Support for High Performance using Heterogeneous Embedded Systems - a Ph.D. Research Proposal

Gabriel Campeanu
Mälardalen University
Västerås, Sweden
gabriel.campeanu@mdh.se

ABSTRACT

Nowadays it is more common to build embedded system on a heterogeneous platform, i.e. a platform containing different computational units such as mCPU, GPU and FPGA. This enables better performance, but also introduces additional complexity with respect to the software deployment. For complex systems it is not obvious which deployment is the best. For different constraints and requirements, different deployment configuration can be optimal. To address this problem, an approach is to model the system, including both software and hardware parts, with specification of extra-functional properties. The deployment can be then modeled and an (semi)optimal solution can be provided. In this paper we present an overview of our planned research on software modeling and software deployment of heterogeneous embedded systems, which enables assisting the developer in designing this type of systems.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: [Design tools and techniques]

General Terms

Architecture, Deployment

Keywords

Heterogeneous systems, component-based software engineering, model-based engineering, extra-functional properties, component deployment

1. INTRODUCTION

The latest advances in technology and frequency scaling favored the computer applications to increase their performance. As along with the technology advances, the software complexity increases, the applications performance is confronting several hardware obstacles such as memory size or computational power. The resources constraints are in particular important for embedded systems. Significant improvements in increasing performance and decreasing the dependencies of the resources can be achieved by heterogeneous embedded systems, which bring together, along with a general purpose processor (GPP), other specialized resources such as graphics processing unit (GPU), or field-programmable gate array (FPGA). This mixture of various computational units makes a heterogeneous system to behave in a similar way as a parallel computing or multi-core computing system.

The heterogeneous embedded systems, which allow high amount of data to be processed in real-time, have various type of applications in different areas of autonomous systems. For example, in the automotive industry a considerable effort is made in researching and implementing vision systems which can identify obstacles such as people or other vehicles. The massive amount of data which is produced by the sensors need to be processed in real-time in order to make the 3D reconstruction. In this context, a heterogeneous system processes the raw data using an FPGA, and send it to a multi-core CPU which may use a GPU as a co-processor.

In the domains of general-purpose systems two proven approaches to meet the challenges related to complexity are component-based software engineering (CBSE) and model-based development (MBD).

Our objective is to use CBSE and MBD techniques and apply them for heterogeneous embedded systems. Using MBD and CBSE, our goal is to model the system deployment, and to provide for the model a feasible and possibly optimal deployment solution. The modeling includes the system architecture (so both software and hardware) along with certain extra-functional properties (EFPs), e.g., performance, resource utilization. The (optimal) deployment will be calculated to satisfy requirements for the EFPs.

The rest of the paper is structured as follows. We start with a background of heterogeneous systems in Section 2. Section 3 presents the motivation, the goal and the plan of our research. The related work is presented in Section 4, followed by conclusion in Section 5.

2. BACKGROUND

This section presents the characteristics of embedded heterogeneous systems, followed by challenges in present software design.

2.1 Embedded heterogeneous systems

WCOP'13, June 17, 2013, Vancouver, BC, Canada.

A typical modern embedded system heterogeneous platform includes single-processor units or multi-core general purpose processors (GPPs) with graphics processing units (GPUs) and field-programmable gate arrays (FPGAs). Each new technology has its own architecture, properties and limitations. Deployment is an important aspect in designing heterogeneous systems; it may lead to bad performances when the software is not optimally deployed onto the hardware.

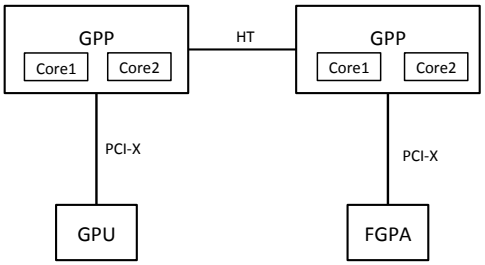

Figure 1: Example of heterogeneous system

Figure 1 presents an example of a heterogeneous platform, which contains two multi-core GPP (e.g., two AMD dual-core processors), one GPU and one FPGA. The processors are connected using an Hyper Transport bus, and the GPU and FPGA are connected to the processors using a PCI bus. On this platform we can execute various applications such as audio and video processing or sensor based signal processing applications. The execution of the application can be divided between the units of the platform in such a way that we can run, for example, pattern recognition on the mGPP, while on the GPU we can run parallel computing activities. Some software components which run on the GPP may have better performances if are executed on the FPGA, but requires different programming and different specifications. An important aspect of this type of application is that it requires the data to be processed in real-time; if is not, then the system is useless. This streaming application is a typical example which shows that EFPs are as important as functional properties (FPs). EFPs like performance, energy consumption, dependability, maintainability, usability, security, etc., need to be specifically addressed during the development of the system to ensure the quality level of the desired service.

The new heterogeneous platforms are suitable for streaming applications because of their diversity in processing units and memory distribution. While, for a multi-core GPP, all of its cores can access the same processor memory, the other units such as GPUs or FPGAs can access their own and the main memory. This distributed way of the memory increases the software time execution, which can be exploited by the applications which are processing high amount of data.

One of the main challenges of heterogeneous embedded systems is the distribution of the workload over the various computing units of the system. For a large embedded system, many deployment schemes (i.e., mapping portion of software onto hardware nodes) are possible. Some schemes will not satisfy the requirements; others can improve a particular property of the system while satisfying the requirements. For example, the performance of a system can be improved if the workload is distributed in such a way that the more powerful processors do more work, while the less powerful processors execute less work.

2.2 Challenges in software design

Modern ways of designing systems are using MBD or CBSE, which are raising the abstraction level of the implementation, focusing on models. Using models in designing embedded system has its challenges, one of it referring to the specification of components. In most of the component models, the components comply to the same specification rules, i.e., FPs and EFPs. In embedded systems design, we need to use different approaches which may use distinctive component instances or different specifications of EFPs. A component instance describes the component with the same FPs, but with different EFPs. This implies that, the design process may address different instances of the same components. Because of heterogeneity of the EFPs for different implementations, there is a need to extend standard modeling languages, or define a domain-specific language.

Using MBD, both software and hardware should be modeled. The software model should contain components with platform-independent properties, but also properties which are depending on different platforms. The hardware model should contain components with hardware nodes specifications. During the code generation process, the code should be generated according to the platform context. Using CBSE to design embedded systems, the system is seen as a composition of components. To address the reusability of the components, which can be used to model different hardware platforms, a special attention should be placed on the way EFPs are specified. The management and specification of EFPs are even more challenging, if we add real-time constraints.

3. PROPOSED RESEARCH

In this section, we first present the motivation and goal of our intended research in heterogeneous embedded systems, followed by the research questions and the steps planned to take in achieving our goal.

3.1 Motivation

The hardware technologies breakthrough in the last years allow building embedded systems with high input data rates. The new systems, which can process huge amount of data in real-time, can find various application usage from different domains, such as automotive industry. For instance, many research initiatives are focused on different types of vision systems for various vehicles. These vision systems need to process high amount of data from the input sensors, in a real-time perspective. The raw input data may be handled by an FPGA, after which may be sent to a CPU which may use a GPU as a co-processor. Comparing with homogeneous systems, the new heterogeneous systems bring a high performance due to the diversity of processing units and their memory distribution. The need and use of such products will grow in the future, which will make the industry that master the new technology to gain a significant leverage on the global market.

To optimize a performance metric (e.g., system utilization, latency) of a system, is not a straightforward issue, and it deals with finding the right deployment of software onto hardware. Finding a suitable distribution is a complex process which deals with various factors such as a large

number of possible deployment schemes, hardware resource restrictions or software resource demands.

There is a lack of methods to model the software of heterogeneous embedded systems. The software system is defined by its properties, i.e. FPs and EFPs. While reusing a software component in different heterogeneous architectures, its EFPs may be very different. This requires novel methods to specify EFPs which should include platform-specific parameters and software parameters.

3.2 Research questions

Considering there are various ways in building embedded systems, some being more efficient then others, we are addressing a general research question which can be formulated as follows: *How can we facilitate the development of heterogeneous embedded systems?* This question addresses a large area of research; in order to refine it, we propose two contributions as follows:

- Developing a domain-specific modeling language for heterogeneous systems; the language covers software, hardware and deployment models.
- Developing a semi-automatic deployment method which can optimize the system performance.

3.3 Research context

We propose to use a model-based approach when developing heterogeneous embedded systems. While following this approach, our research will be focused on modeling the software and hardware infrastructure, and on optimization of the deployment.

We see the *software model* as a composition of components, where each component is characterized by its own FPs and EFPs (e.g., required memory). While there are means to specify the FPs of a components (e.g., interfaces), we need to develop novel methods or adapt existing methods for EFPs specification which should include platform-specific parameters and also software parameters. In addition, we need to provide support for management of EFPs, as well as composition of them.

A heterogeneous hardware platform consists of various processing units, where each unit it is characterized by its own architecture, properties and limitations. The *hardware model* is a composition of nodes, each one being characterized by a set of properties (e.g., available memory). The *deployment model* can be explicitly defined by the developer, or it can be created in a semi-automatically process, and it will define which component runs on which hardware node. The consequence of a particular deployment, comparing to other deployments, may have different quality aspects (e.g., average time, worst-case time, precision, energy consumption, memory usage). In order to improve a particular quality property of a heterogeneous system, the developer needs to define one (or several) optimization criterion. The semi-automatically process will determine one best distribution, with respect to the optimization criterion. As input, the optimization model will take some restricted form of software and hardware models, along with the optimization criterion.

We propose, as an initial representation of the software model, to use, for simplicity, a graph notation where the nodes stand for the software components, while edges act as the components communication. Both directed and undirected graphs may be used to represent the software model. While the directed graphs may represent the execution dependencies between the components, the undirected graph may represent the information exchanged between components. Similar to the software model, the hardware infrastructure may be represented as an undirected graph, where the vertices stand for the processing units, while the edges act as the communication between the hardware units. Later in the research, when we will have complex models, we propose to use a more formal representation (i.e., meta-model) to describe the models and their specifications.

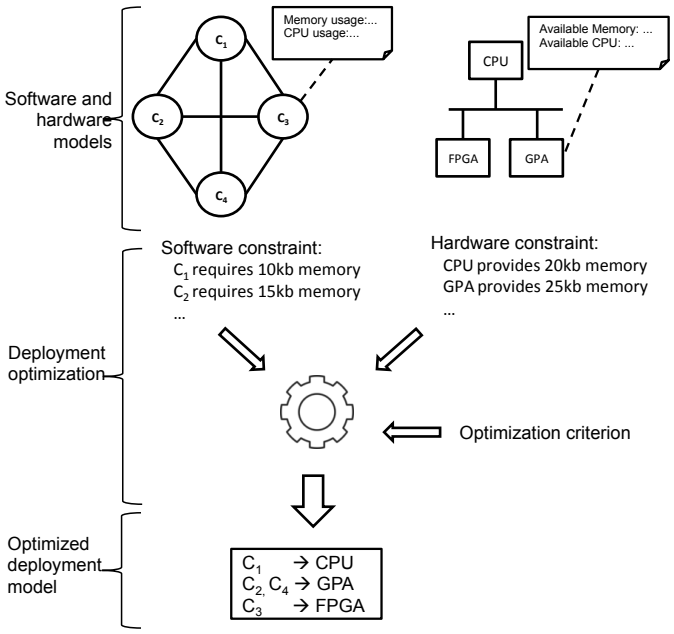

Figure 2: Overview of the proposed approach

The overview of our approach can be visualized in figure 2. In the upper part of the figure, the software model and hardware infrastructure are described. The software model is presented as an undirected graph with four nodes, each node being characterized by two properties, i.e., memory usage and CPU power usage. In the same manner, the hardware platform is described. In the middle part of the figure, based on the software demands and hardware constraints, along with the optimization criterion provided by the developer, a good optimization deployment is searched. In the bottom part of the figure, an optimized deployment model is provided, which is further allocated on the hardware platform.

The factors which are considered in the optimization process are:

- Hardware resource constraints. Based on different hardware nodes properties described in the hardware model, certain distributions are ruled out. For example, we can no deploy software components onto a hardware node which has insufficient memory to hold all the units assigned to it.
- Software resource demands. The software components, through their specifications described in the software

model, place different resource demands on the hardware infrastructure. For example, the bandwidth requirement between two components placed on different hardware nodes, demands a certain available bandwidth communication between the two nodes.

- Optimization criterion. There are many feasible deployments, but we need to select the "good" deployment in order to enhance a particular system quality (e.g., performance).

3.4 Research plan

We have started our research with a literature review on existing modeling languages and how they manage to handle the EFPs. We need to adapt an existing one, or to develop a new modeling language which can address the heterogeneous embedded systems. The language should model both the software and hardware parts of the system. To do that, we need to identify the properties relevant in our research. Having in mind that our research is focusing on augmenting performance of heterogeneous systems, the extra-functional properties should mainly be focused on performance properties, such as execution time, response time, throughput, latency or resource utilization. After studying the relevant properties, the focus should be placed on their specification and instantiaton for different embedded contexts. In the next step, a framework for managing EFPs should be developed. The framework should handle the properties specification independent of deployment, but also should manage the component properties which specify all possible deployment values.

As a result of our previous research work, we can start developing a prototype tool for software and hardware modeling.

The next step in our research is addressing the optimization challenge. In order to understand how to distribute the functionality over an hardware platform, it demands of understanding the consequences of a particular deployment. We need to study the importance of different quality aspects such as average time, worst-case time, memory usage. In the next step, we need to develop a way to derive the system properties. Starting from a allocation mapping, we need to study how to derive a system property from the EFPs of individual components and hardware platform specification. Once we have the method to derive properties, we can use it to guide an automated search of suitable deployment.

The modeling language and optimization method will be developed and evaluated in several iterations.

Evaluation - case study

To examine the findings from our research we will use an underwater robot with stereo cameras, as a demonstrator. The robotic industry is a domain which is increasing fast, and is finding new application areas, such as robots for environmental rescue operations. We consider an underwater robot to be an appropriate demonstrator for our research area, in this way connecting our research with the industry demands.

The hardware platform used by our demonstrator is composed of a multi-core GPP, one GPU and one FPGA. We propose to use, as a starting point of our case study, a system characterized by a small number of EFPs (i.e., memory, CPU). Using our deployment optimization method, we can find a good software deployment onto hardware platform. One way to test the feasibility of our optimization method is to compare the results of our found optimized deployment against several random feasible distributions. In the next iteration phase, we will consider a more complex system by adding and refining the system properties, such as the static and dynamic memory, the frequency of component communication or the size of components data exchanged.

We can further test the flexibility of our novel modeling language by using it on a different heterogeneous embedded platform, and examine how it manages to handle the domain-specific EFPs composition. In the future, we intend to enhance the performance of our demonstrator by adding several numbers of computational units. This will increase the complexity in finding a good optimized deployment, which can be a good test on the method feasibility.

We can also verify the easiness in designing heterogeneous systems using our methods by comparing with the tools and methods already existing on the market (e.g., MARTE).

3.5 Benefits

The benefits of our research will be represented by a novel domain-specific modeling language and a semi-automatic deployment method. The language will assist the developer in modeling the system, while the deployment method will ease the distribution process of the software onto the hardware platform. In this way, a developer will have efficient means to design heterogeneous embedded systems.

A tool can be developed based on our novel methods; it can provide assistance to the developer during the process of designing heterogeneous systems. The tool can be used in the academic community, in form of open development for learning purposes.

Having an working demonstrator in the form of an underwater robot, will motivate the industry to show interest in exploiting our methods and tool, or even to further develop them.

4. RELATED WORK

In the first part of this section, we present an overview of different component-based and model-driven models for embedded systems and how they manage to provide support for EFPs. The second part will cover different methods of software deployment onto hardware platforms.

4.1 Modeling

UML [16] is a tool which allows modeling of about any type of application and helps in visualizing and documenting the software models. Its flexibility allows to model distributed applications, build OO concepts (e.g., classes) or real-time, fault-tolerant systems. MARTE [18] is an UML profile, which adds capabilities for model-driven development of real-time and embedded systems, providing ways of modeling both hardware and software aspects. MARTE enables also analysis aspects of the models. SysML [17] is another UML profile that provide modeling support for complex system such as hardware and software.

The **SaveCCM** [2] is a component model developed at Mälardalen University. The model is intended to be used in the vehicular and safety-critical embedded systems domains. The model is build by interconnecting three distinct type of elements (i.e., components, switches and assemblies) with well defined interfaces. The pattern of the model is in-

spired from the pipe-and-filter paradigm with an important attribute appended, given by the distinction between data transfer and control flow.

The **Rubus** component model [11], developed by the cooperation of Malardalen University and Articus system, is intended to be used in the development of distributed real-time systems. The model intents to support the following activities: design, analysis and synthesis, which can have sometimes contradictory requirements. The goal of the model is to support and balance the common requirements of the previously enumerated activities, in order to gain industrial usefulness.

The **Palladio** component model [6] is a meta-model used to describe component-based software architectures, and which addresses the prediction of components attributes, especially performance and reliability. The performance of a software component is highly influenced by its usage. The resource demands of a component is directed connected with the input parameters. The model is using some probability function to express the component resource demand. The Palladio model is important to our research because of the prediction of EFPs, which are relevant to our study.

In [19] Sentilles et al. propose a model to specify the component attributes and their model integration. The attribute is defined as a tuple $<$*TypeIdentifier, Value*$>$, where *TypeIdentifier* defines the EFP, and *Value* is defined as triple $<$*Data, Metadata, ValidityConditions*$>$, where *Data* is the property value, *Metadata* provides extra information of the property, and *ValidityConditions* presents the conditions under the property data is valid.

Another model which tackles the management of EFPs is the **Robocop** [15] model. This is done by using a resource model which describes, using mathematical cost functions, the resource usage of components. Besides Robocop, the **Koala** [20] model addresses also the management of components EFPs but is limited to only one, i.e., the static memory usage of components. In Koala model, each component has attached an extra interface which describes the information of this static property. The interface can not be added to already created components.

A classification of component models is presented in [8]. The paper presents the fundamental principles of component models and provides a classification framework for component models.

4.2 Deployment optimization

The deployment of the software model onto the hardware platform is know to be NP-hard [5], which means it would take years to investigate all potential deployment schemes when we have large number of units. This challenge was addressed in different forms, the most used one being the heuristic methods. There are several literature reviews on software architecture optimization methods [3], performance evaluation of component-based systems [13] and model-based performance prediction [4].

The heuristics approaches provide a fast way to obtain sub-optimal solutions. These methods are useful on applications where finding the solutions is not bounded by a time limit. One of the heuristic approaches, the **genetic algorithm** [9] was used for solving different allocation problems, such as task allocation onto distributed systems [1, 21]. The method is based on the methods of natural section and natural genetics. In this technique, a possible solution to the problem is represented by an individual which is composed of a chain of genes. A pool of individuals construct a population. The population evolves to a next generations, in which a new individual is created in three steps: 1) A pair of individual parents are selected, 2) A crossover mechanism is performed, in which partial solution are exchanged between the parents, and 3) A mutation mechanism is performed by changing few randomly selected genes on the new child individual.

Another often used heuristic approach, the **simulated annealing** (SA) method [12], was used for solving various optimal allocation problems, such as distribution of program modules to processors [10]. The method is inspired by the annealing process in metallurgy, where a number of steps involving heating and cooling of materials are made in order to increase the size of the metal's crystals while its defects are reduced. If every point from the search space corresponds to a state of a physical system, then the goal of the method is to take the system from an arbitrary initial state to a state characterized by a minimum energy using a number of iterative steps. At each step, SA considers some neighbor state of the current state and, based on some probabilities, decides to move or not to it in such a way that the current state will be characterized by a lower energy.

The **branch and bound** [7] method is another heuristic method used to find optimal solutions for several allocation problems, such as task allocation onto processors in distributed systems [14]. It uses a search tree to describe the allocation problem. The method is characterized by four steps: branching, bounding, selection and elimination, which are used on several iterations, during which the a best solution is progressively improved. In the first step, the problem domain is divided into several smaller subsets, on which the same optimization problem is defined. The second step defined the bounds of the optimal solution of the considered problem. The selection step identifies a new solution, while the last step eliminate nodes which do not lead to optimal solution.

Several of the optimization solutions for the allocation problem are not addressing the performance properties and others are addressing only few performance properties, for example optimizing the distribution taking in consideration only the latency of the system. Also, the optimization solutions are constructed for different type of systems, e.g., wireless systems, distributed systems, which requires adaptation to heterogeneous type of system. Our research is not trying to develop new optimization methods, but to adapt and use the existing methods onto heterogeneous environments.

5. CONCLUSION

This article has described our research plan on supporting high performance for heterogeneous embedded systems. In the first part of the paper, we presented the background followed by the motivation and goal of our research, while the last part discussed our research plan, expected results and related work.

The latest hardware technology which allows high amount of input data, opens new opportunities to the industry market. Heterogeneous embedded systems, containing multicore processors with GPUs and FPGAs which are nowadays quite straightforward to build, are characterized by various performance capabilities, limitations and architectures. In

parallel with the increasing of the hardware platform complexity, the software of such systems is becoming more and more complicated. The increased complexity of a system makes it difficult to handle and distribute the software model onto the hardware platform using traditional development methods. Our research focuses on facilitating software development across heterogeneous embedded systems using new software development paradigms such as MBD or CBSE. The optimization of the software deployment onto the hardware platform, considering both the software and hardware properties, along with their limitations and constraints, represents a significant challenge in the development of heterogeneous embedded systems.

6. REFERENCES

[1] J. Aguilar and E. Gelenbe. Task assignments and transaction clustering heuristics for distributed systems. *Journal of Information sciences*, 1997.

[2] M. Akerholm, J. Carlson, J. Hakansson, H. Hansson, M. Nolin, T. Nolte, and P. Pettersson. The SaveCCM Language Reference Manual. Technical report, Malardalen Univeristy, 2007.

[3] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. In *IEEE Transactions on Software Engineering*, 2012.

[4] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: A survey. In *IEEE Transactions on Software Engineering*, 2012.

[5] S. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004.

[6] S. Becker, H. Koziolek, and R. Reussner. Model-Based performance prediction with the Palladio component model. In *Proceedings of the 6th international workshop on Software and performance*, 2007.

[7] E. G. Coffman and J. L. Bruno. *Computer and Job-Shop Scheduling Theory*. Wiley, 1976.

[8] I. Crnkovic, S. Sentilles, V. Aneta, and M. R. V. Chaudron. A classification framework for software component models. *IEEE Trans. Softw. Eng.*, 2011.

[9] D. E. Goldberg. *Genetics algorithms in search optimization and machine learning*. Addison-Wesley, 1989.

[10] Y. Hamam and K. Hindi. Assignment of program modules to processors: A simulated annealing approach. *European Journal of Operational Research*, 2000.

[11] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindbergand, and J. Lundback. The Rubus component model for resource constrained real-time systems. In *International Symposium on Industrial Embedded Systems*, 2008.

[12] S. Kirkpatrick. Optimization by Simulated Annealing: Quantitative Studies. *Journal of Statistical Physics*, 1983.

[13] H. Koziolek. Performance evaluation of component-based software systems: A survey. *Journal of Performance Evaluation*, 2010.

[14] P. Y. R. Ma, E. Y. S. Lee, and M. Tsuchiya. A Task Allocation Model for Distributed Computing Systems. In *IEEE Transactions on Computers*, 1982.

[15] H. Maaskant. *A Robust Component Model for Consumer Electronic Products*. Springer, 2005.

[16] OMG. Unified Modeling Language. `http://www.uml.org/`, 1990.

[17] OMG. System Modeling Language. `http://www.omgsysml.org/`, 2001.

[18] OMG. Modeling and Analysis of Real-Time and Embedded Systems. `http://www.omgmarte.org/`, 2009.

[19] S. Sentilles, P. Stepan, J. Carlson, and I. Crnkovic. Integration of Extra-Functional Properties in Component Models. In *Proceedings of the 12th International Symposium on Component-Based Software Engineering*, 2012.

[20] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The koala component model for consumer electronics software. In *Computer*, 2000.

[21] D. P. Vidyarthi and A. K. Tripathi. Maximizing reliability of distributed computing systems with task allocation using genetic algorithm. *Journal of System Architecture*, 2011.