Building partitioning graphs in Parallel-DEVS context for parallel simulations

C. Herbez
ULCO LISIC
50, rue Ferdinand Buisson
BP 719 62228 Calais Cedex
France
herbez@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

É. Ramat
ULCO LISIC
50, rue Ferdinand Buisson
BP 719 62228 Calais Cedex
France
ramat@lisic.univ-littoral.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 work, 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 methods. 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.

Author Keywords

Simulation, Optimisation, Graph, Partition, Multilevel, GGGP, DEVS

ACM Classification Keywords

I.6.8 SIMULATION AND MODELING: Discrete event, Distributed; G.2.2 DISCRETE MATHEMATICS: Graph Theory

INTRODUCTION

Modeling and analysis of complex system dynamics is now a full science. Models derived therefrom are becoming increasingly complex in terms of components (submodels) and interactions. Therefore, we need to develop both modeling tools and efficient simulators. However, this process leads to the increase in computation demand and therefore, the increase of computation time.

Multi-modeling is a response to the increased complexity of the models [13]. The multi-modelling approach allows to couple heterogeneous models (i.e. each models can use different formalisms). DEVS [16] (*Discrete Event Specification* is a good candidate to develop the multi-modeling approach[12]. DEVS is a discrete events modeling and simulation theory with a hierarchical approach. The global model, called *structure of the model* in DEVS terminology, is a graph of coupled models.

To improve simulation computation time, the DEVS community provides several simulation algorithms and softwares based on classical distributed and parallel computing algorithms or techniques [2, 3, 5] for example, in the CD++ platform [14].

In this paper, we propose to transform and optimise the structure of the model provided by the modeler into a new one, optimised for the simulation. These works show how possible it is to design a graph of simulators which guarantees a high level of performance DEVS algorithms.

In the first part, we describe the 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 optimise. Then we show how it is possible to make a partition of the graph model to optimise the simulation algorithms. And to finish, various tests will be offered by illustration of the results.

DEVS MODELING AND SIMULATION

As we mentioned in the introduction, DEVS [16] is a high level formalism based on the discrete events for the modeling of complex discrete and continuous systems. The model is a network of interconnections between atomic and coupled models. These models are in interaction via time-stamped events exchanges.

In this section, we present the Parallel-DEVS (PDEVS) formalism [4]. PDEVS is an extension of the classic DEVS. It introduces the concept of simultaneity of events 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 en-

courage to read the section 3.4.2 in chapter 3 and the section 11.4 in chapter 11 of Zeigler's book [16].

PDEVS 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 δ_{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 δ_{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.

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 PDEVS 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.

PDEVS 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 PDEVS 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). A PDEVS feature is the possibility to parallelize the set of events to reduce time calculation.

The association between the modeler's structure of the model and the simulator hierarchy may be underperformant and/or not easily suitable to distribute or to parallel the simulation over a calculator:

- The coordinators sub-graphs can be not balanced i.e. schedulers sizes may be not balanced.
- Atomic models and simulators with high output frequency must be move closer to reduce overhead of events between simulator and the hierarchy of coordinators.
- In the same way, atomic models with expensive internal or external transition (in term of computation time) must be placed alone to use more processing resources.

Figure 1 shows an simple example of an optimised graph.

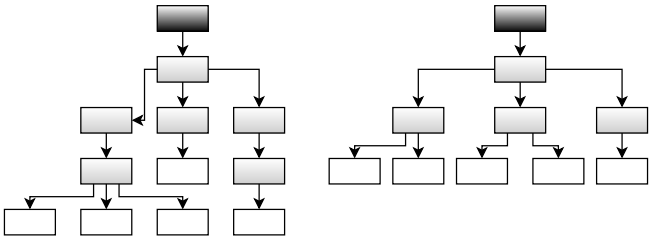

Figure 1. In left part of the picture, we show a typical DEVS model provides by modeler. It contains fives atomic models and six coupled models and a root coordinator. In right part, a optimised graph for simulation with three processors.

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 [16]. This property formally describes the coupled model is equivalent to an atomic model. Thus an atomic model can be move into a new

coupled model and all the hierarchy of coupled model can be merge into a unique coupled model.

For our partitioning 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 existing partitioning methods and to introduce the one we chose for Section Results.

Generality on the graph partitioning

The k-way graph partitioning is a constraints optimisation problem. This involves cutting up a graph G into k sub-graphs $\{G_1, G_2, \ldots, G_k\}$ 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.

In our works, we are interested in a constrained partitioning problem. It consist in obtaining a partition whose parts have similar weight. The weight of a partition P_k is defined by the following formula :

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

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

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 :

$$\text{Cut}(V_1, V_2) = \sum_{v_1 \in V_1, v_2 \in V_2} \text{weight}(v1, v2) \tag{2}$$

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

$$\text{Cut}(P_k) = \sum_{i<j} \text{Cut}(V_i, V_j) \tag{3}$$

This objective function was already used by Brian Kernighan and Shen Lin in [9].

Another function allows simultaneous management the minimization of the cut cost and weight balance between the parts : the ratio cut. It's introduced by Yen-Chuen Wei and Chung-Kuan Cheng in [15].

$$\text{Ratio}(P_k) = \sum_{i=1}^{k} \frac{\text{Cut}(V_i, V - V_i)}{\text{weight}(V_i)} \tag{4}$$

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

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 neighbourhood 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 [8]. 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 neighbourhood. 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 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 [1], 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 [8], to solve this problem.

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 2 :

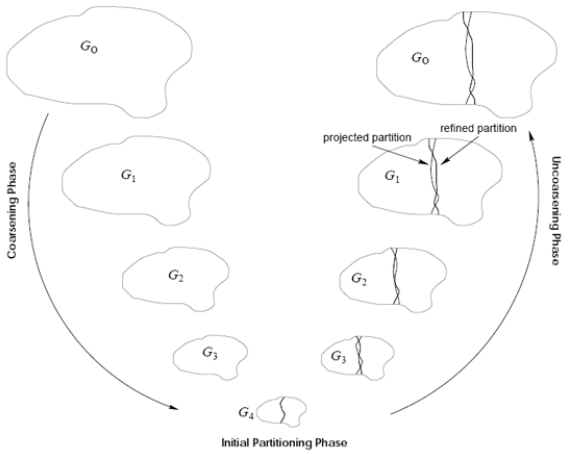

Figure 2. Multilevel Graph Bisection

- 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.

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.

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 [7].

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 neighbouring vertex eligible for a merger. The algorithm 1 presents the iterative structure common to both methods of contraction.

Algorithm 1: Graph contraction for one iteration

Function : Contraction
Input: Graph $G_i(V_i, E_i)$
Output: Graph $G_{i+1}(V_{i+1}, E_{i+1})$
Initialisation :
  $G_{i+1} \leftarrow G_i$
  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 neighbour $\rightarrow v_n$ **(*)**
  Remove edge (v_i, v_n)
  $weight(v_i) \; += \; weight(v_n)$
  for *each neighbour* α *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, α)
    Remove edge (v_n, α)
  Delete v_n
  Modification of Vertex_list

Now, let's talk about the specificity of each method: the selection criterion of the neighbour (represented by **(*)** in the algorithm 1). In Random Matching method, the selection criterion of neighbour is purely random. There are no specific regulations concerning the neighbour choice. Unlike it, the HEM method selects the neighbour 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.

Partitioning Phase
The partitioning process is accomplished using the GGGP method described in last section. 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 optimisation approach related to starting vertex. This approach is described by the algorithm 2.

This consists in choosing *nbr_select* different starting vertices and to keep the best partition that minimizes 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 2: Optimisation of partitioning

Function : Optim_Rec_GGGP
Input: Graph $G(V, E)$, Integer *nbr_select*
Output: Partition $P(k)$

Initialisation :
- 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

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 3 represents the optimal selection method following a notion of distance.

Algorithm 3: Optimal selection

Function : Iterative_Optimal_Selection
Input: Graph $G(V, E)$, Integer d_max, Integer *nbr_select*
Output: V_list

Initialisation :
- 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
- $cpt \leftarrow cpt + 1$

Uncoarsening Phase
As George Karypis and Vipin Kumar find in [6], 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.

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 4: Refining by local displacement

Function : Refining_Local_Displacement
Input: Graph $G(V, E)$, Partition $P(k)$
Output: Partition $P(k)$

Initialisation :
- $D \leftarrow \emptyset$ cutting difference

while *Ratio Cut decreases* **do**
- **for** *each partition parts* **do**
  - $D \leftarrow$ calculation of cutting difference for each part vertices
  - **for** *each part vertex* **do**
    - **if** $D(v) > 0$ **then**
      - $Gain \leftarrow$ gain for each adjacent part
      - $g \leftarrow \max(Gain)$ 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 optimisation algorithm based on algorithm of Kernighan-Lin [9]. 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 neighbouring 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 realised. The structure of the method is described in algorithm 4.

The detection of eligible vertices for displacement (vertices of the periphery) is realised using the concept of cut difference introduced by Kernighan-Lin in [9]. To define what is the cut difference, we introduce the notions of internal and external cut. Let v a vertex V_i, the internal cost $I(v)$ is defined as the sum of the weights of edges adjacent to v whose the second vertex is in V_i and the external cost $E(v)$ is defined as the sum of the weights of edges between v and the vertices not belonging to V_i.

$$I(v) = \sum_{v' \in V_i} \text{weight}(v, v') \quad (5)$$

$$E(v) = \sum_{v' \in V - V_i} \text{weight}(v, v') \quad (6)$$

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

$$D(v) = E(v) - I(v) \quad (7)$$

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

The objective of this section is to show the impact of the structure of the model in a DEVS simulation. Thanks to the closing under coupling DEVS' property, we can change the original modeler's structure into a new one, improved for simulation algorithms.

For this, we propose to make a comparison of simulation time between a set of classic modeler's' graphs with optimised graph for simulation. We choose the following structures: grid, linked and tree.

The benchmark characteristics:

- The *internal transitions* of all atomic models computes the linkpack benchmark for performing numerical linear algebra and force the processor to compute data. The calculations are limited to 3m̃s.
- Only the coordinator (child of the root coordinator) uses the PDEVS threading mode with all threads available on hardware processor.
- 37 replicas are run to avoid benchmarks problems (i/o access, processor affinity etc.).
- We vary the method from orig (original modeler's graph), GGGP and coarsening and parts number 2 to 16 every two.

To bring us to the conditions of use of the partitioning method GGGP, we have parameterized our multilevel as follows:

- coarsening method HEM until size 200,
- partitioning method: modeler, gggp and coarsening,
- application of the refining method using difference.

But before observing the results obtained by applying the partitioning on the PDEVS models, the following section introduces the type of hardware and software architecture used for tests.

Hardware and Software architecture for tests

The tests were performed on a PDEVS simulation kernel called *Echll* (A C++ open-source software available at `https://github.com/vle-forge/Echll`). *Echll* is a part of the modeling and simulation software suite VLE [11] and provides several DEVS extensions. It will replace the existing simulation kernel of VLE. The simulations were done on an SMP cluster node equipped with 20 Intel XEON ES 2670 processors at 2.5 Ghz and 256 GB of memory.

Datas presentation

The tests presented in the following section are realised from three types of graphs derived from the flattening of different hierarchical models. This graphs have a particular structure, they were designed to represent a fluid flow in a watershed in the context of the project in which we work.

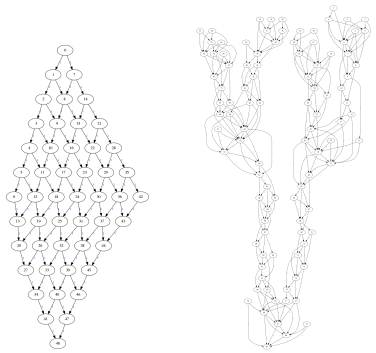

Figure 3. Example of grid graph and tree graph of little size

The graph of grid type shown in Figure 3 models a uniform fluid flow starting from a single source to a single outlet. Each vertex communicates with its two neighbors of bottom until reach the last vertex.

The graph of tree type shown in Figure 3 is composed of several levels. Each level consists of one or more branches where the vertices are connected to each other following a single direction. The branches are branched until reaching the single outlet. Unlike the grid, the tree graph has multiple sources tops (n sources by branches).

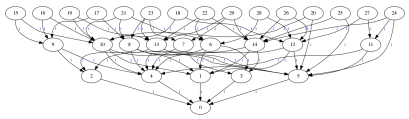

Figure 4. Linked graph example of little size

The graph of linked type shown in Figure 4 consists of several levels. It has the particularity of being hyper connected because each vertex of a level is connected to 2 or 3 vertices of the lower level. The vertices of the penultimate level are connected only to the outlet. As the tree graph, the linked graph have several sources vertices that are the vertices of the first level.

For the test phase, we assume that the execution time is equal for each model (vertices weight equal and fixed at 1) and the cost of message transfer from a simulator to another is equivalent to the entire simulation (edge weight equal and fixed at 1).

Results observation

The aim of this part is to compare the simulation durations for a parallel mode, i.e. when the coordinator and this childs are parallelized. The comparison is made on the choice of the coupled model hierarchy and structure. Two kinds of approaches are studied for the choice of the structure : original modeler graph and partitioning methods, used to coarsening or GGGP method

To highlight this, we propose to compare the simulation durations obtained for each of the following graphs: tree graph of size 20000, linked graph of size 10000 and grid graph of size 4900 (the graph size correspond to the simulation model number). For each graph, we observe the evolution of the simulation time in seconds in function of the number of parts (submodels) for each partitioning methods. These methods are represented by the following captions:

- _____ (solid line) coarsening method
- - - - - (dashed line) GGGP method
- -.-.-.- (dashed and point line) original model

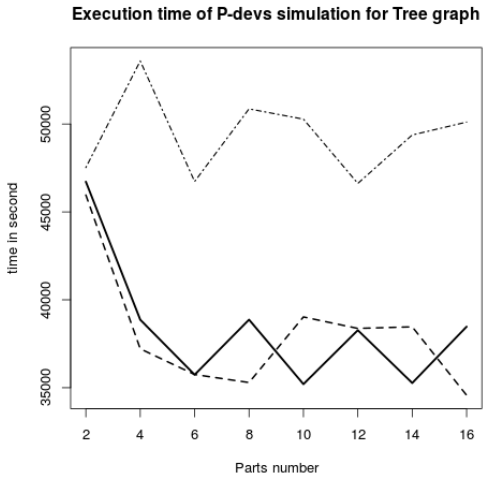

Figure 5. Evolution of parallel simulation time against parts number for a tree graph

The figure 5 present results for tree graphs with 20,000 models. In this figure, we observe that partitioning methods are 1.5 times faster than the original models. However, we can see that the coarsening and GGGP methods offer similar results. This is due to the fact that these methods create a partition that is generally respect to the balance between the parts. Therefore, execution time results are close to each part, which is not necessarily the case for original models.

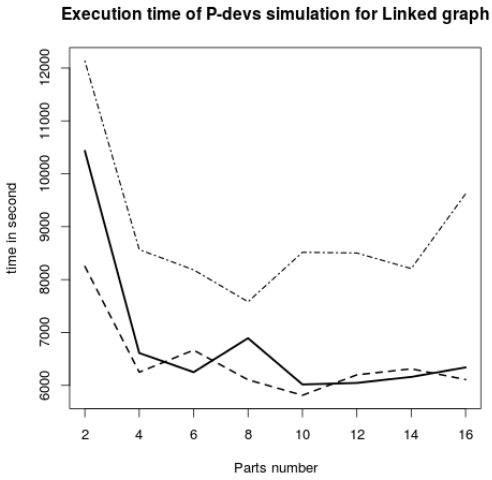

Figure 6. Evolution of parallel simulation time against part number for a linked graph

In the case of strongly connected graphs, observation is the same. The simulation duration is shorter (factor 1.2) for partionning methods compared to original models. It's a bit weaker than the previous case.

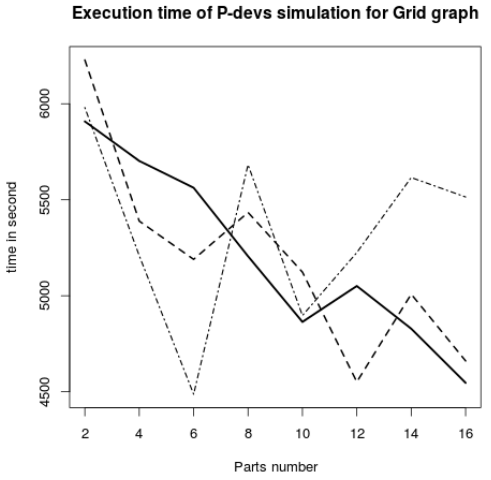

Figure 7. Evolution of parallel simulation time against part number for a grid graph

In grid case, results are completely different. Indeed, no method stands out. It is a special case: precedence graph implies a spread of cascading events and one model is responsible for the spread. However, remember that for our simulations, the cost of message transfer is fully insignificant compared to computation time. The partionning methods, including GGGP, are designed to minimize the transfer costs between the parts while balancing at best. Therefore it can be expected to a significant improvement of the results of partitioning methods for simulations with large transfer cost.

We also used the devstone bench [10] to test our method. Results are similar to those obtained with our graphs.

CONCLUSION

This paper presents a method to improve simulation time. This method consists in partitioning a DEVS graph models in a optimised DEVS graph models for the parallel simulation, i.e. by minimizing number of message exchange and by balancing of models execution time. This minimizes the transfer of messages between parts and have an equivalent execution time for each of them.

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 and by parallelization of the root child and his coupled models children.

When building sub-graphs, it was essential to choose an objective function that reflects both the balancing of the parts and the minimisation of the links between the parts.

Concerning the results, we observe that the gggp and coarsening partitioning methods provide a better structure than the random partitioning method which is close to the structure provided by a modeler. What affects the time distributed simulation which are between 1.5 and 2 times faster for a simulation with inexpensive message transfers.

ACKNOWLEDGMENTS

This work is carried out in research project named Escapade (Assessing scenarios on the nitrogen cascade in rural landscapes and territorial modeling - ANR-12-AGRO-0003) funded by French National Agency for Research (ANR).

We are grateful to the genotoul bioinformatics platform Toulouse Midi-Pyrenees for providing help and computing resources.

REFERENCES

1. Bichot, C.-E. *A Partitioning Requiring Rapidity and Quality: The Multilevel Method and Partitions Refinement Algorithms*. John Wiley & Sons, Inc., 2013, 27–63.
2. Chandy, K. M., and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Software Eng. 5*, 5 (1979), 440–452.
3. Chandy, K. M., and Misra, J. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM 24*, 4 (1981), 198–206.
4. Chow, A. C. H., and Zeigler, B. P. Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th conference on Winter simulation* (Orlando, Florida, United States, 1994), 716–722.
5. Fujimoto, R. M. Parallel discrete event simulation. *Commun. ACM 33*, 10 (Oct. 1990), 30–53.
6. Karypis, G., and Kumar, V. Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, ACM (New York, NY, USA, 1995).
7. Karypis, G., and Kumar, V. Multilevel graph partitioning schemes. In *Proc. 24th Intern. Conf. Par. Proc., III*, CRC Press (1995), 113–122.
8. Karypis, G., and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 1 (Dec. 1998), 359–392.
9. Kernighan, B. W., and Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal 49*, 2 (1970), 291–307.
10. M. Gutierrez-Alcaraz, G. W. Experiences with the devstone benchmark.
11. Quesnel, G., Duboz, R., and Ramat, E. The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory 17* (April 2009), 641–653.
12. Vangheluwe, H. DEVS as a common denominator for hybrid systems modelling. In *IEEE International Symposium on Computer-Aided Control System Design*, A. Varga, Ed., IEEE Computer Society Press (Anchorage, Alaska, 2000), 129–134.
13. Vangheluwe, H., Lara, J., and Mosterman, P. J. An introduction to multi-paradigm modelling and simulation. In *AIS'2002. Simulation and Planning in High Autonomy Systems*, F. Barros and N. Giambiasi, Eds., Society for Modelling and Simulation International (Lisbon, Protugal, April 2002), 9–20.
14. Wainer, G. A., Liu, Q., and Jafer, S. *Advanced parallel simulation of DEVS models in CD++*. Taylor and Francis, 2010, ch. 9, TBD. Authors: G. Wainer, P. Mosterman Eds, Book: Discrete-Event Modeling and Simulation: Theory and Applications.
15. Wei, Y.-C., and Cheng, C.-K. Towards efficient hierarchical designs by ratio cut partitioning. In *Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on* (Nov 1989), 298–301.
16. Zeigler, B. P., Kim, D., and Praehofer, H. *Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed. Academic Press, 2000.