Optimizing the Cost of an Heterogeneous Distributed Platform

Jean-Marc Nicod, Laurent Philippe, Hala Sabbah
LIFC/INRIA GRAAL, University of Franche-Comté
16 route de Gray, 25000 Besançon, France
[nicod,philippe,sabbah]@lifc.univ-fcomte.fr

***Abstract*—Distributed platforms become heterogeneous in more and more domains, as heterogeneous computing (HC) onto grids or reconfigurable factories in the industry. For production grids and factories, it is mandatory to control and optimize the economic cost of a such platforms regarding performance objectives. We present in this paper a study which purpose is to optimize the size of such environments depending on the workflow to execute or product to realize. The target platforms are either micro-factories, sized to manufacture products at the micrometric scale, or the heterogeneous computing domain where the key point is to reserve processors of an execution platform onto a grid to compute workflows like medical imaging applications. Thanks to the sizing of the platform, optimal or not, scheduling a workflow in HC environment or a production in the micro-factory is easy because the size of the platform already takes the performance constraints into account. In this paper, we present general results on the platform size optimization. Numerical results are also presented to illustrate 3 cases of our study.**

***Keywords*—grid, micro-factory, heterogeneous platform, cost optimization, workflow, scheduling.**

I. Introduction

Distributed platforms become heterogeneous in more and more domains. In particular this the case for the heterogeneous computing onto the grids and on reconfigurable factories in the production industry. This heterogeneity could come from different origins. The size of these platforms become larger and larger. It is thus too expensive to keep the homogeneity of the architecture when the platform size is extended. An other reason is that these platforms are often made by using existing processors or machines. It is also the case of the micro-factory production units (cells) that are dedicated to few tasks in a process.

The miniaturization of the products become mandatory in different domains such as mechanical, electronic, electromechanical or optic. New applications could be considered as the integration and the assembly of optic elements of very small size (mm) to achieve optic benches. Downsizing of manufacturing systems can lead to smart solutions, improving space utilization factor, reducing the price and energy consumption, including environmental conditioning such as temperature, humidity and cleanliness, as well as facility investment. The agility in reconfiguring the manufacturing lines in the factories will be elevated. Furthermore, the machines can be placed off the factory floor, to the design offices or classrooms, and distributed to small manufacturing laboratories, even in residential areas [?], [?].

However, the manipulation of this order of size poses several problems: the balance of the physical strengths in presence is not anymore the same that the one of the "macro" world and the capacities of human intervention are limits. Now a day, this type of production is mainly achieved by remote-operation which limits its wider development. The automation is therefore necessary to consider a production at a bigger scale. This requires the collaboration between roboticians, producticians and programmers.

In this context, the platform behavior is very close to the grids in general and SOA grids in particular because of both heterogeneity and specialization of processors/cells. Indeed, designing a platform by reserving processors in a SOA grid is very similar to the design of a micro-factory with respect to a performance or cost objective. The dedicated processors, which are able to compute only the tasks defined by the libraries deployed on them, are similar to the actions performed by the reconfigurable micro-factory cells. The input or output data transfers can be compared with the moving of the micro-products. Moreover, because of the very small size of the products, we can easily buffer these products.

In the following, we present a study which purpose is to optimize the design of the micro-factory or grid context. The paper is organized as the following. Section II details the micro-factory specificities and the associated issues. The next section gives the architecture model and the notations we use to solve the optimization problem of the platform sizing. The section IV is the study of 5 different cases that cover the 16 possible cases. The section V shows algorithms and numerical results for the first three cases. The next section presents the scheduling step after the designing step. We conclude and give future work in the last section VII.

II. Specificities of the micro-factory

Micro-factory has a great potential for innovation of manufacturing systems for small sized products. The concept of micro-factory dates from the nineties [?] and becomes more closely related to our daily lives. It rests on two ideas: on the one hand to achieve a factory of very small size, that can hold in a case [?], and on the other hand to automate the production of micro-products [?].

A. Considerations on the size

In a research of flexibility, the micro-factory opposes the classic vision of the factories of the ”macro” world by the size and functions of its robots. Macro-robots are complex and able to achieve a big number of operations. Micro-robots are rather of small size, i.e. few dozen centimeters large, and propose elementary operations. Some ”macro” robots have a sufficient precision degree to manipulate objects of the ”micro” world, but their precision is not on no account micrometric. On the other hand, for reasons of size, of flexibility, of energy consumption [?] or time of reconfiguration, they are not suited to the ”micro” world. According to Tanaka [?], driving energies of facilities and energy required to control the environment of the system such as air conditioning and illumination decrease with decreasing size. The choice of micro-robots, with little liberty degree and which cooperate, allows to reach the objective of size, precision and reconfiguration.

Here comes another thinking way about manufacturing systems. If the size of the manufacturing system is remarkably reduced, we will able to realize various styles of manufacturing, such as front shop manufacturing, mobile manufacturing in a vehicle. It implies a conception which is radically different from the factory in term of organization and management of the production: each micro-robots provides only a limited set of operations and it is necessary to cluster them to realize a complex function. This regrouping is called a cell. Then, the micro-factory is made by a set of cells.

The size of a cell is small, about ten centimeters. According to the task to be performed, the presence of independent actuators allows a configuration in the initialization phase for the calibration of the cell, and, in a least measure, a reconfiguration to dynamically change of the provided functions. These modifications can intervene in operate or inhibit some actuators or while modifying their order.

The factory designed to realize the final product is thus made by a cluster of basic cells on the same support. The size of this support is about one meter large and the organization of the cells is not necessarily linear there compared to most production units. The product moving function of the platform could be achieved by portico for instance. Because of the product size and the fact that products can be buffered, the transportation can be recovered by the work of the cells. Therefore, the micro-factory is very flexible.

B. The automation

The automation of the production of micro-product is a major issue of the micro-factory. Several factors act this level. First of all, it is necessary to note that strengths in presence to this order of size do not allow to use the traditional models of manipulation anymore. For example, the gravity strength of is not the most important anymore and uncontrollable electrostatic strengths can disrupt the products manipulation. For these reasons, the manner to handle the products must evolve to make place to a more flexible control that takes the increase of the mistakes into account. Besides these problems of order of size, the human intervention is difficult or generaly simply not possible. For instance, it is not possible to take such small products by hands or simply to see them without the help of a microscope.

Then one privilege an automated treatment where the non-managed cases are considered as mistakes. The organization is also an interesting point in the survey of the management of the micro-factories. Indeed, the regrouping of the actuators within cells allows to identify three levels of control for the automation. The order of the actuator is called a classic manner, that is to say some functions serve to enslave it in positioning and in time.

Cooperation between these actuators must also be managed since the action of just aone actuator is not sufficient for the realization of a function on the product. Therefore, we must integrate the synchronization between the actuators to guarantee the execution of a function. It is thus also submitted to strong temporal constraints and must offer real-time guarantees.

In opposite, the level of cooperation between cells is not necessarily synchronized since it is possible to stock the products between two cells: their very small size allows to consider buffer with a big number of products without difficulty. In this case, there is no real time constraints anymore and the organization of the production between the cells can be considered of the viwe point of the flow rather than of the control one. Thus, the automation of the cells management looks like the management of processes onto an heterogeneous distributed execution platform, as the grid.

C. Issues

The factory is known for the realization of micro-products. A product is achieved from components on which are applied a set of the functions, for example an assembly.

The application of a function to a component makes itself under the shape of a task. The set of the tasks of a product and their dependencies define a graph of tasks (DAG: Direct Acyclic Graph), called process. We limit it to DAGs without fork (intree) since it is not possible to duplicate a component. Different DAGs can generate the same product.

This context allows us to identify different optimization problems. We propose to optimize the design of an heterogeneous platform made of components (cells or processors) to ensure the fixed execution rate (number of DAGs per unit of time). The problem is well adapted to both contexts of the production of micro-products onto a micro-factory and the execution of workflows onto selected processors onto the grid. The scheduling step to allow the predicted execution rate is not difficult because the size of the platform already takes this constraint into account.

III. Optimization

To simplify the reading of this section, we use the micro-factory terminology. But when you read product, cell, micro-factory, you can also understand respectively result, processor, heterogeneous set of processors on the grid. The both points of view are considered at the same level in this section.

In this section, we try to give an answer to the following question: What micro-factory for what production? Indeed, one of the issues for a micro-factory is the manner to size it in the perspective of a production of one or several products, each described by one or several graphs of tasks (DAG) as some alternative processes can exist for a given product.

We set he hypothesis where the cells exist and can perform the necessary operations for the tasks of the processes. Thus, every types of tasks can be done by at least one type of cell. In these conditions, knowing how to design a micro-factory consists in choosing how many cells of each types to use to allow one or several productions described by the tasks graph. The answer to the previous question consists then in guaranteing a level of performance, for example in number of finished products per time unit for a minimal cost. This last criteria not being the only one possible, the final size of the micro-factory is another one. The construction of a micro-factory in this context is therefore a problem of optimization for which we propose some tracks for its resolution in different cases. But, before treating the most general case, it is important to exhibit intermediate cases for which optimal solutions to the optimization problem proposed here can be given. In the other cases, sub-optimal solutions must be proposed.

What impacts the level of generalities of the optimization problem is the fact that a cell is mono or multitask, the products defined with one or several processes or graphs of tasks and the productions thrown simultaneously with one or several products. From all combinations, we identify five cases:

(i) the cells are mono-task – can just perform one type of action –, one product is defined by only one process and only is manufactured for a production;

(ii) same case that previously with simultaneously several products, so several processes, to be performed;

(iii) the cells are always mono-task, but different cells can perform identical tasks with a different economical cost and with a different execution speed, for production of only one product defined by an unique DAG;

(iv) cells are mono-task with the production of an unique product defined by several DAGs;

(v) multi-tasks cells, only one product manufactured according to the definition of one process. All the other cases can be derive from the solutions exhibit here.

A. Architecture and Model

An heterogeneous computing grid is composed of processors P (hosts) which communicate between them by a network with high speed links. A grid G_p is represented by a graph $G_p = (P, E)$. The nodes of P are the m processors P_m of the grid. The Edges E are the network links between processors. The micro-factory is represented by the same description.

This grid treats a batch of I identical non preemptive tasks. The batch is then defined as a set of N instances $G_A^{(N)}$ of work G_A. The work G_A is an oriented acyclic graph (DAG) composed of the T_i tasks of T and the dependencies D, the set of edges between tasks.

All tasks of G_A may be all of different types or not. The example on figure Fig. 1 shows the graph of tasks and the table Table 1 is an example of platform.

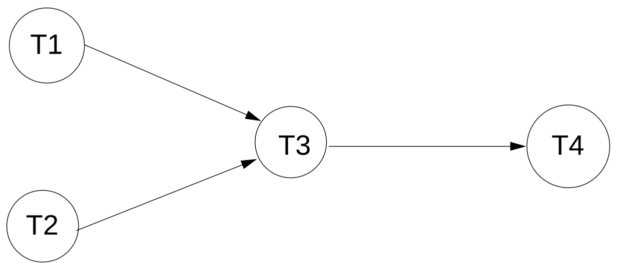

Fig. 1: Graph of tasks

TABLE I: Matrix of processors/cells performances

		p_1	p_2	p_3	p_4
Tasks types	T_1	200	∞	∞	∞
	T_2	∞	100	∞	∞
	T_3	∞	∞	200	∞
	T_4	∞	∞	∞	300

All the nodes of the platform are not able to perform all the task types. So, each tasks can just be executed only onto a sub-set of processors. The conditions of execution of each task type are given in matrices of execution cost that described the platform, as the table Table 1 for example. In the table, ∞ implies that the task described in the corrsponding line cannot be executed on the current processor.

One more time, this architecture and the model is available in the context of the micro-factory.

B. Notation

The following notations will be used in the following:

- M: number of processors used in the architecture platform;
- m: processor (cell) index, $m = 1, .., M$;
- PT: Set of processor types;
- i: index of task types, $i = 1, .., I$ with I different tasks in the application graph;
- $\sharp i_j$: Occurrence number of task i in the graph (process) j;
- r_{im}: number of tasks of type i executed per time unit on the processor m;
- r_j: the quantity of instances of graph (process) j executed per unit of time or flow;
- c_{im}: execution cost of task of type i on a processor m;
- CT is the total cost by flow r;
- N: the total number of instances of graph executed on the platform.

IV. Problem Formalization

In this part we detail the five cases identified previously.

a) Case i: In this case where one result of the workflow is computed by only one DAG and only one processor type can perform one task type. As in this case we have the following assumptions that a result is computed by only one DAG, and a processor type can perform only one task type, we introduce the following simplifications:

- Let c_{im} = c_i;
- Let r_{im} = r_i;
- $\sharp i_j$= $\sharp i$;
- r_j = r.

Therefore, the cost of the platform is:

$$CT = \sum_i \lceil \frac{\sharp i}{r_i} \cdot r \rceil c_i$$

It is also possible to find the flow that optimizes the use of the platform and thus the minize the cost r_{min} . This flow is the one that allows to ignore all the integer parts in the previous expression. We show how to obtain this value of r:
$\forall i, \frac{\sharp i}{r_i} \cdot r = k_i$ with k_i integer such that $\frac{a_i}{b_i} \times r_i = k_i$ such that $\lceil \frac{\sharp i}{r_i} \cdot r_{min} \rceil = \frac{\sharp i}{r_i} \cdot r_{min}$

we assume

$$\sharp i = a_i \cdot GCD(r_i, \sharp i)$$

$$r_i = b_i \cdot GCD(r_i, \sharp i)$$

for a given i we suppose that $r_{min} = b_i$, so
$\forall i, \quad r_{min} = \frac{r_i}{GCD(r_i, \sharp i)}$

for the set of i (to be integer everywhere):
$r_{min} = LCM_{i \in I}(\frac{r_i}{GCD(r_i, \sharp i)})$

b) Case ii: Several results are concurrently computed at the same time by processors such that each of them is able to perform only one task type. Moreover, every results are described by only one DAG. For this case, we find the total number of processors needed to execute each type of tasks on all graphs $\left(\lceil \frac{1}{r_i} \sum_j \sharp i_j \cdot r_j \rceil\right)$. Then we multiply it by the economical cost of each processors. To get the total execution cost of a throughput r ($CT(r)$), we sum the previous value on all tasks as it is expressed by the following formula:

$$CT = \sum_i \left(\lceil \frac{1}{r_i} \sum_j \sharp i_j \cdot r_j \rceil\right) c_i$$

We conclude that it is a similar case of the previous case except that we produce more than one result at the same time for this reason we introduce the term $\sum_j$ in the expression of cost to summarize on all instances of process.

It is possible to find the flow that optimizes the use of the platform. This maximum flow is the one that allows to ignore all non integer parts in the previous expression.

c) Case iii: Only one result defined by only one DAG, computed by processors performing only one task type. A same task can be done by processors of different types, of different speeds and of different costs.

A recursive expression computes the optimal cost of the platform according to the flow r of the DAG per unit of time. Let assume that α_{im} is the number of processors of type m used in the execution platform and allowing the treatment of a task of type i view to a flow r_{im}. The cost and the flow associated to the tasks of type i are therefore respectively $\sum_m \alpha_{im} c_{im}$ and $\sum_m \alpha_{im} r_{im}$.

A dynamic programming algorithm allows to give this optimal cost thanks to its recursive expression the tasks of type i, knowing that their flow is $r_i = \sharp i \times r$, with r the total flow of the workflow per unit of time. A sum on all tasks gives the global optimal cost of the factory offering a flow r:

$$C_i(r_i) = \quad min_{1 \leq m \leq M} \Big(c_{im} + C_i(w(r_i - r_{im}))\Big)$$

with $w(x) = 0$ if $x \leq 0$ and x otherwise.
The cost of the platform is then :

$$CT(r) = \sum_m C_m(\sharp i \times r)$$

d) case iv: It is the case where we have K possible DAGs for only one result, every type of tasks is being achieved by only one type of processor. The cost of a flow r is given by the following formula, with s_k is the flow associated to the DAG k. $r = \sum_k s_k$, $\#i_k$ is the number of tasks of type i in the process k and r_i is the flow attached to the task i:

$$CT(r) = \sum_i \Big(\lceil \frac{1}{r_i} \sum_k \sharp i_k \cdot s_k \rceil\Big) c_i$$

The difficulty is to find how to decompose the sum of the s_k. A sub-optimal solution could chooses first the low cost processors. An asymptotic approach to solve this case could also be considered.

e) case v: In this last case, a processor can compute several different tasks, the speed of execution of a task is different according to the type of the task and the type of the processor. Finally, the unique result to compute is defined by only one DAG. The expression of the constraints of this case drives the writing of the quadratic- program, represented below:

Objective $Min \sum_{t=1}^{|TP|} x_t c_t$

Subject To:

$$\begin{cases} \forall t \in TP, \quad \sum_{i=1}^{|T|} \frac{r_i^t}{r_{ti}} < 1 \\ \forall i \in T \quad \sum_{t=1}^{|TP|} x_t r_i^t = \sharp i \, r \\ \forall i \in T \quad et \quad \forall t \in TP, \quad \frac{r_i^t}{r_{ti}} < 1 \\ r_i^t \geq 0, \quad \forall i \in T, \quad \forall t \in TP \end{cases}$$

with:

- $\sharp i$: number of occurrence of task i in the process
- r: flow of the graph;
- r_{im}: the maximal number of tasks of type i executed on the processor of type m;
- r_i: the number of tasks of type i per unit of time;
- r_i^m: the number of tasks of type i executed by a processor of type m;
- x_m: the number of tasks of processors of type m used in the architecture platform;
- c_{im}: the execution cost of task of type i by a processor of type m.

In the following section, we give numerical results for the three first cases described before.

V. Computing of the minimal total execution cost of I independent tasks

A. *Computation of total execution cost for the first case*

In this experience we suppose that our application graph is composed of four tasks (T_1, T_2, T_3, T_4) as shown on figure Fig. 2, four processor/cells types (p_1, p_2, p_3, p_4) each of them execute a type of tasks with a different speed as shown on table Table 1. The cost of each processor/cell is respectively 15 euro, 20 euro, 25 euro and 45 euro. To find the needed number of each type of processor/cell we have to know the number of occurrences of each type of tasks in the application graph. So we suppose that the occurrence of each type of tasks is respectively 3, 2, 2 and 3. This DAGs is shown in the figure Fig. 2. We aim to produce 500 graphs per time unit. Recall that the total execution cost of throughput $r = 500$ is calculated according to the formula $CT(r) = \sum_i \lceil \frac{\sharp i}{r_i} \cdot r \rceil c_i$. The experience shows that the total execution cost for a throughput of 500 tasks per time unit is equal to: 670 euro; the size of the platform is 28 processors/cells among them 8 processors/cells of type p_1, 10 processors/cells of type p_2, 5 processors/cells of type p_3, and 5 processors/cells of type p_4. Also we provide the computation of the maximal throughput (r_{min}) that minimizes the total execution cost and we have obtained as result 200 tasks per time unit.

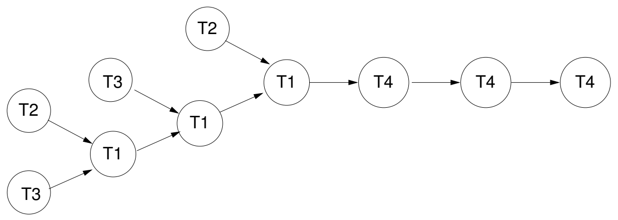

Fig. 2: Graph of tasks

We study the total execution cost in the interval of r from 0 to 500, we conclude that the function of cost increases obviously with r (see figure Fig. 3).

We consider another platform (Table 2) with the same input data as the previous experience the total execution cost become equal to 475 euro. The platform size is reduced to 18 processors: 5 processors/cells of type p_1, 5 processors/cells of type p_2 , 3 processors/cells of type p_3 and 5 processors/cells of type p_4 and $r_{min} = 1400$ tasks per time unit. The figure Fig. 3 also plots the cost of this platform for values of r from 0 to 500.

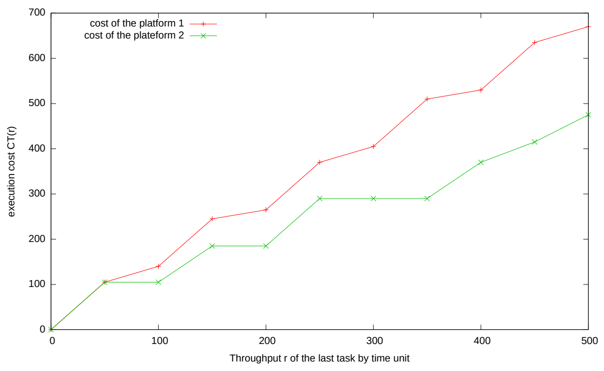

Fig. 3: Cost of 2 platforms with the graph Fig.2

TABLE II: Matrix of processor performances

Tasks types		p_1	p_2	p_3	p_4
	T_1	350	∞	∞	∞
	T_2	∞	200	∞	∞
	T_3	∞	∞	400	∞
	T_4	∞	∞	∞	350

B. *Computation of the total execution cost for the second case*

As mentioned above several products are collected at the same time by cells performing only one type of tasks, every product being defined by only one DAG. For this case the expression of cost is expressed as follow:

$$CT = \sum_i \Big(\lceil \frac{1}{r_i} \sum_j \sharp i_j \cdot r_j \rceil \Big) c_i$$

The architecture platform we use in this section is shown in the table Table 3.

TABLE III: Matrix of processor performances

Tasks types		p_1	p_2	p_3
	T_1	150	∞	∞
	T_2	∞	100	∞
	T_3	∞	∞	200

In this case, we want to produce two products from two identical graphs. The throughput of the first graph is $r_1 = 100$ tasks per time unit and the occurrence of tasks T_1, T_2, T_3 is respectively 3, 2 and 2. The throughput of the second graph is $r_2 = 150$ tasks per time unit and the occurrence of tasks T_1, T_2, T_3 is respectively $2, 2, 3$. The corresponding DAGs are shown respectively in the figure Fig. 4 and Fig. 5. We suppose that the cost of the processor/cell p_1 is 8 euros, the cost of the processor/cell p_2 is 5 euro and the cost of the processor/cell p_3 is 10 euro. We obtain a cost of 97 euro with a platform formed of 13 processors/cells distributed as follow:

4 processors/cells of type p_1, 5 processors/cells of type p_2, 4 processors of type p_3. The total number of executed task of each type is derived by the formula $r_i = \sum_{j=1}^{j=J} \sharp i_j r_j$, $\forall i$, where J is the number of graphs, and equals respectively 600 tasks per time of units of type T_1, 500 tasks T.U of type T_2, 650 Tasks T.U of type T_3 . The maximal throughput that minimizes the total execution cost is calculated by the formula $r_{min} = LCM_i(\frac{r_i}{(GCD(r_i, \sharp i_j))})$ and is equal to 600 tasks per time unit.

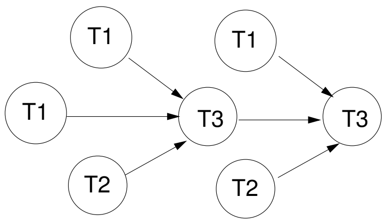

Fig. 4: Graph of tasks

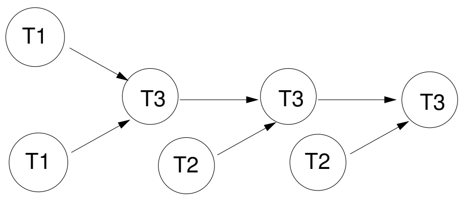

Fig. 5: Graph of tasks

We study the total execution cost in the interval [0..250] and we conclude that the total execution cost of r increases with r as shown in figures Fig. 4 and Fig. 5, knowing that the total throughput is equal to the sum of the throughput of all graphs ($r = r_1 + r_2$). We take another platform (Table 4) with the same assumption as for the previous experience. The computation shows that the total execution cost to produce two products with two processes described above is equal to 61 euro. The total number of processors/cells from this platform is equal to 8 processors/cells, like follow 2 processors/cells of type p_1, 3 processors/cells of type p_2 and 3 processors/cells of type p_3. The maximal throughput that minimizes the total execution cost is equal to 300 tasks per time unit. We study the variation of cost with the variation of r on the interval ([0..250]) and we plot it on the figure Fig. 6.

TABLE IV: Matrix of processor performances

		p_1	p_2	p_3
Tasks types	T_1	300	∞	∞
	T_2	∞	200	∞
	T_3	∞	∞	300

C. Computation of total execution cost of the third case

In this case, a task can be executed in processors/cells of different types with different cost and speed. The table Table 5 shows the platform used in this section. As shown before,

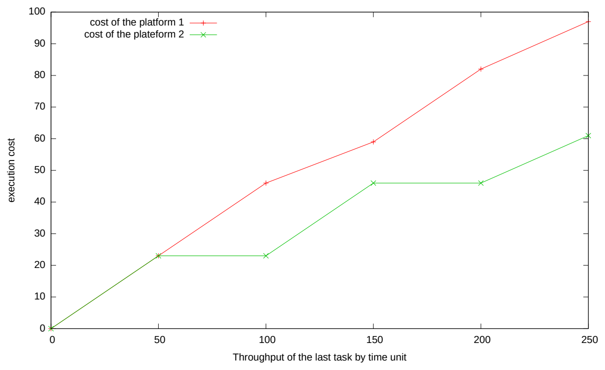

Fig. 6: Cost of platform in the case (ii)

if we select a processor/cell, we dedicated the processor/cell to the task for which it is selected. So, we can compute the cost of the platform as the sum of cost of each task type ($CT(r) = \sum_i CT(r_i)$). For the reason, we only give in this section numerical experiments for the optimal design for execution of the first task. Indeed, if we know the final throughput r, we can compute the throughput r_i concerning each task T_i ($r_i = \sharp i \times r$). The processor/cell performances to perform the first task T_1 are shown in the first line of the table Table 5. The cost of the processors/cells are respectively 10 euros, 20 euros and 40 euros.

TABLE V: Matrix of processor/cell performances: r_{im}

		p_1	p_2	p_3
Tasks types	T_1	10	50	130
	T_2	20	30	100
	T_3	30	50	120

The algorithm Fig. 1 shows the algorithm to find the greatest step size and the minimal number of iterations while studying the cost in the interval from 0 to r_i. The algorithm Fig. 2 explains the computing of the total execution cost in this case. As we explained before, this algorithm is a dynamic programing based algorithm.

Algorithm 1: cas3Parameters(r_i: array of integer, n: integer): integer

```
$r_i$: array of integer from which we compute the gcd.
$n$: the size of $r_i$
$k$: the number of iteration using in the algorithm 2
$d$: the greatest step to compute the platform design (algorithm 2)
begin
    $d \leftarrow$ gcdN($r_i, n$) /*gcd of $n$ integer in $r_i$    */
    $k \leftarrow d/r$
    if $(d\%r) > 0$ then
        $k \leftarrow r/d$
    else
        $k \leftarrow r/d + 1$
    endif
    return $d, k$
end
```

Algorithm 2: `TotalExecutionCost`(cost, r_i, C)

```
$w(x)$: the function which returns 0 or $x$ if $x < 0$ or not
$r_i[p]$: the throughput available on the processor $p$ for the type $i$
$k$: the number of iterations
$d$: the greatest step value that allows us to cover the possible values of
the throughput before $r$
$C[p]$: cost of the processor $p$
$cost[j]$: optimal cost of the platform for the throughput $j \times d$
$addedProc[j]$: processors added at the step $j$ for the optimal solution
with the current throughput $j \times d$
$procToAdd$: number of the processor to add at this step to the
platform to reach the optimal cost for the current throughput
begin
    for $j = 1$ to $k$ do
        $min \leftarrow cost[w(j - r_i[0]/d)] + C[0]$
        $procToAdd \leftarrow 0$
        for $i = 1$ to $n - 1$ do
            if $(cost[w(j - r_i[i]/d] + C[i]) < min)$ then
                $min \leftarrow cost[w(j - r_i[i]/d)] + C[j]$
                $procToAdd \leftarrow i$
            endif
        endfor
        $cost[j] \leftarrow min$
        $addedProc[j] \leftarrow procToAdd$ ;
    endfor
    /*reconstruction of the solution to this optimization problem */
    printf "throughput" $k \times d$ "with the cost" $cost[k]$
    $j \leftarrow k$
    while $j > 0$ do
        print ("processor" $addedProc[j]$, "next throughput"
        $j \times d - r_i[addedProc[j]]$)
        $j \leftarrow j - r_i[addedProc[j]]/d$
    endw
    return $addedProc, cost[k]$
end
```

We ran the two previous algorithms with two different throughputs, 450 and 300 tasks per time unit, and deduced the following:

- For a throughput of 450 tasks per time unit, we obtain the following results: the step to verify the cost is equal to 10, number of iterations is 45, the total execution cost is 150 euro, the size of platform is 5 processors/cells: 1 of type p_1, 1 of type p_2 and 3 of type p_3.
- For a throughput of 300 tasks per time unit, we obtain the following results: the step to verify the cost is equal to 10 , number of iterations is 30, the total execution cost is 100 euro, the size of platform is 2 processors/cells: 1 of type p_2, 1 of type p_3.

The figure Fig. 7 gives the optimal cost of the platform for the execution of task T_1 in the case (iii) in the experimental conditions described before.

The table Table 6 shows the optimal number of processors/cells of each type p_1, p_2 and p_3 each different target value of r_1 when optimizing the cost of the platform.

VI. Scheduling

In our study, we also interested ourselves to scheduling flows of micro-products on a set of processors/cells as we defined them. from the context description and according to the $\alpha|\beta|\gamma$ classification [?] of the scheduling problems, the problem is defined by: Ur—batch of intrees—Cmax. Indeed, the platform is heterogeneous as the execution times are not related and what we try to optimize is the Cmax – makespan or optimal execution time – of a flow of intrees.

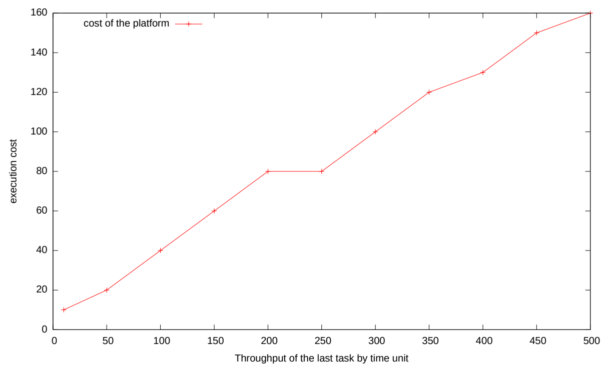

Fig. 7: Cost of the set of processors/cells which performed the task T_1 in the case (iii)

TABLE VI: Optimal number of processors/cells p_1, p_2 and p_3 for each throughput r_1

		number of p_i		
		p_1	p_2	p_3
throughput r_1	50	0	1	0
	100	0	2	0
	150	2	0	1
	200	2	1	1
	250	0	0	2
	300	0	1	2
	350	0	2	2
	400	1	0	3
	450	1	1	3
	500	0	0	4

This problem is known to be NP-Complete [?] and we must therefore either use an heuristic or modify the problem. The model presented at the global level offers to come closer to the models used in the distributed systems and more especially of the computation grids. The main differences rest on the type of the products we perform and on the functions of transportation, but the realization of a function on a micro-product or a computer data is modulated in the same way. With regard to the type of the products, we have seen that it is limited to the intree since we cannot duplicate a micro-component without new task. We are therefore in one under-case of computer applications. Concerning the function of transportation, it is possible in spite of all to assimilate the network topology to a complete network in the case of a portico or to a graph in the case of transporter.

From these observations, we looked for what could be the solutions brought to this problem and explored three approaches [?], [?]. The first is a dynamic approach that allocates the tasks dynamically to the processors, according to their availability. The second [?], [?], [?], [?] uses a heuristic of scheduling, based on the genetic algorithm, having good results for one DAG and we adapted it for the flow. The third [?], [?], [?], rests on a vision of the problem a little different since it is interested to the flow but it gives an optimal solution, we also adapted it to the management of flows. The assessment of these three techniques has been achieved by simulation and the results show without big surprise that, in a general case, the dynamic solution almost always offers optimal results, the heuristic solution is good on the small flows and the solution-oriented flow is good on the flows of important size. The interest of work resides therefore more in the borders that limit these techniques and the differences of performances gotten. Another interesting result is the dependence of the results with the flow of the graph that characterizes the process, some approaches giving better results for certain processes of equivalent flow size.

We work currently to the optimization of these different approaches and more especially the flows oriented one because several issues can be addressed our context to improve its results when the number of workflows or products to perform decreases.

VII. Conclusion

In this paper we have presented several results for the optimization of cost in the context of distributed heterogeneous platforms as grids or micro-factories. In particular we give formal results for the simpliest cases and a dynamic programming algorithm for a much complex case. We have illustrated through examples of different platforms and products.

As we have just seen, the micro-factory presents many domains of interests with regard to its optimization: as much to the level of its scheduling as of its control, because of the flexibility and the new constraints that are attached. The multi-levels aspects and the interactions between these levels defined new issues that will study in the future.

The automatic organization of micro-factories is not however in a phase of realization, even though some exploratory studies permit to consider its implementation in the coming years. The goal of our works is therefore to anticipate this realization in order to be able to come with it, to guide it and to put the potentialities of the view point of the production optimization forward.

References

[1] Y. Okazaki, N. Mishima, and K. Ashida, "Okazaki: Microfactory and micro machine tools," in *Reported in the first Korea-Japon Conference on Positionning Technology, Daejeon, Korea*, 2002.

[2] M. Tanaka, "Development of desktop machining microfactory," *Journal RIKEN Rev*, no. 34, pp. 46–49, April 2001.

[3] A. Ferreira, "Vers les micro-usines automatises du futur..." *J'automatise*, no. 18, pp. 47–51, 2001.

[4] E. Descourvières and al, "Towards automatic control for microfactories," in *5th Int. Conf. on Industrial automation*, Montréal, Québec, Canada, june 2007.

[5] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on computers. Vol 53, NO. 5, May 2004*, 2004.

[6] R.L.Graham and al, "optimization and approximation in deterministic sequencing and scheduling," *Ann. Discreat Math.*, vol. 4, pp. 287 – 326, 1979.

[7] E.Ilvarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *Journal of computer sciences*, vol. 3, no. 2, pp. 94–103, 2007.

[8] S. Diakité, J.-M. Nicod, and L. Philippe, "Comparison of batch scheduling for identical multi-tasks jobs on heterogeneous platforms," in *16th Conf. on Parallel, Distributed and Network-Based Processing*, Toulouse, France, 2008, pp. 374–378.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction To Algorithms 2nd Edition*, M. I. of Technology, Ed. MIT press, Jan 2001.

[10] M. Daoud and N. Kharma, "Gats: A novel ga-based scheduling algorithm for task scheduling on heterogeneous processor nets," in *Genetic And Evolutionary Computation Conference*, 2005.

[11] S. Y. Chen, "A robust genetic algorithm for structural optimization," *FEA-Opt Technology*, 2001.

[12] L. Min and W. Cheng, "Genetic algorithms for the optimal common due date assignment and the optimal scheduling policy in parallel machine earliness scheduling problems," *Robotics and computer-integrated manufacturing*, vol. 22, pp. 279 – 287, 2006.

[13] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms," in *IEEE Conference on Heterogeneous Computing*, 2004, pp. 296–302.

[14] D. N. Tahar, F. Yalaoui, C. Chu, and L. Amodeo, "A linear programming approach for identical parallel machine scheduling with job splitting and sequence dependent setup times." *International journal of production economics*, vol. 99, pp. 63 –73, 2006.

[15] V. Ramabhatta and R. Nagi, "An integrated formulation of manufacturing cell formation with capacity planning and multiple routings," *Annals of operations research*, vol. 77, pp. 79–95, 1998.