Scheduling instructions on hierarchical machines

Florent Blachot, Guillaume Huard, Johnatan Pecero, Erik Saule, Denis Trystram
LIG, Grenoble university, 51 avenue Jean Kuntzmann, 38380 Montbonnot St Martin, France

***Abstract*—The aim of this work is to study the problem of scheduling fine grain task graphs on hierarchical distributed systems with communication delay. We consider as a case study how to schedule the instructions on a processor that implements incomplete bypass (ST200). We show first how this problem can be expressed as scheduling unitary tasks on a hierarchical architecture with heavy communications between clustered units. The proposed analysis is generic and can be extended to other challenging problems like scheduling in clusters of multi-cores.**

Our main result is an approximation algorithm based on list scheduling whose approximation ratio is the minimum of two expressions, the first one depends on the number of clusters while the second one depends on the communication delay. Experiments run on random graphs and on structured graphs demonstrate the effectiveness of the proposed approach.

I. Introduction

In this paper, we study the problem of scheduling instructions on a processor composed of several hierarchical functional units with incomplete bypass. The ST200 (designed by ST microelectronics) is an example of such a processor that will be used as a case study for our theoretical analysis. The main contribution is to design a new low cost scheduling algorithm and to prove that it is within a factor from the optimal that is the minimum of two expressions, the first one depends on the number of groups induced by the incomplete bypass mechanism, whereas the second depends on the communication time. Moreover, we prove that the approximation ratio can not be improved for this algorithm and we demonstrate empirically its effectiveness on extensive experiments.

The remainder of this paper is organized as follows: existing works related to scheduling for clustered architectures are presented in Section II. Section III presents the hierarchical scheduling problem and its complexity. Section IV discusses basic methods to deal with this problem and proves a guaranteed approximation factor that applies to the class of algorithms based on list scheduling. Section V proposes a new approximation algorithm and its analysis. This algorithm is finally assessed experimentally by some experiments on both structured and random graphs in Section VI.

This work was supported by the CIFRE convention 722361 between STMicroelectronics and CNRS

II. Related Works

Several authors already addressed the issue of scheduling for a clustered processor architecture. The most known result is probably the Unified Assign and Schedule (UAS) algorithm proposed by E. Özer *et al.* [12]. This algorithm is derived from the well-known list scheduling, it performs cluster assignment and scheduling using a priority list updated after each placement. Actually, UAS is a framework that can be adapted to specific scheduling environment by choosing the relevant priority function for the choice of operations in the list. The Earliest Task First (ETF) algorithm, used in the experiments of Section VI, is a special case of UAS (giving priority the earliest schedulable task). Other authors have also proposed variants of this approach [14].

Another significant study has been proposed by Chu, Fan and Mahlke in [5]. This approach share some ideas with the well-known DSC heuristic [15]. The idea is to assign priority to communication edges in the precedence graph depending on their impact on the critical path. The graph is then passed to a standard graph partitioning tool and the resulting clusters are scheduled. Because of the priority chosen, the partitioning will be very similar to DSC partitioning which groups operations based on the critical path (dominant sequence). A similar algorithm were previously investigated in [2].

Other results have been presented, based on straightforward modulo scheduling extension [6], iterative partition and schedule schemes [1], [11] or more unusual operations affinity iterative refinement [9]. Nevertheless, these approaches have the drawback of iterating scheduling attempts without warranty that the convergence is fast or that local minima are avoided.

Finally, all these existing results propose various heuristics that intent to improve experimentally over

other ones, but none deals with the quality of the schedule relative to the optimal solution. In this article our goal is different, closer to what Graham did with list scheduling in [7].

III. Scheduling on Hierarchical Machines

In this section we present a formal model of a hierarchical architecture induced by an incomplete bypass system, and a tasks dependence graph induced by the piece of code to schedule.

As usual, a program is represented by a directed graph $G = (T, E)$ where the set of vertices T is a set of n tasks. Each edge $e = (t_i, t_j) \in E$ is a precedence constraint meaning that the results of task t_i must be available before t_j starts its execution [10]. We denote by p_i the processing time of task t_i. In the context of ST200 architecture all tasks have unit execution time (that is $p_i = 1$ for all i). The set of predecessors (resp. successors) of t_i is denoted by $\Gamma^-(i)$ (resp. $\Gamma^+(i)$). In the transitive closure, the set of predecessors (resp. successors) of t_i is $\Gamma^{-*}(i)$ (resp. $\Gamma^{+*}(i)$).

In terms of classical scheduling problems, the application graph has to be scheduled on P, a set of mM processors (*i.e.* the functional units). Those processors are organized in M groups of size m often called *clusters* in the literature [4]. The communication delay between two processors in the same cluster is negligible, while it takes ρ units of time if the two processors belong to different clusters. On the ST200, the communications delays are constant. In the case where they can be different, we can use only the maximum communication delay. The l-th cluster is denoted $P_{l\bullet}$ whereas the k-th processor of the l-th cluster is denoted $P_{lk} \in P_{l\bullet}$.

A solution of the problem of scheduling an application graph on the processors is to determine two functions $\pi : T \to P$ and $\sigma : T \to \mathbb{Z}^+$ that give respectively a processor assignment and an execution date for each task of the graph. These functions have to match the scheduling constraints: if t_i is scheduled on processor $\pi(t_i)$ at time $\sigma(t_i)$ then :

- a processor can execute only one task at a time: there are no tasks t_i and t_j such that $\pi(i) = \pi(j)$ and $\sigma(i) = \sigma(j)$
- the precedence constraints must be fulfilled: for all edges $e = (t_i, t_j) \in E$, $\sigma(i) < \sigma(j)$ if $\pi(i) \in P_{l\bullet}, \pi(j) \in P_{l\bullet}$ and $\sigma(i) < \sigma(j) + \rho$ if $\pi(i) \in P_{l\bullet}, \pi(j) \notin P_{l\bullet}$

The associated optimization problem is to minimize $C_{max} = \max_{t_i} \sigma(t_i) + 1$. C_{max} is the completion time of the last instruction. The standard 3-fields notation $\alpha \mid \beta \mid \gamma$ has been extended by Bampis *et al.* to hierarchical architectures [4]. The problem we consider is denoted by $P_M(P_m)|prec, p_j = 1, c = (\rho, 0)|C_{max}$. The ST200 is a particular case where $M = 2$, $m = 3$ and $\rho = 2$.

In its general form, the problem $P_M(P_m)|prec, p_j = 1, c = (\rho, 0)|C_{max}$ is NP-complete as it contains NP-complete problems as particular cases. For instance, with $m = 1$ and $\rho = 1$, $P_M \mid prec, p_j = 1, c_{ij} = 1 \mid C_{max}$ is already NP-complete [13].

IV. List Based Algorithms

List Scheduling [7] is a well-known class of scheduling algorithms. Informally, the general principle is to schedule the application iteratively one time slot after the other. At each step, each processor executes a ready task if any. A general list scheduling applied to the hierarchical case is a $(2 - \frac{1}{mM} + \rho)$-approximation algorithm [3]. The question of the existence of an instance on which the $(2 - \frac{1}{mM} + \rho)$ bound is achieved is open. However, instances on which the ratio between the makespan of a List Scheduling schedule and the optimal one equals to $(\frac{3+\rho}{2})$ can be constructed (with the following instance: 2 identical interleaved chains of k tasks and $2k(m-1)$ independent tasks). Earliest Task First (ETF) is a well-known particular list scheduling which will be used for experimental validation in Section VI.

In ETF, the ρ factor comes from tasks which have predecessors on different clusters. Such tasks induce idle times which come from communications. Getting rid of the ρ factor could be achieved by avoiding these communications. In the following, we need to distinguish three kind of idle times. The first one is related to the critical path of a schedule. The second one is related to the delay induced by communications. The third one is generated explicitly by delaying a task. It is possible to show that any idle time is exactly in one of these three kind. They are formally defined as follows:

Definition 1: In a schedule (π, σ), an **idle time** is a pair (P_{lk}, t) such that $\nexists i \in T$ such that $\pi(i) = P_{lk}$ *and* $\sigma(i) = t$.

An idle time (P_{lk}, t) is said to be **due to the critical path** if $\forall i \in T$ such that $\sigma(i) > t, \exists j \in \Gamma^-(i), \sigma(j) \geq t$.

An idle time (P_{lk}, t) is said to be **due to communication** if it is not due to critical path and all the tasks delayed at this time are waiting for the completion of some communication. This translates into : $\forall i$ such that $\sigma(i) > t, (\forall j \in \Gamma^-(i), \sigma(j) < t)$ *and* $(\exists j' \in \Gamma^-(i), \pi(j') \notin P_{l\bullet}$ *and* $\sigma(j') > t - \rho)$.

The remaining idle times (P_{lk}, t) are said to be **due to postponing**. Their characterization is : $\exists i$ such

that $\sigma(i) > t, \forall j \in \Gamma^-(i), (\sigma(j) < t \; and \; \pi(j) \in P_{l\bullet})$ *or* $(\sigma(j) < t - \rho \; and \; \pi(j) \notin P_{l\bullet})$.

Schedules that contain only idle times due to the critical path are very interesting in the hierarchical case. As stated in the following result, they have a performance guarantee that depends on the number of clusters.

Proposition 1: A schedule with only idle time due to the critical path on at least one cluster is a $(M+1-\frac{1}{m})$-approximation.

Proof: The proof is inspired by the classical analysis of List Scheduling in [7]. Let $P_{1\bullet}$ be a cluster on which there is only idle times due to the critical path in the schedule (if at a time t there are ready tasks that are not scheduled, then $P_{1\bullet}$ is completely busy). Let t^∞ be the length of the longest path of G.

Using the same argument as Graham's, in the schedule, there are at most t^∞ times where $P_{1\bullet}$ has some idle resource. Indeed, if $P_{1\bullet}$ is partially idle, it is not waiting for the result of a task executed on another cluster otherwise, the idle time would be due to communication. Thus, the Graham's argument holds and the idle time is related to the longest path. During these time slots, at least t^∞ tasks are executed.

At all other time slots, at least $P_{1\bullet}$ is completely busy. This means that it takes at most $\lfloor \frac{n-t^\infty}{m} \rfloor$ units of time to schedule the remaining tasks.

Thus, $C_{max} \leq t^\infty + \lfloor \frac{n-t^\infty}{m} \rfloor$. Since both t^∞ and $\lceil \frac{n}{mM} \rceil$ are lower bound of C^*_{max}, we obtain $C_{max} \leq (M + 1 - \frac{1}{m})C^*_{max}$ ■

Notice that scheduling the whole graph only on the processors of a single cluster using classical list scheduling produces a schedule with only idle time due to the critical path on one cluster. Let GSingle be the name of this scheduling algorithm. It is an $(M + 1 - \frac{1}{m})$-approximation algorithm.

V. A New Approximation Algorithm

GSingle uses only one M-th of available resources. Thus, it is likely to produce almost systematically worst case schedule scenario. In this section, we propose the FavoriteCluster algorithm (see details in Algorithm 1) that produces schedules with only idle times due to the critical path on one cluster while making use of all the available resources.

Our heuristic works by considering $P_{1\bullet}$ as a master cluster and applying a modified List Scheduling on all the resources. List Scheduling maintains a list of ready tasks that are greedily scheduled on available resources. A task is scheduled on another cluster than $P_{1\bullet}$ only if it is ready for all the clusters (however, it is scheduled as soon as possible). If at any time a processor on the master cluster is idle due to a communication, recent schedule decisions (in the last ρ units of time) are examined: if some task recently scheduled on another cluster than the master prevents all its successors to be ready, then it is brought back to the master cluster. This slight schedule modification eliminates any possibility of generating an idle time due to communications on the master cluster. Furthermore, because it uses a list scheduling as a basis, our heuristic does not generate idle due to postponing at least on the master cluster. An example of the algorithm is illustrated in Figure 1.

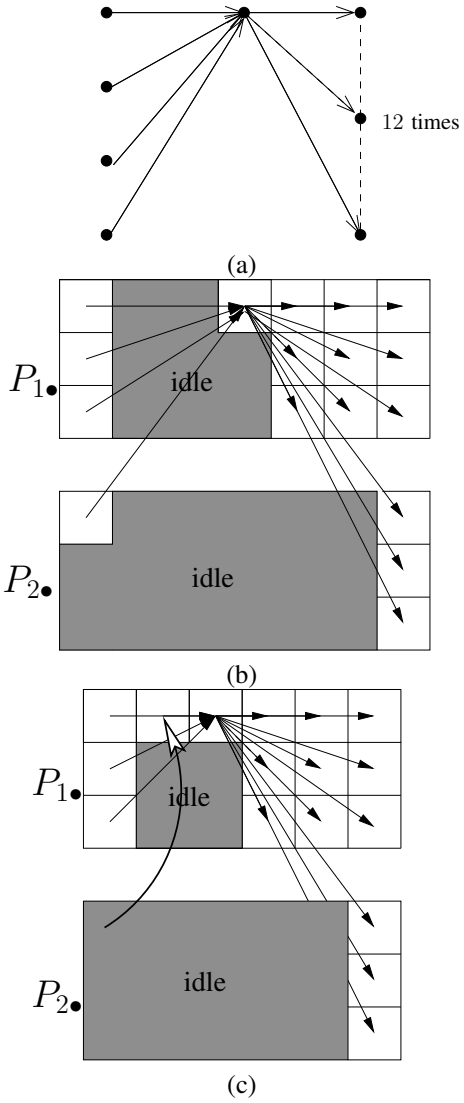

Fig. 1. (a)A Directed Acyclic Graph of tasks (b)Basic List Scheduling (c)FavoriteCluster schedule. In (c), a task was scheduled on $P_{2\bullet}$ at the beginning, but was moved to $P_{1\bullet}$ in order to keep it loaded.

FavoriteCluster belongs to the class of scheduling

Algorithm 1 FavoriteCluster

```
1  input : G = (T, E), m, M, ρ
2  output : π and σ
3  begin
4      t = 0; π = undef; σ = undef
5      while ∃i ∈ T, π(i) = undef
6          let rel_1 = {i ∈ T | ∀j ∈ Γ⁻(i), (π(j) = P_1x and σ(j) < t) or (π(j) = P_kx and σ(j) < t − ρ, k ≠ 1)}
7          if |rel_1| < m then
8              schedule all rel_1 on P_1•
9              let past = {i ∈ T | Γ⁺(i) ≠ ∅, σ(i) ≥ t − ρ and π(i) ∉ P_1•}
10             reschedule at most m − |rel_1| task of past on P_1•
11         else
12             schedule m tasks of rel_1 on P_1•
13             let ft = {i ∈ T | ∀j ∈ Γ⁻(i), σ(j) < t − ρ}
14             schedule at most (M − 1)m tasks of ft on non master clusters asap
15         end if
16         t = t + 1
17     end while
18 end
```

algorithms that meet the requirements of Proposition 1. Furthermore, it is easy to predict that FavoriteCluster behaves better in practice than GSingle because it makes use of all the clusters and better balances the load.

Proposition 2: FavoriteCluster generates schedule with only idle time due to the critical path on $P_{1\bullet}$.

Proof: Let $I = (P_{1x}, t)$ be an idle time on $P_{1\bullet}$.

By contradiction, assume that I is due to communication. Then, there is a task i scheduled on another cluster than $P_{1\bullet}$ where $t - \rho \leq \sigma(i) \leq t - 1$. This task is by construction in the set $past$ (line 9). Either it has been rescheduled on $P_{1\bullet}$ or $P_{1\bullet}$ is not idle at all. In both cases, this contradicts the fact that I is an idle time due to communication.

Assume now that I is due to postponing. Then, there exists a task i that could have been scheduled at time t. i is by construction in the set rel_1 (line 6). Either it is scheduled on $P_{1\bullet}$ or $P_{1\bullet}$ is not idle at all. In both cases, this contradicts the fact that I is an idle time due to postponing. ∎

When $M \leq \rho$, the $(M + 1 - \frac{1}{m})$ bound is tight. Figure 2(a) depicts a family of instances depending on a parameter k for which the approximation ratio of C^{FC}_{max} to C^{*}_{max} tends to the bound when parameter k tends to infinity. The optimal schedule in the case of ST200 is given in (b) while the schedule constructed by FavoriteCluster is given in (c).

The respective performances for the optimal and the

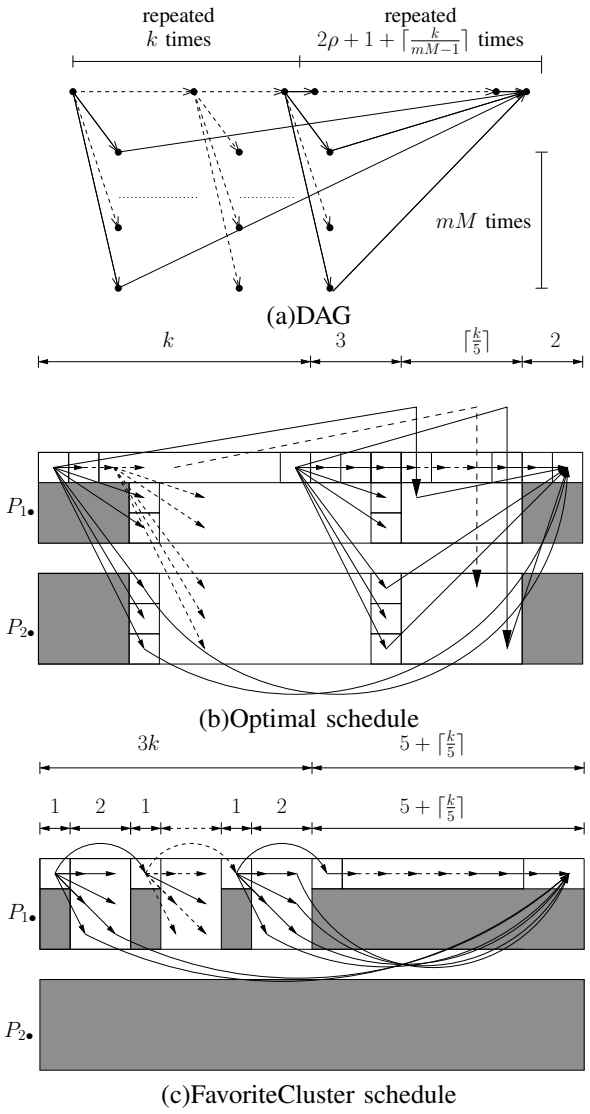

Fig. 2. The bound is tight for FavoriteCluster ($M \leq \rho$). Gantt charts are given using the ST200 characteristics (case $m = 3, M = 2, \rho = 2$). (in (b) and (c) some edges were omitted for a better readability)

approximation algorithms are:

$$C^{*}_{max} = k + 2\rho + 1 + \lceil \frac{k}{Mm - 1} \rceil \quad (1)$$

$$C^{FC}_{max} = kM + k + 2\rho + 1 + \lceil \frac{k}{Mm - 1} \rceil \quad (2)$$

Leading to the ratio :

$$\frac{C^{FC}_{max}}{C^{*}_{max}} = \frac{kM + k + 2\rho + 1 + \lceil \frac{k}{Mm-1} \rceil}{k + 2\rho + \lceil \frac{k}{Mm-1} \rceil} \geq \frac{k(M+1) + 2\rho + 1 + k/(Mm-1)}{k + 2\rho + k/(Mm-1) + 1}$$

$$\lim_{k\to\infty} \frac{C_{max}^{FC}}{C_{max}^{*}} \geq \frac{M+1+1/(Mm-1)}{1+1/(Mm-1)} = M+1-\frac{1}{m}$$

When k tends to infinity, the ratio tends to $M+1-\frac{1}{m}$ which is the worst case ratio.

The issue with this approximation ratio for FavoriteCluster is that it gets larger as M grows. In other words, when more parallel capabilities are available (additionnal clusters) our guarantee is less interesting. Fortunately, the example that proves the tightness of the bound is only valid for $M \leq \rho$. In other cases, when many clusters are available, only a fraction of parallel tasks will be rescheduled to the master cluster: just enough to overlap communications. In this situation, the worse that could happen is that all the tasks are scheduled on the master cluster because the dependence graph is not sufficiently parallel. But this means that, when $M > \rho$, the approximation ratio should grow along with ρ and not with M. This is what proves the following proposition.

Proposition 3: FavoriteCluster is a $(2+2\rho-\frac{2\rho}{M}-\frac{1}{mM})$-approximation algorithm.

Proof: Let us denote by I_{cp} the number of idle times due to the critical path in the schedule, by I_{comm} the number of idle times due to communication and by I_{post} the number of idle times due to postponing. Remark that there is no idle due to communication or postponing on the master cluster. The following equation states the occupation of the area of the schedule:

$$mMC_{max} = n + I_{cp} + I_{comm} + I_{post}$$

It is well-known from the proof of List Scheduling [7], that $I_{cp} \leq (mM-1)t^{\infty}$.

We are now going to prove that $I_{comm} \leq \rho m(M-1)t^{\infty}$. By contradiction, suppose that we are able to find t^{∞} idle times separated by ρ time units. By definition of idle time due to communication, all the tasks scheduled after the idle time have a predecessor before the idle time. Thus, it is possible to extract a chain of tasks of length $t^{\infty}+1$ which is impossible by definition of t^{∞}.

The same proof stands for I_{post}. If there is an idle time due to postponing, then there was an idle time due to communication on the master cluster. From that point, the same proof as for I_{comm} applies and leads to $I_{post} \leq \rho m(M-1)t^{\infty}$.

All those expressions lead to:

$$C_{max} \leq \frac{n+(mM-1)t^{\infty}+2\rho m(M-1)t^{\infty}}{mM}$$

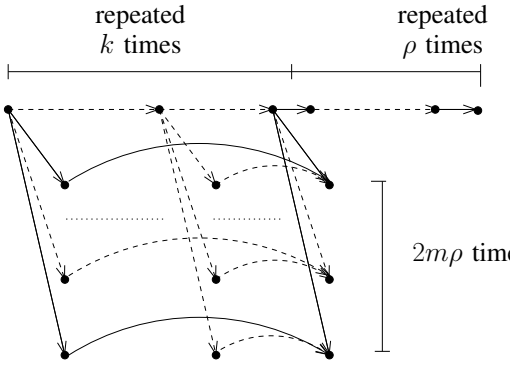

Fig. 3. The second bound is almost tight for FavoriteCluster on this family of instances ($\rho < (M-1)/2$).

$$C_{max} \leq \left(2+2\rho-\frac{2\rho}{M}-\frac{1}{mM}\right)C_{max}^{*}$$

■

When $\rho < (M-1)/2$, the bound is almost tight. Figure 3 depicts a family of instances depending on a parameter k for which the approximation ratio $\frac{C_{max}^{FC}}{C_{max}^{*}}$ tends to $2+2\rho-\frac{2\rho}{M}-\frac{2}{mM}$ when k tends to infinity.

The optimal solution schedules all the tasks in $k+\rho$ time units with no idle time since there are exactly enough independent tasks to fill the gaps. FavoriteCluster starts by scheduling all the independent tasks first. Then, it schedules the core of the instance by not scheduling the critical chain in priority. After each task i of this chain, $m\rho$ of the lower tasks are scheduled on the master cluster. Then, all the other successors of i are scheduled in parallel onto the parallel clusters. To ensure that the first cluster does not have idle due to communications, some tasks are sent back to the master cluster, starting by the lower tasks and finishing by the task of the critical chain. 2ρ time units are spent between 2 tasks of the critical chain. Using FavoriteCluster, the makespan is dominated by: $k(2+2\rho-\frac{2\rho}{M}-\frac{2}{mM})$.

Let us discuss the bound of FavoriteCluster and its interest. First, we summarize the previous results in the following theorem.

Theorem 1: The competitive ratio of FavoriteCluster is $\min(2+2\rho-\frac{2\rho}{M}-\frac{1}{mM}, M+1-\frac{1}{m})$.

The two bounds that apply on the makespan come from different principles. The first bound appears when there is not enough parallelism for using all the processors efficiently. In the most extreme case, the solution

generated by FavoriteCluster is equivalent to the one generated by GSingle. The second bound states that when there is some parallelism and all processors are used, communication delays do not induce a too large overhead. It proves that FavoriteCluster is theoretically better than GSingle.

One could argue that the bound of FavoriteCluster can be worse than the bound of List Scheduling. However, there exist cases where List Scheduling is not optimal and GSingle does not uses the parallelism but where FavoriteCluster is able to find a good solution (the instance depicted in Figure 1 is such an example). Moreover, because it does not try to group related tasks, List Scheduling could be better than FavoriteCluster when the graph contains much parallelism. But such cases are often easy in practice and almost all algorithms are optimal. In a certain sense, FavoriteCluster is a compromise between GSingle and List Scheduling. Its goal is to be efficient in cases where the parallelism exists but is not easy to detect.

VI. Experiments

To assess the practical efficiency of FavoriteCluster, we conducted experiments on two classes of instances: structured graphs such as LU factorization and random layered graphs. These instances should both provide a wide range of distinct situations and allow us to validate our heuristic when scaling to very large graphs (thousands of nodes). The objective is to assess the efficiency of FavoriteCluster in the general case and not only when scheduling small canonical benchmarks.

In these experiments, we compare FavoriteCluster with GSingle as well as with the well-known ETF list scheduling (Earliest Task First). We use the values corresponding to the ST200, that is $M = 2, m = 3$ and $\rho = 2$.

A. Structured Graphs

Kwok and Ahmad [8] proposed a set of tasks graphs instances to benchmark scheduling algorithms that take into account communication delays on homogeneous processors. The benchmark contains some trace graphs which are actual instances extracted from parallel compiler logs. Among others, trace graphs contain Cholesky, FFT, Gauss, Laplace, LU and MVA algorithms. This subsection presents the results obtained using each of the three scheduling algorithms on those graphs.

The graphs studied in this work have constant computation and communication times. Thus, experiments have been run using only the structure of Kwok and Ahmad graphs where the communication delays and computation times have been replaced by constants.

Figure 4 presents the results obtained on LU factorization. Let us remind that Gsingle is a list scheduling algorithm applied on a single cluster. Makespans (on y-axis) are normalized by dividing them by the makespan obtained by GSingle. The x-axis is the number of nodes in the DAG.

The experiments look very similar. FavoriteCluster outperforms GSingle. Despite GSingle and FavoriteCluster having the same worst case performance ratio, FavoriteCluster is better than GSingle in practice (intuitively this was expected since it makes use of more resources).

Furthermore, the FavoriteCluster curve is always below the ETF curve. On small graphs, FavoriteCluster is significantly better than ETF. When the number of nodes in the graph increases, the difference between the two algorithms tends to disappear. This is because the edges number does not grow quadratically with the nodes number: the parallelism increases in the tasks graph and the problem becomes easier. However, FavoriteCluster remains always better than ETF on those graphs.

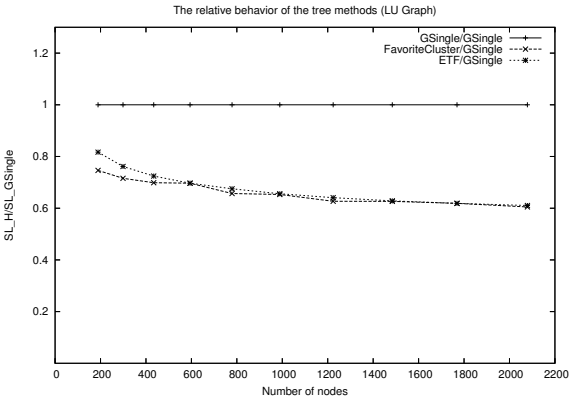

Fig. 4. Normalized makespans for the heuristics on LU factorizations.

As results obtained on the other graphs (Cholesky, FFT, MVA, Laplace and Gauss) lead to similar conclusions, they are not presented here.

B. Layered Random Graphs

This sub-section presents results obtained using a layered random graphs generator such as the one proposed in [15]. The algorithm takes 3 parameters N, L and p that can be described as follows: N nodes are distributed on L layers and for each pair of successive layers $(l, l+1)$ and each pair $(a, b) \in l \times (l+1)$, there

is an edge (a, b) with probability p. No other edges are present (especially between non successive layers).

During a first batch of experiments, we compared the three heuristics in the realistic case of the ST200 processor. We generated 500 sparse random graphs using $L = N/3$ and $p = 0.2$ and 500 denser graphs with the same values. On these graphs, the results for the three heuristics are close, most of the time the three heuristics ended up with the same makespan (50% to 70% of the cases). When the results are different, FavoriteCluster outperforms ETF in roughly 70% of the case by a small margin which seems to grow linearly with the size of the instance. For the sake of clarity, we do not include the details of these experiments as they do not outline a definitive difference between the heuristics.

Nevertheless, FavoriteCluster has been designed to take advantage of multiple clusters of functional units and to be clever in presence of large communication delays. Therefore, we have conducted additional experiments to foresee the advantages brought by FavoriteCluster for a hypothetical processor that includes more clusters and/or larger bypass latency. These experiments are presented in the following sections.

1) Increasing the communication delay: We first conducted experiments with the same machine structure ($M = 2$, $m = 3$) but with larger communication delays. We studied the cases when ρ equals 2, 5, 7, 9 and 11. For each of these values we have generated 500 layered graphs with 40 nodes each taking respectively 5 and 0.2 as L and p parameters. The figure 5 presents the resulting resulting Makespan for ETF and FavoriteCluster depending on the communication delay. The error bars represented at each point of the curve, correspond to the interval containing the real mean with a 99% confidence.

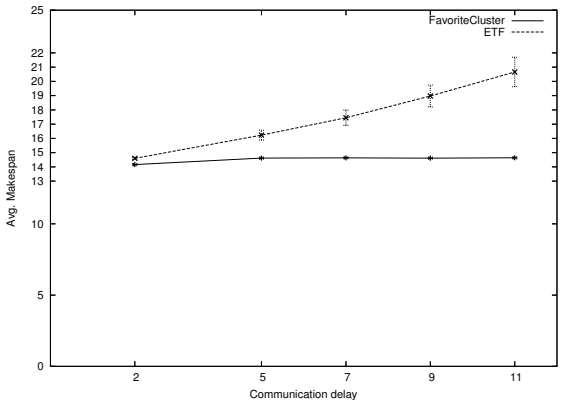

Fig. 5. Average makespan for FavoriteCluster and ETF on layered random graphs with increased communication delays.

The experimental results show that FavoriteCluster outperforms ETF whatever the communication delay. Furthermore, we can notice that the performance of ETF degrades as the communication delay increases, which is not the case for FavoriteCluster. In the case of ETF, the average makespan grows roughly linearly from 14.6 ($\rho = 2$) to 20.66 ($\rho = 11$). Indeed, this heuristic tends to split the first layer onto different clusters which later induces prohibitive communication delays. In contrary, the average makespan computed by FavoriteCluster increases only slightly when the communication delay is in the range from 2 to 5 (it goes from 14.16 to 14.61). When the communication is greater than 5, the average makespan does not increase, since FavoriteCluster allocates the tasks on the master cluster avoiding delays induced by the communication between tasks allocated on different clusters.

2) Increasing the number of clusters: We also conducted experiments in which M (the number of available clusters) varied from 2 to 7 (keeping $m = 3$ and $\rho = 2$). These experiments have been performed on the same layered graphs as in the previous section. The corresponding results are presented in Figure 6.

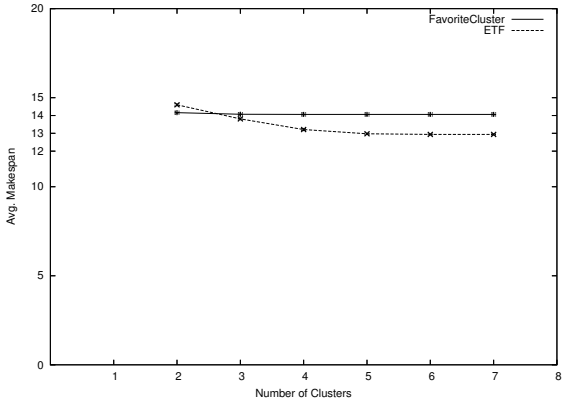

Fig. 6. Average makespan for FavoriteCluster and ETF on layered random graphs with an increased number of clusters.

The average makespan of ETF schedules slightly decreases when the number of clusters increases until it reaches 5. On the contrary, FavoriteCluster does not seem to be able to exploit all the available resources, but its performance does not degrade. Above 5 clusters, there are more resources than available parallelism in the application graph and the Makespan does not change anymore. In these tests, ETF performs slightly better than FavoriteCluster although the gap between both algorithms stays small (and constant).

3) Increased communication delays and large number of clusters: Finally, we have studied the behavior of the algorithms when both the number of clusters and the communication delay increase. The same layered graphs are used for this experiment, but the communication delays are increased to 5 and the number of clusters vary from 2 to 7. The results of this experiment are shown in Figure 7.

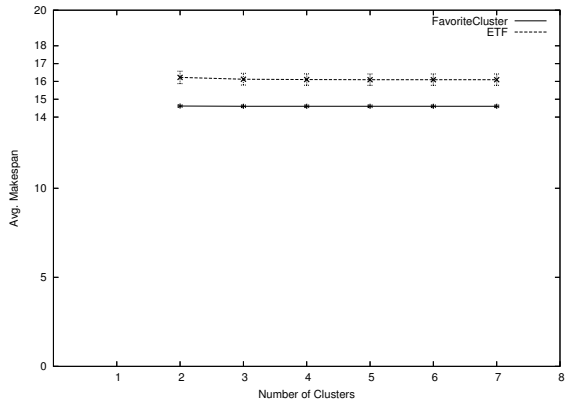

Fig. 7. Average makespan for FavoriteCluster and ETF on layered random graphs with large communication delays and increased number of clusters.

The average makespan for both algorithms is now constant. Indeed, in this experiment, extra clusters cannot be exploited efficiently due to the high communication costs. Thus, FavoriteCluster performs consistently better than ETF. In this experiment, the confidence interval is larger for ETF than for FavoriteCluster. This indicates that the makespans of ETF schedules are spread, probably due to bad tasks placement and resulting communications stall hazards.

VII. Conclusion

In this work, we proposed a new scheduling analysis for dealing with incomplete bypass in VLIW processors. This model can be directly used with classical algorithms for scheduling tasks with communication delays. Nevertheless, we proved that in the hierarchical platforms, it is possible to improve the general approximation bounds. We showed that a simple list scheduling algorithm applied on only a part of the machine was actually a $(M + 1 - \frac{1}{m})$-approximation (where the processor has M clusters of m units). We improved this algorithm by proposing FavoriteCluster, an algorithm with the same guarantee that makes use of all the available resources. It is also guaranteed by a second factor which depends on communication delays: $2 + 2\rho - \frac{2\rho}{M} - \frac{1}{mM}$. Actually, our algorithm is a good tradeoff between locality and parallelism exploitation. Finally, FavoriteCluster was assessed by intensive experiments (structured and random graphs). We believe that this work may be the basis for the analysis of scheduling on clusters of multi-core systems.

References

[1] C. Akturan and M. F. Jacome. Caliber: A software pipelining algorithm for clustered embedded VLIW processors. In *ICCAD*, pages 112–118, 2001.

[2] A. Aleta, J. Codina, J. Sanchez, and A. Gonzalez. Graph partitioning based instruction scheduling for clustered processors. In *Proc. of 34th Int. Symp. on Microarchitecture*, 2001.

[3] F.D. Anger, Y-C. Chow, J-J. Hwang, and C-Y. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):87–95, January 1989.

[4] E. Bampis, R. Giroudeau, and J-C. König. An approximation algorithm for the precedence constrained scheduling problem with hierarchical communications. *Theor. Comput. Sci.*, 290(3):1883–1895, 2003.

[5] M. Chu, K. Fan, and S. Mahlke. Region-based hierarchical operation partitioning for multiclusters processors. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 300–311, 2003.

[6] M. M. Fernandes, J. L., and N. P. Topham. Distributed modulo scheduling. In *HPCA*, pages 130–134, 1999.

[7] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[8] Y-K. Kwok and I. Ahmad. Benchmarking the task graph scheduling algorithms. In *IPPS/SPDP*, pages 531–537, 1998.

[9] W. Lee, D. Puppin, S. Swenson, and S. Amarasinghe. Convergent scheduling. In *MICRO-35*, pages 111–122, 2002.

[10] J.K. Lenstra and D.B. Shmoys. *Scheduling Theory and its Applications*, chapter 1 - Computing Near-Optimal Schedules, pages 1–14. Wiley, 1995.

[11] R. Leupers. Instruction scheduling for clustered VLIW DSPs. In *IEEE PACT*, pages 291–300, 2000.

[12] E. Ozer, S. Banerjia, and T. M. Conte. Unified assign and schedule: A new approach to scheduling for clustered register file microarchitectures. In *International Symposium on Microarchitecture*, pages 308–315, 1998.

[13] V. J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18:55–71, 1987.

[14] J. Sánchez and A. González. Instruction scheduling for clustered vliw architectures. In *13th International Symposium on System Synthesis*, 2000.

[15] T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems*, pages 951–967, September 1994.