ARTE: an Application-specific Run-Time Management Framework for Multi-core Systems

Giovanni Mariani
ALaRI - University of Lugano
Lugano, Switzerland
E-mail: giovanni.mariani@lu.unisi.ch

Gianluca Palermo, Cristina Silvano, Vittorio Zaccaria
Politecnico di Milano
Dipartimento di Elettronica e Informazione
E-mail: {gpalermo, silvano, zaccaria}@elet.polimi.it

Abstract—Programmable multi-core and many-core platforms increase exponentially the challenge of task mapping and scheduling, provided that enough task-parallelism does exist for each application. This problem worsens when dealing with small ecosystems such as embedded systems-on-chip. In fact, in this case, the assumption of exploiting a traditional operating system is out of context given the memory available to satisfy the run-time footprint of such a configuration.

An efficient Run-time Resource Management (RRM) becomes of paramount importance to dispatch tasks to the cores by taking into account the task-parallelization options that each application provides. State-of-the-art approaches to RRM try to allocate resources to maximize the instantaneous throughput while meeting a power budget constraint. In this paper, we will show that queuing theory can be an alternative yet effective way of solving resource allocation by presenting ARTE, an Application-specific Run-Time management framework. The framework exploits few assumptions about the target many-core computing fabric such as profiling information about the platform applications. We will show that this information can be combined, at run-time, with queuing models to enhance the response time of the applications by pounding the actual effect on the system power consumption better than previous approaches.

Experimental results show that, compared to reference state-of-the-art RRM techniques, ARTE is able to efficiently improve system performance by pro-actively reducing the response time while meeting the same power consumption requirements. Besides, we will show that the run-time overhead of ARTE does not significantly impact neither the system performance nor the on-chip-memory occupation.

I. INTRODUCTION

The steady improvement of VLSI technology enabled a pervasive revolution of programmable systems by allowing multiple computing elements to be integrated within the same chip. Chip Multi Processors (CMPs) and Multi-Processors Systems on Chip (MPSoCs) are expected to become a de-facto standard in the next few years in both the general purpose and embedded computing arenas.

The availability of many processors (or computing elements) on a single chip brought new challenges to system designers; in particular, efficient power management has become a primary factor for product success with equal (if not more) impact on the value of the system perceived by the consumer. This is evident for portable devices but has subtle yet important consequences on reliability and cooling costs for non-portable systems.

Traditional techniques for power management in multi-core systems consider switching off or slowing down the frequency of computational elements which are underutilized [1], [2], [3]. In fact, if the switching overhead is negligible and the performance is not saturated, it is possible to meet performance requirements with fewer active resources and lower power consumption. In this context, the RRM acts by identifying, among other things, a suitable frequency and voltage to be assigned to each power island; practically, the RRM tunes a set of run-time knobs (or parameters) by employing a wide range of heuristics to minimize the power consumption and, ideally, maximize the predicted system performance. Overall, state-of-the-art RRM frameworks maximize the instantaneous throughput measured either in terms of instructions per second [1], [2] or, from a higher level point of view, in terms of tasks completed (or jobs) per second [4].

In this paper, we will exploit some queuing theory concepts as an alternative yet effective way of solving the resource allocation problem. Our technique, named ARTE (Application-specific Run-Time management), leverages few assumptions about the target many-core computing fabric such as profiling information about the platform applications. To introduce our approach, we point out that portable devices dedicated to interactive multimedia services may benefit up to a certain point from the above approaches since the performance is actually a minimum requirement which varies with the quality of service expected by the user (e.g., a smooth reproduction of video). Nevertheless, we pounded this consideration with the fact that, once performance requirements are met, the system response time becomes a more flexible function to be traded off to meet a pre-determined instantaneous power budget while not sacrificing performance on the long term (i.e., making other threads starve). In this new perspective, our solution carefully relaxes the average time between the instant a job arrives (e.g., a new video frame has to be decompressed) and the instant in which the job is adequately served (e.g., the frame has been actually decompressed in raw pixels stored in the frame-buffer) to satisfy the requirements on power. Restating the problem in terms of queuing theory, response time allows us to adopt policies for minimizing the overall queue length (or memory occupation) associated with the execution of each application.

ARTE aims at determining the relationships between power consumption, system performance and application parallelization during a design-time Design Space Exploration (DSE) phase. The information gathered at design-time (or at compile time) are then used at run-time to enable a queuing model-based processor allocation policy. Dynamic voltage and frequency scaling, DVFS [2], [3], is somewhat orthogonal to our methodology; since our focus is to explore the actual impact of application parallelization on the Quality of Service (e.g. throughput) and power, DVFS has been omitted from the analysis done in the paper. It is not difficult, however, to extend the proposed methodology to comprehensively address these additional dimensions.

The reminder of this paper is organized as follows. Section
II introduces a motivating example that unfolds the complexity of the problem of processor assignment to applications in a multi-core system. Section III introduces our mixed design-time/run-time optimization flow named ARTE while Section IV introduces experimental data supporting the efficiency of our approach with respect to other state-of-the-art approaches. Section V summarizes previous approaches to the problem of run-time resource management for multi-core systems, while Section VI.

II. MOTIVATING EXAMPLE

In this paper we consider a platform in which the platform throughput is typically limited (i.e. there is a limited amount of computational power requested by the user) and there is a power consumption budget to be allocated to the running applications. We would like to stress the fact that this is a pretty common scenario for multimedia portable devices.

As stated above, the fundamental problem to be solved by the RRM is to assign the available computing elements to a set of running applications (workload) in order to satisfy their throughput requirements, given the actual arrival rate of requests, and the allocated power budget.

In this paragraph, we would like to show that a greedy optimization of the instantaneous throughput may lead to a sub-optimal average response time on the long-term. To stress this point out, let us consider a chip-multi-processor composed of 16 out-of-order MIPS processors which communicate via a shared bus.

We consider a system with two running applications (called $\alpha_1, \alpha_2$) that process a certain amount of data in chunks (or jobs) whose rate depends on the interaction between the user and the system. We assume that neither of the two applications has strict real time constraints (thus they are served with a best effort strategy).

The task-parallelization (i.e., the number of cores assigned to each application for executing a single job) cannot be changed within the elaboration of the job itself but only between two distinct jobs. If an application is busy elaborating a job, all the successive jobs are maintained in a FIFO structure (we will call it queue in the remaining sections of the paper).

Let us assume that each application queue is initially set up with 15 jobs and that design-time profiling of the different binary versions of the applications provided us with the service rate and average power consumption data for 1, 2, 4, and 8 processors, as shown in Table I.

<table>
<thead>
<tr>
<th>Resource Allocated (no. of processors)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Rate $\alpha_1$ [Job/s]</td>
<td>20.5</td>
<td>37.3</td>
<td>44.6</td>
<td>49.8</td>
</tr>
<tr>
<td>Service Rate $\alpha_2$ [Job/s]</td>
<td>17.2</td>
<td>31.2</td>
<td>49.7</td>
<td>74.2</td>
</tr>
<tr>
<td>Power Consumption $\alpha_1$ [W]</td>
<td>0.8</td>
<td>1.7</td>
<td>3.2</td>
<td>6.1</td>
</tr>
<tr>
<td>Power Consumption $\alpha_2$ [W]</td>
<td>0.7</td>
<td>1.5</td>
<td>3.0</td>
<td>5.6</td>
</tr>
</tbody>
</table>

In order to show that maximizing instantaneous application-specific throughput may lead to suboptimal solutions on the long-term, let us consider two use-cases, Case 1 (C1) and Case 2 (C2). In both cases, we use a state-of-the-art greedy heuristic RRM (as proposed in [7]) that allocates processors such that the instantaneous sum of throughputs is maximized with a fixed budget on the instantaneous power consumption. The only difference between the two use-cases is that the power budget of C1 is 4.0 W while the power budget of C2 is 3.5 W; the other boundary conditions (such as an initial state of 15 jobs per-queue) are the same.

![Jobs to be completed in the systems for the cases in the example with two different power budgets.](image)

Effects of the greedy heuristic. In case C1, the RRM greedy heuristic assigns 1 processor to $\alpha_1$ and 4 processors to $\alpha_2$, serving, in the initial phase, 70.2 Job/s with 3.8 W instantaneous power. In particular, $\alpha_1$ performs at a speed of 20.5 Job/s for the first 340 ms and at a speed of 44.6 Job/s for the last 179 ms (average 28.8 Job/s) since, in the last phase, $\alpha_2$ has finished to run all jobs and becomes idle leaving more resources available to $\alpha_1$ which can run at full speed.

On the other hand, in case C2, the simultaneous throughput is maximized assigning 2 processors to both applications, serving 68.5 Job/s and 3.3 W of instantaneous power.

We can note that reducing the power budget in C2 decreases the overall instantaneous throughput. On the other hand, case C2 has a shorter overall completion time than C1 (in fact, completion of both application jobs happens at 455ms (C2) vs 520ms (C1), i.e., 12% earlier). As said above, C1 presents a maximum throughput sum of 70.2 Job/s (in the initial phase, when $\alpha_2$ is given 4 cores) but slows down the consumption of jobs from the queue of $\alpha_1$. This is a known **starvation effect** [8] that may arise when much faster applications compete for the resources with a priority that is somewhat inversely correlated to their job execution time. Please note that the problem cannot be efficiently solved replacing the throughput sum with the arithmetic, geometric or harmonic mean throughput for active applications [4]. In fact it would be possible to generate examples with different initial conditions that would lead to similar behaviors.

The problem with the above scenario is that maximizing a function of the instantaneous throughput does not take into account how many jobs are in the queues waiting to
be served (average waiting time in the queue). In fact, it could require more resources than those actually needed and imply an overall increased completion time. Conversely, we can say that the average response time, i.e., the average time a job spends in the system before being served by the platform, can bring, instead, to a reduction of the completion time (note that the shorter response time of C2 allows for a shorter overall completion time). We will exploit this basic observation to build the core contribution of this work.

III. ARTE

The proposed Application-specific Run-Time managEment framework (ARTE) aims at providing a suitable run-time policy for assigning processors to applications (whose performance is supposed to scale with the number assigned cores) that minimizes the response time while meeting a specified constraint on the power consumption. We assume that the number of cores assigned to each application for executing a single job cannot be changed within the elaboration of the job itself but only between two distinct jobs. In this paper, we use code versioning [9] to change the parallelization of the application.

In the proposed framework, the RRM decides how to allocate processors to active applications based on the following information:

- **The power budget**, which we assume to be set either at design-time or at run-time by the OS.
- **The user activity**, which generates, at run-time, a sequence of jobs to be served by the system with a set of minimum throughput requirements for each application. These requirements can be set according to system settings (e.g., the system is plugged into a power supply or not). The throughput offered by the system should at least match the requirement expected by the user. As an example, if the user is re-producing a video, he expects a smooth reproduction at 25 frames per second.
- **An application characterization** done at design-time by using a design space exploration heuristic. The application characterization consists of generating a set of operating points that specify, for a number of processor allocation configurations, the associated measures of throughput and power.

The overall framework is composed of a design-time and run-time parts as shown in Figure 2.

At design-time, given the target applications set which is supposed to permanently run on the platform, an exploration algorithm is used to identify a set of operating points that represent, for each application $\alpha$ in the applications set $A$, a tuple containing the parallelization of the application combined with its associated power consumption $\psi_\alpha$ and application throughput $\mu_\alpha$. The tuple can be extended with information regarding other run-time parameters (such as, for example, the optimal frequency of the processors running the application). Figure 3 shows a typical representation of operating points, where each configuration of allocated processors $\pi$ is associated with the corresponding power consumption $\psi_\alpha$ and application throughput $\mu_\alpha$.

Given the operating points gathered at design time (i.e., the application characterization), the RRM main task is to select the number of processors to be allocated to each application to meet the power budget and to offer the required throughput. We point out that, given the finite amount of memory used for storing jobs in the system, job queues should be consumed at an average rate which is at least equal to the throughput required by the user (otherwise the system dynamic behavior is not stable, as we shall see later). Besides, a finite response time in the steady operation of the system implies a finite and predictable memory occupation and a minimum required throughput. Whenever the required throughput cannot be sustained for the long-term (because of a significant divergence with respect to the design-time assumptions), corrective actions should be taken (such as, for example, dropping jobs) to maintain stability. This may be perceived by the user as sluggishness in the user interaction or multimedia reproduction.

ARTE addresses the maximization of quality of service (QoS) given the instantaneous power budget constraint. In particular, for each application $\alpha$, we define the QoS $Q_\alpha$ as the inverse of the average response time $R_\alpha$, i.e., the average time measured between the submission of a job to the application (whether it be a frame to be decoded or an audio packet to be processed) and the time the job is completed (thus, including the average waiting time in the queue). We define the overall platform QoS $Q$ as the geometric mean of the set $A$ of permanently running applications:
The geometric mean, unlike the arithmetic mean, tends to dampen the effect of very high or low values, which might bias the mean if a straight average (arithmetic mean) were calculated [10]. Indeed this is our case since we deal with a set of heterogeneous applications with very different workloads and throughput ranges. The actual response time \( R_\alpha \) is known only after the application execution is completed thus, to enable run-time management, ARTE maintains and evaluates, at run-time, a prediction model of the expected average response time \( \hat{R}_\alpha \).

**Prediction of the response time using queuing models.**

In this paper, we propose to model \( \hat{R}_\alpha \) by using fundamental results coming from the theory of queuing networks [11]. The model is based on the following assumptions:

1) The job arrival rate \( \lambda_\alpha \) of an application \( \alpha \) is the throughput required by the user. It is measured in Job/s and it depends on the user activity to be monitored at run-time. An application is considered active if \( \lambda_\alpha > 0 \).

2) The job arrival rates \( \lambda_\alpha \) of different applications are mutually independent.

3) The job arrival times can be modeled as continuous time Markov process, i.e., for a single application, they are mutually independent. In particular job inter-arrival times are exponentially distributed with mean \( 1/\lambda_\alpha \).

4) The average job execution time \( 1/\mu_\alpha \) for an application can be measured at design time via simulations. This information is part of the application characterization done at design-time and exposed at run-time to ARTE (measured in Job/s). In the following we refer to the average job execution time as the service time.

5) For a given application, before starting the execution of a new job, all previous jobs should be completed.

6) All jobs should be served and no starvation is allowed. Thus, in the steady state, the average application throughput \( \mu_\alpha \) is equal to the average arrival rate \( \lambda_\alpha \).

**Fig. 4.** Queuing system for a single application.

The queuing model used in this paper is composed of queues and servers. The queue, in our case, is a structure residing in the system memory in which the jobs are stored waiting to be executed while the server is the set of system resources required for processing the jobs (i.e., the processors allocated to an application). Given that the jobs should be executed in order (Assumption 5, in the list above) and that arrival rates are mutually independent among applications (Assumption 2), we model each application with an independent queuing system with one server and one queue (Figure 4).

We assume that the arrival of jobs can be modeled as a Markov process (Assumption 3) with a deterministic service time (M/D/1, i.e., Markov/Deterministic/1 server). In the M/D/1 model [11], the expected number of jobs in the system (either waiting in queue or being served) in the steady state associated with application \( \alpha \) is given by:

\[
Q = \left( \prod_{\alpha \in A} Q_\alpha \right)^{\frac{1}{|A|}} = \left( \prod_{\alpha \in A} \frac{1}{R_\alpha} \right)^{\frac{1}{|A|}}
\]  

(1)

\[
N_\alpha(\mu_\alpha, \lambda_\alpha) = \rho_\alpha(\mu_\alpha, \lambda_\alpha) + \frac{\lambda_\alpha^2}{2(1 - \rho_\alpha(\mu_\alpha, \lambda_\alpha))}
\]

(2)

\[
\rho_\alpha(\mu_\alpha, \lambda_\alpha) = \frac{\lambda_\alpha}{\mu_\alpha}
\]

(3)

where \( \rho_\alpha \) is the server utilization, i.e., the fraction of time the server is busy. Given equations (2) and (3), we can build a prediction model for \( \hat{R}_\alpha \) by using the Little’s Law [11]:

\[
\hat{R}_\alpha(\mu_\alpha, \lambda_\alpha) = \frac{N_\alpha(\mu_\alpha, \lambda_\alpha)}{\lambda_\alpha}
\]

(4)

We can note that \( \hat{R}_\alpha \) actually depends only on the service rate \( \mu_\alpha \) and the arrival rate \( \lambda_\alpha \), all of which are attributes that are either profiled at design-time or measured at run-time by the ARTE framework. In particular, the service rate \( \mu_\alpha \) is characterized at design-time for each available parallelization \( \pi \).

**Formalization of the run-time management problem.**

For each application \( \alpha \) and each parallelization \( \pi \), we define an operating point \( \nu_\alpha^\pi \) as the following tuple:

\[
\nu_\alpha^\pi = (\psi_\alpha^\pi, \mu_\alpha^\pi)
\]

(5)

Where \( \psi_\alpha^\pi \) is the average power consumption of the application \( \alpha \) and \( \mu_\alpha^\pi \) is the application throughput for the given parallelization \( \pi \). Each operating point is fully characterized at design time.

We assume that each application \( \alpha \) has a set \( \Pi_\alpha \) of code-versions, each version providing a different parallelization \( \pi_\alpha \in \Pi_\alpha \). We assume an operating point \( \nu_\alpha^\pi \) for each code-version and we call \( N_\alpha \) the set of all possible operating points for application \( \alpha \). We assume that for each parallelization \( \pi_\alpha \) of the application \( \alpha \), the values of \( \psi_\alpha^\pi \) and \( \mu_\alpha^\pi \) can be characterized at design time via simulation. This characterization, in the form of data structure containing profiling information, can be used dynamically by the RRM.

In the remaining part of the paper, we will indicate with boldface the vectors that contain all the arrival rates and parallelizations used for each application running on the system (\( \lambda \) and \( \pi \)). As an example, given two permanently running applications in the system (\( \alpha_1 \) and \( \alpha_2 \)), the vectors:

\[
\pi = \begin{bmatrix} \pi_{\alpha_1} \\ \pi_{\alpha_2} \end{bmatrix}, \quad \lambda = \begin{bmatrix} 12.2 \\ 29.8 \end{bmatrix}
\]

(6)

indicate that \( \alpha_1 \) is given 4 processors with estimated job arrival rate equal to 12.2 Job/s while \( \alpha_2 \) is given 2 processors with an estimated arrival rate of 29.8 Job/s.

The estimated QoS (Equation 1) is a function of the response time \( \hat{R}_\alpha \) of each application. Please note that, in the previous section, we have introduced a set of assumptions under which the \( \hat{R}_\alpha \) depends on \( \mu_\alpha^\pi \) and on the arrival rates of each application \( \alpha \). In particular, \( \mu_\alpha^\pi \) is a function of the parallelization currently used for \( \alpha \). Thus we can rewrite the QoS as a function of overall arrival times \( \lambda \) and the parallelizations \( \pi \) associated with each running application:

\[
Q \sim \hat{Q}(\lambda, \pi)
\]

(7)

We can formalize the run-time management problem as finding the optimal \( \pi \) that maximizes \( \hat{Q}(\lambda, \pi) \), given an
instantaneous set of arrival rates $\lambda$ and a set of constraints on the maximum power consumption and the number of processors available. More formally, the problem is:

$$\text{Find } \pi \text{ such that } \hat{Q}(\lambda, \pi) \text{ is maximum,}$$

$$\sum_{\alpha \in A} \psi(\nu_\alpha^\pi) \leq P, \quad (9)$$

$$\sum_{\alpha \in A} \pi_\alpha \leq R \quad (10)$$

where $P$ is the power-budget constraint set by the OS, $R$ is the total amount of homogeneous computing resources available on the multi-core platform and $\psi(\nu_\alpha^\pi)$ is the power consumption associated with the operating point $\nu_\alpha^\pi$ (we note that, given a parallelization $\pi$, we will enforce, during design-time characterization, that only one operating point $\nu_\alpha^\pi$ is associated with it). Equation 8 represents a constrained single objective combinatorial optimization problem which is instantaneous once the arrival rates $\lambda$ are known. In particular, we assume that $\lambda$ is unknown at design-time and it is inferred at run-time from the user activity. To maximize the QoS, we profile the arrival rates at run-time and we solve the optimization problem periodically across time windows. Given the randomness in the arrival time distribution, a straightforward profiling of $\lambda$ might be subject to some drastic variations among time windows (especially when the window is small). We thus introduced a moving-average filter to smooth this effect.

In the next section, we will describe how the optimization problem above can be efficiently solved at run-time by ARTE.

**Run-time resource management heuristic for ARTE.**

Algorithm 1 shows the algorithm used by ARTE to solve the problem presented in Equation 8. The algorithm is activated periodically (considerations on the period size will be presented later) in order to fit with the dynamically changing user activity profiled in terms of average job requests per time unit i.e. $\lambda$. Then, for each application (steps 3-6), an initial minimal parallelization is identified to keep the actual throughput satisfying the arrival rates $\lambda$. The heuristic then tries to improve the QoS by exploiting additional processing elements or some available power within the budget (steps 8-12). Finally each application parallelization $\pi_\alpha$ is set at step 13.

**Algorithm 1 Resource allocation routine for ARTE**

1. **for** each $\alpha \in A$ **do**
2. \hspace{1em} Update $\lambda$
3. \hspace{1em} $\pi_\alpha = 1$
4. \hspace{1em} while $\mu(\nu_\alpha^\pi) < \lambda_\alpha'$ **do**
5. \hspace{2em} Increase $\pi_\alpha$ to the next value in ascending order of $\Pi_\alpha$.
6. \hspace{1em} **end while**
7. **end for**
8. **while** $\exists \alpha, \phi(\alpha)$ **do**
9. \hspace{1em} $A_f = \{ \alpha \in A | \phi(\alpha) \}$
10. \hspace{1em} $\pi_f = \text{argmax}_{\alpha \in A_f} \gamma(\alpha)$
11. \hspace{1em} Increase $\pi_{\pi_f}$ to the next value in ascending order of $\Pi_{\pi}$
12. **end while**
13. Load target parallelizations

The run-time heuristic adopted (steps 8-12) is a gradient-based search that iteratively increases the parallelism $\pi$ allocated to the running applications. The search iterates until it exists an application $\alpha$ whose parallelization $\pi_\alpha \in \Pi_\alpha$ can be increased (without incurring in an unfeasible solution) to the next available value $\pi_\alpha' \in \Pi_\alpha$ (we consider that $\Pi_\alpha$ has ascending order of parallelizations). The predicate $\phi(\alpha)$ verifies that the solution is feasible, i.e., the overall power and resource usage does not exceed neither the power budget $P$ nor the available resources $R$:

$$\phi(\alpha) = \left[ \left( \psi(\nu_\alpha^\pi) + \sum_{\beta \in A - \{\alpha\}} \psi(\nu_\beta^\pi) \right) < P \right] \land \left[ \left( \pi_\alpha' + \sum_{\beta \in A - \{\alpha\}} \pi_\beta \right) < R \right]$$

The objective function of the search is the gain per unitary resource $\gamma_\alpha$:

$$\gamma_\alpha = \frac{\hat{Q}(\pi_\alpha') - \hat{Q}(\pi_\alpha)}{\pi_\alpha' - \pi_\alpha} \quad (11)$$

where $\gamma_\alpha$ quantifies the positive benefits, for application $\alpha$, to switch to a greater parallelization $\pi_\alpha'$. Step 8 considers the maximum improvement achievable for any application. The application $\pi_f$ that offers the maximum improvement is selected (step 10) and its parallelism increased (step 11).

**IV. Experimental Results**

In this paper, we address an embedded CMP platform composed of a general purpose processor (host processor) and an application-specific computing fabric consisting of a set of computing elements. The host processor provides flexibility to run a common operating system, while computing elements serve as relatively simple processors to run the task parallel applications. The ARTE algorithm is meant to be integrated in the general purpose OS to dispatch application tasks over the available processors of the computing fabric. In this sense, the host processor acts as a Fabric Controller. We consider here a computing fabric which is composed of 16 MIPS-like processors (Figure 5) with a shared memory architecture, but the approach is general enough to be applied to a relatively larger computing fabric with a distributed memory.

![Fig. 5. Overview of the target Chip Multi Processor.](image_url)
To ensure the coherence of shared data in the memory hierarchy, this protocol generates invalidate/write requests between L1 and L2 caches.

To estimate system-level metrics, we leveraged the SESC simulation tool [5], a well-known MIPS instruction set simulator for CMPs providing dynamic power ψ and performance μ, associated to the execution of a job of a user-selected application α with thread-level parallelism πα.

To provide a fairly broad evaluation of the proposed approach, we leverage the SPLASH-2 parallel benchmark suite [6]. In particular we focused our attention on the following application kernels: Complex 1D FFT (fft) Integer Radix Sort (radix), Ocean Simulation (ocean) and Blocked LU Decomposition (lu). For brevity, we will use the symbols α1...α4 to indicate each application.

Each application has been task-parallelized manually for a number of processors varying as a power of 2, as it is common for such kind of evaluations [12].

The optimal size of the periodic windows, at the end of which ARTE is invoked, is expected to be platform and scenario dependent. In particular, we set the RRM period to the lowest arrival rate expected for the applications under study λmin. This setting provided good stability of the run-time profiling of λ.

Evaluation of the queue model. Before analyzing the efficiency of the proposed framework, we evaluate validated the model of the response time as presented in Section III.

In particular, we are interested in verifying that the estimation of the response time is robust with respect to the interference given by the fact that different applications are running on the same platform with job bursts of arbitrary size.

To do so we considered a number of different combinations of computing resources allocation and random arrival rates for the target benchmarks (given the amount of applications and computing resources, we considered 360 as a reasonable minimum number of simulations suitable for such evaluation). During the simulation campaign, we simulated with SESC the average job response time Rα and we compared it to the one estimated by Rα as introduced in Equation 4. The measured mean relative error of the queuing model has shown to be 6.5% for α1; 6.8% for α2; 6.5% for α3; and 7.2% for α4. Given the entity of the error, we are confident that the queue-based model of the response time is accurate enough for our purposes.

Evaluation of ARTE. The proposed ARTE RRM methodology has been compared against two state-of-the-art approaches:

- The first approach consists of the maximization of the actual throughput (maxT) as presented in [7]. The heuristic maximizes the actual throughput \( \mu = \sum \mu(\nu^\alpha) \) considering the constraints in terms of total resources and power budget. Similarly to ARTE, this approach needs a design-time application characterization.

- The second approach is derived from the pull-high, push-low (PHPL) heuristic as presented in [2]. Basically, the PHPL policy periodically verifies the power consumption of the different applications and decides how to allocate resources by decreasing the number of allocated processors of application that consume the highest power whenever the power budget is not met.

To evaluate the performance of ARTE, maxT and PHPL, we generated 20 different input sequences of job arrivals by modulating the job arrival rates for each application by using a Markovian random process.

For the input sequence, we investigated the behavior of the three RRM schemes by varying the power budget constraint (up to 12W). Figure 6 presents the average system performance in terms of: power consumption, job response time and queue lengths (maximum and mean) by varying the maximum power budget as a percentage of the maximum allowed.
directly correlated with the on-chip memory requirement of the system which is a particularly scarce resource on embedded systems.

We also note that, when the power budget is relatively high, the response time of ARTE is very close to the one of maxT. This happens because a higher power budget can be used to exploit more computational resources. The proposed workload becomes thus relatively low and fewer jobs are in the system (on average). Thus, when a new job arrives it has a higher probability to find an empty queue and can be efficiently allocated using the maxT policy as well.

We may note that maxT allows for performance degradation when increasing the power budget from 58% to 67% of the maximum power (respectively 7W and 8W). This might be justified by the fact that maxT tends to create an unbalanced workload by making some low-throughput application starve (as observed in [4], [8] and Section II). In particular, considering a power budget of 8W, maxT provides 8 cores to α4, 2 cores to α3 and one core to α2 and α1 while, when the power budget is decreased to 7W, α4 is given only 4 cores dramatically increasing the resources available to the other three applications.

**In-depth study of RRM behavior.** To provide more insight in the run-time behavior of the proposed approach, we analyzed the case in which the power budget is set to 9W. Figure 7 shows the arrival rate, QoS, mean queue length and power consumption as a function of time.

As Figure 7(a) shows, we can divide the simulated timeframe in three periods: a first period characterized by a high arrival rate of α4 (0s-25s); a period in which the arrival rate of α4 decreases (25s-45s); a period in which α3 arrival rate increases (45s-60s).

Concerning QoS (shown in Figure 7(b)), we can note that ARTE performs slightly better than the traditional approaches in the first period. The second period is instead characterized by low arrival rates (thus, both maxT and PHPL are able to perform as good as ARTE).

In the last period, the arrival rate of α3 requires many resources to be allocated to the application. In this case, ARTE allocates all the resources to provide the requested QoS. PHPL instead distributes resources equally among applications and it is not able to serve with appropriate resources the needs of α3. maxT is in the same league since the maximization of the actual throughput suggests to increase the parallelization of applications other than α3. In fact, given a 9W power budget, maxT provides 2 cores to α1, 2 cores to α2, 8 cores to α4 and only 1 core to α3. For this reason the queue of α3 grows together with the response time while reducing the QoS. This effect is visible in Figure 7(c).

We note that when the job arrival finishes at 60s, ARTE almost immediately finishes its pending jobs (given the steady reduced size of the queues). The other two RRM still have to consume jobs from the queue to finish about 5 seconds later (a very high response time). Besides we can note that the instantaneous power consumption is, on average, below the power budget (see Figure 7(d)).

**Overhead Evaluation.** We evaluated the run-time overheads of the different RRM approaches when executed on the target MIPS based host processor (that we assume running at 300MHz). We noted that ARTE’s usage of an analytical queue model and the greedy heuristic introduce some additional cost with respect to PHPL, which is a pretty straightforward heuristic. We can finally note that, for all RRM approaches, the mean overheads are below 0.3 mJ for the energy and 0.2 ms for the time. Given the time window considered in this paper, we consider the overheads as negligible.

V. RELATED WORK

In a multiprogrammed multi-core scenario, different applications compete to access the system resources making the overall platform behavior very complex and sometime counter-intuitive. For example, the authors of [13] introduced experimental evidence about performance gains when some of the system processors are left idle to address unpredictable workload instead of being assigned to the active applications. The authors observed that the efficiency of different RRM schemes is strongly dependent on application specific features.
such as the scaling of the performance with respect to the number of allocated resources.

Among other works, the authors of [14] observed that, for multiprocessor workloads, the Instructions Per Cycle (IPC) is a poor performance index and does not automatically reflect performance as a whole. It has also been noted that for multi-programmed multi-core scenarios, system level performance metrics such as throughput measured in terms of Job/s or other user-oriented performance metrics are more suitable for maximizing performance [4].

**RRM heuristics.** So far, several works appeared in the literature assume that the applications set is known at design-time (either in the virtual prototyping stage or earlier). In [15], [16], the authors introduce an RRM that solves a multi-dimension multiple-choice knapsack problem at run-time. In particular, at design-time, different operating configurations (e.g., allocated processors) are simulated for each application and several indices such as average throughput and power consumption are saved. At run-time, once the set of active applications is given, the RRM heuristically chooses for each application one of the operating configurations previously identified. The authors of [17] extend [15] by applying it to an industrial MPSoC scenario and reducing further the run-time overhead of the heuristic. In [7], a joint design-time/run-time DSE methodology has been proposed where the run-time operating configurations are chosen together with an optimal tuning of architectural parameters.

Other approaches appeared in the literature addressed unknown relationships between resource allocation and system performance. These relationships are learnt at run-time by fitting an analytical model to observed system-level indices. In [18] a piecewise linear regression model is used for modeling the relationships between resource allocation, application performance and energy consumption while [19] proposed machine learning approaches such as Artificial Neural Networks and Q-learning. Among such works, we may also cite [20] that proposes predictive models for run-time adaptation addressing multi-threaded scientific applications. We would like to stress the fact that the construction of complex analytical models at run-time is computationally expensive and, in order to be done efficiently, it may require ad-hoc hardware.

**VI. Conclusions**

In this paper we proposed an Application-specific Run-Time Management framework (ARTE) embedded multi-core systems. To address long-term performance, ARTE maximizes the platform QoS measured as the inverse of the response time by exploiting a queuing model that is characterized by application specific information collected during the design-time analysis. ARTE has surpassed state-of-the-art approaches to RRM by considering response time instead of simultaneous throughput. Eventually, a coarse-grained analysis of the run-time overheads has shown that ARTE’s overhead is still acceptable with respect to simpler greedy approaches which are, however, less effective in terms of long-term performance.

**Acknowledgments**

The work has been partially funded by the European Community under the FP7-ICT-248716-2PARMA, FP7-ICT-247999-COMPLEX and ARTEMIS-100230-SMECY projects and the Swiss Hasler Foundation under the Project EMME (Grant No. 11096).

**References**


