THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Techniques to Tighten the Upper Bound on the Execution Time of Task-based Parallel Applications

PETROS VOUDOURIS

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2018

Techniques to Tighten the Upper Bound on the Execution Time of Task-based Parallel Applications

PETROS VOUDOURIS

Copyright ©2018 Petros Voudouris
except where otherwise stated.
All rights reserved.

Technical Report No 191L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2018.

Abstract

To use multiprocessors in hard real-time systems, schedulability analysis is needed to provide formally proven guarantees for the timing behavior of the system. Programming models for parallel applications, such as OpenMP, use pragmas to specify parts of the application as parallel tasks, for example, a function or a body of a loop. Worst-case-execution-time (WCET) analysis is used to find a safe upper bound of the execution time of a task (i.e., sequential code). However, determining a safe upper bound on the execution time of the entire parallel application on a multiprocessor platform, called the makespan, is a challenging problem.

Parallel applications can be modeled as directed acyclic graphs (DAG) (nodes are tasks and edges dependencies) where every node is characterized by its WCET. On a homogeneous platform, the simulation of a greedy, i.e., work-conserving schedule *cannot* be used to find a safe upper bound on the execution time due to timing anomalies. Timing anomalies is the main obstacle to calculate a safe upper bound of the makespan which is necessary to provide timing guarantees for parallel real-time applications. In the presence of timing anomalies, analytical approaches with pessimistic assumptions regarding the schedule of the tasks are used in the earlier works to calculate a safe upper bound on the makespan of parallel application on a homogeneous platform.

This thesis first provides a simulation based approach to calculate the makespan, with the use of time predictable and dynamic schedulers. A first contribution is a scheduler called `Strict Lazy` that fulfills the basic requirements to provide a timing anomaly-free schedule. As a result, a safe estimation of the makespan for homogeneous multiprocessors is calculated. Furthermore, the thesis builds upon `Strict Lazy` to develop another scheduler, called the `Lazy` scheduler, that has proven to be timing-anomaly free. As a result, the simulation of the schedule of a DAG with `Lazy` where all the nodes are executed for their WCET calculates a safe upper bound of the makespan. The proposed approach provides tighter and more scalable (to the number of processors) makespan estimations compared to the state-of-the-art.

A heterogeneous multiprocessor model is more general than a homogeneous multiprocessor model as it can cover a broad range of multiprocessor platforms including platforms with single instruction set architecture (ISA) but different microarchitectures, coexistence of processors with different ISAs and architectures with special-purpose accelerators. To the best of our knowledge, no earlier work considers the problem of how to calculate the makespan for schedules of parallel applications on unrelated multiprocessor platforms. This thesis finally proposes a polynomial time complexity, closed-form expression to calculate a safe upper bound on the makespan of DAGs for the unrelated multiprocessor model. The proposed method is applicable to a wide range of (greedy) schedulers and is also reducible to the state-of-art results for homogeneous and related multiprocessor models.

Keywords

Hard, Real-Time Systems, Parallel applications, Homogeneous and heterogeneous multiprocessors, WCET, DAG, Dynamic Scheduling, Makespan

Acknowledgment

First of all, I would like to thank my supervisor Per Stenström for giving me the opportunity to pursue a PhD degree under his supervision. His knowledge, guidance and support have been invaluable to my development as student. I really enjoyed the stories that he was telling that helped me to continue this work.

Special thanks to my co-supervisor Risat Pathan for the countless hours that we have spend to find and solve problems. Without him this thesis would not be possible.

Also, I would like to thank to Vassilis Papaefstathiou, Miquel Pericas, Madhavan Mannivanan, Yiannis Sourdis and Jan Jonsson for their valuable feedback and guidance.

I would like to thank my colleagues and friends here in Gothenburg Chloe, Babis, Aras, Ivan, Stavros, Vaggelis, Albin, Lea, Prajith, Alirad, Ashen, Stefano, Waqar, Mehrzad, Alexandra, Nadja, Angelos, Yiannis, Iosif, Maria, Vassiliki, Hannah, Christos, Dimitris, Georgia and Bilio.

This research has been funded by the MECCA project under the ERC grant ERC-2013-AdG 340328-MECCA

List of Publications

Appended publications

This thesis is based on the following publications:

I Petros Voudouris, Per Stenström, Risat Pathan "Timing-Anomaly Free Dynamic Scheduling of Task-Based Parallel Applications"
IEEE Real-Time Systems Symposium (RTSS), Work in progress, 2016.

II Petros Voudouris, Per Stenström, Risat Pathan "Timing-Anomaly Free Dynamic Scheduling of Task-Based Parallel Applications"
IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017.

III Petros Voudouris, Per Stenström, Risat Pathan "Bounding the Execution Time of Task-based Parallel Applications on Unrelated Multiprocessors"
Technical report, https://research.chalmers.se/en/publication/505944, 2018.

Other publications

The following publications were published during my PhD studies, but they are not part of this thesis.

IV Risat Pathan, Petros Voudouris, Per Stenström "Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms"
IEEE Transactions on Parallel and Distributed Systems (TPDS), 2018.

V Petros Voudouris, Per Stenström, Risat Pathan "A Safe and Tight Estimation of the Worst-Case Execution Time of Dynamically Scheduled Parallel Applications"
MULTIPROG workshop, High-Performance and Embedded Architectures and Compilers (HiPEAC), 2016.

Contents

Abstract iii

Acknowledgement v

List of Publications vii

1 Introduction 1
- 1.1 Background . . . 2
- 1.2 Timing-anomaly free execution . . . 3
  - 1.2.1 Priority assignment . . . 4
  - 1.2.2 Scheduling Policy: `Strict Lazy` . . . 4
  - 1.2.3 Scheduling Policy: `Lazy` . . . 4
  - 1.2.4 Summary of experimental results . . . 5
- 1.3 Makespan for unrelated multiprocessors . . . 6
  - 1.3.1 Platform characterization . . . 6
  - 1.3.2 Efficient makespan calculation . . . 6
  - 1.3.3 Summary of experimental results . . . 7
- 1.4 Conclusions and future work . . . 8

2 Paper I 11

3 Paper II 15

4 Paper III 27

Chapter 1

Introduction

In real-time systems, the correctness of the system does not depend only on the functional correctness of the result but also on when the result is produced [1]. There is an increasing demand for computation power from real-time applications. Multiprocessors can offer high and predictable performance, through parallelism, for real-time applications. To use them in real-time systems requires multiprocessor schedulability analysis [2] to provably guarantee the timing behavior of parallel applications.

Parallel applications can be modeled as a Directed Acyclic Graph (*DAG*), where every node is a task (sequential code) characterized by its worst-case execution time (WCET) [3] and the edges are dependencies between the tasks. To provide guarantees when they are executed on a multiprocessor platform, the primary challenge is to provide tight bounds of the worst-case schedule length, also called *makespan*.

Scheduling policies can be separated into two categories: static and dynamic. In static scheduling tasks are pre-assigned to fixed cores offline. With static scheduling, the multiprocessor platform will be underutilized due to load imbalance or communication overheads. Dynamic scheduling, on the other hand, can significantly improve resource utilization by assigning the tasks to the cores online, at run-time. An important class of dynamic schedulers is *greedy*, i.e., work-conserving, it schedules an available task whenever there is an idle processor [4]. However, any greedy scheduler may suffer from timing anomalies. Specifically, the execution time of a dynamically scheduled parallel application may increase when some tasks take less than their WCETs at run-time. This is known as an *execution-time-based timing anomaly* [5–7], which is a main obstacle to minimize the pessimism for the calculation of makespan when we consider any greedy scheduler. The first problem that this thesis addresses is how to dynamically schedule the tasks of a parallel application so we can avoid execution-time-based timing anomalies.

Modern multiprocessor platforms through parallelism [8–16] and acceleration [17–22] can provide performance and energy efficiency gains for real-time applications. Based on the speed relation that tasks have with the processors, the platform models can be separated into three categories: *homogeneous*, *related* and *unrelated* [2]. In homogeneous multiprocessors, there is a single processor type. Hence, the WCET for a specific task is the same on all processors. In related multiprocessors, each processor type is associated with a speed factor. The WCET of *any* task is scaled with the speed factor of the processor type (related multiprocessors are also known as *uniform* multiprocessor platforms [23]). In unrelated multiprocessors, a speed

factor is associated with each task-type and processor-type pair. Hence, the unrelated multiprocessor model is one of the most general model for a heterogeneous multiprocessor platform. Note that, since there are execution-time-based timing anomalies for the homogeneous multiprocessor model which is a special case of related and unrelated multiprocessor models, timing anomalies also exist in the context of related and unrelated heterogeneous multiprocessors.

The unrelated multiprocessor model can cover a broad range of heterogeneous platforms including platforms with single instruction set architecture (ISA) but different microarchitectures [19], coexistence of processors with different ISAs [17], architectures with special purpose accelerators for example, convolution [20], inference [21] and matrix multiplication [22] for machine learning. The second problem that this thesis addresses is how to calculate the makespan for parallel applications modeled as DAGs executed on unrelated multiprocessors. To the best of our knowledge no related work considers this problem.

The contributions of this thesis are the following:

- **Paper I** introduces a fixed priority, non-preemptive, non-greedy, dynamic scheduler called `Strict Lazy` that fulfills the basic requirements to provide an execution-time-based timing anomaly-free schedule and, as a result, a safe estimation of the makespan for homogeneous multiprocessors.

- **Paper II** presents a fixed priority, non-preemptive, non-greedy, dynamic scheduler called `Lazy` that, for the first time, has proven to be execution-time-based timing anomaly-free. As a result, the simulation of the schedule with `Lazy` of a DAG, where all the nodes are executed for their WCET calculates the makespan.

- **Paper III** proposes a polynomial time-complexity method to calculate the makespan of task-based parallel applications modeled as a DAG using the unrelated multiprocessor model.

The rest of the introduction of this thesis is organized as follows: Section 1.1 provides the background for the analytical approach for the makespan calculation and presents the timing anomalies. Section 1.2 presents the contributions of **Paper I** and **Paper II**. Section 1.3 presents the contributions of **Paper III**. Section 1.4 concludes the introduction of the thesis and discusses future work.

1.1 Background

Initially, we provide an example of timing anomalies that we have defined as the limiting factor for the makespan calculation in **Paper I** and **Paper II**. Then we present related work on analytical approaches to calculate the makespan on a homogeneous platform with any greedy scheduler. This analysis is used as baseline for the evaluation of **Paper I** and **Paper II** and is the specialized version of the makespan calculation for related and unrelated heterogeneous multiprocessors in **Paper III**.

An example of an execution-time-based timing anomaly is illustrated in Figure 1.1. The value alongside each node is the WCET of the corresponding task. The DAG is executed based on a non-preemptive Breadth First Schedule (BFS) on two processors P_0 and P_1. Consider, the DAG and the schedule on the left-hand side of Figure 1.1. The execution time of the application is 9 units. Now consider the case when node B does not execute for 3 units but finishes after 1 unit and all other nodes execute

according to their WCET. The DAG and the schedule for this scenario are shown on the right-hand side of Figure 1.1. The execution time of the application is 10 units. Hence, the overall execution time of the application is increased (from 9 to 10 units) when node B takes fewer time units than its WCET.

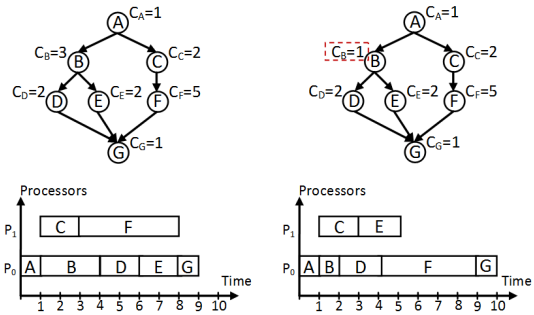

Figure 1.1: Example of execution time-based timing anomaly.

We can characterize a DAG by two parameters T_1 and T_∞ where, T_1 is the sum of the WCET of all the tasks and T_∞ is the sum of the WCET of the tasks that belong to the longest path of the DAG. Due to the problem of timing anomalies, the makespan calculation given by equation Eq. (1.1) of a parallel application modeled as a DAG executed on M processors [4, 24, 25], requires to make pessimistic assumptions about the schedule of the task on the multiprocessor platform.

$$T_M = T_\infty + \frac{(T_1 - T_\infty)}{M} \tag{1.1}$$

The second term of Eq. (1.1) shows the sum of the WCET of the nodes that do not belong to the longest path divided by the number of processors. The addition of the T_∞ implies that no task that belongs to the longest path can be executed in parallel with some task that **does not** belong to the longest path. Since during the actual execution, this is a highly unlikely scenario, this method introduces unnecessary pessimism which is an opportunity for improvement of the makespan calculation.

1.2 Timing-anomaly free execution

The state-of-the-art analytical approach given by Eq. (1.1) abstracts the details of the dynamic greedy scheduler and pessimistically assumes that no task can run in parallel with the tasks that belong to the longest path of the DAG. The main advantage of using a dynamic scheduler for the tasks is to be able to utilize the platform efficiently. However, an overestimated makespan would under-utilize the platform since it would need to reserve the computing power of the processors of the platform for some unnecessary extra time. The main challenge to reduce the pessimism of the makespan calculation for dynamically scheduled tasks is to prevent execution-time-based anomalies.

This section presents the proposed schedulers of **Paper I** and **Paper II**. Two fixed priority, non-preemptive, non-greedy dynamic schedulers are introduced that are execution-time-based timing-anomaly free. As a result, the simulation of their schedule where all the nodes are executed for their WCET calculates a safe upper bound for the makespan, even if some task execute for less than their WCET during

run-time. The dispatch condition for the two schedulers are introduced in Section 1.2.2 and 1.2.3 and finally brief results are presented in Section 1.2.4.

1.2.1 Priority assignment

Many priority assignments are used in the literature to achieve high utilization of the underlying platform. In addition to that, the goal of the proposed priority assignment is to provide unique priorities to the tasks. If all tasks have unique priorities a total order of the tasks can be enforced to achieve a time predictable execution.

Each node in the DAG is assigned a fixed priority. The fixed priority of a child node is assigned based on the fixed priority of its parent. Parallel tasks generated by the same parent usually need to synchronize their results (e.g., using the `taskwait` pragma in OpenMP). Such synchronization nodes are generally the sequential bottleneck in exploiting parallelism at the higher level of a DAG. Special priorities are assigned to these nodes to ensure that are executed with higher priority to exploit parallelism.

Parallel applications implemented in OpenMP [26] and Cilk [4] dynamically (during run-time) generate the parallel work (nodes of the DAG) with the use of recursions and loops. The proposed priority assignment has constant time complexity, so it can be used as a run-time mechanism that monitors the current level of a node and assigns priorities to dynamically generated nodes of a DAG, without prior knowledge about the structure (topology) of the DAG. It requires as input the maximum degree of the DAG and the current level of a currently executing node that would spawn new nodes of the DAG. **Paper I** and **Paper II** use different priority assignments which provide different makespan estimations. If we know statically the topology of the graph we can apply a topological sort [27] that provides unique priorities and guarantees that the priority of a parent node is higher than its children is suitable with the proposed scheduler and can offer a safe makespan.

1.2.2 Scheduling Policy: `Strict Lazy`

In **Paper I** a scheduler is introduced that is based on a constant-time check of the priority of the highest priority ready task. The dispatch condition checks if all the highest priority tasks have already been dispatched. In other words, the tasks are executed in *strict decreasing-priority order*. This dispatch condition provides an execution-time-based anomaly-free execution and can provide a safe estimation of the makespan. The main idea behind the anomaly freedom proof is the fact that the dispatch order of the tasks used for estimating the makespan is maintained during actual execution. Hence, it can be guaranteed that even if some task's actual execution time is smaller than its WCET, no timing anomaly can occur. However, the strict ordering of the dispatching condition limits the performance since some processors may remain idle while there are nodes awaiting execution in the ready queue.

1.2.3 Scheduling Policy: `Lazy`

In **Paper II** the dispatch condition presented in **Paper I** is extended to allow a higher number of tasks to be dispatched safely. The scheduler, called `Lazy`, is based on a constant-time check of the priority of the current highest priority ready task. The dispatch condition checks if there are *available processors for all the higher priority tasks* that may come in the future. If it is true, the task is dispatched to a processor.

Otherwise, the task is not dispatched for execution even if some processor is idle (i.e., `Lazy` is non-greedy).

The `Lazy` scheduler is able to dispatch a lower-priority task if there are enough processors to execute the higher priority tasks that are ready for execution or may become ready in the future. Consequently, during run-time, it is guaranteed that a lower priority task cannot start executing later compared to the starting time that is used offline for the estimation of the makespan.

To prove the anomaly freedom, we compare two schedules of the DAG on the same platform. We compare the schedule S_{WCET} where all tasks execute for their WCET and schedule S where some nodes execute less than their WCET. The starting time of a task in S can be at most as in S_{WCET} since we have reserved processors for all higher priority tasks. We have assumed that the scheduler is non-preemptive and consequently the same holds for the completion time. So the makespan of S cannot be longer than S_{WCET}.

1.2.4 Summary of experimental results

The `Lazy` scheduler presented in **Paper II** dominates (always smaller or same makespan) the `Strict Lazy` scheduler from **Paper I** and always will perform better. The dispatch condition of the `Strict Lazy` scheduler will allow a subset of tasks that the `Lazy` scheduler will allow being dispatched. To preserve the strict decreasing priority order of the task it will idle the processors frequently which leads to under-utilization of the platform and as a result to pessimistic makespan. However, the `Strict Lazy` scheduler will never be worse than the sequential execution since there is always at least one task that is executing.

To assess the effectiveness of the `Lazy` scheduler in determining makespan of parallel applications, we study its performance in the dynamic scheduling of Fibonacci, Sort, Strassen and FFT task-based parallel OpenMP applications from the BOTS benchmark suite [28]. As a baseline, we establish a safe upper bound of the makespan using the analytical approach of Eq. (1.1) which is pessimistic but safely estimates the makespan of parallel applications for any greedy dynamic scheduling algorithm. We use *tightness* and *scalability* (based on Gustafson's Law [29]) as key metrics to compare the effectiveness of `Lazy` in determining the makespan with the baseline.

For all the cases, the estimation of makespan of the `Lazy` scheduler is tighter compared to state of the art for each application. The worst-case assumption in deriving Eq. (1.1) is that the nodes that are not on the longest path do not run in parallel (i.e., always interfere) with the nodes of the longest path. However, the structure of a DAG may allow the nodes in the longest path to execute in parallel with nodes that are not part of the longest path. For different applications and different number of processors, the simulation of the `Lazy` scheduler achieves on average 9% and a maximum of 36% tighter estimation of the makespan in comparison to the state-of-the-art in Eq. (1.1). Furthermore, for all the applications, the `Lazy` scheduler scales better or similar to the state-of-the-art. For the different applications and configurations the increase in scalability, of the `Lazy` scheduler in comparison to the state-of-the-art is on average 14% and maximally 30%.

1.3 Makespan for unrelated multiprocessors

To the best of our knowledge, no related work provides a makespan calculation for parallel applications modeled as DAGs when they are executed on an unrelated multiprocessor platform under any greedy scheduler. This section presents the contributions of **Paper III** where a polynomial time complexity method to calculate the makespan of task-based parallel applications modeled as a DAG using the unrelated multiprocessor model is introduced. Section 1.3.1 presents the method to characterize the unrelated multiprocessor platform. Section 1.3.2 presents the makespan calculation methodology. Finally, Section 1.3.3 presents the evaluations of the makespan calculation methodology regarding tightness and pessimism.

1.3.1 Platform characterization

Formally characterizing the platform by specifying its capacities is a prerequisite for the schedulability analysis presented in **Paper III**. We assume a greedy scheduler that dispatches a task to an idle processor on which the task would execute the fastest with respect to other (if any) idle processors. Tasks can migrate to a faster processor if it becomes available during the execution of the task. The assumptions about the scheduler cover a broad classes of well-known scheduling principles like fixed-priority and earliest deadline first (EDF) which are also greedy.

The parameters to characterize a the uniform (related) platform, **processor capacity** and **uniformity** already presented in [23,30] are adapted for the unrelated platform model. The term "**heterogeneity**" instead of "uniformity" is used for the unrelated platform model.

Since in the unrelated multiprocessors, a speed factor is associated with each task-type and processor-type pair, to characterize a platform we need to analyze to which processors the tasks are mapped for execution. We split the execution of the tasks in two cases. First, when all the processors are busy and second when at least one processors is idle. The parameter processor capacity, denoted by S_M (where M is the number of processors), shows the *minimum* capability of the platform to consume the workload of the application when all the processors are busy. The parameter heterogeneity, denoted by λ, shows the *maximum* wastage of processor capacity that we can have throughout the execution of the tasks when some processors are idle.

The processor capacity and heterogeneity are used to enhance the analysis of the homogeneous multiprocessors given by Eq. (1.1). With these parameters, the impact of the different speeds of the tasks for the different processor types is modeled for the worst-case.

1.3.2 Efficient makespan calculation

In the context of unrelated multiprocessors, every task in the DAG of the application has multiple WCETs; there is one WCET for each processor type. To characterize the applications we use the DAG^{min} which is isomorphic to the DAG of the application where every node has one WCET which is equal to the minimum WCET between the different processor types for this task. We can characterize the DAG with the use of DAG^{min} in a similar way that we have done for the homogeneous multiprocessors by two parameters C and L, where C is the sum of the WCET of all the tasks and L is the sum of the WCET of the tasks that belong to the longest path.

We use a combinatorial approach to analyze all the possible execution scenarios of the tasks on different processor types exhaustively and we propose two approaches to calculate the makespan. These two approaches have exponential time-complexity to the number of processors and the number of tasks. As a result, they are useful only to analyze small-scale platforms. However, we use this analysis as a stepping stone to develop a third efficient makespan calculation (denoted by, T_M^U) which has polynomial time complexity and can also be used for large-scale platforms.

The makespan calculation for homogeneous multiprocessors given by Eq. (1.1) is extended with S_M and λ that characterize a platform with M processors to estimate the makespan for unrelated multiprocessors given by Eq. (1.2).

$$T_M^U \leq \frac{C + \lambda \cdot L}{S_M} \tag{1.2}$$

Initially, from Eq. (1.2) it can be seen that the processor capacity S_M influences the workload of all the tasks (C) since all the tasks can be executed when all processors are busy. Next, we can see that the heterogeneity λ influences only the workload of the tasks that belong to the longest path (L). The assumed scheduler is greedy, so when there are idle processors it is guaranteed that tasks that belong to the longest path are executing; as a result, we have processor capacity wastage. Finally, the makespan calculation method can also be applied to the more specialized platform models: related and homogeneous multiprocessors. The proposed makespan calculation will be the same as the approaches that are provided from [25, 30] for the related and the homogeneous platform model, respectively. For example, the homogeneous model can be modeled by setting the $S_M = M$ and $\lambda = M - 1$. As a result, the makespan calculation given by Eq. (1.2) is the same as the Eq. (1.1), for an application where $C = T_1$ and $L = T_\infty$.

1.3.3 Summary of experimental results

To evaluate our proposed makespan calculations we use four OpenMP, task-based parallel applications from the BOTS benchmark suit [26] modeled as DAGs: Fibonacci, Sort, Strassen and FFT. Also, synthetic DAGs are used, to measure the sensitivity of the proposed approach concerning different simulation parameters.

We could not find any literature on makespan computation of DAGs on unrelated machines. Instead, we measure the tightness of T_M^U by comparing the makespan with our two proposed exhaustive approaches. To find the level of pessimism, a lower bound of the makespan is derived by simulating the actual execution of parallel applications under the assumed scheduler and compare it with the makespan T_M^U .

By comparing the results of the exhaustive approach and the T_M^U approach for up to 8 processors with a fixed number of processors types, the T_M^U approach overestimates the makespan of the four applications on average only by 1% and up to 3%. By comparing the T_M^U with the simulation of the execution for up to 1024 processors with up to 8 processor types, we have on average 23% and up to 59% pessimism. In other words, our estimated makespan is at most 59% longer than the exact makespan. Next, for a platform with 8 processors and *varying* the number of processors types, the tightness of the T_M^U , compared to the two-permutation based approach, is on average 1% and at most 1.3%. By comparing the T_M^U to the simulation of the execution, we have on average 12% and up to 24% pessimism.

1.4 Conclusions and future work

This thesis presents techniques to improve the makespan for task-based parallel applications for homogeneous and unrelated heterogeneous platforms. The goal is to minimize the pessimism of the makespan to such a level that parallel platforms would be beneficial for real-time applications.

Paper I and **Paper II** propose a simulation based approach to calculate the makespan of a parallel application modeled as a DAG. With the use of formally proven time predictable dynamic schedulers, namely `Strict Lazy` and `Lazy`, the makespan can be calculated by the simulation of the schedule by assuming that every node of the DAG is executed for its WCET. The proposed schedulers can be seen as part of an makespan analysis tool for parallel applications or as a run-time mechanism with constant time priority check on a already sorted ready queue.

In **Paper III**, unrelated heterogeneous multiprocessor platforms and parallel applications modeled as DAGs are considered. An analytically approach is used to propose a closed-form solution for the calculation of the makespan. The main advantages of the proposed approach are: (i) the applicability to a wide range of already used and future coming schedulers and platforms (ii) the reducibility to the state-of-art for homogeneous and related multiprocessor platforms models.

Regarding the future work, we have seen that the makespan calculation for the heterogeneous multiprocessor platform that is proposed in Section 1.3 and in **Paper III** is based on Eq. (1.1). As a result, it makes the same fundamental assumption that the tasks that are not in the longest path do not run in parallel (i.e., always interfere) with the tasks of the longest path. A simulation-based approach of dynamic scheduled parallel applications that efficiently utilizes the underlying platform and was followed in **Paper I** and **Paper II** can also be applied in the context of unrelated multiprocessor platforms. The main challenge will be the development of a new dynamic scheduler for the unrelated heterogeneous platform and the proof of anomaly freedom. With this method we expect to achieve tighter and more scalable makespan calculation compared to the makespan proposed in the previous section.

Furthermore, for the presented work, we have assumed a simplified model of the platform where the executed tasks do not share any hardware resources. However, in practice, the tasks are sharing many common resources like memory, interconnect and power budget. A possible research direction would be to extend the platform model of the homogeneous multiprocessor platform with shared hardware resources and then propose a makespan calculation. The main challenge will be the mapping of the shared resources to the different tasks. Different shared-resource mappings would lead to different WCET for a task. A preliminary observation is that the makespan on a homogeneous platform with shared resources can be calculated by the makespan calculation of unrelated multiprocessor platform presented in **Paper III** if we can determine all the possible mappings. The minimum WCET of task (which is needed to calculate L and C) will be determined by providing to the task all the resources. Different mappings will act as the different processor types.

Multiprocessor platforms can provide computation power for real-time applications that require schedulability analysis to provide formal timing guarantees. The presented techniques and the future work can provide time-predictable and high performance execution for real-time systems.

Bibliography

[1] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer*, 1988.

[2] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor scheduling for real-time systems*. Springer, 2015.

[3] Wilhelm Reinhard et al. The worst-case execution-time problem overview of methods and survey of tools. *ACM TECS*, 2008.

[4] Robert D Blumofe and Charles E Leiserson. Scheduling multithreaded computations by work stealing. *JACM*, 1999.

[5] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 1969.

[6] Thomas Lundqvist and Per Stenström. Timing anomalies in dynamically scheduled microprocessors. In *IEEE RTSS*, 1999.

[7] Jan Reineke et al. A definition and classification of timing anomalies. In *WCET*, 2006.

[8] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: a predictable sdram memory controller. In *CODES+ISSS*. ACM, 2007.

[9] Martin Schoeberl. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems*, 2009.

[10] Theo Ungerer et al. parmerasa–multi-core execution of parallelised hard real-time applications supporting analysability. In *Digital System Design (DSD), 2013 Euromicro Conference on*. IEEE, 2013.

[11] Dumitru Potop-Butucaru and Isabelle Puaut. Integrated worst-case execution time estimation of multicore applications. In *WCET Workshop*, 2013.

[12] Michael Zimmer, David Broman, Chris Shaver, and Edward A Lee. Flexpret: A processor platform for mixed-criticality systems. In *RTAS*. IEEE, 2014.

[13] Christine Rochange, Pascal Sainrat, and Sascha Uhrig. *Time-Predictable Architectures*. John Wiley & Sons, 2014.

[14] Martin Schoeberl et al. T-crest: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 2015.

[15] Hokeun Kim et al. A predictable and command-level priority-based dram controller for mixed-criticality systems. In *RTAS*. IEEE, 2015.

[16] KALRAY. Mppa 2, 256 bostan processor. In *Kalray Whitepaper*, 2016.

[17] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *IEEE ISPASS*, 2009.

[18] NVIDIA. Tegra x1 nvidia new mobile superchip. In *White paper "https://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf"*, 2015.

[19] ARM Peter Greenhalgh. Big.little processing with arm cortex-a15 and cortex-a7 improving energy efficiency in high-performance mobile platforms. In *White paper, "http://www.cl.cam.ac.uk/ rdm34/big.LITTLE.pdf"*, 2011.

[20] Wajahat Qadeer et al. Convolution engine: balancing efficiency & flexibility in specialized computing. In *ACM ISCA*, 2013.

[21] Han Song et al. Eie: efficient inference engine on compressed deep neural network. In *IEEE ISCA*, 2016.

[22] Jouppi Norman P et al. In-datacenter performance analysis of a tensor processing unit. *ACM ISCA*, 2017.

[23] Shelby Funk, Joel Goossens, and Sanjoy Baruah. On-line scheduling on uniform multiprocessors. In *IEEE RTSS*, 2001.

[24] Richard P Brent. The parallel evaluation of general arithmetic expressions. *JACM*, 1974.

[25] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C Buttazzo. Response-time analysis of conditional dag tasks in multiprocessor systems. In *ECRTS*, 2015.

[26] Eduard Ayguadé et al. The design of openmp tasks. *IEEE TPDS*, 2009.

[27] Risat Pathan, Petros Voudouris, and Per Stenström. Scheduling parallel real-time recurrent tasks on multicore platforms. *IEEE TPDS*, 2018.

[28] Alejandro Duran et al. Barcelona openmp tasks suite: A set of benchmarks targeting the exploitation of task parallelism in openmp. In *ICPP*, 2002.

[29] John L Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 1988.

[30] Xu Jiang, Nan Guan, Xiang Long, and Wang Yi. Semi-federated scheduling of parallel real-time tasks on multiprocessors. *IEEE RTSS*, 2017.