# Building partitioning graphs in Parallel-DEVS context for parallel simulations

**C. Herbez, E. Ramat**

ULCO LISIC
50 rue Ferdinand Buisson, BP 719
62228 Calais Cedex - France
[herbez , ramat]@lisic.univ-littoral.fr

**G. Quesnel**

INRA MIAT
24 chemin de Borde Rouge - Auzeville CS 52627
31326 Castanet-Tolosan cedex - France
gauthier.quesnel@toulouse.inra.fr

**ABSTRACT:** *With the emergence of parallel computational infrastructures at low cost, reducing simulation time becomes again an issue of the research community in modeling and simulation. This paper presents a method to improve simulation time through handling the structure of the model. This operation consists in partitioning the graph models based on several criteria. In this works, we use the DEVS formalism which is a discrete event formalism with a modular and hierarchical structure of models. To improve simulation time, we use partitioning method. We will present the partitioning method chosen to achieve this division and quantify the resulting time savings. Many tests are performed from graphs with different sizes and shapes.*

**KEYWORDS:** *DEVS, Simulation, Optimization, Graph, Partition, Multilevel.*

## 1 INTRODUCTION

Modelling and analysis of complex system dynamics is now a full science. Models derived therefrom are becoming increasingly complex in terms of components (sub-models) and interactions. Therefore, we need to develop both multi-modeling tools and efficient simulators. Indeed, the multimodelling is a response to the increase demand for coupling heterogeneous models. Model becomes an assembly of sub-models representing subsystems of the global system. Obviously, this process leads to the increase in computation demand and therefore, the increase of computation time. It is therefore important to think about the good use of new physical processor infrastructure (multicore and multiprocessor). Work in this area is it not new : includes all work around distributed simulation [**Chandy79, Chandy81**] but also work on parallel computing [**Fujimoto90**]. However, what interests us is the construction of an optimized organization of simulators as part of DEVS (Discrete Event Specification) [**ZKP00**]. The DEVS formalism is a possible answer to the multimodelling and has specific properties as discrete events and coupling models. The global model, called structure of the model in DEVS terminology, is a graph of coupled models. Recent works about the DEVS community through parallelization exist, for example, PhD thesis of Qi Liu and Ernesto Posse [**Liu10 , Posse08**] but also works around CD++ platform [**Wainer10**].

We will show how possible it is to design a graph of simulators which guarantees a high level of performance DEVS algorithms. This work involves the study of DEVS algorithms and graph partitioning. On the one hand, we describe Parallel-DEVS formalism and we show how models are structured. We will have a focus on the kernel of some algorithms to understand what we are looking for to optimize. Then we show how it is possible to partition the graph model to optimize the simulation algorithms. And to finish, various tests will be offered by illustration of the results.

## 2 DEVS MODELING AND SIMULATION

As we mentioned in the introduction, DEVS [**ZKP00**] is a high level formalism based on the discrete events for the modeling of complex discrete and continuous systems. The model is seen as a network of interconnections between atomic and coupled models. The models is in interaction via the exchange of timestamped events.

We will present Parallel-DEVS that is an extension of classic DEVS and that introduces the concept of simultaneity of events.

Parallel-DEVS [**Chow94**] extends the Classic DEVS [**Zeigler00**] essentially by allowing bags of inputs to the external transition function. Bags can collect inputs that are built at the same date, and process their effects in future bags. This formalism offers a solution to manage simultaneous events that could not be easily managed with Classic DEVS.

For a detailed description, we encourage to read the section 3.4.2 in chapter 3 and the section 11.4 in chapter 11 of Zeigler's book [**Zeigler00**].

Parallel-DEVS defines an atomic model as a set of input and output ports and a set of state transition functions:

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

With: $X$, $Y$, $S$ are respectively the set of input values, output values and sequential states

$ta : S \to \mathbb{R}_0^+$ is the time advance function

$\delta_{int} : S \to S$ is the internal transition function

$\delta_{ext} : Q \times X^b \to S$ is the external transition function

where:

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$

$Q$ is the set of total states,

$e$ is the time elapsed since last transition

$X^b$ is a set of bags over elements in $X$

$\delta_{con} : S \times X^b \to S$ is the confluent transition function, subject to $\delta_{con}(s, \emptyset) = \delta_{int}(s)$

$\lambda : S \to Y$ is the output function

If no external event occurs, the system will stay in state $s$ for $ta(s)$ time. When $e = ta(s)$, the system changes to the state $\delta_{int}$. If an external event, of value $x$, occurs when the system is in the state $(s, e)$, the system changes its state by calling $\delta_{ext}(s, e, x)$. If it occurs when $e = ta(s)$, the system changes its state by calling $\delta_{con}(s, x)$. The default confluent function $\delta_{con}$ definition is:

$$\delta_{con}(s, x) = \delta_{ext}(\delta_{int}(s), 0, x)$$

The modeler can prefer the opposite order:

$$\delta_{con}(s, x) = \delta_{int}(\delta_{ext}(s, ta(s), x))$$

Indeed, the modeler can define its own function.

Parallel-DEVS formalism is a kind of finite state automaton. Unlike at the conventional automata, it relies on the decomposition of the transition function in three sub-functions : internal, external and confluent. Moreover, the time is an integral part of the definition of a model via the function time advance, *ta*. This function sets the "timing" of simulation. It will be fundamental in the continuation of our work. To simplify our tests, in the next part, we consider this function as a constant in order to relate to a "discrete time" case.

Every atomic model can be coupled with one or several other atomic models to build a coupled model. This operation can be repeated to form a hierarchy of coupled models. A coupled model is defined by:

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\} \rangle$$

Where $X$ and $Y$ are input and output ports, $D$ the set of models and:

$\forall d \in D, M_d$ is a Parallel-DEVS model
$\forall d \in D \cup \{N\}, I_d$ is the influencer set of $d$ :
$I_d \subseteq D \cup \{N\}, d \notin I_d, \forall d \in D \cup \{N\}$,
$\forall i \in I_d, Z_{i,d}$ is a function,
the i-to-d output translation:
$Z_{i,d} : X \to X_d$, if $i = N$
$Z_{i,d} : Y_i \to Y$, if $d = N$
$Z_{i,d} : Y_i \to X_d$, if $i \neq N$ and $d \neq N$

The influencer set of $d$ is the set of models that interact with $d$ and $Z_{i,d}$ specifies the types of relations between models $i$ and $d$.

DEVS is an operational formalism. This means that the formalism is executable and thus it provides algorithms for its execution. These algorithms define the sequence of the different functions of the DEVS structure. Moreover, the atomic and coupled models are respectively associated with simulators and coordinators. The aim of simulators is to compute the various functions while the coordinators manage the synchronisation of exchanges between simulators (or coordinators in a hierarchical view). Many implementations reduce the hierarchy to only one level : a coordinator and several simulators. [**MuzyNut05**] introduce this type of structure. In this case, it is called flattening.

To manage the synchronization between simulators and/or coordinators, coordinators are based on a scheduler. This scheduler contains the dates when simulators must be awakened. These dates are computed using the time advance function *ta*. It is important to choose the right data structure and algorithms for the schedulers as it will be highly used.

## 3 INTEGRATION OF THE GRAPH PARTITIONING IN DEVS SIMULATION

Our approach consists to transform the structure of a model into another. The source model comes from the modeler activity (plant for agronomist for example). The destination structure is optimized for parallel simulation (i.e. with minimal hierarchy and minimal interaction between coupled model) with our partitioning algorithms. This work is possible thanks to the *closure under coupling* property of DEVS [**Zeigler00**].

For our portioning algorithms, we propose to use a method of undirected graph partitioning on the graph stemming from the simulation. Knowing that graph stemming from simulation is oriented, the first step is to convert this in a undirected flat graph.

The aim of this section is to make a short state of the art of graph by partitioning existing methods and to introduce the one we chose for Section 4.

### 3.1 Generality on the graph partitioning

The $k$-way graph partitioning is a constraints optimization problem. This involves cutting up a graph $G$ into $k$ subgraphs $\{G_1, G_2, \ldots, G_k\}$ in order to minimize one or more criteria often called "objective functions". This dividing is translated by the creation of $k$ subsets of vertices called partition. Given a graph $G = (V, E)$ where $V$ is a set of vertices and $E$ is a set of edges, $P_k = \{V_1, \ldots, V_k\}$ is a partition of $G$ if the following conditions are respected: (i) neither subset of $V$ which is a member of $P_k$ is not empty, (ii) the subsets of $V$ which are members of $P_k$ are pairwise disjoint. (iii) The union of all the elements of $P_k$ is $V$. In this case, the number $k$ was called cardinal partition or number of parts of the partition.

In our works, we are interested in a constrained partitioning problem. It consist in obtaining a partition whose parts have similar weight, this translated mathematically by a partitioning balance less or equal to 1.05. The partitioning balance of a partition $P_k$ is the ratio between the weight of the heaviest part and the average weight of parts of the partition. It's defined by the following formulas :

$$\text{weight}_{mean} = \lceil \frac{\text{weight}(V)}{k} \rceil \tag{1}$$

$$\text{balance}(P_k) = \frac{\max_i \text{weight}(V_i)}{\text{weight}_{mean}} \tag{2}$$

A partition is balanced if its partitioning balance is higher than 1. It is translated by a small difference between the weight of the heaviest part and the average weight.

The balance between the parties is not the only constraint which has to be respected, it must also minimize one of the objective functions presented in section 3.1.1. The choice of this function occur according to the type of problem to solve.

#### *3.1.1 The objective functions*

The objective functions quantify a criterion that we seek to respect when creating a partition. The researched goal during partitioning is to find the best partition that minimizes the studied objective function. Under the partitioning graph, the objective functions revolve around two concepts : cost cutting between the parts of the partition and weight of these parts.

Given a graph $G = (V, E)$ and two subsets $V_1 \subseteq V$ and $V_2 \subseteq V$, we define the cut cost, named *cut*, between this two parts as the weight sum of edges connecting $V_1$ and $V_2$ :

$$\text{Cut}(V_1, V_2) = \sum_{v_1 \in V_1, v_2 \in V_2} \text{weight}(v1, v2) \tag{3}$$

And the cut cost of a partition $P_k$ as the weight sum of edges between the partition parts :

$$\text{Cut}(P_k) = \sum_{i<j} \text{Cut}(V_i, V_j) \tag{4}$$

This objective function was already used by Brian Kernighan and Shen Lin in [**KerLin70**].

Another function allows simultaneous management the minimization of the cut cost and weight balance between the parties : the ratio cut. It's introduced by Yen-Chuen Wei and Chung-Kuan Cheng in [**WeiCheng89**].

$$\text{Ratio}(P_k) = \sum_{i=1}^{k} \frac{\text{Cut}(V_i, V - V_i)}{\text{weight}(V_i)} \tag{5}$$

In our works, we seek to minimize the objective function of the ratio cut.

#### *3.1.2 The main methods of the graph partitioning*

There are many methods related to the problem of graph partitioning. Each one has its features and functionality. The main categories of methods are: greedy methods, spectral methods, meta-heuristics and region expanding methods. The choice of the method depends on the nature of the problem and the objective function to solve. In our case, we seek a simple and effective method to minimize the ratio cut. We have opted for a graph definition by adjacency so, it is natural to seek a method that uses the concept of neighborhood to generate the partition. That is why we have chosen a method of expanding region: the Greedy Graph Growing Partitioning (GGGP).

The GGGP method is an amelioration of the "Graph Growing Partitioning" method introduced in [**KarKum98**]. GGGP is a bisection method, which aims to divide the set of vertices of the graph into two parts of equal weight using the concept of neighborhood. It works such as : two sets *Vertex_source* ($Vs$)

and *Vertex_destination* (*Vd*) are defined such as *Vs* contains all vertices of the graph and *Vd* is empty in the initial state. The algorithm starts by randomly to select a vertex in *Vs* and moves it in *Vd*. An array containing the adjacent vertices to those of *Vd* is created: *Vertex_neigh* (*Vn*). The process is to select the vertex of *Vn* whose gain with respect to the studied objective function is maximum. The selected vertex is moved from *Vs* to *Vd* and *Vn* is updated. The process stops when the weight of *Vd* is equal to the half weight of graph vertices. The algorithm of the GGGP method is presented by the algorithm 1.

**Algorithm 1:** Greedy Graph Growing Partitioning

**Function :** GGGP
**Input**: Graph $G(V, E)$
**Output**: $V_s$, $V_d$
**Initialization :**
  $V_s \leftarrow V$
  $V_d, V_n \leftarrow \emptyset$

Random selection of a vertex $v \in Vs$
Moving the vertex $v$ from $V_s$ to $V_d$
**while** $weight(V_d) < weight(V_s)$ **do**
  $V_n \leftarrow$ neighbor vertices of $V_d$
  $Gain \leftarrow$ gain of each vertex of $V_n$
  $v_g \leftarrow$ vertex of maximum gain
  Moving the vertex $v_g$ from $_V s$ to $V_d$

The presented method is a variation of the original GGGP method because the selection criterion of adjacent vertices is initially designed to reduce exclusively the edge-cut Furthermore, it is important to remember that it is originally designed for the bisection. It is necessary to perform a recursive application of the latter in the context of a k-way partitioning.

As Charles Edmond Bichot reports in his book [**BichSiar**], the major problem of this method is that it gives good results only on graphs of small size (less than 200 vertices). In order to apply this method on large graphs, it is necessary to establish a method to reduce its size without changing its structure. We propose to implement a multi-level scheme, presented in [**KarKum98**], to solve this problem.

### 3.2 Multilevel Graph Partitioning

The interest of multi-level is in this question : how can we create quickly a partition of a graph G of given size, knowing that it is very expensive to take care of each vertex one by one?

The answer to this question is in three phases of the multilevel introduced in the figure 1 :

- Coarsening: Generating of a reduced graph by successive matching vertices, while maintaining the nature of the original graph. Iterative process generating a graph base $\{G_1, \cdots, G_n\}$, where $G_1 = G$ the original graph and $G_n$ the contracted graph.
- Partitioning: The purpose of this step is to create a partition $P_k$ of the graph $G_n$. For this, the graph $G_n$ resulting from the step of contraction can be partitioned by using a heuristic of partitioning such as, for example, expanding region method.
- Uncoarsening: The refining step is to project the solution of the partition $P_k$ on the initial graph $G$. However, the partition can not be projected directly on the original graph on pain of getting a result of poor quality. An overall good solution on the graph $G_n$ may not be on the graph $G$. That's why it is necessary to realise a refinement after each projection. The solution obtained on the original graph remains globally and locally good.

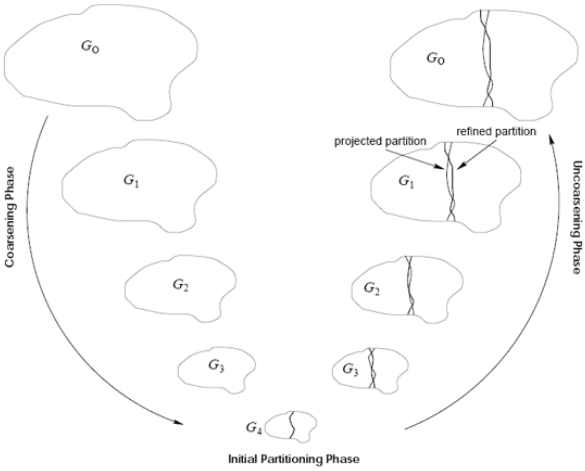


Figure 1 – Multilevel Graph Bisection

To implement our multilevel structure, we were inspired by algorithms present in literature and have contributed to their development for some of them. This is especially the case for the partitioning and uncoarsening phase algorithms. These algorithms are introduced in the following subsections.

#### *3.2.1 Coarsening Phase*

There are different methods of graph contraction orienting mainly around the aggregation of vertices. As part of our researches, we were interested in methods Random Matching and Heavy Edge Matching (HEM) introduced in [**KarKum**].

These methods have a common iterative structure. At each iteration, adjacent vertices are merged until

there are no more available vertices. The difference between these two methods lies in the choice of the neighboring vertex eligible for a merger. The algorithm 2 presents the iterative structure common to both methods of contraction.

**Algorithm 2:** Graph contraction for one iteration

**Function :** Contraction
**Input**: Graph $G_i(V_i, E_i)$
**Output**: Graph $G_{i+1}(V_{i+1}, E_{i+1})$
**Initialization :**
  $G_{i+1} \leftarrow G_i$
  V_list vertices set of $G_i$ sorted randomly

**for** *each vertex* $v_i$ *in V_list* **do**
  Search of adjacent vertices at $v_i$
  Select a neighbor $\rightarrow v_n$ **(*)**
  Remove edge $(v_i, v_n)$
  $weight(v_i) \; += \; weight(v_n)$
  **for** *each neighbor* $\alpha$ *of* $v_n$ **do**
    **if** *edge* $(v_i, \alpha) \in G_{i+1}$ **then**
      $weight(v_i, \alpha) \; += \; weight(v_n, \alpha)$
    **else**
      Creation of edge $(v_i, \alpha)$
    Remove edge $(v_n, \alpha)$
  Delete $v_n$
  Modification of Vertex_list

Now, let's talk about the specificity of each method: the selection criterion of the neighbor (represented by **(*)** in the algorithm 2). In Random Matching method, the selection criterion of neighbor is purely random. There are no specific regulations concerning the neighbor choice. Unlike it, the HEM method selects the neighbor whose weight of the edge is the strongest. The advantage of this approach is to regroup the vertices connected by edges whose weight is maximal. Thus, the edges of low weight should be most likely to be cut during the partitioning step.

We chose to use the HEM contraction method in the multilevel because it has the advantage to guide the partitioning towards a minimizing the edge-cut criterion.

#### *3.2.2 Partitioning Phase*

The partitioning process is accomplished using the GGGP method described in section 3.2.1. It has the advantage of being fast and efficient, but the disadvantage of having a highly dependent outcome of the chosen starting vertex. To relieve this, we propose an partitioning optimization approach related to starting vertex. This approach is described by the algorithm 3.

This consists in choosing *nbr_select* different starting vertices and to keep the best partition that minimizes

**Algorithm 3:** Optimization of partitioning

**Function :** Optim_Rec_GGGP
**Input**: Graph $G(V, E)$, Integer *nbr_select*
**Output**: Partition $P(k)$
**Initialization :**
  *V_list* vertices set of $G$ sorted randomly
  $crit \leftarrow \infty$ criterion to minimize
  $P_i(k) \leftarrow \emptyset$ partition with starting vertex $i$

**for** *i in 1 to nbr_select* **do**
  Partitioning for starting vertex V_list(i) $\rightarrow P_i(k)$
  Compute the criterion $\rightarrow$ *tmp_crit*
  **if** *tmp_crit* $<$ *crit* **then**
    $P(k) = P_i(k)$ recording of the partition
    *crit* = *tmp_crit*
  **else**
    $P_i(k) \leftarrow \emptyset$ destruction of the partition

the studied objective function. Considering the solution space that is the partition quality, the interest of this approach is to look into the solution space in order to find a good solution (not necessarily the best). For this, we propose to realise between 10 and 20 "smart" selections. We mean by this that the failure to take two vertices sensible to give a similar result.

**Algorithm 4:** Optimal selection

**Function :** Iterative_Optimal_Selection
**Input**: Graph $G(V, E)$, Integer *d_max*, Integer *nbr_select*
**Output**: *V_list*
**Initialization :**
  *V_list* vertices set of $G$ sorted
  *cpt* $\leftarrow 0$ counter

**while** *cpt* $<$ *nbr_select* **do**
  Random selection of a vertex $v \in$ *V_list*
  Search of vertices from a distance lower or equal to *d_max*
  Remove all vertices in V
  *cpt* $\leftarrow$ *cpt* $+ 1$

Starting from the postulate that a group of vertices locating in the same area may provide an equivalent partition, we set up a system of selection using a notion of distance between vertices. Distance is defined as the number of arcs defining a path (without cycle) between two vertices. Either the selection of a initial vertex, the selection policy implementation prohibits the selection of another vertices located at a distance less than *d_max*. This approach allows a depth route of the space solutions. However, if this one is poorly conditioned it may result a restriction of the solution space and thus be the cause of a lower quality. To avoid this type of overflow, we recommend to fix the *d_max* parameter between 2 and 3. Algorithm 4

represents the optimal selection method following a notion of distance.

#### 3.2.3 Uncoarsening Phase

The uncoarsening phase of the multilevel method consists in projecting on the initial graph, the partition of the contracted graph obtained during the partitioning phase. However, as George Karypis and Vipin Kumar find in [**KarKum95**], a locally optimal partition of a contracted graph $G_{i+1}$ is not necessary after the projection on $G_i$. Knowing that the contraction phase has built a family of graphs contracted $\{G_1, \cdots, G_n\}$, the uncoarsening phase consists in recursively projecting the partition of a contracted graph $G_{i+1}$ on his father $G_i$, then to apply on this partition a refining algorithm in order to maintain the quality (minimization of the studied objective function) of the partition obtained from the contracted graph $G_n$.

The method introduced is a variation of the original GGGP method because the selection criterion of adjacent vertices is initially designed to reduce exclusively the edge-cut. Furthermore, it is important to remember that it is originally designed for the bisection. It is necessary to perform a recursive application of the latter in the context of a k-way partitioning.

**Algorithm 5:** Refining by local displacement

```
Function : Refining_Local_Displacement
Input: Graph G(V, E), Partition P(k)
Output: Partition P(k)
Initialization :
    D ← ∅ cutting difference

while Ratio Cut decreases do
    for each partition parts do
        D ← calculation of cutting difference for
        each part vertices
        for each part vertex do
            if D(v) > 0 then
                Gain ← gain for each adjacent part
                g ← max(Gain) and α best adjacent
                part
                if g > 0 then
                    move v from current part to alpha
                else
                    Next vetex
            else
                Next vetex
```

The method that we have implemented is a local optimization algorithm based on algorithm of Kernighan-Lin [**KerLin70**]. It consists in moving successively vertices located on the periphery of a part. A vertex is considered on the periphery of a part $V_i$ if it has a common edge with a vertex that does not belong to $V_i$. For each periphery vertex of a part, we notice that the gain associated at the movement thereof toward each neighboring parts. The gain represents the difference between the old value of the studied objective function and the new one. If at least one gain is positive, the vertex is moved to the maximum gain part. This process is applied to each part of the partition and is repeated as long as one gain is realized. The structure of the method is described in algorithm 5.

The detection of eligible vertices for displacement (vertices of the periphery) is realised using the concept of cut difference introduced by Kernighan-Lin in [**KerLin70**]. To define what is the cut difference, we introduce the notions of internal and external cut. The internal cost $I(v)$ of a vertex $v$ of the part $V_i$ of the partition is defined as the sum of the weights of edges adjacent to $v$ whose the second vertex is in $V_i$ :

$$I(v) = \sum_{v' \in V_i} \text{weight}(v, v') \tag{6}$$

The external cost $E(v)$ of a vertex $v$ of $V_i$ is defined as the sum of the weights of edges between $v$ and the vertices not belonging to $V_i$ :

$$E(v) = \sum_{v' \in V - V_i} \text{weight}(v, v') \tag{7}$$

The cut difference $D(v)$ is the difference of cost between the external and the internal cost of the vertex $v$ :

$$D(v) = E(v) - I(v) \tag{8}$$

All vertex with a cut difference bigger than 0 is eligible for a moving. Instead, if it has a negative $D(v)$, it implies that it is not on the periphery or its movement is not likely to make a gain.

Results introduced in the section 4 are obtained from multilevel method using the HEM method for the coarsening phase, the GGGP method with draw distance and minimization of the ratio cut for the partitioning phase and refining method by local displacement using the cut difference for uncoarsening phase.

## 4 RESULTS

The objective of this section is to show the impact of the structure of the model in a DEVS simulation. We transform a modeler graph into a new one, optimized for simulation algorithms.

For this, we propose to make a comparison of simulation time between a classic modeler' graph which uses many coupled models and an optimized graph.

We will seek to quantify the gain (in simulation time) associated to the redefinition of the structure of the model by cutting in $k$ sub-models and determine an optimal parts number for the partition. To bring us

to the conditions of use of the partitioning method GGGP introduced in 3.1.2, we have parameterized our multilevel as follows:

- application of the HEM contraction until it reaches a graph of size 200,
- application of the GGGP with optimization of the start selection with a distance of 2 for a parts number varying from 2 to 100 every two,
- application of the refining method using the difference of cut.

But before observing the results obtained by applying the partitioning on the PDEVS models, the section 4.1 introduces the data on which were conducted the tests.

### 4.1 Data and hardware architecture for tests

This work is carried out in research project named Escapade (Assessing scenarios on the nitrogen cascade in rural landscapes and territorial modelling - ANR-12-AGRO-0003) funded by French National Agency for Research (ANR). We are particularly interested in one of the models of project : TNT (nitrogen and hydrologic pathways in topology) [**Beauj01**]. TNT offers to represent nitrogen transfers by a tree of pixels where each pixel represents an area and the different soil layers. Exchange processes and nitrogen transformations are described at this spatial scale. The number of spatial elements is several tens of thousands to the studied catchment area. The following tests and results are therefore based on both graphs used in TNT and graphs generated with similar properties to those of TNT.

The tests were performed on a C++11 program that implements the Parallel-DEVS abstract simulators. It was compiled with gcc 4.8 on Linux. The hardware architecture was based on an intel core i7 quad-core 2.8 GHz processor and 16 GB of memory.

### 4.2 Results observation

The results introduced in this section are in the form of relative gain curves. The gain as a percentage, is the ratio between the time obtained by applying the method of cutting and the reference time. Our interest is focused on the gain including the partitioning time of the graph ($RG$) and on the pure gain ($PRG$), that is to say the gain does not include the partitioning time.

These relative gains are obtained from the following equations:

$$time_{diff} = time_{Simu\_ref} - time_{Simu\_parti} \quad (9)$$

$$RG = \frac{time_{diff}}{time_{Simu\_ref}} * 100 \quad (10)$$

$$time_{diffp} = time_{diff} - time_{Parti} \quad (11)$$

$$PRG = \frac{time_{diffp}}{time_{Simu\_ref}} * 100 \quad (12)$$

Where, $time_{Simu\_parti}$ is the simulation time obtained by applying the structure original cut, $time_{Simu\_ref}$ is the reference time obtained by applying the original structure and $time_{Parti}$ is the partitioning time.

A simulation with 8000 models is used to realise this testing phase. Each simulation is subject to a number of iterations and has a duration. This time is arbitrarily set at 40 time units. It enables to capture the effects of optimization. The result is shown in figure 2.

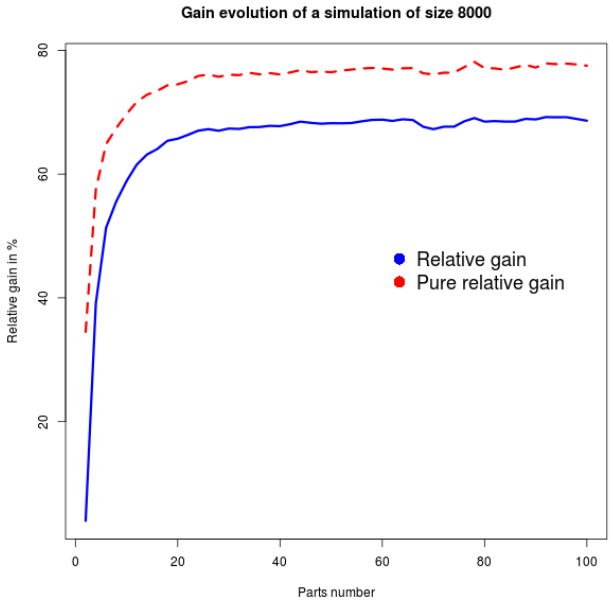


Figure 2 – Relative gain evolution in % in function of the number of part for a simulation duration of 40 e.t.

As part of this simulation, the maximum gain obtained is around 70%. Regarding the pure gain, i.e. the gain without the partitioning time, it is close to 80%. It is interesting to take a look at the pure gain because for a large simulation, partitioning time becomes negligible versus the simulation time. Concerning the evolution of the gain in function of the number of parts, we can observe that there are gain from the first bisection. It is in the order of 4% with partitioning and 34% without. Furthermore, we can observe an exponential increase of gain to a parts number close to 20%. The gain varies from 4 to 65% with partitioning and from 34 to 77% without. Beyond that, the gain stagnates to achieve 80%. The parts

number corresponding to this change of state is called optimal number of parts.

The origin of this gain comes from the minimization of the number of message exchange between models.

The gain obtained by the implementation of the hierarchical structure is translated by a reduction factor of the simulation time. Given $P$ the percentage of gain obtained by applying the hierarchical structure and $\alpha$ the factor of time reduction, $\alpha$ is obtained as follows:

$$\alpha = \frac{100}{100 - P} \tag{13}$$

We consider a simulation with a gain of about 70 %, it means that the reduction factor of time is approximately $\frac{100}{100-70} = 3.33$. It implies that in this case, the hierarchical structure provides a simulation time 3.33 times faster than for the simple structure. However, this result is not only an indicator of the performances of the algorithms. Remembering that we work with simulations not containing neither important calculations nor heavy. The implementation of parallelization becomes essential whenever calculations are introduced in the models. It is hoped that the objective functions used for this preliminary work will lead to similar results in a parallel context.

| **Execution Time** | **10** | **20** | **40** | **80** | **160** |
|---|---|---|---|---|---|
| $gain_{min}$ | −67 | −24 | 4 | 20 | 28 |
| $gain_{max}$ | 33 | 56 | 70 | 77 | 81 |
| $gain_{min}$ pure | 27 | 30 | 34 | 36 | 37 |
| $gain_{max}$ pure | 60 | 72 | 78 | 82 | 84 |

Table 1 – Gain evolution in function of the execution time (E.T.)

We will now detail the results and analyze the impact of each step. Different times are measured in order to observe their impact on the gain provided by our cutting. Table 1 gives the values of minimum and maximum gain (in %) recorded for different execution time on a simulation of size 8000.

This table shows that the more a simulation is long, the more the maximum gain associated tends to 85 %. This is explained by the fact that the more a simulation is long, the more the number of operations on schedulers is important. Putting in place a cutting of schedulers allows to reduce the cost of these operations, especially when simulation is long. Moreover, this table allows to confirm that for the long simulations,the time partitioning becomes negligible. This is explained by the fact that the more the execution time increases the more the difference between the maximum gain and the maximum pure gain decreases (30 for a E.T of 10 versus 3 for a E.T of 160).

A similar study was conducted to observe the impact of the number of models on the pure maximum gain that can be expected. Here is a comparison of pure gains for simulations where the number of models varies from 1000 to 16,000 simulators. The execution time is always set at 40 time units. The results are given in figure 3.

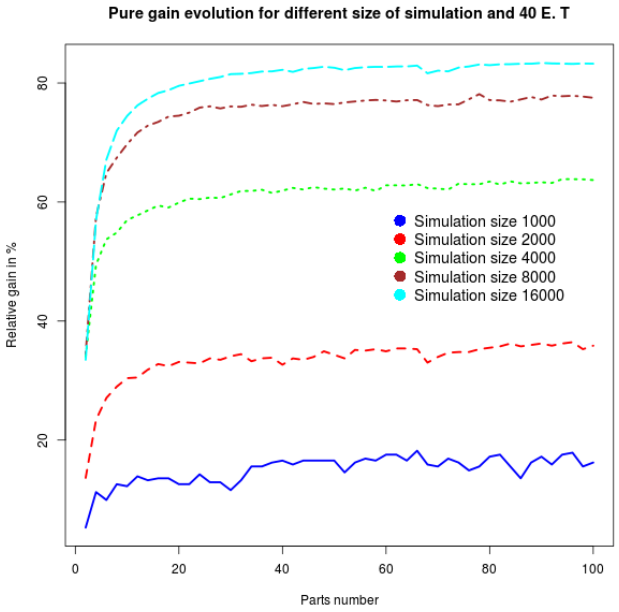


Figure 3 – Comparison of the maximum gains for different sizes of simulation

Table 2 summarizes the values of minimum and maximum gain pure, and the optimal number of part for the different numbers of models.

| **Size** | **1000** | **2000** | **4000** | **8000** | **16000** |
|---|---|---|---|---|---|
| $gainp_{min}$ | 5 | 13 | 34 | 35 | 33 |
| $gainp_{max}$ | 18 | 36 | 64 | 78 | 83 |
| $Opt\_part$ | 24 | 30 | 30 | 26 | 32 |

Table 2 – Gain evolution in function to the simulation size

These results show that more the number of models is big, the more the associated gain is important. It is perfectly consistent with the results observed for execution time important. These results tend to confirm that 85 % is the maximum pure gain that can be achieved by applying the hierarchical structure. In addition, we found that the optimal number of parts for all simulations studied is neigh to 30. Beyond, the increase is not significant, it is not necessary to take a larger number of parties unless in search of maximum gain.

## 5 CONCLUSION

This paper presents a method to improve simulation time. This method consists in partitioning a DEVS graph models in a optimized DEVS graph models for the simulation i.e. by minimizing number of message exchange. This work is possible thanks to the *closure*

*under coupling* property of the DEVS formalism.

This article highlights the importance of integrating the graph partitioning within DEVS simulations to reduce the execution time. This time saving is obtained by a reconstruction of a two-level hierarchy of the original model. When building subgraphs, it was essential to choose an objective function that reflects both the balancing of the parties and the minimization of the links between the parties.

Concerning the results, the maximum gain observed (close to 85%) is purely theoretical, it is valid only for simulations where the computation time of the models is negligible. The optimized properties in this context will be then exploited with a Parallel-DEVS parallelized kernel.

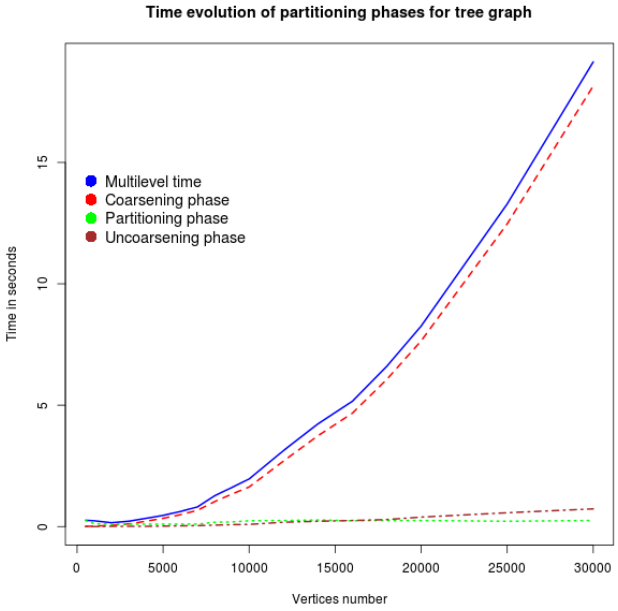


Figure 4 – Multilevel and phases time in function of the simulation size

Indeed, the parallelization is based on the quality of the partitioning that reflects the balance of execution time between subgraphs and the amount of data that passes between the parties.

The short-term prospects of our works follow the optimization of multilevel phases. At the moment, only the contraction phase has not yet improved and as shown in Figure 4 the time partitioning is largely affected by the phase of contraction. The purpose of this approach is the implementation of parallelization on parallel architectures (multi-processors / multi-core) and the quantification of the gains for expensive simulations.

## REFERENCES

**Beauj01** V. Beaujouan, P. Durand and L. Ruiz, *Modelling the effect of the spatial distribution of agricultural practices on nitrogen fluxes in rural catchments*, Ecological Modelling, n. 137 (1), pp. 93-105, 2001.

**BichSiar** C. H. Bichot, P. Siarry, 2010. *Graph Partitioning*, p. 29-64

**Chandy79** , K. M. Chandy and J. Misra, *Deadlock absence proofs for networks of communicating processes*, Information Processing Letters, vol. 9, n. 4, pp 185–189, 1979.

**Chandy81** , K. M. Chandy and I. Misra, *Asynchronous Distributed Simulation via a Sequence of Parallel Computations*, Communications of the ACM, vol. 24, n. 4, pp 198–206, 1981.

**Chow94** A.C.H. Chow and B.P. Zeigler, *Parallel DEVS: a parallel, hierarchical, modular, modeling formalism*, Proceedings of the 26th conference on Winter simulation, pp. 716-722, 1994, Orlando, Florida, United States.

**Fujimoto90** R. M. Fujimoto, 1990. *Parallel discrete event simulation*, Communications of the ACM - Special issue on simulation Volume 33 Issue 10, p. 30-53.

**KarKum** G. Karypis, V. Kumar, *Multilevel graph partitioning schemes.*

**KarKum95** G. Karypis, V. Kumar, 1995. *Analysis of Multilevel Graph Partitioning. In Proceedings of Supercomputing*, p. 44-50

**KarKum98** G. Karypis, V. Kumar, 1998. *A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.* SIAM Journal of Scientific Computing, p. 359–392.

**KerLin70** B. W. Kernighan, S. Lin, 1970. *An Efficient Heuristic Procedure for Partitioning Graphs. Bell System Technical Journal*, p. 291–307.

**Liu10** Qi Liu, *Algorithms for Parallel Simulation of Large-Scale DEVS and Cell-DEVS Models*, PhD thesis, Ottawa-Carleton Institute for Electrical and Computer Engineering, Department of Systems and Computer Engineering Carleton University Ottawa, Ontario, Canada, 2010.

**MuzyNut05** A. Muzy, J.J. Nutaro, 2005. "DEVS & DSDEVS abstract simulators", 1st Open International Conference on Modeling and Simulation (OICMS, Clermont-Ferrand), 273-279.

**Posse08** Ernesto Posse, *Modelling and simulation of dynamic structure discrete-event systems*, PhD thesis, School of Computer Science, McGill University, 2008.

**Wainer10** Gabriel A. Wainer, Qi Liu, and Shafagh Jafer. *Advanced parallel simulation of DEVS models in CD++*, chapter 9, page TBD. Taylor and Francis, 2010. Authors : G. Wainer, P. Mosterman Eds, Book : Discrete-Event Modeling and Simulation : Theory and Applications.

**WeiCheng89** Y.C. Wei, C.K. Cheng, 1989. *Towards efficient hierarchical designs by ratio cut partitioning. In Proceedings of the IEEE International Conference on Computer-Aided Design*, p. 298–331.

**Zeigler00** B. P. Zeigler and D. Kim and H. Praehofer, *Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2000.