CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Towards Holistic Analysis for Fork-Join Parallel/Distributed Real-Time Tasks

Ricardo Garibay-Martinez
Luis Lino Ferreira
Geoffrey Nelissen
Luis Miguel Pinho

CISTER-TR-140707
Version:
Date: 7/9/2014

Towards Holistic Analysis for Fork-Join Parallel/Distributed Real-Time Tasks

Ricardo Garibay-Martinez, Luis Lino Ferreira, Geoffrey Nelissen, Luis Miguel Pinho

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: rgmaz@isep.ipp.pt, llf@isep.ipp.pt, grrpn@isep.ipp.pt, lmp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Parallel/distributed processing is a solution for providing scaling computing power for computational-intensive applications. Parallel/distributed applications are commonly based on the fork-join model, where precedence constraints have to be considered on the development of an adequate timing analysis. Moreover, as the main difference with multicore architectures, distributed systems entail the transfer of messages upon a communication network that should be integrated in the timing analysis. In this context, this paper presents the current status of the work towards holistic analysis for fixed priority fork-join parallel/distributed tasks. This analysis takes into consideration the interactions between parallel threads and their respective messages. These considerations will be helpful for the improvement of the combination of existing results for computing the worst-case response time and the specific case of fork-join parallel/distributed real-time tasks.

Towards Holistic Analysis for Fork-Join Parallel/Distributed Real-Time Tasks

Ricardo Garibay-Martínez[1], Luis Lino Ferreira[1], Geoffrey Nelissen[1], Paulo Pedreiras[2], Luís Miguel Pinho[1]
[1]CISTER/INESC-TEC, ISEP, Porto, Portugal
[2]DETI/IT/University of Aveiro, Aveiro, Portugal
[1]{rgmaz, llf, grrpn, lmp}@isep.ipp.pt; [2]pbrp@ua.pt

***Abstract*—Parallel/distributed processing is a solution for providing scaling computing power for computational-intensive applications. Parallel/distributed applications are commonly based on the fork-join model, where precedence constraints have to be considered on the development of an adequate timing analysis. Moreover, as the main difference with multicore architectures, distributed systems entail the transfer of messages upon a communication network that should be integrated in the timing analysis. In this context, this paper presents the current status of the work towards holistic analysis for fixed priority fork-join parallel/distributed tasks. This analysis takes into consideration the interactions between parallel threads and their respective messages. These considerations will be helpful for the improvement of the combination of existing results for computing the worst-case response time and the specific case of fork-join parallel/distributed real-time tasks.**

Keywords—Real-time; parallel execution; distributed systems; holistic analysis.

I. INTRODUCTION

Modern real-time applications are increasingly complex requiring the use of more powerful computing resources. The current trend of using parallel processing in the embedded domain seems a promising solution to cope with the requirements of such demanding applications. Therefore, the real-time community has been making efforts to extend traditional real-time tools and scheduling algorithms to consider parallel task models. However, in some embedded applications, the use of powerful enough multi-core processors, is prohibited due to Size, Weight, and Power (SWaP) constraints. But it is also possible to comply with the requirements of computational-intensive applications by allowing single-core embedded devices connected through a local real-time network, to distribute its workload to remote neighbour nodes and execute the applications in parallel.

In this work we consider fork-join distributed real-time applications [1] which are composed of a set of fork-join Parallel/Distributed real-time tasks (P/D tasks), executing in a distributed system. When considering such tasks, the processing of tasks and messages must comply with their associated time constraints. A P/D task starts by a master thread executing sequentially; and then forks to be executed in parallel on *remote processors*. When the parallel execution has completed on each of the remote processors, the results are aggregated by performing a join operation and the execution of the sequential thread is resumed within the master thread. We call to those operations, the Distributed-Fork (D-Fork) and Distributed-Join (D-Join).

We also consider that P/D tasks are scheduled with the *Partitioned/Distributed - Deadline Monotonic Scheduling* (P/D-DMS) algorithm, proposed in [2]. The P/D-DMS algorithm can be used for partitioning a set of threads onto uniprocessor nodes connected through a real-time network. The algorithm makes use of the *Distributed Stretch Transformation* (DST) [2]. After applying the DST, a set of P/D threads have to be assigned onto the nodes of the distributed systems, which is done by using the Fisher Baruah Baker - First Fit Decreasing (FBB-FFD) algorithm [3].

Goal of this work. In a previous work, Axer *et al.* [4] presented a method for computing the response time of fixed-priority parallel tasks on multiprocessors, which considers the synchronization effects of fork-join tasks. In this paper, we extend the existing holistic analysis for the computation of the Worst-Case Response Time (WCRT) of sequential tasks in a distributed system, to *parallel* distributed (P/D) tasks. When considering P/D tasks, the transmission delays due to the messages exchanged by communicating threads within a P/D task cannot be considered negligible as it is the case on multiprocessor platforms. Furthermore, we consider the specific structure of the P/D tasks after applying the P/D-DMS algorithm, and its impact when computing their WCRT.

II. SYSTEM MODEL

Formally, we consider that a distributed real-time application is composed of a set $\tau = \{\tau_1, \dots, \tau_n\}$ of n *P/D tasks* [2]. Figure 1 shows an example of a P/D task τ_i. A P/D task τ_i is activated periodically every T_i time units and is characterised by an implicit end-to-end deadline D_i. Also, it is considered that all P/D tasks are released synchronously. A P/D task τ_i ($i \in \{1, \dots, n\}$) is composed of a sequence of sequential and parallel/distributed (P/D) segments $\sigma_{i,j}$ with $j \in \{1, \dots, n_i\}$. Where, n_i represents the number of segments composing τ_i, n_i is assumed to be an *odd* integer, as a P/D task should always start and finish with a sequential segment. Therefore, odd segments $\sigma_{i,2j+1}$ identify sequential segments and *even* segments $\sigma_{i,2j}$ identify P/D segments. Each segment $\sigma_{i,j}$ is composed of a set θ of threads $\theta_{i,j,k}$ with $k \in \{1, \dots, n_{i,j}\}$, where $n_{i,j} = 1$ for sequential segments and $n_{i,j} = m_i \leq m$ threads for P/D segments. m_i is the number of P/D threads in

each P/D segment, and it is considered to be the same for all P/D segments within a P/D task τ_i. m is the number of distributed nodes.

All sequential segments within a P/D task τ_i must execute within the same processor. This means that the processor that performs a D-Fork operation (*invoker processor*) is in charge of aggregating the result by performing a D-Join operation. Threads within a P/D segment are possibly executed on remote processors. Consequently, for each thread $\theta_{i,2j,k}$ belonging to a P/D segment (P/D thread), two P/D messages $\mu_{i,2j-1,k}$ and $\mu_{i,2j,k}$ are considered for realizing the communication between the invoker and remote processors. That is, P/D threads and messages that belong to a P/D segment and execute on a remote processor, have a precedence relation: $\mu_{i,2j-1,k} \longrightarrow \theta_{i,2j,k} \longrightarrow \mu_{i,2j,k}$. For each sequential and P/D segment, there exists a synchronisation point at the end of each segment, indicating that no thread that belongs to the segment after the synchronisation point can start executing before all threads of the current segment have completed execution. P/D threads are preemptive, but messages packets are non-preemptive, although large messages can be divided in several non-preemptive packets.

Also, each sequential thread $\theta_{i,2j+1,1}$ has a Worst-Case Execution Time (WCET) of $C_{i,2j+1,1}$. A P/D thread $\theta_{i,2j,k}$ has a WCET of $P_{i,2j,k}$, and each message $\mu_{i,j,k}$ has a Worst-Case Message Length (WCML) $\mathcal{M}_{i,j,k}$. It is assumed that for a task τ_i, every P/D thread $\theta_{i,2j,k}$ and their respective messages $\mu_{i,j,k}$ within a P/D segment $\sigma_{i,2j}$, have identical WCETs $P_{i,2j,k}$ and identical WCMLs $\mathcal{M}_{i,j,k}$, respectively. However, the WCET and the WCML of P/D threads and their messages can vary between different P/D segments. Also, P/D threads and P/D messages within a task τ_i, share the same period T_i.

To summarise, it is possible to describe a P/D task as:

$$\tau_i = ((C_{i,1,}, \mathcal{M}_{i,1}, P_{i,2}, \mathcal{M}_{i,2}, C_{i,3}, \dots, \mathcal{M}_{i,n_i-1}, C_{i,n_i}), m_i, T_i),$$

where:

- n_i is the total number of segments of a task τ_i,
- $C_{i,j}$ is the WCET of each sequential segment $\sigma_{i,2j+1}$,
- $\mathcal{M}_{i,j}$ is the WCML of a single messages (all m_i P/D messages on the same P/D segment have the same WCML),
- $P_{i,2j}$ is the WCET of a single P/D thread within a segment $\sigma_{i,2j}$ (all m_i P/D threads on the same P/D segment have exactly the same WCET),
- m_i is the number of P/D threads (two messages are created for each P/D thread within a P/D segment $\sigma_{i,2j}$) in each P/D segment,
- T_i is the period of a task, which is equal to its deadline ($D_i = T_i$).

A. Preliminaries

For notational convenience we introduce some definitions that will simplify the explanation of the P/D-DMS [2] algorithm.

Definition 1. (Master thread). *The master thread of a P/D task τ_i is the collection of all threads $\theta_{i,j,1}$ belonging to all segments $\sigma_{i,j}$. A master thread can be represented as:*

$$\tau_i^{master} = \{\theta_{i,1,1}, \theta_{i,2,1}, \theta_{i,3,1}, \dots, \theta_{i,n_i-1,1}, \theta_{i,n_i,1}\} \quad (1)$$

Definition 2. (Minimum execution length). *The minimum execution length η_i represents the minimum execution time a P/D task τ_i needs to execute, if all P/D threads are executed in parallel. This is equal to the sum of the WCET of all the threads described in the master thread:*

$$\eta_i = \left(\sum_{j=0}^{\frac{n_i-1}{2}} C_{i,2j+1}\right) + \sum_{j=1}^{\frac{n_i-1}{2}} P_{i,2j,1} \quad (2)$$

Definition 3. (Maximum execution length). *The maximum execution length C_i, represent the maximum execution time a P/D task τ_i needs to execute when all P/D threads are executed sequentially on the invoker processor. This is equal to the sum of WCET of all threads in a task τ_i:*

$$C_i = \left(\sum_{j=0}^{\frac{n_i-1}{2}} C_{i,2j+1}\right) + \left(\sum_{j=1}^{\frac{n_i-1}{2}} P_{i,2j,1}\right) \times m_i \quad (3)$$

Definition 4. (Slack time). *The positive slack time L_i is the temporal difference between the task's deadline D_i and the minimum execution length η_i:*

$$L_i = D_i - \eta_i \quad (4)$$

If the slack L_i is a negative number, it means that η_i is larger than its deadline ($T_i = D_i$). Therefore, such a task is not schedulable on any number of processors with a speed of 1.

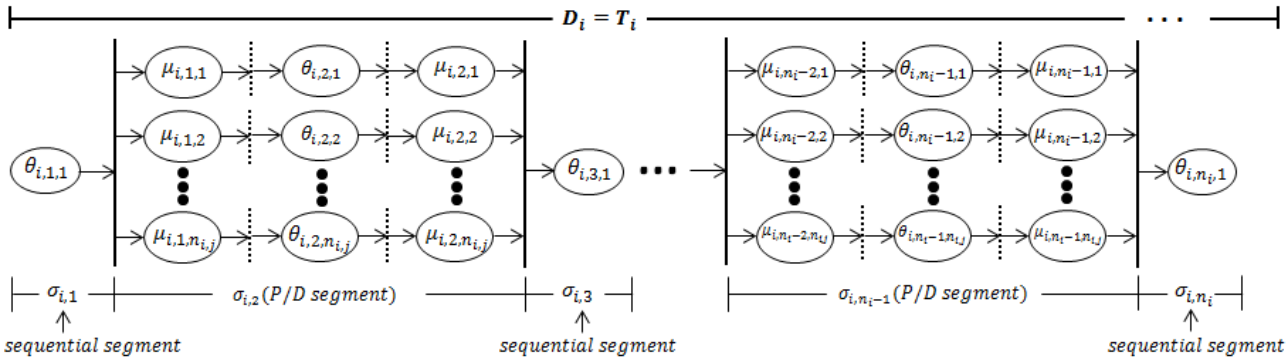

Fig. 1. The fork-join parallel/distributed periodic real-time tasks (P/D task) model.

Definition 5. (Task Capacity). *The task capacity f_i is defined as the capacity of the master thread of a task τ_i to execute extra P/D threads from all P/D segments without missing its deadline:*

$$f_i = \frac{L_i}{\sum_{j=1}^{\frac{n_i-1}{2}} P_{i,2j,1}} \quad (5)$$

III. The P/D-DMS Algorithm

P/D-DMS algorithm [2] is a dispatching algorithm for partitioning a set τ of P/D tasks τ_i onto the elements of the distributed system. The P/D-DMS algorithm realizes the dispatching by: (i) applying the DST [2] to each P/D task τ_i in τ, and (ii) partitioning the set of remaining P/D threads after applying the DST onto processors according to the FBB-FFD algorithm [3]. P/D messages $\{\mu_{i,j,k}^{cd}\}$ are scheduled according to the fixed priority scheduling policy of the network.

The DST was inspired by the SST transformation model [5]. The DST also opts for the formation of a stretched master thread $\tau_i^{stretched}$ for each P/D task τ_i. However, the DST addresses some specific constraints that are related to distributed systems. For example, when realising a D-Fork operation, it implies that some messages will be transmitted within the network, affecting the execution length of the P/D tasks. Let us illustrate the DST transformation with an example. Consider two tasks: $\tau_1 = ((1,1,2,1,1),3,8)$, and $\tau_2 = ((1,1,3,1,1),3,10)$ to be scheduled on 3 processors. The DST transformation is illustrated in Figure 2. By calculating the maximum execution length (Definition 3) of tasks τ_1 and τ_2, we obtain $C_1 = 8$ and $C_2 = 11$. Then, by looking at Figure 2, it is possible to observe two cases:

1. $C_i \leq T_i$. This is the case of τ_1 in our example; whenever such a case appears for a task τ_i, the task τ_i is fully stretched into a master thread and handled as a sequential task with execution time equal to C_i, a task period of T_i, and an implicit deadline equal to D_i. Therefore, no messages are generated for transmission on the network.
2. $C_i > T_i$. This is the case of τ_2 in our example; for such tasks, the DST transformation inserts (coalesces) as many P/D threads of τ_i into the master thread as possible. To do so, it is first needed to calculate the available slack and capacity of task τ_i as indicated in Eq. (4) and (5). For τ_2, it gives $L_2 = 10 - 5 = 5$ and, $f_2 = 5/3$. Thus, the number of P/D threads that each P/D segment can fully insert into the master thread without causing τ_i to miss its deadline is given by:

$$i_{i,2j} = \lfloor f_i \rfloor \quad (7)$$

In the case of τ_2, $i_{2,2} = \lfloor f_2 \rfloor = 1$. Figure 2 shows that τ_2 executes two P/D threads per P/D segment on the invoker processor rather than only one when considering the non-stretched master thread.

The number $q_{i,2j}$ of the remaining P/D threads that have not been coalesced into the master thread is given by:

$$q_{i,2j} = m_i - i_{i,2j} \quad (8)$$

The slack f_i of task τ_i is equally distributed between all the P/D segments of a P/D task τ_i. Thus, the maximum scheduling length for the subset of P/D threads and their respective P/D messages is determined by defining a set of P/D intermediate deadlines $d_{i,2j}$:

$$d_{i,2j} = (f_i + 1) \times P_{i,2j} \;\; \forall\, 1 \leq j \leq \frac{n_i - 1}{2} \quad (9)$$

Thus, at the end of the DST transformation, a P/D task τ_i will be composed of: (i) a single stretched master thread $\tau_i^{stretched}$, and a set of constrained deadline P/D threads $\{\theta_{i,j,k}^{cd}\}$, and their respective constrained deadline messages $\{\mu_{i,jk}^{cd}\}$; per each P/D segment $\sigma_{i,2j}$, or (ii) a single fully stretched sequential task. The stretched master thread $\tau_i^{stretched}$ is assigned to its own processor. The remaining single fully stretched sequential tasks and P/D threads $\{\tau_i^{cd}\}$ are assigned to processors with the FBB-FFD algorithm [3]. Messages $\{\mu_{i,jk}^{cd}\}$ are assigned to the real-time network and scheduled accordingly.

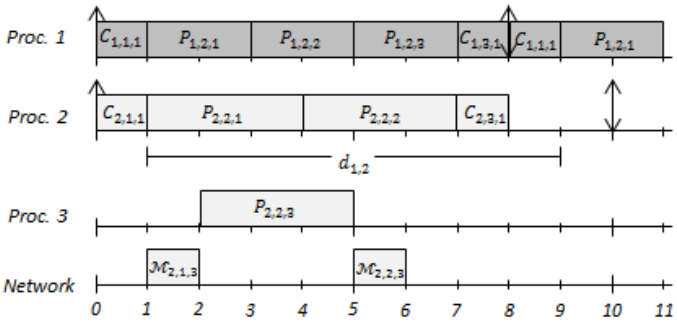

Fig. 2. A task scheduled by the P/D-DMS algorithm.

IV. Holistic Analysis for P/D tasks

The holistic analysis has been conceived as a solution for the analysis of the interaction of a system composed by a set of different processing devices (e.g. processors and networks) [6]. One of the main goals of the holistic analysis approach is to calculate the so called end-to-end delay. The end-to-end delay is the WCRT associated to a chain of tasks executing on the same processor or different processors and interchanging messages for communication purposes. A holistic approach considers the analysis of such a chain of dependencies, which implies higher degree of difficulty when compared to the analysis of components in isolation. However, since the parameters in a holistic analysis are dependent but monotonic, it is possible to formulate a recurrence and progressively iterate until finding a stable solution.

The holistic analysis relies on a simple concept of attribute inheritance, for instance, the activation (release) of a P/D message or P/D thread is based on the response time of previous processing event (e.g. P/D threads or P/D message, respectively). It is possible to observe in Figure 2, that after thread $\theta_{2,1,1}$ has completed execution, the transmission of the message $\mu_{2,1,3}$ is triggered, and in turn this message triggers the execution of the P/D thread $\theta_{2,2,3}$.

Fully stretched tasks: when considering tasks that are fully stretch into a sequential task, no transmissions are required (Case 1, Section III). Therefore, their WCRT only depends on the suffered interference caused by other higher priority threads executing on the same processor.

Non-fully stretched tasks: when considering non-fully stretched tasks, it is necessary to consider the sequential and parallel segments independently. Let us recall that for each sequential and P/D segment, there exist a synchronisation point at the end of each segment, in which threads of the next segment can only continue their execution whenever all threads of the current segment have completed their execution. Therefore, the WCRT of a task τ_i is computed based on the sum of the maximum execution paths of each segment $\sigma_{i,j}$:

$$WCRT(\tau_i) = \sum_{j=1}^{n_i} \max_j(WCRT(\sigma_{i,j})) \quad (10)$$

Sequential segments $\sigma_{i,2j+1}$ within a P/D task, are executed on their own processor, therefore, they do not suffer any interference from other threads. Thus, the maximum WCRT is equal to the WCET ($C_{i,2j+1,1}$) of the corresponding thread $\theta_{i,2j+1,1}$:

$$WCRT(\sigma_{i,2j+1}) = C_{i,2j+1,1} \quad (11)$$

For parallel segments $\sigma_{i,2j}$ within a P/D task, the maximum WCRT is given by the maximum WCRT of two possible scenarios:

1. the sum of all coalesced P/D threads (denoted as $CThr_{i,2j}$) within the master thread which are executed sequentially:

$$WCRT(CThr_{i,2j}) = \sum_{\forall \theta_{i,2j,k} \in \sigma_{i,2j} \wedge \in master\ thread} P_{i,2j,k} \quad (12)$$

2. or, the $WCRT^{max}_{DP_{i,2j}}$, which is the maximum WCRT of the k distributed execution paths (denoted as $DP_{i,2j,k}$); per each P/D segment. A distributed execution path is the execution of a P/D thread that has not been coalesced with the master thread and their respective P/D messages that have a precedence relation: $\mu_{i,2j-1,k} \longrightarrow \theta_{i,2j,k} \longrightarrow \mu_{i,2j,k}$. For calculating WCRT of a distributed execution path, it is possible to use the following equation:

$$WCRT(DP_{i,2j,k}) = WCRT(\mu_{i,2j-1,k}) + WCRT(\theta_{i,j,k}) + WCRT(\mu_{i,2j,k}) \quad (13)$$

Then, it is needed to find the maximum $WCRT(DP_{i,2j,k})$ as:

$$WCRT^{max}_{DP_{i,2j}} = \max_k(WCRT(DP_{i,2j,k})) \quad (14)$$

Thus, for each P/D segment $\sigma_{i,2j}$ within a P/D task, the maximum WCRT is equal to:

$$WCRT(\sigma_{i,2j}) = \max(WCRT(CThr_{i,2j}), WCRT^{max}_{DP_{i,2j}}) \quad (15)$$

Also, it is important to note that the WCRT of threads and messages depend on the characteristics of the processing elements, and on the particular method to calculate the WCRT. For example, for computing the WCRT of a P/D thread, it is needed to consider the characteristics of the computing nodes (e.g. uniprocessor nodes, multicore nodes, etc.). Likewise for computing the WCRT of P/D messages, it is needed to consider the specific characteristics of the real-time networks (e.g. CAN, FTT-SE, etc.). However, when considering the P/D task model and using a task transformation as the DST, the reasoning of our generic holistic analysis can be used.

V. Conlusions and Future work

In this paper, we presented a generic holistic analysis approach. This analysis studies the specific structure of the P/D tasks after applying the P/D-DMS algorithm, and its impact when computing their WCRT. Although the methods to compute the WCRT of P/D tasks depend on the specific characteristics of computing resources and networks, the holistic analysis reasoning presented in this paper is completely generic. Hence, we are currently working on extracting some characteristics of the P/D task model along with properties of specific communication protocols such as the Flexible Time Triggered protocol, with the intention of improving the computation of the WCRT for parallel tasks in a distributed environment.

Acknowledgments

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project FCOMP-01-0124-FEDER-037281 (CISTER); by FCT and the EU ARTEMIS JU funding, within projects ENCOURAGE (ARTEMIS/0002/2010, JU grant nr. 269354), ARROWHEAD (ARTEMIS/0001/2012, JU grant nr. 332987), CONCERTO (ARTEMIS/0003/2012, JU grant nr. 333053); by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/71562/2010.

References

[1] R. Garibay-Martinez, L. L. Ferreira and L. M. Pinho, "A framework for the development of parallel and distributed real-time embedded systems," in *Proc. of 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2012)*, 2012.

[2] R. Garibay-Martínez, G. Nelissen, L. L. Ferreira and L. M. Pinho, "On the Scheduling of Fork-Join Parallel/Distributed Real-Time Tasks," in *Proc. of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES'14), to appear*, 2014.

[3] N. Fisher, S. Baruah and T. P. Baker, "The partitioned scheduling of sporadic tasks according to static-priorities," in *Proc. of the IEEE 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, 2006.

[4] P. Axer, S. Quinton, M. Neukirchner and R. Ernst, "Response-Time Analysis of Parallel Fork-Join Workloads with Real-Time Constraints," in *Proc. IEEE 25th Euromicro Conference on Real-Time Systems (ECRTS'13)*, 2013.

[5] M. Qamhieh, F. Fauberteau and S. Midonnet, "Performance Analysis for Segment Stretch Transformation of Parallel Real-time Tasks," in *Proceedings of the 5th Junior Researcher Workshop on Real-Time Computing (JRWRTC 2011)*, 2011.

[6] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and Microprogramming*, vol. 40, no. 2-3, pp. 117 - 134 , 1994.