# Formalization and Static Optimization of Parallel Implementations

Annie Vicard and Yves Sorel

INRIA-Rocquencourt Domaine de Voluceau B.P. 105 - 78153 Le Chesnay Cedex - France
{Annie.Vicard, Yves.Sorel}@inria.fr

*Abstract*— **In this paper, we present a relational formalism based on graphs theory to build all the possible implementations of an algorithm onto a parallel machine. Here, the term implementation means, not only as usual, the distribution and the scheduling of the computations, as well as the distribution of the inter-processor communications, but also, and this is a crucial issue, the scheduling of the inter-processor communications, possibly routed. Then, we present a greedy heuristic improved by a back-tracking method, which chooses among the previous set, a sub-optimal implementation taking into account the critical path length and the schedule flexibility. This type of optimization is static, because the distribution and the scheduling choices are done during the compilation, instead of during the real-time execution. Finally, we compare our heuristic using several examples, with other ones of the same type presented in the literature.**

*Keywords*—**Formalization, Parallel Implementation, Distribution and Scheduling Heuristic, Greedy List Scheduling Algorithm.**

## I Introduction

We are interested in signal and image processing, and control applications which interact with their environment and must satisfy real-time constraints. We are seeking among the set of all the possible parallel implementations of a given application algorithm [1] onto a given parallel architecture, the one inducing the smallest as possible execution duration.

In order to achieve this goal, the implementations set is described with a common formalism based on graphs models, by taking advantage of the potential parallelism of the algorithm according to the available parallelism of the architecture [7], and then, because these problems are NP-hard [2], an optimization heuristic is used to choose a sub-optimal solution among this implementations set. The optimization is static, i.e. performed during the compilation, instead of during the real-time execution, because this approach allows to generate automatically static distributed real-time executives, involving a very low overhead and preventing from dead-locks. These executives may be generated, with a system level CAD software such as SynDEx.[2]

In this paper, we first present the implementation formalism through the algorithm, the architecture and the implementation models. Then, we present a more efficient variant, with back-tracking, of the optimization heuristic proposed in [5]. Finally, we briefly describe classical heuristics and compare them with ours.

## II Implementation Formalism

### *II.A Architecture model*

We model a parallel architecture by an undirected graph. The architecture model is not, as usually, a simple processors network. Indeed, in order to accurately model the scheduling of inter-processor communications [3], each processor is composed of one computation unit connected by a point to point link to at least one communication unit. This leads to the developed model proposed in [1], which is also a network where communication units belonging to different processors, are connected by point to point links. There is no link between computation units inside a processor. In the architecture graph, vertices are units and edges are links.

We assume that the architecture is homogeneous, i.e. all the computation units have identical characteristics, as well as, all the communication units and the inter-unit links. We also assume that the communication units are not fully connected.

### *II.B Algorithm model*

We model an algorithm by a directed data flow graph. It is a factorized data dependences graph, i.e. an infinitely repeated pattern which is represented only once. In the data dependences graph, vertices are computation *operations* and edges are *data dependences* between operations.

Using the architecture characteristics, we associate an execution duration to each operation of the algorithm graph. We do not associate an execution duration to each data dependence, because we do not know before the implementation where it will be distributed and scheduled. However, these execution durations will be computed by the optimization heuristic described further on.

### *II.C Implementation model*

The possible implementations set of a given algorithm onto a given architecture is formalized as the composition of three relations [4]: the *routing*, the *distribution* and the *scheduling*, which are successively described.

[1] Later on, we shall simply use the term algorithm instead of application algorithm.

[2] http://www-rocq.inria.fr/syndex

[3] Later on, we shall use the term communication instead of inter-processor communication.

[4] Here, actually, we refer to the sets theory.

The *routing* is a relation which modifies the architecture graph in order to completely connect the communication units belonging to different processors. It is necessary when there is no direct link between two processors.

The *distribution* is a relation which successively performs two partitions of the algorithm graph. Firstly, the algorithm graph is decomposed into disjointed sub-graphs under the constraint that the number of sub-graphs is lower or equal to the number of computation units. Secondly, the data dependences between operations belonging to different partition elements, are in turn partitioned according to the number of links and associated communication units. Then, in order to assign algorithm graph vertices to communication units vertices of the architecture graph, new operations, called *communication operations*, are added to the algorithm graph between dependent operations which belong to different elements of the first partition. Thus, each set of communication operations assigned to a communication unit forms an element of the second partition. There exists a finite number of possible distributions.

The *scheduling* is a relation which must reinforce each partial order associated to each element of both previous partitions in order to obtain a total order. Indeed, each element of these partitions is assigned to either a computation unit or a communication unit, which both are sequential machine. Each partial order associated to each computation unit (resp. communication) is reinforced in a total order by adding edges between computation (resp. communication) operations of this unit.

Note that, when a processors network is assumed, the communications are only distributed and not scheduled on the links. Each link is seen as an infinite resource and then, the scheduling of the communications is not taken into account.

By composing the three previously defined relations, we describe the set of all the possible implementations of an algorithm onto a given architecture. We propose in the following section an optimization heuristic which gives a sub-optimal solution to the minimization problem of the implementation execution duration.

## III Optimization heuristic

Because implementation mainly means to distribute and to schedule the algorithm onto the architecture, the optimization heuristic will examine for each operation, the possible distributions, and a distribution being chosen, will examine the possible scheduling, taking into account communications which may be possibly routed.

The proposed heuristic is a *greedy list scheduling* algorithm presented in [5], followed by a *back-tracking method*. The list scheduling algorithm rapidly builds a quite good quality solution [6] and this solution is then improved by back-tracking. Here is a brief description of the heuristic.

At each step, a list of *ready operations* is built from the algorithm graph vertices (computation as well as communication operations). One operation of this list is selected thanks to a cost function, and thus is distributed and scheduled. Then, the selected operation is removed from the list, and its ready successors are added to the list. Finally, the heuristic ends when the list is empty.

An operation is said to be ready to the distribution and the scheduling, when all its predecessors are already distributed and scheduled. The set $O^{(n)}_{ready}$ is the set of the ready operations at the $n$-th step.

The cost function $\sigma^{(n)}(o_i/p_j)$, called *schedule pressure*, is computed as follow: $\sigma^{(n)}(o_i/p_j) = S^{(n)}(o_i/p_j) + \bar{s}(o_i)$ for $o_i \in O^{(n)}_{ready}$ and $p_j \in P$ units set. $S^{(n)}(o_i/p_j)$ is the earliest start (from start) date of $o_i$ on $p_j$, it takes into account the communications between $o_i$ and its predecessors, which may be, not scheduled, on the same unit. Similarly, $\bar{s}(o_i)$ is the latest start (from end) date of $o_i$. The schedule pressure takes into account both the schedule flexibility and the critical path lengthening.

The selected operation is obtained as follow: (1) select the best unit $p_i^{best}$ for each ready operation $o_i$, $\forall o_i \in O^{(n)}_{ready}$ , $\sigma^{(n)}_{best}(o_i/p_i^{best}) = \min_{p_k \in P} \sigma(o_i/p_k)$. Thus, we obtain the couples (ready operation, best corresponding unit) $= (o_i, p_i^{best})$. (2) select the best couple among the previous set: $\sigma^{(n)}_{opt}(o_./p_.^{best}) = \max_{o_i \in O^{(n)}_{ready}} \sigma^{(n)}_{best}(o_i/p_i^{best})$. The operation $o_.$ is then distributed and scheduled on $p_.^{best}$.

If there exists several equivalent couples $(o_i, p_i^{best})$ such that $\sigma^{(n)}_{opt}(o_i/p_i^{best}) = \sigma^{(n)}_{best}(o_./p_.^{best})$, then a couple is randomly chosen among these equivalent couples, in order to be scheduled at the $n$-th step. The other couples are marked, and will be exploited by the back-tracking further on.

The back-tracking method deschedules each operation, from the last scheduled operation, until to reach a step containing marked equivalent couples. Then, the back-tracking schedules a not yet chosen equivalent couple, and another implementation solution is computed as described above. Thanks to the back-tracking the random aspect is canceled in the optimization heuristic.

## IV Heuristics comparisons

In order to evaluate our heuristic, we briefly present heuristics of the same type and then we compare them to ours using algorithm and architecture graphs examples.

In [3], Ahmad and Kwok distinguish three heuristics categories:

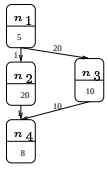


Fig. 1. Algorithm graph 1

•*BNP* -Bounded Number of Processors-. The number of processors is finite, the architecture is fully connected and the communication routing and scheduling are not taking into account. ETF (Earliest Time First) is a BNP-

algorithm. It gives the highest priority to the operation which has the smallest Earliest Start Time.

• *UNC* -Unbounded Number of Clusters-. The number of processors is infinite, the architecture is fully connected and neither the routing of the communications, nor their schedulings are performed. The DCP (Dynamic Critical Path), EZ (Edge Zeroing) and MD (Mobility Directed) belong to this category. For DCP, the operation which has the highest priority is the one which is both the more critical, and which generates the heaviest communication towards its successors. EZ distributes and schedules the operations according to the decreasing order of the communication costs. Finally, MD distributes and schedules a critical operation on a processor which has enough idle time to schedule it. If necessary and only if the partial order on the processor is maintained, the operations already scheduled on that processor can be shifted on the right, in order to insert the critical operation.

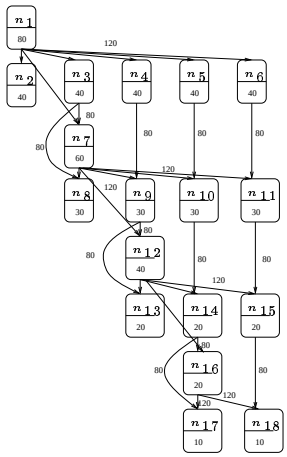

Fig. 2. Algorithm graph 2

• *APN* -Arbitrary Processor Network-. The network topology is arbitrary. There exists only few algorithms of this category. We distinguish the heuristics which: (a) route and distribute the communications on the links, and (b) route, distribute and schedule the communications on the links.

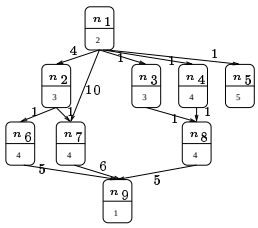

Fig. 3. Algorithm graph 3

The heuristic that we propose belongs to this last category, with one more feature: the routed communications have variable durations according to the number of links used to perform the communication. Here are three examples of APN-algorithms: the BSA (Bubble Scheduling Algorithm), BU (Bottom Up), and MH (Mapping Heuristic) algorithms presented in [3]. In BSA algorithm, all the operations are first scheduled on one processor. Then, an operation and its successors are moved to another processor if the resulting start time of this operation decreases. BU distributes and schedules the critical operations on a specific processor, and distributes and schedules the remaining operations on the other processors by optimizing load balancing on each processor. Finally, for MH scheduling algorithm, the ready to be scheduled operation which has the highest priority is the one with the highest static b_level. The static b_level of an operation is the highest sum of execution durations of the operations belonging to all the possible paths beginning from this included operation to an output operation (operation without successor).

Now we compare, using algorithm graphs examples, our heuristic with several ones described in the literature and briefly presented above. Although some of them do not really belong to the same category as ours, we accordingly restrict the assumptions related to the architecture in order to be in the same case.

Let $\sigma$, be our initial heuristic and $\sigma + BT$, the heuristic with back-tracking. Afterwards we use the term scheduling length (SL) in order to denote the execution duration of the algorithm.

| Type | ETF | EZ, MD | $\sigma$ |
|---|---|---|---|
| SL | 43 | 35 | 34 |

TABLE I
SCHEDULING ALGORITHMS COMPARISONS

With the example depicted in figure 1, we compare, using the data given in [4], scheduling algorithms belonging to the BNP and UNC categories. The number of processors in this example is equal to two. The results are given in table I. We note that our heuristic gives the smallest scheduling length for the example 1.

| Type | ETF | EZ | MD | DCP | $\sigma$ |
|---|---|---|---|---|---|
| SL | 520 | 600 | 460 | 440 | 450 |

TABLE II
SCHEDULING ALGORITHMS COMPARISONS

With the example depicted in figure 2, we compare, using the data given in [4], scheduling algorithms belonging to the BNP and UNC categories. The architecture is a four processors network which is fully connected. Neither the routing, nor the scheduling of the communications are performed. Unlike the previous comparison, in this example there are several communications per link. These communications are distributed and scheduled by our heuristic, but only distributed by BNP, UNC. In order to be not penalized by the routing and the scheduling of the communications which are done by $\sigma$, we chose an architecture with two links between each processor. Under such conditions, $\sigma$ comes in the second place (cf. table II).

| Type | BSA | BU | MH | $\sigma$ | $\sigma + BT$ |
|---|---|---|---|---|---|
| SL | 16 | 24 | 20 | 20 | 18 |
| SCD | 11 | 27 | 16 | 14 | 8 |
| SL+SCD | 27 | 51 | 36 | 34 | 26 |

TABLE III
SCHEDULING ALGORITHMS COMPARISONS

With the example depicted in figure 3, we compare, using the data given in [3], scheduling algorithms belonging to the APN category. The architecture is a ring with four processors. The architecture is not fully connected, the communications must be routed, and are scheduled. But unlike the $\sigma$ algorithm, the BSA, BU and MH scheduling algorithms assume constant communication duration for a data dependence, whatever the number of used links for this communication is. Let SCD be the sum of communication durations in the table III built by using the data coming from [3]. BSA is the best scheduling algorithm for the minimization of the scheduling length of the example 3, with a length of 16. $\sigma + BT$ gives a scheduling length equal to 18, which corresponds to the second place (cf table III).

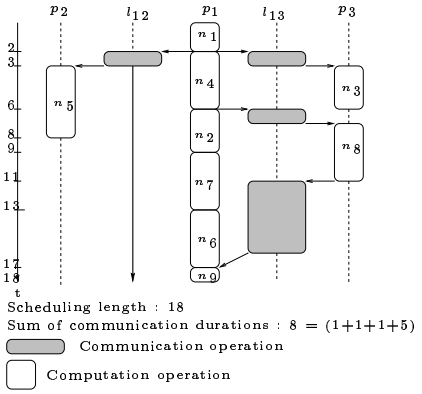


Fig. 4. Scheduling given by $\sigma + BT$

If we compare the scheduling algorithms not only on the scheduling length, but also on the SCD added to the SL, the $\sigma+BT$ algorithm is the best for this example (cf. figure 4).

However, we may note that the $\sigma$ algorithm without back-tracking gives good results. Indeed, it is in the third position for the scheduling length, and it is in the third position for the scheduling length and the communication durations, both taking into account $\sigma + BT$.

Finally, we compare $\sigma$ and $\sigma + BT$ using an algorithm graph example with 41 vertices and 62 edges not represented here. The table IV gives the scheduling length for the step and the number of corresponding equivalent operations ($n$).

Thus, at the 24th step, there exists four operations which give the same result for the cost function, and this is the back-tracking on the fourth operation which gives the smallest scheduling length (cf. table IV). In this example, the back-tracking improves the scheduling length of 22 %.

| step/$n$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 5 | 77959 | 77959 | 77959 | |
| 6 | 77959 | 77959 | | |
| 24 | 77959 | 75725 | 72671 | *60383* |
| 25 | 77959 | 72671 | 78815 | |
| 28 | 77959 | 65671 | | |
| 31 | 77959 | 77959 | | |
| 32 | 77959 | 77959 | | |

TABLE IV
BACK-TRACKING IMPROVEMENT

## V Conclusion

We proposed an implementation formalism based on graphs theory to describe, the application algorithm, the parallel architecture, and the implementations, where both the computations and the inter-processor communications may be distributed and scheduled. This formalization allows to describe the set of all the possible implementations of an algorithm onto a given parallel architecture. Because our applications must satisfy real-time constraints, we are seeking among this set, an implementation which execution duration is minimized. This problem being NP-Hard, we proposed a greedy list scheduling heuristic improved by a back-tracking method which builds a sub-optimal implementation solution. The performances of the scheduling heuristics, with and without back-tracking, were evaluated by comparing, with other scheduling heuristics of the same type, the scheduling length obtained on several application examples. We obtained good performances on these examples, and plan now, to statistically evaluate our heuristic on an important number of application examples.

## References


[1] C. Aiglon, C. Lavarenne, Y. Sorel, and A. Vicard. Utilisation de SynDEx pour le traitement d'images temps-réel. Rapport de Recherche 2968, INRIA, Septembre 1996.

[2] Garey and Johnson. *Computers and intractability : a guide to the theory of NP-completeness.* W.H. Freeman and Company, 1979.

[3] Y. K. Kwok and I. Ahmad. Bubble scheduling : A quasi dynamic algorithm for static allocation of tasks to parallel architectures. In *Proc of 7th IEEE symposium on Parallel and Distributed Processing*, pages 36–43, Octobre 1995.

[4] Y. K. Kwok and I. Ahmad. Dynamic critical-path scheduling : An effective technique for allocating task graphs to multiprocessors. In *IEEE Trans. on Parallel and Distributed Systems*, volume 7, pages 506–521. May 1996.

[5] C. Lavarenne and Y. Sorel. Performance Optimization of Multiprocessor Real-Time Applications by Graph Transformations. In *Parallel Computing*, Grenoble, Septembre 1993.

[6] Z. Liu and C. Coroyer. Effectiveness of heuristics and simulated annealing for the scheduling of concurrent task. an empirical comparison. *Proc. of PARLE'93, 5th international PARLE conference, Munich, Germany, June 14-17*, pages 452–463, Nov. 1993.

[7] Y. Sorel. Massively parallel systems with real time constraints. the "Algorithm Architecture Adequation Methodology". In *Proc. Massively Parallel Computing Systems*, Italy, May 1994.