Journal of Systems Architecture 000 (2017) 1–15

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms

Houssam-Eddine Zahaf[a,b,*], Abou El Hassen Benyamina[b], Richard Olejnik[a], Giuseppe Lipari[a]

[a] Université Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique, Signal et Automatique de Lille, F-59000 Lille, France
[b] Oran1 University, Ahmed BenBella, Algeria

ARTICLE INFO

Article history:
Received 26 April 2016
Revised 25 November 2016
Accepted 6 January 2017
Available online xxx

Keywords:
Moldable tasks
Real-time
DVFS
DPM
Energy consumption
Partitioned scheduling
Frequency selection
Task allocation
INLP

ABSTRACT

In this paper, we address the problem of executing (soft) real-time data processing applications on heterogeneous computing platforms with the goal of reducing the energy consumption. The typical application domain is *edge computing* (or *fog computing*), where a certain amount of data, collected from the environment, needs to be pre-processed in real-time before being sent to the central server for storage and final processing. The kind of applications we address here can be easily parallelized, and we also need to reduce as much as possible the necessary energy to perform such tasks. Heterogeneous Multicore Processors (HMP) such as ARM big.LITTLE are designed to combine both performances and energy efficiency, so they are one of the preferred choices for this kind of applications. Here we focus on Dynamic Voltage and Frequency Scaling (DVFS), parallelization, real-time scheduling and resource allocation techniques. In the first part of the paper, we present a model of the performance and energy consumption of a parallel real-time task executed on an ARM bigLITTLE architecture. We use this model in the second part of the paper where we first define the optimization problem as an Integer Non-linear Programming (INLP) problem, and then propose heuristics for efficiently solving it. Finally, we present a wide range of synthetic experiments that demonstrate the performance of our approach.

1. Introduction

Energy consumption is an increasing concern in *cyber-physical* real-time systems, especially when processing elements operate on battery power. Embedded real-time distributed systems must support increasingly complex applications such as distributed video surveillance, processing a large amount of sensor data, etc.

To reduce the network traffic, instead of collecting and sending all data to a remote central server, part of the processing is done on-site with multicore embedded computing boards, so that only a small part of pre-processed data needs to be sent. For example, a surveillance system analyses the full video frames on-site by extracting the important features, and sends only the features for further processing at the central site. This model of computation is now known as “Fog-computing” [1] (also called “Edge-computing”).

Many of these applications can be easily parallelized by distributing data across the parallel computing elements. Reducing energy consumption in these systems is a very serious problem when they are operated by batteries. However, even when they are connected to the electric grid, we need to keep the consumption as low as possible. Heterogeneous multicore technology can help us in achieving timeliness and low energy consumption: in fact, even when the computational load is not very high, multicore processors are more energy efficient than an equivalent single-core platform [2]. The idea is to operate the system at a “lower frequency”, set some cores into a deep power state and, at the same time, reduce task’s response time by decomposing a sporadic task into parallel threads to be partitioned across the active cores.

Task decomposition is a well-known problem in the parallel programming community. For example, OpenMP [3] is an API specification for parallel programming. The sections of code that are meant to run in parallel are marked with pre-compiler directives. One example of such a directive is *pragma parallel for*. In the OpenMP parallel model, a parallel ***for*** loop preceded by the STATIC schedule clause is implemented by distributing the N iterations on p threads into approximately $\frac{N}{p}$ iterations per each thread (if no further specification is provided). After the execution

* Corresponding author.
E-mail addresses: houssam-eddine.zahaf@univ-lille1.fr (H.-E. Zahaf), benyamina.abouelhassen@univ-oran1.dz (A.E.H. Benyamina), Richard.olejnik@univ-lille1.fr (R. Olejnik), Giuseppe.lipari@univ-lille1.fr (G. Lipari).

http://dx.doi.org/10.1016/j.sysarc.2017.01.002
1383-7621/© 2017 Elsevier B.V. All rights reserved.

of the parallel code, the threads join back into the master thread, which continues onward to the end of the program. However, this task decomposition does not take into account scheduling, the respect of the real-time and energy constraints.

Parallel real-time task models are classified according to the way the level of parallelism is specified and to the moment of this specification. We can distinguish three types of models:

- *rigid*: the number of processors assigned to a task is specified independently before the scheduling analysis and does never change;
- *moldable*: the number of processors assigned to a task is specified off-line using an *off-line analysis algorithm*, for example during pre-processing or compilation;
- *malleable*: similar to moldable, but the number of processors assigned to a task may dynamically change during execution.

According to [4,5], most parallel applications in the real world are moldable. In this work we focus on moldable real-time tasks.

In order to achieve an optimal decomposition with respect to the energy consumption for a set of tasks on heterogeneous multi-core platform, we need to (i) set the operating frequency of cores; (ii) decompose each task into a set of parallel threads (if possible/desirable); (iii) perform a schedulability analysis and allocate the threads onto the cores to guarantee that each thread completes before its deadline.

Organization. This paper is structured as follows. In Section 2, we present the related work. In Section 3, we propose a model of a heterogeneous architecture, and a benchmarking technique for measuring the computation time and the energy consumed by a task when executed on processor(s) with a certain operating frequency. Using our technique, we obtain a mathematical model of the energy and performance profile of a task.

Then, in Section 3.3 we propose a new task model for parallel tasks. In our task model, a task is a collection of alternative *cut-points*, and each cut-point is a set of parallel threads. A cut-point represent a possible implementation of the task into a set of parallel threads.

We proceed by formalizing the problem of scheduling and allocation as an Integer Non-Linear Programming (INLP) problem in Section 4.1 and we confirm that the problem complexity explodes even for medium-sized task sets. We then propose a scheduling heuristics in Section 4.2 and we show, by performing a wide set of experiments in Section 5, that our heuristic performs better than existing heuristics.

2. Related work

In the real-time systems literature, many papers [6–10] focus on static and dynamic voltage and frequency scaling for uni-processor architectures, and some researchers proposed similar techniques for homogeneous multiprocessor architectures. Some recent works [11–14] focus on heterogeneous multi-core architectures.

Bini et al. [6] proposed a semi-linear model for modelling the worst-case execution time of a task as a function of the frequency, which we reuse in our own task model (see Section 3.3). They also proposed an algorithm to set the minimum processor speed on single processor architectures, so that all the deadlines are respected. Concerning real-time parallel task scheduling on multi-processors, many works (e.g.[15–21]) focused on intra-task parallelism, without considering energy consumption.

Kato et al. [17] adapted the *gang task* model to a gang real-time task model. In this model a task is defined by the number of processors used simultaneously, its execution time, period, and constrained deadline. All threads of a parallel section in a gang model have to be run in parallel and at the same time they must start and end the execution on their processors simultaneously. In contrast to classic global EDF, in gang EDF a high priority task at time t may not be run at that time if the number of available processors is less than the number of the parallel threads of this task. This model is difficult to implement, because the scheduler needs to synchronize the execution of the threads among different processors including possible preemptions.

A different model, easier to implement, is the *multi-thread* model, where parallel threads are scheduled independently, and in this work we will adopt it. Lakshmanan et al.[15] proposed a *fork-join* model for representing parallel tasks: each task is a sequence of alternating parallel and sequential *segments*. Saifullah et al.[21] proposed a parallel task model where a task is composed of segments, and each segments consists of a set parallel threads with the same real-time characteristics (e.g. release time, execution time, deadline). At the end of each segment, parallel threads must synchronise. They also proposed a general method to express all models proposed in [15,16]. They use global EDF and partitioned DM for scheduling. They assume that their work can be used on uniform processors with different speeds, and that the thread execution time scales inversely on speed. However, they ignore the memory effect on execution time variation. Courbin and Goossens [16] proposed a less restrictive model where parallel sub-tasks of each *phase*[1] have the same real-time characteristics except for the execution time. They state that the model proposed in [21] is a specific case of their model. They also proposed an algorithm to assign real-time parameters to a fork-join model in order to express it in their model, called *Multi-phase Multi-Thread*.

Jing Li et al. [22] proposed a federated scheduling approach for parallel real-time task scheduling. They considered a general task model (directed acyclic graph DAG to express the data dependency) with implicit deadlines. The federated scheduling algorithm proposes to divide tasks into two disjoint sets, the high-utilisation tasks ($u \geq 1$) and the low-utilisation tasks ($u < 1$). The low-utilisation tasks are run in competition with each other on shared cores using well-know multiprocessor scheduling algorithms. The high-utilisation tasks are run each on a dedicated core. However, this is not always the *optimal* choice when the goal is to optimize the energy consumption.

All these works [15,16,21,22] only address the scheduling problem of parallel task, without considering the energy consumption and assume a fixed decomposition of a task to a set of parallel threads. However, parallel threads *size* may be adjusted. For example, a parallel **for** loop with 100 iteration may be decomposed into two threads with each 50 iteration, or the first with 80 iteration and the second 20 iteration or any other decomposition that ensures that the sum of iterations is 100. This may allow more flexibility in scheduling. Our task model allows specifying numerous alternative parallel decompositions without any particular restriction.

Paolillo et al. [23] defined the *optimal* frequency for minimizing energy consumption on homogeneous platforms with gang *malleable* tasks. They target homogeneous processor platforms with the ability of turning off some processors. They considered sporadic implicit deadline tasks and used the canonical parallel scheduler proposed in [20], which is an optimal scheduling algorithm for sporadic tasks on homogeneous multiprocessor platforms.

Colin et al. in [14] proposed a partitioned-EDF heuristic to allocate implicit deadline tasks on Single-ISA heterogeneous multicore architecture, such as the ARM big.LITTLE architecture, with the goal of minimizing the energy consumption. They present several experiments on Exynos 5410,5420,5422 ARM big.LITTLE processors. Our model of the architecture is similar to their model. However,

[1] A phase is equivalent to a *segment*.