Multi-Criteria Software Component Allocation on a Heterogeneous Platform

Ivan Švogor[1], Ivica Crnković[2], and Neven Vrček[1]
[1]*University of Zagreb, Faculty of Organization and Informatics, Pavlinska 2, 42000 Varaždin, Croatia*
[2]*Mälardalen University, School of Innovation, Design and Engineering, Box 883, 72123 Västerås*
E-mail(s): isvogor@foi.hr, ivica.crnkovic@mdh.se, nvrcek@foi.hr

Abstract. *A recent development of heterogeneous platforms (i.e. those containing different types of computing units such as multicore CPUs, GPUs, and FPGAs) has enabled significant improvements in perfo-rmance processing large amount of data in realtime. This possibility however is still not fully utilized due to a lack of methods for optimal configuration of software; the allocation of different software components to different computing unit types is crucial for getting the maximal utilization of the platform, but for more complex systems it is difficult to find ad-hoc a good enough or the best configuration. In this paper we present an approach to find a feasible and locally optimal solution for allocating software components to processing units in a heterogeneous platform.*

Keywords. Software components, heterogeneous platform, component allocation.

1. Introduction

The computer systems today are becoming heterogeneous; alongside multicore Central Processing Units (CPU), Graphical Processing Units (GPU) and Field Programmable Gate Array (FPGA) are gaining an important role [1]. Such systems consist of different types of computing units, where each unit can be dedicated for a particular type of computation. This is already a proved approach in high-performance computer systems, in which GPUs process highly parallel computation, and in which cores from multicore CPUs perform different tasks in parallel. This is also becoming of significant importance in embedded systems where such systems enable processing of large amounts of data streams in real time. However this great potential for increased performance is still not fully utilized due to a lack of software methods for an efficient and optimal placement of software to the heterogeneous platform by which an optimal, or sufficiently good performance is obtained. Different software allocation results in different performance, due to a type of computation, and communication between software components that are allocated on different units. It is not obvious which allocation would enable the best performance. In addition, a possibly best allocation might not be allowed due to different constraints; this can be due to limitation of resources of a particular unit (such as memory or communication capacity), or due to some architectural decisions related to specific requirements (such as a requirement that two components are not allowed to be allocated to the same physical node). A "trial and error" method by repeated allocation and measurement is inefficient, in particular when the software implementation may depend on which platform will be executed. For this reason, an allocation method in an early development phase is desired.

The main goal of this paper is to define a model for the software allocation optimization on computing units in heterogeneous systems.

We assume that we are dealing with component-oriented systems, so each software component[1] can be allocated to a computing unit. Further we define a model that can be used in an early phase of the development lifecycle, in the early architectural design of the system. The components may already be implemented, or we can use models, not yet implemented, but specified, and in which case each component is defined by a set of attributes, estimated or obtained in a certain way. The result of the model is a proposed deployment configuration that is optimal, or nearly optimal for the overall system performance.

The rest of the paper is organized as follows. In the second chapter we define the problem and its formal description. In the third chapter we present our solution for the allocation problem

[1] In further text when we refer to a component we mean "software component"

using genetic algorithm. Chapter 4 illustrates the model on an example. Chapter 5 briefly discusses the related work, and finally Chapter 6 concludes the paper.

2. Software Allocation Optimization Model

Our model is composed of a Software System S, which consist of components, and a hardware platform H which consist of a set of computing units as shown in Figure 1.

Every component $s_i \in S,\ i = 0, \dots, n$ needs to be allocated to a computing unit $h_i \in H,\ i = 0, \dots, m$. All possible allocations of software components to computing units is m^n (permutation with repetition). This number increases rapidly with number of the components and the platforms, so to find the optimal component allocation with a respect to a particular goal is (at least) very time-consuming if the process is performed manually. For this reason we provide a theoretical model for finding an optimal (or locally-optimal) allocation with a respect to a particular system property for given characteristics of the software systems, components and the hardware platform.

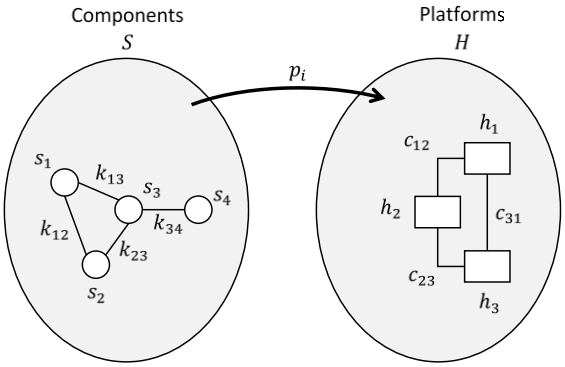

Figure 1. Software components allocation to heterogeneous hardware platform

Our model is defined as follows.

a) A node provides a set of resources for the components being executed on it, e.g. CPU power, or available memory. The capacity, i.e. the amount of the resources of each node must satisfy the needs for the component resources.

b) We enable components' mapping to the nodes, and try to find the optimal distribution with a respect to a cost function. I this case our cost function will be performance, i.e. the minimal average execution time of the system.

In our model we take some assumptions, which might be a simplification, but which still may lead towards a local optimization of software allocation. We assume (a) the system is built of set of components that are characterized as computational and deployable units (i.e a component cannot be distributed over several nodes), (b) components have the property of *isomorphism*, i.e. each component can be deployed on every computing unit. However we can provide additional constraints by which one can specify whether a particular component is supposed to be (or not to be) mapped to a particular node.

The more formal specification that will enable the automatic allocation computation is the following.

First, we define a set of resources that each computing unit can provide (e.g. total execution time, static memory, dynamic memory, etc.):

Definition 1*: $\mathcal{R} = [r_{ij}]_{m \times l}$ is a computing unit resource matrix where r_{ij} represents j-th resource of a i-th computing unit.*

Each component will be defined by a set of properties, where each property can have a different value (i.e. different resource demand) on different node. Therefore we define a 3-dimensional matrix with information about resource demand for every component allocated on every computing unit:

Definition 2: *With n software components, m computing units and l different resources, $\mathcal{T} = [t_{ijd}]_{n \times m \times l}$ is a resource consumption matrix where t_{ijd} represents the amount of d-th resource of the i-th software component allocated on the j-th computing unit.*

In addition to the resource demands, we also need information about communication channel cost between computing units. This is because the computing units are heterogeneous, so they vary in clock frequency, synchronization, performance, processing type etc, and they can be connected via different type of communication channels (e.g. Ethernet, CAN-bus, Wi-Fi, etc). [10]. To specify this communication cost we define the matrix $\mathcal{C}$:

Definition 3: *$\mathcal{C} = [c_{ij}]_{m \times m}$ is a platform communication cost matrix where c_{ij} represents a communication cost between computing i-th and j-th computing unit. For $i = j$, $c_{ij} = 0$.*

The total communication cost between the components does not only depend on the

characteristics of the communication channels between the platforms, but also on the communication between components defined by their internal logic; components which communicate intensively between computing units with high communication cost will have a larger impact on overall performance than those communicating sporadically with less data exchange. To express this dependency we define the communication intensity matrix:

Definition 4: $\mathcal{K} = [k_{ij}]_{n\times n}$ *is a communication intensity matrix where* k_{ij} *represents a communication intensity between* i*-th and* j*-th software component.*

In Definition 4, if components i and j are not communicating then $k_{ij} = 0$, also notice that $\mathcal{K}$ is symmetric.

Finally, we need to define a set of all the possible allocations of components to computing units.

Definition 5: *Let* $p_i = (p_1, \dots, p_n) \in \mathcal{P}$ *be one allocation from S to H, with* $i = 1, \dots, m^n$.

Now we have all the necessary elements required to find the optimal component allocation, i.e. all possible allocations, component resource demand and communication cost. We define the cost function w that evaluates one allocation p.

$$w = \left(\sum_{d=1}^{l} f_d \sum_{i=1}^{n} t_{ip_id} + f_c \sum_{i\le j} k_{ij} \cdot c_{p_ip_j} \right) \cdot a$$

where $a = \begin{cases} 0, \sum_{i=1}^{n}\sum_{d=1}^{l}(t_{ip_id}) < \sum_{j=1}^{l} r_{p_ij} \\ 1, \sum_{i=1}^{n}\sum_{d=1}^{l}(t_{ip_id}) \ge \sum_{j=1}^{l} r_{p_ij} \end{cases}$

The cost function w consists of two main sums; the first one considers computing unit resource constraints and the second one considers the communication cost between allocated components. Notice that in first sum we defined a vector $F = [f_i]_l$. It is used as a tradeoff weight factor by which we define which resource is of greater importance for the application. For instance, in our consideration the execution time is more important than the memory allocation, it would get a higher value of f. The factor f_c denotes the weight factor for the node intercommunication cost.

Parameter a is used to verify whether the particular allocation is allowed. The solution is not allowed if the current allocation does not have sufficient resources (the sum of demanded resources exceeds the available resources).

To find the optimal software allocation we need to compare all the solutions from the set $\mathcal{P}$ and select the one with the smallest w (greater than 0). However, a problem emerges from the size of $|\mathcal{P}| = m^n$ which is not searchable in a polynomial time.

3. Evolving Allocation with Genetic Algorithm

We are dealing an optimization problem where the goal is to minimize the cost function with large number of variables, in this case w. Since it is not feasible to search the entire solution space we propose using a heuristic (e.g. greedy algorithm) or metaheuristic (e.g. genetic algorithm, particle swarm optimization, simulated annealing) which will find a semi-optimal solution.

Genetic Algorithm (GA) comes from the field of artificial intelligence and it is used for optimization problems by mimicking the process of evolution. The evolution of solution is done by crossover between different solutions (genes) and mutation. Bad solutions (genes) die out and good solutions are reinforced. The solutions need to be comparable, and since we have a w function, GA is a good starting point. For our implementation we used *Python* and *Pyevolve* library with the following settings:

Table 1: GA settings, the execution environment was a server ran by Intel Xeon E7-4830 and 8GB of memory

Generations	50
Mutation rate	0,05
Crossover rate	0,95
Population size	80
Selection algorithm	Roulette wheel

With settings from table 1, the process of evolving the solution converges to the exact solution with average deviation of 3%. The exact solution was calculated by brute force (BF) algorithm which tested all possible allocations. Figure 2 shows the comparison between results of BF and GA, and confirms that the GA is a good method for solving the allocation problem. During the test of GA we also measured execution time. Figure 3 shows that initially BF was faster, however with slow growth of inputs search space and time grow exponentially and GA was more efficient as expected.

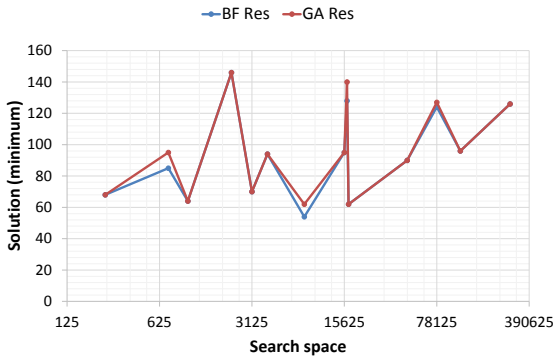

Figure 2: Comparison of solutions from genetic algorithm and brute force search (logarithmic scale)

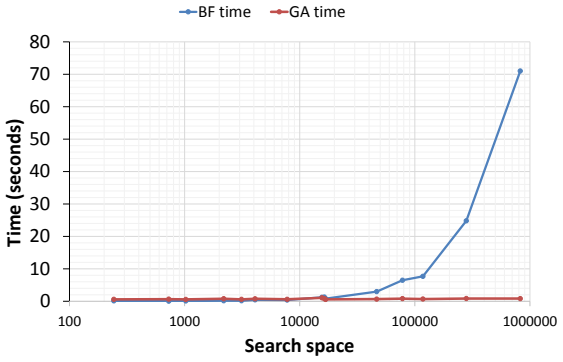

Figure 3: Execution time comparison (logarithmic scale)

There are two issues with generating solutions. The first one concerns same platform solutions where all software components get allocated to only one processing unit. This is in practice solved as usually some of the components are pre-allocated on particular node, and some of components cannot be executed on a particular node. The second issue concerns the optimal solution. GA provides one solution which is not necessary the optimal (in difference to the BF solution). Anyhow, each GA solution provides a solution that is sufficiently good. A multiple execution of GA would give a possibility to get different solutions and select between the optimal one.

4. Example setting

The example on which we demonstrate the approach is the software model of an autonomous underwater robot (AUR) that is being developed in RALF3 project [2], and that competes on the annual RoboSub contest (AUVSI Fundation) [3]. One of the challenges in the competition is recognition of particular objects while interacting with them in a real-time. A high and accurate performance is the key challenge in the competition. To meet these challenges a best utilization of a heterogeneous platform is needed. The system uses multicore CPU, GPU, two FPGA units. We present a simplified software and hardware architecture to perform the allocation of software components.

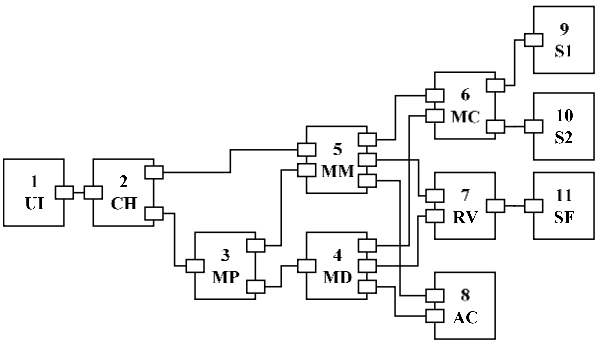

Figure 4: Simplified software architecture

Figure 4 shows the software architecture. It consists of 11 following components; (1-UI) user interface used for manual control and displaying data from sensors and camera, (2-CH) communication handler is used to handle the communication between the user interface and the data from sensory, (3-MP) message parser is used to translate internal communication e.g. to convert user input into the commands for movement, (4-MD) manual drive enables driving the robot and communicating with movement, vision and actuators, (5-MM) mission manager is used for autonomous mode and it contains the behavior model, (6-MC) movement control is used on the low level to translate high level commands to low level commands for motors, (7-RV) rudimentary vision for object recognition, (8-AC) actuator control, (9-S1) sensor set 1 (sonar, orientation), (10-S2) sensor set 2 (depth), (11-SF) video stream filtering form camera.

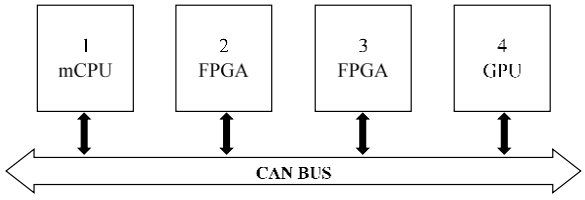

Figure 5: Simplified hardware architecture

Figure 5 shows hardware architecture with 4 computing units; (1) Multicore CPU which is intended to handle logics of the robot, (2, 3) FPGA is intended for use in camera image processing, (4) GPU is intended for object recognition.

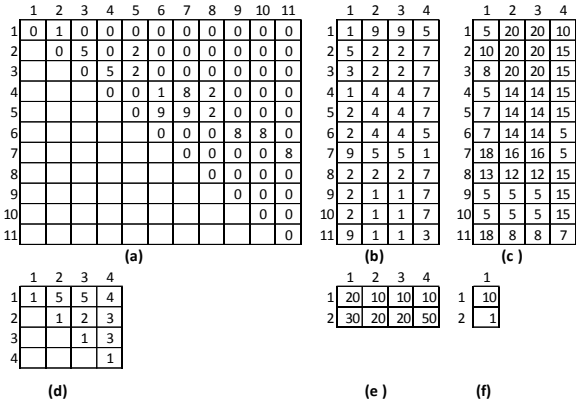

Figure 6: The input matrices

To find the solution we need matrices $\mathcal{K}$, $\mathcal{C}$ and $\mathcal{T}$. Since this is currently a work in progress, we do not yet have access to actual parameters, so for this purpose we will make an assumption about them, while keeping the proportions realistic. In Figure 6, (a) is the component communication matrix $\mathcal{K}$. Numbers from the matrix caption row and column represent the components from Figure 4. Since the resource consumption matrix is three dimensional (components, computing units, resources) we used two matrices to display two different resources (3rd dimension); (b) processing time and (c) memory. Matrix (d) is the platform communication matrix $\mathcal{C}$, (e) is the resource availability matrix $\mathcal{R}$, and (f) is the trade-off vector by which we indicate that for this calculation processing time is ten times more important than memory.

Table 2 shows the results from five subsequent GA executions on the input data from the Figure 6. There are some small deviations in different solutions and the best ones are from the first and second execution. Notice that in certain solutions some computing units are not even selected for the deployment of the components. Depending on the preference you can choose the solution. This preference should consider (a) which components should be together, (b) which components should not be together and (c) which component must be on a particular computing unit, (d) development time. Since we want to use all the computing units and the *vision component* on the GPU, we select the solution 1.

From matrix $\mathcal{T}$ you can notice that most of the components have a good execution time for FPGA unit (column 2, 3), so as expected, most of the software gets allocated to those units. In the next segment we will present the related work, after which we will discuss the results of this example.

Table 2: The results of multiple execution of GA (score: less is better)

	1	2	3	4	5
1-UI	mCPU	mCPU	mCPU	mCPU	mCPU
2- CH	FPGA 1	FPGA 1	FPGA 2	FPGA 1	FPGA 1
3-MP	FPGA 2	FPGA 2	FPGA 1	FPGA 2	FPGA 2
4-MD	mCPU	mCPU	mCPU	FPGA 1	mCPU
5-MC	FPGA 1	FPGA 1	FPGA 2	FPGA 1	FPGA 2
6-MV	FPGA 2	FPGA 2	FPGA 1	FPGA 2	FPGA 1
7-RV	GPU	GPU	GPU	FPGA 2	FPGA 1
8-AC	mCPU	mCPU	FPGA 1	FPGA 2	FPGA 2
9-SS1	FPGA 1	FPGA 1	FPGA 2	FPGA 1	FPGA 2
10-SS2	FPGA 1	FPGA 1	FPGA 2	FPGA 1	FPGA 2
11-SF	FPGA 2	FPGA 1	FPGA 1	FPGA 1	FPGA 2
SCORE	526	526	519	512	527

5. Related work

There are a lot of component-oriented frameworks for modeling the software architecture listed in [4], that enable reasoning about extra-functional properties (e.g Palladio component model and performance [5], or ProCom component model and worse-case execution time [6] where software components are allocated on virtual nodes, and later those virtual nodes to physical nodes [7], or and in some cases managing deployment, but without optimization [8]). A trade-off analysis of utilization of different resources in real-time system is discussed in [9]. However, not a lot of work addresses component-oriented frameworks targeted for heterogeneous platform, and specifically allocating software components to heterogeneous computing units. Several works relate to tasks allocation to different processing units with some resource constraints and to searching for an optimal load balancing across the system [10],[11] or a good average-case performance [12], but they do not address heterogeneous platforms. The second group relates to frameworks where software component allocation is part of the deployment process. Problems related to heterogeneous platforms and challenges in components synchronization between the platforms is described in [13]. In [14], a dynamic reallocation is enabled in combination with performance monitoring.

Our method enables efficient placement of software components on computing units of a heterogeneous platform. It considers multiple criteria and results in semi-optimal software allocation.

6. Discussion and future work

In this paper we have presented a model for optimization of component allocation to heterogeneous embedded platform. Our solution provides a semi-optimal allocation model which uses genetic algorithm. Although the presented model provides a good theoretical basis, it still needs further refinement due to some initial assumptions, e.g. deriving input parameters.

The resource consumption matrix can be acquired by measurements, calculation or empirically. For instance the execution time can be measured as a time which passes from the moment when the input signal arrives to the component until the output signal exits the component (i.e. for non-preemptive scheduling) and the task is finished with execution (preemptive scheduling).

Communication intensity can be measured by the number of function calls between two components, or the data type (e.g. data stream vs. signal data), and some values can be derived by estimation, e.g. component performance.

One also must consider non-functional constraints, e.g. development effort. As shown in Table 2, fourth allocation is the best one. Most of the software components get to be deployed on FPGA since it will offer the best performance, however, in practice this is not a realistic due to development efforts. Our choice would be first or the second allocation, for it uses more platforms. To manage this we can define a new "property" that identifies development cost of each component for particular platform. This will be addressed in the future work. The other possible improvement is a more general specification of different constrains that have impact on the component allocations, for example some pre-defined allocation definitions or mutual conditions for the component allocations. A next refinement step is analysis based on usage scenario and consequently defined the best allocation in respect to usage scenarios.

Acknowledgements

This work was partially supported by the Swedish Foundation for Strategic Research via project RALF3.

References

[1] P. Liggesmeyer, "Trends in Embedded Software Engineering," IEEE Software, 26:3, 2009.

[2] RALF3 project [Accessed Jan 2013], "http://www.mrtc.mdh.se/projects/ralf3/"

[3] AUVSI Foundation, [Accessed Jan 2013]. "http://www.auvsifoundation.org/foundation/competitions/robosub/,"

[4] Ivica Crnkovic, Séverine Sentilles, Aneta Vulgarakis, Michel R. V. Chaudron: A Classification Framework for Software Component Models. IEEE Trans. Software Eng. 37(5): 593-615 (2011)

[5] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the Palladio component model," the 6th international workshop on Software and performance, 2007.

[6] Autosar D. Partnership, 1 2013. [Online]. Available: http://www.autosar.org/.

[7] J. Carlson, J. Feljan, J. Mäki-Turja and M. Sjödin, "Deployment Modelling and Synthesis in a Component Model for Distributed Embedded Systems," 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) , pp. 74-82 , 2010.

[8] S Sentilles, A Vulgarakis, T Bureš, J Carlson, I Crnkovic, A component model for control-intensive distributed embedded systems Component-Based Software Engineering, pp. 310-317, 2008

[9] Johan Fredriksson, "Optimizing Resource Usage in Component-Based Real-Time Systems," CBSE'05 Proceedings of the 8th international conference on Component-Based Software Engineering, pp. 49-65, 2005.

[10] B. L. T. Ristau and G. Fettweis, "A Mapping Framework for Guided Design Space Exploration of Heterogeneous MP-SoCs," Design, Automation and Test in Europe, pp. 780-783 , 2008.

[11] S. Wang, J. Merrick and K. Shin, "Component allocation with multiple resource constraints for large embedded real-time software design," 10th IEEE Real-Time and Embedded Technology and Applications Symposium , p. 2004, 219-226 .

[12] J. Feljan, J. Carlson and T. Seceleanu, "Towards a model-based approach for allocating tasks to multicore processors," 38th EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 117-124 , 2012.

[13] Benaoumeur Senouci, "Multi-CPU/FPGA Platform Based Heterogeneous Multiprocessor Prototyping: New Challenges for Embedded Software Designers," The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, pp. 41-47, 2008.

[14] Malek, S., Medvidovic, N. and Mikic-Rakic, M.. "An Extensible Framework for Improving a Distributed Software System's Deployment Architecture." IEEE Transactions on Software Engineering, Vol. 38, No. 1, pp 73-100, Jan/Feb2012.