Quantifying WCET reduction of parallel applications by introducing slack time to limit resource contention*

Sébastien Martinez
Université de Rennes 1/IRISA
Campus de Beaulieu
Rennes, France 35042
sebastien.martinez@
telecom-bretagne.eu

Damien Hardy
Université de Rennes 1/IRISA
Campus de Beaulieu
Rennes, France 35042
damien.hardy@irisa.fr

Isabelle Puaut
Université de Rennes 1/IRISA
Campus de Beaulieu
Rennes, France 35042
isabelle.puaut@irisa.fr

ABSTRACT

In parallel applications, concurrently running tasks cause contention when accessing shared memory. In this paper, we experimentally evaluate how much the *Worst-Case Execution Time* (WCET) of a parallel application, already mapped and scheduled, can be reduced by the introduction of slack time in the schedule to limit contention. The initial schedule is a time-triggered non-preemptive schedule, that does not try to avoid contention, generated with a heuristic technique. The introduction of slack time is performed using an optimal technique using Integer Linear Programming (ILP), to evaluate how much at best can be gained by the introduction of slack time. Experimental results using synthetic task graphs and a Kalray-like architecture with round-robin bus arbitration show that avoiding contention reduces WCETs, albeit by a small percentage. The highest reductions are observed on applications with the highest memory demand, and when the application is scheduled on the highest number of cores.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**;

KEYWORDS

Slack time introduction, Real-time systems, Scheduling, WCET, Shared memory contention, Integer linear programming

ACM Reference format:
Sébastien Martinez, Damien Hardy, and Isabelle Puaut. 2017. Quantifying WCET reduction of parallel applications by introducing slack time to limit resource contention. In *Proceedings of RTNS'17, October 2017, International Conference on Real-Time Networks and Systems,* 10 pages.
https://doi.org/10.1145/3139258.3139263

1 INTRODUCTION

Calculating the *Worst-Case Execution Time* (WCET) of parallel applications executing on multi-cores requires to put particular attention to shared hardware resources (shared bus, memory or input/output subsystems). Concurrent accesses to shared resources may force tasks to wait for their availability as other tasks may concurrently be using these resources. As a consequence, the WCET of a task, estimated in isolation without any parallel task competing for shared resources, is no longer an upper bound of the time required for the task to complete. An overhead considering concurrent accesses to shared resources must be added to the WCET of each task to obtain a safe upper bound. The resource contention overhead depends on the task itself, but also on the execution platform and on the concurrent tasks, which make its calculation complex and tightly linked to the way tasks are scheduled.

The execution platform defines the resource arbitration policy when concurrent accesses to shared resources occur and therefore impacts the overhead on each task. For example, for round-robin bus arbitration, a bus access by one core is delayed by at most $N_c - 1$ potential pending accesses from tasks running on the other cores, with N_c the number of cores. Moreover, to calculate an upper bound on the execution time of a task, it is also necessary to identify which other task may be executed concurrently with it. Although this identification of concurrent tasks may be very pessimistic when no knowledge of the application and scheduling is available, some knowledge of the scheduling helps reducing the pessimism, in particular when considering static time-triggered schedules.

In this paper, our objective is to evaluate if the introduction of slack time in a static time-triggered schedule, to avoid interferences between tasks, is beneficial, and to quantify the improvement.

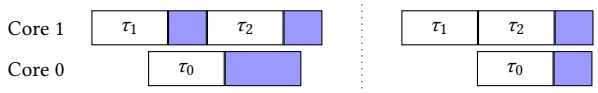

Figure 1: Example of slack time introduction

Figure 1 depicts a motivating example. On the left part, tasks τ_0, τ_1 and τ_2 are scheduled without trying to minimize interferences. To take contention into account, a worst case overhead (shaded area), induced by concurrent accesses to the shared memory, is added to the tasks WCET. On the right part of the figure, slack time is introduced before task τ_0, such that it now runs concurrently only with τ_2. By avoiding contention with τ_1, the overhead on τ_0 is reduced and the overhead on τ_1 is removed, thereby reducing the global execution time of the application.

In order to evaluate the interest of introducing slack time in time-triggered schedules to mitigate interferences, we developed

*This work was partially supported by PIA project CAPACITES (Calcul Parallèle pour Applications Critiques en Temps et Sûreté), reference P3425-146781 and by H2020 project ARGO (http://www.argo-project.eu/), funded by the European Commission under Horizon 2020 Research and Innovation Action, Grant Agreement Number 688131.

RTNS'17, International Conference on Real-Time Networks and Systems, October 2017

ACM ISBN 978-1-4503-5286-4/17/10…$15.00
https://doi.org/10.1145/3139258.3139263

an Integer Linear Programming (ILP) system that calculates the optimal slack time to be introduced before each task to minimize the application's WCET, that we compare with the initial WCET (*i.e.* WCET without contention reduction).

This work focuses on programs modeled as Directed Acyclic Graphs (DAGs) in which nodes represent tasks and edges represent precedence relations between them. Experiments are conducted on synthetic task graphs, with representative WCETs and memory demand parameters. The original schedules are generated using a contention-agnostic heuristic based on list scheduling, augmented to consider contentions using the algorithm described in [14]. Experimental results show an improvement from 0.01% to 20% depending on the application's memory demand and the number of cores, the highest reductions occurring for the highest memory demands and number of cores.

Our interest here being to quantify the gain resulting from the introduction of slack time, the solution we developed was designed to compute the *optimal* solution (best location in the initial schedule where to introduce slack time). It appears that time required to calculate the optimal schedule does not scale well with the number of tasks in the initial schedule. Designing more time-efficient solutions is left for future work.

Although designed and evaluated for a single application modeled as a DAG, the proposed technique can be used for more general task models (multiple applications, independent tasks, periodic tasks) provided that a static time-triggerred schedule exists.

The rest of the paper is organized as follows. Sections 2 and 3 present respectively the architecture and task models. Section 4 details the ILP system used for optimally inserting slack time in time-triggered schedules to reduce contention. Section 5 details the experimental protocol we followed and analyzes the obtained results. Section 6 uses a more precise, but more computation expensive, contention model to estimate a bias on the obtained results due to the selected model. Section 7 presents related work. Finally, Section 8 concludes the paper.

2 PLATFORM MODEL

This paper considers an execution platform based on the Kalray MPPA-256 [5] processor. The latter is composed of compute clusters, each consisting of 16 processing elements plus one resource manager, sharing the same memory subsystem. Compute clusters are interconnected by a torus Network on Chip (NoC). In a compute cluster of the Bostan version of the MPPA-256, the shared memory is divided into 16 banks, each accessed through its own bus. Accesses to the buses by the processing elements are arbitrated by a round-robin policy, allowing each processing element to make one access to the shared memory before releasing the resource and letting the other processing elements access it. In the Bostan version of the MPPA-256, one access through the bus to the shared memory costs up to 10 cycles and loading a 64 bytes block from shared memory costs up to 17 cycles (9 cycles with 8 bytes fetched on each consecutive cycle [5]). Without contention, an access to the shared memory by a processing element takes up to 17 cycles while contention may cause a single access to take up to 167 cycles (1 access delayed by one access by each other 15 processing elements).

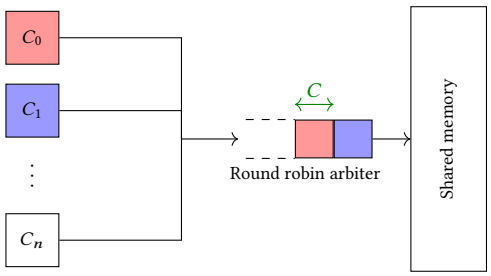

Figure 2: Bus arbitration (round-robin)

The platform considered in this study is an abstraction of a single MPPA-256 compute cluster. The parallel application is mapped on a variable number of cores (denoted by $N_c \leq 16$). A shared memory subsystem is accessed through one single bus, arbitrated by a round-robin policy, as depicted on Figure 2. All cores have access to a common time base. The maximum cost (in cycles) of an access to the shared memory (without contention) is denoted by S and the cost (in cycles) of an access through the bus is denoted by C. In the MPPA-256, $S = 17$ and $C = 10$. This platform was selected not only because of the context of the work (collaborative research project involving Kalray) but also because and most importantly because of the predictability of the MPPA-256 architecture and its timing-compositionality [10].

3 TASK MODEL

This paper adopts the following task model: a program is modeled by a Directed Acyclic Graph (DAG) with tasks as nodes and dependency relations between tasks as edges. For each task, two properties are considered: its Isolated Worst-Case Execution Time (IWCET) and its highest memory demand. The IWCET of a task is its worst-case execution time when executed alone on one core of the platform, without any concurrent task running on the other core. Its highest memory demand is an upper bound on the number of shared memory accesses the task can make. The IWCET of a task includes the time taken by accesses to the shared memory, including the constant S of the execution platform. Tasks do not have individual deadlines. Without loss of generality, we focus in this paper on a single instance of an application modeled by a DAG. The proposed technique can be used for more general task models (multiple applications, periodic tasks, independent tasks) as long as a static time-triggered schedule is available.

4 MINIMIZING CONTENTION INDUCED OVERHEADS

We consider a non-preemptive time-triggered schedule, assigning N tasks from a single task graph to N_c cores and defining start and finish dates for each of them (*e.g.* left schedule on Figure 1). We assume this schedule respects dependencies: the start date of a task is always higher than or equal to the finish date of all its dependencies. We call that schedule *the initial schedule*.

We define an Integer Linear Programming (ILP) system for producing a new contention-aware schedule with minimal WCET by optimally delaying the start dates of some tasks, in order to limit

	Constants	
N	Number of tasks in the schedule	
N_c	Number of cores used by the schedule, in the execution platform	
τ_j, τ_k	Tasks	
$IWCET_j$	Isolated Worst-Case Execution Time of τ_j (without contention induced overhead)	
B_j	Highest memory demand of τ_j (number of accesses)	
C	Cost (in cycles) of an access through the shared memory bus	
S	Maximum cost (in cycles) of an access to the shared memory without contentions	
	Variables	
s_j	Start date of τ_j in the produced schedule	
f_j	Finish date of τ_j in the produced schedule	$f_j = s_j + IWCET_j + o_j$
o_j	Overhead induced by tasks interfering with τ_j	
	Boolean variables	
p_j^k, q_j^k	Indicates precedence between τ_j and τ_k	$p_j^k = 1 \Leftrightarrow \tau_j$ finishes before τ_k starts. $q_j^k = \neg p_j^k$
$\delta_{j,k}$	Equals 1 if τ_j and τ_k interfere, 0 if they don't	$\delta_{j,k} = 1 \Leftrightarrow \tau_j$ and τ_k interfere
y_j^k	Indicates whether interference on τ_j from core k are bounded by B_j	
	Objective function	
x	Global WCET of the program (finish date of the latest finishing task)	

Figure 3: Notations of the ILP model

contention in some parts of the schedule. Two types of information are extracted from the initial schedule: (i) the mapping of tasks to cores and (ii) the ordering of tasks on each core. The produced schedule keeps these mappings and orderings and defines new start and finish dates for each task, minimizing contention induced overhead. The produced schedule is optimal: it corresponds to an optimal introduction of slack time, achieving minimal WCET of the whole schedule (*i.e.* the finish date of the latest finishing task is minimized). The notations used in the ILP system are summarized in Figure 3.

4.1 Objective function

Our objective is to minimize the WCET of the produced schedule (denoted by x), which is the finish date of the latest finishing task in the produced schedule. We define the objective of the ILP system as:

$$\textit{Minimize } x \textit{ where } x \geq f_j \quad \forall j \in [0, N-1]$$

4.2 Variables

Figure 3 details the variables, constants and their definitions. The main variables are s_j, f_j (start and finish times of task τ_i), and x which altogether describe the produced schedule and its duration. The other variables are used as intermediary for calculating the values of the main variables.

4.3 Constraints

In our modeling of the problem, the total execution time of a task is equal to its IWCET plus the overhead induced by shared memory contention. This gives the following constraint for each task:

$$\forall j \in [0, N-1] \; f_j = s_j + IWCET_j + o_j \tag{1}$$

Dependencies, precedences and non-preemption. Delaying the execution of tasks in the schedule must neither contradict the ordering of tasks on a particular core nor contradict dependency relations between tasks. To express these constraints, we note O the set of ordered pairs (j,k) of $[0, N-1]^2$ such that τ_j is either a dependency of τ_k in the task graph, or precedes τ_k on the same core in the initial schedule. These constraints also ensure that the produced schedule remains non-preemptive because any two jobs scheduled on the same core are ordered by the initial schedule.

$$\forall (j,k) \in O : \quad s_k \geq f_j \tag{2}$$

Tasks happening in parallel. Let us note $\mathscr{P}$ the set of all pairs $\{j,k\}$ with $j \neq k$ of $[0, N-1]^2$ such that τ_j and τ_k happen in parallel in the original schedule. Set $\mathscr{P}$ can be built using dependency relations between tasks and the order given by the initial schedule. Any two tasks that are not bonded by a dependency relation and that are not mapped on the same core are part of $\mathscr{P}$. The definition of p_j^k and q_j^k gives the following constraints.

$$\begin{cases} \forall \{j,k\} \in \mathscr{P}, \; q_j^k - \mathscr{M}\, p_j^k \leq f_j - s_k < \mathscr{M}\, q_j^k + p_j^k \\ \forall \{j,k\} \in \mathscr{P}, \; q_k^j - \mathscr{M}\, p_k^j \leq f_k - s_j < \mathscr{M}\, q_k^j + p_k^j \\ \forall \{j,k\} \in \mathscr{P}, \; p_j^k + q_j^k = 1 \\ \forall \{j,k\} \in \mathscr{P}, \; p_k^j + q_k^j = 1 \end{cases} \tag{3}$$

With $\mathscr{M}$ a *big-M* constant, upper bound on all f_j.

For a given pair of jobs $\{\tau_j, \tau_k\}$, three situations can happen : (1) τ_j finishes before τ_k starts ($p_j^k = 1, p_k^j = 0$), (2) τ_k finishes before τ_j starts ($p_k^j = 1, p_j^k = 0$) and (3) neither finishes before the other starts, both tasks happen in parallel ($p_k^j = 0, p_j^k = 0$). The latter is expressed with the following constraints:

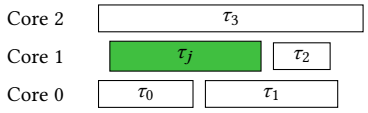

Figure 4: Example of interfering tasks

$$\begin{cases} \forall \{j,k\} \in \mathscr{P},\ \delta_{j,k} = q_j^k \wedge q_k^j \\ \forall \{j,k\} \notin \mathscr{P},\ \delta_{j,k} = 0 \end{cases} \tag{4}$$

4.4 Contention induced overhead

In our modeling of the problem, any two tasks executed in parallel cause shared memory contention. To calculate the overhead induced on a task τ_j, we consider the round-robin arbiter of the shared memory bus in the execution platform. Each access through the bus takes a fixed number of cycles denoted by C as previously depicted in Figure 2. When several cores request an access to the bus at the same time, the arbiter lets them make one access before letting the other cores access the bus. Therefore, one access to the shared memory may be delayed for up to $(N_c - 1) * C$ cycles because of contention.

Let us consider the example on Figure 4. Task τ_j is executed in parallel with τ_3, τ_0 and τ_1. In the worst case, all the tasks make their maximum number of accesses to the shared memory. On core 2, τ_3 accesses the shared memory up to B_3 times and, on core 0, τ_0 and τ_1 make up to $B_0 + B_1$ accesses. As a consequence, τ_j mapped on core 1 undergoes up to $\texttt{min}(B_j, B_3)$ delays from core 2 and up to $\texttt{min}(B_j, B_0 + B_1)$ delays from core 0. The overhead on τ_j induced by shared memory contention is therefore

$o_j = C * (\texttt{min}(B_j, B_3) + \texttt{min}(B_j, B_0 + B_1))$.

We define $o_j^k\ \forall k \in [0, N_c - 1]$, the overhead induced on τ_j by accesses from core k and we generalize the previous example to write the following constraints.

$$\forall j \in [0, N-1], \text{assuming } \tau_j \text{ is mapped on core } k_j$$

$$\begin{cases} \forall k \in [0, N_c - 1] \backslash \{k_j\},\ o_j^k \geq C * \sum_{t=0}^{N-1} \delta_{j,t} * B_t - y_j^k * \mathscr{M}' \\ \forall k \in [0, N_c - 1] \backslash \{k_j\},\ o_j^k \geq C * B_j * y_j^k \\ \forall k \in [0, N_c - 1] \backslash \{k_j\},\ o_j^k \leq C * \sum_{t=0}^{N-1} \delta_{j,t} * B_t \\ \forall k \in [0, N_c - 1] \backslash \{k_j\},\ o_j^k \leq C * B_j \\ o_j^{k_j} = 0 \\ o_j = \sum_{k=0}^{N_c - 1} o_j^k \end{cases} \tag{5}$$

With $\mathscr{M}'$ a *big-M* constant, upper bound on the sum of all B_t. The first four lines of the equation linearize the `min` function in the expression of the overhead induced on τ_j by concurrent tasks mapped on core k. The first two lines express lower bounds on o_j^k while the next two lines express upper bounds. The last two lines of the equation express that the total overhead on τ_j is the sum of the overhead induced by each core and that core k_j on which τ_j is mapped induces no overhead.

The contention model used in this section implicitly assumes that tasks, when executing in parallel, interfere to the full extent of their memory demand, and is thus pessimistic. A tighter but harder to analyze contention model is presented in Section 6.

5 EXPERIMENTS

This section evaluates the interest of slack time introduction in static time-triggered schedules and estimates the best WCET gains that can be expected from that method. It first details the experimental protocol, then presents and analyzes the obtained results.

5.1 Experimental protocol

The experimental protocol is depicted in Figure 5. It begins by the generation of 100 task graphs using the TGFF task generator [6] (step (a)) and the random generation of the IWCET and highest memory demand of each tasks (step (b)). On step (c), the task graph is scheduled using a contention-agnostic list scheduling algorithm. The schedule is then updated by the algorithm designed by Rihani et al. [14] to take contention into account (step (d)). On step (e), the updated schedule and the task graph are used as inputs of the ILP system described in Section 4 for producing a schedule of minimal global WCET. On step (f), the minimal WCET obtained at the end of step (e) is then compared with the non-optimized WCET obtained at the end of step (d).

On step (a) and (b), 100 task graphs of 30 to 70 tasks with IWCETs varying between 10.000 and 300.000 cycles are generated. The number of parallel branches varies between 1 and 31 with an arithmetic mean of 16. These parameters correspond to the number of tasks and the IWCETs observed in parallel programs (here we used the IWCET ranges observed on the StreamIt benchmarks [16]). The highest memory demand B_τ of each task is generated randomly, based on each task's IWCET. We define r_τ the memory ratio of task τ as the percentage of its IWCET dedicated to accessing the shared memory, without contention and assuming τ makes its maximum number of accesses (B_τ).

$$r_\tau = \frac{S * B_\tau}{IWCET_\tau} \text{ with } S \text{ the worst case cost (in cycles) of an access to shared memory} \tag{6}$$

Each of the 100 task graphs is instantiated 10 times with different bounds on r_τ in percent: [0.1 - 1], [1 - 2], [2 - 4], [4 - 6], [6 - 8], [8 - 10], [10 - 15], [15 - 20], [20 - 25], [25 - 30]. The intervals are narrow for small values of r_τ and wider for larger values of r_τ. For each task τ and each interval, a value of r_τ is randomly generated within the bounds an B_τ is calculated using equation 6 with $S = 17$ (which is the value of S in the MPPA-256).

A separate control group is created, to obtain experimental results for tasks with more heterogeneous memory ratios. The control group contains the 100 aforementioned task graphs, with a value of r_τ randomly generated in interval [0.1 - 30].

The bounds on r_τ were determined using the static WCET analysis tool Heptane [11] on benchmarks from the Mälardalen WCET benchmark suite[1] with the cache sizes of the MPPA-256 (32KB for the instruction cache, 8KB for the data cache), and with smaller cache sizes to simulate memory intensive programs.

[1] http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

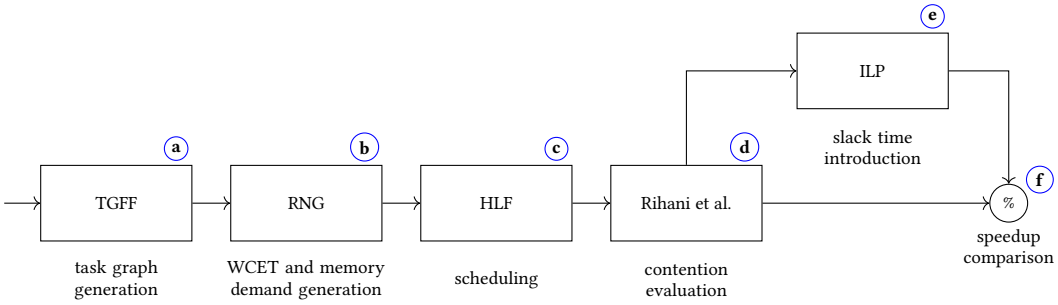

Figure 5: Experimental protocol

On step (c), a Highest Level First (HLF) list scheduling algorithm is used to schedule each instance of the task graphs on 2, 4, 8 and 16 cores. Given a task graph, the algorithm defines the weight of each task as the sum of its IWCET and the weight of the heaviest (*i.e.* of highest weight) task that depends on it. Tasks are sorted by decreasing weight then scheduled one by one, without backtracking, on the core that allows the earliest start date. The HLF algorithm is contention-agnostic and guarantees that no task starts before all its predecessors are finished.

On step (d), we use the algorithm by Rihani et al. [14] to produce a contention-aware schedule. The algorithm iteratively updates the schedule by calculating the overhead on each task induced by contention, then updating the start and finish date of each task. The algorithm safely estimates contention overheads, but when doing so does not try to reduce contentions, leaving room for further optimizations by our technique.

On step (e), we use the Cplex ILP solver on the system described in Section 4 with $C = 10$, which is the number of cycles it takes to access the shared memory bus in the MPPA-256. The ILP system is used on each of the 4400 generated schedules with a time limit of four hours. The results presented in the next subsection were obtained after six days of computation on a computing grid, and represent 1033 different configurations.

On step (f), for each scheduled instance of the task graphs, we compare the global WCET obtained after step (d) ($WCET_d$) and the optimal global WCET obtained after step (e) ($WCET_e$). We define the gain g_p as follows:

$$g_p = \frac{WCET_e - WCET_d}{WCET_d} \tag{7}$$

Please note that because the ILP system uses the same model as Rihani et al. for shared memory contention, g_p is never negative. In the worst case, $WCET_e$ is already the minimal $WCET$ and $g_p = 0$.

5.2 Results

Figure 6 presents statistics, in percentage, on the gains measured for different bounds on r_τ and the different numbers of cores (arithmetic mean, standard deviation σ, minimum and maximum gain). In general, for a given set of parameters, the mean gain is low compared to the maximum gain observed. This can be explained by a consequent number of schedules that do not benefit from slack time introduction. We also observe that when r_τ and N_c increase, the mean gain and the standard deviation have higher values. While some schedules still benefit poorly from slack time introduction, more schedules reach higher gains. For example, when $2\% \leq r_\tau \leq 4\%$ and $N_c = 16$, the mean gain is 0.91% and $\sigma = 1.01\%$ for a maximal gain of 4.43% whereas, when $20\% \leq r_\tau \leq 35\%$ and $N_c = 16$, the mean gain is 5.22% and $\sigma = 4.21\%$ for a maximal gain of 11.3%.

A lower mean gain with a lower value of σ indicate that the majority of observed gains are close to the mean gain which is low. The majority of schedules benefit poorly from slack time introduction and reach lower gains. This is observed for lower values of r_τ and N_c. A higher mean gain with a higher value of σ indicate that a wider range of gains was observed. As r_τ and N_c increase, more schedules benefit from slack time introduction, and while some of them reach higher gains, some of them also reach smaller gains.

Figures 7 (respectively 8) presents the evolution of gains when increasing the number of cores (respectively the memory ratio). They also show that the mean gain increase with N_c (respectively r_τ). However, their values stay low compared to Figure 6. This can be explained by the impact of r_τ (respectively N_c). For example, on figure 8, the statistics for $15\% \leq r_\tau \leq 20\%$ consider schedules on any number of cores. A large number of schedules on 2 or 4 cores benefit poorly from slack time introduction while some schedules on 8 or 16 cores reach higher gains. As a result, the mean gain and the standard deviation stay low.

Figure 9 presents the statistics for the control group, corresponding to all task graphs, with heterogeneous values of r_τ. As observed on Figures 6 and 7, the mean gain and σ increase with N_c. Although their values do not reach values obtained on Figure 6, with homogeneous values of r_τ, they show a positive impact of slack time introduction in general and confirm that the gains increase when N_c increases.

Results show that task graphs with greater memory ratios, scheduled on more cores have a higher probability of benefiting from slack time introduction and also can expect a higher gain. The more cores used by the schedule, the more tasks run concurrently and cause contention. The higher the memory ratio, the higher the impact of contention on the WCET of a task. Indeed, in our execution platform, the overhead induced by contentions on a given task can be approximated as an additive function of the highest memory demand of the concurrent tasks.

Parameters		Gain statistics (in percentage)				Parameters		Gain statistics (in percentage)			
$a \leq r_\tau \leq b$	N_c	mean	σ	min	max	$a \leq r_\tau \leq b$	N_c	mean	σ	min	max
0.1% - 1%	2	0.01	0.01	0	0.04	8% - 10%	2	0.08	0.10	0	0.48
0.1% - 1%	4	0.04	0.03	0	0.13	8% - 10%	4	0.45	0.23	0	0.96
0.1% - 1%	8	0.12	0.11	0	0.36	8% - 10%	8	1.12	0.71	0.32	2.33
0.1% - 1%	16	0.22	0.24	0	0.91	8% - 10%	16	1.59	1.74	0	6.14
1% - 2%	2	0.02	0.02	0	0.12	10% - 15%	2	0.13	0.17	0	0.89
1% - 2%	4	0.09	0.10	0	0.42	10% - 15%	4	0.60	0.36	0.08	1.60
1% - 2%	8	0.16	0.17	0	0.48	10% - 15%	8	1.22	0.77	0.13	2.53
1% - 2%	16	0.36	0.34	0	1.37	10% - 15%	16	1.96	2.13	0.01	8.04
2% - 4%	2	0.04	0.04	0	0.17	15% - 20%	2	0.23	0.21	0	0.79
2% - 4%	4	0.15	0.15	0	0.70	15% - 20%	4	1.22	0.66	0.42	2.53
2% - 4%	8	0.45	0.63	0	2.81	15% - 20%	8	1.05	1.09	0	3.17
2% - 4%	16	0.91	1.01	0	4.43	15% - 20%	16	2.64	3.42	0	9.21
4% - 6%	2	0.05	0.06	0	0.25	20% - 25%	2	0.20	0.26	0	1.34
4% - 6%	4	0.23	0.20	0.02	0.73	20% - 25%	4	1.24	0.80	0.21	3.08
4% - 6%	8	0.61	0.39	0	1.25	20% - 25%	8	1.83	1.53	0.23	5.21
4% - 6%	16	1.22	1.18	0	4.31	20% - 25%	16	5.22	4.21	0	11.3
6% - 8%	2	0.07	0.09	0	0.35	25% - 30%	2	0.25	0.24	0	1.00
6% - 8%	4	0.48	0.28	0.06	1.20	25% - 30%	4	1.24	0.99	0.06	3.63
6% - 8%	8	0.54	0.43	0	1.59	25% - 30%	8	1.75	1.27	0	3.91
6% - 8%	16	1.19	1.40	0	5.55	25% - 30%	16	9.18	7.64	0	20.2

Figure 6: Gain statistics per memory ratio and number of cores

N_c	Gain statistics (in percentage)			
	mean	σ	min	max
2	0.10	0.16	0	1.34
4	0.48	0.61	0	3.63
8	0.73	0.89	0	5.21
16	1.43	2.72	0	20.2

Figure 7: Gain statistics per number of cores (all values of r_τ considered)

6 A MORE PRECISE CONTENTION MODEL

The contention model used in the previous sections gives a pessimistic estimation of contention induced overheads. Any two tasks executing concurrently are considered interfering to the full extent of their memory demand. Even if the tasks τ_j and τ_k are executing concurrently only during a few cycles, the model considers they make respectively B_j and B_k accesses during these few cycles. As a consequence, the impact of contention may be overestimated and slack time may be introduced when unnecessary, leading to bias in the results presented in the previous section.

To estimate the bias, this section details a more precise (but more complex) contention model estimating the maximum overhead on each task based on a more precise duration of their concurrent execution. Figure 11 presents two interfering tasks τ_j and τ_k. Their execution time is divided in two parts : the white part corresponds to their *actual* execution and the shaded part corresponds to the overhead induced by their mutual interference. The length of the shaded part is $C * to_j^k$ where to_j^k is the number of accesses of τ_j delayed by accesses of τ_k (respectively, to_k^j is the number of accesses of τ_k delayed by τ_j). to_j^k is linked to the duration $f_{j,k} - s_{j,k}$ when both white parts are executed concurrently. During that period of time, both tasks make up to $\lceil \frac{f_{j,k}-s_{j,k}}{S} \rceil$ accesses. We define $d^o_{j,k} = \lceil \frac{f_{j,k}-s_{j,k}}{S} \rceil$ the maximum number of accesses on which τ_j and τ_k may interfere. All the new variables used in this section are summarized in Figure 10.

6.1 ILP model update

Considering the more precise contention model, we define a new ILP model for calculating optimal slack time introduction. It has the same objective function as the previous ILP model and uses constraints from equations 1 and 2. Using the $\mathscr{P}$ set defined in subsection 4.3, the following constraints define the variables we just presented. Please note that $d^o_{j,k}$ may have a negative value when tasks τ_j and τ_k are not executed concurrently.

$$\begin{cases} \forall \{j,k\} \in \mathscr{P},\ s_{j,k} = \mathtt{max}(s_j, s_k) \\ \forall \{j,k\} \in \mathscr{P},\ f_{j,k} = \mathtt{min}(f_j - C * to_j^k, f_k - C * to_k^j) \\ \forall \{j,k\} \in \mathscr{P},\ f_{j,k} - s_{j,k} + 0.999 * C \geq C * d^o_{j,k} \\ \forall \{j,k\} \in \mathscr{P},\ f_{j,k} - s_{j,k} \leq C * d^o_{j,k} \\ \forall \{j,k\} \in \mathscr{P},\ d_{j,k} = \mathtt{min}(B_j, B_k, \mathtt{max}(0, d^o_{j,k})) \\ \forall \{j,k\} \notin \mathscr{P},\ d_{j,k} = 0 \end{cases} \tag{8}$$

The linearization of functions `min` and `max` is not detailed here to keep constraints simple. The first two lines of the equation define the start date and the end date of the common execution interval of τ_j and τ_k. The definition of $f_{j,k}$ excludes the overheads the tasks cause each other because the interval $f_{j,k} - s_{j,k}$ is used in the next two lines of the equation to calculate $d^o_{j,k}$. By definition, $d^o_{j,k}$ is greater than $\frac{f_{j,k}-s_{j,k}}{C}$ and to ensure that $d^o_{j,k}$ is rounded up, we provide the upper bound $\frac{f_{j,k}-s_{j,k}}{C} + 0.999$. If τ_j and τ_k

$a \leq r_\tau \leq b$	Gain statistics (in percentage)				$a \leq r_\tau \leq b$	Gain statistics (in percentage)			
	mean	σ	min	max		mean	σ	min	max
0.1% - 1%	0.08	0.15	0	0.91	8% - 10%	0.55	1.02	0	6.14
1% - 2%	0.14	0.24	0	1.37	10% - 15%	0.62	1.06	0	8.04
2% - 4%	0.35	0.67	0	4.43	15% - 20%	0.83	1.57	0	9.21
4% - 6%	0.38	0.69	0	4.31	20% - 25%	1.25	2.32	0	11.3
6% - 8%	0.42	0.75	0	5.55	25% - 30%	1.27	3.20	0	20.2

Figure 8: Gain statistics per memory ratio (all values of N_c considered)

N_c	Gain statistics (in percentage)			
	mean	σ	min	max
2	0.34	0.37	0	1.65
4	1.91	1.20	0.11	4.38
8	3.46	2.09	0.02	6.47
16	4.44	3.95	0	14.9

Figure 9: Gain statistics of the control group ($0.1\% \leq r_\tau \leq 30\%$)

are not executed concurrently, $\frac{f_{j,k}-s_{j,k}}{C}$ is negative and so is $d^o_{j,k}$. As a consequence, a lower bound of 0 to the maximal number of accesses on which τ_j and τ_k mutually interfere is necessary. We define $d_{i,j} = \min(B_j, B_k, \max(d^o_{j,k}))$ that maximal number of interfering accesses. It is lower-bounded by 0 and upper-bounded by the maximum memory demand of each task. Finally, if τ_j and τ_k may never be executed concurrently ($\{j,k\} \notin \mathscr{P}$), $d_{i,j} = 0$.

Using the calculated $d_{j,k} \forall \{j,k\}$ a more precise estimation of the overhead of the tasks can be achieved. The constraints on o_j can be expressed by replacing B_t by $d_{j,t}$ in equation 5.

$$\forall j \in [0, N-1], \text{assuming } \tau_j \text{ is mapped on core } k_j$$

$$\begin{cases} \forall k \in [0, N_c-1] \backslash \{k_j\},\ o^k_j \geq C * \sum_{t=0}^{N-1} d_{j,t} - y^k_j * \mathscr{M}' \\ \forall k \in [0, N_c-1] \backslash \{k_j\},\ o^k_j \geq C * B_j * y^k_j \\ \forall k \in [0, N_c-1] \backslash \{k_j\},\ o^k_j \leq C * \sum_{t=0}^{N-1} d_{j,t} * B_t \\ \forall k \in [0, N_c-1] \backslash \{k_j\},\ o^k_j \leq C * B_j \\ o^{k_j}_j = 0 \\ o_j = \sum_{k=0}^{N_c-1} o^k_j \end{cases} \tag{9}$$

With $\mathscr{M}'$ a *big-M* constant, upper bound on the sum of all B_t.

The number to^k_j of accesses of τ_j delayed by τ_k is bounded by the maximal number of mutually interfering accesses $d_{j,k}$. Summing to^k_j for each task interfering with τ_j and multiplying the result by C gives the total overhead of τ_j (o_j). Please note that to^k_j and to^j_k are not necessarily equal. Indeed, on Figure 11, τ_0 and τ_k interfere with τ_j and $o_j = \min(d_{j,0} + d_{j,k}, B_j)$ while $o_k = \min(d_{j,k}, B_k) = d_{j,k}$. If $o_j = d_{j,0} + d_{j,k}$ then $to^0_j = d_{j,0}$ and $to^k_j = d_{j,k} = to^j_k$. If $o_j = B_j$ and $B_j < d_{j,0} + d_{j,k}$, then τ_0 interferes with $to^0_j = \min(d_{j,0}, B_j)$ accesses of τ_j and τ_k interferes with $to^k_j = B_j - to^0_j$ accesses which is inferior to $d_{j,k} = to^j_k$.

We define $\mathscr{S}(\tau) = \{j : \tau_j \text{ is mapped on the same core as } \tau \text{ and is ordered after it}\}$, and define the following constraints on to^k_j.

$$\forall \{j,k\} \in \mathscr{P}, \begin{cases} o_j = C * \sum_{k:\{j,k\} \in \mathscr{P}} to^k_j \\ to^k_j \leq d_{j,k}\ ,\ to^k_j \leq d_{j,k} \\ to^k_j \geq 0\ ,\ to^k_j \geq 0 \\ to^k_j \geq d_{j,k} - yd^k_j * \mathscr{M}'' \\ \forall t \in \mathscr{S}(\tau_k),\ to^t_j \leq (1 - yd^k_j) * \mathscr{M}'' \\ to^j_k \geq d_{j,k} - yd^j_k * \mathscr{M}'' \\ \forall t \in \mathscr{S}(\tau_j),\ to^t_k \leq (1 - yd^j_k) * \mathscr{M}'' \end{cases} \tag{10}$$

With $\mathscr{M}''$ a *big-M* constant, upper bound on all B_j.

The last four lines of Equation 10 generalize the situation previously discussed. For a given task τ_j, assigned to a different core than τ_j, each task will interfere with the maximum number of accesses of τ_j. The first ordered interfering task on this core will interfere with $\min(d_{j,k_0}), B_j$ accesses. If $to^{k_0}_j = d_{j,k_0}$ then the second ordered task will interfere with $to^{k_1}_j = \min(d_{j,k_1}, B_j - to^{k_0}_j)$ and so on. If a task interferes with a number of accesses to^t_j inferior to $d_{j,t}$ then all the following tasks will not interfere with $\tau_j : \forall k \in \mathscr{S}(\tau_t),\ t^k_j = 0$.

6.2 Estimating the bias

Using the same experimental protocol as in Section 5, we measure gain statistics using the more precise contention model. We use an adaptation of Rihani et al. algorithm [14] to this model to calculate the global WCET without optimization and the updated ILP model to calculate the global WCET with optimization.

Because the updated ILP is more complex, calculating optimal slack time introduction on the same task graphs as in Section 5 takes too long. We therefore use smaller task graphs of 10 to 20 tasks with IWCET varying between 1000 and 3000. The number of parallel branches varies between 1 and 10 with an arithmetic mean of 5. Instances with varying r_τ are created as in Section 5. Over the 4400 generated schedules, 3099 were successfully optimized within four hours of calculation (corresponding to a total of eight days of computation).

The obtained gain statistics are compared to gain statistics obtained using the pessimistic contention model in Figure 12. The last column on the figure presents an overestimation factor defined as $m = \frac{\text{pessimistic mean gain} - \text{precise mean gain}}{\text{precise mean gain}}$.

The gain statistics for both contention models are similar. The mean gain obtained with the pessimistic model is generally superior to the mean gain obtained with the precise model but stays close to that value, as shown in the last lines of the table figure 12 corresponding to the control group. When the memory ratio is low, the overestimation factor is under 10%. As the memory ratio

Variables	
$s_{j,k}$	Start date of the common execution interval of τ_j and τ_k
$f_{j,k}$	End date of the common execution interval of τ_j and τ_k (without considering mutual caused overhead)
to_j^k	Number of accesses of τ_j delayed by τ_k
$d_{j,k}^o$	Maximum number of mutually interfering accesses of τ_j and τ_k
$d_{j,k}$	Maximum number of mutually interfering accesses of τ_j and τ_k $\quad d_{j,k} = \min(B_j, B_k, \max(d_{j,k}^o, 0))$
Boolean variables	
yd_j^k	Indicates that $to_j^k < d_{j,k}$

Figure 10: Notations introduced in the updated ILP model

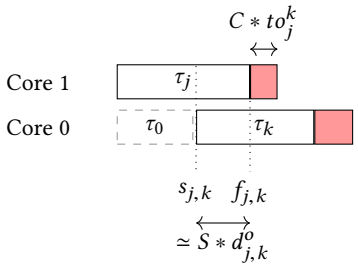

Figure 11: A more precise contention model

increases, m tends to increase while the number of cores does not seem to affect it. Higher bounds on r_τ can bring the overestimation factor up to 30% and more.

When analyzing the schedules produced by both contention models we observe that the global WCET calculated with both contentions models are either equal or close (they generally differ by less than 100 cycles). The difference between gains observed can be explained by the overestimation of contention of the pessimistic model while calculating the global WCET without slack time introduction. When the memory ratio is low, the calculated overheads induced by contention are close to the overheads obtained with the precise model. As a consequence, the overestimation factor remains low. When the memory ratio is high, the pessimistic model overestimates contention induced overheads compared to the precise model. It calculates a higher value for the global WCET without slack time introduction, while finding an optimal global WCET with slack time introduction, similar to the precise model. As a consequence, m has higher values.

The purpose of slack time introduction is to avoid contention by scheduling some tasks so that they do not interfere anymore. Both pessimistic and precise models achieve this goal. The precise model also allows the introduction of smaller slack time in order to reduce inferences: the tasks still interfere but on fewer accesses. In some cases, this allows the precise model to find a lower global WCET than the pessimistic model.

The pessimistic contention model tends to overestimate the overhead on interfering tasks, which has consequences especially in the case of high memory demand. Although it can overestimate by over 30% compared to the precise contention model, the gains observed in Figure 6 are still significant after taking the overestimation factor into account. For example, on Figure 6, in the case $25\% \leq r_\tau \leq 30\%, N_c = 8$ the mean gain is 1.75% or 1.29% when considering the overestimation factor of 35.8% observed for this situation. Comparison of the results given by both contention models show a bias on the results presented in Section 5 which presents overestimated gains. Such an observation does not invalidate the analysis using the pessimistic model and does not question the interest of slack time introduction. Moreover, the pessimistic model gives a good estimation of the optimal global WCET that can be achieved through slack time introduction.

7 RELATED WORK

This section presents work on scheduling techniques for multi-core platforms handling shared memory contentions.

Real-time scheduling techniques for multi-cores are surveyed in [4]. According to their taxonomy the class of schedules manipulated in this work are partitioned, time-triggered and non preemptive, and the schedules are generated off-line.

Multi-core platforms feature hardware resources (caches, buses, main memory) that may be concurrently accessed by tasks executing on the different cores. A contention analysis has to be defined to determine the delays to gain access to the shared resources (see [8] for a survey).

There are many approaches proposed recently to analyze contention delays to access shared resources. For architectures with caches, Dasari et al. [3] assume task mapping known and estimate contention delays for a variety of bus arbiters. Rihani et al. [14] assume both task mapping on cores and execution ordering known, and adds contention delays to tasks that execute in parallel in the schedule. Kim et Yun [12, 17] tightly bound interference delays on DRAM banks. Our interest in this work is not to have the tighter upper bounds on interferences due to shared resources, but rather to show if modifications of an existing schedule could be used to reduce the interference delay. In particular, the *request-driven* approach presented in [17] would refine the access time part of our delay to access the DRAM, considered constant in our approach (constant S).

Some scheduling techniques consider concurrent access to shared resources to take their scheduling decisions.

Xiakang et al. [7] proposed a method for managing contention online based on task profiling. During an offline phase, each task is executed in isolation and in concurrence with other tasks, in order to measure its *pressure* on shared resources. During the online phase, the scheduler uses these measures to enforce a fixed value for the maximum pressure accepted. Controlling which tasks may

Parameters		Gain statistics (in %)								m (%)
		Precise contention model				Pessimistic contention model				
$a \leq r_\tau \leq b$	N_c	mean	σ	min	max	mean	σ	min	max	
0.1% - 1%	2	0.01	0.02	0	0.11	0.01	0.02	0	0.11	0
0.1% - 1%	4	0.02	0.04	0	0.19	0.02	0.04	0	0.18	0.10
0.1% - 1%	8	0.02	0.05	0	0.23	0.02	0.05	0	0.23	-0.2
0.1% - 1%	16	0.03	0.08	0	0.61	0.03	0.08	0	0.61	0
1% - 2%	2	0.03	0.03	0	0.20	0.03	0.04	0	0.25	4.07
1% - 2%	4	0.09	0.11	0	0.46	0.10	0.12	0	0.47	8.31
1% - 2%	8	0.19	0.17	0	0.86	0.20	0.18	0	0.86	6.36
1% - 2%	16	0.31	0.27	0	1.03	0.33	0.28	0	1.03	5.10
2% - 4%	2	0.04	0.06	0	0.35	0.05	0.06	0	0.35	5.16
2% - 4%	4	0.17	0.19	0	1.02	0.17	0.20	0	1.02	4.90
2% - 4%	8	0.41	0.32	0	1.31	0.42	0.32	0	1.31	3.57
2% - 4%	16	0.57	0.46	0	1.82	0.58	0.47	0	2.10	2.93
4% - 6%	2	0.08	0.09	0	0.43	0.08	0.09	0	0.43	2.25
4% - 6%	4	0.25	0.26	0	1.11	0.28	0.27	0	1.16	8.28
4% - 6%	8	0.73	0.56	0	3.08	0.79	0.63	0	3.57	8.64
4% - 6%	16	0.90	0.77	0	3.45	0.98	0.80	0	3.45	8.37
6% - 8%	2	0.11	0.12	0	0.52	0.12	0.13	0	0.52	8.51
6% - 8%	4	0.41	0.48	0	2.12	0.47	0.52	0	2.12	15.1
6% - 8%	8	1.04	0.89	0	3.15	1.12	0.90	0	3.15	8.15
6% - 8%	16	1.27	1.21	0	4.70	1.35	1.20	0	4.70	6.10
8% - 10%	2	0.12	0.13	0	0.63	0.13	0.16	0	0.90	10.2
8% - 10%	4	0.50	0.58	0	2.23	0.60	0.62	0	2.34	18.5
8% - 10%	8	1.40	1.20	0	5.20	1.50	1.25	0	5.75	7.25
8% - 10%	16	1.64	1.53	0	6.14	1.76	1.59	0	6.36	7.28
10% - 15%	2	0.23	0.26	0	1.48	0.25	0.27	0	1.48	7.53
10% - 15%	4	0.76	0.77	0	3.74	0.79	0.80	0	3.74	4.53
10% - 15%	8	1.70	1.31	0	4.14	1.87	1.39	0	5.13	10.1
10% - 15%	16	1.94	1.69	0	7.33	2.18	1.85	0	7.33	12.5
15% - 20%	2	0.22	0.24	0	1.13	0.29	0.33	0	1.44	30.2
15% - 20%	4	1.02	1.08	0	3.98	1.09	1.16	0	3.98	6.87
15% - 20%	8	2.42	2.45	0	10.6	2.64	2.71	0	12.1	9.13
15% - 20%	16	2.53	2.58	0	10.6	2.84	2.92	0	12.1	12.1
20% - 25%	2	0.29	0.33	0	1.43	0.35	0.38	0	1.43	20.6
20% - 25%	4	0.89	1.03	0	3.35	0.97	1.17	0	4.72	9.81
20% - 25%	8	1.97	1.80	0	5.84	2.41	2.08	0	6.68	21.9
20% - 25%	16	2.00	1.98	0	5.85	2.54	2.35	0	8.06	27.0
25% - 30%	2	0.29	0.30	0	1.54	0.37	0.44	0	1.98	27.9
25% - 30%	4	0.83	0.97	0	3.47	1.45	1.68	0	6.77	75.0
25% - 30%	8	2.90	3.16	0	12.4	3.95	3.50	0	12.4	35.8
25% - 30%	16	2.76	3.23	0	12.4	3.66	3.43	0	12.4	32.7
Control group										
0.1% - 30%	2	0.49	0.55	0	3.39	0.52	0.59	0	3.39	0.06
0.1% - 30%	4	1.02	1.02	0	3.88	1.15	1.07	0	3.92	0.12
0.1% - 30%	8	1.99	1.84	0	7.21	2.09	1.91	0	7.54	0.04
0.1% - 30%	16	2.42	2.28	0	7.72	2.52	2.40	0	7.72	0.04

Figure 12: Comparison of gain statistics with the precise and the pessimistic memory model

be executed concurrently based on their shared resource demand is also the key idea of this paper although it follows a different approach. The proposed technique is a best-effort strategy, no real-time guarantee is provided to the executed tasks.

Rihani et al. [14] designed an algorithm for updating a contention-free static time-triggered schedule by calculating the overheads induced on the tasks by shared memory contention. The schedule is iteratively modified to identify tasks that are executed concurrently and calculate their WCET with contention. In contrast to their work, we do not transform a contention-free schedule to account for interference, but examine if modifications of an initial contention-aware schedule may reduce the cost of interference. We used their algorithm to calculate the WCET of non-optimized schedules.

Becker et al. [2] proposed an execution framework for avoiding contention by taking advantage of memory privatization features available in processors such as the Kalray MPPA-256. Tasks are divided into three sub-tasks: *read* which copies input data from a public memory bank to a private memory bank, *execute* which only accesses the private memory bank and *write* which copies output data to the shared memory bank. Using a specific scheduling policy it is possible to completely avoid contention. Giannopoulou et al. [9] proposed methods for mapping and scheduling task sets of mixed criticality on processors such as the Kalray MPPA-256 by limiting contention on two shared resources: shared memory and inter cluster communications. Alhammad et al. [1] proposed a heuristic for mapping and scheduling fork/join task graphs on many-core processors, minimizing the total execution time by avoiding contention. Compared to the aforementioned works, our intent is not to completely avoid contention in the produced scheduled, but rather to see if limiting contention on existing schedules is beneficial regarding schedule length.

Rouxel et al. propose in [15] contention-aware task mapping and scheduling techniques for multi-core platforms. Kim et al. propose in [13] a scheduling technique that allocates tasks to cores and partitions memory among tasks to reduce the memory interference delays. Compared to these works, that accounts for contention *during* schedule generation, we proceed in two steps. Our intent is to identify by how much an existing schedule can be shortened by avoiding some of the contention that exists in an existing schedule. Quantifying the quality of our proposed two step method as compared to global approaches is left for future work.

8 CONCLUSION AND FUTURE WORK

Calculating the WCET of a parallel program requires estimating the impact of shared resources contention on the WCET of its tasks. Using an empiric approach, this paper showed that introducing slack time on the schedule to limit contention can reduce the program WCET by a percentage depending on the memory demand of the tasks and on the number of cores used by the schedule. In the case of memory intensive tasks spread on 16 cores, we could improve the program WCET by up to 20%. While the adopted contention model is pessimistic, its overestimation was measured and its results still are a fair estimation of the gains that can be expected from slack time introduction, providing references for the development of future heuristics. However, finding an optimal schedule guaranteeing minimal WCET is complex, and in the case of complex task graphs with many independent branches, the ILP systems presented in this paper can take a full day of computation to calculate an optimal solution. For that reason, we intend on designing a heuristic approach for inserting slack time in a schedule, using the precise memory model.

REFERENCES

[1] A. Alhammad and R. Pellizzoni. 2014. Time-predictable execution of multithreaded applications on multicore systems. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1–6. https://doi.org/10.7873/DATE.2014.042

[2] Matthias Becker, Dakshina Dasari, Borislav Nicolic, Benny Åkesson, Vincent Nélis, and Thomas Nolte. 2016. Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform. In *28th Euromicro Conference on Real-Time Systems*.

[3] Dakshina Dasari, Vincent Nelis, and Benny Akesson. 2016. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems* 52, 3 (2016), 272–322.

[4] RI Davis and A Burns. 2011. A survey of hard real-time scheduling algorithms for multiprocessor systems. *in ACM Computing Surveys* (2011).

[5] Benoît Dupont de Dinechin, Duco van Amstel, Marc Poulhiès, and Guillaume Lager. 2014. Time-critical Computing on a Single-chip Massively Parallel Processor. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE '14)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, Article 97, 6 pages.

[6] Robert P. Dick, David L. Rhodes, and Wayne Wolf. 1998. TGFF: Task Graphs for Free. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign (CODES/CASHE '98)*. IEEE Computer Society, Washington, DC, USA, 97–101.

[7] Xiaokang Fan, Yulei Sui, and Jingling Xue. 2016. Contention-Aware Scheduling for Asymmetric Multicore Processors. *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)* 00, undefined (2016), 742–751. https://doi.org/doi.ieeecomputersociety.org/10.1109/ICPADS.2015.98

[8] Gabriel Fernandez, Jaume Abella Ferrer, Eduardo Qui nones Moreno, Christine Rochange, Tullio Vardanega, Francisco Javier Cazorla Almeida, et al. 2014. Contention in multicore hardware shared resources: Understanding of the state of the art. (2014).

[9] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, Lothar Thiele, and Benoît Dupont de Dinechin. 2016. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems* 52, 4 (2016), 399–449. https://doi.org/10.1007/s11241-015-9227-y

[10] Sebastian Hahn, Jan Reineke, and Reinhard Wilhelm. 2015. Towards Compositionality in Execution Time Analysis: Definition and Challenges. *SIGBED Rev.* 12, 1 (March 2015), 28–36.

[11] Damien Hardy, Benjamin Rouxel, and Isabelle Puaut. 2017. The Heptane Static Worst-Case Execution Time Estimation Tool. In *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017) (OpenAccess Series in Informatics (OASIcs))*, Jan Reineke (Ed.), Vol. 57. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 1–12. https://doi.org/10.4230/OASIcs.WCET.2017.8

[12] Hyoseung Kim, Dionisio De Niz, Bj"orn Andersson, Mark Klein, Onur Mutlu, and Ragunathan Rajkumar. 2014. Bounding memory interference delay in COTS-based multi-core systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*. IEEE, 145–154.

[13] Hyoseung Kim, Dionisio de Niz, Björn Andersson, Mark Klein, Onur Mutlu, and Ragunathan Rajkumar. 2016. Bounding and reducing memory interference in COTS-based multi-core systems. *Real-Time Systems* 52, 3 (01 May 2016), 356–395. https://doi.org/10.1007/s11241-016-9248-1

[14] Hamza Rihani, Matthieu Moy, Claire Maiza, Robert I. Davis, and Sebastian Altmeyer. 2016. Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS '16)*. ACM, New York, NY, USA, 67–76. https://doi.org/10.1145/2997465.2997472

[15] Benjamin Rouxel, Isabelle Puaut, and Steven Derrien. 2017. Tightening contention delays while scheduling parallel applications on multi-core architectures. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*. To appear.

[16] William Thies, Michal Karczmarek, and Saman Amarasinghe. 2002. *StreamIt: A Language for Streaming Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 179–196. https://doi.org/10.1007/3-540-45937-5_14

[17] Heechul Yun, Rodolfo Pellizzon, and Prathap Kumar Valsan. 2015. Parallelism-aware memory interference delay analysis for cots multicore systems. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*. IEEE, 184–195.