Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms

O. Beaumont
LaBRI, UMR CNRS 5800
Bordeaux, France
Olivier.Beaumont@labri.fr

A. Legrand and L. Marchal and Y. Robert
LIP, UMR CNRS-INRIA 5668
ENS Lyon, France
{Arnaud.Legrand,Loris.Marchal,Yves.Robert}@ens-lyon.fr

***Abstract*— In this paper, we consider steady-state scheduling techniques for mapping a collection of task graphs onto heterogeneous systems, such as clusters and grids. We advocate the use of steady-state scheduling to solve this difficult problem. Due to space limitations, we concentrate on complexity results. We show that the problem of optimizing the steady-state throughput is NP-Complete in the general case. We formulate a compact version of the problem that belongs to the NP complexity class but which does not restrict the optimality of the solution.**

We provide many positive results in the extended version [5]. Indeed, we show how to determine in polynomial time the best steady-state scheduling strategy for a large class of application graphs and for an arbitrary platform graphs, using a linear programming approach.

I. Introduction

The traditional objective of scheduling algorithms is makespan minimization: given a task graph and a set of computing resources, find a mapping of the tasks onto the processors, and order the execution of the tasks so that: (i) task precedence constraints are satisfied; (ii) resource constraints are satisfied; and (iii) a minimum schedule length is provided. However, makespan minimization turned out to be NP-hard in most practical situations [24], [1]. The advent of more heterogeneous architectural platforms is likely to even increase the computational complexity of the process of mapping applications to machines.

An idea to circumvent the difficulty of makespan minimization is to lower the ambition of the scheduling objective. Instead of aiming at the absolute minimization of the execution time, why not consider asymptotic optimality? After all, the number of tasks to be executed on the computing platform is expected to be very large: otherwise why deploy the corresponding application on computational grids? To state this informally: if there is a nice (meaning, polynomial) way to derive, say, a schedule whose length is two hours and three minutes, as opposed to an optimal schedule that would run for only two hours, we would be satisfied.

This approach has been pioneered by Bertsimas and Gamarnik [8]. Steady-state scheduling allows to relax the scheduling problem in many ways. Initialization and clean-up phases are neglected. The initial integer formulation is replaced by a continuous or rational formulation. The precise ordering and allocation of tasks and messages are not required, at least in the first step. The main idea is to characterize the activity of each resource during each time-unit: which (rational) fraction of time is spent computing, which is spent receiving or sending to which neighbor? Such activity variables are gathered into a linear program, which includes conservation laws that characterize the global behavior of the system.

In this paper, we consider the execution of a complex application on heterogeneous computing platforms. The complex application consists of a suite of identical, independent problems to be solved. In turn, each problem consists of a set of tasks, modeled by an *application graph*. There are dependences (precedence constraints) between these tasks. A typical example is the repeated execution of the same algorithm on several distinct data samples (see the simple application graph depicted in Figure 1(a)). We use another graph, the *platform graph*, for the grid platform. We model a collection of heterogeneous resources and the communication links between them as the nodes and edges of an undirected graph. See the example in Figure 1(b) with four processors and five communication links. Each node is a computing resource (a processor, or a cluster, or even a router with no computing capabilities) capable of computing and/or communicating with its neighbors at (possibly) different rates. The underlying interconnection network may be very complex and, in particular, may include multiple paths and cycles (just as the Internet does).

Deriving a steady-state solution for this complex mapping problem amounts to characterize the usage of processors and communication links: for a given processor, which fraction of time is spent executing which task type? for a given communication link, which fraction of time is spent communicating which file type? The objective

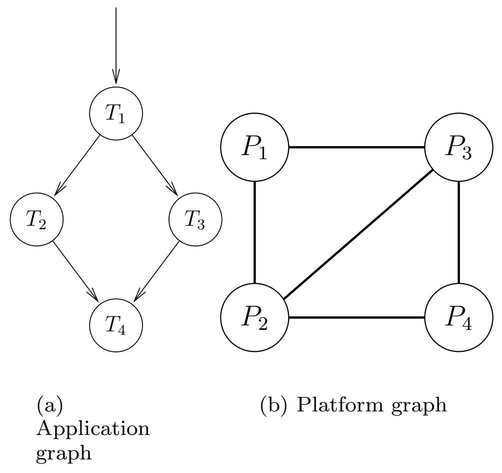

Fig. 1. The application and platform graphs

is to maximize the throughput, which is the number of problem instances solved per time-unit, i.e. the number of copies of the application graph which are processed per time-unit. Of course, the previous activity fractions and the throughput are rational numbers. To derive the actual periodic schedule there will remain to scale everything, so as to derive an integer time-period. But the beauty of steady-state scheduling is that this reconstruction can be automatically computed from the rational values (Theorem 1). We can derive a periodic schedule and express it in compact form, contrarily to the traditional makespan minimization approach which would require a scheduling date for all tasks and files.

The objective of the paper is to assess the limits of steady-state scheduling when applied to the difficult mapping problem that we just described. When is this approach asymptotically optimal? What is the inherent complexity of computing the optimal steady-state throughput? The contribution of the paper is a complexity result assessing that the most general instance of the problem is NP-Complete (Section III). Showing that the problem does belong to the class NP (i.e. that a solution can be verified in polynomial time) already is a challenging problem. Nevertheless, the optimal steady-state throughput can be computed in polynomial time for most practical instances, and the corresponding actual schedule is asymptotically optimal.

The rest of the paper is organized as follows. In Section II, we introduce our base model of computation and communication, and we formally state the steady-state scheduling problem to be solved. Then, in Section III, we prove that the most general instance of the problem is NP-Complete. We describe related work in Section IV. We give some final remarks and conclusions in Section V. Due to space limitations, many details and proofs are omitted: please refer to the extended version of the paper [5].

II. Models

A. Constraints on platform and application graphs

The application is a suite of problem instances, each instance being modeled by the same *application graph*: let $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \dots, \mathcal{P}^{(N)}$ be the N problems to solve, where N is large. Each problem $\mathcal{P}^{(m)}$ corresponds to a copy $G_A^{(m)} = (V_A^{(m)}, E_A^{(m)})$ of the application graph $G_A = (V_A, E_A)$. The number $|V_A|$ of nodes in G_A is the number of task types. In the example of Figure 1(a), there are four task types, denoted as T_1, T_2, T_3 and T_4. Overall, there are $N.|V_A|$ tasks to process, since there are N copies of each task type.

The target heterogeneous platform is represented by a directed graph, the *platform graph* $G_P = (V_P, E_P)$. There are $p = |V_P|$ nodes $P_1, P_2, \dots, P_p$ in V_P that represent the processors. In the example of Figure 1(b) there are four processors, hence $p = 4$. See below for processor speeds and execution times. Each edge represents a physical interconnection. Each edge $e_{ij} \in E_P : P_i \to P_j$ is labeled by a value $c_{i,j}$ which represents the time to transfer a message of unit length between P_i and P_j, in either direction: we assume that the link between P_i and P_j is bidirectional and symmetric. We assume a *full overlap, single-port* operation mode, where a processor node can simultaneously receive data from one of its neighbor, perform some (independent) computation, and send data to one of its neighbor. At any given time-step, there are at most two communications involving a given processor, one in emission and the other in reception. Other models have been considered in [4], [2].

In our model, processor P_i requires $w_{i,k}$ time units to process a task of type T_k. Note that this framework is quite general, because each processor has a different speed for each task type, and these speeds are not related: they are *inconsistent* with the terminology of [11]. At last, each edge $e_{k,l} : T_k \to T_l$ in the task graph is weighted by a communication cost $data_{k,l}$ that depends on the tasks T_k and T_l. It corresponds to the amount of data output by T_k and required as input to T_l. Recall that the time needed to transfer a unit amount of data from processor P_i to processor P_j is $c_{i,j}$. Thus, if a task $T_k^{(m)}$ is processed on P_i and task $T_l^{(m)}$ is processed on P_j, the time to transfer the data from P_i to P_j is equal to $data_{k,l} \times c_{i,j}$; this holds for any edge $e_{k,l} : T_k \to T_l$ in the task graph and for any processor pair P_i and P_j. Again, once a communication from P_i to P_j is initiated, P_i (resp. P_j) cannot handle a new emission (resp. reception) during the next $data_{k,l} \times c_{i,j}$ time-units.

B. Allocations and cyclic schedules

We need the following definitions: allocation, schedule, makespan, cyclic schedule, K-periodic schedule, throughput. Because of lack of space, some of the definitions will be omitted here. We refer to the extended version [5] for a rigorous introduction to cyclic scheduling.

Definition 1 (Allocation): An allocation $\mathcal{A}$ is a pair of mappings $\pi : V_A \mapsto V_P$ and $\sigma : E_A \mapsto \{\text{paths in } G_P\}$ such that for each edge $e_{k,l} : T_k \to T_l$:

$$\sigma(e_{k,l}) = (P_{i_1}, P_{i_2}, \ldots, P_{i_p}) \quad \textit{with} \begin{cases} P_{i_1} = \pi(T_k), P_{i_p} = \pi(T_l) \text{ et} \\ (P_{i_j} \to P_{i_{j+1}}) \in E_P \text{ for all } j \in [\![1, p-1]\!] \end{cases} .$$

Obviously, π maps the tasks of the application graph G_A onto the platform nodes and σ maps the (files associated to the) edges of G_A onto the paths of the platform network. Thus $\mathcal{A}$ represents a possible way to execute the application graph on the platform graph. In what follows, we will consider a set of weighted allocations $(\mathcal{A}_m, \alpha_m)$. We define the weight α_m of allocation $\mathcal{A}_m$ as the fractional number of application graphs processed according to allocation $\mathcal{A}_m$ during one time unit.

Definition 2 (Schedule): A schedule associated to an allocation (π, σ) is a pair of mappings $t_\pi : V_A \mapsto \mathbb{Q}$ and $t_\sigma : E_A \times E_P \mapsto \mathbb{Q}$ satisfying to the following constraints:

- precedence constraints for the application graph
- resource constraints for the platform graph (computations, communications and 1-port constraints)

Theorem 1: Given a set of r weighted allocations $(\mathcal{A}_m, \alpha_m)$, such that the superposition of the r weighted allocations satisfies all resource constraints, then we can reconstruct a valid periodic schedule of throughput $\rho = \sum_{m=1}^{r} \alpha_m$.

The significance of Theorem 1 is that it allows to go from purely local constraints to a global steady-state schedule. The formal proof of this result is technical, and the detailed proof is available in [5]. The proof is divided in two parts. First, we prove that we can build a valid schedule of communications from a set of allocations satisfying 1-port constraints, using a weighted decomposition of bipartite graphs [22, vol.A chapter 20]. Then, we prove how to build a periodic schedule achieving the throughput $\rho = \sum_{m=1}^{r} \alpha_m$.

III. Complexity results

In this section, we derive some complexity results for the problem of maximizing the throughput when mapping an application graph $G_A = (V_A, E_A)$ onto a given platform graph $G_P = (V_P, E_P)$. In particular, we prove that finding the set of allocations that maximizes the throughput is NP-Complete in general. Nevertheless, it is proven in [5] that this problem can be solved in polynomial time for large classes of graph, in particular if the dependency depth of the application graph is bounded (which covers many important cases in practice, such as independent tasks, tree-shaped application graphs, series of fork-join application graphs, etc.).

In order to prove the NP-Completeness, we first define a restricted version of the problem, whose solutions can be verified in polynomial time. We will show the NP-Completeness of the restricted version. Before that, we show that we do not lose anything by sticking to the restricted version: if the general problem admits a solution of given throughput, so does the restricted version. Altogether, these results fully demonstrate the difficulty of the general problem. Because of lack of space, we cannot provide full proofs in this short version (see [5]), but we provide the main ideas. The target decision problems can be stated as follows:

Definition 3 (GRAPH-THROUGHPUT(G_A, G_P, ρ)): Given a platform graph G_P, an application graph G_A and a rational bound for the throughput ρ, does there exist a periodic schedule whose throughput is at least ρ?

However, we need a version where the solution can be verified in polynomial time:

Definition 4: (COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ)) Given a platform graph G_P, an application graph G_A and a rational bound for the throughput ρ, does there exist a periodic schedule consisting of at most $k \leqslant 3|V_P|$ allocations $\mathcal{A}_1, \ldots, \mathcal{A}_k$, where the weight α_i is the average number of graphs processed by the allocation $\mathcal{A}_i$ within one time unit, $\alpha_i = \frac{a_i}{b_i}$, and a_i and b_i are integers such that

$$\forall i, \quad \log a_i + \log b_i \leqslant 6|V_P|(2 + \log(|V_P|) + \log(M)),$$

where $M = \max(1, |V_A| \max w_{i,k}, |V_P||E_A| \max c_{i,j} \max \mathit{data}_{k,l})$

and such that the throughput is at least $\rho = \sum \alpha_i \geqslant \rho$?

In the latter definition, we restrict the search to solutions where a bounded number ($k \leqslant 3|V_P|$) of allocations is used, whose weights can be expressed in a compact way ($\alpha_i = \frac{a_i}{b_i}$, where a_i and b_i are integers such that $\log a_i + \log b_i \leqslant 6|V_P|(\log(|V_P|) + \log(M))$. This restriction is necessary in order to keep the problem in the class NP, since an optimal solution may have a size exponential in the size of the initial data: indeed, from any periodic solution with period T, we can trivially build another solution, achieving the same throughput, with period $r \cdot T$, for any integer r. However, the following theorem asserts that this restriction on the size of the solution does not affect the optimal throughput:

Theorem 2: Given a weighted platform graph G_P and a task graph G_A if there exists a periodic schedule to GRAPH-THROUGHPUT that achieves a throughput ρ, then there also exists a solution of COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ).

Proof: In order to prove this result, we first derive a set of constraints that will be satisfied by any periodic solution periodic to the GRAPH-THROUGHPUT problem. Let us denote by $\mathcal{A}$ the set of all possible allocations. There may be an exponential number of such allocations (with respect to the size of the application and platform graphs), but the number of allocations is nevertheless finite, since it consists to associate a given processor to any task, and a given path in the platform graph (of size at most $|V_P|$ since cycles are clearly useless) to any dependence in the task graph. A solution of the GRAPH-THROUGHPUT problem is then a set of weighted allocations $\{(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_r, \alpha_r)\}$: here the weight α_m is the number of times per time-unit where allocation $\mathcal{A}_m$ is used by the schedule. Let us denote by $\pi(k,m)$ the index of the processor that processes task T_k in the allocation $\mathcal{A}_m$, and by $\Sigma(k,l,m)$ the set of oriented links used to send data from $\pi(i,m)$ to $\pi(j,m)$ in the allocation $\mathcal{A}_m$. Then, any solution of the GRAPH-THROUGHPUT problem satisfies the following set of constraints:

$$\begin{cases} (1,i) \quad \forall P_i, \quad \displaystyle\sum_{\mathcal{A}_m} \alpha_m \sum_{k, \pi(k,m)=i} w_{i,k} \leqslant 1 \\ (2,i) \quad \forall P_i, \\ \displaystyle\sum_{\mathcal{A}_m} \alpha_m \sum_{P_i \to P_j} \sum_{(T_k,T_l) \in E_A, \ (P_i,P_j) \in \Sigma(k,l,m)} c_{i,j}\, data_{k,l} \leqslant 1 \\ (3,i) \quad \forall P_i, \\ \displaystyle\sum_{\mathcal{A}_m} \alpha_m \sum_{P_j \to P_i} \sum_{(T_k,T_l) \in E_A, \ (P_j,P_i) \in \Sigma(k,l,m)} c_{j,i}\, data_{k,l} \leqslant 1 \\ (4,m) \quad \forall \mathcal{A}_m, \quad \alpha_m \geqslant 0 \end{cases}$$

Indeed, for any solution to the GRAPH-THROUGHPUT problem, the processing capability of each processor cannot be exceeded (constraint $(1,i)$); one-port constraints for sending $(2,i)$ and receiving $(3,i)$ messages must be fulfilled at any node. Conversely, from any solution of previous set of inequalities, one can derive a valid schedule, where $\sum \alpha_m$ messages are processed every time-unit (this is exactly Theorem 1). Thus, the solution of the linear program where we aim at maximizing $\sum_m \alpha_m$ under above constraints provides an optimal solution of the GRAPH-THROUGHPUT problem. Let us denote by $\rho_{\max}$ the optimal value of the objective function. The previous linear program is of little practical interest since both the number of constraints and the number of variables are possibly exponential in the size of the original instance of the GRAPH-THROUGHPUT problem. Nevertheless, using linear programming theory [21], it is possible to prove that one of the optimal solution to the linear program is one instance of COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ).

Indeed, the linear program has $|\mathcal{A}| + 3|V_P|$ constraints, where $|\mathcal{A}|$ is the number of all allocations. There is a vertex V of the polyhedron defined by linear constraints which is optimal, and V is given by the solution of a $|\mathcal{A}| \times |\mathcal{A}|$ linear system, such that at vertex V, at least $|\mathcal{A}|$ inequalities among $|\mathcal{A}| + 3|V_P|$ are tight. Since only $3|V_P|$ constraints are not of the form $(4,m)$, we know that at least $|\mathcal{A}| - 3|V_P|$ constraints of the form $(4,m)$ are tight, i.e. that at most $3|V_P|$ allocations have a non-zero weight. Thus, there exists an optimal solution where at most $3|V_P|$ allocations are actually used.

In order to achieve the proof of the theorem, we need to bound the size of the weights of these allocations. Again, consider the optimal solution defined by vertex V, which is given by the solution of a $|\mathcal{A}| \times |\mathcal{A}|$ linear system, where at most $m \leqslant 3|V_P|$ constraints are not of the form $\alpha_i = 0$. Let us consider the $m \times m$ linear system containing non-trivial equations. We can easily prove that the coefficients of the resulting linear system are bounded by $M = \max(1, |V_A| \max w_{i,k}, |V_P||E_A| \max c_{i,j} \max data_{k,l})$ and, using Cramer's rule, that $\log(a_i) + \log(b_i) \leqslant 6|V_P|(2 + \log(|V_P|) + \log(M))$, and thus that the sizes of both a_i and b_i satisfy the constraints of the instance of COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ). Thus, among the optimal solutions of the GRAPH-THROUGHPUT problem, there exists a solution to COMPACT-WEIGHTED-GRAPH-THROUGHPUT$(G_A, G_P, \rho_{\max})$. ■

Theorem 3: COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ) is NP-Complete.

Lemma 1: COMPACT-WEIGHTED-GRAPH-THROUGHPUT$(G_A, G_P, \rho) \in$ NP

Proof: In order to prove that COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ) $\in$ NP, we use the set of allocations as a certificate. We know that there the solution consists in at most $3|V_P|$ allocations, whose weights $\alpha_i = \frac{a_i}{b_i}$ satisfy $\log a_i + \log b_i \leqslant 6|V_P|(2 + \log(|V_P|) + \log(M))$, where

$$M = \max(1, |V_A| \max w_{i,k}, |V_P||E_A| \max c_{i,j} \max data_{k,l}).$$

Theorem 1 asserts that it is possible to build a valid schedule of communications and task processing, that achieves the throughput $\sum_m \alpha_m$, using a weighted decomposition of the bipartite graph. The weights on the edges of the bipartite graph represent the overall communication time between the P_i and P_j. In order to prove that COMPACT-WEIGHTED-GRAPH-THROUGHPUT$(G_A, G_P, \rho) \in$ NP, we only need to prove that both the bipartite graph and the processing times can be encoded in size polynomial to the size $\mathcal{S}$ of the original instance, which can be done since both the number of allocations and their weights are bounded by construction. ■

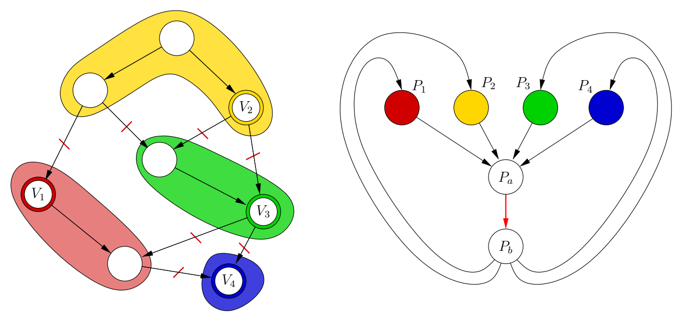

Fig. 2. Reduction from the instance of MINIMUM-MULTIWAY-CUT: the weight of the cut is equal to the communication volume.

Lemma 2: COMPACT-WEIGHTED-THROUGHPUT(G_A, G_P, ρ) is complete.

Proof: In order to prove that COMPACT-WEIGHTED-THROUGHPUT(G_A, G_P, ρ) is complete, we use a reduction from MINIMUM-MULTIWAY-CUT, which is NP-Complete (and even APX-Complete) [1]. MINIMUM-MULTIWAY-CUT is the following decision problem:

Definition 5 (MINIMUM-MULTIWAY-CUT(G_M, B)): Given a weighted graph $G_M = (V_M, E_M)$, a set $S \subset V_M$ of terminals, a weight function t on the edges and a rational bound B, is there a multiway cut, i.e. a set $E'_M \subset E_M$ such that the removal of E'_M from E_M disconnects each terminal from all the others, and $\sum_{e \in E'_M} t(e) \leqslant B$?

Consider the following instance of COMPACT-WEIGHTED-THROUGHPUT(G_A, G_P, ρ), built from an instance of MINIMUM-MULTIWAY-CUT(G_M, B). First, the application graph is built as shown on Figure 2). It has the same number of vertices and the same number of edges as G_M. Each (non-oriented) edge (V_k, V_l) (of weight $t(V_k, V_l)$) in G_M is transformed in an (oriented) edge $(T_{\min(k,l)}, T_{max(k,l)})$ (of weight $data_{k,l} = t(V_k, V_l)$) in G_A. The resulting graph is clearly oriented and acyclic, thus representing a valid application graph. Then, the platform graph is built as follows. It consists of $|S| + 2$ processors $P_1, \ldots, P_{|S|}, P_a, P_b$. The communication times of the edges of the platform graph depicted in Figure 2 are given by $\forall i, \quad c_{P_i,P_a} = 0$, $c_{P_b,P_i} = 0$ and $c_{P_a,P_b} = 1$, all the other communication times being $+\infty$. The times to process the tasks of G_A on the processors of G_P are the following. If $V_k \in S$, then we will refer T_k as a "terminal task". Terminal task T_k is associated to terminal processor P_k, so that $w_{k,k} = 0$ and $w_{i,k} = +\infty$ if $i \neq k$. All the other tasks are not associated to a particular processor and can be processed in time 0 whatever the processor P_i executing it. Finally, P_a and P_b are unable to process any task ($w_{a,k} = w_{b,k} = +\infty$). Finally, we set $\rho = \frac{1}{B}$.

Let us first suppose that there is a solution to the original instance of MINIMUM-MULTIWAY-CUT(G_M, B), and let $\mathcal{C}_i$ denote the set of nodes connected to the terminal $V_i \in S$ in the graph $G_M = (V_M, E_M \setminus E'_M)$. Then, consider the following allocation (see Figure 2):

$$\forall T_k \in \mathcal{C}_i, T_k \text{ is done on } P_i, \quad \forall (T_k, T_l) \in \mathcal{C}_i \times \mathcal{C}_j, \quad i \neq j,$$
$$\sigma(k,l) = \{(P_i, P_a), (P_a, P_b), (P_b, P_j)\}. \quad (1)$$

In this allocation, only the communication time between P_a and P_b is not 0. We can easily see that this communication time is equal to the weight of Multiway Cut in G_M. Thus, by Theorem 1, the platform G_P is able to process one application graph G_A every B time units using this allocation.

Suppose now that we have a solution to the instance of COMPACT-WEIGHTED-GRAPH-THROUGHPUT(G_A, G_P, ρ) that we have built, i.e. a collection of weighted allocations $(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_m, \alpha_m)$ such that the platform G_P is able to process $\sum_m \alpha_m \geqslant \frac{1}{B}$ application graphs G_A every time unit. For every allocation, the fraction of time spent by any processor (P_i, P_a or P_b) is necessarily 0 since otherwise, the processing time would be infinite. The fraction of time spent by P_i or P_b sending data, and the fraction of time spent by P_i and P_a receiving data is 0 by construction. For every terminal processor P_i let $\mathcal{C}_i$ be the set of tasks in G_A processed on P_i. Clearly, $T_i \in \mathcal{C}_i$ (otherwise the overall processing time would be infinite). Thus, it can be proven that every allocation induces a multiway cut in G_M. Using resource constraints, we can prove that at least one of the weights of the allocations is less than B, thus providing a solution to MINIMUM-MULTIWAY-CUT(G_M, B). This achieves the proof of the NP-Completeness of COMPACT-WEIGHTED-THROUGHPUT(G_A, G_P, ρ). ■

IV. Related problems

We classify several related papers along the following three main categories. The interested reader will find more references in [5].

a) Scheduling task graphs on heterogeneous platforms.: Several heuristics have been introduced to schedule (acyclic) task graphs on different-speed processors, see [20], [28] among others. Unfortunately, all these heuristics assume no restriction on the communication resources, which renders them somewhat unrealistic to model real-life applications. Recent papers [14], [16], [25] suggest to take communication contention into account. Among these extensions, scheduling heuristics under the one-port model [17], [18] are considered in [3]: just as in this paper, each processor can communicate with at most another processor at a given time-step.

b) Collective communications on heterogeneous platforms.: Several papers deal with the complexity of collective communications on heterogeneous platforms: broadcast and multicast operations are addressed in [10], [19], gather operations are studied in [12]. Broadcasting and multicasting on heterogeneous platforms have been studied under different models, in the context of heterogeneous target platforms. Asymptotically optimal algorithms have been derived for series of broadcasts [7] on an heterogeneous platform, under the communication model presented in Section II. On the other hand, it has been proved in [6] that under the same communication model, optimizing the throughput of a series of multicasts in NP-Hard.

c) Master-slave on the computational grid.: Master-slave scheduling on the grid can be based on a network-flow approach [23] or on an adaptive strategy [13]. Note that the network-flow approach of [23] is possible only when using a full multiple-port model, where the number of simultaneous communications for a given node is not bounded. This approach has also been studied in [15]. In [27], Taura and Chien prove that finding the best allocation (when restricting to a single allocation, i.e. when mapping all instances of a given task type onto the same processor) is NP-Complete in the strong sense.

V. Conclusion

In this paper, we have dealt with the implementation of mixed task and data parallelism onto heterogeneous platforms. Due to space limitations, we have mainly concentrated on complexity results. We have shown that the problem of optimizing the steady-state throughput is NP-Complete in the general case. We have been able to formulate a compact version of the problem that belongs to the NP complexity class but which does not restrict the optimality of the solution.

We provide many positive results in the extended version [5]. Indeed, we show how to determine in polynomial time the best steady-state scheduling strategy for a large class of application graphs and for a arbitrary platform graphs, using a linear programming approach. In particular, we provide positive results for all task graphs whose dependency depth is bounded (what includes atomic and tree shaped task graphs, among others).

This work can be extended in the following two directions. On the theoretical side, we could try to solve the problem of maximizing the number of tasks that can be executed within a time period K. This scheduling problem is more complicated than the search for the best steady-state, but a smaller time period limits memory requirements and may be necessary in order to derive more dynamic schedules, where the allocations may change according to changes in platform capabilities. On the practical side, we need to run actual experiments rather than simulations. Indeed, it would be interesting to capture actual architecture and application parameters, and to compare heuristics on a real-life problem suite, such as those in [9], [26].

References

[1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, Berlin, Germany, 1999.

[2] C. Banino, O. Beaumont, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor grids. In *PARA'02: International Conference on Applied Parallel Computing*, LNCS 2367, pages 423–432. Springer Verlag, 2002.

[3] O. Beaumont, V. Boudet, and Y. Robert. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In *HCW'2002, the 11th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[4] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium (IPDPS'2002)*. IEEE Computer Society Press, 2002.

[5] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogenous platforms. Research Report RR-2004-20, LIP, ENS Lyon, France, April 2004. Available at http://www.ens-lyon.fr/LIP/.

[6] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Complexity results and heuristics for pipelined multicast operations on heterogeneous platforms. Research Report RR-2004-07, LIP, ENS Lyon, France, January 2004.

[7] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining broadcasts on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2004*. IEEE Computer Society Press, 2004.

[8] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[9] Michael D. Beynon, Tahsin Kurc, Alan Sussman, and Joel Saltz. Optimizing execution of component-based applications using group instances. *Future Generation Computer Systems*, 18(4):435–448, 2002.

[10] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Efficient collective communication in distributed heterogeneous systems. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*. IEEE Computer Society Press, 1999.

[11] T. D. Braun, H. J. Siegel, and N. Beck. Optimal use of mixed task and data parallelism for pipelined computations. *J. Parallel and Distributed Computing*, 61:810–837, 2001.

[12] J.-I. Hatta and S. Shibusawa. Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. In *2000 International Conference on Parallel Processing (ICPP'2000)*. IEEE Computer Society Press, 2000.

[13] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In R. Buyya and M. Baker, editors, *Grid Computing - GRID 2000*, pages 214–227. Springer-Verlag LNCS 1971, 2000.

[14] L. Hollermann, T. S. Hsu, D. R. Lopez, and K. Vertanen. Scheduling problems in a practical allocation model. *J. Combinatorial Optimization*, 1(2):129–149, 1997.

[15] B. Hong and V.K. Prasanna. Bandwidth-aware resource allocation for heterogeneous computing systems to maximize throughput. In *Proceedings of the 32th International Conference on Parallel Processing (ICPP'2003)*. IEEE Computer Society Press, 2003.

[16] T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce. Task allocation on a network of processors. *IEEE Trans. Computers*, 49(12):1339–1353, 2000.

[17] S. L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, 1989.

[18] D. W. Krumme, G. Cybenko, and K. N. Venkataraman. Gossiping in minimal time. *SIAM J. Computing*, 21:111–139, 1992.

[19] R. Libeskind-Hadas, J. R. K. Hartline, P. Boothe, G. Rae, and J. Swisher. On multicast algorithms for heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 61(11):1665–1679, 2001.

[20] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Seventh Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1998.

[21] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.

[22] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, 2003.

[23] G. Shao. *Adaptive scheduling of master/worker applications on distributed computational resources*. PhD thesis, Dept. of Computer Science, University Of California at San Diego, 2001.

[24] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

[25] O. Sinnen and L. Sousa. Comparison of contention-aware list scheduling heuristics for cluster computing. In T. M. Pinkston, editor, *Workshop for Scheduling and Resource Management for Cluster Computing (ICPP'01)*, pages 382–387. IEEE Computer Society Press, 2001.

[26] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the grid. In *2002 ACM/IEEE Supercomputing Conference*. ACM Press, 2002.

[27] K. Taura and A. A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115. IEEE Computer Society Press, 2000.

[28] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Eighth Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1999.