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:
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 depicts a motivating example. On the left part, tasks \( t_0 \), \( t_1 \) and \( t_2 \) 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 \( t_0 \), such that it now runs concurrently only with \( t_2 \). By avoiding contention with \( t_1 \), the overhead on \( t_0 \) is reduced and the overhead on \( t_1 \) 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 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 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 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...
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:

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

### 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 \( j \)), 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 WCET plus the overhead induced by shared memory contention. This gives the following constraint for each task:

\[
\forall j \in [0, N - 1] \quad f_j = s_j + \text{IWCET}_j + o_j
\]

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 \( j \) is either a dependency of \( k \) in the task graph, or precedes \( k \) 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.

\[
\forall (j, k) \in O : \quad s_k \geq f_j
\]

Tasks happening in parallel. Let us note \( P \) the set of all pairs \((j, k)\) with \( j \neq k \) of \([0, N - 1]^2\) such that \( j \) and \( k \) happen in parallel in the original schedule. Set \( P \) 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 \( P \). The definition of \( p^j_k \) and \( q^j_k \) gives the following constraints.

\[
\begin{align*}
\forall (j, k) \in P, \quad p^j_k - M p^j_k &\leq f_j - s_k < M q^j_k + p^j_k \\
\forall (j, k) \in P, \quad q^j_k - M q^j_k &\leq f_k - s_j < M q^j_k + p^j_k \\
\forall (j, k) \in P, \quad p^j_k + q^j_k & = 1
\end{align*}
\] (3)

With \( M \) a big-M constant, upper bound on all \( f_j \).

For a given pair of jobs \( \{j, k\} \), three situations can happen: (1) \( j \) finishes before \( k \) starts \((p^j_k = 1, p^j_k = 0)\), (2) \( k \) finishes before \( j \) starts \((p^j_k = 1, p^j_k = 0)\) and (3) neither finishes before the other starts, both tasks happen in parallel \((p^j_k = 0, p^j_k = 0)\). The latter is expressed with the following constraints:
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 \( B_t \) of each task is generated randomly, based on each task’s IWCET. We define \( r_t \), the memory ratio of task \( r \) as the percentage of its IWCET dedicated to accessing the shared memory, without contention and assuming \( r \) makes its maximum number of accesses (\( B_t \)).

\[
r_t = \frac{S \ast B_t}{IWCET_t} \quad \text{with } S \text{ the worst case cost (in cycles)}
\]

of an access to shared memory

Each of the 100 task graphs is instantiated 10 times with different bounds on \( r_t \) 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 \( r_t \) and wider for larger values of \( r_t \). For each task \( r \) and each interval, a value of \( r_t \) is randomly generated within the bounds an \( B_t \) 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 \( r_t \) randomly generated in interval [0.1 - 30].

The bounds on \( r_t \) were determined using the static WCET analysis tool Heptane [11] on benchmarks from the Mälardalen WCET benchmark suite1 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.

---

1http://www.mrtc.mdh.se/projects/wcet/benchmarks.html
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 \(r_c\) and the different numbers of cores (arithmetic mean, standard deviation \(\sigma\), 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 \(r_c\) 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 \(2\% \leq r_c \leq 4\%\) and \(N_c = 16\), the mean gain is 0.91% and \(\sigma = 1.01\%\) for a maximal gain of 4.43% whereas, when \(20\% \leq r_c \leq 35\%\) 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 \(\sigma\) 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 \(r_c\) and \(N_c\). A higher mean gain with a higher value of \(\sigma\) indicate that a wider range of gains was observed. As \(r_c\) 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 \(r_c\)). However, their values stay low compared to Figure 6. This can be explained by the impact of \(r_c\) (respectively \(N_c\)). For example, on figure 8, the statistics for \(15\% \leq r_c \leq 20\%\) 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 \(r_c\). As observed on Figures 6 and 7, the mean gain and \(\sigma\) increase with \(N_c\). Although their values do not reach values obtained on Figure 6, with homogeneous values of \(r_c\), 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.
Figure 6: Gain statistics per memory ratio and number of cores

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Gain statistics (in percentage)</th>
<th>Parameters</th>
<th>Gain statistics (in percentage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \leq r_x \leq b$</td>
<td>$N_c$</td>
<td>$a \leq r_x \leq b$</td>
<td>$N_c$</td>
</tr>
<tr>
<td>0.1% - 1%</td>
<td>2</td>
<td>mean</td>
<td>0.01</td>
</tr>
<tr>
<td>0.1% - 1%</td>
<td>4</td>
<td>$\sigma$</td>
<td>0.01</td>
</tr>
<tr>
<td>0.1% - 1%</td>
<td>8</td>
<td>min</td>
<td>0.12</td>
</tr>
<tr>
<td>0.1% - 1%</td>
<td>16</td>
<td>max</td>
<td>0.22</td>
</tr>
<tr>
<td>1% - 2%</td>
<td>2</td>
<td></td>
<td>0.22</td>
</tr>
<tr>
<td>1% - 2%</td>
<td>4</td>
<td></td>
<td>0.09</td>
</tr>
<tr>
<td>1% - 2%</td>
<td>8</td>
<td></td>
<td>0.16</td>
</tr>
<tr>
<td>1% - 2%</td>
<td>16</td>
<td></td>
<td>0.36</td>
</tr>
<tr>
<td>2% - 4%</td>
<td>2</td>
<td></td>
<td>0.04</td>
</tr>
<tr>
<td>2% - 4%</td>
<td>4</td>
<td></td>
<td>0.15</td>
</tr>
<tr>
<td>2% - 4%</td>
<td>8</td>
<td></td>
<td>0.45</td>
</tr>
<tr>
<td>2% - 4%</td>
<td>16</td>
<td></td>
<td>0.91</td>
</tr>
<tr>
<td>4% - 6%</td>
<td>2</td>
<td></td>
<td>0.05</td>
</tr>
<tr>
<td>4% - 6%</td>
<td>4</td>
<td></td>
<td>0.23</td>
</tr>
<tr>
<td>4% - 6%</td>
<td>8</td>
<td></td>
<td>0.61</td>
</tr>
<tr>
<td>4% - 6%</td>
<td>16</td>
<td></td>
<td>1.22</td>
</tr>
<tr>
<td>6% - 8%</td>
<td>2</td>
<td></td>
<td>0.07</td>
</tr>
<tr>
<td>6% - 8%</td>
<td>4</td>
<td></td>
<td>0.48</td>
</tr>
<tr>
<td>6% - 8%</td>
<td>8</td>
<td></td>
<td>0.54</td>
</tr>
<tr>
<td>6% - 8%</td>
<td>16</td>
<td></td>
<td>1.19</td>
</tr>
</tbody>
</table>

Figure 7: Gain statistics per number of cores (all values of $r_x$ 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 $\tau_j$ and $\tau_k$ 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 $\tau_j$ and $\tau_k$. 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 $C + t_{o_j}^k$ where $t_{o_j}^k$ is the number of accesses of $\tau_j$ delayed by accesses of $\tau_k$ (respectively, $t_{o_j}^k$ is the number of accesses of $\tau_k$ delayed by $\tau_j$). $t_{o_j}^k$ is linked to the duration $f^k_{j,k} - s_{j,k}$ when both white parts are executed concurrently. During that period of time, both tasks make up to $\lceil \frac{f^k_{j,k} - s_{j,k}}{s} \rceil$ accesses. We define $d_{j,k}^o = \lceil \frac{f^k_{j,k} - s_{j,k}}{s} \rceil$ the maximum number of accesses on which $\tau_j$ and $\tau_k$ 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 $\mathcal{P}$ set defined in subsection 4.3, the following constraints define the variables we just presented. Please note that $d_{j,k}^o$ may have a negative value when tasks $\tau_j$ and $\tau_k$ are not executed concurrently.

$$\begin{align*}
\forall \{j,k\} \in \mathcal{P}, \quad & s_{j,k} = \max(s_j, s_k) \\
\forall \{j,k\} \in \mathcal{P}, \quad & f_{j,k}^o = \min(f_j - C + t_{o_j}^k, f_k - C + t_{o_j}^k) \\
\forall \{j,k\} \in \mathcal{P}, \quad & f_{j,k}^o - s_{j,k} + 0.999 \cdot C \geq C \cdot d_{j,k}^o \\
\forall \{j,k\} \in \mathcal{P}, \quad & f_{j,k}^o - s_{j,k} \leq C \cdot d_{j,k}^o \\
\forall \{j,k\} \in \mathcal{P}, \quad & d_{j,k}^o = \min(B_j, B_k, \max(0, d_{j,k}^o)) \\
\forall \{j,k\} \notin \mathcal{P}, \quad & d_{j,k} = 0
\end{align*}$$

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 $\tau_j$ and $\tau_k$. The definition of $f_{j,k}^o$ excludes the overheads the tasks cause each other because the interval $f_{j,k}^o - s_{j,k}$ is used in the next two lines of the equation to calculate $d_{j,k}^o$. By definition, $d_{j,k}^o$ is greater than $\frac{f_{j,k}^o - s_{j,k}}{C}$ and to ensure that $d_{j,k}^o$ is rounded up, we provide the upper bound $\frac{f_{j,k}^o - s_{j,k}}{C} + 0.999$. If $\tau_j$ and $\tau_k$
Figure 8: Gain statistics per memory ratio (all values of $N_c$ considered)

![Table](image)

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

not executed concurrently, $\frac{d_{j,k} - d_{i,k}}{d_{i,k}}$ is negative and so is $d_{i,k}^p$.

As a consequence, a lower bound of 0 to the maximal number of accesses on which $t_j$ and $t_k$ mutually interfere is necessary. We define $d_{i,j} = \min(B_i, B_j, \max(d_{i,k}^p))$ 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 $t_j$ and $t_k$ may never be executed concurrently ($(j,k) \notin \mathcal{P}$), $d_{i,j} = 0$.

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

$\forall j \in [0, N - 1]$, assuming $t_j$ is mapped on core $k_j$

$$
\begin{align*}
\forall k \in [0, N_c - 1]\{k_j\}, & \quad o_j^{k_j} \geq C + \sum_{i=0}^{N} d_{i,k,t} - y_j^k \mathcal{M} \quad (9) \\
\forall k \in [0, N_c - 1]\{k_j\}, & \quad o_j^{k_j} \geq C + B_j + y_j^k \\
\forall k \in [0, N_c - 1]\{k_j\}, & \quad o_j^{k_j} \leq C + \sum_{i=0}^{N} d_{i,k,t} + B_t \\
\forall k \in [0, N_c - 1]\{k_j\}, & \quad o_j^{k_j} = 0 \\
& \quad o_j = \sum_{k=0}^{N_c-1} o_j^{k_j}
\end{align*}
$$

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

The number $t_{i,j}^k$ of accesses of $t_j$ delayed by $t_k$ is bounded by the maximal number of mutually interfering accesses $d_{i,k}$. Summing $t_{i,j}^k$ for each task interfering with $t_j$ and multiplying the result by $C$ gives the total overhead of $t_j$ ($o_j$). Please note that $t_{i,j}^k$ and $t_{i,j}^f$ are not necessarily equal. Indeed, on Figure 11, $t_0$ and $t_1$ interfere with $t_j$ and $o_j = \min(d_{j,0} + d_{j,k}, B_j)$ while $o_j = \min(d_{j,0}, B_k) = d_{j,k}$. If $o_j = d_{j,0} + d_{j,k}$ then $t_{i,j}^k = d_{j,k}$ and $t_{i,j}^f = d_{j,0}$. If $o_j = B_j$ and $B_j < d_{j,0} + d_{j,k}$ then $t_0$ interferes with $t_i = \min(d_{j,0}, B_j)$ accesses of $t_j$ and $t_k$ interferes with $t_{i,j}^f = B_j - t_{i,j}^f$ accesses which is inferior to $d_{j,k} = t_{i,j}^k$.

We define $\mathcal{S}(\tau) = \{j : t_j$ is mapped on the same core as $\tau$ and is ordered after it$\}$, and define the following constraints on $t_{i,j}^f$.

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 IWCT varying between 1000 and 3000. The number of parallel branches varies between 1 and 10 with an arithmetic mean of 5. Instances with varying $r_t$ 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 $m = \frac{\text{pessimistic mean gain}}{\text{precise mean gain}}$.

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
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 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 \textit{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 \textit{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
Quantifying WCET reduction of parallel applications by introducing slack time

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Precise contention model</th>
<th>Pessimistic contention model</th>
<th>( m ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a \leq r_T \leq b )</td>
<td>( N_c )</td>
<td>mean</td>
<td>( \sigma )</td>
</tr>
<tr>
<td>0.1% - 1%</td>
<td>2</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>0.1% - 1%</td>
<td>8</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>1% - 2%</td>
<td>2</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>1% - 2%</td>
<td>8</td>
<td>0.19</td>
<td>0.17</td>
</tr>
<tr>
<td>2% - 4%</td>
<td>2</td>
<td>0.04</td>
<td>0.06</td>
</tr>
<tr>
<td>2% - 4%</td>
<td>8</td>
<td>0.41</td>
<td>0.32</td>
</tr>
<tr>
<td>4% - 6%</td>
<td>2</td>
<td>0.08</td>
<td>0.09</td>
</tr>
<tr>
<td>4% - 6%</td>
<td>8</td>
<td>0.73</td>
<td>0.56</td>
</tr>
<tr>
<td>6% - 8%</td>
<td>2</td>
<td>0.11</td>
<td>0.12</td>
</tr>
<tr>
<td>6% - 8%</td>
<td>8</td>
<td>0.12</td>
<td>0.13</td>
</tr>
<tr>
<td>8% - 10%</td>
<td>2</td>
<td>0.50</td>
<td>0.58</td>
</tr>
<tr>
<td>8% - 10%</td>
<td>8</td>
<td>1.27</td>
<td>1.21</td>
</tr>
<tr>
<td>10% - 15%</td>
<td>2</td>
<td>0.23</td>
<td>0.26</td>
</tr>
<tr>
<td>10% - 15%</td>
<td>8</td>
<td>1.70</td>
<td>1.31</td>
</tr>
<tr>
<td>15% - 20%</td>
<td>2</td>
<td>0.22</td>
<td>0.24</td>
</tr>
<tr>
<td>15% - 20%</td>
<td>8</td>
<td>2.42</td>
<td>2.45</td>
</tr>
<tr>
<td>20% - 25%</td>
<td>2</td>
<td>0.29</td>
<td>0.33</td>
</tr>
<tr>
<td>20% - 25%</td>
<td>8</td>
<td>1.97</td>
<td>1.80</td>
</tr>
<tr>
<td>25% - 30%</td>
<td>2</td>
<td>0.29</td>
<td>0.30</td>
</tr>
<tr>
<td>25% - 30%</td>
<td>8</td>
<td>2.90</td>
<td>3.16</td>
</tr>
</tbody>
</table>

Control group

<table>
<thead>
<tr>
<th>Parameters</th>
<th>( N_c )</th>
<th>mean</th>
<th>( \sigma )</th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>( \sigma )</th>
<th>min</th>
<th>max</th>
<th>( m ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1% - 30%</td>
<td>2</td>
<td>0.49</td>
<td>0.55</td>
<td>0</td>
<td>3.39</td>
<td>0.52</td>
<td>0.59</td>
<td>0</td>
<td>3.39</td>
<td>0.06</td>
</tr>
<tr>
<td>0.1% - 30%</td>
<td>4</td>
<td>1.02</td>
<td>1.02</td>
<td>0</td>
<td>3.88</td>
<td>1.15</td>
<td>1.07</td>
<td>0</td>
<td>3.92</td>
<td>0.12</td>
</tr>
<tr>
<td>0.1% - 30%</td>
<td>8</td>
<td>1.99</td>
<td>1.84</td>
<td>0</td>
<td>7.21</td>
<td>2.09</td>
<td>1.91</td>
<td>0</td>
<td>7.54</td>
<td>0.04</td>
</tr>
<tr>
<td>0.1% - 30%</td>
<td>16</td>
<td>2.42</td>
<td>2.28</td>
<td>0</td>
<td>7.72</td>
<td>2.52</td>
<td>2.40</td>
<td>0</td>
<td>7.72</td>
<td>0.04</td>
</tr>
</tbody>
</table>

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 schedule, 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