> **It is possible to build a credible and valid model of a generalized multicore platform on the basis of which analysis, design and/or portability of a software for multicores can be facilitated.**

C. Research Goals and Roadmap

1) Goals: Their list includes, but is not limited to

- scalable multicore solutions of problems related to task/ISR-level context switching, mutually exclusive access and task/kernel-level synchronization, task scheduling with potential core affinity/migration (i.e., with partitioned/global policy support),
- techniques to achieve robustness of an operating system and consistency of its data structures such as a task control block (TCB) and its adaptation to changes in a multicore platform,
- scheduling policies able to meet multiple task constraints (posed on time, power, safety etc.) under various fault/load scenarios,
- selection of comparison basis (such as benchmarks) and of proper methods and instruments to verify crucial properties (such as thread-safe or deadlock-free operation, worst-case latencies of system calls etc.) of the proposed concepts under AMP and/or SMP scenarios as well as to show their practical applicability using several case-studies of recent platforms and operating systems.

2) Roadmap: Basic blocks of the expected roadmap is depicted below. Estimation of the overall time (in months) reserved for a particular block is indicated in the circle by the block. Let it be noted there that activities w.r.t. particular blocks can overlap and be performed in parallel. The green blocks are almost completed, red ones not started yet and white ones have just started. Recent activity is highlighted using bold borders/arrows.

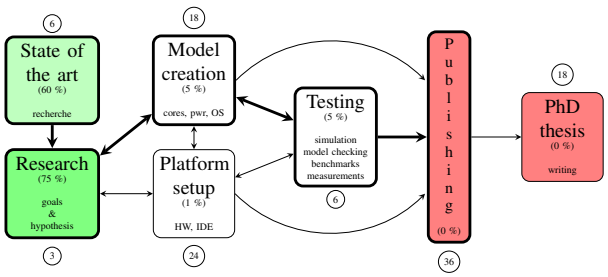

Figure 4. Research roadmap.

D. Methods and Instruments to Reach the Goals

To reach the goals outlined in III-C1, it is necessary to choose appropriate methods and instruments.

1) Modeling and Analysis Phase: Actually, we focus to the model creation and analysis phase, for which we have decided to apply the *stochastic timed automata* approach combined with *statistical model checking* (SMC) technique. Those instruments, available e.g. in the UPPAAL SMC tool [11] are able to facilitate the process of creation and analysis of models of dynamic systems e.g. by description and analysis of timing attributes and probabilistic behavior or their digital/analog/hybrid components and their parameters such as dependability or power consumption [12] [8]. The expected output of that phase is a generalized, parameterizable behavioral model of a multicore platform, consisting of key sub-models for the parts such as cores (i.e., computational elements able to execute a task), interconnections as well as interrupt, communication and memory subsystems including their parameters such as estimates of reliability, load, latencies and power consumption.

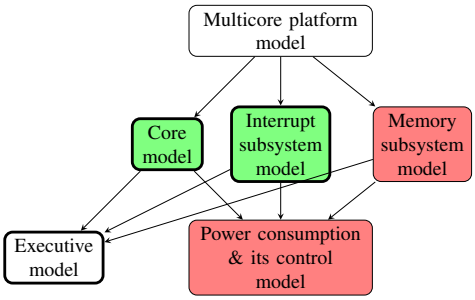

Figure 5. Generalized model of a multicore platform.

On top of the above-mentioned, it is necessary to create models of software layers such as an operating system or an application in order to study how much they are able to affect properties of a multicore system. This includes modeling and analysis of concepts such as tasks, partitioning/scheduling policies, their parameters and behavior within the context of timing, power, reliability, safety etc.

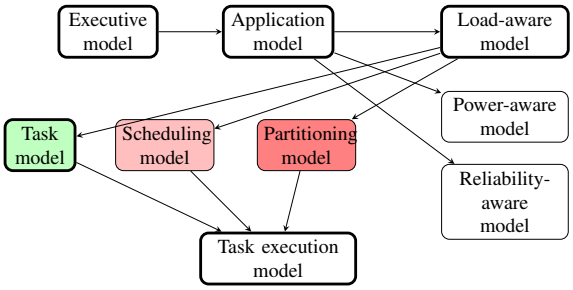

Figure 6. Top level models.

2) Model Validation Phase: To guarantee credibility and validity of models created within III-D1, the models must be made according to practical observations such as experiments over real multicore platforms. However, models for many components such as CPUs or memories and their parameters such as power consumption, load monitoring etc. which be created yet before those experiments are prepared and performed. Actually, the phase is under the preparation; later, it will overlap with III-D1. It is planned that properties of our models (such as deadlock-free operation, liveness, safety, timeliness and reliability) are going to be intensively analyzed by a technique such as SMC [11].

3) Evaluation Phase: In this phase, we plan to be inspired by existing works such as [2] [4] [13] in order to create and/or choose an appropriate comparison basis for evaluation of our approach. In this phase, we plan to focus on proving or disproving our hypothesis stated in III-B, p. 3 and start to summarize the achieved results into the PhD thesis.

Acknowledgment

This work was supported by the Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science – LQ1602" and the inner university project No. FIT-S-14-2297 (Architecture of parallel and embedded computer systems).

The authors would like to thank Mr. Josef Strnadel for PhD study supervision, Mr. Pavol Korček and Mr. Jan Viktorin for information and experience they have provided me.

References

[1] TI, "Multicore Programming Guide," Texas Instruments, Tech. Rep., 2012, http://www.ti.com/lit/an/sprab27b/sprab27b.pdf.

[2] L. Yavits, A. Morad, and R. Ginosar, "The Effect of Communication and Synchronization on Amdahl's law in Multicore Systems," *Parallel Computing*, vol. 40, no. 1, pp. 1 – 16, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819113001324

[3] K.-M. Cho, C.-W. Tsai, Y.-S. Chiu, and C.-S. Yang, "A High Performance Load Balance Strategy for Real-Time Multicore Systems," *TheScientificWorldJournal*, vol. 2014, p. 101529, 2014, DOI: 10.1155/2014/101529. [Online]. Available: http://europepmc.org/articles/PMC4009124

[4] R. Harding, "Synchronization on multicore architectures," Master's thesis, Carnegie Mellon University, 2010.

[5] S. Baruah, M. Bertogna, and G. Buttazzo, *Multiprocessor Scheduling for Real-Time Systems*, ser. Embedded Systems. Springer International Publishing, 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08696-5

[6] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri, *Scheduling in Real-Time Systems*. Hoboken NJ, United States: John Wiley & Sons, 2002.

[7] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978. [Online]. Available: http://doi.acm.org/10.1145/359545.359563

[8] S. Pagani, J. J. Chen, and M. Li, "Energy Efficiency on Multi-Core Architectures with Multiple Voltage Islands," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1608–1621, 2015, DOI: 10.1109/TPDS.2014.2323260.

[9] M. Riša, "Asymmetric Multiprocessing on the ARM Cortex-A9," Master's thesis, Brno University of Technology, 2015.

[10] Home page of the Open Asymmetric Multi Processing (OpenAMP) framework project. [Online]. Available: https://github.com/OpenAMP/open-amp

[11] A. David, K. Larsen, A. Legay, M. Mikučionis, and D. Poulsen, "Uppaal smc tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10009-014-0361-y

[12] W. Dargie, "A Stochastic Model for Estimating the Power Consumption of a Processor," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1311–1322, 2015, DOI: 10.1109/TC.2014.2315629.

[13] J. Mistry, M. Naylor, and J. Woodcock, "Adapting freertos for multicores: An experience report," *Softw. Pract. Exper.*, vol. 44, no. 9, pp. 1129–1154, Sep. 2014. [Online]. Available: http://dx.doi.org/10.1002/spe.2188