Bi-objective Scheduling Algorithms for Optimizing Makespan and Reliability on Heterogeneous Systems

Jack J. Dongarra
University of Tennessee
Department of Computer Science
1122 Volunteer Blvd.
Knoxville, TN 37996, USA
dongarra@cs.utk.edu

Emmanuel Jeannot
INRIA-Lorraine, LORIA
615, rue du Jardin Botanique
54600 Villers les Nancy, France
emmanuel.jeannot@loria.fr

Erik Saule
ID-IMAG/LIG
Laboratoire LIG - UMR 5217
51, rue Jean Kuntzmann
38330 Montbonnot St. Martin, France
erik.saule@imag.fr

Zhiao Shi
University of Tennessee
Department of Computer Science
1122 Volunteer Blvd.
Knoxville, TN 37996, USA
shi@cs.utk.edu

ABSTRACT

We tackle the problem of scheduling task graphs onto a heterogeneous set of machines, where each processor has a probability of failure governed by an exponential law. The goal is to design algorithms that optimize both makespan and reliability. First, we provide an optimal scheduling algorithm for independent unitary tasks where the objective is to maximize the reliability subject to makespan minimization. For the bi-criteria case, we provide an algorithm that approximates the Pareto-curve. Next, for independent non-unitary tasks, we show that the product {*failure rate*}× {*unitary instruction execution time*} is crucial to distinguish processors in this context. Based on these results we are able to let the user choose a trade-off between reliability maximization and makespan minimization. For general task graphs we provide a method for converting scheduling heuristics on heterogeneous cluster into heuristics that take reliability into account. Here again, we show how we can help the user to select a trade-off between makespan and reliability.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Reliability, availability, and serviceability]; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Reliability, Performance

Keywords

Scheduling, pareto-curve, reliability, DAG

1. INTRODUCTION

Unlike homogeneous parallel machines, heterogeneous systems are composed of computers with different speeds. As the number of interconnected heterogeneous resources is growing tremendously, the need for algorithmic solutions that efficiently use such platforms is increasing as well. One of the key challenges of heterogeneous computing is the scheduling problem. Given an application modeled by a dependence graph, the scheduling problem deals with mapping each task of the application onto the available heterogeneous resources in order to minimize the application runtime (makespan). This problem has been studied in the literature for years and is known to be NP-hard. Several heuristics have been proposed to solve this problem [8, 4, 2]. However, as heterogeneous systems become larger and larger, the issue of reliability of such an environment needs to be addressed. Indeed, the number of possible failures increases with the size of the hardware. Therefore, nowadays, it is not possible to ignore the fact that an application running on a very large system can crash due to hardware failure. Several approaches can be employed to solve this problem. One approach is based on task duplication where each task is executed more than once in order to decrease the probability of failure. The main problem of this approach is that it increases the number of required processors. Alternatively, it is possible to checkpoint the application and restart the application after a failure [3, 1]. However, in case of failure, the application is slowed down by the restart mechanism, which requires the user to restart the application on a subset of processors and repeat some communications and computations. Hence, in order to minimize the impact of the restart mechanism, it is important to reduce the risk of failure. Moreover, even in the case where there is no checkpoint-restart mechanism, it is important to guarantee that the probability of failure of the application is kept as low as possible. Unfortunately, as we will show in this paper, increasing the reliability implies, most of the time, an increase of the execution time (a fast schedule is not necessarily a reliable one). This justifies the search for algorithms that both minimize makespan and maximize reliability.

In this paper, we take on the problem of scheduling an application modeled by a task graph on a set of heterogeneous resources. The objectives are to minimize the makespan and to maximize the reliability of the schedule. In the literature, this problem has been studied [5, 6, 7], but not from a theoretical optimization point of view. In [5], Dogan and Ozgüner proposed a bi-criteria heuristic called RDLS. In [6], the same authors improved their solution using a genetic algorithm based approach. In [7], Hakem and Butelle proposed BSA, a bi-criteria heuristic that outperforms RDLS. However all these results are focused on the general case where the task graph is unstructured. We lack fundamental solutions for this problem. Some unanswered questions are:

- Is maximizing the reliability a difficult (NP-Hard) problem?
- Is it possible to find polynomial solution of the problem for special kind of task graph?
- Is it possible to approximate this problem?
- Can we build approximation schemes?
- How to help the user in finding a good trade-off between reliability and makespan?

The content and the organization of this paper are as follows. In section 2, we introduce the notation and the definition of reliability and makespan. In section 3, we describe the problem more precisely. In particular, we show that maximizing the reliability is a polynomial problem and is simply obtained by executing the application on the processors that have the smallest product of {*failure rate*} and {*unitary instruction execution time*} sequentially. This means that minimizing the makespan is contradictory to the objective of maximizing the reliability. Furthermore, we show that for the general case, approximating both criteria at the same time is not possible. In section 4, we study the problem of scheduling a set of unitary (each tasks have the same duration : one instruction) and independent tasks. For this case, we propose an optimal algorithm for the problem that maximizes the reliability subject to makespan minimization. We also propose an $(1+\epsilon,1)$ approximation of the Pareto-curve[1] of the problem (these terms will be defined later). This means that we can provide a set of solutions to the problem (the size of this set being polynomial) that approximates, at a constant ratio, all the optimal makespan/reliability trade-offs. In section 5, we investigate the problem for non-unitary independent tasks. We propose an approach to help the user choose a suitable makespan/reliability trade-off driven by the fact that, in order to favor reliability, it is necessary to schedule tasks on a subset of processors that have the smallest product of {*failure rate*} and {*unitary instruction execution time*}. Based on that fact, in section 6 we show, for the case where the task graph is unstructured, how to easily transform a heuristic that targets makespan minimization to a bi-criteria heuristic. Here again, we demonstrate how to help the user choose a suitable makespan/reliability trade-off. We conclude the paper in section 7.

[1]Intuitively, the Pareto-curve divides the solution space between the feasible and the unfeasible.

2. NOTATIONS

We model the application by a task graph: let $G = (V, E)$ be a Directed Acyclic Graph (DAG) with V the set of $n = |V|$ vertices (that represenst tasks) and E the set of edges that represents precedence constraints among the tasks. Each task $v_i \in V$ is given a number of operations o_i. Each edge $e_i = (i, j)$ is associated to l_i and the time to send data from task v_i to task v_j if they are not executed on the same processor. We are given a set P of m processors and processor $p_i \in P$ is associated with two values: τ_i the *unitary instruction execution time*, *i.e.* the time to perform one operation, and λ_i the {*failure rate*}. We assume that, during the execution of the DAG, the failure rate is constant. This means that the failure model follows an exponential law. Hence, each task v_i executed on processor p_j will last $o_i \times \tau_j$ and the probability that this task finishes (correctly) its execution is given by $e^{-o_i \times \tau_j \times \lambda_j}$.

A schedule is an assignment of the tasks to the processors such that, at most, one task is executed at a time on any given processor and that the precedence constraints are respected. We call proc(i) the processor assigned to task v_i and C_i its completion date. $C_{\max} = \max\{C_i\}$ is the makespan.

The reliability of a schedule is the probability that it finishes correctly and is given by the probability that all the processors be functional during the execution of all their assigned tasks, *i. e.*, $p_{\text{succ}} = e^{-\sum_{j=1}^{m} C^j_{\max}\lambda_j}$, where $C^j_{\max} = \max_{i|\text{proc}(i)=j}\{C_i\}$ is the completion date of the last task executed on processor j. Finally, the probability of failure of a schedule is given by $p_{\text{fail}} = 1 - p_{\text{succ}}$.

The problem we address is, that given a task graph and a set of heterogeneous processors, find a schedule such that p_{succ} is maximized and $C_{\max}$ is minimized.

3. A BI-CRITERIA PROBLEM

In this article we try to solve a bi-criteria problem. We aim to minimize the makespan and to maximize the reliability (or to minimize the probability of failure). Unfortunately, these criteria are two conflicting objectives. More precisely, as shown in Proposition 1, the optimal reliability is obtained when mapping all the tasks to processor j such that, $j = \text{argmin}(\tau_i\lambda_i)$, *i.e.*, the one for which the product of {*failure rate*} and {*unitary instruction execution time*} is minimal. However, such a schedule, in the general case, can be arbitrarily far from the optimal one.

PROPOSITION 1. *Let S be a schedule where all the tasks have been assigned, in topological order, to a processor i such that $\tau_i\lambda_i$ is minimal. Let p_{succ} be the probability of successful execution of schedule S. Then any schedule $S' = S$, with probability of success p'_{succ}, is such that $p'_{succ} \leq p_{succ}$.*

Proof of proposition 1.

Suppose without loss of generality that $i = 0$ (*i. e.*, $\forall j : \tau_0\lambda_0 \leq \tau_j\lambda_j$). Then $p_{\text{succ}} = e^{-C^0_{\max}\lambda_0}$. Call $C'^j_{\max}$ the completion date of the last task on processor j with schedule S'. Therefore, $p'_{\text{succ}} = e^{-\sum_{j=0}^{m} C'^j_{\max}\lambda_j}$. Let T be the set of tasks that are not executed on processor 0 by schedule S'.

Then, $C'^{0}_{\max} \geq C^0_{\max} - \tau_0 \sum_{v_i \in T} o_i$ (there are still tasks of $V \backslash T$ to execute on processor 0). Let $T = T_1 \cup T_2 \cup \ldots \cup T_m$, where T_j is the subset of the tasks of T executed on processor j by schedule S' (these sets are disjoint: $\forall j_1 \neq j_2$, $T_{j_1} \cap T_{j_2} = \emptyset$). Then, $\forall 1 \leq j \leq m$, $C'^{j}_{\max} \geq \tau_j \sum_{v_i \in T_j} o_i$. Let us compare the exponent of p_{succ} and p'_{succ}. We have:

$$\begin{aligned}
& \sum_{j=0}^{m} C'^{j}_{\max} \lambda_j - C^0_{\max} \lambda_0 \\
\geq \;& C^0_{\max}\lambda_0 - \tau_0 \lambda_0 \sum_{v_i \in T} o_i + \sum_{j=1}^{m} \left(\tau_j \lambda_j \sum_{v_i \in T_j} o_i \right) - C^0_{\max}\lambda_0 \\
= \;& \sum_{j=1}^{m} \left(\tau_j \lambda_j \sum_{v_i \in T_j} o_i \right) - \tau_0 \lambda_0 \sum_{v_i \in T} o_i \\
= \;& \sum_{j=1}^{m} \left(\tau_j \lambda_j \sum_{v_i \in T_j} o_i \right) - \tau_0 \lambda_0 \sum_{j=1}^{m} \left(\sum_{v_i \in T_j} o_i \right) \\
& \text{(because the } T_j\text{'s are disjoint)} \\
= \;& \sum_{j=1}^{m} \left((\tau_j \lambda_j - \tau_0 \lambda_0) \sum_{v_i \in T_j} o_i \right) \\
\geq \;& 0 \\
& \text{(because } \forall j : \ \tau_0 \lambda_0 \leq \tau_j \lambda_j\text{)}
\end{aligned}$$

Hence,

$$\frac{p_{\text{succ}}}{p'_{\text{succ}}} = e^{\sum_{j=0}^{m} C'^{j}_{\max}\lambda_j - C^0_{\max}\lambda_0} \geq 1$$

■

Since it is not possible to optimize both criteria at the same time, we will tackle the problem as follows: we consider maximizing the reliability subject to the condition that the makespan is minimized ([10] Chap. 3, pp. 12). This means that we will try to find the most reliable schedule among the ones that minimize the makespan. However, since finding the optimal makespan is most of the time NP-hard, we aim at designing an (α, β)-approximation algorithm. In our case an (α, β)-approximation algorithm is an algorithm with which the solution found has a makespan at most α times larger than the optimal one, and the probability of failure is at most β times larger than the optimal probability of failure (among the schedules that minimizes the makespan). Before proceeding, we shall make two important remarks.

1. Proposition 1 shows that the problem of minimizing the makespan subject to the condition that the reliability is maximized is the problem of minimizing the makespan using only processors having a minimal $\tau_i \lambda_i$. If there is only one single processor that minimizes $\tau_i \lambda_i$, it reduces to a trivial problem. In this case, the reliability is maximized only if all the tasks are sequentially executed on the processor whose $\lambda_i \tau_i$ is minimal. However, in the case where there are several processors that have the same minimal $\lambda_i \tau_i$ value, the problem is NP-Hard as it requires minimizing the makespan on all of these processors.

2. Theorem 1 (above) shows that the problem of optimizing both criteria is unapproximable. That is to say, in general, there exists no solution which is not too far from the optimal value on both criteria at the same time. Our study is related to the search of one *Pareto optimum*[2] when the makespan is minimized. Therefore, the problem we tackle here may be seen as giving the priority to the makespan, and trying to optimize the reliability as a secondary criteria. It is a useful approach for many cases as it can lead to approximating the Pareto-curve of the problem. An approximation of a Pareto curve is presented in Section 4.2 on UET independent tasks.

THEOREM 1. *The problem of minimizing the C_{max} and maximizing p_{succ} is unapproximable within a constant factor.*

Proof of Theorem 1.

We consider the instance of the problem involving two machines such that $\tau_2 = \tau_1/k$ and $\lambda_2 = k^2 \lambda_1$ ($k \in \mathbb{R}^{+*}$). Consider a single task t_1. Only two solutions are feasible S_1 in which the task is scheduled on processor 1 and S_2 in which the task is scheduled on processor 2. Note that S_2 is optimal for $C_{\max}$ and that S_1 is optimal for the reliability.

$C_{\max}(S_1) = o_1 \tau_1$ and $C_{\max}(S_2) = o_1 \tau_1 / k$. This leads to $C_{\max}(S_1)/C_{\max}(S_2) = k$. This ratio goes to the infinity with k. Thus, S_1 is not a constant-factor approximation for this instance.

$p_{\text{succ}}(S_1) = e^{-o_1 \tau_1 \lambda_1}$ and $p_{\text{succ}}(S_2) = e^{-o_1 \tau_1 \lambda_1 k}$. This leads to $p_{\text{succ}}(S_1)/p_{\text{succ}}(S_2) = e^{o_1 \tau_1 \lambda_1 (k-1)}$. This ratio goes to the infinity with k. Thus, S_2 is not a constant-factor approximation for this instance.

None of the solutions can approximate both criteria within a constant-factor.

■

We aim at finding a solution for which probability of success is at most β times the optimal probability of failure (again, subject to makespan minimization). However, in some cases, such a bound might not be very useful. For instance imagine that for a given algorithm $\beta = 5$ and the optimal probability of failure $\tilde{p}_{\text{fail}} = 0.3$, then we have $p_{\text{fail}} \leq \beta \cdot \tilde{p}_{\text{fail}} = 5 \times 0.3 = 1.5$, which means that we have no useful information on the probability of failure since, as for any probability, $p_{\text{fail}} \leq 1$. One way to tighten the bound is given by proposition 2.

PROPOSITION 2. *Let p_{succ} (resp. p_{fail}) be the probability of success (resp. of failure) of a schedule S. Let $\tilde{p}_{succ}$ (resp. $\tilde{p}_{fail}$) be the optimal probability of success (resp. of failure) for the same input as S. Let β be a real in $[1, +\infty[$. Then $p_{succ} = \tilde{p}^{\beta}_{succ} \Rightarrow p_{fail} \leq \beta \cdot \tilde{p}_{fail}$.*

Proof of proposition 2.

The proof is based on the Bernoulli's inequality where, $\forall x \in [0, 1], \forall n \in [1, +\infty[, (1 - x)^n \geq 1 - nx$.

$$\begin{aligned}
p_{\text{fail}} &= 1 - p_{\text{succ}} = 1 - \tilde{p}^{\beta}_{\text{succ}} = 1 - (1 - \tilde{p}_{\text{fail}})^{\beta} \\
&\leq 1 - (1 - \beta \cdot \tilde{p}_{\text{fail}}) = \beta \cdot \tilde{p}_{\text{fail}}
\end{aligned}$$

■

Based on that proposition, in the remainder of the paper, we will target solutions where the probability of success is bounded by a power of the optimal probability of success.

[2]In a multi-criteria problem, a solution is called *Pareto optimal* if any other solution increases at least one of the criteria.

4. UNITARY INDEPENDENT TASKS

4.1 Optimal Algorithm

Given a makespan objective M, we show how to find a task allocation that is the most reliable for a set of N unitary independent tasks ($\forall v_i \in V,\ o_i = 1$ and $E = \emptyset$).

We consider minimizing the probability of failure subject to the condition that the makespan is constrained. Since tasks are unitary and independent, the problem is then to find for each processor $p_j,\ j \in [1, m]$ a positive integer n_j such that the following constraints are met: (1) $\sum_{j\in[1,m]} n_j = n = |V|$. (2) The makespan is α-constrained $n_j\tau_j \leq \alpha M_{\text{opt}}$, $\forall j \in [1, m], \alpha \in [1, +\infty[$, and α is given by the user. (3) Subject to the previous constraints, the probability of success is maximized: $e^{-\sum_{j\in[1,m]} n_j\lambda_j\tau_j}$ is maximized, *i.e.*, $\sum_{j\in[1,m]} n_j\lambda_j\tau_j$ is minimized.

First, it is important to note that finding a schedule whose makespan is smaller than a given objective M can be found in polynomial time. Indeed, Algorithm 1 finds the minimal makespan allocation for any given set of independent unitary tasks as shown in [9], pp. 161.

Algorithm 1 Optimal allocation for independent unitary task

```
for i=1:m
  $n_i \leftarrow \left\lfloor \frac{1/\tau_i}{\sum 1/\tau_i} \right\rfloor \times n$
while $\sum n_i < n$
  $k = \text{argmin}(\tau_k(n_k + 1))$
  $n_k \leftarrow n_k + 1$
```

Second, we propose Algorithm 2 to solve the problem. It finds an optimal allocation as proved by Theorem 2. It is a greedy algorithm that simply allocates tasks to the processors taken in the increasing $\lambda_i\tau_i$ value up to the makespan objective M.

Algorithm 2 Optimal reliable allocation for independent unitary task

```
Input: $\alpha \in [1, +\infty[$
Compute $M = \alpha M_{\text{opt}}$ using algorithm 1.
Sort the processor by increasing $\lambda_i\tau_i$
$X \leftarrow 0$
for i=1:m
  if $X < N$
    $n_i \leftarrow \min\left(N - X, \left\lfloor \frac{M}{\tau_i} \right\rfloor\right)$
  else
    $n_i \leftarrow 0$
  $X \leftarrow X + n_i$
```

THEOREM 2. *At the end of Algorithm 2, all the constraints are fulfilled.*

Proof of Theorem 2.

Let X be the number of tasks already assigned. Since when $X < N$ we allocate at most $N - X$ tasks to a processor, at the end $X = N$. For each processor i we allocate at most $\lfloor \frac{M}{\tau_i} \rfloor$ tasks, hence $n_i\tau_i \leq M = \alpha M_{\text{opt}}$. Since in algorithm 1, the order of the tasks and the order of the processors is not taken into account, algorithm 2 is valid (*i.e.*, all tasks are assigned using at most the m processors). We need to show that $\sum_{i\in[1,m]} n_i\lambda_i\tau_i$ is minimum. First let us remark that algorithm 2 fills the processors with tasks in the increasing order of the $\lambda_i\tau_i$ values. Hence, any other valid allocation $\{n'_1, \ldots, n'_N\}$ is such that $n'_i < n_i$ and $n'_j > n_j$ for any $i < j$. Without loss of generality, let us assume that $n'_1 = n_1 - k$, $n'_i = n_i + k$ and $n'_j = n_j$ for $k \in [1, n_i]$, $j \neq 1$ and $j \neq i$. Then the difference between the two objective values is

$$\begin{aligned} D &= n_1\lambda_1\tau_1 + \ldots + n_i\lambda_i\tau_i + \ldots n_N\lambda_N\tau_N \\ &\quad - n'_1\lambda_1\tau_1 - \ldots - n'_i\lambda_i\tau_i - \ldots n'_N\lambda_N\tau_N \\ &= \lambda_1\tau_1(n_1 - n'_1) + \lambda_i\tau_i(n_i - n'_i) \\ &= -k\lambda_1\tau_1 + k\lambda_i\tau_i \\ &= k(\lambda_i\tau_i - \lambda_1\tau_1) \\ &\geq 0 \text{ because } \lambda_i\tau_i \geq \lambda_1\tau_1. \end{aligned}$$

Hence, the first allocation has a smaller objective value. ∎

4.2 Pareto curve approximation

When multi-criteria problems are unapproximable within a constant factor of all criteria with a single solution, we usually look for a set of solutions that are interesting trade-offs. The **Pareto curve** of an instance is the set of all Pareto-optimal solutions. Remember that a Pareto-optimal solution is such that there exists no other solution having a better or equal value on all criteria and improving at most one criterion. Therefore, intuitively, the Pareto-curve divides the solution space between feasible and unfeasible solutions. Unfortunately, the cardinality of the Pareto curve of an instance is often exponential with respect to the size of the instance. Thus, we look for an approximation of Pareto-curve. Informally, $\mathcal{P}$ is a $\rho = (\rho_1, \rho_2, \ldots, \rho_k)$ approximation of the Pareto-Curve $\mathcal{P}^*$ if each solution $S^* \in \mathcal{P}^*$ is ρ approximated by a solution $S \in \mathcal{P}$. Formally, for minimization $\forall S^* \in \mathcal{P}^*, \exists S \in \mathcal{P}, \forall i, f_i(S) \leq \rho_i f_i(S^*)$.

A generic method to obtain an approximated Pareto curve is given in [13]. The idea is to partition the solution space by geometrically increasing the size of ratio $1 + \epsilon$ among all criteria (see Fig. 1. The set formed by one solution in each partition (if any) is an ϵ-approximation of the Pareto curve of the problem. We use the same idea to construct an $(1 + \epsilon, 1)$-approximation of the Pareto-curve of the problem.

Let M be a feasible makespan under constraints of optimal reliability. We can compute M_{max} using Proposition 1 in scheduling all tasks on processors minimizing $\tau_i\lambda_i$. Algorithm 3 constructs a set of solutions $S = \{S_1, \ldots, S_k\}$ such that S_i is constructed by Algorithm 2 using $\alpha = (1 + \epsilon)^i$ and k is the first integer such that $(1 + \epsilon)^k M_{\text{opt}} \geq M_{max}$. Theorem 3 shows that S is an $(1 + \epsilon, 1)$-approximation of the Pareto-curve of the problem.

THEOREM 3. *Algorithm 3 returns an $(1+\epsilon, 1)$-approximation of the Pareto-curve of the problem of minimizing the makespan and maximizing the reliability for independent unitary tasks on heterogeneous speed-related processors.*

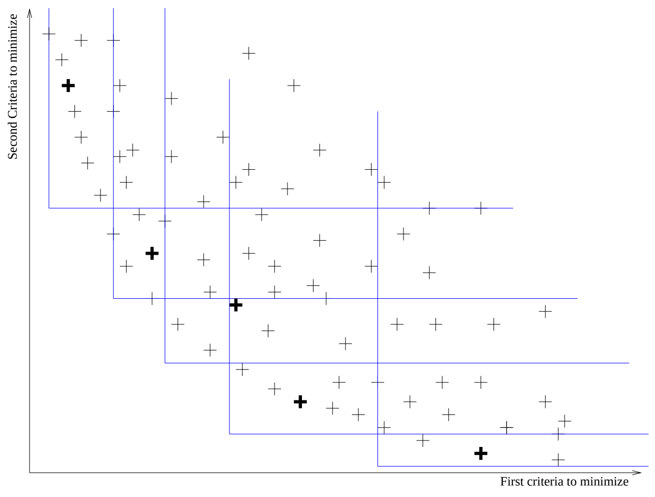

Figure 1: Approximating the pareto curve. An example with two criteria to minimize. The set of the solution is partitioned.Crosses are possible solutions, bold crosses are approximations of partitions.

Algorithm 3 Pareto approximation algorithm for independent unitary task

```
Input: $\epsilon \in [0, +\infty[$
Compute $M_{\text{opt}}$ using Algorithm 1.
Compute $M_{max}$ using Proposition 1.
$S \leftarrow \emptyset$
for i=1:$\lceil log_{1+\epsilon} \frac{M_{max}}{M_{\text{opt}}} \rceil$
    let $S_i$ be the result of algorithm 2 with $\alpha = (1+\epsilon)^i$
    $S \leftarrow S \cup S_i$
return $S$
```

Proof of Theorem 3.
Let σ be a Pareto-optimal schedule. Then, there exists k such as $(1+\epsilon)^k M_{\text{opt}} \leq C_{\max}(\sigma) \leq (1+\epsilon)^{k+1} M_{\text{opt}}$. We show that S_{k+1} is an $(1+\epsilon, 1)$-approximation of σ.

- Reliability. The reliability of a Pareto-optimal schedule increases with the makespan. Thus, $p_{\text{succ}}(S_{k+1}) \geq p_{\text{succ}}(\sigma)$.
- Makespan. $C_{\max}(S_{k+1}) \leq (1+\epsilon)^{k+1} M_{\text{opt}}$ and $C_{\max}(\sigma) \geq (1+\epsilon)^k M_{\text{opt}}$. Thus, $C_{\max}(\sigma) \leq (1+\epsilon) C_{\max}(S_{k+1})$.

■

5. NON-UNITARY INDEPENDENT TASKS

We extend the above problem to the case where tasks are not unitary ($\forall v_i \in V,\ o_i \in \mathbb{N}^*$). As before, we fix the makespan objective and try to find the best possible reliability. However, since the problem of finding if there exists a schedule whose makespan is smaller than a target value given a set of processors and non-unitary independent tasks is NP-complete, it is not possible to find an optimal schedule unless *P=NP*.

In Proposition 1, we have proved that the most reliable schedule is obtained by assigning tasks to a processor that has the smallest $\lambda\tau$ ({*failure rate*} times {*unitary instruction execution time*}). We can improve this proposition for the independent tasks case.

THEOREM 4. *Let's schedule a set of independent tasks without exceeding a makespan of a given value M on m processors. Then, the best possible (but not necessary achievable) reliability among all the schedules of makespan at most M is obtained when tasks are mapped to $\tilde{m} \leq m$ processors in increasing order of $\lambda_i\tau_i$. The $\tilde{m}-1$ first processors execute tasks up to the date M ($C_i = M$) and the last processor executes the remaining tasks ($C_{\tilde{m}} \leq M$).*

Proof of Theorem 4.
Let $S = (T_1, \ldots, T_m)$ be the schedule described in the proposition: T_j is the set of tasks assigned to processor j by S, (for $j > \tilde{m}+1, T_j = \emptyset$) . We need to show that $\sum_{j=1}^{m} \left(\tau_j \lambda_j \sum_{v_i \in T_j} o_i \right)$ is minimum. Let us call $\omega_j = \sum_{v_i \in T_j} o_i$ and $\lambda_j \tau_j = \gamma_i$. For any other valid allocation $S' = (T_1^{'}, \ldots, T_m^{'})$, let us call $\omega_j^{'} = \sum_{v_i \in T_j^{'}} o_i$.

We have two cases, $\omega_{\tilde{m}}^{'} \geq \omega_{\tilde{m}}$ or $\omega_{\tilde{m}}^{'} \leq \omega_{\tilde{m}}$. Let us first consider the case $\omega_{\tilde{m}}^{'} \leq \omega_{\tilde{m}}$. This means that the duration of the tasks of the $\tilde{m}$ processors have not decreased with the new schedule S'. In this case, passing from schedule S to schedule S' implies that:

$$\forall j \in [1, \tilde{m}]), \omega_j^{'} \leq \omega_j \tag{1}$$

and

$$\forall j \in [\tilde{m}+1, m], \omega_j^{'} \geq \omega_j \tag{2}$$

Moreover, since we schedule the same tasks in S and S', we have:

$$\sum_{j=1}^{\tilde{m}} \omega_j - \omega_j^{'} = 0 \tag{3}$$

Equations (1), (2) and (3) imply that:

$$\sum_{j=1}^{\tilde{m}-1} \omega_j - \omega_j^{'} = - \sum_{j=\tilde{m}}^{m} \omega_j - \omega_j^{'} \tag{4}$$

Let us compute the difference between the exponent of the probability of success of S and S'

$$\begin{aligned} X &= n_1\lambda_1\tau_1 + \ldots + n_i\lambda_i\tau_i + \ldots + n_N\lambda_N\tau_N \\ &\quad - n_1'\lambda_1\tau_1 - \ldots - n_i'\lambda_i\tau_i - \ldots + n_N'\lambda_N\tau_N \\ &= \lambda_1\tau_1(n_1 - n_1') + \lambda_i\tau_i(n_i - n_i') \\ &= k\lambda_1\tau_1 - k\lambda_i\tau_i \\ &= k(\lambda_1\tau_1 - \lambda_i\tau_i) \\ &\leq 0 \text{ because } \lambda_i\tau_i \geq \lambda_1\tau_1. \end{aligned}$$

Hence, S has a greater reliability than S'. Since the case where $\omega_{\tilde{m}}^{'} \geq \omega_{\tilde{m}}$ is handled similarly, the theorem is proved.

■

This theorem indicates that, among the schedules that do not exceed a given makespan objective M, the most reliable schedule is obtained when scheduling tasks to a subset of processors. This subset is such that the $\lambda\tau$ values of these processors are the smallest ones.

Note that such a schedule is not always achievable. It might require more than $\tilde{m}$ processors to schedule the tasks or it might not be possible to schedule the tasks with a makespan M.

What highlights this theorem is that if one wishes to use less than the total number of available processors (say k) one can still achieve the optimal reliability if he or she chooses machines with smallest $\lambda\tau$ products.

In order to let the user choose a better trade-off, we propose that the user gives the number of processors he wishes to use, say k. Then we compute a schedule (using any heuristic that targets makespan optimization) of the tasks on the k processors that have the smallest $\lambda\tau$ product. The idea is that the less processors one uses the larger the makespan, but thanks to Theorem 4, the better the reliability. For instance, if $k = 1$ (the user chooses to use only one processor), then any reasonable scheduling heuristic will map the tasks sequentially on the processor with the smallest $\lambda\tau$ and from proposition 1 this will lead to an optimal schedule in terms of reliability. Conversely, if $k = m$ (the user chooses to use all the processors), a scheduling heuristic that targets makespan minimization will use all the processors and lead to a schedule with a small makespan and a high relaibility.

6. GENERAL CASE

In this section,we study the general case where no restriction is given to the task graph. Based on the characterization of the role of the $\lambda\tau$ value ($\{$*failure rate*$\}$ times $\{$*unitary instruction execution time*$\}$ product), we propose a simple way to extend scheduling heuristics that targets the makespan optimization to bi-criteria heuristics.

6.1 Generalizing Scheduling Heuristics : the HEFT case

In [8], Topcuoglu et al. proposed the HEFT (Heterogeneous Earliest Finish Time) heuristic to schedule a task graph on a set of heterogeneous processors. HEFT works as follows. At each step, it considers all the ready tasks and simulates the mapping of each of those tasks on all the processors. Then it chooses the tasks that finishes the earliest and maps it to the processor on which it finishes the earliest.

We propose to change HEFT into RHEFT (Reliable-HEFT) by emphasizing the importance of the product $\lambda\tau$. Let us call ${T_{\text{end}}}_j^i$ the finish time[3] of task i on processor j. For all the ready tasks i and all the processors j, instead of choosing the task for which ${T_{\text{end}}}_j^i$ is minimum (HEFT case), we choose the tasks for which ${T_{\text{end}}}_j^i\lambda_j$ is minimum and allocate them on processor j.

To illustrate the difference between HEFT and RHEFT, consider the example of Figure 2. In this example we have two processors with a unitary processing time τ of respectively 1 and 2 and failure rate λ of respectively 2 and 1. Suppose that when the independent task i is schedulable, processor 1 is ready at time 4 and processor 2 is ready at time 5. Hence we have:

- ${T_{\text{end}}}_1^i = 6$, ${T_{\text{end}}}_1^i \times \lambda_1 = 12$

[3]${T_{\text{end}}}_j^i$ is computed by adding the duration of task i on processor j ($o_i \times \tau_j$) to the maximum of the end time of the all predecessors of i say i' plus the communication time between j and j' ($l_{j',j}$) and the ready time of processor j.

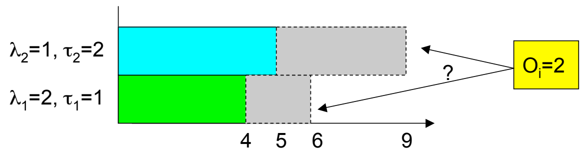

Figure 2: Illustration of the difference between HEFT (that will map task i on processor 1) and RHEFT (that will map task i on processor 2)
.

- ${T_{\text{end}}}_2^i = 9$, ${T_{\text{end}}}_2^i \times \lambda_2 = 9$

This means that HEFT will map task i on processor 1 and RHEFT will map it on processor 2.

It is important to note that if a task i' is submitted again after i has been mapped on processor 2, with the same characteristic ($o_{i'=2}$) then RHEFT will map it on processor 1 because
${T_{\text{end}}}_1^{i'} \times \lambda_1 = 6 \times 2 = 12$ and ${T_{\text{end}}}_2^{i'} \times \lambda_2 = 13 \times 1 = 13$.

6.2 Toward a better trade-off

RHEFT tends to favor processors with high reliability but have a fairly good speed. It may happen that the user wants to choose various trade-offs between speed and reliability. In this case we can generalize the approach presented in section 5. Again, one idea here is to limit the usage to k processors (k given by the user) among the m available.

We have experimentally tested several ways to sort processors in order to choose k out of the m available. As suggested by Theorem 4, experiments have shown that we have to choose the k processors that have the smallest $\lambda\tau$ product first. For instance, in Figures 3, 4 and 5, we have tested the HEFT and RHEFT on the Strassen task graph with more than 4800 tasks (corresponding to the matrix multiply of 1000 blocks by 1000 blocks). We have generated randomly a set of 100 machines where λ is chosen uniformly in $[10^{-2}, 10^{-3}]$ and τ is chosen uniformly in $[10^{-5}, 10^{-7}]$. These numbers are not very realistic but provide comprehensive results that are easy to read on the graphs. Experiments made with different values lead to similar results. In each of these figures, we plot the makespan (y axis on the left) and the reliability (y-axis on the right) when we use between one and 100 processors (x-axis). In Fig. 3 (resp. Fig. 4, Fig. 5), when we use less than 100 processors we choose the most reliable (resp. the fastest, the ones with the smallest $\lambda\tau$ value) first.

Results show that when we order the processor by reliability, we get a very bad makespan when we use a small number of processors, and the reliability is also low. We can improve the makespan by ordering processors by speed but, in this case, the reliability increases when we add some processors (it reaches 0.8 for 15 processors). When we order the processors by increasing $\lambda\tau$, we notice that the reliability slowly decreases from the optimal one (with one processor), to 0.6 with 100 processor. In this case, the makespan is a little smaller than the one obtained when we order the processors by speed. But the ratio of the two makespans is not very large. Hence, here again, we see that we can provide a good way to choose a trade-off between makespan and reliability if we order the processor by increasing value of the $\lambda\tau$ product and use as few as possible processors to meet the makespan goal.

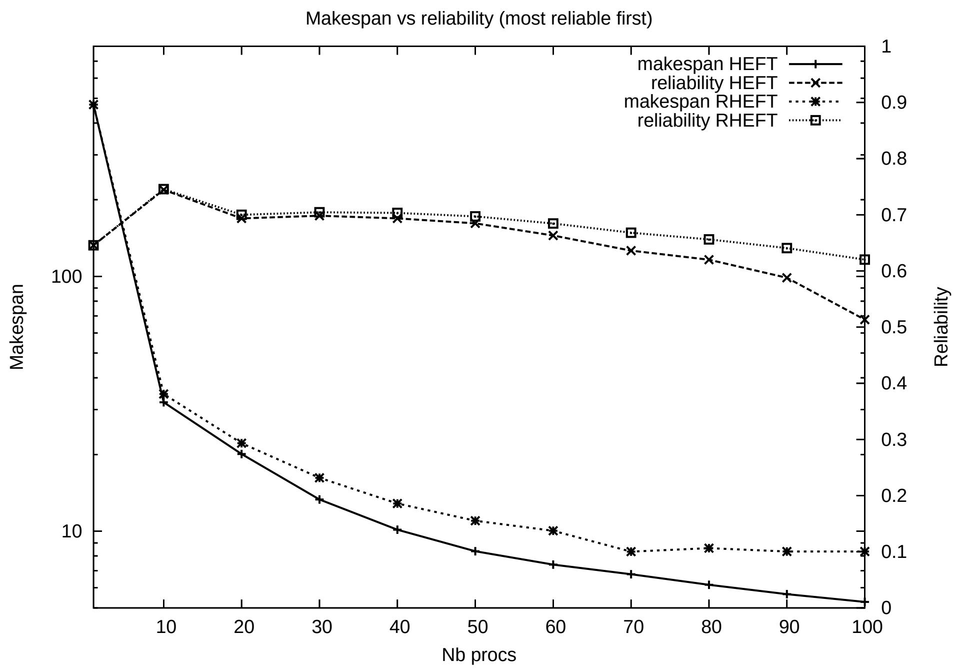

Figure 3: Ordering the processors: most reliable first.

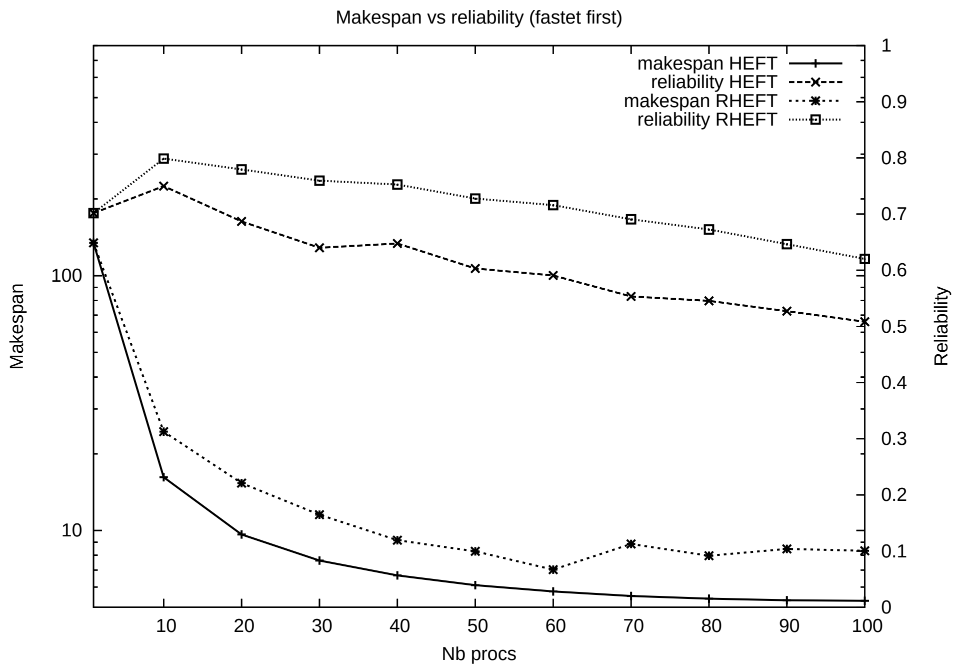

Figure 4: Ordering the processors: fastest first.

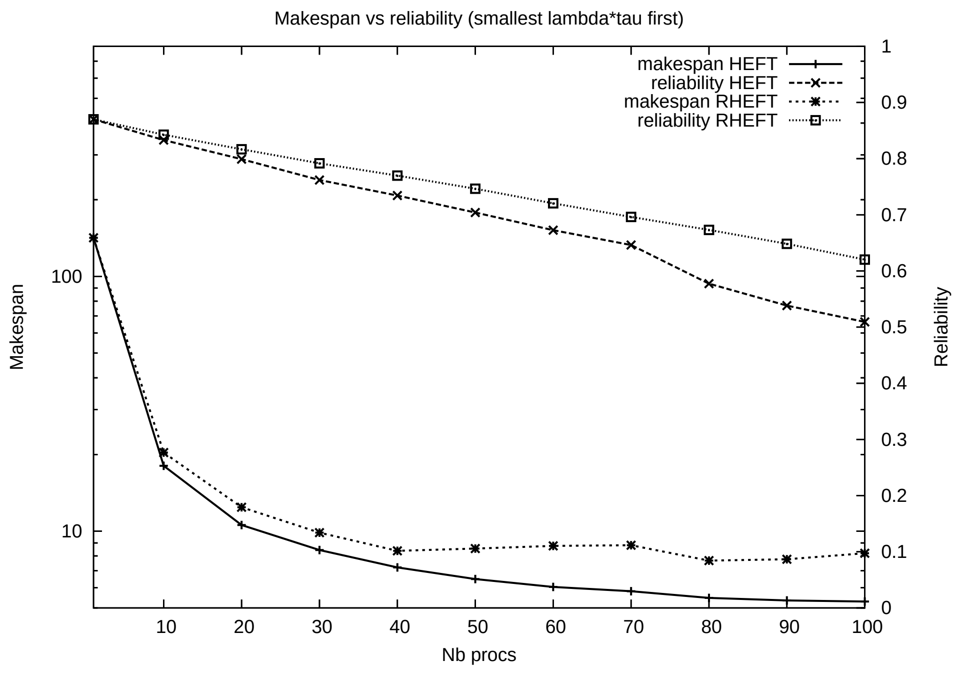

Figure 5: Ordering the processors: smallest $\lambda\tau$ first.

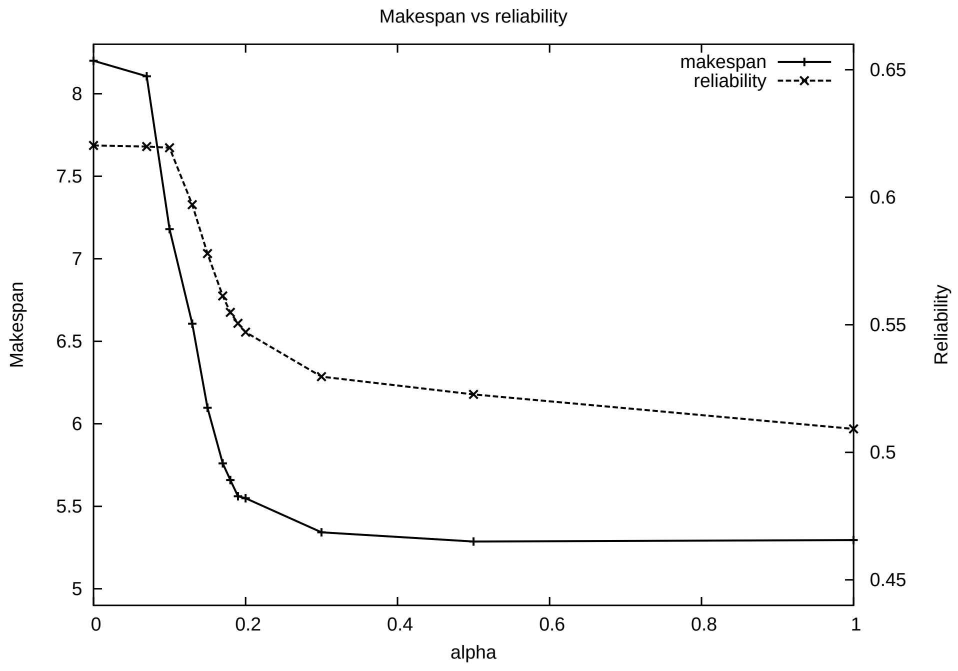

Figure 6: Impact of the compromise variable on 100 processors ($\alpha = 0 \Rightarrow$ RHEFT; $\alpha = 1 \Rightarrow$ HEFT).

In each figure we have also plotted the reliability and the makespan given by the HEFT heuristic when using between one and 100 processors. We find that when we use one processor we get the same value of the makespan and the reliability, while the difference increases up to 100 processors where $M_{\mathrm{HEFT}} = 5.28$ when $M_{\mathrm{RHEFT}} = 8.33$ and $R_{\mathrm{HEFT}} = 0.51$ when $R_{\mathrm{RHEFT}} = 0.63$.

If a user wants RHEFT to have a behavior closer to HEFT, we can modify the RHEFT heuristic with a trade-off variable α, such that when $\alpha = 1$ we switch to HEFT and when $\alpha = 0$ we switch to RHEFT, and for any value α between 0 and 1 the heuristic builds an intermediate solution. In Figure 6, we show how the makespan and the reliability change when α varies from 0 to 1. Here we have expended α by putting $\alpha = e^{1-1/\alpha}$ when $\alpha \neq 0$ for visualizing the influance of small value of the trade-off variable.

We see that by setting the α trade-off variable to a value between 0 and 1, the user can choose a heuristic with a behavior between the HEFT and RHEFT.

6.3 Extending to other heuristics

It is easy to extend other scheduling heuristics for heterogeneous system to take reliability into account. It appears to be straight forward for heuristics such as for the BIL [12], PCT [11], GDL [14] or CPOP [8]. For instance, in CPOP it requires to change the notion of critical path by multiplying the duration of the tasks by the failure rate (λ) of the processor on which it is mapped.

Here, also, we can use less than all the available processors to increase the reliability and decrease the makespan. We can also use the trade-off variable to choose a different compromise between the original heuristic and the reliability-enabled one.

7. CONCLUSION

In this paper, we have studied the problem of scheduling tasks graph on heterogeneous platforms. We have tackled two metrics: reliability and makespan. As these two objectives are unrelated and sometimes contradictory, we need to search for bi-criteria approximation algorithms.

In previous work, some heuristics have been proposed to solve this problem [8, 4, 2]. However, none of them discuss the fundamental properties of a good bi-criteria scheduling algorithm. In this paper, we have tackled important subproblems in order to see how to efficiently solve this problem.

We have shown that minimizing the reliability is a polynomial problem and we have shown that optimizing both the makespan and the reliability is not approximable.

For the case where we schedule independent unitary tasks, we have proposed an approximation algorithm that finds, among the schedules that do not exceed a given makespan, the one with the best reliability. For the case of independent non-unitary tasks and uniform processors, we have highlighted the role of the $\{$*failure rate*$\} \times \{$*unitary instruction execution time*$\}$ ($\lambda\tau$). Based on the importance of this product, we are able to let the user choose a trade-off between makespan and reliability by scheduling the tasks on a given subset of the processors (the ones with the smallest $\lambda\tau$ value). For general task graphs, we have shown that it is easy to extend most of the heuristics designed for optimizing the makespan by taking into account the reliability. Here again, based on our experiments, we have shown the importance of the $\lambda\tau$ product. This helps the user to choose a trade-off by selecting a subset of the available processors. The strategy is the following: schedule tasks on your fastest and most reliable processors and use as few processors as possible to meet the makespan goal.

If required, we also propose adjusting the compromise between the original heuristic and the reliability-enabled one.

All these results are based on the fact that, for each resource, the product $\lambda\tau$ gives the best compromise between speed and reliability.

8. REFERENCES

[1] T. Angskun, G. Fagg, J. Bosilca, G.and Pjesivac-Grbovic, and J. Dongarra. Scalable Fault Tolerant Protocol for Parallel Runtime Environments. In *Euro PVM/MPI*, Bonn, Germany, 2006.

[2] Vincent Boudet. Heterogeneous task scheduling : a survey. Technical Report RR2001-07, ENS Lyon, February 2001.

[3] Aurélien Bouteiller, Thomas Herault, Géraud Krawezik, Pierre Lemarinier, and Franck Cappello. Mpich-v: a multiprotocol fault tolerant mpi. *International Journal of High Performance Computing and Applications*, 2005.

[4] B. Cirou and E. Jeannot. Triplet: a Clustering Scheduling Algorithm for Heterogeneous Systems. In *IEEE ICPP International Workshop on Metacomputing Systems and Applications (MSA'2001)*, Valencia, Spain, September 2001.

[5] Atakan Dogan and Fusun Ozgüner. Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):308–323, 2002.

[6] Atakan Dogan and Fusun Ozgüner. Bi-objective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems. *Comput. J.*, 48(3):300–314, 2005.

[7] Mourad Hakem and Frank Butelle. A Bi-objective Algortithm for Scheduling Parallel Applications on Heterogeneous Systems Subject to Failures. In *Renpar 17*, Canet en Roussillon, October 2006.

[8] Salim Hariri Haluk Topcuoglu and Min-You Wu. Task scheduling algorithms for heterogeneous processors. *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 3–14, April 1999.

[9] Arnaud Legrand and Yves Robert. *Algorithlmique Parallèle.* Dunod, 2005.

[10] Joseph Y-T. Leung, editor. *Handbook of Scheduling. Algorithms, Models and Performance Analysis.* Chapman & Hall/CRC, 2004.

[11] Muthucumaru Maheswaran and Howard Jay Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Heterogeneous Computing Workshop*, pages 57–69, 1998.

[12] Hyunok Oh and Soonhoi Ha. A static scheduling heuristic for heterogeneous processors. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *Euro-Par, Vol. II*, volume 1124 of *Lecture Notes in Computer Science*, pages 573–577. Springer, 1996.

[13] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In Danielle C. Young, editor, *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 86–92, Los Alamitos, California, November 12–14 2000. IEEE Computer Society.

[14] Gilbert Sih and Edward Lee. A compile-time scheduling heuristic for interconnection-constrained heterogenous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, PDS-4(2), February 1993.