Independent and Divisible Tasks Scheduling on Heterogeneous Star-shaped Platforms with Limited Memory*

O. Beaumont[1], A. Legrand[2], L. Marchal[2], and Y. Robert[3]

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

2: ID, UMR CNRS-INRIA-IMAG 5132, Grenoble, France
{Arnaud.Legrand}@imag.fr

3: LIP, UMR CNRS-INRIA 5668, ENS Lyon, France
{Loris.Marchal,Yves.Robert}@ens-lyon.fr

Abstract

In this paper, we consider the problem of allocating and scheduling a collection of independent, equal-sized tasks on heterogeneous star-shaped platforms. We also address the same problem for divisible tasks. For both cases, we take memory constraints into account. We prove strong NP-completeness results for different objective functions, namely makespan minimization and throughput maximization, on simple star-shaped platforms. We propose an approximation algorithm based on the unconstrained version (with unlimited memory) of the problem. We introduce several heuristics, which are evaluated and compared through extensive simulations. An unexpected conclusion drawn from these experiments is that classical scheduling heuristics that try to greedily minimize the completion time of each task are outperformed by the simple heuristic that consists in assigning the task to the available processor that has the smallest communication time, regardless of computation power (hence a "bandwidth-centric" distribution).

Keywords: *Scheduling, makespan, steady-state, divisible load, memory constraints, bounded buffers, memory limitation.*

1 Introduction

In this paper, we consider the problem of allocating and scheduling a collection of independent, equal-sized tasks on heterogeneous computing platforms. Master-slave tasking on such platforms has received a lot of attention recently [19, 16, 15, 20].

The minimization of the total execution time is a NP-hard problem, even if the platform is a simple tree [13]. In contrast, the optimal steady-state throughput, i.e. the maximal number of tasks that can be processed per time-unit, can be computed in polynomial time, using rational linear programming [1]. Moreover, when the overlay network of computing nodes is tree-shaped, the optimal throughput can be characterized by a set of recursive equations, which are solved via a bottom-up traversal of the tree [3]. From these equations, it is possible to derive a *local* allocation of tasks to resources. This best allocation is *bandwidth-centric*: if enough bandwidth is available at the node, then all children must be kept busy all the time; if bandwidth is limited, then tasks should be allocated only to children which have sufficiently fast communication links, in order of fastest communication time. Counter-intuitively, the maximum throughput in the tree is achieved by delegating tasks to children as quickly as possible, and not by seeking their fastest resolution. The bandwidth-centric strategy is asymptotically optimal, and enjoys the key advantage that the optimal allocation can be computed locally. This enables each component to make autonomous, local scheduling decisions. The approach is thus more scalable than a fully centralized approach.

Nevertheless, the bandwidth-centric periodic solution presented in [3, 1] may well require a huge amount of memory. Indeed, the length of the period may be very large, while the number of buffers required on each resource is proportional to the length of this period. This drawback may prevent the use of the bandwidth-centric protocol in practical situations. In this paper, we take memory constraints into account, and we aim at deriving efficient scheduling strategies for scheduling independent tasks when computing resources have a limited number of buffers.

As expected, the problem becomes more difficult with memory constraints. We first target the makespan minimization problem (which is polynomial [5] on simple star-shaped platforms) in Section 2 and show that it is a strongly NP-complete problem. We also derive an approximation algorithm based on the unconstrained (without memory limitations) version of this problem and introduce several heuristics, which are evaluated and com-

pared through extensive simulations in Section 3. Then we prove in Section 4 that finding the minimum number of buffers required to reach the optimal steady-state throughput is a strongly NP-complete problem. We also address the counterpart of the previous scheduling problem for divisible tasks (a perfect parallel job that can be arbitrarily split into several independent parts) and prove its strong NP-completeness. Last, in Section 5, we survey related work on scheduling under memory constraints and we state some final remarks and conclude the paper in Section 6.

2 Scheduling a finite number of independant tasks under memory constraints

In this section, we introduce the first of our four different problems. We seek to minimize the processing time of a finite number of independant equal-sized tasks on an heterogeneous platform.

2.1 Framework and complexity results

Consider the star-shaped platform depicted in Figure 1. The master processor P_0 initially holds all the identical tasks $\{T_1, T_2, \ldots, T_N\}$. There are p slave processors, numbered from P_1 to P_p. The time needed to send a task from P_0 to P_i is given by c_i. The time necessary for P_i to process a task is given by w_i. We assume that the communication medium is exclusive: the master can only communicate with a single slave at each time-step. We also assume the possibility of overlapping computations with independent computations. More precisely, a slave P_i can simultaneously execute a task whose data was received in one of its buffers, and receive the data for another task in another buffer, provided that is has enough buffers to do so. Throughout the paper, this star-shaped platform and its operating model will be referred to as the *reference* platform.

Definition 1 (UNBOUNDED-MAKESPAN $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, \{T_1, T_2, \ldots, T_N\}), K$). *Let $K > 0$ be a time-bound, and consider the reference platform. Is it possible to process all the N tasks within K time units on this platform?*

A $O(N^2p^2)$ algorithm that solves the UNBOUNDED-MAKESPAN problem is described in [5]. The memory-constrained version can be defined as follows:

Definition 2 (BOUNDED-MAKESPAN $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, \{T_1, T_2, \ldots, T_N\}, K)$). *Let $K > 0$ be a time-bound. Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is it possible to process all the N tasks within K time units on this platform?*

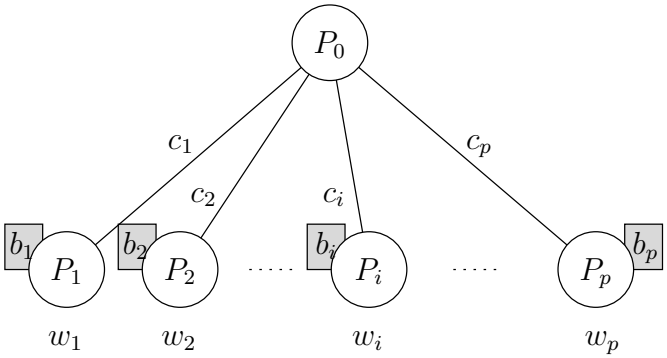

Figure 1. Star Platform

This problem is NP-complete by reduction 3-Partition, which is NP-Complete in the strong sense [14].

Definition 3 (3-Partition). *Given $3n$ integers $a_1, \ldots, a_{3n}$, such that $\sum_1^{3n} a_i = nB$ and $\forall i,\ \frac{B}{4} < a_i < \frac{B}{2}$. Is there a partition of the a_i's into n groups of 3 integers, such that each a_i belongs exactly to one group, and each group sums to B.*

Theorem 1. *BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N) is NP-Complete in the strong sense.*

Proof. We have to polynomially transform the instance of 3-Partition into an instance of BOUNDED-MAKESPAN which has a solution if and only if the original instance of 3-Partition has a solution.

Let us consider the following instance of BOUNDED-MAKESPAN, consisting of a master processor P_0, $3n$ slave processors $P_1, \ldots, P_{3n}$ with the following characteristics.

- $b_i = 1$, i.e. processor P_i cannot start receiving a new task until it has processed the task it holds
- $c_i = a_i, w_i = 2nB$, i.e. it takes a_i time units to P_i to receive a task from P_0 and $2nB$ time units to process it,

and one processor P_B with $b_B = 1, c_B = B, w_B = B$. Moreover let us set $N = 5n$ and $K = 4nB$.

$\boxed{\Rightarrow}$ Let us first suppose that there is a solution to the original instance of 3-Partition and let us suppose, without loss of generality, that

$$\forall 0 \leqslant j \leqslant n-1, \quad a_{3j+1} + a_{3j+2} + a_{3j+3} = B.$$

Then, the schedule depicted in Figure 2 provides a solution to BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N).

$\boxed{\Leftarrow}$ Let us now suppose that there is a solution to the instance of BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N) we have built.

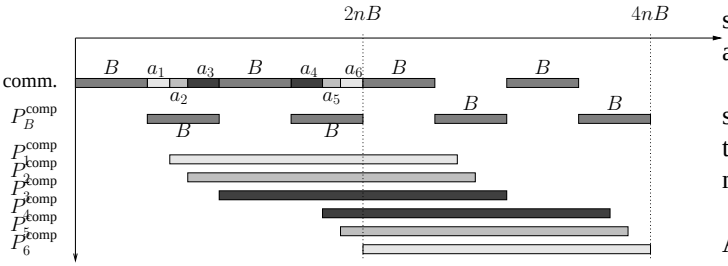

Figure 2. Solution to BOUNDED-MAKESPAN

Lemma 1. *In a solution of* BOUNDED-MAKESPAN (b_i, c_i, w_i, K, N)*, each processor* P_i *processes exactly 1 task, and* P_B *processes exactly* $2n$ *tasks.*

Proof. Let us first consider the case of P_i. It cannot receive its first task before time step a_i, and thus, it cannot finish its processing before time step a_i+2nB. Therefore, it cannot receive its second task before time step $2a_i + 2nB$, and thus, it cannot process a second task within the time bound $K = 4nB$. Thus, each P_i processes at most one task.
Clearly, for the same reasons, P_B cannot process more than $2n$ tasks within $4nB$ time units. Therefore, each P_i cannot process more than one task, and P_B cannot process more than $2n$ tasks, such that, since $N = 5n$, in any optimal solution, each P_i processes exactly one task, and P_B processes exactly $2n$ tasks. ■

Using Lemma 1 and as communications and processing cannot overlap, we can prove that communications to P_B are necessarily organized as depicted in Figure 2. Moreover, since it takes $2nB$ time units to P_i to process one task, all communications to the P_i's must be finished before time step $2nB$. Therefore, those communications must take place in the n disjoint holes of size B left free by the communications to P_B, thus providing a solution to the original instance of 3-Partition. ■

2.2 Approximation Algorithms

In this section, we give an approximation algorithm in presence of limited memory for star graphs when we aim at minimizing the makespan. The approximation algorithm we propose is designed for processors that are able to hold only one task (and thus where tasks have to be distributed one by one and where communications and processing cannot be overlapped), which is the most restrictive case in presence of limited buffer. Therefore, the approximation ratio holds true a fortiori for larger buffers.

The approximation algorithm we propose is a list based scheduling algorithm, whose makespan is not larger than twice the optimal makespan on the platform where all memory constraints have been removed.

The sketch of the list algorithm is depicted in Figure 3. At any time, IdleProc$[i]$ is the next smallest date when processor P_i becomes idle (and thus the smallest date when a task can be sent to P_i since the algorithm makes use of only one buffer), NbTasksSent is the overall number of tasks already sent by P_0, and NbTasksProc$[i]$ is the overall number of tasks already sent to P_i, and NbCommEvent denotes the date of the next communication event. The algorithm we propose requires some pre-processing. We need to know the number n_i of tasks that are sent to P_i in the optimal solution without memory limitation. The n_i's are given by the solution of UNBOUNDED-MAKESPAN, which is described in [5]. In the algorithm described in Figure 3, a task is sent to P_i as soon as

- the communication medium is free
- P_i is idle
- P_i has not processed yet the number of tasks allocated to it in the optimal solution without limited memory.

```
STAR-BOUNDED-BUFFER(c_i, w_i, N)
 1: (n_1, ..., n_p) =STAR-UNBOUNDED-BUFFER(c_i, w_i, N);
 2: NbTasksSent=0;
 3: NextCommEvent=0;
 4: ∀i,  IdleProc[i]=0; NbTasksProc[i]=0;
 5: while NbTasksSent ⩽ N do
 6:    Find P_i, such that IdleProc[i] is minimal and
       NbTasksProc[i] ⩽ n_i
 7:    if IdleProc[i] ⩽ NextCommEvent then
 8:       NbTasksProc[i] + +; NbTasksSent++;
 9:       IdleProc[i]=NextCommEvent+c_i + w_i;
10:       NextCommEvent=NextCommEvent+c_i;
11:    else
12:       NexCommEvent=IdleProc[i];
```

Figure 3. List scheduling approximation algorithm

Surprisingly, this very simple heuristic builds a schedule whose makespan cannot be larger than twice the makespan of the optimal schedule without memory limitations:

Theorem 2. *Let us denote by* T_{alg} *the makespan of the schedule built with limited buffers by*

STAR-BOUNDED-BUFFER*(c_i, w_i, N) (defined in Figure 3), and by T_{opt} the makespan of the (optimal) schedule built with unlimited buffers by* STAR-UNBOUNDED-BUFFER*(c_i, w_i, N) (defined in [5]), then*

$$T_{alg} \leqslant 2T_{opt}.$$

Proof. The proof of this theorem is adapted from Graham's proof for list based scheduling [10]. Let us consider the schedule built by STAR-BOUNDED-BUFFER(c_i, w_i, N) and let us denote by $[t_1, t_1+\alpha_1], \ldots, [t_k, t_k+\alpha_k], [t_l, T_{\text{alg}}]$ the intervals when the communication medium is idle. Clearly $t_l = \sum_1^k \alpha_i + \sum_1^p c_i n_i$ since at each time step, either the communication medium is idle or a task is being sent, and the overall number of tasks sent to P_i is n_i by construction. Let us also denote by P_{last} the processor that finishes its processing at time T_{alg} in the schedule built by STAR-BOUNDED-BUFFER(c_i, w_i, N). The situation is depicted in Figure 4.

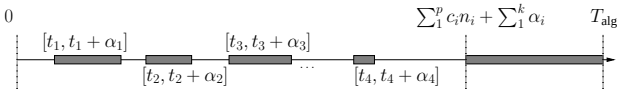

Figure 4. Schedule built by Star-Bounded-Buffer

Let us consider the case of P_{last}. An idle time in the communication medium is generated by STAR-BOUNDED-BUFFER if and only if all the processors that have not processed all their tasks yet are working. Thus, since P_{last} processes the last task, it has been working at least during all the time intervals $[t_1, t_1 + \alpha_1], \ldots, [t_k, t_k + \alpha_k], [t_l, T_{\text{alg}}]$, of overall size $T_{\text{alg}} - \sum_{i=1}^p c_i n_i$. Therefore, the overall processing time of P_{last} is given by $w_{\text{last}} n_{\text{last}}$, so that

$$T_{\text{alg}} - \sum_{i=1}^{p} c_i n_i \leqslant w_{\text{last}} n_{\text{last}} \text{ and thus } T_{\text{alg}} \leqslant \sum_{i=1}^{p} c_i n_i + w_{\text{last}} n_{\text{last}}.$$

Moreover, $T_{\text{opt}} \geqslant \sum_{i=1}^p c_i n_i$ since $\sum_{i=1}^p c_i n_i$ represents the time necessary to send all the tasks to the different slaves in the optimal solution (remember that the numbers of tasks sent to P_i by STAR-BOUNDED-BUFFER and STAR-UNBOUNDED-BUFFER are the same), and $T_{\text{opt}} \geqslant w_{\text{last}} n_{\text{last}}$ since $w_{\text{last}} n_{\text{last}}$ represents the overall processing time on slave P_{last} (again, either with STAR-BOUNDED-BUFFER or STAR-UNBOUNDED-BUFFER). Therefore,

$$T_{\text{alg}} \leqslant 2T_{\text{opt}}. \quad \blacksquare$$

3 Simulation

3.1 Heuristics

In this section, we present several heuristics for the model with independent, equal-sized tasks. All the heuristics are list-based heuristics, and schedule a task as soon as possible. Only the selection function differs from one heuristic to the other. Thus, this selection function plays a key role in this scheme: it selects the next target processor (the one that will execute the next task) among all the different processors that are available at a given time step, as soon as the communication medium becomes free. In the following, we present different selection functions.

A natural idea for choosing among available processors, is to select the one with the highest computing speed, or the one with the smallest communication cost. These selection functions lead to the heuristics denoted by min_w and min_c in the following. It is also possible to choose the processor which will finish to process the task the earliest, given previous scheduling decisions. The heuristic based on this selection function is denoted by mct in the following.

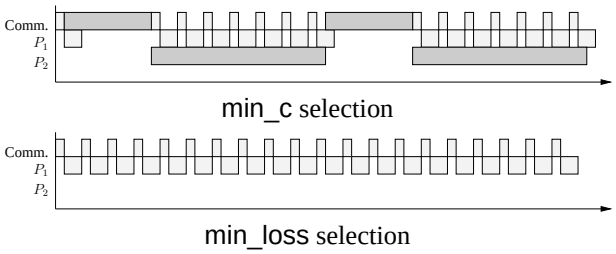

Figure 5. Simple instance with two processors ($b_1 = b_2 = 1$, $c_1 = 1$, $c_2 = 10$, $w_1 = 2$, $w_2 = 20$). Light (resp. dark) grey represent communications from the master to P_1 (resp. P_1) or computations by P_1 (resp. P_2). min_c may perform a very bad choice. Leaving the communication medium idle for a while may lead to much better results.

Nevertheless, all these list-based heuristics may lead to very bad choices like the one depicted on Figure 5. Therefore, we also propose an heuristic based on the evaluation of the gain and loss of a decision to schedule a task a processor. This kind of approach is very close to the commonly used *sufferage* heuristic [9, 18] and may avoid such misleading choices. Assume that we decide, at a given time step t when the communication medium is free, to schedule the next communication as a transfer to P_i. We earn one task, but it is possible that another processor P_j becomes starving between t and the end of the communication to P_i (Available$[i] + c_i$). We can compute the number of tasks that could have been performed by P_j during

this interval in such a case and sum this number over the P_j to get the average penalty incurred by the selection of P_i. This leads to the heuristic min_loss based selecting the P_i that minimize the average penalty and by sending it a task as soon as it gets ready This heuristic may not communicate a task as soon as possible and wait for a better available slave instead, so it is not a "real" list heuristic.

3.2 Simulation platforms

The platform consists of a master and several slaves. The different parameters to take into account are the following:

Number of slaves We performed the experiments with a small number of processors (5) and with a larger number (20). As the results are similar, we only present them in the latter case, where heterogeneity is more likely to play a part.

Sending and computing speeds These parameters were chosen randomly with a Gaussian distribution in the interval [50,150].

Number of tasks As we simulate the scheduling of a fixed number of tasks on a platform, we have to choose the number of tasks to be scheduled. We let this number vary from 10 to 2,048. Note that a small number of tasks is more significant for makespan minimization, while a large number of tasks is relevant for throughput optimization.

Size of the buffers We perform experiments for buffer sizes going from 1 (no overlap between communication and computation) to 32 (almost no limitation on memory).

3.3 Simulation results

To compare the performance of the different heuristics, we compute their performance ratio, defined as the ratio of the number of tasks processed by the slaves over the total time spent to process these tasks. We cannot compute the optimal performance ratio in the case of bounded buffers, but we normalize results by plotting the performance ratio of the heuristics over the optimal performance ratio in the absence of memory limitation [3, 1].

Figure 6 presents the results in the case of a single buffer, for a varying number of tasks. Figure 7 shows the results for a fixed number of tasks (2,048), for a varying size of buffer.

Most scheduling heuristics try to greedily minimize the completion time of each task. Even if some variants exist to cope with task affinity or misleading greedy decisions (like *sufferage*), none of these heuristics is efficient in the situation where communications from the master are the

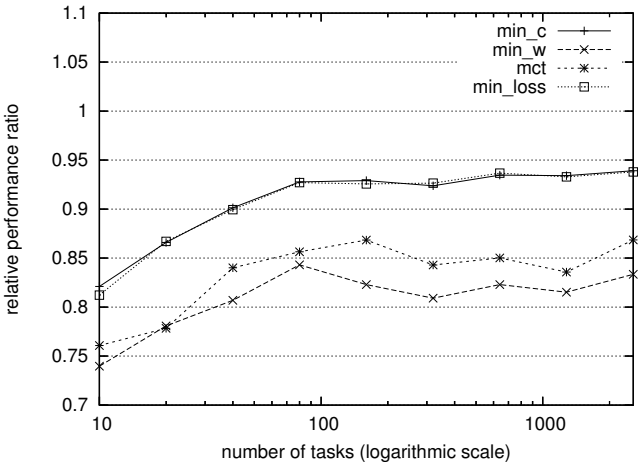

Figure 6. Performance for a single buffer

bottleneck. Surprisingly, the simplest heuristic (min_c) outperforms the more involved ones (like min_loss and mct), and achieves very good results in all situations: min_c always has the best performances when trying to minimize the makespan in the single buffer case; it reaches 90% of the optimal throughput in the single buffer case, and more than 99% of this bound when the size of the buffer is greater than 2.

Another surprising conclusion is that min_c reaches the optimal unbounded throughput with only a few buffers. The good performances of min_c can be explained as follows: if we send a task to a processor P_i with a small c_i and a big w_i, the communication medium will be busy during a short time, and P_i spends a lot of time processing the task; we are able to perform many other communications during this computation. Conversely, if we send a task to a processor with a small c_i and a small w_i, this processor is likely to process the task quickly and to be chosen again soon as a future target: this leads to a larger share of the communication medium for P_i but since it has a small w_i, it contributes to a big fraction of the total throughput of the platform. In conclusion, sending a task to a slave processor with a small c_i is never a bad choice, regardless of its computing power.

4 Relaxed optimization problems

4.1 Throughput maximization under memory constraints

When the platform model is more complicated than a star (e.g. a tree or a general graph), the makespan minimization problem turns out to be very difficult [13]. A nice idea to circumvent this difficulty is to relax the objective function by maximizing the steady-state throughput. This

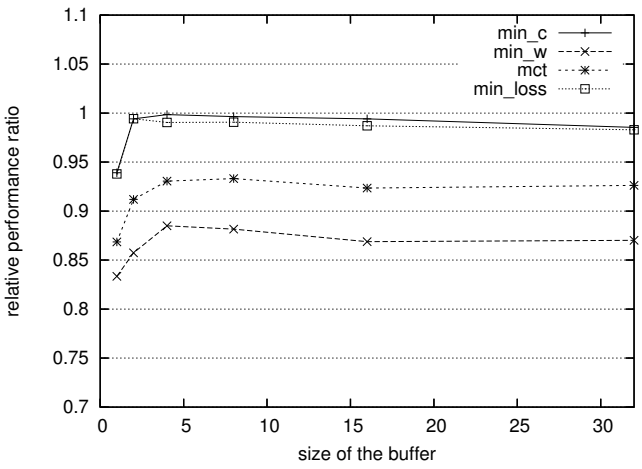

Figure 7. Performance as a function of the buffer size.

problem is polynomial and leads to asymptotically optimal solutions for the makespan minimization problem. On a star platform, finding the optimal steady-state throughput, i.e. the optimal number of tasks that can be processed per time-unit, can be formalized as follows:

Definition 4 (UNBOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, \rho)$). *Let $K > 0$ be a time-bound, and consider the reference platform. Is there a K-periodic schedule of period T, i.e. a schedule that executes K tasks every T time-units in steady state, and such that $\frac{K}{T} \geqslant \rho$?*

Using a linear-programming formulation, this problem can be solved by a $O(p\log(p))$ algorithm [3, 1]. The bounded version of the throughput problem can be defined as follows:

Definition 5 (BOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, \rho)$). *Let $K > 0$ be a time-bound. Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is there a K-periodic schedule of period T, i.e. a schedule that executes K tasks every T time-units in steady state, and such that $\frac{K}{T} \geqslant \rho$?*

The formulation of the UNBOUNDED-THROUGHPUT and BOUNDED-THROUGHPUT problems is questionable, because as stated these problems may not belong to NP. Indeed, the size of K and T could be exponential in the size of the problem instance. For the UNBOUNDED-THROUGHPUT problem, it turns out the polynomial-time algorithm given in [3, 1] does provide a solution where K and T have a polynomial size. For the BOUNDED-THROUGHPUT problem, the size of K has no reason to be polynomial in the size of the original instance. Therefore, we need to define the following "compact" version of the problem:

Definition 6 (COMPACT-BOUNDED-THROUGHPUT $((c_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, S, \rho)$). *Let $K > 0$ be a time-bound. Consider the reference platform, and assume that each slave processor P_i is equipped with a bounded buffer that can hold at most b_i tasks. Is there a K-periodic schedule of period T, i.e. a schedule that executes K tasks every T time-units in steady state, and such that $K \leqslant \log S$ and $\frac{K}{T} \geqslant \rho$?*

COMPACT-BOUNDED-THROUGHPUT belongs to NP but is more constrained than BOUNDED-THROUGHPUT. We omit the proofs for brievety (it can be found in [4] since they are very similar to the proof presented in Section 2.1) but both problems are strongly NP-complete. Hence the difficulty is intrinsically due to the memory limitation, and not to the statement of the problem.

4.2 Divisible load scheduling under memory constraints

A divisible task corresponds to a perfect parallel job that can be arbitrarily split into several independent parts. In the simplest variant, computation and communication times for a given chunk are assumed to grow linearly with the chunk size. However, this is not realistic for communications, and recent papers have added a start-up overhead in the model, to take link latency into account. In this paper we also focus on this affine cost model: it takes $w_i X$ time-units to execute X units of load on worker P_i, and $G_i + X.g_i$ time-units to send X units of load from the master processor P_0 to P_i. Note that in the case of independent tasks, the latency is directly incorporated in the value c_i (since the size X of a task is fixed beforehand). Two algorithmic techniques have been proposed to schedule divisible loads, namely one-round and multi-round algorithms:

- In a one-round distribution, each processor is used at most once. Therefore, the first problem is to select a subset of workers and to determine in which order the chunks should be sent to the different workers, given that the master can perform only one communication at a time. Once the communication order has been determined, the second problem is to decide how much work should be allocated to each worker P_i: each P_i receives α_i units of load, where $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$. The final objective is to minimize the makespan, i.e. the total execution time. Selection and ordering are the most difficult parts of the problem since the α_i's can then be found by solving a simple linear program (closed-form expressions are also available [7, 2]).

- One-round distributions lead to a poor utilization of the workers. To alleviate this problem, *multi-round* algorithms have been proposed. These algorithms dispatch the load in multiple rounds of work allocation, and thus improve the overlapping of communications with computations. By comparison with one-round algorithms, work on multi-round algorithms has been scarce. The two main questions that must be answered are: (i) what should the chunk sizes be at each round? and (ii) how many rounds should be used? It is widely acknowledged that the latencies introduced in the affine model make the model more realistic and cannot be avoided when dealing with multi-round algorithms.

Definition 7 (UNBOUNDED-DIVISIBLE $((g_i)_{1\leqslant i\leqslant p}, (G_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, W, T)$). *Let $T > 0$ be a time-bound, and consider the divisible platform. Is it possible to process all the W load units within T time-units on this platform, using a multi-round distribution?*

The complexity of this problem is still an open problem, even for one-round distributions.

Definition 8 (BOUNDED-DIVISIBLE $((g_i)_{1\leqslant i\leqslant p}, (G_i)_{1\leqslant i\leqslant p}, (w_i)_{1\leqslant i\leqslant p}, (b_i)_{1\leqslant i\leqslant p}, W, T)$). *Let $T > 0$ be a time-bound. Consider the divisible platform, where each slave P_i cannot hold more than b_i units of load at any moment. Is it possible to process all the W load units within T time-units on this platform, using a multi-round distribution?*

However BOUNDED-DIVISIBLE is strongly NP-complete, for both one-round and multi-round distributions. Once again, the proofs, even if more technical, are very similar to the proof presented in Section 2.1. Therefore they are omited for brievety but can be found in [4].

5 Related Work

To the best of our knowledge, the closest work on throughput maximization under memory constraints is presented in [8, 17] and [11].

In [8, 17], the authors study the number of buffers required to reach the optimal steady-state throughput. They experimentally state that with non-interruptible communications, a bandwidth-centric protocol using a fixed number of buffers will not reach optimal steady-state throughput in all trees. Therefore they propose an autonomous buffer growing protocol that automatically adjusts the number of required buffers. To solve the anarchic growth of buffers problem, they study the situation where communications are interruptible. They experimentally show that, when allowing interruptible communications, three buffers are sufficient to reach optimal steady-state throughput in 99,6% of the cases. Allowing interruptible communications allows communications to flow continuously at a fixed rate and therefore amounts to modify the granularity, hence minimizing buffering.

Another theoretical result whose framework is close to ours is given by Drozdowski et al. [11]. The authors consider scheduling divisible loads on a distributed computing system with limited available memory. They use the same model as in this paper and show that finding the optimal one-round load distribution is NP-hard under memory constraints (using a reduction to 2-Partition that is weakly NP-hard [14]). Using integer linear programming, they propose a robust (albeit possibly non-polynomial) algorithm to tackle the difficulty of this problem and demonstrate its efficiency using extensive simulations.

The complexity results for the distribution of independent tasks on different platforms, with or without memory limitations, are summarized in Table 1.

6 Conclusion

In this paper, we have studied the allocation of a large number of independent, equal-sized tasks, on simple star platforms, under different application models and different objective functions. We have also studied the same problem in the context of divisible tasks. In all these situations memory limitations lead to NP-completeness results. We believe that the classification of these scheduling problems will prove useful to the community, and will foster more work on the open problems listed in Table 1.

For the objective of makespan minimization, we have been able to derive an approximation algorithm. We have introduced several heuristics which have been evaluated and compared by performing extensive simulations. Unexpectedly, classical list-based scheduling heuristics that aim at greedily minimizing the completion time of each task are outperformed by the simplest heuristic that consists in delegating data to the available processor that has the smallest communication time, regardless of its computation power.

References

[1] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[2] G. Barlas. Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees. *IEEE Trans. Parallel Distributed Systems*, 9(5):429–441, 1998.

[3] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent

Objective function	Memory limitation	Star and Spider Graphs	Trees and General Graphs
Makespan Min	No	Polynomial [5, 12]	NP Complete [13]
Makespan Min	Yes	NP Complete (this paper)	NP Complete (this paper)
Throughput Max	No	Polynomial [3]	Polynomial [1]
Throughput Max	Yes	NP Complete (this paper)	NP Complete (this paper)
Divisible Linear One-round	No	Polynomial [6]	Polynomial for trees [6] **Open** for general graphs
Divisible Linear One-round	Yes	**Open**	**Open**
Divisible Affine One-round	No	**Open**	**Open**
Divisible Affine One-round	Yes	Weakly NP Complete [11] NP Complete (this paper)	Weakly NP Complete [11] NP Complete (this paper)
Divisible Linear Multi-round	No	Asymptotically optimal algorithm [6]	Asymptotically optimal algorithm [6]
Divisible Linear Multi-round	Yes	**Open**	**Open**
Divisible Affine Multi-round	No	Asymptotically optimal algorithm [6]	Asymptotically optimal algorithm [6]
Divisible Affine Multi-round	Yes	NP Complete (this paper)	NP Complete (this paper)

Table 1. Summary of complexity results.

tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium (IPDPS'2002)*. IEEE Computer Society Press, 2002.

[4] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Independent and divisible tasks scheduling on heterogenous star-shape platforms with limited memory. Technical Report 2004-22, LIP, ENS Lyon, France, Apr. 2004.

[5] O. Beaumont, A. Legrand, and Y. Robert. A polynomial-time algorithm for allocating independent tasks on heterogeneous fork-graphs. In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 115–119. CRC Press, 2002.

[6] O. Beaumont, A. Legrand, and Y. Robert. Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, 29:1121–1152, 2003.

[7] V. Bharadwaj, D. Ghose, and V. Mani. Optimal Sequencing and Arrangement in Single-Level Tree Networks with Communication Delays. *IEEE transactions on parallel and distributed systems*, 5(9), 1994.

[8] L. Carter, H. Casanova, J. Ferrante, and B. Kreaseck. Autonomous protocols for bandwidth-centric scheduling of independent-task applications. In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.

[9] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Ninth Heterogeneous Computing Workshop*, pages 349–363. IEEE Computer Society Press, 2000.

[10] E. G. Coffman. *Computer and job-shop scheduling theory*. John Wiley & Sons, 1976.

[11] M. Drozdowski and P. Wolniewicz. Divisible Load Scheduling in Systems with Limited Memory. *Cluster Computing*, 6(1):19–29, 2003.

[12] P.-F. Dutot. Master-slave tasking on heterogeneous processors. In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.

[13] P.-F. Dutot. Complexity of master-slave tasking on heterogeneous trees. *European Journal of Operational Research*, 2004. Special issue on the Dagstuhl meeting on Scheduling for Computing and Manufacturing systems (to appear).

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[15] J. P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An enabling framework for master-worker applications on the computational grid. In *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*. IEEE Computer Society Press, 2000.

[16] 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.

[17] B. Kreaseck. *Dynamic autonomous scheduling on Heterogeneous Systems*. PhD thesis, University of California, San Diego, 2003.

[18] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Eight Heterogeneous Computing Workshop*, pages 30–44. IEEE Computer Society Press, 1999.

[19] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *Heterogeneous Computing Workshop HCW'00*. IEEE Computer Society Press, 2000.

[20] J. B. Weissman. Scheduling multi-component applications in heterogeneous wide-area networks. In *Heterogeneous Computing Workshop HCW'00*. IEEE Computer Society Press, 2000.