Multi-Objective Distributed Run-time Resource Management for Many-Cores

Stefan Wildermann, Michael Glaß, Jürgen Teich
University of Erlangen-Nuremberg, Germany
{stefan.wildermann, michael.glass, teich}@fau.de

***Abstract*—Dynamic usage scenarios of many-core systems require sophisticated run-time resource management that can deal with multiple often conflicting application and system objectives. This paper proposes an approach based on *non-linear programming* techniques that is able to trade off between objectives while respecting targets regarding their values. We propose a *distributed application embedding* for dealing with soft system-wide constraints as well as a *centralized* one for strict constraints. The experiments show that both approaches may significantly outperform related heuristics.**

I. Introduction

Mixed workload and multi-application scenarios characterize modern multi-core and future many-core systems. On the one hand, such systems consist of applications with different performance objectives and requirements which may dynamically change during different execution phases. On the other hand, the designer or user specifies objectives and requirements for the overall system, e.g., regarding its power consumption. Therefore, one of the main challenges is the partitioning of the heterogeneous platform resources between the applications in a way such that the different objectives are optimized while fulfilling the requirements, involving a *dynamic resource allocation* problem. Consequently, assigning resources to applications is a multi-objective optimization problem, commonly handled by optimization techniques at design-time [1] for obtaining highly optimized design points. Multi-objective optimization results in multiple non-dominated design points. The advantage of having a set of options rather than a single compromise solution is that a designer can trade off objectives while selecting the design point for implementing the system, e.g., accepting a small increase in power consumption to obtain a strong increase in computing performance.

Nonetheless, workloads of many embedded systems cannot be fully predicted at design-time due to dynamic usage scenarios and unexpected unavailability of hardware resources caused by aging or temperature effects. As a consequence, run-time approaches will be necessary. However, considering multiple objectives in *run-time resource management (RRM)* has several challenges which are summarized in the following, including drawbacks of how they are tackled currently in related work.

Challenges and Related Work: We identify three main research questions for which we provide solutions in this paper.

How to automatically trade off between multiple often conflicting objectives at run time in the absence of a human decision maker? The problem of trading off multiple objectives is twofold. Not only are multi-objective optimization techniques computationally expensive with a too high computational overhead for being performed at run-time, but Pareto-optimal solutions are also non-comparable. In run-time optimization, the quality numbers of multiple objectives are often aggregated to a scalar value by using a weighted sum. This single objective function also delivers a total order on otherwise non-comparable points. For example, [2] uses a weighted sum over multiple objectives, such as performance and energy consumption, and [3] weights the objectives of minimizing communication cost and the overall computational load on the processors. It is, however, hard to anticipate the influence of the weighting on the quality numbers of the objectives, particularly on how they may deviate from expected target values. While it is possible to calibrate the weighting such that it obtains acceptable results for one scenario (number of applications, available platform resources, etc.), the results might completely degrade for another.

How to specify and obtain desired target values for the objectives? For several application domains of embedded systems, there are target values regarding objectives, e.g., desired throughput in multimedia applications or signal processing. They may even be set by the user during the operation of the system. In this realm, both [4] and [5] propose RRMs optimizing the energy consumption while respecting hard performance constraints. However, the drawback of both approaches is that common constraints are soft, particularly when specifying expected target values in multimedia and signal processing. Considering them as strict bounds reduces the possibility to trade off objectives, e.g., performance against power consumption. An approach where power optimization is performed with soft performance constraints is provided in [6].

How to distribute RRM in the presence of global and system-wide constraints? Distribution of run-time management has been identified as one of the most important prerequisites to scale with the increasing size of near future many-core systems [7]: Centralized RRM forms a single point of failure, requires high monitoring and processing overhead, and creates bottlenecks and hotspots around the processor performing the run-time management. Only few approaches enable distributed RRM despite their advantages with respect to near future many-core systems. Existing approaches like [8], [9], [10] support physical constraints concerning the availability of platform resources. They, however, do not provide solutions for system-wide requirements regarding non-functional objectives like power budgets. A serious problem is that the presence of system-wide requirements hampers the possibility to distribute the run-time management.

Our contribution: In this paper, we provide a run-time management scheme based on nonlinear programming. The resource allocation problem is NP-hard. However, through

978-3-9815370-2-4/DATE14/©2014 EDAA

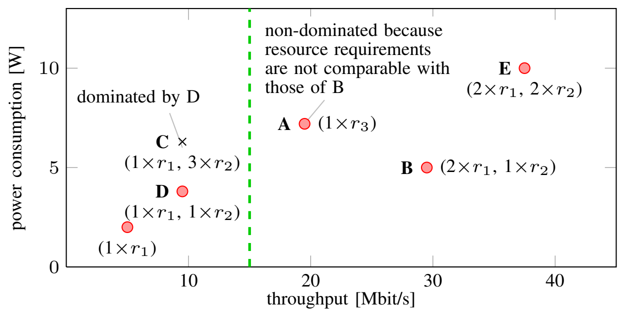


Fig. 1. Example of implementations depicted as a Pareto front for objectives throughput, power consumption, and usage of three resource types (r_1, r_2, r_3). Examples of resource types are processors of different types, memory usage, communication bandwidth, etc.

application of Lagrangian relaxation, we are able to transform it into a convex optimization problem, which approximates the result of the original problem but can be solved much more efficiently. We present an algorithm for solving this approximation problem and derive (a) a distributed RRM for dealing with soft system-wide requirements, and (b) a centralized RRM capable of dealing with strict bounds on global constraints. Our approaches (c) automatically trades off between objectives while (d) a target value can be specified for each of them.

II. Hybrid Strategies for Run-time Resource Management

There is an emerging trend of providing hybrid strategies to perform resource management in many-core systems. Hybrid strategies combine computationally expensive optimization and verification techniques at design-time with run-time management techniques [7]. The goal is to enhance the predictability of executing real-time applications on dynamic many-core platforms. Such techniques take a formal model [4] or the application source code [11] as entry point where the application is partitioned into multiple actors or tasks. Based on this specification, design space exploration and profiling techniques are applied [4], [11] to evaluate various *operating points* of an application by generating different mappings of tasks onto the resources.

Formally, an operating point of application i can be characterized by a vector x_i, containing the respective code version, quality numbers, e.g., with respect to execution time and power consumption, and the required platform resources, e.g., types and number of used processors, memory usage, communication bandwidth, etc. (cf. [2], [4], [12]). The set of all possible operating points available for application i is denoted by D_i. It is efficient to formulate D_i as a Pareto front, only containing points which are non-dominated regarding their objectives and resource requirements [12]. Fig. 1 illustrates an example. Note that it might be necessary to prune D_i at design-time for being able to reduce the memory requirements for storing operating points.

Whenever resource availability or status, or application requirements change, the system should be reorganized to efficiently utilize the available resources by triggering the run-time management. For this purpose, applications should contain *control points* [11] which are points in the code where it is possible to switch between different operating points. Thus, whenever an application reaches a control point, RRM is able to adapt its resource allocation. The main goal of the run-time management is then to identify an allocation of the resources between the applications which is optimal with respect to possibly conflicting goals, e.g., maximizing application performances while minimizing power consumption. Mathematically, this basically becomes a *knapsack problem* [5], [2]: Select one operating point for each application such that the objectives are optimized while all constraints and resource restrictions are fulfilled. In the next section, we analyze this problem and then provide an approximation algorithm for solving it in a distributed fashion.

III. Run-time Resource Allocation

There is no best practice for dealing with multiple objectives at run-time because RRM requires a total order over all Pareto-optimal, non-comparable solutions. We therefore propose to split the set of objectives such that one objective becomes the *main optimization goal*. For the remaining objectives, *target values* can be specified which constitute desired values which should be reached, but solutions may also deviate from these values. RRM therefore becomes the minimization[1] problem shown in Eq. (1).

The *main optimization goal* is expressed in Eq. (1a). The (possibly vector-valued) remaining *objective functions* $f_i(x_i)$ are incorporated into the optimization problem as soft constraints acc. to Eq. (1b), where $\overline{f}_i$ specifies the desired (vector-valued) target value as upper bound[2]. Applications may additionally have hard constraints acc. to Eq. (1c). The target values or hard constraints regarding the (possibly vector-valued) system objective function $g(x_1, \ldots, x_n)$ are handled in Eq. (1d). The resource restrictions in Eq. (1e) are strict due to the availability of physical resources. Note that all function can be non-linear and non-convex and do not have to be differentiable.

$$\begin{aligned}
&\text{minimize} && h(x_1, \ldots x_n) && \text{(1a)}\\
&\text{subject to} && f_i(x_i) \leq \overline{f}_i, \quad i = 1, ..., n && \text{(1b)}\\
& && F_i(x_i) \leq \overline{F}_i, \quad i = 1, ..., n && \text{(1c)}\\
& && g(x_1, \ldots, x_n) \leq \overline{g} && \text{(1d)}\\
& && r(x_1, \ldots, x_n) \leq \overline{r} && \text{(1e)}
\end{aligned}$$

For the remainder of this paper, we assume that above functions depending on multiple applications can be formulated as sums of application-specific functions, e.g., $h(x_1, \ldots, x_n) = \sum_{i=1}^{n} h_i(x_i)$, which is, e.g., the case for objectives and constraints regarding power/energy, average performance metrics, area, processor load, etc. Objective functions, where this is not the case, can however be decoupled by augmenting x_i with additional variables and introducing further constraints (cf. [13]).

A. Dealing with Hard Application Constraints

Dealing with the hard application constraints in Eq. (1c) is alleviated due to the design-time characterization of operating

[1]This is w.l.o.g., because an objective $f(x)$ that should be maximized can be rewritten as $f'(x) = -f(x)$ and we consequently obtain an objective to be minimized.

[2]This is again w.l.o.g., because an objective $f(x) \geq \overline{f}$ with lower-bounded target value can be rewritten as the equivalent upper-bounded inequality $-f(x) \leq -\overline{f}$.

points. All operating points that do not fulfill these requirements are removed from the Pareto set. RRM only selects operating points $x_i \in D_i'$ where

$$D_i' = D_i \setminus \{x_i | F_i(x_i) > \overline{F}_i\}. \tag{2}$$

Consider Fig. 1 for an example: If the application constraint is to reach a throughput of at least 15 Mbit/s (dashed line), only operating points **A**, **B**, and **E** should be considered. This drastically enhances the predictability of dynamic RRM. For the further analysis of the RRM optimization problem, we can therefore omit Eq. (1c).

B. Approximation Algorithm for Resource Allocation Problem

Generally, the optimization problem as formulated in Eq. (1) is NP-hard. We apply non-linear programming techniques [14] to derive an approximation algorithm for the solution of the resource allocation problem. First, the *Lagrangian* of the problem is formulated, which relaxes the constraints by summarizing the objective and the costs for violating constraints in a scalar function. Second, the *dual optimization problem* of the resource allocation problem is formulated. It considers an objective function, called the *dual function*, that is based on the Lagrangian for the combined optimization of the objective and the constraint violations. We then propose an algorithm which solves the dual optimization problem. As the dual problem is a convex optimization problem, we can solve it much more efficiently than the original problem. The solution of the dual problem only approximates the solution of the original problem. In the next section, we therefore derive two RRM techniques based on this algorithm, where one enables distributed application embedding and one enables the handling of strict system-wide constraints.

The *Lagrangian* of the resource allocation problem is

$$\begin{aligned}\mathcal{L}(x,\lambda,\mu,\nu) = \sum_{i=1}^{n} h_i(x_i) + \sum_{i=1}^{n} \lambda_i^{\mathrm{T}} \left(f_i(x_i) - \overline{f_i}\right) + \\ + \mu^{\mathrm{T}} \left(\sum_{i=1}^{n} g_i(x_i) - \overline{g}\right) + \\ + \nu^{\mathrm{T}} \left(\sum_{i=1}^{n} r_i(x_i) - \overline{r}\right).\end{aligned} \tag{3}$$

$\mathcal{L}(x,\lambda,\mu,\nu)$ is a weighted sum of the main objective function $h(x_1,\ldots,x_n)$ (with a weight of 1) and the constraints, where λ_i is the *Lagrangian multiplier* associated with the (soft) constraints of application i, μ is associated with the system constraints, and ν is associated with the resource restrictions.

The *dual function* serves as the objective function of the dual problem. The dual function of the resource allocation problem determines the minimal value of $\mathcal{L}(x,\lambda,\mu,\nu)$ for given Lagrangian multipliers acc. to

$$\begin{aligned}d(\lambda,\mu,\nu) = \inf_{x=(x_1,\cdots,x_n)} \mathcal{L}(x,\lambda,\mu,\nu) = \\ = \sum_{i=1}^{n} \inf_{x_i \in D_i} \left\{ h_i(x_i) + \lambda_i^{\mathrm{T}} f_i(x_i) + \mu^{\mathrm{T}} g_i(x_i) + \nu^{\mathrm{T}} r_i(x_i) \right\} \\ - \sum_{i=1}^{n} \lambda_i^{\mathrm{T}} \overline{f_i} - \mu^{\mathrm{T}} \overline{g} - \nu^{\mathrm{T}} \overline{r}.\end{aligned} \tag{4}$$

Algorithm 1: Approximation algorithm for solving the dual optimization problem in $\#rounds$ iterations.

```
1  while $t \le \#rounds$ do
2      for each $i = 1, \ldots, n$ do
           /* Application Subproblem */
3          Find $x_i \in D_i$ that minimizes
             $\left\{ h_i(x_i) + \lambda_i^{\mathrm{T}} f_i(x_i) + \mu^{\mathrm{T}} g_i(x_i) + \nu^{\mathrm{T}} r_i(x_i) \right\}$ (5)
           /* Application Constraints */
4          $\Delta_{\lambda_i} = f_i(x_i) - \overline{f_i}$;   // subgradient
5          $\lambda_i = \max\{0, \lambda_i + \alpha_t \cdot \Delta_{\lambda_i}\}$;   // update rule
       /* System Constraints */
6      $\Delta_\mu = \sum_{i=1}^{n} g_i(x_i) - \overline{g}$;   // subgradient
7      $\mu = \max\{0, \mu + \alpha_t \cdot \Delta_\mu\}$;   // update rule
       /* Resource Restrictions */
8      $\Delta_\nu = \sum_{i=1}^{n} r_i(x_i) - \overline{r}$;   // subgradient
9      $\nu = \max\{0, \nu + \alpha_t \cdot \Delta_\nu\}$;   // update rule
10     t = t + 1;
```

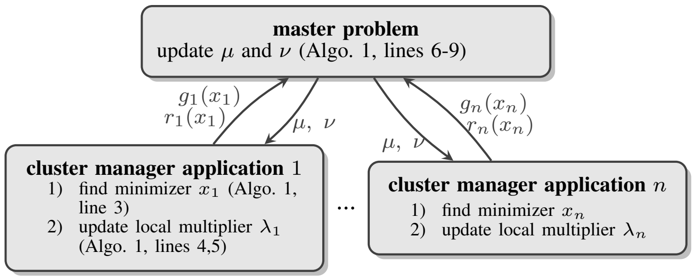
This function is *concave* even when the objective and constraint functions of the original problem are not. The *dual optimization problem* is

$$\begin{aligned}\text{maximize} \quad & d(\lambda,\mu,\nu) \\ \text{subject to} \quad & \lambda \ge 0,\ \mu \ge 0,\ \nu \ge 0\end{aligned}$$

which is a convex optimization problem as it maximizes the concave function $d(\lambda,\mu,\nu)$. As such, it is possible to apply standard methods to determine the optimal values for λ, μ, ν. We apply the *subgradient method* [14] and obtain Algorithm 1, which specifies an iterative scheme for solving the dual problem. In each iteration, we first consider each application i: For given values of λ, μ, ν, we determine that $x_i \in D_i$ which constitutes the application-specific minimum of Eq. (5) (line 3), which is derived from Eq. (4). Then, all Lagrangian multipliers are updated by the subgradient method, which is based on the gradient descent method, extended to non-differentiable functions. For updating each multiplier, it chooses a subgradient of $d(\lambda,\mu,\nu)$ into the direction of the respective multiplier (lines 4, 6, 8, respectively). The multipliers are recomputed by scaling the subgradient by a *step size* α_t and then adding it to the respective multiplier, which are then projected to feasible values so that $\lambda \ge 0,\ \mu \ge 0,\ \nu \ge 0$ (lines 5, 7, 9, respectively).

IV. Decomposing the Resource Allocation Problem

A general description of *distributed RRM* method is provided in [7]. The RRM problem is divided into subproblems (also called *clusters*). One processing core is assumed to manage the resource allocation for one cluster. These *cluster managers* communicate with each other via a global manager. The idea of mathematical decomposition theory is quite similar: In decomposition theory, the original large problem is decomposed into distributively solvable *subproblems* that are coordinated by a high-level *master problem.* Algorithm 1



master problem
update μ and ν (Algo. 1, lines 6-9)
$g_1(x_1)$
$r_1(x_1)$
$\mu,\ \nu$
$\mu,\ \nu$
$g_n(x_n)$
$r_n(x_n)$
cluster manager application 1
1) find minimizer x_1 (Algo. 1, line 3)
2) update local multiplier λ_1 (Algo. 1, lines 4,5)
...
cluster manager application n
1) find minimizer x_n
2) update local multiplier λ_n

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

specifies a *dual decomposition* of the resource allocation problem, illustrated in Fig. 2. It basically defines a round-based negotiation scheme: The master problem sets the prices (i.e., the corresponding multipliers) of constraints and resources, while every subproblem has to decide how much to contribute to each constraint and how many resources to use. Through iterative execution, the values of the multipliers are negotiated at master level, and the operating points x_i that minimizes Eq. (5) on the subproblem level. For a start, we consider subproblems at a granularity of single applications. Nonetheless, several applications could also be summarized together into one cluster as a coarser granularity of decomposition.

The main challenge is that the dual problem only approximates the optimum of the primal resource allocation problem in Eq. (1). This means that there might be a gap between the optimal value h^* of the primal problem and the optimal value d^* of the dual optimization problem. However, d^* is always a lower bound for h^*, i.e.,

$$d^* \leq h^*. \tag{6}$$

The consequence of this *duality gap* is that the negotiated outcome may violate constraints. This means when embedding the applications according to the negotiation result, there might arise conflicts due to, e.g., over-utilization of resources. We therefore provide two heuristics to deal with this; (a) a distributed embedding for soft system constraints and (b) a centralized embedding for hard system constraints. Note that when the duality gap is zero, the embedding approaches will guaranteed yield a Pareto-optimal solution of the original multi-objective optimization problem.

Distributed Application Embedding (DAE): One challenge of distributed application embedding is the potentially conflicting resource requirements of the applications. This can be tackled by applying mechanisms which allow cluster managers to explore the availability of platform resources and to enforce their exclusive reservation, e.g., by using mechanisms known from *invasive computing* [15]. During embedding of application i, the operating point x_i is selected that minimizes Eq. (5) while being feasibly implementable on the set of free resources. By doing this, the hard resource restrictions are self-enforced, while also respecting the soft system constraints by considering the costs arising due to the Lagrangian multipliers.

Centralized Application Embedding (CAE): Whenever hard system constraints are present, the decentralized embedding cannot be chosen as it does not check for the violation of system constraints. This case can be solved in a centralized fashion. Applications are prioritized according to the value of Eq. (5) of the negotiated result, where the application with highest value has highest priority. Applications are considered sequentially ordered according to their priorities, and the operating point is determined that minimizes Eq. (5) while being feasibly implementable on the unallocated resources without violating any constraint.

V. Experimental Evaluation

We experimentally evaluate the approaches to analyze scalability, execution time, and applicability for an embedded system case study.

A. Scalability

In the following experiments, we consider a heterogeneous many-core consisting of 4 resource types (e.g. processors) each with 25 instances. For each considered synthetic application, 100 operating points are generated each randomly assigned with the required amount of resources of the four resource types, power consumption and speedup/throughput. Dominated points are removed, thus only keeping Pareto-optimal operating points for application embedding. Note that applications are also allowed to select no operating point during negotiation and embedding. We investigate (1.) performance maximization under a given power budget, and (2.) power consumption minimization under performance constraints for each application.

1) Performance Maximization under Power Requirement: The first optimization scenario is formulated as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} \text{speedup}_i(x_i) \\ \text{subject to} \quad & \sum_{i=1}^{n} \text{power}_i(x_i) \leq \overline{\text{power}} \\ & \sum_{i=1}^{n} r_i(x_i) \leq \bar{r} \end{aligned}$$

The goal is to maximize the average speedup were the constant normalizing factor can be neglected [9], [10]. Only the available resources can be used according to the resource restrictions. The target value for the power consumption is set to $\overline{\text{power}}$=10 W. We choose a *non-diminishing step size rule* with $\alpha_t = 0.1/\sqrt{t}$ for Algo. 1. As the dual solution specifies an upper bound for the problem, we execute the algorithm for 800 rounds per experiment to obtain this bound and use it as reference for each test case. First, we analyze the impact of the number of rounds on the quality of the application embedding. Several experiments are conducted with 10, 20, and 35 applications for 20, 50, and 95 rounds. For each setup, 100 test cases are generated and solved by *DAE*. Fig. 3 shows the median of the speedups relative to the upper bound for 100 experiments per setup. The results show that the number of rounds has to increase approx. linearly with the number of applications.

Next, we compare *CAE* and *DAE* to RRM approaches based on the knapsack heuristic from [5]: (*KS*) knapsack heuristic with strict power constraint, and (*KW*) knapsack heuristic with weighted sum for speedup and power consumption, calibrated for the case of 10 applications.

We analyze experiments with (a) 10 applications and 20 negotiation rounds, (b) 20 applications and 50 rounds, and (c) 35 applications and 95 rounds. Fig. 4 summarizes the results of 100 experiments per setup obtained for the main

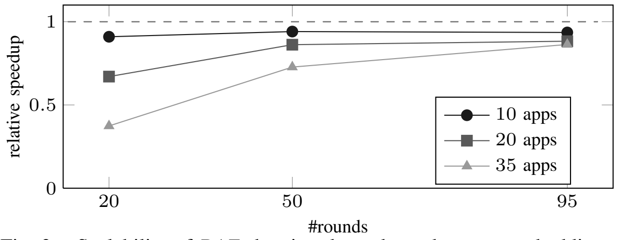


Fig. 3. Scalability of *DAE* showing dependency between embedding quality (median of 100 experiments per setup) and amount of applications and rounds.

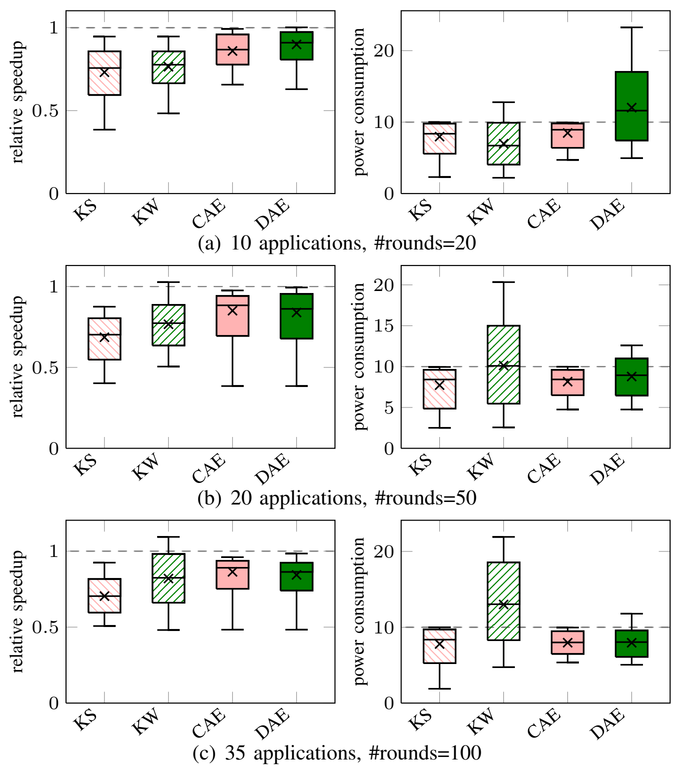


Fig. 4. Results of performance maximization under power requirements (10 W) for different number of applications. The boxplots summarize 100 experiments per setup, giving the minimum, the 10th percentile, median, 90th percentile, and maximum together with the average (×).

optimization goal relative to the upper bound and the power consumption. The proposed application embedding algorithms achieve significantly better results than *KS* in over 50% of the experiments in all setups, and also than *KW* with the exception of setup (c). However, the *KW* improvement comes with a tremendous excess of the target value for power consumption. The reason is, that *KW* is calibrated for the case of 10 applications, where it keeps the power restriction for 90% of the experiments. This calibration does not scale with the number of applications. Contrary, the centralized heuristics with strict constraints (*KS*, *CAE*) always fulfill the power consumption constraint. But due to the negotiation of the Lagrangian multipliers and their consideration during the distributed embedding, also *DAE* is able to keep the violation of the constraint in an acceptable range for setups (b) and (c), showing that the distributed application embedding is competitive with the centralized embedding for cases with many applications. For setup (a) with only a few applications, the distributed approach may lead to a tremendous violation. For this lower complex scenario, however, it is conceivable to apply a centralized approach like *CAE*.

The deviation of the speedup selected during negotiation and the actually obtained speedups during embedding is within acceptable bounds: It deviates for less than 21% of the applications in the experiments for setup (a) for *DAE*, less than 14% for setup (b), and less than 7% for setup (c).

2) Power Consumption Minimization under Performance Requirements: The second optimization scenario is formulated as:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} \text{power}_i(x_i) \\ \text{subject to} \quad & \text{throughput}_i(x_i) \geq \overline{\text{throughput}}_i, \ \forall i \\ & \sum_{i=1}^{n} r_i(x_i) \leq \bar{r} \end{aligned}$$

where the goal is to minimize the power consumption under individual performance constraints of n resource-competing applications. *KS* and *CAE* use strict throughput constraints, meaning only operating points fulfilling this requirement are selected. Contrary, *DAE* handles these as soft constraints, and *KW* rates each operating points by a weighted sum of its power consumption and throughput.

Fig. 5 summarizes the results of 100 experiments and 20 applications. The left side shows the number of applications achieving at least 66% (for *KW* and *DAE*) and 100% of the specified throughput. Our proposed approaches enable the execution of significantly more applications in almost all experiments, where *CAE* achieves the best results. Fig. 5 (right) shows the power consumptions achieved in the experiments relative to the value of *CAE*. *KS* achieves the best results in a significant amount of experiments but due to the fact, that only a fraction of the 20 applications are actually executed (not more than 1 application in 50% of the test cases, and not more than 5 out of 20 in 90% of the test cases), constituting a bad trade-off compared to *CAE* and *DAE*. Note that the power consumption of *KS* (and *KW*) even is over 7 times higher than that of *CAE* in one case. *KW* is able to execute more applications which fulfill their throughput requirements compared to *KS*, and also applications which fulfill at least 66.6% of their throughput requirements. However, results of *KW* for power consumption have an immense variance. Due to the variance and the fact that still significantly less applications are executed, *KW*'s results constitute bad trade-offs compared to results of *CAE* and *DAE*.

Finally, as *DAE* does not consider the throughput requirements as strict, it has more freedom in trading off between throughput and power consumption. As a result, it can significantly reduce the main optimization goal compared to *CAE* – of course, at the expense of reducing the number of applications reaching their target values.

B. Execution Time

For evaluating the execution times of the RRM approaches, we execute the performance maximization test cases from Sec. V-A1 on an i7 Quad-Core. Note that the negotiation phase can be executed distributively for both *CAE* and *DAE*. We measure the execution times for centralized and distributed negotiation, both also applicable for *CAE*, while application embedding takes less than a millisecond. In the distributed case, we generate four cluster managers which are executed as separate threads. Application subproblems are evenly assigned

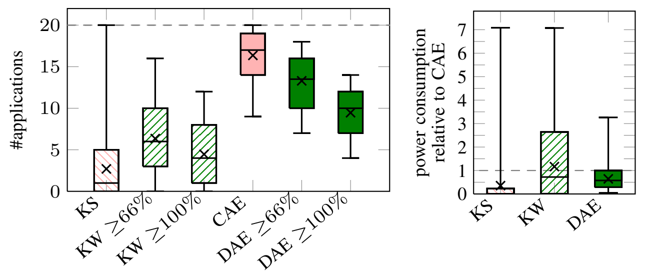


Fig. 5. Boxplots for power consumption minimization under performance requirements.

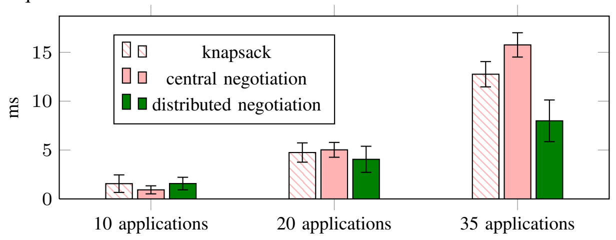


Fig. 6. Average execution times and standard deviation of RRM approaches for 100 experiments per setup.

to them. The centralized negotiation is executed sequentially. Fig. 6 depicts the average results which show that the execution time of the proposed negotiation increases faster than for the knapsack heuristic as the number of negotiation rounds has to be modified when dealing with more applications. However, the overhead can be significantly reduced when exploiting the parallelism obtained by decomposing the problem. While for the small test case (10 applications), the synchronization overhead of the distributed RRM imposes an overhead, it scales much better than both centralized approaches when increasing the number of applications.

C. Case Study

For evaluating the applicability of our RRM for embedded systems, we use the *E3S* benchmark [16]. First, we derive operating points for each of the benchmark's five applications (*consumer, auto, office, network, telecom*) by performing design space exploration using an optimization framework [17] with the objectives of minimizing resource usage and power consumption, and maximizing speedup (compared to the operating point with highest latency). We then test the RRM approaches for the goal of maximizing the average speedup with power budget set to 1 W and 10 W, respectively, on a heterogeneous 48-core system with three types of processors. Table I shows the average speedups (S) and the power consumption (P) for the cases of instantiating each of the five application one, five, and ten times (i. e., 5, 25, and 50 apps). It shows that *CAE* is able to find better allocations than *KS* for most cases. *DAE* is able to determine a good trade-off between objective and soft constraints when many applications have to be managed and constraints are not tight (particularly 50 apps and 10 W power budget). In cases of strict or tight constraints however (1 W power budget), *CAE* should be applied.

VI. Conclusion

This paper presents an RRM technique based on Lagrangian relaxation: a distributed version that can deal with soft system-wide constraints and a centralized version for strict constraints. The results show that these techniques achieve significantly better results in trading off multiple objectives than their respective counterparts known from existing work. This shows that it is promising to further improve the convergence criteria of the proposed RRM approaches. Future work therefore aims at analyzing different options of clustering applications together and decentralizing the master problem, which is alleviated by the convexity of the problem, as well as methods for distributing the solving of strict constraints.

TABLE I
Average speedup (S) and the power consumption (P) for case study (bold: best results, green: tradeoff between speedup maximization and power budget exceedance by DAE).

		1 W		10 W	
		S	P [W]	S	P [W]
5 apps	KS	**7.63**	**0.88**	8.72	3.31
	CAE	6.86	0.72	**10.53**	**3.60**
	DAE	8.59	1.38	**10.53**	**3.60**
25 apps	KS	1.50	0.77	2.00	1.02
	CAE	**2.02**	**1.00**	**4.38**	**9.92**
	DAE	2.90	1.53	**4.42**	**11.08**
50 apps	KS	0.75	0.77	1.00	1.02
	CAE	**0.89**	**0.88**	**1.11**	**9.97**
	DAE	1.59	1.58	**2.00**	**10.85**

Acknowledgement

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89).

References

[1] S.-H. Kang *et al.*, "Multi-objective mapping optimization via problem decomposition for many-core systems," in *ESTIMedia*, 2012, pp. 28–37.
[2] P. Bellasi *et al.*, "A RTRM proposal for multi/many-core platforms and reconfigurable applications," in *Proc. of ReCoSoC*, july 2012, pp. 1–8.
[3] P. Zipf *et al.*, "A decentralised task mapping approach for homogeneous multiprocessor network-on-chips," *Int. J. Reconfig. Comput.*, vol. 2009, pp. 3:1–3:14, Jan. 2009.
[4] A. K. Singh *et al.*, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," *ACM TODAES*, vol. 18, no. 1, pp. 9:1–9:29, Jan. 2013.
[5] C. Ykman-Couvreur *et al.*, "Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management," in *Proc. of SOC*, nov. 2006, pp. 1 –4.
[6] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," in *Proceedings of SOPS*, 2003, pp. 149–163.
[7] A. K. Singh *et al.*, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of DAC*. ACM, 2013, pp. 1:1–1:10.
[8] M. Al Faruque *et al.*, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," in *Proceedings of DAC*, 2008, pp. 760–765.
[9] S. Kobbe *et al.*, "DistRM: distributed resource management for on-chip many-core systems," in *Proceedings of CODES+ISSS*, 2011, pp. 119–128.
[10] S. Wildermann *et al.*, "Game-theoretic analysis of decentralized core allocation schemes on many-core systems," in *Proc. of DATE*, 2013, pp. 1498–1503.
[11] 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.
[12] C. Ykman-Couvreur *et al.*, "Pareto-based application specification for MP-SoC customized run-time management," in *Proceedings of IC-SAMOS*, 2006, pp. 78–84.
[13] C. W. Tan *et al.*, "Distributed optimization of coupled systems with applications to network utility maximization," in *Procceedings of ICASSP*, vol. 5, 2006, pp. 981–984.
[14] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, Sep. 1999.
[15] 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.
[16] R. Dick, "Embedded system synthesis benchmarks suite (E3S)," 2010, http://ziyang.eecs.umich.edu/dickrp/e3s/.
[17] 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.