Partitionned Scheduling for Moldable Real-Time Tasks

Houssam Eddine ZAHAF & Giuseppe LIPARI
Richard OLEJNIK & Abou ElHassen BENYAMINA
University of Lille1, University of Oran1
{houssam-eddine.zahaf, giuseppe.lipari, richard.olejnik@univ-lille1.fr}
{benyamina.abouelhassen@univ-oran1.fr}

Abstract

Cyber-physical systems are compound of a set of computational elements controlling physical entities. These elements operate, almost, on battery power and run intensive real-time applications. Thus, The power must be managed in order to extend the battery life while granting the respect of the real-time requirements of all applications.

In this paper, we present a multi-thread moldable real-time task model and partitioned heuristic for uniform architecture.

keywords: Moldable tasks; real-time; partitioned scheduling; parallel, OpenMP, decomposition

Introduction

Computationally intensive real-time applications (i.e. real-time image processing, pattern recognition, and in general every processing of a large amount of data in real-time), require powerful computing platforms. Many of these applications can be easily parallelized by decomposing their execution into a set of parallel threads running the same algorithm on different subsets of data, and then integrating the results. This process is called task decomposition (Figure 1).

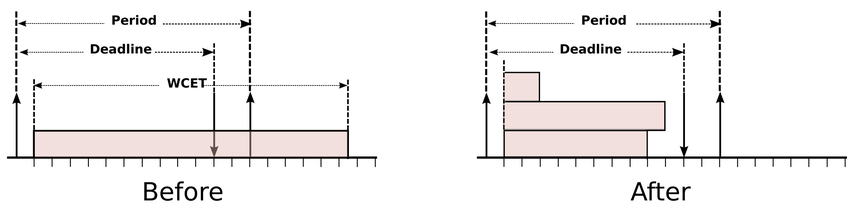

Figure 1: task decomposition

Main Objectives

1. Grant the respect of all deadlines;
2. Decompose the task execution;
3. Minimiz the energy consumption.

System Model

We consider a set of n synchronous periodic moldable tasks. Each task τ_i is characterized by tuple $\tau_i = (\mathsf{T}_i, \mathsf{C}_i, \mathsf{D}_i)$, where

- T_i: is the *period*, that means the time between the releases of two consecutive instances of the same task.
- C_i is the worst-case execution time *WCET*, assuming a core speed equal to 1. We assume that the WCET scales uniformly with the core speed ([2].
- D_i is the task's relative deadline.

Under the assumptions:

1. **Free to cut task model**
   - Each task can be decomposed at any point of it's code into an arbitrary number of threads
   - The load (execution time) can be different from a thread to another for the same task
   - The sum of execution time of all threads of the same task is equal to the initial execution time of the task.
2. **Hardware architecture**
   - Plateform is uniform multi-core;
   - Same instruction set & architecture for all cores
   - Each core has its own frequency
   - We define the speed of a core j as $s_j = \frac{Fmax}{f_j}$

We use demand bound function as an analysis tool

1 Demand Bound Function

let t be a non-negative integer. The *Demand bound function* $dbf(\mathcal{T}, t)$ denotes the maximum cumulative execution requirement that could be generated by jobs of $\mathcal{T}$ that have both ready times and deadlines within any time interval of duration $[0 - t]$.

$$dbf(\mathcal{T}, s, t_0) = \sum_{\tau_i \in \mathcal{T}} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \frac{C_i}{s} \leq t_0 \tag{1}$$

To be schedulable under EDF [1], the *dbf* value of a task set $\mathcal{T}$ at each moment of time $t_0 \in [0 - t^*]$, must be less then t_0. For uniform architecture, the dbf value can be expressed by modifiying the execution time to be adapted on different speeds (equation (1)).

2 Partitioning

we consider partitioned-EDF scheduling We propose iterative heuristic for solving the task decomposition and allocation problems.

At each iteration, the heuristic selects a task and a core, and tries to allocate the task into the current core. If the task can not be allocated, it defines the execution time of the thread (a part of the task) that will be allocated on the current core and defines execution time of the sub-task (*the rest*) to be be queued with the task list in order to be allocated in the next iteration.

Let $\mathcal{T}_j$ be a task set already allocated on the selected core j and τ the selected task to be allocated on this core. First we check the schedulability of the new task set $\mathcal{T}_{new} = \{\mathcal{T}_j \cup \{\tau\}\}$. So, We check $\forall t, t \in [0 - h]$, the equation 1.

1. If it is verified then the task is allocated on the core j;
2. If, for some t values, Equation (1) is not verified, then the task set $\mathcal{T} \cup \{\tau\}$ is not schedulable. In this case, we seek to split the task in order to the reduce the execution time of task τ so that it will fit between the dbf and t, and then equation 1 is verified. We call the time to be reduced *excess-time*. To evaluate the excess-time, first we evaluate the cumulative excess of all instances on the task form 0 to t $(excess = dbf(\mathcal{T}_p \cup \{\tau\}, s_p, t_0) - t)$. so the *excess-time* per one instance is equal to the ratio of the excess by the number of instances of the task τ in the interval $[0, t]$:

$$\begin{cases} excess = dbf(\mathcal{T}_j, j, t) - t_0 \\ \#instances = \dfrac{t_0}{T_i} \\ \textit{excess-time}(t_0) = \lceil \dfrac{excess}{\#instances} \rceil \end{cases} \tag{2}$$

So, For all values of t_0 , where condition (equation 1) is not verified, *excess-time* is evaluated and only the maximal *excess-time* is maintained.

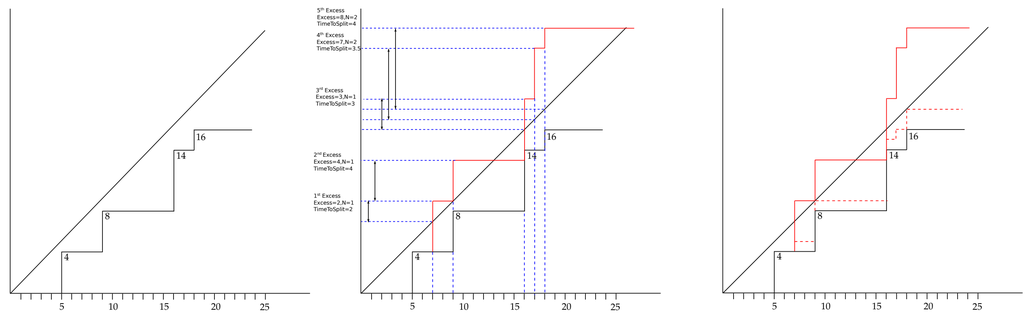

Figure 2: task decomposition

We iterate these operations till the task queue is empty. If all processors are investigated and the current task is not allocated, the task set is, then, not schedulable.

3 Resuts & discussion

The figures (3,4) represent the utilization and energy of the same task set for different partitioning heurstics on a uniform archiecture of 8 cores. We notice that FTC dominates all the other heursitics and can reach utilization rates very close to 1, which means that this heuristic is very close to the optimal algorithm. Thus, it consumes less energy (figure **??**) because it loads the core of lowest frequencies first (in the assumption that cores with lower frequencies consume less power). We notice, also, that our heursitic uses a less number of cores than all the other heuristics.

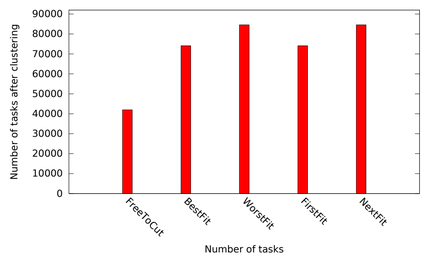

Figure 3: task decomposition

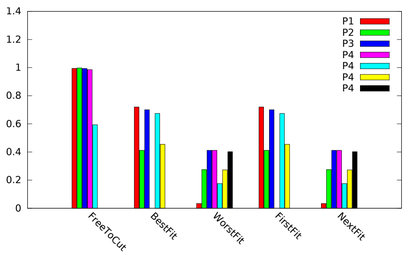

Figure 4: task decomposition

References

[1] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, 1990.

[2] Robert I Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.