# Goal Formulation: Abstracting Dynamic Objectives for Efficient On-chip Resource Allocation

Elham Shamsa[1], Anil Kanduri[1], Amir M. Rahmani[2,3], Pasi Liljeberg[1], Axel Jantsch[3], and Nikil Dutt[2]

[1]*Department of Future Technologies, University of Turku, Turku, Finland*
[2]*Department of Computer Science, University of California, Irvine, USA*
[3]*Institute of Computer Technology, TU Wien, Vienna, Austria*
{elsham, spakan, pakrli}@utu.fi, {amirr1, dutt}@uci.edu, axel.jantsch@tuwien.ac.at

***Abstract*—Run-time resource management of mobile heterogeneous systems is challenging due to the limited energy budget that has to be allocated among diverse workloads. User interaction within these systems alter the performance requirements, which often conflicts with concurrent applications' objectives and system constraints. Current resource allocation approaches focus on optimizing fixed objectives, ignoring the variation in system and applications' constraints at run-time. For adaptive resource allocation, it is necessary to abstract the applications' and system's requirements into goals, which can be dynamically formulated as a weighted combination of different objectives. We highlight the problem by illustrating the limitation of state-of-the-art resource allocation approaches and motivate the need of a goal management solution, which dynamically prioritizes different objectives and switches between them to adapt to the environment.**

***Index Terms*—On-chip Resource Management, Heterogeneous Multi-core Systems, Autonomous Systems**

## I. Introduction

Increasing usage of battery powered embedded systems enhances the importance of run-time management to maximize resource efficiency [1]. Considering system, application and user constraints including power and energy budgets, on-chip temperature, performance (latency and throughput), responsiveness and quality-of-service (QoS), often results in conflicting resource allocation decisions [2]. Run-time management policies dynamically allocate resources with the objective of optimizing for one or more of these constraints. The *objective* could be expressed as a linear cost function that combines different constraints to satisfy a specific criteria [2]. Practically, different objectives could be overlapping, being in conflict or orthogonal, making multi-objective resource management challenging. For example, under nominal conditions, the objective of high performance results also in high power consumption, which conflicts with the objective of low power operation.

With concurrent applications, user activity patterns and battery life over time, there is a variation in workload intensity, power consumption, thermal profile, energy budget available (battery life), and user requirements in terms of interaction and satisfaction. Each instance of a change in above parameters changes the requirements, which could be formulated possibly as a different objective. Despite continuously changing requirements, resource allocation policies strive to meet *fixed objectives* which are determined at design time, while the environment in which they operate is largely dynamic.

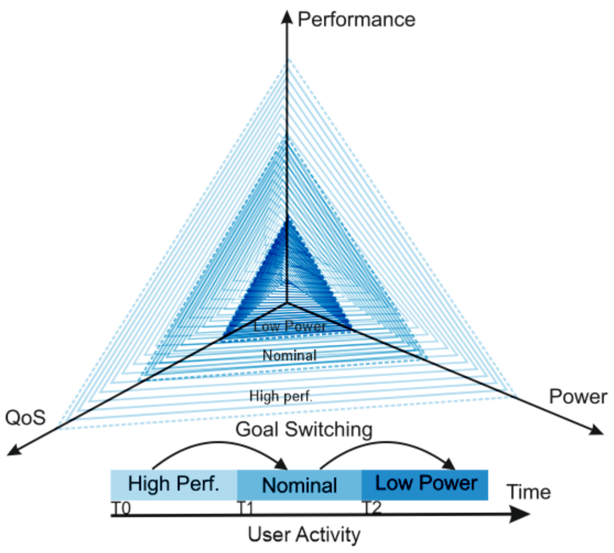

Fig. 1. Three dimensional space of requirements.

State-of-the-art resource management policies focus on satisfying such fixed objectives, restricting their efficacy to a fixed set of operating conditions. Figure. 1 shows the trinity of major requirements viz., power, performance and QoS, along with the Pareto-space of operating points with fixed objective policies. For simplicity, we present three major classes of policies, targeted at *low power*, *nominal* and *high performance* operation. Each region (shown in different shades) illustrates the possible operating points in terms of power, performance and QoS mode with each specific policy, which are aimed at optimizing a single objective. Each policy is efficient under specific conditions and requirements, however their efficiency is restricted to those specific modes of operation. For instance, a system using *low power* policy is hard-wired to minimize power consumption and thus cannot deliver a higher performance in comparison with a *high performance* policy. Depending on user activity patterns and available energy budget (e.g., when the *power saving* mode is chosen by a smartphone user to prolong battery life or when the *high performance* mode is selected for 3D gaming), requirements and thus objectives keep changing, where a specific policy fails in allocating and utilizing resources efficiently. As shown in Figure 1, user requirements evolve over time from high performance through to low power. Choosing any of the three specific policies would result in failing to satisfy at least one of the key parameters among power, performance and QoS. An ideal solution to this issue would be a design policy that can navigate through different operation modes subject to changing requirements. However, designing a dynamically adaptive policy with conflicting objectives can be extremely challenging. A more holistic solution is to abstract away the requirements into *goals* which can be formulated as a weighted combination of different objectives. Policies can be driven by goals, which will qualitatively embed dynamic changes in priorities of objectives and requirements. Goal formulation and goal driven management provides the necessary autonomy and adaptivity to current resource management techniques.

In the following sections, we elaborate on the problem with fixed objective policies by showcasing the limitation of state-of-the-art resource management approaches and present open directions towards goal management solutions.

## II. Limitations of State-of-the-art Fixed-objective Policies

To motivate the need for goal formulation and goal driven management, we present state-of-the-art resource management policies which are focused on fixed objectives and highlight their limitations. We consider three major classes of policies viz., `low-power` [3], `high-performance` [4], and `nominal` [1], covering possible modes of operation. We customize and implement these policies on the experimental platform, detailed in the following.

**Experimental Setup:** We use a Hardkernel Odroid XU3 board running Linux Ubuntu 15.04 as a platform to evaluate various resource management policies. This board integrates an Octa-core Exynos 5422 system-on-chip (SoC) as a heterogeneous multi-processor (HMP) with two clusters (4 big (A15) and 4 little (A7) CPU cores). Smartphones such as the Samsung Galaxy series have used this SoC or a similar HMP as their main processor [5]. The big cores provide higher performance while the LITTLE cores are optimized for low power. For thermal safety, we set the Thermal Design Power (TDP) to 5W in our experiments. The Odroid board has per-cluster power sensors for power measurement. We use the Heartbeats API [6] to monitor each application's performance (the application periodically issue heartbeats and informs the system about its performance). The micro kernels used in our experiments are least squares curve fitting, k-nearest neighbors classification, k-means clustering and linear regression [4]. We chose these machine learning kernels for their wide range of use cases in several application domains [1].

The implemented policies are discussed in the following.

### A. Low-power Policy

**Objective:** Minimizing power consumption with graceful degradation of performance.

**Problem Definition:** Resource managers in HMPs need to consider TDP constraint and performance efficiency while avoiding task migration oscillation between different cores. Therefore, the policy needs to satisfy the application requirements and conserve energy with respect to the TDP constraint [3].

**Solution:** A representative approach presented in [3] deploys several proportional-integral-derivative (PID) controllers to track the desired reference values. The first controller is a per-task resource share controller; the objective of the resource share controller is to keep the measured heart rate (Heartbeat per second) for each application in the specific range by regulating the time slice considered for each task in the scheduler. The second controller is a per-cluster DVFS controller. This cluster-level PID controller actuates on the frequency of each cluster to achieve the target utilization. The third one is a per-task QoS controller. This controller decreases the reference heart rate when thermal emergency occurs.

This policy is effective in providing low-power operation, by deliberately degrading performance under power/thermal emergency. For instance, if a user runs a compute-intensive application, this policy may not meet the requirements as the objective of the policy is fixed to conserve energy. Figure 2 (a) illustrates the chip power consumption when this policy is executed on the Odroid board. As shown in the figure, when the TDP is violated (red dotted line), the resource manager actuates and conservatively reduces the power, which is reflected in performance degradation. Figure 3 (a) illustrates the average performance of four running applications when this policy is used. In the case of TDP violation, the performance is significantly sacrificed to satisfy the low-power objective.

### B. High-performance Policy

**Objective:** Maximizing performance by fully utilizing available power budget.

**Problem Definition:** Maximizing performance through software approximation while considering power cap, workload intensity, and utilization metrics [4].

**Solution:** The authors in [4] use the combination of DVFS, power gating, and approximation to perform power capping while maximizing performance. When a power violation occurs, in the first step, the policy applies DVFS to downscale the voltage and frequency of running applications. In the next step, if power violation persists, the power gating (PG) knob is invoked to power off all the unoccupied cores. Finally, if the performance requirements are violated due to DVFS/PG, approximation knob is used to boost the performance of the system via switching the mode of approximable applications to the approximate mode.

Figure 2 (b) and Figure 3 (b) show the total power consumption of the system and the average performance of running applications, respectively, when using this policy

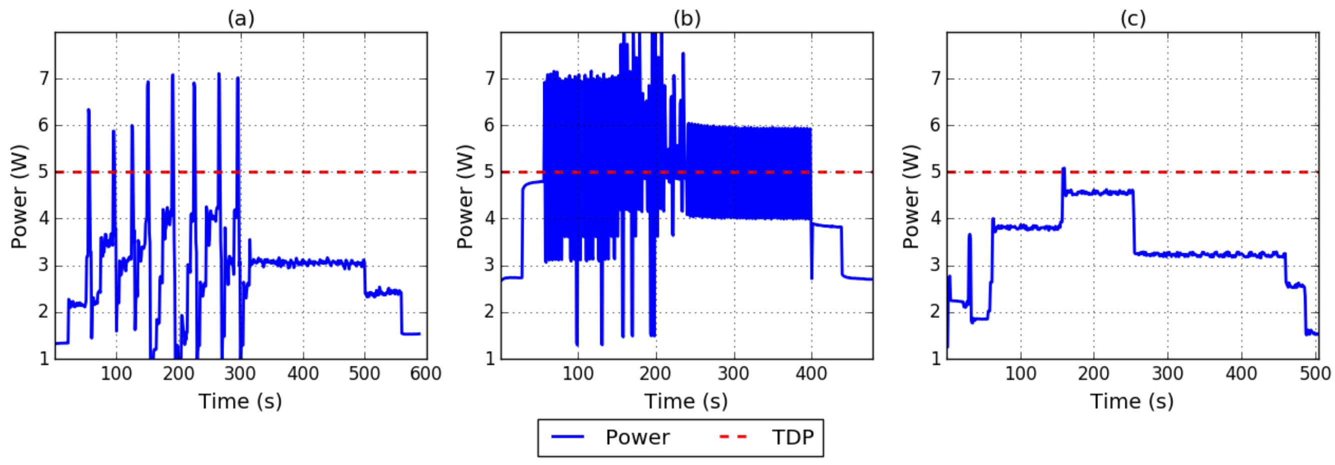

Fig. 2. Power consumption with different policies. (a) Low power [3], (b) High performance [4], (c) Nominal mode[1]

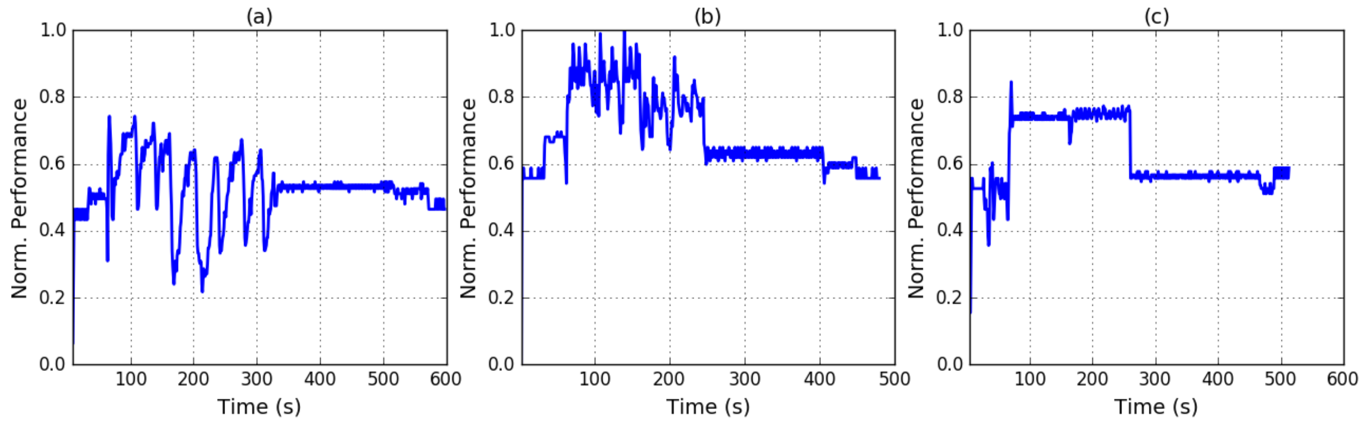

Fig. 3. Average performance with variable workloads with different policies. (a) Low power [3], (b) High performance [4], (c) Nominal mode [1]

execute. As mentioned, the objective of this policy is to aggressively maximize the performance, therefore, power violations in this policy are more frequent than the low-power policy. This policy over-boosts performance by sacrificing the QoS (using approximation) which is a proper choice when a high performance is needed, however, the policy is fixed and cannot adapt to possible changes.

## C. Nominal-mode Policy

**Objective:** Maximizing performance within minimal power consumption.

**Problem Definition:** Resource managers in HMPs need to consider unknown/dynamic workloads to meet power and performance constraints in a coordinated way [1].

**Solution:** A representative approach is presented in [1] deploying a coordinated actuation of approximation, DVFS, CPU quota scaling, and task migration to satisfy both power and performance constraints while considering system dynamics. This policy works in three phases: *idle*, *decide*, and *refine*. The system is in *idle* phase when it is balanced and power and performance requirements are satisfied. When an application enters or leaves the system, or a power violation occurs, the system state changes to *decide*. In this phase, at first, the new application is mapped on a LITTLE (i.e., low-power) core. If the required performance is not met or a power violation occurs then the controller will decide between two combinations of knobs: *approximate* setting or *accurate* configuration. This decision is made based on estimation models. In other words, if the models predict that *accurate* setting will be able to meet the requirements, the resource manager will select it, otherwise the approximate mode will be chosen to trade accuracy for power/performance. After the decision making, *refine* phase will take place. In this phase, the policy monitors power and performance and provides fine-tune controls over resource allocation.

In this presented policy there is a co-optimization of power and performance with a possible QoS degradation. Figure 2 (c) shows the total power consumption of the system when this policy is used in our setup. As can be seen from the figure, the power does not violate the TDP and the performance (Figure 3 (c)) is balanced. This policy is a suitable choice for a nominal mode of operation when a user does not ask for energy saving and no performance-intensive application is running on the system.

Each of these three policies focuses on satisfying a fixed objective and is optimal for a certain case. However, in a dynamic environment, requirements and thus objectives keep changing and using a fixed objective policy would fail to allocate and utilize resources efficiently. While sometimes these different objectives can be reconciled, very often one has to be compromised for the benefit of another objective through prioritization of different constraints w.r.t. changes in the user preferences, environment, or system's own state. A more holistic solution is to abstract away the requirements into *goals* which can be formulated as a weighted combination of different objectives.

## III. Dynamic Goal Formulation

We argue that the state-of-the-art resource management policies with fixed objective are not effective in a dynamic environment. There is a growing need for a comprehensive method that can monitor the environment and dynamically determine the new objectives. Dynamic goal formulation can prioritize objectives at run-time and reassign weights to each objective based on the user, system and applications requirements. It considers all the requirements with their conflicts, then abstracts away the objectives into the goal. Fig. 4 illustrates a high-level architecture of the run-time goal driven management framework. The goal driven management receives applications and user requirements, then the goal formulation module formulates the goal according to the priority of each required objective, based on the determined priorities the resource allocation module run the best policy for allocation. This framework has the following properties:

- Interact with user and applications that have different requirements,
- Receive the user and applications requirements,
- Consider the system constraints,
- Formulate goals dynamically by assigning weights to each individual objective based on the system priority and inputs,
- Select one of the most effective policies based on the goal.

In Section II, we discussed three different policies with fixed objective which are determined at the design time. In the first policy, power has priority over other system requirements. In the second one, performance has higher priority and in the third one, both power and performance have the same priority. Goal formulation can abstract these objectives into the goal at run-time and then switch between these three policies based on the environment requests.

## IV. Conclusion and Future Work

In this paper, we illustrated limitation of the state-of-the-art resource management policies. These policies strive to meet the fixed objective and can not dynamically adapt to the environment. To address this problem, we motivate the need for a goal formulation which can abstract the different requirements into the goal by a weighted combination of different objectives at run-time. The presented framework can be a good solution for effective resource allocation by considering applications requirements, user activity patterns, and system constraint (e.g., battery life). We believe that there are some challenges in implementing this goal formulation because of conflicting objectives and system constraints. The proposed framework can be a good initial point for goal formulation and goal driven management but it needs to find a theoretical solution that can formalize goals and switch between various policies subsequently with considering implementation and verification complexity. Therefore, there are some open problems for future work.

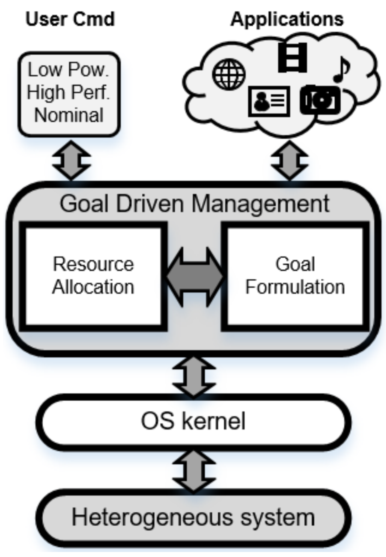

Fig. 4. A high-level architecture of run-time goal driven management framework.

## References

[1] A. Kanduri, A. Miele, A. M. Rahmani, P. Liljeberg, C. Bolchini, and N. Dutt, "Approximation-aware Coordinated Power/Performance Management for Heterogeneous Multi-cores," in *Proc. of the 55th Annual Design Automation Conference*, 2018, pp. 68:1–68:6.

[2] A. M. Rahmani, A. Jantsch, and N. Dutt, "Hdgm: Hierarchical dynamic goal management for many-core resource allocation," *IEEE Embedded Systems letters*, 2017.

[3] T. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin., "Hierarchical power management for asymmetric multi-core in dark silicon era," *In Proceedings of the 50th Annual Design Automation Conference*, p. 174, 2013.

[4] A. Kanduri, M. H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, N. Dutt, and H. Tenhunen, "Approximation knob: Power capping meets energy efficiency," *In Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.

[5] N. Peters, D. Füß, S. Park, and S. Chakraborty, "Frame-based and thread-based power management for mobile games on hmp platforms," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 169–176.

[6] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 79–88.