Journal of Computing and Information Technology - CIT 21, 2013, 4, 211–222
doi:10.2498/cit.1002284

An Extended Model for Multi-Criteria Software Component Allocation on a Heterogeneous Embedded Platform

Ivan Švogor[1], Ivica Crnković[2] and Neven Vrček[1]

[1] Faculty of Organization and Informatics, University of Zagreb, Croatia
[2] School of Innovation, Design and Engineering, Mälardalen University, Sweden

A recent development of heterogeneous platforms (i.e. those containing different types of computational units such as multicore CPUs, GPUs, and FPGAs) has enabled significant improvements in performance for real-time data processing. This potential, however, is still not fully utilized due to the lack of methods for optimal configuration of software; the allocation of different software components to different computational 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. With respect to system and user defined constraints, in this paper we are applying analytical hierarchical process and a genetic algorithm to find feasible, locally optimal solution for allocating software components to computational units.

Keywords: software, component, allocation, genetic algorithm, AHP, optimization, model, multi-criteria, embedded system

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 to a particular type of computation. Using different computational units 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 amount of data streams in real time. This great potential for increased performance is still not fully utilized due to the lack of software methods for an efficient and optimal placement (allocation) of software to the heterogeneous platform by which an optimal, or sufficiently good performance is obtained. Different software allocations result with different performance and it is not obvious which allocation would enable the best performance. In addition, best allocation candidate 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 restricted energy consumption), 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 computational unit). A "trial and error" method by repeated allocations and then measurements is an inefficient procedure, in particular when the software implementation may depend on which computational unit type will be executed. For this reason, allocation method is desired in an early development phase.

The main goal of this paper is to define a model for the software allocation optimization on computational units in heterogeneous systems according to system constraints and user provided architectural decisions, i.e. constraints.

We assume that we are dealing with component-based systems, so each software component[1] can be allocated to a computational unit. We define a method for finding an optimal allocation of components in respect to optimal performance and different constraints. The method is assumed to 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 their models. In the latter case the components may not be yet implemented, but are specified with a set of attributes, estimated or obtained in a certain way. Current model abstraction level does not consider dependencies which might arise from operating system level. Also, at this stage we are considering cross platform communication, i.e. that between components allocated on different computational units. Internal communication between cores in multicore CPUs is not so influential on the rest of the system (more on this can be found in [18]). Similar assumption applies to GPU. Therefore, we focus only on components and their interaction, the result of the method is a proposed system 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 using genetic algorithm. Chapter 4 describes method for finding optimal solution. Chapter 5 illustrates the model on an Autonomous Underwater Vehicle Case Study. Chapter 6 briefly discusses the related work, and finally Chapter 7 concludes the paper.

2. Component Allocation Method

We define a model which consists of a Software System S as a set of components, and a Hardware Platform H as a set of computational units, as shown in Figure 1.

Every component $s_i \in S, i = 0, \ldots, n$ needs to be allocated to a computational unit $h_i \in H, i = 0, \ldots, m$. From a mathematical viewpoint, the problem is reduced to finding a permutation (with repetition) which provides the best performance considering a set of defined resources and constraints of a particular system. The number of all possible solutions, i.e. allocations of components to computational units is m^n. Obviously, the search space increases rapidly with the number of the components and computational units, so to find the optimal component allocation with 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 respect to a particular set of system and component properties for given characteristics of the software system, components and the hardware platform.

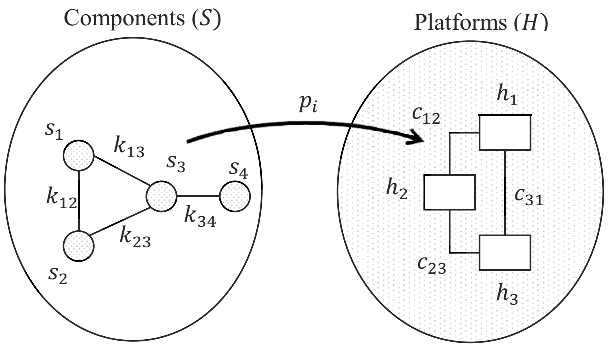


Figure 1. Software components allocation to heterogeneous hardware platform.

Our model is defined as follows.

1. A computational unit 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 computational unit must satisfy the needs of the components executed on that unit.
2. We enable allocation of the components to the computational units, and try to find their optimal distribution with respect to cost function. In this case our cost function is the overall performance of a certain allocation.

The model also takes some assumptions:

1. The system is built of a set of components which are characterized as atomic computational and deployable units (i.e. a component cannot be distributed over several computational units).

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

2. Component allocation has the property of isomorphism (i.e. each component can be deployed on every computational unit).
3. The system can be exposed by a set of constraints, which specifies whether a particular component is supposed to be (or not to be) mapped to a particular computational unit.

2.1. Elements of the model – definitions

This section presents a formal specification of information necessary for making a component allocation decision.

A component can be allocated on any computational unit (if not explicitly defined otherwise). Each component requires a certain amount of resources that should be provided by the computational unit. This amount depends on the computational unit on which the component is deployed. To specify the resources required for each component on each computational unit, we define a 3-dimensional matrix called Resource Consumption Matrix:

Definition 1. *For n software components, m computational units, and l different resources,* $\mathcal{T} = [t_{ijk}]_{(n\times m\times l)}$ *is a Resource Consumption Matrix where* t_{ijk} *represents the necessary amount of k-th resource of the i-th software component allocated on the j-th computational unit.*

With Resource Consumption Matrix we know the amount (quantity) of resources necessary for each component on each of the platforms. In addition, we need to define a new matrix which will specify a set of resources each computational unit can provide (e.g. total execution time, static memory, dynamic memory, energy etc.).

Since there are m computational units, and the Resource Consumption Matrix defines l resources, the size of the new matrix, the computational Unit Resource Matrix, is $m \times l$.

Definition 2. $\mathcal{R} = [r_{jk}]_{(m\times l)}$ *is a Computational Unit Resource Matrix where* r_{jk} *represents k-th resource of a j-th computational unit.*

With matrices $\mathcal{T}$ and $\mathcal{R}$ defined, we know how much of some resource a certain component needs while allocated to any computational unit. Still, the provided information is insufficient to make a decision about the component allocation since we also need to consider communication. It can be realized between components within the same computational unit, or among different computational units.

We recognize two different communication aspects; software and hardware. From the software viewpoint we can talk about the intensity of communication between components. For instance, components handling data acquisition and data processing would have larger communication intensity than a component which is responsible for task delegations.

We define a new matrix which contains information about the communication channel cost between computational units. This is due to heterogeneous computational units, which can be connected via different types of communication channels (e.g. Ethernet, CAN-bus, Wi-Fi, etc.) [10]. To specify the communication cost, we define the Platform Communication Cost Matrix:

Definition 3. $\mathcal{C} = [c_{ij}]_{(m\times m)}$ *is a Platform Communication Cost Matrix where* c_{ij} *represents a communication cost between i-th and j-th computational unit. For* $i = j$, $c_{ij} = 0$.

The communication is also limited by a physical constraint – the bandwidth, which must be taken into consideration.

Definition 4. $\mathcal{B} = [b_{ij}]_{m\times m}$ *is a Bandwidth Matrix where* b_{ij} *represents communication bandwidth available between i-th and j-th computational unit.*

Cost of the total communication 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 components behavior; components which communicate intensively between computational units with high communication cost will have a larger impact on overall performance than those communicating sporadically with less data exchange. To express this constraint, we define the communication intensity matrix:

Definition 5. $\mathcal{K} = [k_{ij}]_{(n\times n)}$ *is a Communication Intensity Matrix where* k_{ij} *represents a communication intensity between i-th and j-th software component.*

If components i and j are not communicating, then $k_{ij} = 0$; also notice that $\mathcal{K}$ is symmetric so the direction of the communication is irrelevant at this point. Although we are not limiting the user on how to quantify $\mathcal{K}$ in Section 4, we provide a suggestion.

By the definitions (1-5) all necessary resources and constraints are defined. In order to find an allocation which satisfies the criteria of a particular system, one should evaluate all allocations (i.e. all permutations) and find the allocation which provides desired performance. The set of all possible allocations (of components to computational units) is defined as follows:

Definition 6. *Function $p_i : S \rightarrow H$ is a Component Mapping Function where $p_i = (p_1, \ldots, p_n) \in \mathcal{P}$ defines a particular allocation of components from S to computational units from H, with $i = 1, \ldots, m^n$.*

In order to make different allocations, i.e. solution vectors comparable and to get the optimal allocation with respect to the overall system performance, we need a mathematical model to evaluate every solution – a cost function, which is the subject of the following section.

Elements of vector p_i represent one mapping of a component to the computational units. The example below shows the case where number of components is equal to number of computational units ($n = m$), and every software component is allocated on only one computational unit. The *position* in solution vector represents a component and its *value* represents the computational unit on witch it is allocated. In real world, it is more often that there are more components than computational units, and therefore some computational units h will occur on several positions across the vector p_i.

$$\begin{array}{rcccc} & h_1, & h_2, & \ldots, & h_m \\ & \uparrow & \uparrow & & \uparrow \\ p_i = & (s_1, & s_2, & \ldots, & s_n) \end{array} \tag{1}$$

2.2. The allocation cost function

Now we will define a mathematical model in the form of a cost function which evaluates an allocation. Therefore, providing a normalized value of resource usage costs. In addition, the predefined constraints could exclude some particular allocations. With the ability to compare allocations p_i, one can find the optimal one. In order to create a cost function, we must consider: a) different system resources, b) communication, which is further explained in the following subsections.

System Resource Constraints

System Resource Constraints function, *res* sums up normalized values of all resources used by the components for a particular allocation configuration p_i, and it is defined as follows:

$$res = \sum_{k=1}^{l} f_k \sum_{i=1}^{n} t_{ip_ik} \tag{2}$$

where
f – element of a trade-off vector F
t – element of a resource consumption matrix $\mathcal{T}$
l – number of different resources for matrix $\mathcal{T}$
n – number of components

Equation 2 consists of two sums. The inner one calculates total costs of each resource for all components. There are l different resources and they are marked with k. The outer one sums up all l resources. Summing different costs is possible due to normalization of resources values (more on this topic in Section 2.3). In addition, for a particular system, particular properties may be of a larger importance (for example, energy consumption can be more important than system reliability: A solution to put two components on one computational unit can be better for energy consumption, but for reliability it can be better to allocate these two components on different computational units). For this reason we introduced a trade-off vector $F = [f]_{l+1}$ which defines relative importance of each resource cost.

Notice that vector is $l + 1$ long, where l is the number of resources considered. First l elements of the vector are marked as f_k, and they are used for giving importance to resources defined with matrix $\mathcal{T}$. $(l+1)$-th element, marked as f_c, is used for importance of communication in the following section (Equation 4).

Resource constraints

Equation 2 provides a resource cost for any configuration p_i. However, there are configurations which are not feasible. Such configurations exceed the resource demand which computing units can provide. For this reason we introduce a

Resource constraint factor $\rho = 1$, which defines the feasibility of an allocation configuration.

$$\rho = \begin{cases} 0, \sum_{i=1}^{n} \sum_{k=1}^{l} \left(t_{ip_ik}\right) < \sum_{j=1}^{l} r_{p_ij} \\ 1, \sum_{i=1}^{n} \sum_{k=1}^{1} \left(t_{ip_ik}\right) \geq \sum_{j=1}^{l} r_{p_ij} \end{cases}$$

where
r – element of the computational unit resource matrix $\mathcal{R}$.

The multiplier ρ is used to remove unfeasible configurations; If the required resources exceed maximum available resources on a particular computational unit, we set $\rho = 0$, effectively disregarding the current allocation. In all other cases, $\rho = 1$, meaning the available resources are not exceeded by the required resources in the current allocation p_i.

Now Equation 2 multiplied by the factor ρ gives a cost for a feasible allocation configuration.

$$res = \left(\sum_{k=1}^{l} f_k \sum_{i=1}^{n} t_{ip_ik} \right) \cdot \rho \tag{3}$$

Communication resources and constraints

In addition to the computational unit resources that have impact on the performance, we have resources related to the communication between the components. The communication costs are expressed by *com*:

$$com = \sum_{i \leq j} k_{ij} \cdot c_{p_ip_j} \tag{4}$$

where
k – element of a communication intensity matrix $\mathcal{K}$
c – element of a platform communication cost matrix $\mathcal{C}$
b – element of the bandwidth matrix $\mathcal{B}$

The sum in Equation 4 handles communication channels between computational units and multiplies the platform communication cost (defined by matrix $\mathcal{C}$) with the communication intensity (defined by matrix $\mathcal{K}$). Communication intensity relates to the data exchanged between different software components, while the platform communication cost relates to physical characteristics. This will vary depending on which computational unit a component is allocated.

Like before, we define a multiplier κ that prevents an allocation in which the required communication resources exceed the available bandwidth.

$$\kappa = \begin{cases} 0, \sum_{i \leq j} \left(k_{ij} \cdot c_{p_ip_j}\right) < \sum_{i \leq j} b_{p_ij} \\ 1, \sum_{i \leq j} \left(k_{ij} \cdot c_{p_ip_j}\right) \geq \sum_{i \leq j} b_{p_ij} \end{cases}$$

where b – element of the bandwidth matrix $\mathcal{B}$.

The final form of the communication resource cost function is defined as follows:

$$com = \left(f_c \sum_{i \leq j} k_{ij} \cdot c_{p_ip_j} \right) \cdot \kappa \tag{5}$$

where
f_c – the communication trade-off factor (the last element in vector F).

Final form of the cost function

In order to get the complete model, Equation 3 and Equation 5 need to be joined in a cost function $w = (res + com) \cdot \rho \cdot \kappa$:

$$w = \left(\sum_{k=1}^{l} f_k \sum_{i=1}^{n} t_{ip_ik} + f_c \sum_{i \leq j} k_{ij} \cdot c_{p_ip_j} \right) \cdot \rho \cdot \kappa \tag{6}$$

To find the optimal component allocation, we need to compare all feasible solutions p_i from $\mathcal{P}$ and select the one with the smallest w (greater than 0).

As already mentioned, the problem is in the size of $|\mathcal{P}| = m^n$ which is a very large number. Small increases in sets S and H cause enlargement of search space, which is not searchable in a polynomial time.

2.3. Handling different measurement units

With final form of the evaluation model, one can compare different allocations using the cost function. However, there is one more problem to address and it is related to different measurement units, e.g. execution time is expressed in milliseconds, memory in megabytes, energy consumption in watts per hour, etc. To solve

$$\begin{array}{c} \\ k_1 \\ k_2 \\ \vdots \\ k_{l+1} \end{array} \begin{array}{c} \begin{array}{cccc} k_1 & k_2 & \dots & k_{l+1} \end{array} \\ \left(\begin{array}{cccc} 1 & m_{1,2} & \dots & m_{1,l+1} \\ (m_{1,2})^{-1} & 1 & \dots & m_{2,l+1} \\ \vdots & \vdots & \ddots & \vdots \\ (m_{1,l+1})^{-1} & (m_{2,l+1})^{-1} & \dots & 1 \end{array} \right) = M_c \end{array} \tag{7}$$

this problem, we use Analytic Hierarchy Process (AHP) [15], a commonly used method for complex multidimensional choices, alternatives and tradeoffs.

A great benefit of this method is disregarding measurement units. It uses trade-off weights which assess the importance of different decision criteria. Since it also uses human subjective judgment, AHP provides a procedure to verify consistency of the decision model.

For the model we present here, one level of hierarchy (Figure 2) is sufficient and it considers the importance of different resources. For importance of each resource, a trade-off vector F is used as shown in Equation 6. Its values should be calculated using AHP. To determine F, we first need to address the importance of all different resources, i.e. AHP criteria used in the allocation model. AHP criteria are given by the third dimension of matrix $\mathcal{T}$.

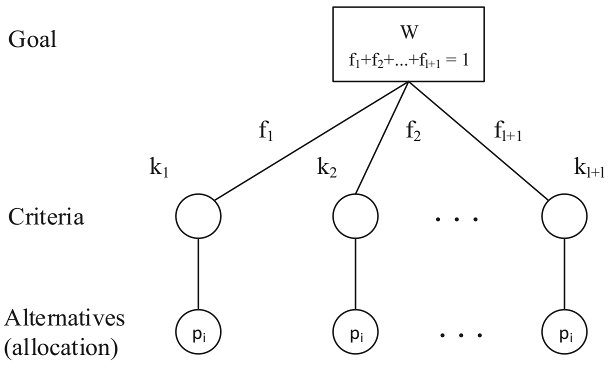


Figure 2. Hierarchy for defining the criteria.

The first step in AHP is to perform a pairwise comparison of all AHP criteria and form the comparison matrix M_c along with hierarchy of the criteria. This is shown in Figure 2. The solution vector p_i, is an input for each AHP criterion.

Matrix M_c provides a pairwise comparison of resources considered by the model. For comparison, we are using a standard AHP scale *(1 – equal importance, 3 – slightly favoring, 5 – strong favors, 7 – very strong favoring, 9 – extreme favors).* For instance, if resource k_1 is slightly more important than resource k_2, consequently $m_{1,2} = 3$, and k_2 compared to k_1 would give the reciprocal value, $m_{2,1} = 1/3$. For $i = j$, $m = 1$.

The next step, according to AHP, is to calculate normalized principal Eigenvector and principal Eigenvalue. The largest Eigenvalue is called principal Eigenvalue (λ_{max}). The Eigenvector that corresponds to principal Eigenvalue is called Principal Eigenvector (ω^*). The Principal Eigenvector should also be normalized so that sum of all elements equals 1, and its values are vector F.

The final step, before attributing the Eigenvector with the created hierarchy is to verify consistency of the model. Since pairwise comparison matrix is created by a subjective human judgment, AHP deals with this by measuring the consistency of prioritization. As it may happen that pairwise comparison is not done in a consistent way, consistency ratio C_R should be calculated. It is given as $C_R = C_I/R_I$, where C_I is Consistency index and R_I is Random consistency index.

C_I is calculated as:

$$C_I = \frac{\lambda_{max} - (l+1)}{l}$$

Random consistency index is calculated by generating comparison matrix with values $(1/9, 1/8, \dots, 8, 9)$, more on this topic can be found in [16].

Finally, if C_R is less or equal to 10%, the inconsistency of pairwise comparison is acceptable.

Since different system properties (matrix $\mathcal{T}$, $k - th$ dimension – see Figure 2) are measured using different units, it is likely that the values of those properties will have different orders of magnitude (e.g. tens of milliseconds, thousands of megabytes or hundredths of milliampers per hour). Hence, before the calculation, the input matrices $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{C}$ need to be normalized, so all the values are in range from 0 to 1. 1 represents a maximum available amount of a resource; it is found in matrix $\mathcal{R}$. Now, the only factor to decide the importance of a certain system property is the trade-off vector F.

2.4. Additional constraints – architectural decisions

To get the optimal solution, we need to find p_i which gives minimum result $min(w)\forall p_i \in \mathcal{P}$ (Equation 6) considering all the given platform constraints. As opposed to platform constraints, system architectural decisions can also be considered as a constraint, which somewhat decreases the search space. The solution vector p_i is becoming partially defined $p'_i \in \mathcal{P}'$, where $\mathcal{P}' \subset \mathcal{P}$ is the reduced search space with valid solutions. Architectural decisions are included in our allocation model as an additional constraint.

There are two architectural decisions (additional constraints) which can be specified:

1. a particular component *should be* or *should not be* allocated on a particular computational unit
2. a set of components *should be* or *should not be* allocated on the same computational unit

3. Evolving the Solution with Genetic Algorithm

In order to get an optimal solution, the goal is to minimize the cost function (w). Since it is not feasible to search the entire solution space (due to large number of variables), a heuristic (e.g. greedy algorithm) or meta-heuristic (e.g. genetic algorithm, particle swarm optimization, simulated annealing) is a good choice to find a semi-optimal solution. We have chosen Genetic Algorithm (GA) which is frequently used for solving optimization problems by mimicking the process of evolution. The evolution of the solution is done by crossover between different solutions, in our case allocations (genes) and mutation. Bad allocations are disregarded and the good ones are reinforced. Since GA needs a comparison function, and w can be used as one, GA is a favorable choice. Also, our comparisons have shown that it provides very accurate solutions Figure 3. For the implementation we used *Python* and *Pyevolve* library with the following settings:

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

Table 1. GA settings.

With the settings shown in Table 1, GA converges to the solution with average deviation of 3%. The exact solution was calculated by exhaustive search (ES), which is a common technique to systematically enumerate all solutions. Figure 3 shows the comparison between results of ES and GA. It is obvious that GA provides very accurate results. During the test of GA we also measured the execution time. Figure 4 shows that initially ES was faster, however with slow growth of inputs search space and time grew exponentially and GA was more efficient, as expected[2].

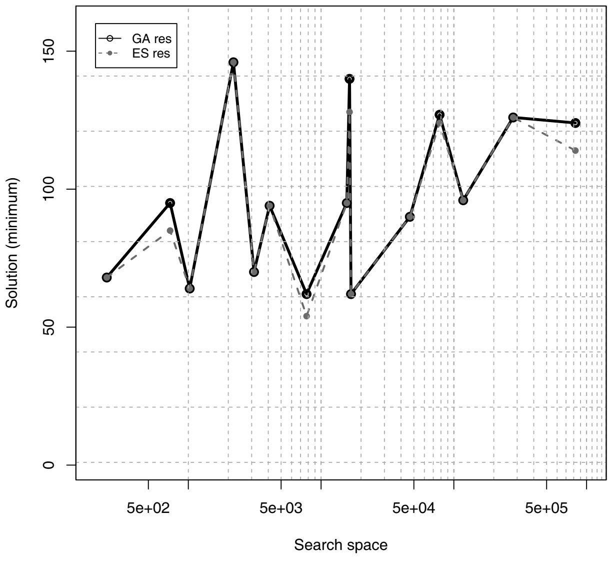


Figure 3. Accuracy comparison between solutions from GA and EF (logarithmic scale).

[2] GA is executed on a server with Intel Xeon E7-4830 CPU and 8GB of RAM.

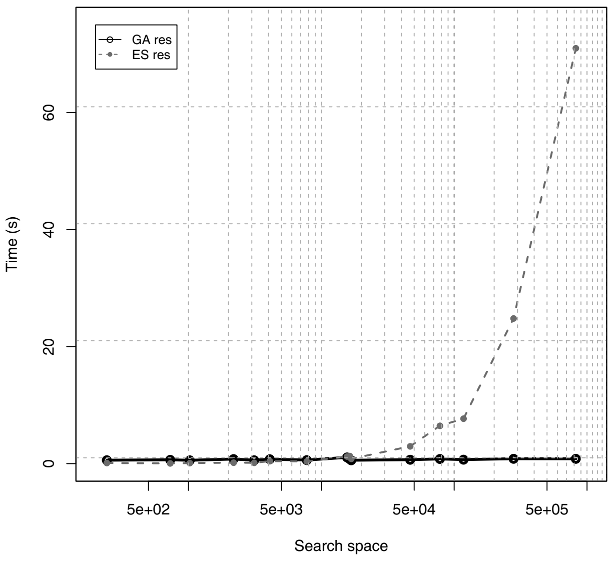


Figure 4. Execution time comparison for GA and EF search(logarithmic scale).

The application of Genetic Algorithm does not guarantee an optimal solution, however it will provide one which is sufficiently good, given the time necessary to calculate it. Here, we used GA simply to gain the solution faster, however at the cost of accuracy. For the purpose of the following use case, the accuracy was acceptable.

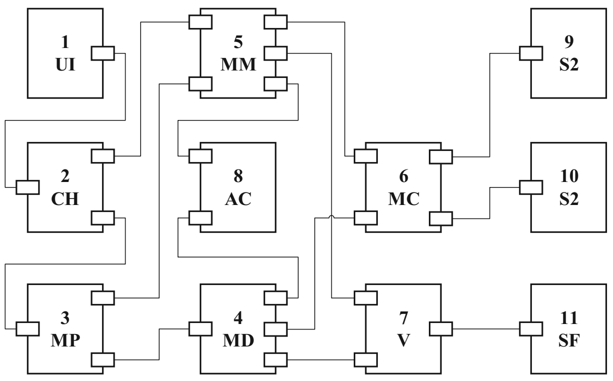


Figure 5. Simplified software architecture.

4. Case study: Underwater Autonomous Vehicle

The system on which we demonstrate our allocation model is an autonomous underwater vehicle (AUV) that is being developed as a part of RALF3 project [2]. Since 2012 it competes on annual RoboSub contest (AUVSI Foundation) [3] in San Diego, California. One of the main challenges is handling and interpreting vision data, i.e. recognition of particular objects, while simultaneously interacting with them in real time. Therefore all the components should be allocated in a way that will fully utilize the heterogeneous platform, which consists of a multicore CPU, GPU, two FPGA units. Here we present a simplified software and hardware architecture for which component allocation should be performed.

Figure 5 shows the software architecture. It consists of eleven components:

1-UI	*User interface* used for manual control and displaying data from sensors and camera.
2-CH	*Communication handler* used to handle communication between the user interface and the data from sensors.
3-MP	*Message parser* used to translate internal communication e.g. to convert user input into the commands for movement.
4-MD	*Manual drive* enables manual control of the robot and communication with movement, vision and actuators.
5-MM	*Mission manager* used for autonomous mode, it contains the behavior model.
6-MC	*Movement control* used for low to high level software communication handling.
7-V	*Vision* for recognition of basic objects and shapes.
8-AC	*Actuator control* used for controlling actuators (bite shift operations, message parsing).
9-S1	*Sensors layer 1* used for collecting the data from sensors (sonar, orientation).
10-S2	*Sensors layer 2* used for collecting the data from sensors (depth).
11-SF	*Stream filtering* handles video filtering chain (color normalization, white balance, color isolation, edge detection, etc.).

Figure 6 shows hardware architecture with 4 computational units:

1-mCPU multicore CPU

which is intended to handle top level control of the robot and sequential tasks with a few branches.

(2-FPGA, 3-FPGA) FPGA

which are intended for raw image processing and stream data processing.

(4-GPU)]

which is intended for object recognition algorithms and tasks which can be parallelized.

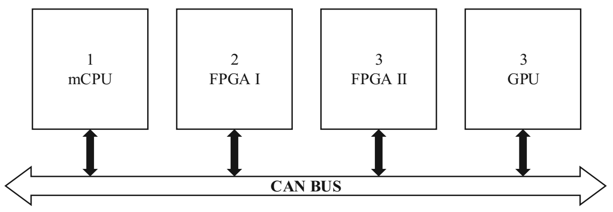


Figure 6. Simplified hardware architecture.

Finding the right allocation

To find the optimal allocation of components to the computational units we need the data for matrices $\mathcal{T}$, $\mathcal{R}$, $\mathcal{K}$, $\mathcal{B}$ and $\mathcal{C}$. Since this is currently a work in progress, we do not yet have access to all actual parameters, so for this purpose we will make an assumption about them, while keeping the proportions realistic as possible.

In Figure 7, (a) is the component communication matrix $\mathcal{K}$. The values we use are almost the same as those in standard AHP scale (the communication can be: 0 – none, 1 – low, 3 – occasional, 5 – moderate, 7 – frequent, 9 – intensive). One can get the values from the number of calls and an average data packet size or measuring data load on the channels between components. Since $\mathcal{T}$, the resource consumption matrix, is three-dimensional (components, computational units, resources), we used three tables to display three different resources (i.e. the 3rd dimension); (b) average execution time (milliseconds), (c) memory (megabytes) and (d) average energy consumption (milliamperes per hour). Matrix (e) is the platform communication matrix $\mathcal{C}$, (f) is the resource availability matrix $\mathcal{R}$, and (g) is the bandwidth between components, matrix $\mathcal{B}$. Since all computational units communicate through common CAN-bus, the bandwidth for communication between them is the same.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 5 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 5 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 1 & 7 & 3 & 0 & 0 & 0 \\ 0 & 3 & 3 & 0 & 0 & 9 & 9 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 9 & 0 & 0 & 0 & 7 & 7 & 0 \\ 0 & 0 & 0 & 7 & 9 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 3 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 10 & 90 & 90 & 55 \\ 50 & 20 & 20 & 72 \\ 30 & 20 & 20 & 72 \\ 10 & 40 & 40 & 72 \\ 20 & 40 & 40 & 72 \\ 20 & 50 & 50 & 55 \\ 90 & 20 & 20 & 15 \\ 20 & 10 & 10 & 70 \\ 20 & 10 & 10 & 70 \\ 20 & 15 & 15 & 70 \\ 90 & 10 & 10 & 33 \end{bmatrix} \quad \begin{bmatrix} 48 & 256 & 256 & 128 \\ 128 & 256 & 256 & 148 \\ 64 & 256 & 256 & 148 \\ 48 & 168 & 168 & 148 \\ 64 & 168 & 168 & 148 \\ 64 & 168 & 168 & 64 \\ 168 & 128 & 128 & 64 \\ 148 & 96 & 96 & 148 \\ 48 & 32 & 32 & 148 \\ 48 & 32 & 32 & 148 \\ 168 & 64 & 64 & 96 \end{bmatrix} \quad \begin{bmatrix} 2 & 18 & 18 & 11 \\ 10 & 4 & 4 & 14 \\ 6 & 4 & 4 & 14 \\ 2 & 8 & 8 & 14 \\ 4 & 8 & 8 & 14 \\ 4 & 10 & 10 & 11 \\ 18 & 4 & 4 & 3 \\ 4 & 2 & 2 & 14 \\ 4 & 2 & 2 & 14 \\ 4 & 3 & 3 & 14 \\ 18 & 2 & 2 & 7 \end{bmatrix}$$

(b) **(c)** **(d)**

$$\begin{bmatrix} 1 & 5 & 5 & 4 \\ 5 & 1 & 2 & 3 \\ 5 & 2 & 1 & 3 \\ 4 & 3 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} 100 & 150 & 150 & 100 \\ 256 & 640 & 640 & 256 \\ 50 & 25 & 25 & 15 \end{bmatrix} \quad \begin{bmatrix} 0 & 512 & 512 & 512 \\ 512 & 0 & 512 & 512 \\ 512 & 512 & 0 & 512 \\ 512 & 512 & 512 & 0 \end{bmatrix}$$

(e) **(f)** **(g)**

Figure 7. The input matrices.

There are also two additional architectural decisions to consider:

- Vision component (7-V) should be allocated on GPU.
- Manual drive (4-MD), mission management (5-MM) and movement control (6-MC) components should be on FPGAII.

$$\begin{bmatrix} 1 & 3 & 1/7 & 3 \\ 1/3 & 1 & 1/9 & 3 \\ 7 & 9 & 1 & 9 \\ 1/3 & 1/3 & 1/9 & 1 \end{bmatrix}$$

(i)

Figure 8. Pairwise comparison (average execution time, memory, energy, communication).

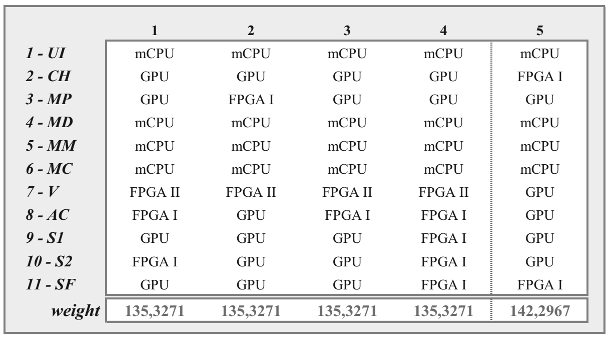
Figure 8 shows the pairwise comparison by which we determine the importance of different resources (AHP criteria, matrix M_C). The rows and columns of this matrix are: 1) average execution time importance, 2) memory importance, 3) energy consumption importance, 4) communication channel load. The importance of resources for our case study is the following (respectively): 1) energy consumption, 2) average execution time, 3) memory load, 4) communication load (also visible from F).

Calculated trade-off vector is:

$$F = (0.1557, 0.0856, 0.7095, 0.0491)$$

with the eigenvalue[3] $\lambda_{\max} = 4.2457$. Consistency ratio is $8.18\% < 10\%$, hence acceptable.

Figure 9 shows the results from five consecutive GA executions for data defined in Figure 7. There are small deviations in different solutions. The ones with lowest cost function results 1-4, the fifth result however omits the *6-MC → FPGAII* constraint to demonstrate how this change reflects on the result.



	1	2	3	4	5
1 - UI	mCPU	mCPU	mCPU	mCPU	mCPU
2 - CH	GPU	GPU	GPU	GPU	FPGA I
3 - MP	GPU	FPGA I	GPU	GPU	GPU
4 - MD	mCPU	mCPU	mCPU	mCPU	mCPU
5 - MM	mCPU	mCPU	mCPU	mCPU	mCPU
6 - MC	mCPU	mCPU	mCPU	mCPU	mCPU
7 - V	FPGA II	FPGA II	FPGA II	FPGA II	GPU
8 - AC	FPGA I	GPU	FPGA I	FPGA I	GPU
9 - S1	GPU	GPU	GPU	FPGA I	GPU
10 - S2	FPGA I	GPU	GPU	FPGA I	GPU
11 - SF	GPU	GPU	GPU	FPGA I	FPGA I
weight	135,3271	135,3271	135,3271	135,3271	142,2967

Figure 9. The results of multiple execution of GA (score: less is better).

Since the solutions 1-4 are valid, software architect can choose, according to some own preference, one of the solutions. However, he/she is certain that all of them minimize energy consumption with regard to given resources.

Also notice that user-defined constraints for components 7-V, 4-MD, 5-MM, and 6-MC are taken into account on all the solutions. So component 7-V is allocated on GPU as requested, while components 4-MD, 5-MM, 6-MC are allocated on multicore CPU.

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 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 computational 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 are described in [13]. In [14], a dynamic reallocation is enabled in combination with performance monitoring.

Our method enables efficient placement of software components on computational units of a heterogeneous platform. It considers multiple criteria which are both system-defined and architect-defined. The result is a semi-optimal component allocation for a particular system.

6. Discussion and Future Work

In this paper we presented an extension of our previous model [17] for optimization of component allocation on a heterogeneous embedded platform. We improved the model by addressing the following: a) handling different measurement units, b) handling the subjective judgment of the criteria for allocation decision with

[3] In the calculation we rounded vector F to four decimal places, and eigenvalue to thirteen. Calculations were done with NumPy and SciPy libraries.

AHP consistency index verification, c) including the bandwidth constraint for the communication and d) enabled architect defined constraints.

The solution provides a semi-optimal allocation model which uses a Genetic Algorithm (any other optimization technique can be applied) and Analytical Hierarchical Process. The improved model presented in this paper provides a strong theoretical basis, however it still needs further refinement due to some initial assumptions, e.g. deriving input parameters, which is our goal for the future work on this topic, and measuring actual data used for the model.

The resource consumption matrix $\mathcal{T}$ can be acquired by measurements, calculation or empirically. For instance, the execution time can be measured as the 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 $\mathcal{K}$ needs further discussion and research. Its intent is to envelop frequency of communication between components so one can quantify the usage of channels between them. One way to look at it is the number of function calls, channel data type i.e. signal data or streaming data, or the approach which we used in this paper: approximation with AHP scale.

One must also consider non-functional constraints, e.g. development effort. As shown in Figure 9, first four allocations provide the same result, however different allocations. In real world some components require great development efforts to be implemented on certain platforms. Partially, this issue is addressed with enabling user preference to the solution. To further address this issue, we can define a new “property” which identifies development cost of each component for particular platform and also express sequential and parallel processing needs.

Further, we also plan to provide a graphical tool with appropriate EMF model which would allow automatic component allocation in early architecture design phase. Since this is an ongoing research we will also work to improve the demonstrator (case study) and verify how operating system and other platform specific dependencies influence the architecture of components and refine the model accordingly.

7. Acknowledgment

This paper was supported by the Swedish Foundation for Strategic Research (SSF) via the project RALF3 [2] and Croatian Ministry of Science, Education and Sport via the project Information Infrastructure and Interoperability.

References

[1] P. Liggesmeyer, M. Trapp, Trends in Embedded Software Engineering. *IEEE Software*, **26**(3) (2009).

[2] Ralf3 Project Web, `http://www.mrtc.mdh.se/projects/ralf3/`, [Accessed: Jan 2013].

[3] AUVSI Fundation Web, `http://www.auvsifoundation.org/foundation/competitions/robosub/`, [Accessed: Jan 2013].

[4] I. Crnkovic, S. Sentilles, A. Vulgarakis, M. R. V. Chaudron, A Classification Framework for Software Component Models. *IEEE Transactions on Software Engineering*, **37**(5) (2011), 593–615.

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

[6] Autosar, `http://www.autosar.org/`, [Accessed: Jan 2013].

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

[8] S. Sentilles, A. Vulgarakis, T. Bureš, J. Carlson, I. Crnkovic, A component model for control-intensive distributed embedded systems. 11th International Symposium on Component-Based Software Engineering, (2008), 310–317.

[9] J. Fredriksson, K. Sandström, M. Akerholm, Optimizing Resource Usage in Component-Based Real-Time Systems. *8th International Symposium on Component-Based Software Engineering*, (2005), 49–65.

[10] B. Ristau, T. Limberg, G. Fettweis, A Mapping Framework for Guided Design Space Exploration of Heterogeneous MP-SoCs. *Design, Auto-mation and Test in Europe*, (2008), 780–783.

[11] S. WANG, J. R. MERRICK, K. G. SHIN, Component allocation with multiple resource constraints for large embedded real-time software design. *10th IEEE Real-Time and Embedded Technology and Applications Symposium*, (2004), 219–226.

[12] J. FELJAN, J. CARLSON, T. SECELEANU, Towards a model-based approach for allocating tasks to multi-core processors. *38th EUROMICRO Conference on Software Engineering and Advanced Applications*, (2012), 117–124.

[13] B. 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*, (2008), 41–47.

[14] S. MALEK, N. MEDVIDOVIC, M. MIKIC-RAKIC, An Extensible Framework for Improving a Distributed Software System's Deployment Architecture. *IEEE Transactions on Software Engineering*, **38**(1) (2012), 73–100.

[15] T. L. SAATY, *What is the Analytic Hierarchy Process?* in *Mathematical Models for Decision Support*. Springer, Berlin Heidelberg, 1988, 109–121.

[16] T. L. SAATY, *Fundamentals of Decision Making and Prority Theory with the Analytic Hierarchy Process*. Rws Publications, 1994. `http://books.google.se/books?id=nmtaAAAAYAAJ`

[17] I. ŠVOGOR, I. CRNKOVIĆ, N. VRČEK, Multi-Criteria Software Component Allocation on a Heterogeneous Platform. In *Proc. of 35th International Conference on Information Technology Interfaces*, (2013) SRCE – University of Zagreb, pp. 341–346.

[18] J. FELJAN, J. CARLSON, The Impact of Intra-core and Inter-core Task Communication on Architectural Analysis of Multicore Embedded Systems. *The Eighth International Conference on Software Engineering Advances*, (October 2013) IARIA, pp. 402–407. `http://www.es.mdh.se/publications/3021-`

Received: November, 2013
Revised: January, 2014
Accepted: Junuary, 2014

Contact addresses:

Ivan Švogor
Department of Information Systems Development
Faculty of Organization and Informatics
University of Zagreb
Croatia
e-mail: `isvogor@foi.hr`

Ivica Crnkovic
Software Engineering Division
School of Innovation, Design and Engineering
Mälardalen University
Sweden
email: `ivica.crnkovic@mdh.se`

Neven Vrček
Department of Information Systems Development
Faculty of Organization and Informatics
University of Zagreb
Croatia
email: `nvrcek@foi.hr`

IVAN ŠVOGOR is a PhD student of information science at the University of Zagreb, Faculty of Organization and Informatics, and a visiting PhD student at Mälardalen University, Sweden. His research interests include optimization of software component allocation, component based (software) engineering, heterogeneous embedded systems and robotics software.

IVICA CRNKOVIC is a professor of industrial software engineering at Mälardalen University where he is the scientific leader of the industrial software engineering research group. His research interests include component-based software engineering, software architecture, software configuration management, software development environments and tools, as well as software engineering in general. He is the author of more than 150 refereed publications on software engineering topics and a co-author and co-editor of two books.

NEVEN VRČEK is a full professor of information science at the University of Zagreb, Faculty of Organization and Informatics. He received his PhD from the University of Zagreb, Faculty of Electrical Engineering and Computing. His research interests are embedded systems software development, electronic business, cloud computing and interoperability of information systems.