Distributing Simulation Work Based on Component Activity: A New Approach to Partitioning Hierarchical DEVS Models

Sunwoo Park and Bernard P. Zeigler
The Department of Electrical and Computer Engineering,
University of Arizona, Tucson, Arizona, 85717
{Sunwoo, Zeigler}@ece.arizona.edu

Abstract

In this paper, we propose a new Generic Model Partitioning (GMP) algorithm for hierarchical, modular Discrete Event System Specification (DEVS) models. The GMP algorithm decomposes a given hierarchical model into a set of partition blocks and provides reasonable solutions for distinct partitioning problems based on a cost analysis methodology. The proposed algorithm minimizes model decomposition during the partitioning process and guarantees incremental quality of partitioning (QoP) improvements until a best partitioning is attained. Since a cost measure is a parametric method, subject to certain axioms, the proposed algorithm is generic and applicable any family of models provided there is a way to manipulate the appropriate cost information. An application to partial differential equation simulation using activity as the cost measure is discussed.

1. Introduction

For more than a decade, research has been conducted to develop Modeling and Simulation (M&S) frameworks over various distributed network infrastructures. As a result, numerous frameworks have been implemented to achieve a certain level of satisfaction with respect to *connectivity*, *speedup*, and *resource utilization*. By connecting numbers of computers that are loosely or tightly coupled in distributed network infrastructure and sharing resource information between M&S entities, desired M&S activities are conducted faster and more efficiently as compared the same activities on a standalone system. A certain level of interoperability and collaboration between M&S entities is also achieved in those frameworks.

In distributed simulation, there are many issues effecting simulation performance. Time synchronization and communication overhead are some of the major issues. Those issues are mainly related to simulation entities (e.g., logical simulator, coordinator, and activator) that are involved in the actual simulation process. By synchronizing time information with desired accuracy and minimizing communication overhead between simulation entities, the overall performance of simulation becomes improved.

Model partitioning is a major issues effecting simulation performance. Simulator performance can be significantly improved by optimally distributing simulation models into simulation entities before initiating the simulation process. Optimal model distribution is closely related to how models are partitioned and deployed to those entities. Thus, it is important to develop partitioning algorithms that can find optimal or, at least, acceptable partitioning results for given simulation models. Simulated annealing, random partitioning, and heuristic partitioning are some of them [1][2].

The main objective of this research is to design and implement a new generic model partitioning algorithm of hierarchical, modular Discrete Event Specification System (DEVS) models for distributed simulation[3]. To attain this goal, a new hierarchical model partitioning algorithm is designed based on a cost analysis methodology[4]. The cost analysis approach leads to algorithms that are concise, generic, and adaptable. The heterogeneous nature of models is captured and manipulated by the homogenous measure *cost*. Furthermore, the proposed algorithm minimizes model decomposition during the partitioning process and guarantees incremental Quality of Partitioning (QoP). The incremental QoP property is used to produce better partitioning results until a desired partitioning solution is attained. Partitioning improvement occurs during the partitioning process. Improvements are denoted in the partitioning tree and are easily tractable through the tree hierarchy.

The remainder of this paper is organized as follows. Section 2 and 3 briefly introduce DEVS model partitioning and cost analysis methodology, respectively. Section 4 describes a new hierarchical model partitioning algorithm proposed in this paper. Section 5 shows how the proposed algorithm is applied to a partitioning problem. And, finally, Section 6 summarizes and concludes this paper.

2. DEVS Model Partitioning

Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE'03)
0-7695-1984-9/03 $17.00 © 2003 IEEE

The Discrete Event System Specification (DEVS) formalism provides a means of specifying a mathematical object called a system[3]. Basically, a system has a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs. Discrete event systems represent certain constellations of such parameters just as continuous systems do. For example, the inputs in discrete event systems occur at arbitrarily spaced moments, while those in continuous systems are piecewise continuous functions of time. DEVS supports hierarchical, modular model construction, with coupled models composed of both coupled model and atomic model components. The DEVJAVA environment[4][5] is an implementation of the DEVS formalism in Java that enables the modeler to specify models directly in its terms.

DEVS Model partitioning refers to the process of creating a set of partition blocks from given model(s) based on certain decision-making criteria. Numerous model partitioning algorithms have been developed in the last few decades. Among them, algorithms based on graph partitioning are mainly focused in this paper[1][2].

Partitioning algorithms are mainly classified into *random partitioning algorithm*, *partitioning improvement algorithm,* and *heuristic algorithm*. The random partitioning algorithm is the simplest partitioning approach that creates a set of partition blocks with randomly selected models. The partitioning improvement algorithm refers to an algorithm in which the quality of partitioning is improved as partitioning proceeds[6][7]. The Kernighan-Lin algorithm performs partitioning by randomly assigning models to partition blocks at the initial time and swapping models between partition blocks during the partitioning process if swapping produces a better partitioning result[8]. The heuristic partitioning algorithm refers to any algorithm that uses domain-specific knowledge for performing the partitioning process.

Partitioning algorithms can be realized by utilizing certain optimization techniques. The most widely used technique is simulated annealing. Simulated annealing is a general-purpose optimization technique based on a statistical methodology[9]. Partitioning algorithms can also be constructed based on geometric information of models. Recursive Coordinate Bisection (RCB) and Recursive Graph Bisection (RGB) are some of them[10][11].

Hierarchical model partitioning is the process of building a set of partition blocks by decomposing or constructing hierarchical structures based on certain decision-making criteria. The hierarchical structure is generally represented by a tree structure. In the structure, a node having no children is an atomic (or non-decomposable) node. A node having children is a coupled (or decomposable) node. The coupled node could contain other coupled nodes and a collection of atomic nodes. During partitioning, the hierarchical model structure may be dynamically permutated over time and space, if necessary. A partitioning policy specifies what kind of partitioning approach or technique is applied to models process. There are three partitioning policies which are widely accepted and applied in many application domain problems; *flattening, deepening,* and *heuristic*. *Flattening* is a structural decomposition technique that transforms a hierarchical structure into non-hierarchical structure. *Deepening*, also known as *hierarchical clustering*, is a structural aggregation technique that transforms non-hierarchical structure into hierarchical structure [12]. A *heuristic* policy is an ad-hoc policy that uses techniques other than those that are associated with flattening and deepening.

3. Cost Analysis Methodology

Cost analysis is an analytical approach of abstracting heterogeneous *resource* information into homogeneous *cost* information and performing certain analytical operations with respect to the cost information. The methodology covers a wide spectrum of analytical activities ranging from harvesting cost information to performing sophisticated analytical operations regarding the information. Specifically, it copes with how to acquire cost information from particular resource information *(cost harvesting)*; how to create cost information based on a certain generation scheme *(cost generation);* how to coalesce a group of small costs into a bigger cost *(cost aggregation)*; how to compare a cost with another cost *(cost evaluation)*; how to analyze cost information *(cost analysis)*, etc.

A cost is an abstract value that represents the cost information of a model in the format of a particular data type such as a single value, a set of discrete objects, a range of continuous values, a probability distribution, a stochastic process, etc. A *cost measure* is a conceptual metric that captures heterogeneous resource information in terms of cost. Various metrics can be used to harvest, generate, and evaluate cost information based on distinct decision-making criteria. *Complexity*, *I/O connectivity*, and *behavior functions* are some of the cost measures that are widely used. A *cost function* is a mathematical representation of a cost measure. It is an abstract function. Thus, for a single cost function, numerous implementations are possible. *Cost aggregation* is the process of coalescing a group of small costs into a bigger cost. By representing a group of costs by its aggregated cost, sophisticated relationships between costs can be significantly simplified. This makes manipulating cost information much easier with respect to cost analysis.

A *cost tree* is a tree structure that is created from a model that contains cost information. A node in the tree is either *atomic* or *coupled*. An atomic node is a terminal node containing no child nodes. A coupled node is an intermediate node holding more than one child node. Each node contains a *model* and *its cost* regardless of its classification type (i.e., atomic or coupled). The cost of the node is implicitly retrieved from its associated model or explicitly assigned to the node. A *node* is evaluated by retrieving the cost information of the node. An atomic node contains its own cost only and a coupled node keeps all costs of the node and its descendents that are reachable through the tree hierarchy. Thus, the cost of a subtree starting from a particular node is acquired by simply retrieving cost information of the node without further expansion of the tree. It reduces considerably the amount of time and space required for parsing all descendants of the node and aggregating their costs during the cost evaluation process.

Table 1. Cost Measures and Cost Functions

Cost measure	Cost function	Decision-making criteria
I/O connectivity	Size(X_{model})* Size(Y_{model})	The cost of a system is generally proportional to the number of I/O interfaces if the system is dedicated to serving I/O requests.
System Complexity (without I/O connectivity)	Size(Γ_{model})	The cost of a system is represented by the number of internal states rather than the number of I/O access points if system performance relies on its complexity.
System Complexity (with I/O connectivity)	Size(Γ_{model})* Size(X_{model})* Size(Y_{model})	The cost of a system can be captured more appropriately by considering both I/O interfaces and system complexity
System Activity	Numberof ($Transition_{model}$)	The cost of a system can be capture more appropriately by considering dynamic system behaviors.

X_{model}: a set of input interfaces, Y_{model}: a set of input interfaces
Γ_{model}: a set of internal states,
$Transition_{model}$: internal transitions for a certain period of time[13]

4. A Generic Model Partitioning (GMP)

A new Generic Model Partitioning (GMP) algorithm for a hierarchical, modular Discrete Event System Specification (DEVS) model is proposed in this paper. The proposed algorithm decomposes a given hierarchical model into a set of partition blocks and provides solutions for distinct partitioning problems based on the cost analysis methodology. Unlike previous research on DEVS model partitioning[14][15], the GMP algorithm provides adaptability (or flexibility) for distinct partitioning requirements based on the cost analysis methodology.

The algorithm is characterized by a set of generic cost measures for cost generation, cost evaluation, and cost aggregation. Since a cost measure is a parametric method, subject to certain axioms, the algorithm is generic and applicable to any family of models provided there is a way to manipulate the appropriate cost information. *Activity* is one of possible cost measure. However, the more general concept potentially includes other important determiners of simulation work such as number of messages sent and received. By applying one or more cost measures, a model is abstracted to a cost regardless of its complexity or heterogeneity. The homogeneity of the cost allows the proposed algorithm to be applicable to heterogeneous problems by simply changing cost measures without any modification of the algorithm itself. This is due to the homogeneous nature of the cost analysis methodology. Thus, the proposed algorithm is highly adaptable and can be applied to various application domains.

The GMP algorithm minimizes model decomposition by breaking the given model only until a best partitioning result is attained, instead of fully decomposing the model before or during the partitioning process. With minimization of model decomposition, the GMP algorithm becomes less sensitive to the depth or the complexity of a hierarchical model. It makes the algorithm much more flexible and scalable compared to other partitioning algorithms based on full decomposition (or flattening). One unique feature of the GMP algorithm is its support for incremental Quality of Partitioning (QoP) improvement that guarantees partitioning results evolve into the best result without any degradation of QoP during the partitioning process. This allows the algorithm to produce a high degree of QoP for the given model. The QoP is easily traceable through a partitioning tree hierarchy.

The GMP algorithm has two phases; *initial partitioning* and *evaluation-expansion-selection (E^2S) partitioning.* In the initial partitioning phase, a root node of the partitioning tree is constructed. While, in the phase of E^2S partitioning, child nodes of the tree are constructed, evaluated, and expanded until a best partitioning result is attained.

4.1. Initial Partitioning

Initial partitioning creates a root node of a partitioning tree. The partitioning tree is a tree that is created during the partitioning process. The root node of the tree is built based on a cost tree and the number of partition blocks. The initial partitioning is started by creating a set of empty partition blocks. Once the partition blocks are created, each block will be populated with one or more

cost nodes. Those nodes are obtained from the cost tree by decomposing a specific node of the tree. The total number of cost nodes increases or decreases during the partitioning process. Specifically, before assigning a cost node to an empty partition block, the total number of available cost nodes is compared to the requested number of partition blocks. If the number of nodes is smaller than the number of partition blocks, node expansion occurs. The node having the highest cost among the available coupled nodes is selected and expanded. Nodes expanded from the selected node become available along with existing cost nodes. Expansion is repeated until the total number of cost nodes is equal to or larger than the number of partition blocks. After expansion, every partition block is filled with a cost node. Once all partition blocks become non-empty, the remaining cost nodes are distributed into those blocks based on some decision-making criteria. Evaluation of the initial partitioning result is done by applying a partitioning evaluation function to those blocks. To describe initial partitioning, we define the followings;

Constant:
T: a cost tree
p: a number of partition blocks

Objects:
PB: a partition block
PB_{empty}: an empty partition block
PB_{lowest}: a partition block having the lowest cost
$PB_{highest}$: a partition block having the highest cost
$Node_{lowest}$: a node having the lowest cost
$Node_{highest}$: a node having the highest cost
$Node_{coupled}$: a coupled node
$Node_{coupled}^{highest}$: a coupled node having the highest cost

Operations:
remove(*node*, *c-list*): remove a node, *node*, from the c-list
node ← remove(*node*, c-list)
expand(*node*): expand a node, *node*
a set of child nodes of the *node* ← expand(*node*)
assignTo(*node, PB*): assign a node, *node*, into a partition block, PB

With the above definitions, the initial partitioning algorithm is represented as follows

```
1:  PB[] Initial-partitioning(T, p)
2:   // phase 1: initialize c-list and p-array
3:   c-list = child nodes of a root node in T
4:   p-array = PB[p] // create p empty partition blocks
5:   // phase 2: expand node(s), if necessary
6:   while lengthOf(c-list) < numberOf(p-array) do
7:     if c-list contains at least one Node_coupled then
8:       c-list += expand(remove( Node_coupled^highest ,c-list))
9:     else
10:      return error("can't expand…")
11:    endif
12:  endwhile
13:  // phase 3 : fill empty partition blocks
14:  while p-array contains an PB_empty do
15:    assignTo(remove(Node_highest,c-list), PB_empty)
16:  endwhile
17:  // phase 4 : distribute nodes in c-list into partition blocks
18:  while c-list is not empty do
19:    assignTo(remove(Node_lowest,c-list), PB_lowest)
20:  endwhile
21:  return p-array
22: endprocedure
```

Algorithm 1. Initial Partitioning Algorithm

4.2. Evaluation-Expansion-Selection (E^2S) Partitioning

Evaluation-Expansion-Selection (E^2S) partitioning constructs, evaluates, and expands child nodes of a partitioning tree and increases the QoP until a best partitioning result is attained. The E^2S partitioning is started by identifying the partition block having the highest cost given initial partitioning result that is stored at the root node of the partitioning tree. Once the block is identified as an expandable partition block, *e-partition*, the existence of a coupled model is checked at the block. If it exists, the coupled node having the highest cost is identified as an expandable node, *e-node*. Otherwise, a new partitioning block having the highest cost, excluding previously selected blocks, is selected as the expandable partition block. If a coupled node does not exist at the newly identified partition block, the same procedure is repeated until both an expandable block and an expandable node are found. By expanding the expandable node, a collection of cost nodes is created. If the expandable block becomes empty after expanding the node, the cost node having the highest cost in the collection is assigned to the block. Remaining nodes in the collection are distributed to partition blocks in ascending order, as described in distribution phase in the previous chapter. After distributing remaining nodes into partition blocks, the partitioning result is compared to the previous partitioning result. The cost measure, the partitioning evaluation function, is used to perform the cost comparison. If the new partition result is superior to the previous one, the E^2S partitioning is applied recursively to find a better partitioning result. Otherwise, the previous partitioning result is identified as a best partitioning result for the given cost tree and the requested number of partition blocks.

```
1:  PB[] e-square-s-partitioning(PB[] p-array)
2:  // phase 1: initialize e-array and e-partition
3:   e-array = p-array
4:  e-partition = a PB_highest in e-array
5:  e-node = null
6:  // phase 2: identify an expandable PB from e-array
7:  while true do
8:    if e-partition ≡ null then return p-array
9:    else
10:     if e-partition contains Node_coupled then
11:       e-node = Node_coupled^highest in e-partition
12:       if e-node ≠ null then break
13:       else return p-array
```

```
14:       endif
15:     else
16:       e-partition = select the PB_highest from e-array
17:                 excluding previously selected PBs
18:     endif
19:   endif
20:  endwhile
21:  // phase 3: expand e-node and put them into c-list
22:  c-list = expand(remove(e-node,e-partition))
23:  // phase 4: fill the e-partition with Node_highest
24:  //            if e-partition is empty
25:  if empty(e-partition) then
26:       assignTo(remove(Node_highest,c-list), e-partition)
27:     endif
28:  // phase 5: distribute nodes to e-array
29:  while c-list is not empty do
30:    assignTo(remove(Node_lowest,c-list), PB_lowest)
31:   endwhile
32:   // phase 6: evaluate a new partitioning result
33:   if superiorTo(evaluate(e-array), evaluate(p-array)) then
34:     return e-square-p-partitioning(e-array)
35:   else
36:     retrun p-array
37:   endif
38:  endprocedure
```

Algorithm 2. Evaluation-Expansion-Selection (E^2S) Partitioning Algorithm

5. Partitioning 1-Dimensional Activity Distribution Problem

The DEVS formalism has been applied to a number of continuous as well as discrete phenomena[16][17]. The use of discrete events, rather than time steps, as a basis for simulation has been shown to reduce computation time by orders of magnitude in many applications [18][19][20] showed how discrete event abstraction captures information in rich continuous data streams in terms of events and their timed occurrences. Recently, there has been significant progress in extending DEVS to represent partial differential equation (PDE) models. In a gas expansion shockwave problem, [21] showed that the time to execute the solution is significantly reduced when a discrete event integration scheme is employed compared to a representative conventional approach. Recent theory suggests that speed advantages are to be expected for pdes that are characterized by heterogeneity in their time and space behavior. In such cases, as exemplified by an outwardly moving shock wave, discrete events are a natural way to focus attention on the portions of the solution that are exhibiting high activity levels at the moment. In fact, theory suggests a way to characterize the activity of solutions over time and space independently of the solution technique that might be employed. This activity measure, when divided by a quantum size, predicts the number of boundary crossings (computations) required by the DEVS simulator for the accuracy afforded by that quantum size. Where significant heterogeneity of activity exists, the number of discrete event computations may be orders of magnitude lower than that required by a uniform allocation of computational resources across both space and time. Since the DEVS hierarchical, modular framework accommodates coupled models containing both discrete and continuous components, it offers a scalable, efficient framework for very large scale distributed simulation.

5.1. Approach

A simple 1-dimensional activity distribution is presented in this section to show how the GMP algorithm works. In this example, 1-dimensional distribution is considered as cost distribution between models. A node of a cost tree is defined by a pair of activity and spatial information of a model. The activity and the spatial information are represented by a positive value and 1-dimensional distance (or offset) between a model and a reference point, respectively. The reference point can be set to the left-most or the right-most point of the given cost distribution. An activity distribution and its corresponding cost tree are shown in **Figure 1**. Since the cost tree contains no hierarchical structure, the second part of the GMP algorithm, (i.e., E^2S partitioning) is not required to solve this problem.

To implement the GMP algorithm for this problem, we need to specify abstract functions and objects introduced in the algorithm precisely. Thus, we define the following before applying the GMP algorithm to the given partitioning problem.

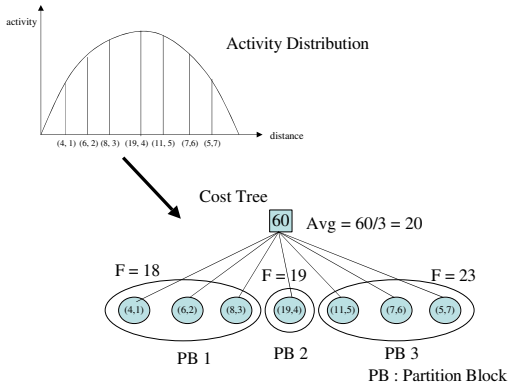

Figure 1. Activity Distribution and Associated Cost Tree

Constants;
p: the total number of partition blocks
m_i: the total number of cost nodes in PB_i

Objects;
$Node_i$: A pair of an activity value and an offset value
$Node_i^{activity}$: Activity value of a node, $Node_i$
$Node_i^{offset}$: Offset value of a node, $Node_i$

$Node_{root}$: Root node of a cost tree

$Node_{root}^{avg}$: Average of total activity of $Node_{root}$ (i.e., $\frac{Node_{root}^{activity}}{p}$)

$Node_{lowest}$: the node having the lowest cost

$$Node_{lowest} \quad \min_{i \ 1..k} Node_i^{offset} \ ,$$

where, $Node_i$ c $list$ k $lengthOf(c \ list)$

$PB_i^{aggregated}$: Cost aggregation of a partition block, PB_i

$$PB_i^{aggregated} \quad \sum_{j \ 1}^{m_i} Node_j^{Activity}, \quad Node_j \quad PB_i$$

PB_{lowest} : the PB having the lowest cost aggregation value

$$PB_{lowest} \quad \min_{j \ 1..p} PB_j^{aggregated}$$

Operations;

$addTo(Node_i, PB_i)$: add $Node_i$ into PB_i and return updated PB_i

$$PB_i' \quad addTo(Node_i, PB_i)$$

$removeFrom(Node_i, PB_i)$: remove $Node_i$ from PB_i and return updated PB_i

$$PB_i' \quad removeFrom(Node_i, PB_i)$$

In additions to above definitions, when a node is assigned to a partition block, the following node assignment rules should be obeyed.

Rule I: $if(PB_{lowest}^{activity} \quad)$
$addTo(Node_{lowest}, removeFrom(Node(\ ,0), PB_{lowest}))$

Rule II: $if \ PB_{lowest}^{aggregated} \quad Node_{lowest} \quad Node_{root}^{avg}$
$addTo(Node_{lowest}, PB_{lowest})$

Rule III: Otherwise,
$if(Node(\ ,0) \quad PB_{any})$
$addTo(Node_{lowest}, removeFrom(Node(\ ,0), PB_{any}))$
$else$
$else \ addTo(Node_{lowest}, PB_{lowest})$

In these assignment rules, it is assumed that every PB is initially created with a node, $Node(\ ,0)$. The node acts as an empty PB indicator. Rule I is applied when a node is assigned to a PB at very first time. Initially, the aggregated activity value of every PB is infinity. Thus, any PB can be identified as PB_{lowest}. Once PB_{lowest} is identified, $Node_{lowest}$, which is removed from the component list, *c-list*, is added into PB_{lowest} after removing the empty PB indicator, $Node(\ ,0)$ from the partition block. Rule II is applied when the sum of $PB_{lowest}^{aggregated}$ and $Node_{lowest}$ is equal to or less than $Node_{root}^{avg}$. Rule III is applied when the sum of $PB_{lowest}^{aggregated}$ and $Node_{lowest}$ is larger than $Node_{root}^{avg}$ or $Node_{lowest}$ is larger than $Node_{root}^{avg}$.

Initial partitioning is started by creating p partition blocks followed by computing $Node_{root}^{avg}$. Each partition is populated with an empty PB indicator, $Node(\ ,0)$. The phase 2 of initial partitioning, *expansion*, is skipped because the given problem has no hierarchical structure and the p is (considerably) less than the total number of models in the cost tree. The phase 3 of the algorithm, *filling*, is also skipped because every PB has $Node(\ ,0)$. In phase 4 of the algorithm, *distribution*, all children nodes of the given cost tree are assigned to the component list, *c-list*. Initial partitioning is achieved by removing $Node_{lowest}$ from the list and assigning it to $PB_{lowest}^{aggregated}$ until the list becomes empty. Node assignment should be performed based on node assignment rules described above.

The given 1-dimensional activity distribution example can be considered as a hierarchical model partitioning problem if we assume an activity value can be decomposed into a set of smaller activity values. In the above example, we implicitly presume every activity value in the activity distribution is non-breakable value. Instead, we assume an activity value in the distribution can be composition of smaller activity values, here. The decomposable activity distribution and its associated cost tree are shown in **Figure 2**.

In the tree, spatial information of each child of a coupled node is represented by appending a relative offset from its parent node into its parent's offset value (e.g., 4.2 and 4.2.2). In addition to definitions in initial partitioning, we define the following to realize E^{2}S partitioning

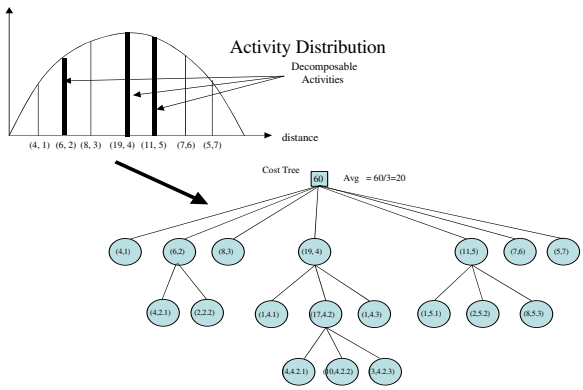

Figure 2. Decomposable Activity Distribution and Associated Cost Tree

***Constant*;**

$Array_{PB}$: a collection of partition blocks

***Objects*;**

$PB_{\max}$: PB$_i$ satisfying $\max_{i \ 1..p} PB_i^{aggregated}$

$PB_{\min}$: PB$_i$ satisfying $\min_{i \ 1..p} PB_i^{aggregated}$

$PB_{selected}$: $PB_{\max}$ except for previously selected $PB_{\max}$

PB_{prev} : PB satisfying $\max_{i \ selected} PB_i^{\max(offset)} \quad PB_{selected}^{\min(offset)}$

PB_{next} : PB satisfying $\min_{i \ selected} PB_i^{\min(offset)} \quad PB_{selected}^{\max(offset)}$

***Operations*;**

$evaulte(Array_{PB})$: Cost disparity between PB_{max} and PB_{min} of $Array_{PB}$ that is, $PB_{\max}^{aggregated}$ $PB_{\min}^{aggregated}$

$superiorTo(Array_{PB}, Array'_{PB})$:

$true$, $evaluate(Array_{PB})$ $evaluate(Array'_{PB})$

$false$, $otherwise$

Also, additional node assignment rules are listed as follows

Rule IV: *if* $PB_{prev}^{aggregated}$ $Node_{lowest}$ $Node_{root}^{avg}$

$addTo(Node_{lowest}, PB_{prev})$

Rule V: *if* $PB_{next}^{aggregated}$ $Node_{highest}$ $Node_{root}^{avg}$

$addTo(Node_{highest}, PB_{next})$

Rules in E^{2}S partitioning are used to redistribute a part of costs of the $PB_{\max}$ into its neighbor(s) (i.e., PB_{prev} and PB_{next}). Based on the previous partitioning result, E^{2}S partitioning is performing by expanding a particular node of $PB_{\max}$ and creating a new partitioning result. Once the result is created, it is compared with the previous partitioning result by executing operation *superiorTo()*. If the new result is better than the previous one, partitioning recursively continues until no better partitioning result is attainted. The previous partition result is initially assigned to initial partitioning result.

Figure 3 shows how E2S partitioning is performed for the given 1-dimensional decomposable activity distribution example. The root of a partitioning tree, $node_1$, is equivalent to the initial partitioning result. The partition block, which has costs, 11, 7, and 5, is selected as *e-partition* and the cost node having the cost 11 in the partition block is identified as *e-node* and is expanded to nodes having costs 1, 2, and 8. After distributing these expanded nodes into partition blocks, a new partitioning result, $node_2$, is obtained. Since the new result is superior to the previous result in terms of cost disparity, the E^{2}S algorithm continues to find a better partitioning result. During the partitioning process, $node_3$ and $node_4$ of the partitioning tree are constructed. Since the partitioning evaluation value of $node_4$ is not superior to the evaluation value of $node_3$, the algorithm terminates with returning the contents of $node_3$ as the final partitioning result. Other nodes in the figure, $node_5$, $node_6$, and $node_7$ are shown to show possible alternative choices during the partitioning process. Those nodes implies inferior partitioning results as compared to nodes associated with the final partitioning result (i.e., $node_2$, and $node_3$). **Figure 4** shows the partitioning result of the given 1-dimensional decomposable activity distribution example.

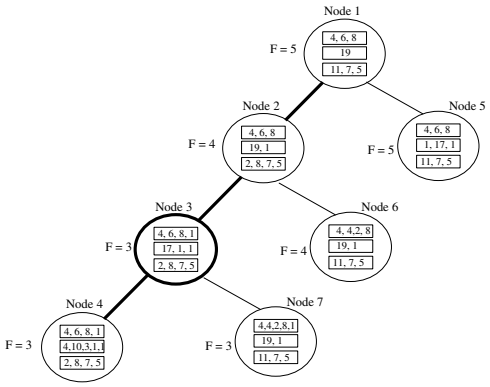

Figure 3. The Partitioning Tree of the Example

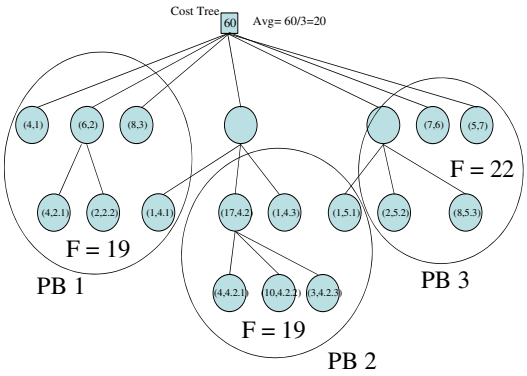

Figure 4. Partitioning Result of the Example

6. Conclusions

Unlike previous research on DEVS model partitioning, the proposed algorithm provides adaptability (or flexibility) for distinct partitioning requirements based on cost analysis methodology. It also minimizes model decomposition during the partitioning process and guarantees incremental quality of partitioning (QoP) improvement until a best partitioning result is attained. In the algorithm, a series of cost measures for cost generation, cost evaluation, and cost aggregation is introduced. Since a cost measure is a parametric method, subject to certain axioms, the proposed algorithm is generic and applicable any family of models provided there is a way to manipulate the appropriate cost information. Activity is an important cost measure to which the GMP algorithm is applicable. In recent experiments, we have demonstrated that activity closely predicts the number of transitions and the execution time required by a DEVS simulator for a PDE. There is an overhead that is required in DEVS simulation but this is of constant order. The overhead becomes negligible,

relative to traditional time stepped computation, for large numbers of cells or when the per-cell transition cost is relatively large.

Owing to its generic nature of the cost analysis methodology, the proposed algorithm is highly adaptable and can be applied to various application domain models and simulations. With minimization of model decomposition, the suggested algorithm becomes less sensitive to the depth or the width of a given hierarchical model. It makes the algorithm much flexible and scalable compared to other partitioning algorithm based on full decomposition (or flattening). One unique feature is to guarantee incremental QoP improvement during partitioning process. This allows the algorithm to produce high QoP for the given hierarchical model. The QoP is easily traceable through a partitioning tree hierarchy.

We plan to do follow up work in which the proposed algorithm is extended to deal with dynamically changing cost. This will allow us to consider PDE and other models that exhibit heterogeneity in their behaviors over time and space. The goal is to track and predict dynamic activity patterns enabling their distributed simulations to be restructured during run time to focus resources, using discrete event scheduling, on regions where activity is highest and to distribute active regions among computing units to equalize simulation work.

7. References

[1] A. Pothen, "Graph Partitioning Algorithms with Applications to Scientific Computing," *Parallel Numerical Algorithms*, pp. 323-368, 1997.

[2] P. Fjallstrom, "Algorithms for Graph Partitioning:A Survey," *Computer and Information Science*, vol. 3, 1998.

[3] B.P. Zeigler, H.P. Praehofer, and T.G. Kim, "Theory of Modeling and Simulation," Academic Press, 2000.

[4] S. Park, "Cost-based Hierarchical Model Partitioning for Distributed Simulation of Hierarchical, Modular DEVS Models," *Ph D. dissertation*, university of Arizona, may. 2003.DEVSJAVA software, http://www.acims.arizona.edu

[5] B.P. Zeigler, H.S. Sarjoughian. "Introduction to DEVS Modeling and Simulation with JAVA: A Simplified Approach to HLA-Compliant Distributed Simulations", http://www.acims.arizona.edu, 2001.

[6] A. Frieze and M. Jerrum, "Improved approximation algorithms for MAX k-CUT and MAX BISECTION," *Alogorithmica*, vol. 18, pp. 61-77, 1994.

[7] M. R. Banan and K. D. Hjelmstad, "Self-organization of architecture by simulated hierarchical adaptive random partitioning," presented at International Joint Conference of Neural Networks (IJCNN), 1992.

[8] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graph," *The Bell System Technical Journal*, vol. 49, pp. 291-307, 1970.

[9] V. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.

[10] M. J. Berger and S. H. Bokhari, "A Partitioning Strategy for Non-Uniform Problems across Multiprocessors," *IEEE Transactions on Computers*, vol. 36, pp. 570-580, 1987

[11] H. D. Simon, "Partitioning of Unstructured Problems for Parallel Processing," *Computing Systems in Engineering*, vol. 2, pp. 135-148, 1991.

[12] Y. Z. a. G. Karypis, "Evaluation of Hierarchical Clustering Algortithms for Document Datasets," *CIKM 2002*, 2002.

[13] B.P. Zeigler, "The brain-machine disanalogy revisited", *BioSystems,* Vol. 64, pp. 127-140, 2002

[14] G. Zhang and B.P. Zeigler, "Mapping Hierarchical Discrete Event Models to Multiprocessor Systems: Algorithm, Analysis, and Simulation," *J. Parallel and Distributed Computers*, Vol. 9, pp. 271-281, 1990.

[15] K.H. Kim, T.G. Kim and K.H. Kim, "Hierarchical Partitioning Algorithm for Optimistic Distributed Simulation of DEVS Models," Journal of Systems Architecture, Vol. 44, pp. 433-455, 1998.

[16] J. Ameghino, A. Tróccoli, G. Wainer. "Models of Complex Physical Systems using Cell-DEVS", Proceedings of the Annual Simulation Symposium, Seattle, Washington, 2001.

[17] Alexandre Muzy, Eric Innocenti, Antoine Aiello, Jean-Francois Santucci, and Gabriel Wainer. "Cell-DEVS Quantization Techniques in a Fire Spreading Application", Winter Simulation Conference, San Diego, California, 2002.

[18] B. P. Zeigler, G. Ball, et al., Bandwidth Utilization/Fidelity Tradeoffs in Predictive Filtering. Simulation Interoperability Workshop, Orlando, 1999.

[19] B. P. Zeigler, G. Ball, et al., Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions. Simulation Interoperability Workshop, Orlando, FL, 1999.

[20] Zeigler, B. P., H. Cho, et al., "Quantization Based Filtering in Distributed Discrete Event Simulation." Journal of Parallel and Distributed Computing **62**(11): 1629-1647, 2002.

[21] J. Nutaro, B. P. Zeigler, R.Jammalamadaka, S.Akerkar,"Discrete Event Solution of Gas Dynamics within the DEVS Framework:Exploiting Spatiotemporal Heterogeneity", Intl. Conf. Computational Sci, Melbourne Australia, July 2003

8. Acknowledgement

This research has been supported in part by NSF Grant No. DMI-0122227, "Discrete Event System Specification (DEVS) as a Formal Modeling and Simulation Framework for Scaleable Enterprise Design" and in part by the Scientific Discovery through Advanced Computing (SciDAC) program of the DOE, grant number DE-FC02-01ER41184.