computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

HARDWARE MODELS FOR AUTOMATED PARTITIONING AND MAPPING IN MULTI-CORE SYSTEMS USING MATHEMATICAL ALGORITHMS

Lukas Krawczyk, Erik Kamsties

Dortmund University of Applied Sciences and Arts
Emil-Figge-Str. 42, 44227 Dortmund, Germany
E-mail: lukas.krawczyk@fh-dortmund.de; erik.kamsties@fh-dortmund.de

Abstract: Multi-core CPUs offer several major benefits in embedded systems. For instance, they usually provide higher energy efficiency and more computing power compared to single-core CPUs. However, these benefits do not come for free: A program has to be divided into tasks, which can be executed in parallel on different cores. Partitioning of software and mapping on cores are nontrivial activities that require detailed knowledge about the underlying hardware platform, e.g., the number of cores, their speed, available memories, etc. Such information is typically stored in handbooks. If this information would be available in a machine readable model, we call it hardware model, the partitioning and mapping activities can be automated. In this paper, we propose a hardware model and illustrate it using an example of a Freescale multi-core CPU. We then discuss a small case study situated in the automotive domain, which illustrates the use of the hardware model in partitioning, mapping, and code generation. *Copyright © Research Institute for Intelligent Computer Systems, 2013. All rights reserved.*

Keywords: Multi-core; hardware model; embedded systems software development; target mapping; model-driven development; Autosar.

1. INTRODUCTION[1]

The demands on mobile and embedded systems are ever increasing. Mobile phones offer strong multi-media capabilities and, for instance, embedded systems in cars implement image recognition to analyze radar images of traffic in an adaptive cruise control. Mobile and embedded systems benefit in several ways from multi-core CPUs. These CPUs provide more computing power at the same clock speed resulting from several cores working in parallel. They provide better energy efficiency because they run on a lower clock speed compared to a single-core with the same computing power and cores may be switched off if their power is not needed. Furthermore, multi-core CPUs allow for high-assurance systems by running two cores redundantly in a so called lockstep mode.

A program which utilizes the benefits of a multi-core CPU has to be divided into a set of communicating tasks, which can be executed in parallel without blocking each other because of synchronization on shared resources. In order to find an optimal partitioning and mapping, hardware-related information must be taken into account. A trivial example is the number of cores, more advanced information includes the type and speed of shared memories.

Such information about a CPU is typically stored in large processor handbooks. If hardware information would be available in a *machine readable* model, we call it hardware model, the partitioning and mapping activities can be further automated.

In this paper, we propose a hardware model which is rich enough to describe systems of heterogeneous multi-core CPUs as well as peripheral hardware. The use of the hardware model in partitioning and mapping is shown in a case study. Additional application for code generation is outlined. Furthermore, we provide an example of a hardware model for a Freescale MPC5668G multi-core SoC popular in the automotive domain.

This paper is organized as follows: Section 2 discusses the related work. Section 3 outlines the hardware model and the example model is shown in section 4. The main part of this paper is a case study on how hardware models support partitioning, mapping, and code generation, followed by a simple

[1] The research leading to these results has received funding from the Federal Ministry of Education and Research (BMBF) as part of the AMALTHEA project.

Yakindu DAMOS[2] based example illustrating the respective steps of the case study for an automated partitioning and mapping of automotive software in section 6. A conclusion and directions for future work close this paper.

2. RELATED WORK

The idea of utilizing models to support particular steps of development has been pursued for years. For instance, the probably best known application of hardware models is hardware synthesis, which allows transforming a given formal description of hardware into an implementation. This topic is being performed and researched since decades and lead to the introduction of several hardware modeling languages, such as SystemC, VHDL, SystemVerilog etc. [1] and tooling which utilized these languages. Hardware models usually describe the structure and behavior of hardware at a high abstraction level e.g. in terms of register-transfers (VHDL). The true hardware models used by the chip manufacturer for hardware synthesis are considered as intellectual property (IP) and thus usually not publicly available.

Hardware models with an even higher abstraction level have been described in EAST-ADL [2] and AUTOSAR [3]. They are used within the development of automotive embedded systems and support preliminary allocation decisions and the configuration of micro controllers. Compared to these types, we need a hardware model which is located in between: Common hardware description languages like SystemC are still far too detailed to support our cases, and important implementation details, like the cycles per second, are yet unknown. On the other hand, AUTOSAR and EAST-ADL are not specialized enough to provide the required amount of information.

An algorithm for the automatic partitioning and mapping of embedded software for a *homogenous* multiprocessor system has been described, among others, by Cordes et al [4]. It is based on integer linear programming and utilizes a model of the hardware platform with information of its communication and task-creation overheads. Our hardware model targets to support algorithms for *heterogeneous* multi-core system partitioning and mapping, hence much more hardware related information is required. This includes, but is not limited to, the specification of unique characteristics of the cores, e.g. a FPU, as well as additional constraints which will restrict mapping decisions.

Our previous publication [5] is dealing with the partitioning and mapping for heterogeneous multi-core systems using a hardware model. It outlines a pragmatic approach for partitioning and mapping of data flow graphs with a simple algorithm. In this paper, we seek to determine the *optimal* allocation of tasks to cores in due consideration of allocation constraints. As such, we need a more versatile approach, e.g. integer linear programming, which has to be supported by the hardware model.

3. HARDWARE MODEL

The purpose of our hardware model is the support of embedded systems development in general, i.e. regardless of the embedded systems field of application (automotive, mobile, …). Common steps within this procedure are usually partitioning and mapping of embedded software as well as code generation. One of our goals is to provide compatibility with AUTOSAR, which is the current standard in the automotive domain.

The meta-model of our hardware model is shown in Fig. 1. Classes shown in the upper left corner represent the three main hierarchies of elements that may be modeled: *Components*, *Ports* and *Pins*. This structure is oriented at AUTOSARs ECU Resource Description and allows a direct mapping between both models elements. Classes in the upper right corner represent the data-types additional attributes of the elements may take (e.g. *Boolean*, *Integer*, *Long*, ...). Classes in the lower left corner represent the extensions that have been made to enhance the amount of information compared to AUTOSARs ECU Resource Description and as such allowing us to utilize the model beyond the automotive scope.

We introduced a hierarchy for descriptions up to system level, allowing describing a system of ECUs, each consisting of an unbound number of System-on-Chips (SoCs) which may contain multiple cores. Each element may operate on a different frequency which is described by a *Prescaler* and its referenced *Quartz*. The Memory class is used to describe any type of memories on different hierarchies, e.g. a cache as well as RAMs or ROMs. The *Network* class is used to span a network which is accessed through *ComplexPorts*. This allows deriving a memory map out of the hardware model which supports the detection of concurrent access to any type of component.

4. HARDWARE MODEL EXAMPLE

One of the domains which favors the usage of heterogeneous multi-core CPUs is the automotive domain, which is the reason why our example focuses on this branch. Yet the hardware model itself is not limited to this domain.

[2] http://blog.yakindu.org/category/damos-2/

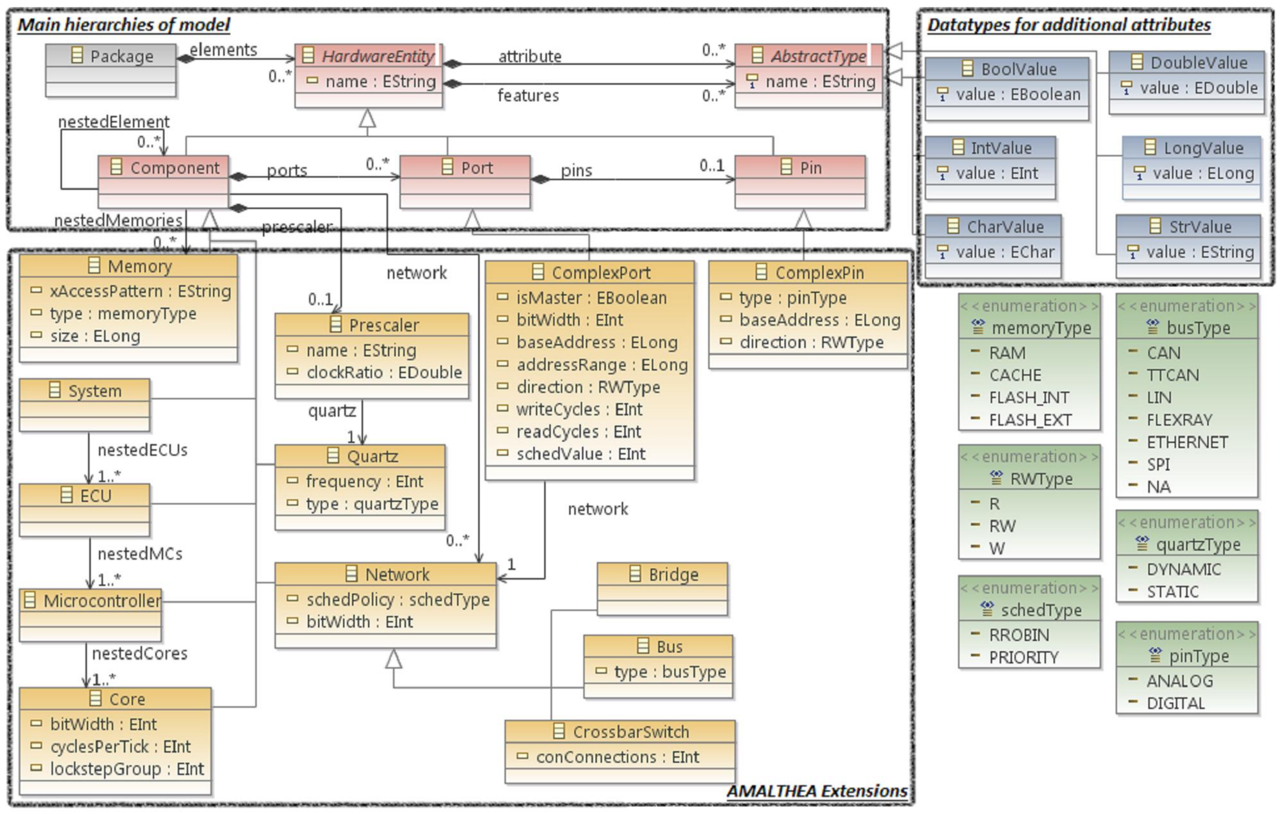

Fig. 1 – EMF based meta-model describing the hardware model.

The simplified hardware model based on the heterogeneous Freescale MPC5668G multi-core SoC is illustrated in Fig. 2. In this figure, blocks in the upper row represent hardware components with master access on the network while the blocks in the lower row indicate slaves. Networks on the SoC which are referenced by Ports (white rectangles) and peripheral elements are represented by blocks in the mid row.

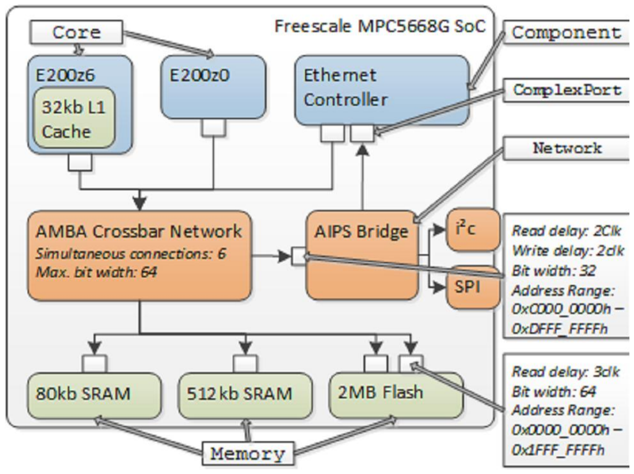

Fig. 2 – Simplified illustration of the MPC5668G SoC hardware model.

The Freescale MPC5668G SoC contains two heterogeneous cores. The main core is the e200z6, which operates at 116 MHz, has 32KB L1 cache and supports floating-point computations. An additional e200z0 core, operating on half the e200z6 frequency, is available as I/O processor. The memory consists of 2MB flash and 592kb RAM which is split up into one 512kb and one 80kb module to allow concurrent access by the masters. The network, which is provided by the AMBA Crossbar Switch, allows up to 6 concurrent connections from masters to slaves with up to 64 bit width. Further interfaces, like I²C and SPI, can be accessed through an AIPS Bridge.

5. USAGE OF A HARDWARE MODEL IN PARTITIONING AND MAPPING

The case study within this paper targets at the hardware model support of the steps which are required to partition and map software to multiple cores and generate target specific ready to compile code. To achieve this, we follow Fosters PCAM methodology for designing parallel algorithms [6]. His methodology specifies the steps partitioning, communication analysis, agglomeration, and mapping. Code generation follows Herrington [7].

5.1. PARTITIONING

The first step is the decomposition of software models, which involves to determine the task granularity and which computations should be part of a coherent set. According to Foster, this step is intended to reveal parallel execution opportunities of a problem by partitioning it into fined-grained decompositions, providing the greatest flexibility for parallel algorithms. However, it should be avoided to replicate data or computations, e.g. both should form disjoint sets.

This step requires information about peripheral elements with their base addresses and address ranges (e.g. a memory map). This allows determining which addresses belong to one specific periphery and which tasks can be merged (as they address the same piece of hardware) or split (as they access different/independent entities of hardware). Based on this information, a partitioning and mapping algorithm is able to analyze software models in consideration of a specific target platform and to decompose tasks in a target optimized manner.

5.2. COMMUNICATION ANALYSIS

The second step is the communication analysis. Once a model has been partitioned into tasks, data dependencies between the respective tasks will have evolved. For instance, a former coherent process might now be split up into multiple processes, with each of them depending on the results of the respective other process.

In communication analysis, such dependencies are identified and the inter-task communication as well as its cost is analyzed. This step has two phases: The first phase involves the definition of a channel structure. Each channel links two tasks and allows the communication between tasks that require data and the respective possessors of these data. In the second phase, the messages which are being communicated on these channels are derived and defined.

To support this step, we need information about data type implementations. As it is well known, the size of data types usually depends on the target operating system and compiler. For instance, data types like *long*, *double* and *int* may have several valid implementations which differ in their size and, as such, affect the communication overhead. Depending on the concrete implementation, the number of transferred bytes by an *int* type variable may vary between 2 and 8 bytes.

5.3. AGGLOMERATION

After an initial set of tasks has been specified and communication dependencies between these tasks identified, it is required to agglomerate the tasks into greater task sets. In the agglomeration stage previous decisions are revisited and the tasks further optimized towards a parallel platform. This may be achieved by simply merging decomposed tasks into one or more greater tasks, for instance, if the tasks have a too fine granularity for a specific underlying hardware platform with a high task creation overhead. Another aspect in the agglomeration step is the replication of computations and data.

To support this step, information about the number of cores, the communication channels and available memories as well as processor cache are required. An agglomeration algorithm will consider cache sizes that will have a significant impact on the decision if data replication should be applied or not. Furthermore, the available capacity of the communication channels of a specific target platform steers the granularity of the agglomerated tasks, e.g. slow channels favor fewer tasks while a high-speed on-chip network could handle even many tasks of fine granularity.

5.4. MAPPING

The mapping step consists of the allocation of software model parts to elements of a hardware platform. The purpose of this step is to specify which task should run on which processor of the target platform, providing the target platform is a multi-core system without automatic task scheduling. The goal of the mapping algorithm is to minimize the execution time by:

- Increasing concurrency, i.e. distributing tasks on different processors.
- Increasing locality, i.e. arrange tasks which communicate frequently on the same processor.

Our hardware model contributes towards this with specific information about cores and their parameters, such as frequency, their target application etc. In addition, a generic possibility to define constraints is provided, as these take a vast variety of options, such as the maximum number of processes, required instruction sets or safety-constraints.

The communication between execution units is the second aspect which has to be taken into account. Naturally coherent computations may only be distributed between interconnected executional units. Additionally, the communication paths between these units may contain several constraints, prohibiting several constellations. It is essential to know these as well as the capacity of the routes. To achieve this, our hardware model provides basic information about the complete network structure of the target system, regardless of the abstraction level its implemented (e.g. network of embedded systems, one specific embedded system or merely a SoC. / micro-controller). The information contains details about any type of map-able network on an abstract level (e.g. the networks speed, address space, scheduling policy etc,) as well as what participants are connected to it. In addition, a generic structure for further constraints is allowing an optimized mapping, e.g. to ensure reliability by safety constraints.

5.5. CODE GENERATION

The final step is code generation for a specific target. Having the tasks and their mapping specified, all required information for a code generator is available and code generation may be performed. Our scope is to develop ready to compile code for a specific hardware, also known as platform dependent code generation.

Two approaches for code generation are available. The first approach is code generation for abstract interfaces. Usually an API for the access to the underlying hardware platform, e.g. a *Hardware Abstraction Layer (HAL)*, is introduced and code utilizing this API generated. However, this approach has several downsides. On the one hand, the complexity of the API to be implemented has to be estimated. An API with little functionality may be realized very fast but will lack in efficiency and/or flexibility. On the other hand, an implementation with a wide scope of functions will be time consuming and only be worth if the platform is used in multiple projects. A tradeoff between these granularities has to be predicted, which is not always possible.

Parameterization of the code generation is the second approach. Here, rules and/or templates are used to customize the code generator for a specific target platform. Templates may be further refined with macros which are replaced by additional hardware related information or code. As it may be considered that the code was transformed correctly, it is unlikely that further maintenance operations by users are necessary. This allows mixing application specific code and target platform specific code, permitting the compiler to perform common optimization techniques. Usually a typical compiler includes a mixture of both approaches, i.e. has several parameterized parts and accesses manually written platform dependent code through a specified interface.

Regardless of the actual approach, the amount of information to support this process is the same. While attributes of certain elements (e.g. ports and pins, registers and memories, data path addresses etc.) provide structural information which is replaced by the code generator, code templates and/or snippets allow to complete more challenging tasks, like the initialization of specific controller or even implement the hardware dependent layers of a messaging protocol with its according functions.

6. CASE STUDY

This section briefly describes the experimental environment for an integer linear programming (ILP) based approach for partitioning and mapping software for an embedded system utilizing the hardware model. A more detailed description of the resp. steps within the tool flow is given by the following subsections. The software is represented by the Yakindu Damos data flow model in Fig. 3 which describes a simple cruise control unit.

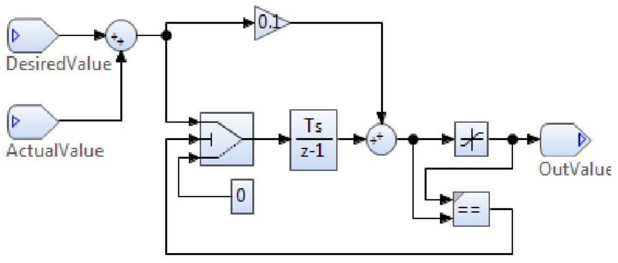

Fig. 3 – Example Data Flow Model

The structure of the steps which are performed with the tool chain in our experimental environment, which has been developed and implemented within the itea2 project AMALTHEA, is shown in Fig. 4.

Its first tool is the *hardware aware partitioning tool*, which will perform the steps partitioning (a) and communication analysis (b) with regard to a chosen hardware platform and pass the resulting model to a so called *graph partitioning tool*. This tool will divide the graph into smaller sub-graphs, which technically equals the agglomeration (c) of smaller executable units into tasks (i.e. each sub graph represents one task). The next step is the mapping (d) which is performed by an ILP based *mapping tool*. In the final step, two *code generators* produce the target platform code (e).

6.1. PARTITIONING

Yakindu DAMOS is capable of extracting a so called *Execution Graph* (i.e. a cycle free graph of the Data Flow Model) which serves as input for our approach. This model has already a very fine granularity, therefore we are able to skip any further decomposition of the model and focus on splitting and merging blocks which are using shared hardware components.

In this example, the blocks `DesiredValue` and `ActualValue` are reading data from the `I²C` and `SPI` interfaces. As shown in the Freescale MPC5668G SoCs hardware model (Fig. 2), these interfaces are located behind the `AIPS Bridge`, therefore it is wise to merge them in order to reduce overheads which might occur by task creations or context switches and prevent mutual exclusion (Fig. 4(a)).

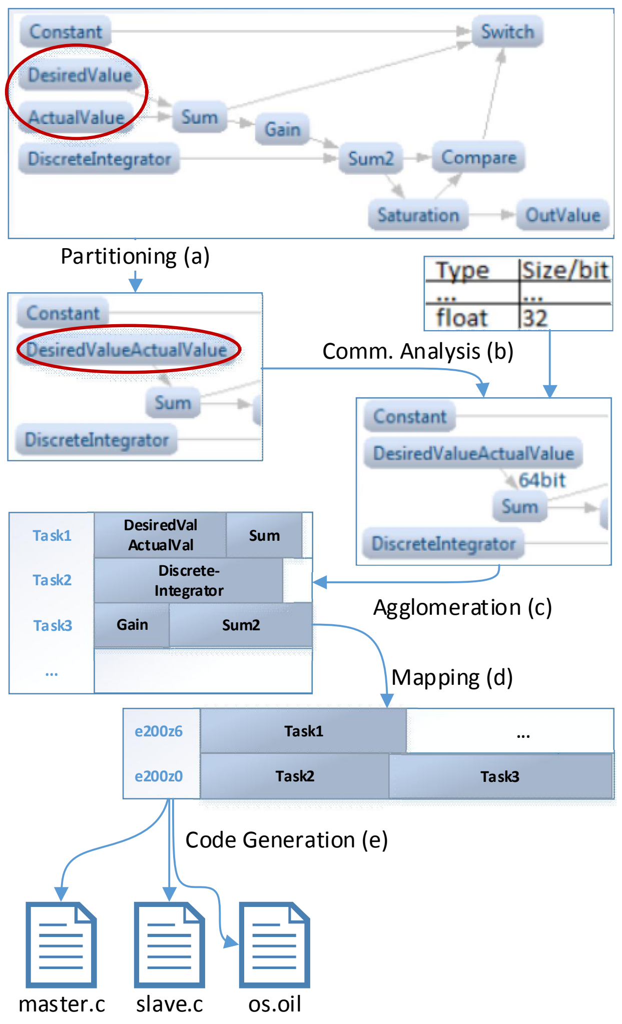

Fig. 4 – Partitioning and mapping approach.

6.2. COMMUNICATION ANALYSIS

The communication cost of the model can be determined by the number of data transfers between the blocks and their respective data type size. As usually several operating systems are available for a specific hardware, the information about the concrete implementation is stored in tables that are attached to the hardware model. This allows to calculate the communication cost and enhance the edges of the execution graph with these values (Fig. 4(b)), providing the required information for the following steps.

6.3. AGGLOMERATION

The agglomeration is performed by a graph partitioning algorithm based on vector clocks (see [8]). This algorithm is implemented in a *graph partitioning tool* which merges the Blocks from the DFG model into larger groups of tasks. The hardware model supplies information about (i) the maximum number of simultaneously executable tasks (i.e. cores, threads per core, …) as well as (ii) the task creation overhead. This allows steering the granularity of the resulting tasks resp. the maximum number of tasks to create. The graph from the previous step describes the relation between the blocks as well as their coherence, which has impact on the sorting order of the Blocks as well as decision which Blocks will be distributed into the which task (Fig. 4(c)).

6.4. MAPPING

The mapping (Fig. 4(d)) in this algorithm is performed by a pragmatic ILP (Integer Linear Programming) approach based on [9], which focusses on minimizing the maximum execution time of concurrently operating cores. The mapping tool utilizes the open source oj!Algorithms[3] project which we use to solve the ILP equations.

The first step in this mapping algorithm is to determine the required execution time $ET_{i,j}$ of each task i for the resp. core j. This is can easily be determined by (1) and (2).

$$ET_{i,j} = CT_i * CC_j, \tag{1}$$

$$CC_j = TC_j * PS_j * Q_j, \tag{2}$$

where CT_i is the number of cycles which are required to process the task and CC_j the number of cycles the core can process per second. The variable TC_j describes the number of ticks that are required to process one cycle, PS_j the prescaler for frequency scaling and Q_j the frequency of the quartz attached to the core.

The second step is the formulation of mapping constraints e.g. to limit the number of cores a task will be allocated to one core (3)

$$\sum_{j=1}^{n} A_{i,j} = 1 \;\; \forall i \in [m], \tag{3}$$

with $A_{i,j}$ describing the allocation of task i to core j, m number of tasks and n number of cores.

This equation however is only valid, if all cores are suitable to process this task. As some of the tasks may contain special requirements on a core, e.g. the presence of a Floating Point Unit (FPU) or a specific instruction set, it would be also required to narrow

[3] See http://ojalgo.org/

down the solution space to valid cores. In our example for instance, only the main core *e200z6* contains a FPU, hence we would need to limit the scope of valid cores to this core only.

A general formulation to narrow down the solution space for multiple valid cores is given in (4)

$$A_{i,j} = 0 \ \ \forall j \notin V , \tag{4}$$

with V being a group of valid cores.

A very simple approach to minimize the execution time can now be achieved solving the equations (5) and (6) as mentioned in [9]

$$z \to \min , \tag{5}$$

$$\sum_{i=1}^{m} A_{i,j} * ET_{i,j} \le z \forall j \in [n] , \tag{6}$$

with z being the maximum execution time of all concurrently executed cores.

6.5. CODE GENERATION

The code generation for this algorithm is performed by two code generators (Fig. 4(e)).

The first code generator is provided by Yakindu DAMOS and innately capable of producing hardware independent code for the resp. blocks of the data flow graph. To support target ready code generation, hardware related information is provided by the hardware model. This consists of libraries for I²C and SPI access as well as the addresses of hardware components, i.e. the memories and peripherals.

However, the code for the blocks on its own is not sufficient to provide target ready code. For instance, it is still necessary to merge the blocks code into tasks and allocate those to cores etc. This is done by the operating system (OS) code generator. Among others, its purpose is to create the task code which will call blocks and distribute the code to the respective cores C files. Furthermore, it will create the input files for the targets compiler which will specify the mapping from tasks to cores as well as to provide the libraries for advanced controllers, e.g. optional CAN, LIN or Ethernet controllers.

7. CONCLUSION AND OUTLOOK

This paper introduces a hardware model which is capable of supporting automated partitioning and mapping in heterogeneous multi-core systems. The case study has shown how our hardware model is able to support the involved steps and which amount of hardware related information is required for an automated execution. Furthermore, it has outlined a feasible implementation of these aspects as part of a seamless tool chain.

Future work will target the development and implementation of advanced partitioning and mapping algorithms with different goals (i.e. energy efficiency), multiple constraints (e.g. bus access) as well as their support with hardware models.

8. REFERENCES

[1] P. Marwedel. *Embedded System Design: Embedded systems foundations of cyber-physical systems*, Springer Science+ Business Media, 2011.

[2] A. P. Consortium et al., EAST-ADL domain model specification, 2010. [Online]. Available: http://www.east-adl.info

[3] AUTOSAR, Automotive open system architecture, 2007. [Online]. Available: http://www.autosar.org

[4] D. Cordes, P. Marwedel, and A. Mallik. Automatic parallelization of embedded software using hierarchical task graphs and integer linear programming, in *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2010, pp. 267–276.

[5] L. Krawczyk and E. Kamsties. Hardware models for automated partitioning and mapping in multi-core systems, in *Proceedings of the 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS 2013)*, Berlin, Germany, 12-14 September 2013, pp. 721-725.

[6] I. T. Foster. *Designing and Building Parallel Programs*. Reading, Mass.: Addison-Wesley, 1995.

[7] J. Herrington. *Code Generation in Action*, Manning Publications Co., USA, 2003.

[8] R. Hoettger, B. Igel and E. Kamsties. A novel partitioning and tracing approach for distributed systems based on vector clocks, in *Proceedings of the 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications* (IDAACS 2013), Berlin, Germany, 12-14 September 2013, pp. 670-675.

[9] M. Drozdowski. *Scheduling for Parallel Processing*, Springer London, 2010.

Lukas Krawczyk *received his B.Sc. degree in informatics from Dortmund University of Applied Sciences and Arts in 2009 and is currently working as research assistant within the AMALTHEA project. His scientific interests are: Embedded Systems Development, hardware level programing.*

Erik Kamsties *is a professor for software engineering and embedded systems at the Dortmund University of Applied Science and Arts. He received a Diploma (M.S.) from the Technical University of Berlin and a doctoral degree (Ph.D.) in computer science from the University of Kaiserslautern, (Germany).*

His current research focuses on requirements engineering, model-driven development, and embedded multi-core systems.