DECOMPOSING RUN-TIME RESOURCE MANAGEMENT IN HETEROGENEOUS RECONFIGURABLE SYSTEMS

Stefan Wildermann, Jürgen Teich

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
{stefan.wildermann, teich}@fau.de

ABSTRACT

Mixed workload and multi-application scenarios characterize modern and future reconfigurable systems. On the one hand, such systems consist of applications with objectives which may dynamically change during different execution phases. On the other hand, the designer or user of the system specifies requirements, e.g., regarding its power consumption, which have to be filled at any time. The main challenge is to partition the resources of the heterogeneous hardware architecture between the applications such that their objectives are optimized while fulfilling the system requirements. In this context, dynamic application objectives and system requirements can only be handled by providing self-adaptive resource management at run-time.

This paper discusses a distributed approach to resource management which is derived by applying Lagrangian dual decomposition. Each application is only aware of its own objectives and determines the desired amount of resources solely based on local information. The proposed mechanism steers the decisions of applications to be in compliance with the objectives and requirements of the overall system. We show that this approach achieves results which are competitive, and in many cases even significantly better than the results of a centralized heuristic used as state-of-the-art for resource management in reconfigurable systems while having the advantages of a distributed approach.

1. INTRODUCTION

Driven by the constant increase of the clock frequency, the functionality and usage scenarios of embedded systems have grown more and more complex and dynamic over the past years. Since 2005 however, semiconductors are increasingly confronted with physical issues concerning heat, power consumption, and leakage problems so that the increase in clock frequency of the computational resources has stagnated. Heterogeneous and highly parallel hardware architectures have emerged as a consequence. Obtaining further performance gains on these architectures requires that programs exploit the available parallelism. At the same time, the architectures have to provide reconfigurability so that the resources can be shared optimally between applications for varying workloads and other dynamic usage scenarios.

In this context, resource allocation is an optimization problem with further objectives in addition to the performance increase of individual applications. For instance, mobile systems usually need an allocation of computational resources with the requirement of the power consumption staying within a power budget. This means that it is required to select an implementation for each application such that their objectives are optimized while fulfilling such system requirements.

In this paper, we provide a mechanism for distributed resource allocation based on a round based negotiation scheme. We derive this negotiation scheme from the formulation of the original optimization problem by applying Lagrangian dual decomposition. This results in a mechanism where self-aware applications optimize their resource requirements based on local information only, while a master steers their decisions to be in compliance with the objectives and requirements of the overall system. One interesting aspect of this approach is the decoupling of master and applications since they do not require any local information of each other. This self-awareness and separation of concerns provides the scalability and flexibility required for dynamic usage scenarios.

2. RELATED WORK

Resource allocation for dynamic embedded systems is often tackled by design-time methodologies in the form of *scenario-based design*, e.g., [1], or *multi-mode system synthesis*, e.g., [2], as it is possible to apply powerful verification and optimization techniques to generate feasible and highly optimized implementations. Nonetheless, near-future embedded systems cannot be fully predicted at design-time due to dynamic usage scenarios and unexpected unavailability of hardware resources because of aging, reliability, or temperature effects.

The only way for being able to deal with this is to provide the system with a run-time resource management (RRM) layer which dispatches the reconfigurable resources to the

applications. *Centralized RRMs* collect all information relevant for calculating an optimized resource allocation. However, they have to deal with scalability and reliability issues as they form a single point of failure and produce communication and computation hot-spots and bottlenecks at the processors running the RRM. Even security issues can arise when an application has to reveal all internal information, and thus introduce the possibility of side-channel attacks. Consequently, several *decentralized RRM* approaches have been proposed, which are often based on multi-agent systems in the embedded domain [3, 4]. Here, also mechanisms from distributed computing (grid computing, cloud computing) could be adopted, which are mostly provided through auctions. In this paper, we present a formal approach to establish a distributed resource allocation approach where we apply techniques from convex optimization to incorporate self-awareness into the RRM.

3. DECOMPOSITION OF THE RUN-TIME RESOURCE MANAGEMENT

An *implementation* of an application i can be characterized by a vector x_i. It contains the implementation's quality numbers regarding application and system objectives, as well as the amount of all resource types required to run this implementation on the heterogeneous system. The set of all possible implementations is denoted by D_i. This turns out to be a Pareto front, only containing implementations which are non-dominated regarding the objectives and resource requirements. Figure 1 illustrates an example. There is of course the question of how to determine D_i, and we observe two major directions in the related work. The first one is to determine the non-dominated implementations D_i at design-time by performing design space exploration (DSE) and applying profiling techniques [5, 6]. The second one is to perform adaptive auto-tuning which uses parametrized code variants [7] for being able to generate various implementations at run-time. They can then be evaluated by monitoring the values of the objectives during their execution.

An application has different *utilities* for running in one of the implementations, which is expressed by utility function $f_i : D_i \to \mathbb{R}$. This utility may depend on the current execution phases of the application. The purpose of the utility function is to assign each implementation with a scalar value, which defines a total order over all elements in D_i. This is necessary for being able to make decisions at run-time. Such functions are commonly achieved by performing a *scalarization* of all relevant application objectives, e.g., [5, 6].

3.1. Resource Allocation Problem

Resource allocation in heterogeneous reconfigurable systems can be formulated as a combinatorial problem with the goal

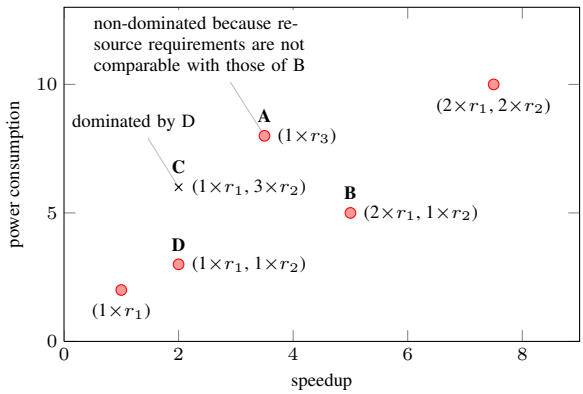

Fig. 1. Example of implementations depicted as a Pareto front for objectives speedup, power consumption, and usage of three resource types (r_1, r_2, r_3). Each point is annotated with the amount of required resources of each type.

of selecting implementations of all applications which maximize their utilities while adhering to the system constraints. Constraints originate from restricted physical and abstract resources (e.g., available amount of computational resources of a specific resource type or restricted power budgets). The upper bound of a constraint j is specified by $\overline{r}_j$, and the amount required by an implementation x_i is given by $r_j(x_i)$.

Whenever an application switches its execution phase or the system environment changes, the system should be reconfigured to optimally utilize the available resources. This problem can be formalized acc. to the following definition.

Definition 1. *Resource allocation problem*

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(x_i) \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_j(x_i) \leq \overline{r}_j, \quad j = 1, ..., m \tag{2}$$

Eq. (1) *represents the objective to maximize the average utility of all applications*[1]*, and Eq.* (2) *the* m *constraints.*

3.2. Decomposing the Resource Allocation Problem

Generally, the optimization problem formulated in Def. 1 is NP-hard. In this section, we therefore propose an approach which is based on solving the Lagrangian dual optimization problem. The main benefits are that this results in a convex optimization problem, which can be solved much more efficiently than the primal problem, and that it can be decomposed to enable a distributed solution method. The mathematical background applied in this section is, e.g., summarized in [8].

The *Lagrangian* of the resource allocation problem is

$$\mathcal{L}(x, \lambda) = -\left(\sum_{i=1}^{n} f_i(x_i)\right) + \sum_{j=1}^{m} \lambda_j \left(\sum_{i=1}^{n} r_j(x_i) - \overline{r}_j\right) \tag{3}$$

[1]The constant normalizing factor $1/n$ can be omitted.

where λ_j is the *Lagrangian multiplier* associated with the j-th constraint.

The *Langange dual function* is then defined as

$$\begin{aligned} g(\lambda) &= \inf_x \mathcal{L}(x,\lambda) = \\ &= \sum_{i=1}^{n} \inf_{x_i} \left(-f_i(x_i) + \sum_{j=1}^{m} \lambda_j \cdot r_j(x_i) \right) - \\ &\quad - \sum_{j=1}^{m} \lambda_j \cdot \overline{r}_j. \end{aligned} \tag{4}$$

The interesting aspects of the dual function are twofold. First, the optimal implementation x_i for given multipliers $\lambda = (\lambda_1, ..., \lambda_m)$ can be calculated by application i independent of other applications. Second, $g(\lambda)$ is concave and continuous in λ even when the primal objective function $\sum_{i=1}^{n} f_i(x_i)$ is not.

Now, the *Lagrange dual optimization problem* is given as

$$\begin{aligned} &\text{maximize} && g(\lambda) \\ &\text{subject to} && \lambda \geq 0, \end{aligned} \tag{5}$$

which, due to the nature of $g(\lambda)$, is a convex optimization problem. As such, it is possible to apply standard methods to determine the optimal value for λ. An algorithm based on the *subgradient method* [8] is summarized in Algorithm 1. All application subproblems can be solved independently, and the master problem of maximizing the dual function is solved by applying the subgradient method for all Lagrange multipliers.

Algorithm 1: Algorithm for solving the dual optimization problem.

```
while !stopping criterion do
    // application subproblems
    for each i = 1,...n do
        Find x_i that minimizes
        ( -f_i(x_i) + sum_{j=1}^{m} λ_j · f_j(x_i) );
    // master problem
    for each j = 1,...m do
        // Calculate subgradient of λ_j
        Δ_j = r̄_j - sum_{i=1}^{n} r_j(x_i);
        // Apply update rule acc. to
           subgradient method
        λ_j = max{0, λ_j - α_{t,j} · Δ_j} ;
```

The algorithm proposes a negotiation scheme, as illustrated in Figure 2. The Lagrange multipliers can be interpreted as the *price* of the respective resource (cf. [9]): The master tries to maximize the *costs*, while each application determines how much resources it wants to *buy* to maximize its *asset* for the current price. The big advantage is the self-awareness inherent in the algorithm: For no component is it necessary to have any internal details about another component.

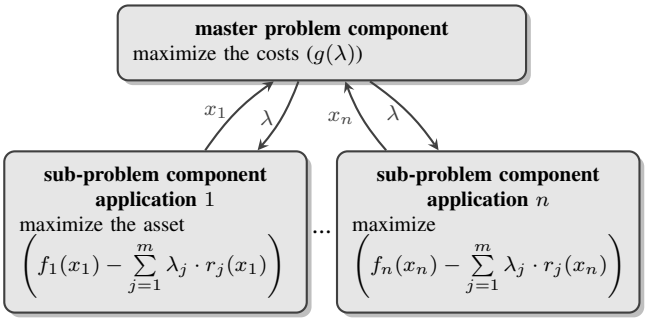

Fig. 2. Schematic illustration of resource allocation based on dual decomposition.

A disadvantage is that due to the Lagrangian relaxation in Eq. (3) the optimum is only approximated. This induces that there might be a gap between the optimal value f^* of the original problem and the optimal value g^* of the dual problem, so that $(-f^*) - g^* \geq 0$ could be non-zero. As a consequence, the negotiated outcome may not be achievable or feasible. We therefore propose the following heuristic. Applications can claim resources, e.g., by using mechanisms known from *invasive computing* [10], which enables the exclusive reservation of resources. Whenever resource conflicts arise during this embedding, applications are prioritized to resolve these conflicts. We choose the priority to be proportional to

$$f_i(x_i) - \sum_{j=1}^{m} \lambda_j \cdot r_j(x_i), \tag{6}$$

which represents the negotiated *asset* of application i. Applications which do not find sufficient resources for their negotiated implementation x_i choose this implementation x_i' that can be feasibly implemented on the remaining resources with maximal asset acc. to Eq. (6).

4. EXPERIMENTS

In this section, we present the results of our approach. In one version, the negotiation is performed for 20 rounds (*rrm20*) and in the other version for 100 rounds (*rrm100*) before the applications are embedded. We compare our approach to the knapsack heuristic from [5] (*knap*). For the first experiments, we generated test cases from the *e3s benchmark* [11]. It contains five applications. The architecture contains three different resource types. For each application, we generated the Pareto sets D_i by performing a DSE using the Opt4J

Table 1. Results for e3s benchmark case study.

α	approach	speedup	power[W]	utility $\sum_i f_i(x_i)$
0	*knap* [5]	3.21	8.25	**-8.25**
	rrm20	5.00	8.80	-8.80
	rrm100	5.00	8.80	-8.80
0.5	*knap* [5]	3.59	6.59	**-1.50**
	rrm20	5.00	8.80	-1.90
	rrm100	5.00	8.80	-1.90
0.75	*knap* [5]	24.14	40.07	8.09
	rrm20	27.07	45.44	**8.94**
	rrm100	26.85	45.08	8.86
1.0	*knap* [5]	27.92	57.67	27.92
	rrm20	31.20	65.39	31.20
	rrm100	31.48	63.66	**31.48**

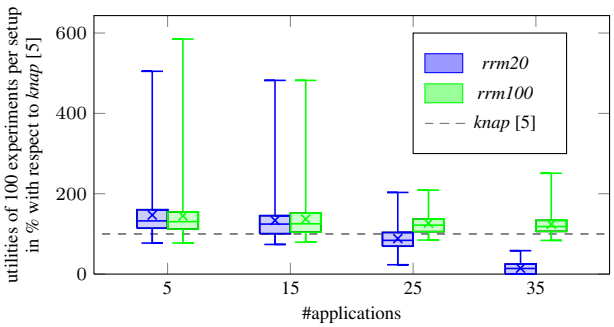

Fig. 3. Boxplots of the results for synthetic test cases relative to the results of *knap* (indicated by dashed line) for 100 experiments per setup.

framework [12] and optimized for resource usage, speedup (compared to the implementation with highest latency) and power consumption. The utility function is chosen according to $f_i(x_i) = \alpha \cdot speedup(x_i) - (1-\alpha) \cdot power(x_i)$, where α is a weight for scalarizing the two objectives to maximize the speedup and to minimize the power consumption. Results for different values of α are shown in Table 1. In all cases, the results are close together, showing that the proposed distributed approach is competitive with a fully centralized heuristic.

We furthermore performed test runs on synthetic test cases with 5, 15, 25, and 35 applications, and an architecture consisting of four resource types. For each such setup, we generated 100 cases and evaluated them. Fig. 3 shows the boxplots of the results for each setup, where the results of *knap* serve as baseline and all other results are given relative to this in percent. The results show that the number of negotiation rounds has to increase with the number of applications as *rrm20* degradates with the number of applications. In case of *rrm100* however, the proposed approach even performs better in a significant amount of experiments.

5. CONCLUSION

This paper demonstrates the application of formal mechanisms to incorporate self-awareness into run-time resource management (RRM) for embedded reconfigurable systems. We have presented the use of Lagrangian relaxation and dual decomposition. But also other techniques, such as game theory [13], are promising mathematical tools to perform this task. We discussed several advantages of distributed and self-aware approaches for RRM compared to centralized heuristics. Nonetheless, they usually perform better than distributed approaches as all details are available. However, the experiments have shown that the proposed distributed approach is competitive and in many cases even significantly better than a state-of-the-art centralized heuristic.

6. REFERENCES

[1] P. van Stralen and A. Pimentel, "Scenario-based design space exploration of MPSoCs," in *Proceedings of ICCD*, oct. 2010, pp. 305 –312.

[2] S. Wildermann, F. Reimann, D. Ziener, and J. Teich, "Symbolic design space exploration for multi-mode reconfigurable systems," in *Proceedings of CODES+ISSS*, 2011, pp. 129–138.

[3] M. Al Faruque *et al.*, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," in *Proceedings of Design Automation Conference (DAC)*, 2008, pp. 760–765.

[4] S. Kobbe *et al.*, "DistRM: distributed resource management for on-chip many-core systems," in *Proceedings of CODES+ISSS*, 2011, pp. 119–128.

[5] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal, "Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management," in *Proc. of SOC*, nov. 2006, pp. 1 –4.

[6] G. Marianik *et al.*, "Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures," in *Proc. of DATE*, 2012, pp. 1379–1384.

[7] Y. Li, J. Dongarra, and S. Tomov, "A note on auto-tuning GEMM for GPUs," in *Proc. of ICCS*. Springer-Verlag, 2009, pp. 884–892.

[8] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE JSAC*, vol. 24, no. 8, pp. 1439 –1451, aug. 2006.

[9] S. Boyd and L. Vandenberghe, *Convex optimization*. New York, NY, USA: Cambridge University Press, 2004.

[10] J. Teich *et al.*, "Invasive computing: An overview," in *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*, M. Hübner and J. Becker, Eds. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

[11] R. Dick, "Embedded system synthesis benchmarks suite," 2010, http://ziyang.eecs.umich.edu/dickrp/e3s/.

[12] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J – a modular framework for meta-heuristic optimization," in *Proc. of GECCO*, Dublin, Ireland, 2011, pp. 1723–1730.

[13] S. Wildermann, T. Ziermann, and J. Teich, "Game-theoretic analysis of decentralized core allocation schemes on many-core systems," in *Proceedings of DATE*, 2013, pp. 1498–1503.