On the Problem of Allocating Multicore Virtual Resources to Real-Time Task Pipelines

Giuseppe Lipari, Enrico Bini Scuola Superiore Sant'Anna, Pisa, Italy
Email: {g.lipari,e.bini}@sssup.it

***Abstract*—Real-time applications that process streams of data can be modelled by a pipeline of tasks, to be executed on a multi-processor system. The pipeline is periodically activated, and each instance must be completed before an end-to-end deadline. Three important problems must be solved by real-time designers: how to allocate tasks to processors, how to assign scheduling parameters to tasks, and how to guarantee than every pipeline will always complete within its deadline. In the literature, often these three interrelated problems have been tackled separately.**

In this paper, we use a component-based approach to the overall problem, and propose to analyse each pipeline in isolation. Starting from an algorithm for assigning intermediate deadlines to tasks called ORDER, we derive a simple expression that bounds the total amount of bandwidth to be allocated to the pipeline, and use such expression as a basis allocating the pipeline's tasks onto the available processors.

I. Introduction

In this paper we consider real-time systems consisting of a set of task pipelines to be executed on a multi-core system. To avoid interferences between pipelines and to isolate potential misbehaviour, each pipeline is assigned an amount of dedicated resources on each core. This approach enables the composition of pipelines which are then analysed in isolation. Isolating the pipelines has also the advantage of allowing the selection of possibly different scheduling policies for the tasks of the same pipeline.

A pipeline is a chain of tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with precedence constraints. A pipeline is associated with a period T and an end-to-end deadline D. The first task τ_1 is activated every period T, whereas any other task τ_j is activated by the completion of the preceding one τ_{j-1}. The end-to-end deadline represents the interval of time, starting from the activation of τ_1, within which all task instances must complete their execution. The system consists of m processors, and each task is bound to execute on a given processor. An example of such a system is depicted in Figure 1. In the figure, we graphically represent the four cores by a dashed rectangle, while the amount of resource dedicated to the pipeline is denoted by a grey share of the rectangle.

In this paper, we consider the problem of scheduling a pipeline such that the end-to-end deadline is met and the amount of required resource is minimal.

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7 under grant agreement n.248465 "S(o)OS Service-oriented Operating Systems."

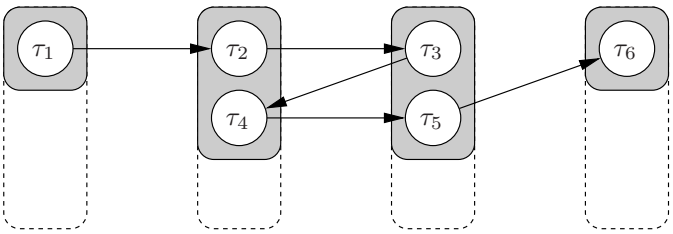

Fig. 1: Resource reservation on multicore.

A. Related work

The *holistic analysis* is a method for determining the feasibility of the problem when *the tasks of all pipelines* in the system are scheduled by FP [1]–[4] or EDF [5]–[8]. The holistic approach consists in reducing the overall distributed schedulability problem into m single-node problems that can be solved using classical schedulability analysis. Task parameters like offsets, jitters, response times are calculated so that the precedence constraints are automatically guaranteed. Since the schedulability of all pipelines depend on one another (i.e. the activation of an intermediate task, and hence its jitter, depends on the response time of the preceding task, which depends on the experienced interference), the analysis is iterated until either a fixed-point solution is found or the set is deemed not schedulable. The literature on holistic analysis for distributed systems is very rich. In classic holistic analysis, the best-case response time is needed to estimate either the task jitter [1], [2] or the task offset [7]. Several techniques have been proposed for the best-case response time by Redell and Sanfridson [9] and Bril et al. [10]. Pellizzoni and Lipari [7] proposed to transform the distributed pipeline problem into m single-node problems, after the intermediate deadlines have been assigned.

While the holistic analysis was also extended to account for the execution over virtual resources [11], the problem of determining the parameters of the virtual resources was not addressed.

An alternative method was proposed by Rahni et al. [8]. The method consists in splitting the overall analysis in two steps: in the first step the aggregate demand bound function of a pipeline is computed; in the second step, all the aggregate demand bound functions are added to test the overall schedulability.

If the tasks of the pipeline are scheduled by EDF, as we assume in this paper, it is necessary to *assign intermediate deadlines* to the tasks. In [12], [13], the authors propose to use

separate windows of execution for the tasks in a pipeline. Each task is assigned a release time coincident with the deadline of the previous task in the pipeline, and all tasks are required to complete before their absolute deadline. By using this technique (also called *slicing*) the problem is over-constrained with respect to the original model used in holistic analysis, since the task activation could be set at the finishing time of the preceding one (and not at its deadline). However, as reported in [13], the slicing technique eliminates release jitter and allows independent analysis of different pipelines. Therefore, we will follow this approach in our paper.

The assignment of intermediate deadlines does clearly influence the schedulability of the entire system: if a task is assigned a large intermediate deadline, it may finish too late leaving too little time for the subsequent tasks to complete before the end-to-end deadline. On the other hand, a too short deadline may be restrictive for the task itself. Two papers [12], [14] independently proposed similar assignment strategies. The first natural idea is divide the end-to-end deadline proportionally to the computation time of all tasks. A slightly different method is based on the even distribution of the laxity among all tasks. Recently it was proposed a deadline assignment method (ORDER) which is better capable to minimise the bandwidth allocated on each node [15]. All these three methods will be recalled in Section III-A.

B. Contributions of this paper

In this paper we review the ORDER algorithm, and propose a simple upper bound on the required amount of resources. Our upper bound simply depends on the number of tasks in the pipeline n, on the number of processor m and on the rate between end-to-end deadline D and period T. The simplicity of our bound can be used to build a simple heuristic for task allocation.

II. System model and notation

The pipeline $\mathcal{T}$ is composed by a set of n tasks $\{\tau_1, \ldots, \tau_n\}$. Task τ_i has a computation time C_i, while $C = \sum_{i=1}^{n} C_i$ is the overall computation time of the pipeline. The first task of the pipeline is activated periodically every T, while any other task τ_i is activated upon the completion of the preceding one τ_{i-1}. The *utilisation* of the pipeline is defined as $U = \frac{C}{T}$. The pipeline $\mathcal{T}$ has an *end-to-end deadline* D that is the maximum tolerable time from the activation of the first task τ_1 to the completion of the last task τ_n.

We assume that the tasks of the pipeline τ_i are scheduled by EDF over m resource reservations, each one allocated on a core. Task τ_i is statically bound to the core $x_i \in \{1, \ldots, m\}$. We observe that, if two consecutive tasks τ_i and τ_{i+1} are assigned the same core, then they can be glued together into a new task τ_i' with computation time $C_i' = C_i + C_{i+1}$. Hence we assume that $x_i \neq x_{i+1}$. We also define $\mathcal{T}_k = \{\tau_i \in \mathcal{T} : x_i = k\}$ as the subset of tasks in $\mathcal{T}$ mapped onto node k and we label by $U_k = \sum_{\tau_i \in \mathcal{T}_k} \frac{C_i}{T_i}$ the fraction of utilisation on processor k. For simplicity, each resource reservation is modelled by the bandwidth α_k it provides. The adoption of more complete reservation models [16] is possible and will be investigate in the future.

Each task is assigned an *intermediate deadline* $\overline{D}_i$, that is the interval of time between the activation of the pipeline and the absolute deadline of the task. Following the slicing technique [12], the precedence relationship between tasks is enforced by setting the *activation offset* ϕ_i of each task equal to the intermediate deadline of the preceding one, as follows:

$$\phi_1 = 0, \quad \phi_i = \overline{D}_{i-1} \quad i = 2, \ldots, n \tag{1}$$

Moreover, we define the task *relative deadline* D_i as

$$D_i \stackrel{\text{def}}{=} \overline{D}_i - \phi_i.$$

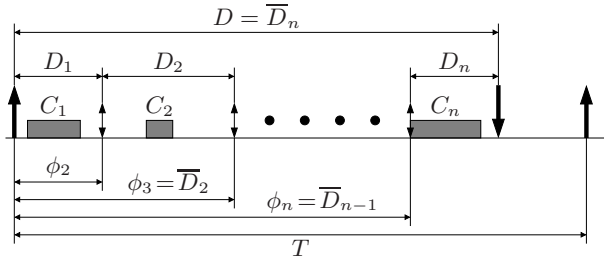

Fig. 2: Notation for tasks.

The relationship between activation offsets and relative deadlines is depicted in Figure 2. The assignment of D_i is being investigated in this paper. Clearly,

$$\sum_{i=1}^{n} D_i = D \tag{2}$$

The slicing assumption (Equation (1)) allows us to decouple the design and analysis of each different pipeline, since the activations depend only on parameters of the same pipeline (deadlines), and not on the completion times, which in turn depend on the interference from other pipelines.

Finally, we use the notation $(\cdot)_0 \stackrel{\text{def}}{=} \max\{0, \cdot\}$.

A. The demand bound function

The computational requirement of the tasks in $\mathcal{T}_k$ is modelled by its *demand bound function*. We denote by $\mathsf{df}_k(t_0, t_1)$ the total computation time of all the instances of the tasks in $\mathcal{T}_k$, having activation time and deadline within $[t_0, t_1]$. It is defined as follows [17]:

$$\mathsf{df}_k(t_0, t_1) \stackrel{\text{def}}{=} \sum_{\tau_i \in \mathcal{T}_k} \left(\left\lfloor \frac{t_1 - \overline{D}_i}{T} \right\rfloor - \left\lceil \frac{t_0 - \phi_i}{T} \right\rceil + 1 \right)_0 C_i \tag{3}$$

The overall demand bound function can be defined as:

$$\mathsf{dbf}_k(t) \stackrel{\text{def}}{=} \max_{t_0} \mathsf{df}_k(t_0, t_0 + t) \tag{4}$$

A necessary and sufficient schedulability test for EDF over a virtual resource consists in checking whether the demand exceeds the amount of available computational resource:

$$\forall k = 1, \ldots, m \;\; \forall t > 0 \;\; \mathsf{dbf}_k(t) \leq \alpha_k t. \tag{5}$$

Since the demand bound function $\mathsf{dbf}_k(t)$ is right-continuous, piecewise constant, and increasing, it can be

represented by the values at each step. For this purpose we introduce the set of *scheduling points* as follows:

$$\mathcal{P}_k \stackrel{\text{def}}{=} \{(t;w) \,:\, \mathsf{dbf}_k(t) = w,\ \lim_{x\to t^-} \mathsf{dbf}_k(x) < w\} \quad (6)$$

Notice that the set $\mathcal{P}_k$ contains infinite scheduling points. However, it has been proved [17] that after some initial transient points, the function repeats every period T. Hence the set $\mathcal{P}_k$ can be finitely represented by its finite transient part $\mathcal{P}_k^t$ and its finite periodic part $\mathcal{P}_k^p$ as follows:

$$\mathcal{P}_k = \mathcal{P}_k^t \cup \{(t;w) : t = t' + qT, w = w' + qC, \\ (t';w') \in \mathcal{P}_k^p, q \in \mathbb{N}\} \quad (7)$$

The set of dbf_k on each core represents the amount of computation required by the pipeline. However the precedence constraint prevents a simple understanding of how the deadline assignment influences the dbf.

III. The minimal bandwidth target

Ideally, the deadlines should be assigned in a way that the dbf_k over any core (the LHS of Eq. (5)) are *as low as possible*. The minimal bandwidth α_k that guarantees the tasks over the core k is [18]:

$$\alpha_k = \sup_{(t;w)\in\mathcal{P}_k} \frac{w}{t} \quad (8)$$

Hence it is quite straightforward to set our primary target at finding the deadline assignment that minimises α_k. However there is a (trivial) lower bound to this minimisation, since it must be $\alpha_k \geq U_k$.

The sum of all tasks deadlines must be equal to the pipeline end-to-end deadline. As a consequence, if we minimise the bandwidth on node k, the required bandwidth on another core may increase. In other words, it is not possible to minimise every dbf_k independently of the others. Therefore the m bandwidths over the cores need to be properly weighted. We choose then to formulate the problem as

$$\begin{aligned} &\text{minimise } \max_k \frac{\alpha_k}{U_k} \\ &\text{subject to } \sum_{j=1}^{n} D_j = D \end{aligned} \quad (9)$$

with the interpretation that the target represents the fraction of extra bandwidth required to schedule the pipelines. The value of the optimisation function is called *energy* of the solution.

This target function is relevant for our goals of component-based analysis: in fact, we would like to set the intermediate deadlines so that the assigned bandwidth α_k to each pipeline on processor k be as close as possible to the minimum utilisation U_k. The optimal solution tells the designer how much utilisation must be allocated on each processor; thus the designer can perform an early allocation of the bandwidth of the processors to each pipeline in the system.

A. Deadline assignment rules

The assignment of intermediate deadlines clearly influences the schedulability of the entire system: if a task is assigned a large intermediate deadline, it may finish too late leaving too little time for the subsequent tasks to complete before the end-to-end deadline. On the other hand, a too short deadline may be restrictive for the task itself. Two papers [12], [14] independently proposed similar assignment strategies. The first natural idea is divide the end-to-end deadline proportionally to the computation time of all tasks, as follows

$$D_j = D\frac{C_j}{C}. \quad (10)$$

This method is called NORM in [12] and it is the most widely used in the literature.

A different method is based on the distribution of the laxity equally among all tasks:

$$D_j = C_j + \frac{D - C}{n} \quad (11)$$

This method is called PURE in [12].

Recently [15], a new deadline assignment method has been proposed, called ORDER, specifically targeted at the minimisation of the bandwidth required by the transactions. Since we are going to exploit this method in depth, we recall it in the next section.

IV. The ORDER algorithm

The ORDER deadline assignment rule requires first to order all tasks mapped on the same core, by non-decreasing computation time. This operation is enabled by defining the mapping $s_k : \{1,\dots,n_k\} \to \{1,\dots,n\}$ such that

$$\forall \ell, j = 1,\dots,n_k \quad \begin{array}{l} \tau_{s_k(\ell)}, \tau_{s_k(j)} \in \mathcal{T}_k \\ \ell \leq j \Leftrightarrow C_{s_k(\ell)} \leq C_{s_k(j)} \end{array}. \quad (12)$$

We also define $z : \{1,\dots,n\} \to \{1,\dots,n_k\}$ as the inverse mapping of the different s_k:

$$\forall k, j = 1,\dots,n_k, \quad s_k(j) = i \quad \Rightarrow \quad z(i) = j,$$

which enables the definition of

$$\forall i = 1,\dots,n, \quad \delta_i = \sum_{j=1}^{z(i)} C_{s_{x_i}(j)} \quad (13)$$

Basically, $z(i) - 1$ is the number of tasks that have shorter computation time than τ_i on the same core x_i. Therefore, δ_i is the sum of all computation times of the tasks in $\mathcal{T}_k$ that precede (and include) τ_i in the mapping s_{x_i}, with x_i being, we remind, the index of the core where τ_i is mapped.

To clarify this notation, we propose a simple example of pipeline with $n = 6$ tasks, running over $m = 3$ cores. The data is reported in Table I.

i	1	2	3	4	5	6
x_i	1	2	1	3	1	2
C_i	3	8	1	3	4	5

TABLE I: Data of the example.

j	1	2	3
$s_1(j)$	3	1	5

j	1	2
$s_2(j)$	6	2

j	1
$s_3(j)$	4

i	1	2	3	4	5	6
$z(i)$	2	2	1	1	3	1

TABLE II: The permutation functions.

For this example, the mappings s_1, s_2, and s_3 and the inverse mapping z are described in Table II.

Thanks to this notation we can then define the ORDER deadline assignment rule.

Definition 1: We define the ORDER deadline assignment as:

$$\forall i = 1, \ldots, n \quad D_i = \delta_i. \tag{14}$$

The following theorem [15] relates the ORDER deadline assignment with the amount of consumed bandwidth. (The interested reader is invited to read [15] for the proof)

Theorem 1 (Theorem 2 in [15]): If the intermediate deadlines are assigned according to the ORDER rule and $\forall k\ U_k \leq 1$, we have

$$\forall k, \forall t \geq 0 \quad \mathsf{dbf}_k(t) \leq t \tag{15}$$

Basically the theorem asserts that the ORDER deadline assignment guarantees the feasibility on all core of the pipeline.

From Theorem 1, it is possible to compute the minimal end-to-end deadline that guarantee at least feasibility according to the ORDER assignment, which is:

$$D^{\mathsf{min}} = \sum_{k=1}^{m} \sum_{j=1}^{n_k} D_{s_k(j)} = \sum_{k=1}^{m} \sum_{j=1}^{n_k} \sum_{\ell=1}^{j} C_{s_k(\ell)} = \sum_{i=1}^{n} \delta_i \tag{16}$$

A. Bandwidth requirements of the deadline assignments

The proposed deadline assignment (Eq. (14)) is built such that the bandwidths α_k on each node are all equal to 1. If the actual deadline is $D < D^{\mathsf{min}}$, then the ORDER deadline assignment cannot guarantee feasibility. However, if $D \geq D^{\mathsf{min}}$ we can take advantage of the slack between D^{min} and D to reduce the required bandwidths.

For this purpose we recall the following Corollary of Theorem 1.

Corollary 1 (Corollary 2 in [15]): If the intermediate deadlines are assigned as follows:

$$\forall k, \forall j = 1, \ldots, n_k, \quad D_{s_k(j)} = \frac{1}{\alpha_k} \delta_{s_k(j)} \tag{17}$$

with $U_k \leq \alpha_k \leq 1$, then:

$$\forall k, \forall t, \quad \mathsf{dbf}_k(t) \leq \alpha_k t$$

If we set all bandwidths α_k equal to the minimal value U_k, then the deadlines becomes

$$\forall j = 1, \ldots, n_k, \quad D_{s_k(j)} = \frac{1}{U_k} \delta_{s_k(j)} \tag{18}$$

Notice that, after the multiplication by $\frac{1}{U_k}$, the largest deadline on any processor is always equal to the period:

$$D_{s_k(n_k)} = \frac{1}{U_k} \sum_{j=1}^{n_k} C_{s_k(j)} = \frac{1}{U_k} U_k T = T.$$

Therefore, by assigning the deadlines as indicated in Eq. (18), the end-to-end deadline D must be at least m times the pipeline period T. Moreover if we define

$$D^{\mathsf{max}} = \sum_{k=1}^{m} \frac{1}{U_k} \sum_{j=1}^{n_k} \delta_{s_k(j)} \tag{19}$$

we can say that if $D \geq D^{\mathsf{max}}$ the assignment of deadline according to (18) is capable of guaranteeing a bandwidth $\alpha_k = U_k$ on all processors.

In the general case (of $D \in (D^{\mathsf{min}}, D^{\mathsf{max}})$) the best bandwidth assignment is subject to a deeper investigation. By changing variable $\xi_k = \frac{\alpha_k}{U_k}$, the problem of (9) can be rewritten as:

$$\begin{aligned} &\text{minimise} \max_k \xi_k \\ &\text{subject to} \begin{cases} \sum_{i=1}^{n} \frac{\delta_i}{U_{x_i} \xi_{x_i}} \leq D \\ 1 \leq \xi_k \leq \frac{1}{U_k} \end{cases} \end{aligned}$$

where x_i is the index of the processor on which task τ_i is allocated.

The first constraint represents the fact that the sum of all intermediate deadlines cannot exceed the end-to-end deadline, while the second constraint tells us that we cannot overload processor k, neither we can allocate less bandwidth than the minimum U_k.

If we ignore for the moment the last constraint, this problem has solution when all ξ_k are equal to the same constant ξ. By substituting in the first constraint, we obtain:

$$\xi = \frac{1}{D} \sum_{i=1}^{n} \frac{\delta_i}{U_{x_i}} \tag{20}$$

and

$$\forall k, \alpha_k = \xi U_k \tag{21}$$

If for some $\overline{k}$ $\alpha_{\overline{k}} > 1$, then we can set $\alpha_{\overline{k}} = 1$, and iterate the previous solution to find the other α_k. The number of iterations, in the worst case, is m; to compute δ_k we need to sum n_k variables; thus the complexity is $O(m^2 \max n_k)$.

B. Examples

We first consider the simple situation in which each task of a given pipeline is allocated on a different node. Consider a pipeline consisting of 2 tasks with computation time $C_1 = 1$ and $C_2 = 2$, allocated on 2 processors. The pipeline period and end-to-end deadline are $T = 20$ and $D = 20$, respectively.

We first compute the energy ξ

$$\xi = \frac{1}{D} \left(\frac{\delta_1}{U_1} + \frac{\delta_2}{U_2} \right) = \frac{1}{D} \left(\frac{C_1}{U_1} + \frac{C_2}{U_2} \right) = \frac{2T}{D} = 2.$$

Then, $\alpha_1 = \xi U_1 = 0.1$ and $\alpha_2 = \xi U_2 = 0.2$. Finally, the deadlines are:

$$D_1 = \frac{C_1}{\alpha_1} = \frac{T}{\xi} = \frac{D}{2} = 10 \quad D_2 = \frac{C_2}{\alpha_2} = 10.$$

Notice that, in this case, the value of the energy ξ does not depend on the values of the execution times. In fact, if we multiply all computation times by a constant, ξ and D_i do not change.

In the general case of n tasks on n processors, it is easy to see that the energy can be computed as $\xi = \max\{1, \frac{nT}{D}\}$, and the relative deadlines are all equal to $D_i = \frac{D}{n}$.

In Table III, we compare the intermediate deadlines, the assigned bandwidth and the energy ξ of the three methods ORDER, NORM, and PURE. The latter two algorithms are described by Equations (10) and (11), respectively. Notice that,

	D_1	D_2	α_1	α_2	ξ
NORM	6.66	12.33	0.15	0.15	3
PURE	9.5	10.5	0.105	0.19	2.1
ORDER	10	10	0.1	0.2	2

TABLE III: Behaviour of ORDER, NORM and PURE on the first example.

for both NORM and PURE, the actual values of the energy and of the deadlines depend on the computation times, and the energy is higher than the energy obtained with ORDER. This means that, with the same pipeline, the distribution of the utilisation among the two processors is more unbalanced.

Now, let us consider a slightly more complex example of 3 tasks on two processors. The three tasks have computation times $C_1 = 1$, $C_2 = 2$, $C_3 = 3$, and are allocated on the first, second and first processor, respectively. The period is $T = 20$ and the end-to-end deadline is $D = 30$. Hence $U_1 = \frac{C_1+C_3}{T} = 0.2$ and $U_2 = \frac{C_2}{T} = 0.1$.

We start by computing

$$\xi = \frac{1}{D}\left(\frac{C_1 + C_1 + C_3}{U_1} + \frac{C_2}{U_2}\right) = \frac{25+20}{30} = 1.5$$

Then, $\alpha_1 = \xi U_1 = 0.3$ and $\alpha_2 = \xi U_2 = 0.15$. Finally, the deadlines are:

$$D_1 = \frac{C_1}{\alpha_1} = 3.33 \quad D_2 = \frac{C_2}{\alpha_2} = 13.33$$
$$D_3 = \frac{C_1 + C_3}{\alpha_1} = 13.33$$

Once again, note that the energy does not depend on the values of the computation times: if we multiply all computations times by a constant K, also the utilisation is multiplied by the same constant. Rather, ξ depends on how the computation times are distributed among the two processors. The comparison with the other algorithms is shown in Table IV.

	D_1	D_2	D_3	α_1	α_2	ξ
NORM	5	10	15	0.27	0.2	2
PURE	9	10	11	0.36	0.2	2
ORDER	3.33	13.33	13.33	0.3	0.15	1.5

TABLE IV: Behaviour of ORDER, NORM and PURE on the second example.

Notice that once again ORDER achieves a lower energy. This means that, if we use NORM or PURE to assign deadlines, we have to allocate two times the minimum possible utilisation on the second processor (0.2 instead of 0.1). If we use ORDER, instead, we only have to assign 50% more utilisation on both processors.

Below, we provide some experiments with randomly generated pipelines.

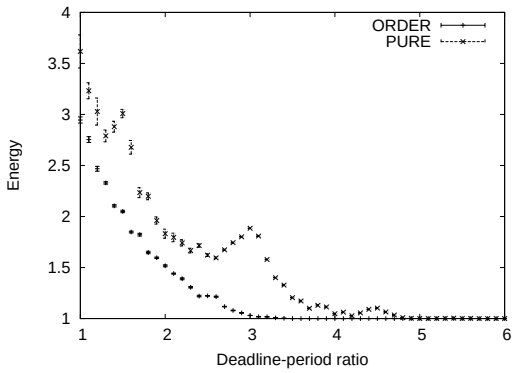

Fig. 3: ORDER vs PURE on 2 processors with 6 tasks

C. Simulations

In [15], Algorithm ORDER has been compared against Algorithm PURE [12]. We report here only one graph to highlight the main differences between the two algorithms.

In Figure 3, we show the comparison between ORDER and PURE for the case of 2 processors and 6 tasks. Since we imposed that two consecutive tasks cannot be allocated on the same processor, in this case the task mapping is unique: $x_1 = x_3 = x_5 = 1$ and $x_2 = x_4 = x_6 = 2$. In this case ORDER always perform better than PURE. Also, notice that PURE has a strange behaviour around values 1.5, 3 and 4.5 of the deadline-period ratio. This is due to the fact that algorithm PURE does not take into account the number of tasks allocated on the same processor. If the three tasks on one processor do not interfere much (for example, when the deadline-period ratio is around 2 their "slices" are not overlapping), then the deadline assignment made by PURE is a "good" assignment. Instead, when the slices overlap (as in the case of $\frac{D}{T} = 3$), then the demand increases. Algorithm ORDER has not such a problem as it directly accounts for tasks on the same processor.

The results are very similar for different number of processors and different number of tasks in the pipeline.

One important consideration can be made. The results of the experiments show that the ratios between α_k and U_k that we obtain with algorithms ORDER and PURE do not depend on the actual values of the computation times, and very little on their distribution.

Also all the experiments shows that the energy is inversely proportional to the ratio D/T. This property will actually be formally proved in the next section.

V. Upper bound on the energy

The energy represents the amount of extra bandwidth that is necessary on each core to guarantee the feasibility with end-to-end deadline D. It is natural to quest for an upper bound of it. This is provided by the following elegant theorem.

Theorem 2: By assigning deadlines according to (17), the minimal energy ξ found as solution of the problem (9) is upper bounded by

$$\xi \leq \frac{m+n}{2d} \tag{22}$$

with $d = \frac{D}{T}$.

Proof: Let us have a closer look at the expression for the energy:

$$\xi = \frac{\sum_{i=1}^{n} \frac{\delta_i}{U_{x_i}}}{D}$$

We start by expanding the numerator.

$$\sum_{i=1}^{n} \frac{\delta_i}{U_{x_i}} = T \sum_{i=1}^{n} \frac{\sum_{j=1}^{z(i)} C_{s_k(j)}}{\sum_{j=1}^{n_{x_i}} C_{s_k(j)}}$$

Notice that for $z(i) = n_{x_i}$ (i.e. τ_i is the task with the largest execution time on processor x_i), the fraction simplifies to 1. There are m tasks like this in the system (one for each processor), so we can rewrite the expression for ξ as:

$$\xi = \frac{mT}{D} + \frac{1}{D} \sum_{k=1}^{m} \frac{\sum_{j=i}^{n_k-1} \delta_{s_k(j)}}{U_k}$$

Now we need a way to bound the second term. Observe that the maximum value for this term is when all computation times on each processor are the same. In this case, let's call $C(k)$ the computation time of all tasks on processor k. Then:

$$\begin{aligned} \xi &\leq \frac{mT}{D} + \frac{1}{D} \sum_{k=1}^{m} \frac{\sum_{j=i}^{n_k-1} \sum_{h=1}^{j} C(k)}{\frac{n_k C(k)}{T}} \\ &= \frac{mT}{D} + \frac{T}{D} \sum_{k=1}^{m} \frac{\sum_{j=i}^{n_k-1} jC(k)}{n_k C(k)} \\ &= \frac{mT}{D} + \frac{T}{D} \sum_{k=1}^{m} \frac{n_k - 1}{2} \\ &= \frac{T}{D} \left(m + \frac{1}{2}(n-m) \right) \end{aligned}$$

which proves (22) ∎

A. Analysis of ξ

Let us analyse this simple bound in more details. The bound depends on m, which is the number of processors on which the pipeline has been allocated; on n which is the number of tasks in the pipeline; and on the ratio D/T. It does not depend on the tasks' computation times, and on their allocation.

Notice that we require that the two consecutive tasks allocated on the same processor be treated as one single task whose computation time is the sum of the computation times of the original tasks. For example, consider a pipeline consisting of 4 tasks $\mathcal{T} = \{\tau_1, \ldots, \tau_4\}$, with computations times $C_1 = 1, C_2 = 2, C_3 = 3, C_4 = 2$, period $T = 12$ and end-to-end deadline $D = 18$; suppose now that the tasks are allocated, respectively, on processor 1, 2, 2, and 3; then, we consider an equivalent pipeline of 3 tasks $\mathcal{T}' = \{\tau_1, \tau_2', \tau_4\}$, with $C_2' = C_2 + C_3 = 5$; therefore, in this case $m = 3$ and $n = 3$.

The energy ξ represent the ratio between the total bandwidth needed and the actual total utilisation of the pipeline. In the previous example, we have $U = \frac{\sum_i C_i}{T} = 8/12 = 0.66$, and $\xi = \frac{m+n}{2d} = 6 \cdot 6/18 = 2$, therefore we have to allocate a total bandwidth of 1.33 on 3 processors. Specifically, $\alpha_1 = 1/6$, $\alpha_2 = 5/6$ and $\alpha_3 = 1/3$. Finally, the intermediate deadlines of the 3 tasks are: $D_1 = D_2' = D_4 = 6$. In this example, the original pipeline has been poorly allocated. Task τ_2 and τ_3 have the largest computation times, and putting them together on the same processor is probably the wrong choice.

Now, let us analyse what happens in general.

We start by considering the limit case of $m = n = 1$. This corresponds to a single task running on one processor. If $D \leq T$, then $\xi = \frac{T}{D}$ and $\alpha = \frac{TC}{DT} = \frac{C}{D}$ which is exact. If instead $D > T$, the second constraint tells us that $\xi \geq 1$, therefore, $\xi = 1$ and $\alpha = U$, and again this is what we expect.

Now consider the case in which $n = m > 1$. This case corresponds to a pipeline of n tasks allocated on n processors. In this case, the bound is exact, because there is only one task on each processor, and $\forall i, \frac{\delta_i}{U_i} = T$. Therefore, $\xi = \frac{nT}{D}$. The intermediate deadlines are all equal and can be expressed by $D_i = \frac{max(T,D)}{n}$ (see the example in the previous section).

Now, let us consider the more general case in which $n > m$ (it is not possible to have $n < m$, since m is the number of processors on which the pipeline tasks have been allocated). Therefore, there must be at least one processor with two or more non-consecutive pipeline tasks allocated to it. In this case, the bound on the energy ξ may be an overestimation of the real ξ, depending on the relative values of the computation times of the tasks that have been allocated on the same processor. If they are identical, then the bound is exact; if they are different, the bound is pessimistic.

These considerations may guide us in deriving an allocation strategy for real-time pipelines of tasks on multi-core processors.

VI. Allocation

Let us consider the following allocation problem. We have a pipeline $\mathcal{T}$ of N *stages*. A stage, denoted by σ_i represents the functionality to be executed by a task. A stage is characterised by its worst-case computation time c_i. A task τ_j is a sequence of one or more consecutive stages of a pipeline, and its WCET is the sum of the WCETs of its stages. For simplicity we assume that two consecutive tasks of the pipeline are always allocated on different processors. We also know the of each stage, the pipeline period T, and the pipeline must be allocated so that it will always complete before its end-to-end deadline D. The tasks must be allocated on a system of M identical processors which are already partially utilised: for each processor, we know its current utilisation factor μ_k.

For example, consider a pipeline with $N = 4$ stages, $\mathcal{T} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$. If we put stage σ_1 into τ_1, stages σ_2 and σ_3 into one single task τ_2, and σ_4 into τ_3, the pipeline becomes $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ and a possible allocation is $x_1 = 1$, $x_2 = 2$, $x_3 = 1$.

Given these definition, we can now mathematically state our allocation problem:

Definition 2 (Allocation Problem): Given a pipeline as a sequence of stages, a period and an end-to-end deadline; group the N stages into $n \leq N$ tasks, and allocate the tasks on a subset of $m \leq M$ processors such that no two consecutive tasks lie on the same processor, and the utilisation μ_k of each processor k, plus the bandwidth requirement α_k of the pipeline, does not exceed 1.

To solve this problem we need an algorithm that explores the different combinations of tasks and allocations. We are currently working at defining such algorithm, and it will be the topic of a future paper. Nevertheless, to give a better idea of the allocation problem, we now present a complete example of a possible strategy that uses the upper bound on ξ as its main driver to solve the problem.

A. An example of allocation

Suppose we have a pipeline of 4 stages, $\mathcal{T} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ with $T = 12$ and $D = 18$, and computation times $c_1 = 1, c_2 = 2, c_3 = 3, c_4 = 3$. We want to allocate such pipeline on a multicore system with 4 processors, already partially utilised, with $\mu_1 = 0.4$, $\mu_2 = 0.4$, $\mu_3 = 0.4$, $\mu_4 = 0.7$.

The pipeline utilisation is $U = 9/12 = 0.75$, therefore it is not possible to collapse the pipeline into a single task to be allocated on any of the processors. Observe that the total amount of bandwidth to be reserved is $\xi \cdot U = \frac{n+m}{4}$, which is directly proportional to $n + m$, therefore we must try to keep both m and n as small as it is possible.

We start by selecting $m = n = 2$. In this case $\xi = 4/3$, and the total required bandwidth is 1 on two processors. The pipeline can be split in two tasks in different ways.

- Consider the following two tasks: $\tau_1 = \{\sigma_1, \sigma_2\}$ and $\tau_2 = \{\sigma_3, \sigma_4\}$. Hence, $C_1 = 3$ and $C_2 = 6$. The first task has utilization $U_1 = 1/4$, and requires a bandwidth of $\alpha_1 = \xi U_1 = 1/3$: hence it can be allocated on any of the first three processors. The second task has utilization $U_2 = 1/2$, and requires a bandwidth of $\alpha_2 = \xi U_2 = 2/3$, which instead does not fit on any processor. Therefore, this combination is not feasible.
- Consider a second combination $\tau_1 = \{\sigma_1, \sigma_2, \sigma_3\}$ and $\tau_2 = \{\sigma_4\}$. Hence, $C_1 = 6$ and $C_2 = 3$. This case is symmetric to the previous one, so it is not schedulable either.
- It is easy to see that any other combination with $n = 2$ is unfeasible.

The problem is that if we split the pipeline in two tasks, one of them has a large computation time and does not fit in any of the processors, while the other one has a low computation time. So, maybe splitting the pipeline in three tasks we can make the pipeline feasible.

We now consider the case of $m = 2$ and $n = 3$. In this case $\xi \leq 5/3$. Again, there are many ways to split the pipeline in three tasks:

- Consider the following three tasks: $\tau_1 = \{\sigma_1\}$, $\tau_2 = \{\sigma_2, \sigma_3\}$, $\tau_3 = \{\sigma_4\}$. Hence, $C_1 = 1$, $C_2 = 5$, $C_3 = 3$. The only possibility is to allocate both τ_1 and τ_3 on the same reservation, and τ_2 on a second reservation. Notice that in this case, the bound of $\xi \leq 5/3$ is pessimistic: in fact, the two tasks τ_1 and τ_3 have very different computation times. The exact value of the energy is instead $\xi = 3/2$ (computed by using Equation (20)). Therefore, $\alpha_1 = \xi U_1 = \xi \frac{C_1 + C_3}{T} = 0.5$, and τ_1 and τ_3 can be allocated on any of the first three processors. Unfortunately, $\alpha_2 = \xi \frac{C_2}{T} = \frac{5}{8} = 0.625$, and it cannot be allocated on any processor.
- Again, it is easy to see that any other combination of the stages into $n = 3$ tasks and $m = 2$ processors is unfeasible.

Thus, we can only increase the number of processors m, hoping that using more processors we can better distribute the load. However, we pay a price because ξ also increases: when $m = 3$ and $n = 3$, $\xi = 2$.

- Consider the following three tasks: $\tau_1 = \{\sigma_1, \sigma_2\}$, $\tau_2 = \{\sigma_3\}$, $\tau_3 = \{\sigma_4\}$. Hence, $C_1 = C_2 = C_3 = 3$. Then
$$\forall k = 1, 2, 3, U_k = \frac{1}{4} \quad \alpha_k = \frac{1}{2}$$
Now it is possible to allocate the three tasks on the first three processors.

It is worth to highlight one important property. Consider the two cases in which $m = n = 3$, and $m' = 2$, $n' = 4$. The upper bound for the energy in these two cases is the same and equal to $\frac{3}{d}$. However, in the first case the bound is exact, while in the second case the bound may be pessimistic, depending on the relative values of the computation times of the tasks allocated on the same reservations. In other words:

$$\xi' \leq \xi = \frac{3}{d}$$

This means that in certain cases it may be convenient to allocate more than one non-consecutive tasks on the same reservation, in order to reduce the overall amount of required bandwidth. Also, this result confirms the fact that using a higher number of parallel processors often requires a larger amount of reserved bandwidth.

VII. Conclusions

In this paper we analysed the performance of the ORDER algorithm [15], which is used to set intermediate deadlines for pipelines of tasks. We derived a simple but tight bound on the total amount of resources that the designer must allocate in order to feasibly schedule the pipeline. We used this simple result to reason on the problem of allocating tasks to processors in a multi-core real-time systems.

Currently, we are working at deriving an allocation algorithm that uses this simple bound to drive the allocation strategy. Also, we plan to extend this work to task graphs.

References

[1] K. W. Tindell, A. Burns, and A. Wellings, "An extendible approach for analysing fixed priority hard real-time tasks," *Journal of Real Time Systems*, vol. 6, no. 2, pp. 133–152, Mar. 1994.

[2] J. C. Palencia and M. González Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings of the* 19*th IEEE Real-Time Systems Symposium*, Madrid, Spain, Dec. 1998, pp. 26–37.

[3] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," in *CODES '02: Proceedings of the* 10*th international symposium on Hardware/software codesign*. Estes Park, Colorado, USA: ACM, 2002, pp. 187–192.

[4] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets," *Real-Time Systems Journal*, vol. 40, no. 1, pp. 77–116, Feb. 2008.

[5] M. Spuri, "Holistic analysis for deadline scheduled real-time distributed systems," INRIA, France, Tech. Rep. RR-2873, Apr. 1996.

[6] J. Palencia and M. G. Harbour, "Offset-based response time analysis of distributed systems scheduled under EDF," in *15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003.

[7] R. Pellizzoni and G. Lipari, "Holistic analysis of asynchronous real-time transactions with earliest deadline scheduling," *Journal of Computer and System Sciences*, vol. 73, no. 2, pp. 186–206, Mar. 2007.

[8] A. Rahni, E. Grolleau, and M. Richard, "Feasibility analysis of non-concrete real-time transactions with EDF assignment priority," in *Proceedings of the* 16^{th} *conference on Real-Time and Network Systems*, Rennes, France, Oct. 2008, pp. 109–117.

[9] O. Redell and M. Sanfridson, "Exact best-case response time analysis of fixed priority scheduled tasks," in *Proceeding of* 14^{th} *Euromicro Conference on Real-Time Systems*, Wien, Austria, Jun. 2002, pp. 165–172.

[10] R. J. Bril, L. Cucu-Grosjean, and J. Goossens, "Exact best-case response time analysis of real-time tasks under fixed-priority pre-emptive scheduling for arbitrary deadlines," in 21^{st} *Euromicro Conference on Real-Time Systems (Work-in-Progress session)*, 2009.

[11] J. L. Lorente, G. Lipari, and E. Bini, "A hierarchical scheduling model for component-based real-time systems," in *Proceedings of the* 20^{th} *International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, Apr. 2006.

[12] M. Di Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proceedings of the* 15^{th} *IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, Dec. 1994, pp. 215–227.

[13] J. Jonsson and K. G. Shin, "Deadline assignment in distributed hard real-time systems with relaxed locality constraints," Chalmers University of Technology, Goteborg, Sweden, Tech. Rep., April 1996.

[14] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1268–1274, Dec. 1997.

[15] N. Serreli, G. Lipari, and E. Bini, "Deadline assignment for component-based analysis of real-time transactions," in *2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Washington, DC, U.S.A., Dec. 2009.

[16] E. Bini, G. C. Buttazzo, and M. Bertogna, "The multy supply function abstraction for multiprocessors," in 15^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Beijing, China, Aug. 2009.

[17] S. K. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[18] E. Bini, G. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *Proceedings of the* 17^{th} *Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, Jul. 2005, pp. 3–10.