Bounding the execution time of parallel applications on unrelated multiprocessors

Bounding the Execution Time of Task-based Parallel Applications on Unrelated Multiprocessors

***Abstract*—Heterogeneous multiprocessors, that consist of processor types with different execution capabilities, are critical today, and in future, to offer high performance and high energy efficiency. In order to use them in hard real-time systems to support parallel processing, a tight estimation of the upper bound on the completion time (WCET) of parallel applications is needed.**

This paper presents, for the first time, a closed-form solution for the calculation of the WCET for task-based parallel applications modeled as directed acyclic-graphs (DAG) using the general unrelated multiprocessor model that is capable of modeling a wide range of heterogeneous multiprocessor platforms. The paper contributes with a polynomial time algorithm to calculate the WCET (i.e., makespan) for the unrelated model. In addition, it presents simulation results that are based on modeling a set of representative OpenMP task-based parallel applications from the BOTS benchmark suite.

***Index Terms*—Real-time Scheduling, Parallel Applications, Heterogeneous multiprocessors, Makespan**

I. Introduction

The end of Dennard scaling at the beginning of this millennium led to a shift to multi/manycore platforms, a.k.a. multiprocessors. Higher performance and energy-efficiency has turned the focus on heterogeneous multiprocessors [1]. Heterogeneous multiprocessors comprise cores with different performance/energy and functional characteristics. Using them in real-time systems, efficient and tight estimations of the worst-case execution time (also known as the makespan) of parallel applications are needed.

This paper focuses on parallel applications run on heterogeneous multiprocessors. As tasks in these applications have different resource requirements, scheduling algorithms taking heterogeneity into account can offer higher performance and energy efficiency. As schedulability analysis for homogeneous multiprocessors cannot be trivially applied to heterogeneous multiprocessors [2], new scheduling algorithms are being developed and analyzed for heterogeneous systems [3], [4].

The worst-case execution time (WCET) of a task on a heterogeneous multiprocessor depends on the execution behavior of the task – the *task type* (i.e., two tasks of different types perform two different functionality) – as well as the type of processor used – the *processor type* [5]. Sometimes, however, it may not even be possible to execute a certain task on a particular processor due to the specialization of the processor (called, an *incompatible* processor type for that task). Based on the relation between the task type and the processor type, multiprocessor architectures can be separated into three categories: *homogeneous*, *related* and *unrelated*. In homogeneous multiprocessors, there is a single processor type. Hence, the WCET for a specific task type 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 platform [5]). In unrelated multiprocessors, a speed factor is associated with each task-type and processor-type pair. Hence, unrelated multiprocessors model is the most general model for heterogeneous multiprocessor platform that we consider in this paper for execution of real-time parallel applications.

Prior work on task- and processor-type speed relations has mainly focused on related multiprocessors [5]–[7]. This model is not capable of modeling today's heterogeneous multiprocessor platforms as the following example shows. Let's consider two tasks, task $u_{\texttt{ILP}}$ with abundant instruction-level parallelism (ILP), where all the instructions are independent of each other and task $u_{\texttt{NO}}$ with no ILP, where all the instructions have data dependencies between each other. Next, let assume that the two tasks are executed on a big.LITTLE platform [8], where the LITTLE processor is in-order with 8 pipeline stages, and the big processor is out-of-order with 15 pipeline stages. Task $u_{\texttt{ILP}}$ would have shorter WCET on the big processor compared to the LITTLE processor because the big processor can exploit the ILP and can finish the execution faster. In contrast, $u_{\texttt{NO}}$ has no ILP that the big processor can exploit. Also, the extra cost of a deeper pipeline and the cost of the mechanisms that are needed to preserve a correct execution of instructions that are (miss speculatively) executed in parallel, the WCET of $u_{\texttt{NO}}$ on the "big" processor may be longer compared to the WCET on the LITTLE processor. The example above shows that the related multiprocessor model, with the same speed factor for all tasks' types, is too restrictive. In contrast, the unrelated multiprocessor model is capable of modeling ideal execution scenario of different task types on a broad range of heterogeneous multiprocessors, including big.LITTLE, as it associates a speed factor with each task-type and processor-type pair.

Prior work using the unrelated multiprocessor model targets independent tasks [4], [9]. However, so far, no prior work has targeted task-based real-time parallel applications for unrelated multiprocessor platform. This paper considers, for the first time unrelated multiprocessor platform composed of processors of different types and parallel applications that are modeled as directed acyclic graphs (DAG), where every node is a task and a directed edge between two nodes is a dependency. Every task is characterized by using different WCET for different processor types and the goal is to calculate the makespan of the application.

We use a combinatorial approach to analyze all possible mappings of the tasks on different processor types exhaustively under a greedy scheduling policy and we propose two methods to calculate the makespan. These two approaches have exponential time complexity to the number of processors and the number of tasks, so 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 method (denoted by, EM) which has polynomial-time complexity and can also be used for large-scale platforms.

To evaluate our proposed makespan calculations we use four OpenMP, task-based parallel applications from the BOTS benchmark suit [10] modeled as DAGs [11]: Fibonacci, Sort, Strassen and FFT. We have also extended our evaluation with synthetic DAGs 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 EM by comparing the makespan with our two proposed exhaustive approaches. In addition, a lower bound on the makespan is derived by simulating the actual execution parallel applications under the assumed scheduler and compare it with the makespan under our proposed analytical approach EM, to find the level of pessimism.

By comparing the results of the exhaustive approach and the EM approach for up to 8 processors with a fixed number of processors types, the EM approach overestimates the makespan of the four applications on average only by 1% and up to 3%. By comparing the EM to 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% larger than the exact makespan. Next, for a platform with 8 processors and for *varying* number of processors types, the tightness of the EM, compared to the two-permutation based approach, is on average 1% and at maximum 1.3%. By comparing the EM to the simulation of the execution we have on average 12% and up to 24% pessimism.

We have also analyzed the impact of processor heterogeneity (i.e., how much different are the WCETs of the tasks on different processor types) on the makespan of all four applications. For the applications under study, we could *not* see a big difference for the tested cases mainly because of the high number of tasks in the applications hides the adverse impact of high heterogeneity. Finally, we have empirically investigated the impact of making some tasks incompatible to some processors types and found that by increasing the number of incompatible tasks, the makespan of the application increases since less parallelism is available.

The main contributions of the paper are:

- To the best of our knowledge this is the first work that provides a closed-form solution to the problem of calculating the makespan for task-based parallel applications modeled as DAGs executed on unrelated multiprocessor platform with polynomial time complexity, called EM.
- Two exhaustive approaches that provide tighter makespan calculation compared to the EM with exponential time complexity that can be applied to smaller platforms.
- Simulation results that are based on modeling OpenMP task-based parallel applications from the BOTS benchmark suite [12] and simulations with synthetic DAGs.

The rest of the paper is organized as follows: Section II introduces the system model. Next, Section III provides the definitions for the platform characterization. Section IV presents the proposed makespan calculations. Furthermore, Section V describes the simulation framework and Section VI reports the simulation results. Section VII presents the related work before we conclude the paper in section VIII.

II. System Model

This section presents the system model considered in this paper. The model of unrelated multiprocessor platform and the model of parallel application are presented in Section II-A and Section II-B, respectively. The task-processor speed relation is formally introduced in Section II-C. Finally, Section II-D presents the run-time scheduler.

A. Platform

We consider an unrelated multiprocessor platform with total M processors and H different processor types. A processor type is denoted by t for $t = 1, 2 \dots H$. For example, $t = 3$ specifies the processor type 3.

B. Application

A parallel application is modeled as a directed acyclic graph (DAG) denoted $G = (V, E)$, where $V = \{u_1, \dots u_N\}$ is a set of N nodes (tasks) and $E \subseteq (V \times V)$ is a set of directed edges. Each task is executed sequentially. If $(u_p, u_q) \in E$, then u_q can start execution only after task u_p completes.

The WCET of task u_i on any processor of type t is denoted by e_i^t. A heterogeneous platform may have specialized accelerators that have limited functionality and not all the tasks are compatible for execution on such accelerators. If task u_i is not compatible with processor type t, then $e_i^t = \infty$. We assume that every task has at least *one* compatible processor type. Two functionally equivalent tasks belong to the same task type. If two tasks u_i and u_j are *functionally equivalent*, then $e_i^t = e_j^t$, $\forall t$, $1 \leq t \leq H$.

Let a path, denoted by γ, be a sequence of tasks that are connected with edges. A task with no incoming and no outgoing edge is called a *source* and *sink*, respectively. Without loss of generality we assume that there is exactly one source (denoted as v_{src}) and one sink (denoted as v_{sink}) of G. If there is more than one source/sink node, a dummy task with WCET zero as a new source/sink node is added.

Let a DAG G^{min} be an isomorphic DAG with G [13] where each node u_i has only one WCET, denoted by e_i^{min} such that $e_i^{min} = \min_{t=1}^{H}\{e_i^t\}$. In other words, G^{min} is a DAG that has the same structure as G but each node u_i has only one (minimum over all processor types) WCET. The critical path cp of G is defined as the same path in G^{min} which has the

longest execution among all the paths in G^{min}. Let L be the sum of the WCETs of the tasks that belong to the cp in G^{min}. The total workload of the DAG G is denoted by C which is the sum of the WCETs of all the tasks belonging to G^{min}.

Figure 1 shows an example DAG where the WCETs of the tasks on an unrelated heterogeneous platform with four processor types are presented in Table I. By calculating the total workload and the workload of the critical path from the corresponding G^{min} we get $C = 6$ and $L = 3$, respectively.

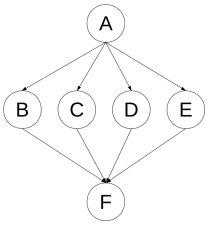

Fig. 1. Example of application model

	t_1	t_2	t_3	t_4
u_A	1	1	3	∞
u_B	1	2	4	5
u_C	2	1	∞	5
u_D	2	1	3	4
u_E	1	3	6	4
u_F	2	1	3	4

TABLE I
WCETs OF THE TASKS OF APPLICATION OF FIGURE 1

C. Task-processor speed relation

We define δ_t^i given by Eq. (1), as the (normalized) speed of processor type t with respect to task u_i.

$$\delta_t^i = \begin{cases} \frac{e_i^{min}}{e_i^t} & \text{if } e_i^t \neq \infty \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The larger is the value of δ_t^i, the smaller is the WCET of task u_i on processor type t. Note that $\delta_t^i = 1.0$ for processor type t on which the task has the smallest WCET.

Note that the tasks experience different speeds on different types of processor and two tasks of different types may not enjoy the same speed on the same processor type. Such a feature is not exhibited in uniform (i.e., related) multiprocessors [5]–[7]). Table II shows the speeds (computed using Eq. (1)) that each task in Table I experiences on four different types of processors.

	δ_1^i	δ_2^i	δ_3^i	δ_4^i
u_A	1.0	1.0	0.33	0
u_B	1.0	0.5	0.25	0.2
u_C	0.5	1.0	0	0.2
u_D	0.5	1.0	0.33	0.25
u_E	1.0	0.33	0.16	0.25
u_F	0.5	1.0	0.33	0.25

TABLE II
THE TASK-PROCESSOR SPEEDS FOR THE EXAMPLE DAG OF FIGURE 1

We denote Prf^i the sequence of non-increasing order of speeds on all the different processor types for task u_i. We also denote Prf_x^i the speed of the x^{th} fastest processor for task u_i. Finally, the set of processors with speed slower than than the x^{th} fastest processor for task u_i is denoted by Sl_x^i.

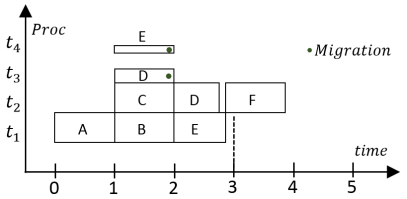

Fig. 2. An example of execution of the scheduler for the DAG of Figure 1

D. Scheduler

This subsection presents the details of our assumed scheduler which is *greedy*. The greedy scheduler dispatches a task u_i to an idle processor on which the task would execute the fastest with respect to other (if any) idle processors. Note that if no compatible processor for u_i is idle, then the task is not scheduled on an incompatible processor. In addition, if task u_i is currently executing on a processor of type t_1 and another (better) processor of type t_2 becomes available later on (for example, due to completion of some other task u_k) such that $e_i^{t_1} > e_i^{t_2}$, then task u_i is allowed to migrate from processor of type t_1 to type t_2 to execute faster. If multiple tasks are eligible for migration, then the selection of the task for migration is arbitrary.

This paper presents an analysis of the greedy scheduler to derive a closed-form equation that evaluates as a safe upper bound on the makespan of parallel application. The proposed analysis is directly applicable to broad classes of well-known scheduling principles like the fixed-priority and EDF which are greedy by nature.

Figure 2 depicts the execution of the parallel application in Figure 1 based on the assumed greedy scheduling algorithm on a heterogeneous platform (speeds specified in Table II) with four processors each of a different type. At time 0, only node u_A is ready for execution and it is dispatched to processor type t_1 since u_A has the minimum WCET on processor of type t_1. When u_A completes its execution after one time unit, nodes u_B, u_C, u_D and u_E are ready for execution. By assuming that the newly released tasks are placed in the ready queue in lexicographic order, node u_B is dispatched next. Since one processor of each type is available, node u_B is also dispatched to a processor of type t_1 which executes it the fastest. Similarly, u_C is dispatched to t_2 because it also executes the task fastest (i.e., provides the minimum WCET. Note that both u_B and u_C execute with speed 1 since they do not compete for the same (fastest) processor type. Next, u_D will be dispatched to processor type t_3 where it is executed with speed 0.33 because both (relatively faster) processors of types t_1 and t_2 are occupied. Task u_E is then dispatched to processor type t_4 and executes with speed 0.25. At time 2, both u_B and u_C complete their execution and processor types t_1 and t_2 become available. At time 2, node u_E and u_D also migrate respectively to relatively higher-speed processors of type t_1 and t_2 to continue the rest of their execution. Task u_D and u_E complete their execution at time 2.67 and 2.75, respectively. After that, u_F is dispatched to processor type t_2

which provides speed 1 for this task since all the processor-types are available. The entire application finishes at time 3.75.

It is clear that when a task u_i is executing on its x^{th} fastest processors, then it is also true that all the processors that are faster than the x^{th} fastest processor for u_i are also busy. In other words, when a task u_i is ready to be dispatched to a platform with $(x-1) \leq M$ busy processors, the scheduler may in the *worst-case* dispatches u_i to the processor type with speed Prf_x^i (x^{th} fastest speed for u_i).

III. Platform Characterization

Formally characterizing the platform by specifying its capacities is a prerequisite for the schedulability analysis presented in this paper. This section shows how the concept of processor capacity and the uniformity already presented in [5], [6] for uniform (related) multiprocessors are adapted for general unrelated multiprocessor platform. The term "heterogeneity" instead of "uniformity" is used in this paper for unrelated multiprocessors.

First, subsection III-A describes an approach to model all the possible mappings (each such mapping is called a permutation) of the tasks to the different types of processors (this is the basis for the permutation based approach to compute the makespan). Second, subsection III-B formally presents the notion of processor capacity and heterogeneity for permutation-based makespan calculation. In the first permutation-based approach, the processor capacity and the heterogeneity are computed for each *individual* permutation. Then individual (i.e., permutation-specific) makespan is computed considering individual permutation. Finally, the maximum over all the computed makespans of all the permutations is reported as the makespan of the entire application. In our second permutation-based approach, we compute a *common* processor capacity and heterogeneity that are applicable to any permutation. Then, the makespan is computed for any arbitrary permutation but using the common processor capacity and heterogeneity.

Subsection III-C presents a modified version of the notion of processor capacity and heterogeneity that do not even need to rely on any permutation and can be computed very efficiently (i.e., in polynomial time), which in turn is the basis for an efficient makespan calculation (our third approach).

A. Modeling of all possible mappings

In this paper, the first two permutation-based approaches determine the makespan of a parallel application by considering all the different possible mappings (i.e., execution scenarios) of the tasks on different processor types. The worst-case execution scenario for which the completion time of the application is maximized can be determined based on such mappings. To that end, we define permutations of the tasks where each such permutation corresponds to one particular execution scenario as follows.

We define π as one permutation of p tasks selected from N tasks of the application. For a given permutation π, we denote π_x as the x^{th} task in π. For example, consider a permutation $\pi =< u_2, u_1, u_5, u_9 >$ of size 4, where $\pi_3 = u_5$. The set of all the permutations[1] of size p selected from N different tasks is denoted by σ_p.

Consider an arbitrary schedule of an application with N tasks. Let B_p^π be the sum of the length of intervals for which exactly p processors are busy for executing the tasks of permutation π of size p. We define B_p to be the sum of the lengths of time intervals during which exactly p processors are busy in the schedule. Consequently,

$$B_p = \sum_{\pi \in \sigma_p} B_p^\pi \tag{2}$$

For the example schedule of Figure 2, we present in Figure 3 the active B_p^π. The horizontal axis is time and the vertical axis presents the processor capacity. Initially, it can be seen that B_1 is composed by three permutations with the tasks A, E and F since they are executed when 1 processor is busy. Next, we can see that B_2 is composed by one permutation with the tasks D and E since these are the only tasks that are executed when 2 processors are busy. Furthermore, we note that $B_3 = 0$ since there is no instance that exactly 3 processors are busy. Finally, the B_4 is composed by one permutation with the tasks that are executing when exactly 4 processors are busy, namely $\{B, C, D, E\}$.

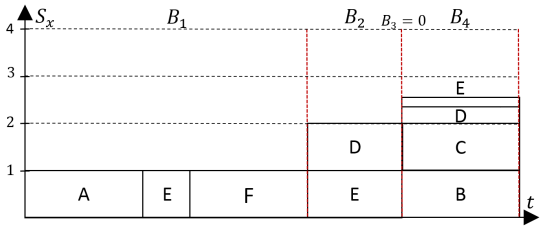

Fig. 3. Busy processor capacity based on schedule given by Figure 2.

When exactly p processors are busy due to executing the tasks that belong to π, a part of the total workload (C) is consumed. When some of these tasks also belong to the critical path, some part of the workload of the critical path (L) is also consumed. We define C_p^π as the amount of the total workload that is consumed when p processors are busy by tasks belonging to π. The amount of the total workload when p processors are busy is denoted by C_p such that $C_p = \sum_{\pi \in \sigma_p} C_p^\pi$. Therefore, the following holds:

$$C = \sum_{p=1}^{M} C_p = \sum_{p=1}^{M} \sum_{\pi \in \sigma_p} C_p^\pi \tag{3}$$

Similarly, let L_p^π be the amount of workload that belongs to the critical path cp when p processors are busy by the tasks that are present in π. The sum of the workload that belongs to cp when p processors are busy for all the permutations of size p is denoted by L_p such that $L_p = \sum_{\pi \in \sigma_p} L_p^\pi$. Since the scheduler is greedy, the workload of the critical path is also executing whenever no more than $(M-1)$ processors

[1]The total number of permutations of size p selected from N tasks is $\frac{N!}{(N-p)!}$.

are busy. Therefore, the workload on the critical path L is lower bounded as follows:

$$L \geq \sum_{p=1}^{M-1} \sum_{\pi \in \sigma_p} L_p^{\pi} \tag{4}$$

Note that if the schedule does not exhibit a certain permutation π of size p, then $L_p^{\pi} = 0$ and $C_p^{\pi} = 0$ for that permutation.

B. Parameters for the permutation based approaches

If exactly x processors are busy when task u_i in permutation π is being executed, then the execution of u_i is the longest if it is executed on its x^{th} fastest processor type where such a processor provides speed $\delta_x^{\pi_x}$ to task u_i. To calculate overall processor capacity and the heterogeneity of the platform, we need to consider — for the sake of worst-case analysis — the speed $\delta_x^{\pi_x}$ for executing u_i whenever exactly x processors are busy.

The overall processor capacity for a particular permutation π, denoted by S_x^{π}, shows the overall capability of the platform to execute the tasks in π and is defined as follows:

$$S_x^{\pi} = \sum_{k=1}^{x} \delta_k^{\pi_k} \tag{5}$$

where $\delta_k^{\pi_k}$ is the speed which the k^{th} task (i.e., task with index π_k) in permutation π uses for execution.

The makespan of a parallel application also depends on how much capacity is wasted in the worst-case, i.e., the capacity that cannot be used to execute the tasks of the parallel application. We will quantify such wastage using the notion of heterogeneity in Eq. (6). The makespan of an application is computed based on how much capacity of the platform is wasted. The higher is the wastage in capacity, the longer is the makespan. When at most x processors are busy for executing the nodes in a permutation π, the accumulated wastage is maximized if the node on the critical path executes on the x^{th} slowest processor. Based on these observations, the heterogeneity for a given permutation π is given by Eq. (6) that shows the maximum accumulated capacity loss of the platform. By assuming that the critical path is executing on a processor having speed $\delta_x^{\pi_x}$, the numerator in Eq. (6) is the sum of the speeds of the processors that are idle while the denominator is the speed of the processor executing the tasks of the critical path.

$$\lambda^{\pi} = \max_{x=1}^{M} \{ \frac{S_M^{\pi} - S_x^{\pi}}{\delta_x^{\pi_x}} \} \text{ where, } \delta_x^{\pi_x} \neq 0 \tag{6}$$

The minimum processor capacity (denoted by S_M^{min}) and maximum heterogeneity (denoted by λ^{max}) over all the permutations in set σ_M are given as follows:

$$S_M^{min} = \min_{\pi \in \sigma_M} \{S_M^{\pi}\} \tag{7}$$

$$\lambda^{max} = \max_{\pi \in \sigma_M} \{\lambda^{\pi}\} \tag{8}$$

C. Parameters for the efficient approach

The definition of busy interval B_p in Eq. (2) requires us to consider all the $O(N^M)$ permutations, which is exponential and is computationally impractical for large N and M. To find the makespan for larger system, we need a computationally efficient way to find the processor capacity and heterogeneity.

The assumed scheduler will dispatch a task u_i in the worst-case to its x^{th} fastest processor type (Prf_x^i) when exactly x processors are busy. In the worst-case, all the $(x-1)$ preferred processors of task u_i are busy for executing some other tasks. As a result, the idle processor that guarantees the shortest WCET for task u_i is Prf_x^i.

The capacity of the platform can also be defined independent of any permutation by assuming the minimum speed that each processor offers to any task. Recall that the greedy scheduler ensures that at least one processor executes some task with the most preferred speed, one processor executes some task at least with the second most preferred speed, one processor executes some task at least with the third most preferred speed, and so on. In other words, the minimum x^{th} speed that the platform provides to some task is $\min_{i=1}^{N}\{Prf_x^i\}$ and can be computed in polynomial time. To this end, the minimum processor capacity (S_M) is determined by summing the minimum speeds of the M processor as follows:

$$S_M = \sum_{x=1}^{M} \min_{i=1}^{N} \{Prf_x^i\} \tag{9}$$

Similarly, we also define the total capacity loss. The maximum speed of the y^{th} processor that is provided to any task is $\max_{j=1}^{N}\{\delta_y^j\}$. The maximum unused capacity $idle_x^i$ when task u_i is executing on its x^{th} faster processor is given by Eq. (10) as follows:

$$idle_x^i = \sum_{y \in Sl_x^i} \max_{j=1}^{N} \{\delta_y^j\} \tag{10}$$

where Sl_x^i is the set of processors having speed slower than the x^{th} fastest processors for task u_i. Note that Eq. (10) can also be computed in polynomial time.

When some processors are idle, we are going to have some processor capacity loss. Because the scheduler is greedy, we know that when some processors are idle, nodes of the critical path are executed. The duration during which we are going to have the capacity loss is determined by the execution time (i.e., cumulative length of intervals of execution of the nodes in the critical path) to consume the workload of the critical path. Since during the actual execution we do not know where the tasks that belong to the critical path are going to be executed we assume that in the worst case a task u_i belonging to the critical path will be executed on its x^{th} fastest processor given that exactly x processors are busy. Eq. (11) maximizes the accumulated capacity loss, denoted by λ, considering all the tasks and all the processors.

$$\lambda = \max_{i=1}^{N} \{ \max_{x=1}^{M} \{ \frac{idle_x^i}{\delta_x^i} \} \} \text{ where, } \delta_x^i \neq 0 \tag{11}$$

By comparing the heterogeneity of Eq. (6) and the heterogeneity of Eq. (11) we note that they both express the processor capacity loss. However, Eq. (11) introduces some new pessimism since it considers the maximum unused processor capacity between all the tasks (N), while Eq. (6) is calculated for a particular permutation. A table with the description of the parameters can be found in Appendix C, Table IV.

IV. Makespan calculation

This section presents three different approaches for calculating an upper bound on the makespan, denoted by T_M, of parallel application considering heterogeneous platform. First, Section IV-A presents the first combinatorial approach to calculate the makespan. Next, Section IV-B initially introduces the second combinatorial approach that is used eventually to prove the efficient makespan calculation.

A. Permutation based makespan

We need the following lemma which states that the value of λ^π in Eq. (6) for permutation π of size M is an upper bound on the value of λ^π for the same permutation π of size p, where $p \leq M$.

Lemma 1. *Given a permutation π of size M, the λ^π in Eq. (6) is larger than or equal to the same permutation of size p where $p \leq M$.*

Proof. From the definition of the permutation, for any permutation π^s of size $p \leq M$ there is a corresponding permutation π^b of size M that includes all the tasks with the first p tasks of π^b. By applying the definition of λ^π for the two permutations π^s and π^b, it can be seen that for π^b all the cases for the different processor speeds of π^s are covered in addition to the extra cases since π^b includes all the tasks in π^s.

□

Theorem 1. ***Permutation based makespan (PM1):*** *The makespan of application G characterized by C and L on an unrelated multiprocessor platform is given by:*

$$T_M \leq \max_{\pi \in \sigma_p}\left(\frac{\lambda^\pi}{S_M^\pi}\right) \cdot L + \max_{\pi \in \sigma_p}\left(\frac{1}{S_M^\pi}\right) \cdot C \tag{12}$$

Proof. Consider the busy interval B_p^π in an arbitrary schedule of G for the permutation B_p^π. During B_p^π, the p^{th} task belonging to π in the worst- case will be executed with speed $\delta_p^{\pi_p} \neq 0$. Let $\chi(\gamma, \delta_p^{\pi_p})$ denote the total amount of workload completed at speed $\delta_p^{\pi_p}$ along an arbitrary path γ where it holds that:

$$B_p^\pi \cdot \delta_p^{\pi_p} \leq \chi(\gamma, \delta_p^{\pi_p})$$

Let the total workload of the nodes that belong to path γ is W_γ^B. Since the longest path is *cp* with total workload L_p^π belonging to permutation π, the actual workload is bounded as follows: $\chi(\gamma, \delta_p^{\pi_p}) \leq W_\gamma^B \leq L_p^\pi$ and we have:

$$B_p^\pi \cdot \delta_p^{\pi_p} \leq L_p^\pi$$

From the definition of λ^π given in Eq. (6) we have:

$$\implies B_p^\pi \cdot \frac{S_M^\pi - S_p^\pi}{\lambda^\pi} \leq L_p^\pi \tag{13}$$

Equivalently,

$$B_p^\pi \cdot (S_M^\pi - S_p^\pi) \leq L_p^\pi \cdot \lambda^\pi \tag{14}$$

When p processors are busy with tasks from permutation π, it means that S_p^π processor capacity is used and the workload that is consumed is bounded by the total workload C_p^π. So it holds that $B_p^\pi \cdot S_p^\pi \leq C_p^\pi$ and by adding this term both sides of Eq. (14) we have:

$$B_p^\pi \cdot (S_M^\pi - S_p^\pi) + B_p^\pi \cdot S_p^\pi \leq L_p^\pi \cdot \lambda^\pi + C_p^\pi$$

$$B_p^\pi \leq \frac{L_p^\pi \cdot \lambda^\pi + C_p^\pi}{S_M^\pi}$$

By summing over all M processors for a given π:

$$\implies \sum_{p=1}^{M} B_p^\pi \leq \sum_{p=1}^{M}\left(\frac{L_p^\pi \cdot \lambda^\pi + C_p^\pi}{S_M^\pi}\right)$$

$$\implies \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} B_p^\pi \leq \sum_{p=1}^{M}\sum_{\pi \in \sigma_p}\left(\frac{L_p^\pi \cdot \lambda^\pi + C_p^\pi}{S_M^\pi}\right)$$

Since the scheduler is greedy, from Eq. (2) it holds that $T_M = \sum_{p=1}^{M} B_p = \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} B_p^\pi$ and we have:

$$\implies T_M \leq \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} \frac{\lambda^\pi}{S_M^\pi} \cdot L_p^\pi + \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} \frac{1}{S_M^\pi} \cdot C_p^\pi$$

From Lemma (1) and Eq. (5) (by setting $x = M$) we have:

$$\implies T_M \leq \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} \max_{\pi \in \sigma_M}\left\{\frac{\lambda^\pi}{S_M^\pi}\right\} \cdot L_M^\pi + \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} \max_{\pi \in \sigma_M}\left\{\frac{1}{S_M^\pi}\right\} \cdot C_p^\pi$$

Since λ^π and S_M^π does not depend on the size of π:

$$T_M \leq \max_{\pi \in \sigma_M}\left\{\frac{\lambda^\pi}{S_M^\pi}\right\} \cdot \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} L_M^\pi + \max_{\pi \in \sigma_M}\left\{\frac{1}{S_M^\pi}\right\} \cdot \sum_{p=1}^{M}\sum_{\pi \in \sigma_p} C_p^\pi$$

From the definitions of L and C (Eq. (4) and Eq. (3)) and because $L_M^\pi = 0$ (since we cannot guarantee that the critical path executes when all the processors are busy) in the worst-case $\forall \pi \in \sigma_p$ we have:

$$T_M \leq \max_{\pi \in \sigma_M}\left(\frac{\lambda^\pi}{S_M^\pi}\right) \cdot L + \max_{\pi \in \sigma_M}\left(\frac{1}{S_M^\pi}\right) \cdot C$$

□

B. Efficient makespan calculation

This section presents an efficient approach to calculate the makespan of a parallel application modeled as a DAG and executed on an unrelated multiprocessor platform. Lemma 2 first derives our second approach to compute the makespan based on all the permutations. Lemma 3 and 4 show that the heterogeneity and capacity given by Eq. (11) and Eq. (9) calculated for the efficient approach in subsection III-C are always more pessimistic than the heterogeneity and capacity given by Eq. (8) and Eq. (7) derived for the permutation-based approach. Finally, we present the EM makespan calculation in Theorem 2.

Lemma 2. ***Permutation based makespan (PM2):*** *The makespan of application G characterized by C and L on an unrelated multiprocessor platform is given by:*

$$T_M \leq \frac{C + \lambda^{max} \cdot L}{S_M^{min}} \tag{15}$$

where λ^{max} and S_M^{min} are given in Eq. (8) and Eq. (7), respectively.

Proof. By applying the Eq. (4) to Eq. (13) and since for an arbitrary π it holds that $\lambda^{\pi} \leq \lambda^{max}$ and form definition of λ^{π} given in Eq. (6) we have:

$$\sum_{p=1}^{M-1} \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot \frac{S_M^{\pi} - S_p^{\pi}}{\lambda^{max}} \leq L$$

Equivalently,

$$\sum_{p=1}^{M-1} \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot (S_M^{\pi} - S_p^{\pi}) \leq \lambda^{max} \cdot L \tag{16}$$

During B_p^{π} the p processors are busy with accumulated processor capacity of S_p^{π}, which means that after B_p^{π} time units, the amount of workload that is done is $(B_p^{\pi} \cdot S_p^{\pi})$. Since the application completes when no processor is busy, the total workload is given by $C = \sum_{p=1}^{M} \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot S_p^{\pi}$. By adding this term in both sides in Eq. (16), we get:

$$\sum_{p=1}^{M-1} \sum_{\pi \in \sigma_p} [B_p^{\pi} \cdot (S_M^{\pi} - S_p^{\pi}) + B_p^{\pi} \cdot S_p^{\pi}] + \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot S_M^{\pi} \leq C + \lambda^{max} \cdot L$$

Equivalently,

$$\sum_{p=1}^{M-1} \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot S_M^{\pi} + \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot S_M^{\pi} \leq C + \lambda^{max} \cdot L$$

Equivalently,

$$\sum_{p=1}^{M} \sum_{\pi \in \sigma_p} B_p^{\pi} \cdot S_M^{\pi} \leq C + \lambda^{max} \cdot L$$

Since the different permutations can have different processor capacity, we lower bound the processor capacity with the minimum processor capacity that is at least offered for any permutation. Based on the definition of S_M^{min} given by Eq. (7) and by the definition of B_p given by Eq. (2), we have:

$$\implies \sum_{p=1}^{M} B_p \cdot S_M^{min} \leq C + \lambda^{max} \cdot L$$

$$\sum_{p=1}^{M} B_p \leq \frac{C + \lambda^{max} \cdot L}{S_M^{min}}$$

Since the scheduler is greedy it holds that $T_M = \sum_{p=1}^{M} B_p$ and consequently

$$T_M \leq \frac{C + \lambda^{max} \cdot L}{S_M^{min}}$$

□

The proofs of the following Lemma (3) and Lemma (4) are given in the Appendix A.

Lemma 3. *The heterogeneity λ is always greater or equal to the maximum heterogeneity λ^{max} between the different permutations.*

$$\lambda^{max} \leq \lambda \tag{17}$$

Lemma 4. *The S_M is always less or equal to minimum processor capacity between the different permutations S_M^{min}.*

$$S_M^{min} \geq S_M \tag{18}$$

Theorem 2. ***Efficient makespan (EM):*** *The makespan of application G characterized by C and L on an unrelated multiprocessor platform is safe.*

$$T_M \leq \frac{C + \lambda \cdot L}{S_M} \tag{19}$$

Proof. From Lemma (3) and Lemma (4) it follows that the makespan calculated by using λ and S_M is always larger compared to the makespan calculate from Lemma (2). Since the makespan from Lemma (2) is safe, the makespan given by Eq. (19) is also safe (i.e., an upper bound). □

Since λ and S_M can be computed in polynomial time in terms of the number of tasks and number of processors, the makespan in Eq. (19) can be computed in polynomial time.

The EM 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 with the approaches that are provided from [6], [14] for the related and the homogeneous platform model respectively. For example, the homogeneous model can be modeled by assuming that $\delta_t^i = 1, \forall u_i \in G, 1 \leq t \leq H$. With this assumption $S_M = M$ and $\lambda = M - 1$. As a result, makespan calculation with the EM approach given by Theorem 2 is the same with the $T_M = L + (C - L)/M$ given by Lemma $IV.1$ in [14].

V. Simulation framework

This section presents the simulation framework that is used to evaluate the proposed methods for the makespan calculation. We use the same technique to generate the DAG models as in [11] for four applications Fibonacci, Sort, FFT and Strassen from the BOTS benchmark suit [12]. These applications are widely used in many different fields of computing (e.g., data processing, sorting, scientific applications, image processing, etc.). The application Fibonacci is a recursive parallel application with tree-like structure and is a good representative of many recursive applications. The application Sort is common operation in almost all fields of computing. The application Strassen is an efficient matrix multiplication algorithm that is used in many scientific applications. Finally, FFT is widely used in signal and image processing.

The considered applications have thousands of tasks. However, we note that the applications have only a few different task types (tasks that are functionally equivalent). For example, Fibonacci with input 20 has 32836 tasks while there is only 1 task type, the one that calculates the Fibonacci numbers. Similarly, for Sort, FFT, Strassen, there are 2, 3 and 1 task types, respectively. Similar is the number of task types for the applications implemented with OmpSs programming model: Cholesky factorization, QR factorization, Heat diffusion, and Integral Histogram that have 4, 4, 3 and 2 task types respectively for few thousands of tasks [15].

Based on the model of the applications from [11], there are three categories of nodes for every task type; $Spawn$, $Base$ and $Sync$ nodes that can have different WCET, where we use 300, 400 and 100 time units respectively for their e_i^{min}. The $Spawn$ node generates the parallel work, the $Base$ node performs the actual functionality of the task type, and the $Sync$ node synchronizes the output of the tasks that are generated from the same $Spawn$ node. Although there are three categories of each node, the total number of possible pairs of task types and node categories is significantly fewer in comparison to the total number of tasks. Tasks that have the same WCET for the different processors will lead to the same permutations. Consequently, we need to calculate all the permutations based on only the different task types which provide exponential complexity to the number of different task types rather than exponential in the number of tasks.

The WCET of a task for the different processor types is generated by adding a randomly generated value to the e_i^{min} (minimum WCET between the different processors types) which is given by the configuration of the applications. The randomly generated value is limited in a range which is given as parameter by the parameter $Limit$. More formally, the WCET of a task for the different processor types is given as follows, $e_i^t = e_i^{min} + Rand(0, Limit)$.

The configuration of the applications is presented in Table III. The columns are the different applications (Fibonacci, Sort, Strassen and FFT). The first row is the input of the applications, next is the total number of nodes that the applications have. At the third and fourth rows are the total workload (C) and the workload of the critical path (L) which characterizes the G^{min} of the applications. Next row presents the ratio of the workload of the critical path to the total workload. Finally, the last row shows the number of task types.

	Fib	Sort	Strassen	FFT
Input	20	32768	512	8192
#Nodes	32836	16043	22410	23748
C	8756400	4403300	7843300	6221400
L	8000	14900	2500	51020
$\frac{L}{C}$	0.0009	0.003	0.0003	0.008
#Task types	3	6	3	9

TABLE III
Application configurations

In addition to the real applications, experiments with synthetic DAGs were performed. A synthetic DAG is modeled by following a similar structure of the applications. We generated a fully balanced tree together with the mirror tree for the $Sync$ nodes. The maximum degree of the $Spawn$ nodes and the maximum height of the DAG, can be set as parameters. A time budget is assigned to every $Spawn$ node which is responsible for distributing it to its child nodes and the corresponding $Sync$ node to get the desirable C and L characteristics of the DAG. Next, the number of task types are given as a parameter to the DAG. The randomly generated WCETs with the use of the $Limit$ parameter are generated, with the same approach that we used for the applications.

For our simulations we use the following evaluation metrics:

Tightness: To the best of our knowledge, no other related work provides a closed form solution for the makespan calculation of parallel applications modeled as DAGs on an unrelated multiprocessor platform. We compare the *three* approaches for makespan calculation that we present in Section IV. The exhaustive makespan calculations $PM1$ and $PM2$ given by Theorem (1) and Lemma (2) respectively are compared to the EM makespan given by Theorem (2). The tightness is defined as the ratios $PM1/EM$ and $PM2/EM$.

Pessimism: To the best of our knowledge there is no analysis that finds the exact makespan of DAGs considering unrelated heterogeneous processor platform. However, we derive a lower bound on the makespan by simulating the actual execution of the parallel applications under the assumed greedy scheduler where all the tasks are executed for their WCET. If the application takes time Sim to finish its execution, then the pessimism of our approach is defined as the ratios Sim/EM. Note that even the optimal way to find the makespan has length not smaller than Sim.

VI. Simulation results

This section presents the results of the evaluation. Section VI-A presents the results of the makespan calculation for different number of processors and Section VI-B for different number of processor types. Section VI-C shows the impact of processor heterogeneity on the makespan calculation. Finally Section VI-D discusses the results from all the simulations.

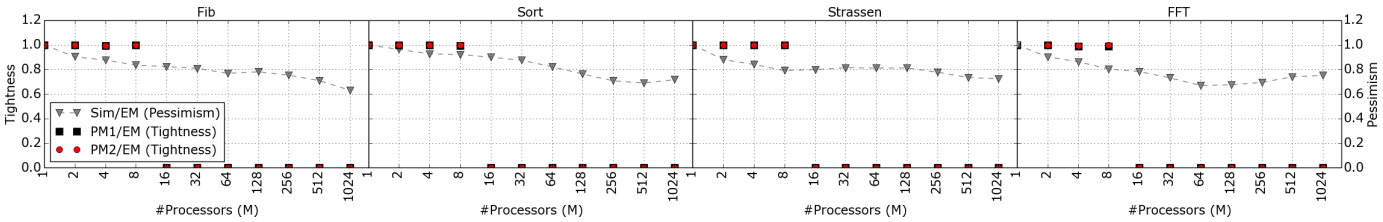

Fig. 4. Tightness and pessimism of EM for different number of processors, where the number of processor types is $H = min\{8, M\}$

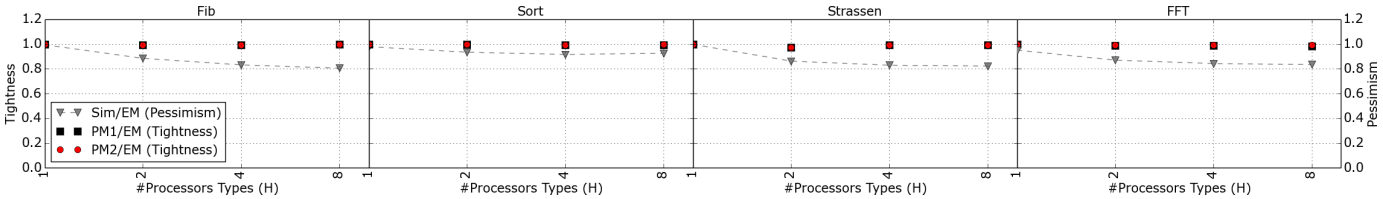

Fig. 5. Tightness and pessimism of EM for different processor types, where $M = 8$.

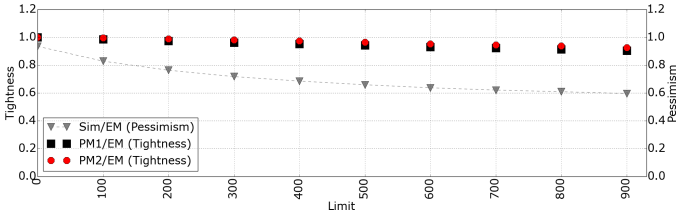

Fig. 6. Tightness and pessimism for different variations of the WCET.

A. Impact of changing the number of processors

Figure 4 presents the tightness and the pessimism of the applications Fibonacci, Sort, Strassen and FFT for the different number of processors where the number of processor types is up to $min\{8, M\}$. The horizontal axis shows the number of processors. The left vertical axis is the tightness and the right vertical axis is the pessimism. The points without the dashed line correspond to the tightness while with the dashed line correspond to the pessimism. In this graph, the closer to 1 are the values the better is the tightness and the pessimism.

The makespan calculation of EM has polynomial time complexity, and as a result, we generate the results for up to 1024 processors. On the contrary, the makespan calculation of $PM1$ and $PM2$ are the permutation-based approaches which have exponential time complexity. With our simulation setup, we can simulate only up to 8 processors. The $Limit$ is set to 100 and every makespan calculation was performed 100 times, and the average is reported.

Initially, it can be seen that for 1 processor, all the approaches are equal to C. Furthermore, we can see that for up to 8 processors, on average the tightness (the overestimation of the makespan) of EM compared to both $PM1$ and $PM2$ is less than 1% on average and up to 1.2% larger for all the applications (please note that the $PM1$ and $PM2$ have little difference and they are overalaping in the graph). We have performed the same simulations, but with $Limit$ equal to 500 and 1000. The average tightness of the makespan is slightly higher than 1% and up to 3%. Next, it can be noted for the pessimism that, by increasing the number of processors exponentially, the pessimism increases linearly. Compare to Sim we have on average 23% and up to 59% pessimism.

B. Impact of changing the number of processor types

Figure 5 shows the tightness and the pessimism of the makespan of EM for the different number of processor types when the total number of processors is eight for the four applications. The horizontal axis is the number of processor types for the four applications, the left vertical axis is the tightness and the right vertical axis is the pessimism. The $Limit$ for the random generation of the WCET is set to 100.

The tightness of the EM, compared to the two-permutation based approach, is on average 1% and at maximum 1.3%. Since for the calculation of heterogeneity and the processor capacity, we did not distinguish the processor types, but we consider the total number of processors, it is expected to have similar behavior with the results shown in Figure 4. As a result, the three approaches would be able to calculate the makespan regardless of the number of different processor types that exist in the system and consequently the determining factor to determine the makespan is the total number processors. Next, we can note that by increasing the number of processor types the pessimism increases since more processors are not executed with speed 1 which as a result leads to longer makespan. Compare to Sim we have on average 12% and up to 24% more pessimism. This result shows that if an exact makespan can be calculated for parallel application (which is very unlikely to happen), our analysis can still provide an upper bound on makespan which is at most 24% larger than the actual makespan. We, therefore, believe that our approach to find the makespan using EM is quite effective for the applications considered from the BOTS benchmark.

C. Impact of processor heterogeneity

To analyze the impact of the $Limit$ factor, which shows the variation of the WCET of a task among the different processor types, we continue the simulations with a synthetic

DAG. Figure 6 illustrates the tightness and the pessimism for different values of the $Limit$. The horizontal axis is the $Limit$, the left vertical axis presents the tightness, and the right vertical axis is the pessimism. Note that with $Limit = 1000$ we can have a variation on the WCET from $2.5x$ to $10x$ for $Spawn$ and $Sync$ nodes respectively that have 400 and 100 time units for their e_i^{min} values, but we use intentionally extreme values to expose the limitations of the EM. We have $C = 191400$ and $L = 5800$ for 938 nodes and 3 task types with ratio $\frac{L}{C} = 0.03$. Note that this ratio is 1 to 2 orders of magnitude higher compared to the BOTS applications, so the impact of heterogeneity would be higher. We use a platform with 4 processors and 2 processor types

By increasing $Limit$, which can be seen as making the platform more heterogeneous, the tightness of the EM approach decreases, on average we have 5% and maximum 11% less tight makespan compared to exhaustive approaches. Such increase in the makespan is due to the calculation of λ. It is taking the maximum unused accumulated capacity regardless of how the tasks are executed while the $PM1$ and $PM2$ only considers permutations where the same task is not allowed to execute on more than one processor. Next, it can be observed that the pessimism of EM compared to Sim increases as the $Limit$ increases since more processors have smaller speed and as a result the makespan of the EM increases. Compare to Sim we have on average 44% and up to 72% more pessimism. Note that although such values may seems quite high for our analysis, we would like to stress that the degree of heterogeneity for higher $Limit$ is quite pessimistic for many practical heterogeneous platform.

D. Discussion

In summary, we have seen that the EM has little extra pessimism compared to the exhaustive $PM1$ and $PM2$ approaches. We also have noted that by increasing the hardware heterogeneity (deviation of the WCET) the makespan can be increased significantly compared to Sim for the synthetic DAG (which are artificially created for stress testing), but for the DAGs of the BOTS applications in consideration we have experienced small differences of the tightness for the tested values. Simulations to investigate compatibility are presented in Appendix B to show how the makespan changes when the number of incompatible tasks is increased. We have seen that by increasing the number of incompatible processors the makespan increases since the parallelism is limited.

VII. Related work

To the best of our knowledge, no other work considers makespan calculation of parallel applications modeled as DAG executed on an unrelated multiprocessor platform.

Plethora of heterogeneous architectures are proposed in the literature; architectures with singe instruction set architecture (ISA) with different microarchitectures [8], coexistence of architectures with different ISA [16], special purpose accelerators for convolution [17] and matrix multiplication [18].

The authors of [6] (in the first part of the paper) proposed a makespan calculation for the related multiprocessor platform [6] with the use of uniformity and processor capacity, which is based on previous work [5]. In [7] the authors extended the scheduler of Cilk [19] for related heterogeneous systems where the speed of the processors are different and fixed throughout the execution. They provide a makespan calculation methodology for Cilk based applications and an approximation of the makespan based on the average speed of the processors. The related multiprocessor model is a special case of the unrelated model. Our proposed makespan calculations can also be applied to related multiprocessors, and our approach will have the same makespan as in [6] for related machines.

The authors of [20], [21] provide static scheduling for unrelated heterogeneous platforms where the applications are modeled as DAGs with the goal to minimize the schedule lenght. In [4], [9] the authors consider the problem of scheduling independent tasks on a two type unrelated heterogeneous multiprocessor platform. Since the independent tasks model is a special case of the DAG model, by setting the $L = 0$ the proposed makespan calculations can also be applied.

In [22], [23] the authors provide upper bounds by using competitive analysis based on the different types of processors for cloud applications. The authors consider a multiprocessor platform where the type of the task and the type of the processors need to match in order to be executed. In [24], the authors provide response-time bounds for DAG based applications. They consider heterogeneous multiprocessor platform where every task is compatible with one processor type. Processors of the same type share a common ready queue and non-preemptive global EDF is used. Initially they transform the DAG to independent tasks with offsets to preserve dependencies between the tasks of the DAG and then they adapt their analysis with previous work [25]. In contrast, we use a general greedy scheduler and consider computing the makespan of a single DAG task without transforming the node as independent tasks with offsets. The contribution of our work is unique for real-time computing considering the more general heterogeneous processor model, a general greedy scheduler, and DAG task model without requiring any transformation.

VIII. Conclusion

This paper considers the problem of calculating the makespan of task-based parallel applications modeled as a DAG executed on heterogeneous multiprocessor platforms model using the unrelated modeling framework. To the best of our knowledge, this is the first work that provides a closed-form solution to the makespan calculation under these assumptions. A combinatorial analysis is used to construct two closed-form makespan calculations that are used to build a third lower time complexity makespan calculation. The evaluation is performed by modeling four OpenMP task-based parallel applications as DAGs and synthetic DAGs. The simulation results have shown that the length of the makespan using the EM approach is very close to that of the two exhaustive approaches.

REFERENCES

[1] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *IEEE ISCA*, 2011.

[2] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *ACM-SIAM SODA*, 2012.

[3] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *IEEE FOCS*, 2014.

[4] Hoon Sung Chwa, Jaebaek Seo, Jinkyu Lee, and Insik Shin. Optimal real-time scheduling on two-type heterogeneous multicore platforms. In *IEEE RTSS*, 2015.

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

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

[7] Michael A Bender and Michael O Rabin. Scheduling cilk multithreaded parallel programs on processors of different speeds. In *ACM SPAA*, 2000.

[8] 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.

[9] Gurulingesh Raravi, Björn Andersson, and Konstantinos Bletsas. Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. *Springer RTS*, 2013.

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

[11] Petros Voudouris, Per Stenström, and Risat Pathan. Timing-anomaly free dynamic scheduling of task-based parallel applications. In *IEEE RTAS*, 2017.

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

[13] Douglas Brent West et al. *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001.

[14] 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.

[15] Kallia Chronaki et al. Criticality-aware dynamic task scheduling for heterogeneous architectures. In *ACM, ICS*, 2015.

[16] 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.

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

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

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

[20] Gilbert C Sih and Edward A Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE TPDS*, 1993.

[21] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE TPDS*, 2002.

[22] Yuxiong He, Hongyang Sun, and Wen-Jing Hsu. Adaptive scheduling of parallel jobs on functionally heterogeneous resources. In *ICPP*, 2007.

[23] Yuxiong He, Jie Liu, and Hongyang Sun. Scheduling functionally heterogeneous systems with utilization balancing. In *IEEE IPDPS*, 2011.

[24] Kecheng Yang, Ming Yang, and James H Anderson. Reducing response-time bounds for dag-based task systems on heterogeneous multicore platforms. In *ACM RTNS*, 2016.

[25] Kecheng Yang and James H Anderson. Optimal gedf-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *IEEE ESTIMedia*, 2014.

APPENDIX

A. Proofs of Lemma 3 and Lemma 4

This section presents the proofs for Lemma (3) and Lemma (4) that are used for the proof of Theorem 2 in Section IV.

Proof of Lemma (3)

Proof. For an arbitrary task $1 \leq \pi_y \leq N$ that it is executed on the y^{th} fastest processor for the task it holds that:

$$\delta_y^{\pi_y} \leq \max_{j=1}^{N}\{\delta_y^j\}$$

Equivalently,

$$\sum_{y \in Sl_{\pi_x}^x}\{\delta_y^{\pi_y}\} \leq \sum_{y \in Sl_{\pi_x}^x} \max_{j=1}^{N}\{\delta_y^j\}$$

By dividing both sides with the speed of the x^{th} fastest processor $\delta_x^{\pi_x} \neq 0$, of task π_x we have:

$$\frac{\sum_{y \in Sl_{\pi_x}^x}\{\delta_y^{\pi_y}\}}{\delta_x^{\pi_x}} \leq \frac{\sum_{y \in Sl_{\pi_x}^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^{\pi_x}}$$

Since the inequality holds for any processor x it holds also for the processor that maximizes the two sides of the inequality

$$\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_{\pi_x}^x}\{\delta_y^{\pi_y}\}}{\delta_x^{\pi_x}}\} \leq \max_{x=1}^{M}\{\frac{\sum_{y \in Sl_{\pi_x}^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^{\pi_x}}\}$$

Since it holds for an arbitrary task with index π_x that belongs to an arbitrary permutation π it holds also for any task of the application $1 \leq j \leq N$ and we have

$$\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_{\pi_x}^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^{\pi_x}}\} = \max_{x=1}^{M}\{\frac{\sum_{y \in Sl_k^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^k}\} \quad (20)$$

Since $1 \leq k \leq N$ it also holds that

$$\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_k^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^k}\} \leq \max_{i=1}^{N}\{\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_i^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^i}\}\} \quad (21)$$

From (20) and (21) we have

$$\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_{\pi_x}^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^{\pi_x}}\} \leq \max_{i=1}^{N}\{\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_i^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^i}\}\}$$

Let π' be the permutation that provides the maximum value for the maximum accumulated capacity loss. Since it holds for any π it holds also for the permutation π', we have:

$$\max_{k=1}^{M}\{\frac{\sum_{y \in Sl_{\pi_x'}^x}\{\delta_y^{\pi_y'}\}}{\delta_x^{\pi_x'}}\} \leq \max_{i=1}^{N}\{\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_i^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^i}\}\}$$

Equivalently,

$$\max_{\pi \in \sigma_M}\{\max_{k=1}^{M}\{\frac{\sum_{y \in Sl_{\pi_x}^x}\{\delta_y^{\pi_y}\}}{\delta_x^{\pi_x}}\}\} \leq \max_{i=1}^{N}\{\max_{x=1}^{M}\{\frac{\sum_{y \in Sl_i^x} \max_{j=1}^{N}\{\delta_y^j\}}{\delta_x^i}\}\}$$

From the definitions of λ^{max}, λ and $idle_x^i$ given by equations (8), (11) and 10 we have: $\lambda^{max} \leq \lambda$. As a result the λ is always higher or equal (for the case where the maximum unused capacity and the speed of the node that executes the critical path are derived form the same permutation) to λ^{max}. □

Proof of Lemma (4)

Proof. For an arbitrary task π_x that it is executing on its x^{th} fastest processor it holds that:

$$\delta_x^{\pi_x} \geq \min_{i=1}^{N}\{Prf_x^i\}$$

Since the size of the Prf^i is equal to the number of processor M it holds that:

$$\sum_{x=1}^{M}\{\delta_x^{\pi_x}\} \geq \sum_{x=1}^{M} \min_{i=1}^{N}\{Prf_x^i\}$$

Since it holds for an arbitrary permutation π it holds also for the permutation that provides the minimum value.

$$\min_{\pi \in \sigma_M}\{\sum_{x=1}^{M}\delta_x^{\pi_x}\} \geq \sum_{x=1}^{M} \min_{i=1}^{N}\{Prf_x^i\}$$

From the definitions of S_M^{min} and S_M given by equations (7) and (9) respectively, the statement of the lemma holds. □

B. Impact of task-processor compatibility

Figure 7 presents the impact of having tasks that are not compatible with all the processors. The horizontal axis presents the speed threshold for the applications Fibonacci, Sort, Strassen, and FFT; the speed for a processor type of a task type is set to be incompatible ($\delta_t^i = 0$) if it has speed smaller than the threshold. The vertical axis presents the pessimism of the EM approach with respect to the Sim. The considered platform has 32 processors for 4 processor types. The $Limit$ is set to 100 for this experiment.

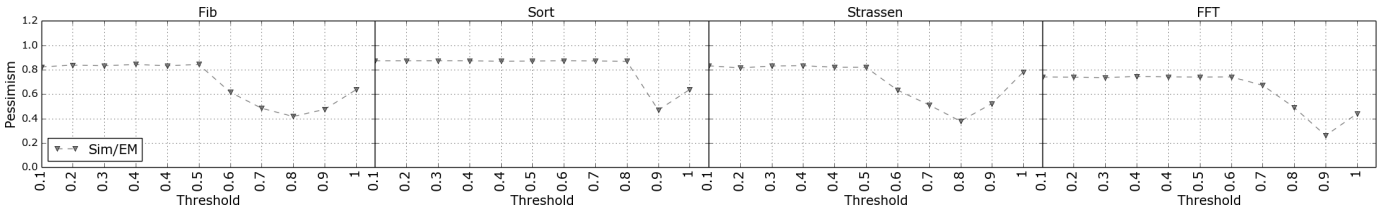

Fig. 7. Impact of making tasks incompatible to a processor type that have speed smaller than a threshold ($M = 32$, $H = 4$).

Initially, for threshold 0.1 all the applications have pessimism close to 25% since the scheduler can find some compatible available processor type due to high number of processors but the calculation of the makespan needs to pessimistically assume the speed equal to 0 for the calculation of EM. Next, it can be seen that for all the application as the threshold increases (relatively more incompatible tasks) the pessimism initially remains constant and then increases since fewer processors are available for the tasks to execute on. Next, we can note that the pessimism starts to increase for Fibonacci and Strassen from speed threshold 0.5 while for Sort and FFT the tightness begins to decrease after 0.8 and 0.7 respectively. Since Fibonacci and Strassen have fewer task types, 3 task types each, compared to Sort and FFT that have 6 and 9 respectively, more tasks are characterized to this single 3 task types for the former (i.e., Sort and FFT) cases. As a result, more tasks have fewer processors to be executed for Sort and FFT. Finally, we can see that the pessimism decreases for large numbers of threshold since the scheduler cannot find available processors to schedule the tasks and the total execution of the schedule increases. Consequently, the pessimism compared to Sim decreases.

C. Table of symbols

	Description
M	Number of processors.
H	Number of processor types.
N	Number of application tasks.
u_i	Task with index i.
e_i^t	WCET of task i for processor type t.
G	DAG of the application.
γ	A path in G.
G^{min}	Isomorphic DAG with G where for each node u_i. it holds $e_i^{min} = \min_{t=1}^{H}\{e_i^t\}$ for $i = 1, 2, \dots N$.
cp	Critical path in G^{min}.
L	The sum of the WCETs of all the tasks of cp.
C	The sum of the WCETs of all the tasks of G^{min}.
δ_t^i	Speed of processor type t with respect to task u_i.
Prf^i	Set of processor-type speeds in non-increasing order for u_i.
Prf_x^i	The speed of the x^{th} fastest processor for task u_i.
Sl_x^i	The set of the slower processors in Sl_x^i when u_i is executing on its x^{th} fastest processor.
π	Different mappings (i.e., execution scenarios) of the tasks on p processors.
π_x	The x^{th} task in permutation π.
σ_p	The set of all the permutations of size p.
B_p	The sum of time intervals where exactly p processors are busy.
B_p^π	The sum of intervals where exactly p processors are busy for the permutation π of tasks of size p.
L_p^π	The amount of the workload that belongs to the cp and it is consumed when p processors are busy by the tasks that are present in permutation π.
L_p	The amount of the workload that belongs to the cp and it is consumed when p processors are busy for all the permutations of size p.
C_p^π	The amount of the total workload that is consumed when p processors are busy by tasks belonging to the permutation π.
C_p	The amount of the total workload that is consumed when p processors are busy.
S_x^π	The processor capacity for a permutation π.
λ^π	The heterogeneity for a permutation π.
S_M^{min}	The minimum processor capacity among all permutations.
λ^{max}	The maximum heterogeneity among all permutations.
S_M	The minimum processor capacity.
$idle_x^i$	The maximum unused capacity, when task u_i is executing on its x^{th} faster processor.
λ	Heterogeneity

TABLE IV
DESCRIPTION OF THE MODEL PARAMETERS