Building partitioning graphs in Parallel-DEVS context for parallel simulations (WIP)

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 (sub-models) 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 [12]. The multi-modelling approach allows to couple heterogeneous models (i.e. each models can use different formalisms). DEVS [15] *Discrete Event Specification* is a good candidate to develop the multi-modeling approach [11]. 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 [13].

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 [15] 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 encourage to read the section 3.4.2 in chapter 3 and the section 11.4 in chapter 11 of Zeigler's book [15].

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

TMS/DEVS 2015 April 12-15, 2015, Alexandria, VA

$\delta_{ext} : Q \times X^b \to S$ 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 $\delta_{int(s)}$. 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.

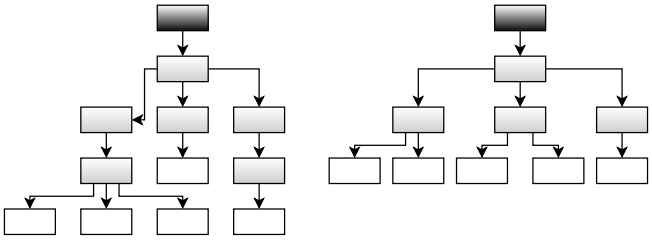

Figure 1. At left a typical DEVS model provides by modeler and at right a optimised graph for simulation.

Minimisation of the models hierarchy provides an optimised graph. TIn order to have as much coupled models in the penultimate level as there are available processors , the hierarchy is modified. Moreover, this coupled models are built to equalize the computation time between processors and for minimizing their interactions.

INTEGRATION OF THE GRAPH PARTITIONING IN DEVS SIMULATION

Our approach consists to transform the model structure in another in order to be optimized for parallelization. For this, we introduce partitioning algorithms. This work is possible thanks to the *closure under coupling* property of DEVS [15]. 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.

This coupled model can be represented by an oriented graph where each vertex is an atomic model and edges represente the communication network between them. We use undirected graph partitioning methods to create the partition. The first step converts the oriented graph in a unoriented graph, in order to apply our methods.

The following subsections present a summary of partitioning graph theory.

Generality on the graph partitioning

The k-way graph partitioning allows to cut a graph $G = (V, E)$ (V vertices set, E edges set) into k subgraphs $\{G_1, G_2, \ldots, G_k\}$, while respecting one or more "objective function". Thus, this cutting provides k subsets of vertices

called partition. To be good quality, a partition must respect some conditions : the parts weight must be similar and connections between parts must be minimal.

In our works, connections represent the information flow between models of different parts. In order to quantify this flow, each edge have a weight reflecting the data proportion transmitted between its models. The vertex weight enable to quantify the execution time of a model. Slower is a model, bigger is his weight. The following subsection present the objective functions used for partitioning.

The objective functions

The objective functions are criterions which give the partitioning quality. More it's little than the partition quality is good. It revolve around two concepts: cost cutting between partition parts and parts weight.

Given two subsets $V_1, V_2 \subseteq V$ and P_k a partition of V, the cut cost between two parts (1) and for a partition (2) is defined by :

$$\text{Cut}(V_1, V_2) = \sum_{v_1 \in V_1, v_2 \in V_2} \text{weight}(v1, v2) \quad (1)$$

$$\text{Cut}(P_k) = \sum_{i<j} \text{Cut}(V_i, V_j) \quad (2)$$

The *Cut* is the edges sum connecting the partition parts. This objective function was already used by Brian Kernighan and Shen Lin in [8].

Another function allows simultaneous management the minimization of the cut cost and weight balance between parts : the ratio cut. It's introduced by Yen-Chuen Wei and Chung-Kuan Cheng in [14].

$$\text{Ratio}(P_k) = \sum_{i=1}^{k} \frac{\text{Cut}(V_i, V - V_i)}{\text{weight}(V_i)} \quad (3)$$

In our works, we seek to minimize this objective function.

The main methods of the graph partitioning

There are lot of methods of graph partitioning. The main categories are: greedy methods, spectral methods, metaheuristics and region expanding methods. We search a simple and effective method to minimize the ratio cut. For create partitions we used the concept of neighbourhood. It's why, we choosed an expanding region method : the Greedy Graph Growing Partitioning (GGGP).

The GGGP method is an amelioration of the "Graph Growing Partitioning" method introduced in [7]. GGGP is a bisection method, which aims to divide the vertices set of the graph into two parts of equal weight. Given two vertices sets *Vertex_source* (V_s) and *Vertex_destination* (V_d), where V_s include all vertices and V_d is empty at the initial step. The algorithm starts by a randomly selection of a vertex in V_s and moves it in V_d. The neighbor vertices of V_d, not included in V_d, which give a maximum gain are moved from V_s to V_d. The process stops when the weight of V_d is equal to the half weight of graph vertices.

Our method is a variation of GGGP because it's extended for reduce the ratio cut and not simply the edge-cut. To realize a k-way partitioning it was necessary to create a recursive application.

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 reduce its size without changing its structure. We propose to implement a multilevel schemes, presented in [7].

Multilevel Graph Partitioning

The multilevel create quickly a graph partition of big size using three phases presented in the figure 2 :

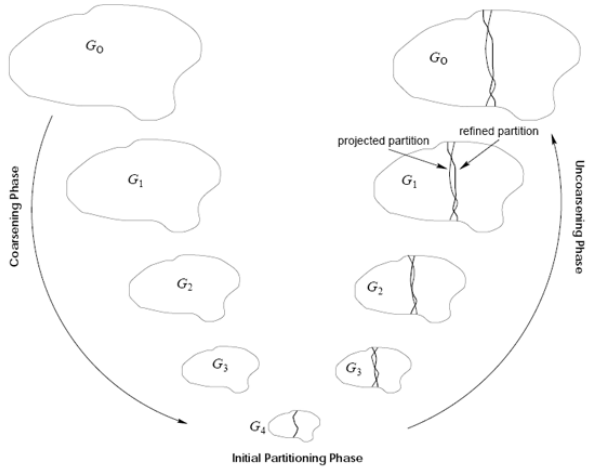

Figure 2. Multilevel Graph Bisection

- Coarsening: Graph reduction by successive vertices matching, while keeping the nature of the original graph.
- Partitioning: Creating of a partition P_k of the coarsening graph using a partitioning heuristic.
- Uncoarsening: Projection of the partition P_k on each contraction graph levels. But after each projection it's necessary to realise a refinement for keep a good quality.

Our multilevel implementation is inspired by algorithms of the literature. For some of them, we have developed techniques to reduce the random selections. The following subsection present these steps.

Coarsening Phase

This step create a graph base $\{G_1, \cdots, G_n\}$ by successive matching of vertices. In our multilevel, we have use the Heavy Edge Matching introduced in [7].

Partitioning Phase

The partitioning phase use the GGGP method described in last section. It's a fast and efficient method, but the result is very dependent of the chosen starting vertex. To relieve this, we propose a partitioning optimisation approach related to starting vertex. This is described in the algorithm 1.

Choose $Nselect$ differents starting vertices and keep the best partition that minimizes the objective function. If we consider the solution space like a partition quality, this approach consist to look into the solution space in order to find a good solution (not necessarily the best). For this, we propose to realise between 5 and 15 "smart" selections. In order to avoid the selection of two vertices that will give a similar result.

Algorithm 1: Partitioning Optimisation

Input: Graph $G(V, E)$, Integer $Nselect$
Output: Partition P

Initialisation :
- V vertices set of G sorted randomly
- $crit \leftarrow \infty$ minimization criterion
- P_i partition with GGGP for starting vertex i

for *i in 1 to* $Nselect$ **do**
- Partitioning for starting vertex V(i) $\rightarrow P_i$
- Compute of the criterion $\rightarrow tmp\_crit$
- **if** *tmp_crit* $<$ *crit* **then**
  - $P = P_i$ partition recording
  - $crit = tmp\_crit$
- **else**
  - Delete P_i

Assuming that vertices located in the same area may provide a similar partition, we create a selection method using the distance between vertices. Distance is the number of arcs defining a path (without cycle) between two vertices. Selection politics of the starting vertex implemented prohibits the selection of vertices located at a distance less than $Dmax$. To avoid too much loss of quality, we propose to fix d-max between 2 and 3. This allows to reduce the start selections number without reduce the partition quality. Algorithm 2 introduce this method.

Algorithm 2: Optimal selection

Input: Graph $G(V, E)$, Integer $Dmax$, Integer $Nselct$
Output: $Vset$

Initialisation :
- $Vset$ vertices set of G sorted
- $cpt \leftarrow 0$ counter

while *cpt* $< Nselect$ **do**
- Random selection of a vertex $v \in Vset$
- Search of vertices from a distance lower to $Dmax$
- Remove these vertices in $Vset$
- $cpt \leftarrow cpt + 1$

Uncoarsening Phase
As George Karypis and Vipin Kumar in [6], the uncoarsening phase of the multilevel projects the partition on the original graph step by step.

The method is a local optimisation algorithm based on Kernighan-Lin algorithm [8]. It's moving successively vertices located on the periphery of a part. A vertex is on the periphery of a part V_i if have a common edge with a vertex which is not in V_i. For each periphery vertex of a part, we save the gain associated at the movement thereof toward each neighbouring parts. The gain is the difference between the previous objective function value and the new one. If at least one gain is positive, the vertex is moved to the maximum gain part. This process is applied for each partition part and it's repeated as long as there is a gain. This method is described in algorithm 3.

Algorithm 3: Refining by local displacement

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

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

while *Ratio Cut decreases* **do**
- **for** *each partition parts* **do**
  - $D \leftarrow$ compute of cutting difference for 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 α
    - **else**
      - Next vertex

The eligible vertices for displacement are obtained using the cut difference introduced by Kernighan-Lin in [8]. To define it, we introduce the internal and external cut. Given a vertex $v \in V_i$, the internal cost $I(v)$ is the sum of the adjacent edges weights to v such as the second vertex is in V_i. The external cost $E(v)$ the sum of the adjacent edges weights to v such as the second vertex isn't in V_i.

$$I(v) = \sum_{v' \in V_i} \text{weight}(v, v') \tag{4}$$

$$E(v) = \sum_{v' \in V - V_i} \text{weight}(v, v') \tag{5}$$

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

$$D(v) = E(v) - I(v) \tag{6}$$

All vertices $v \in V_i$ such as $D(v) > 0$ are eligible for a moving. If $D(v) \leqslant 0$, it's possible that v is not on the periphery or its move not given a gain.

RESULTS

This section show the impact of the model structure in a DEVS simulation. For this, we make two comparisons of the simulation times. The first compare a flat graph and an optimized graph with parallelization. And the second compare a classic modeler's graph and the optimization graph for simulation. For the test phase, we choose two structures inspired by the water flow models.

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.
- 37 replicas are run to avoid benchmarks problems (i/o access, processor affinity etc.).
- We vary the partitioning method (original modeler's graph and GGGP) and parts number 2 to 16 every 2.

We use the PDEVS features to parallelize the simulation for all coordinators. PDEVS approach uses a riskfree and strict causality adherence. It computes a minimum time synchronization and exploits simultaneous events in parallel but it does not employ parallelism in feedback-free coupling. In this example, we use the forties threads available onto the hardware processor.

In our tests, we parameterized the multilevel as follows :

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

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 [10] 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.

Data presentation

We realised tests from two graph types derived from the flattening of hierarchical models. For each graph, the vertices weight is equal to 1 because the models execution time is the same. And the edges weight is equal to 1 because the message transfert cost is the same for each model. This graphs are presented in Figure 3.

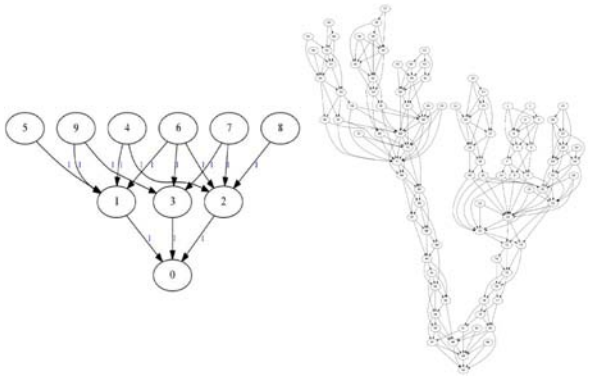

Figure 3. Examples of little graphs size. On the left, a "linked graph" and on the right a "tree graph" (abusively named).

The left graph consists of several levels, where each vertex of level n is connected with one or more vertices of level $n-1$. For the penultimate level, vertices are connected only to the outlet. The first level contains only sources vertices (e.i. sarter models of the simulation).

The right graph is composed of several branches, where each vertex is connected to one or more vertices following a single direction. The branches are branched until reaching the single outlet. This graph have several source vertices on each top branche (n sources by branches).

Results observation

In this subsection, we compare the times obtained for a simple simulation (i.e. simulation without hierarchy and not parallelized) and for a parallelized simulation (i.e. when the coordinator and this childs are parallelized) using partitioning methods. The aim is to compute the time gain generated thanks to the parallelization. We show also the impact of the structure obtained by the partitioning method on the simulation time. Two approaches are used for generate the structure : original modeler graph and partitioning method GGGP. The modeler graph is an intuitive partition gave by the modeler. We can see it as a random partitioning.

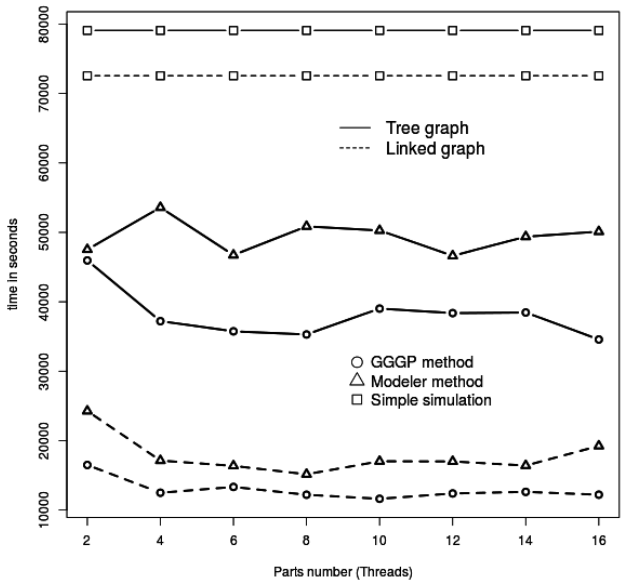

Figure 4. Simulation time for two graph types. Time comparison between a simple simulation and parallelized for different partition numbers. Performances evaluation between the partitioning methods : GGGP and modeler.

The figure 4 presents simulation times obtained for "tree" graph of 20000 vertices (i.e. simulation with 20000 atomic models) and "linked" graph of 10000 vertices. There are three line types : simple time (□), parallelized time with modeler (△) and with GGGP (◯).

For each graph, we can observe that the simple simulation time is greater than the parallelized. For a "tree" graph, the parallelization reduce the time approximately 50%. And for the "linked" graph, approximately 80%. These results can be

explained by the graphs structure. The "linked" contain more edges than "tree", it seems that the parallelization is suitable for this graph type.

We focus on times obtained with modeler and GGGP structure. The parallel structure obtained using GGGP reduce the execution time approximately 20%. For example, for six threads the simulation time of "tree" is reduced by 23% and by 21% for "linked". This gain comes from the objective function respect. Our partition is created to minimize the messages transfer and the charge balance (i.e. equal execution time in each thread).

We can notice that the execution time is reduced for a thread number less than 8. Beyond 8, we observed no more gains. This can be explain by the used algorithm (heap structure) for schedulers when the size of scheduler is small.

We also used the devstone bench [9] to test our method. Results are similar to those obtained with our graphs.

CONCLUSION

This paper presents a method to improve simulation time. It consists in partitioning a DEVS graph models in a optimised graph for the parallel simulation, i.e. by minimizing number of message exchange and by balancing of models execution time. The reduction time 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 subgraphs, it's essential to minimize the objective function that improves simulation at best. The tests shows that our approach offers better resultats that for simple simulation. But it's important to choose a good partitioning method for the creation of the structure. We can observe that the GGGP provide a better times than the modeler ($\approx 20\%$ of additional gain). This is due to the good quality of the structure.

In the future works, we'll develop an automatic graph weighting system based on the parameters provided by the DEVS models (vertex weight depending on the time advanced function t_a, for example).

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. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 1 (Dec. 1998), 359–392.
8. Kernighan, B. W., and Lin, S. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal 49*, 2 (1970), 291–307.
9. M. Gutierrez-Alcaraz, G. W. Experiences with the devstone benchmark.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.