Generic Model Partitioning with LookAhead *k*: A Multi-Scale Partitioning Algorithm for DEVS Biomimetic In Silico Devices

Sunwoo Park, C. Anthony Hunt, and Bernard P. Zeigler[1]
The BioSystems Group, University of California, 513 Parnassus Ave., San Francisco, CA, 94143-0446
[1]**Arizona Center for Integrative Modeling and Simulation, University of Arizona, Tucson, AZ, 85721**
spark4@itsa.ucsf.edu hunt@itsa.ucsf.edu zeigler@ece.arizona.edu

Keywords: DEVS, GMP, partitioning, biological systems

Abstract

In this paper Generic Model Partitioning with look ahead k (GMP-*k*) is discussed. We propose a concise, generic, and configurable new model partitioning approach for modular, multi-scale, Discrete EVent System Specification (DEVS) based, Biomimetic In Silico Devices (BDs). GMP-*k* decomposes a multi-scale BD into a set of partition blocks based on a cost analysis methodology that considers the lookahead parameter *k*. By controlling *k*, GMP-*k* produces a collection of partitioning results that provides different resolutions for the same BD. GMP-*k* minimizes model decomposition and constructs monotonically improved partitioning results during the simulation process. As a consequence of its clean separation between domain specific cost analysis and generic partitioning logic, the proposed algorithm can be applied to a variety of partitioning problems.

INTRODUCTION

As a consequence of the impressive evolution of computational devices and computer networks over the past two decades, various distributed/parallel computing infrastructures have emerged, including Networks of Workstations, Beowulf Clusters, and Peer-to-Peer networking [1][2][3]. Unlike classical super computing fabrics, which tightly integrate processor nodes with shared memory systems, these new infrastructures are mainly based on a set of remote processors and distributed memory systems that are loosely connected through networks.

To perform large-scale simulation tasks seamlessly and efficiently on a heterogeneous computing infrastructure, one must support generic and configurable non-simulation system components as well as simulation system components. Simulation execution time is affected by both. Time management and communication and memory overhead minimization are directly related to the simulation components. Simulation performance can be improved by applying an appropriate time management scheme (e.g., conservative, optimistic, or risk-free) and overhead reduction techniques (e.g., quantization or fossil collection) [4][5][6], whereas deployment, remote activation, and generic naming and directory service are some issues related to the non-simulation components [7].

Model partitioning is a major issue in distributed and parallel simulation. It involves both simulation and non-simulation components, and how it is done can impact overall simulation performance. Performance can be improved by optimally distributing simulation models to a set of simulators either before or during simulation. Because optimal model distribution is closely coherent to partitioning results, it is important to develop algorithms that can produce optimal — or, at least, acceptable — partitioning results with respect to the end points of interest. As a consequence, these algorithms have been the focus of considerable research. The directions taken include simulated annealing, random partitioning, and heuristic partitioning [8][9].

For this project, the main objective has been to design and implement new partitioning algorithms for DEVS BDs. To achieve this goal, we propose a new partitioning approach, GMP-*k*.

BACKGROUND

Model Partitioning

There are three main classes of model partitioning algorithms: *random partitioning*, *partitioning improvement*, and *heuristic partitioning*. Random partitioning algorithms function by randomly aggregating or segregating models to a set of partition blocks. Partitioning improvement algorithms revise partitioning results during the partitioning process [10], [11]. The Kernighan-Lin algorithm initially builds partitioning results by first randomly assigning models to partition blocks and then swapping models between partition blocks during the partitioning process whenever swapping produces a better partitioning result [12]. Heuristic partitioning algorithms use domain-specific knowledge or a particular optimization technique during the partitioning process. For example, the Recursive Coordinate Bisection and Recursive Graph Bisection algorithms use geometric information, whereas simulated annealing exploits statistical methods [13][14][15].

Hierarchical Model Partitioning

Hierarchical model partitioning is a process of constructing a set of partition blocks by decomposing or by building hierarchical model structures based on certain decision-making criteria [16][17][18]. Hierarchical model structure is generally abstracted by a tree structure. During the partitioning process, the tree structure is dynamically permutated over time and space. A partitioning policy specifies tree permutation mechanisms. There are three widely accepted and used policies: *flattening, deepening,* and *heuristic*. Flattening is a structural decomposition technique that transforms a hierarchical structure into a non-hierarchical structure. Deepening, also known as *hierarchical clustering*, is a structural aggregation technique that transforms non-hierarchical structures into hierarchical structures. A heuristic policy is one that specifies any technique other than flattening and deepening.

DEVS and Biomimetic In Silico Devices

DEVS is a modeling and simulation paradigm that describes dynamic systems based on a system-oriented mathematical formalism [5]. In DEVS, a system is elucidated as atomic or coupled models. An atomic model describes the temporal dynamics of an indecomposable system with behavior functions, states, and time. Behavior functions deal with event manipulation, state transition, and time management of the system. Dynamics of an atomic model are mainly described by states. States are a set of feasible discrete state spaces that the system can explore. Time is non-negative and real value. In a formal specification of a DEVS atomic model, *delta external transition function,* δ_{ext}, describes the system's response for incoming external events. *Internal transition function,* δ_{int}, describes the system's autonomous behaviors. The *delta confluent function,* δ_{con}, specifies the behavior causality between these two functions (i.e., δ_{ext} and δ_{int}) when both external and internal events occur simultaneously. The *output function,* λ, generates output events. The state transitions are triggered by δ_{ext} and δ_{int} explicitly, or δ_{con} implicitly. In an atomic model, time management is mainly associated with simulation time advance, τ, after a state transition.

A DEVS coupled model describes a multi-scale system and a network of (multi-scale) systems. A network of systems is considered to be a larger coupled model. Each component of a coupled model can be either atomic or coupled. They enable building and maintaining hierarchical and modular model structures based on the property of *closure-under-coupling*. Closure-under-coupling guarantees the aggregation of systems based on their coupling information is also described as a single system in the same formalism.

In addition to providing a formal description of a dynamic event system, DEVS supports hybrid systems construction in various levels, including the trajectory level (e.g., system I/O intercommunication), the formalism level (e.g., Fuzzy-DEVS and Bio-DEVS), and the system level (e.g., Differential Equation System, Discrete Time System, and Discrete Event System) [19].

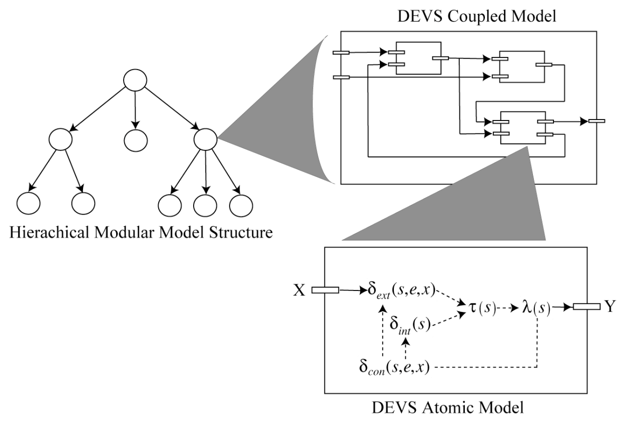


Figure 1. Structural and behavioral properties of DEVS models

BDs are models that delineate structural and behavioral properties of one or more the functional units of a biological system [20]. From a DEVS perspective, BDs are advanced DEVS models that are designed to logically map to biological systems.

Cost Analysis Methodology

Cost analysis is an analytical approach that provides means of transforming heterogeneous *resource* information into homogeneous *cost* information [7]. There is an associ-ated collection of analytical toolkits. The main advantage of the approach is clean separation between application specific knowledge and generic algorithm design. It leads to the construction of concise, generic, configurable algorithms (and systems). In the context of cost analysis, a *cost* is a homogeneous object representing heterogeneous resource information (e.g., single value, a set of discrete objects, and continuous range). A *cost measure* is a conceptual metric that captures heterogeneous resource information in terms of cost. Some of the analytical operations considered in cost analysis are cost harvest, cost generation, cost aggregation, cost evaluation, and cost analysis.

A multi-scale DEVS BD is transformed to a cost tree by applying cost measures and cost aggregation functions. The cost tree represents the cost information of the BD. Various cost measures (e.g., total number of states, dynamic activities, and structural complexity) can be used to obtain or generate cost information.

GENERIC MODEL PARTITIONING WITH LOOKAHEAD *k*

GMP-baseline

GMP-*baseline* is a concise, generic, and configurable model partitioning algorithm for hierarchical modular DEVS models [21]. GMP-*baseline* decomposes a hierarchical model into a set of partition blocks based on cost analysis and a partial flattening approach. A partition block contains more than one component of the coupled DEVS model. The algorithm minimizes model decomposition and produces monotonically improved partitioning results during the simulation process. As a consequence of its clean separation between domain specific knowledge and generic partitioning logic, it can be applied to a variety of partitioning problems. GMP-*baseline* also permits building a collection of different partitioning results for the same model by replacement of cost measures and/or cost functions during the cost tree construction process. However, structural properties of models are related to only cost tree construction, not to GMP based partitioning algorithms.

GMP-*k*

GMP-*k* is an advanced form of GMP-baseline. In addition to all of the above features, GMP-*k* decomposes a multi-level DEVS BD into a set of partition blocks by considering the lookahead parameter *k*. GMP-*k* produces a collection of partitioning results having different resolutions for the same BD by controlling *k*. The algorithm constructs such results by expanding components of a BD up to level *k* either during or just after finding a best partitioning result. Two lookahead schemes are supported: *aggressive look-ahead* and *differed look-ahead*. In the aggressive scheme, *k* level partial expansion occurs during the partitioning process, whereas, in the differed scheme, *k* level expansion is delayed until a best partitioning result is attained.

From the perspective of resource allocation, GMP-*k* produces resource requirements for the nodes in a distributed and parallel computing infrastructure. A node can represent a single host, a cluster of hosts, or a sub network. By regulating the degree of cost homogeneity between elements of each partition block as well as between partition blocks, the results of partitioning can be mapped to one of the above mentioned topologies as shown in Figure 2.

The GMP-*k* algorithm has two steps: *initial partitioning* and *Evaluation-Expansion-Selection (E^2S) partitioning*. The initial partitioning step constructs a set of non-empty partition blocks that represents an inceptive partition result. E^2S partitioning improves Quality of Partitioning until a best partition result is attained. To describe the GMP-k algorithm, we define the following:

Constants:
- T: cost tree
- p: total number of partition blocks
- k: lookahead parameter

Objects:
- PB: partition block
- PB_{empty}: empty partition block
- PB_{lowest}: partition block having the lowest cost
- $PB_{highest}$: partition block having the highest cost
- $PB_{expandable}$: partition block selected to be expanded
- $Node_{lowest}$: cost node having the lowest cost
- $Node_{highest}$: cost node having the highest cost
- $Node_{coupled}$: coupled node
- $Node_{coupled}^{highest}$: coupled node having the highest cost

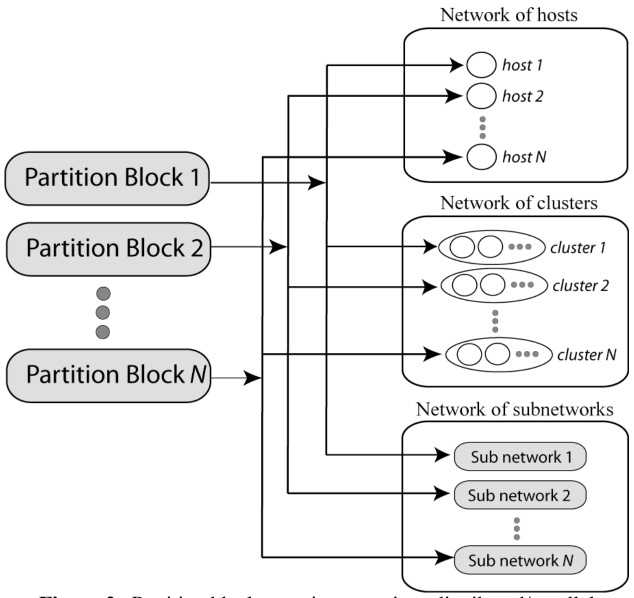


Figure 2. Partition block mapping to various distributed/parallel infrastructures

Functions:
- remove(*node*, *c-list*): remove *node* from *c-list*
  node ← remove(*node*, *c-list*)
- expand(*node*): expand *node*
  all child nodes of *node* ← expand(*node*)
- assignTo(*node,* PB): assign *node* to PB
  boolean ← assignTo(*node*)

Initial Partitioning

The initial partitioning algorithm requires three parameters; a cost tree *T*, the total number of partition blocks, *p*, and a lookahead parameter, *k*. T is obtained by applying a cost analysis methodology to a given multi-level DEVS BD. Various cost measures and functions can be applied. The algorithm is not directly related to cost tree construction; *k* represents a relative offset (or depth) from a cost node to another node that can be reached by traversing the cost tree. The relative offset is represented by total number of links between those nodes. A link is a direct connection between two nodes without any intermediate nodes.

In the aggressive lookahead scheme, the algorithm is divided into five phases: *initialization*, *expansion*, *k-level lookahead computation*, *filling*, and *distribution,* as shown in Algorithm 1. All necessary data structures are created with initial values in the initialization phase (lines 3 – 4); *c-list* and *p-array* are initialized with child nodes of a root node of *T* and *p* number of empty partition blocks, respectively. In the expansion phase, if the length of *c-list* is less than *p*, the coupled node having the highest cost in *c-list* is removed and expanded (lines 6 – 12). All child nodes of the expanded node are stored back to *c-list*. Such expansion continues until the length of *c-list* is equal to or larger than *p*. In the *k*-level lookahead computation, every node of *c-list* is expanded up to level *k* (lines 14 – 21). In the filling phase, every empty partition block is populated with a cost node that is extracted from *c-list* (lines 23 – 25). The cost node having the highest cost is removed first from *c-list*. In the distribution phase, the remaining nodes of *c-list* are distributed to partition blocks (lines 27 – 29). To minimize cost disparity between partition blocks, the cost node having the lowest cost is removed first from *c-list*.

```
1.  procedure PB[] initial-partitioning(T, p, k)
2.    // phase 1 : initialize c-list and p-array
3.    c-list = all child nodes of a root node in T
4.    p-array = create p empty partition blocks
5.    // phase 2 : expand, if necessary
6.    while lengthOf(c-list) < sizeOf(p-array) do
7.      if c-list contains $Node_{coupled}$ then
8.        c-list += expand(remove ( $Node^{highest}_{coupled}$, c-list)
9.      else
10.       return error("can't expand…")
11.     endif
12.   endwhile
13.   // phase 3 : expand every $Node_{coupled}$ in c-list up to k level
14.   depth = 0;
15.   while depth++ < k do
16.     t-list = null
17.     for each $Node_{coupled}$ in the c-list do
18.       t-list += expand(remove($Node_{coupled}$, c-list))
19.     endfor
20.     c-list += t-list
21.   endwhile
22.   // phase 4 : fill empty partition blocks
23.   while p-array contains $PB_{empty}$ do
24.     assignTo(remove ($Node_{highest}$, c-list), $PB_{empty}$)
25.   endwhile
26.   // phase 5 : distribute nodes in c-list into PBs
27.   while c-list is not empty do
28.     assignTo(remove ($Node_{lowest}$, c-list), $PB_{lowest}$)
29.   endwhile
30.   return p-array
31. endprocedure
```

Algorithm 1. Initial partitioning in aggressive lookahead scheme

In the deferred lookahead scheme, the algorithm also consists of the same five phases, but in a different order: *initialization*, *expansion*, *filling*, *distribution,* and *k-level lookahead computation,* as shown in Algorithm 2. The lookahead expansion is performed after the distribution phase (lines 22 - 31).

```
1.  procedure PB[] initial-partitioning(T, p, k)
2.    // phase 1 : initialize c-list and p-array
3.    c-list = all child nodes of a root node in T
4.    p-array = create p empty partition blocks
5.    // phase 2 : expand, if necessary
6.    while lengthOf(c-list) < sizeOf(p-array) do
7.      if c-list contains $Node_{coupled}$ then
8.        c-list += expand(remove ( $Node^{highest}_{coupled}$, 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 $PB_{empty}$ do
15.     assignTo(remove ($Node_{highest}$, c-list), $PB_{empty}$)
16.   endwhile
17.   // phase 4 : distribute nodes in c-list into PBs
18.   while c-list is not empty do
19.     assignTo(remove ($Node_{lowest}$, c-list), $PB_{lowest}$)
20.   endwhile
21.   // phase 5 : expand every $Node_{coupled}$ in all PBs up to k level
22.   depth = 0;
23.   while depth++ < k do
24.     for each PB in the p-array do
25.       t-list = null
26.       for each $Node_{coupled}$ in PB do
27.         t-list += expand(remove($Node_{coupled}$, PB))
28.       endfor
29.       PB += t-list
30.     endfor
31.   endwhile
32.   return p-array
33. endprocedure
```

Algorithm 2. Initial partitioning in deferred lookahead scheme

Evaluation-Expansion-Selection Partitioning

The E^{2}S partitioning algorithm is a recursive algorithm that requires two parameters: partitioning result *p-array*, and a lookahead parameter *k*. In the aggressive lookahead scheme, the algorithm consists of five phases: *initialization*, *k-level lookahead computation*, *filling*, *distribution*, and *evaluation,* as shown in Algorithm 3. In the initialization phase, all necessary data structures are created with initial values (lines 3 – 4); *c-list* and *e-array* are initialized with *null* and the previous partitioning result *p-array*, respectively; *p-array* is initially assigned to partition blocks that initial partitioning algorithm produces. In the *k*-level lookahead computation, all nodes of every partition block in *p-array* are expanded up to level *k* (lines 6 – 15). All expanded nodes are then assigned to *c-list*. The two phases that follow, filling and distribution, are the same as those in the initial partitioning algorithm (lines 17 – 23). In the evaluation phase, the newly created partitioning result is compared to the previous partitioning result (lines 25 – 29). If the new result is superior to the previous one, the E^{2}S algorithm recursively continues until a best partitioning result is attained. Otherwise, it returns the previous partition result.

```
1.  procedure PB[] e-square-s-partitioning(PB[] p-array, k)
2.    // phase 1: initialize e-array and e-partition
3.    c-list = null
4.    e-array = p-array
5.    // phase 2: expand $Node_{coupled}$ in all PBs up to k level
6.    depth = 0;
7.    while depth++ < k do
8.      for each PB in e-array do
9.        t-list = null
10.       for each $Node_{coupled}$ in PB do
11.         t-list += expand(remove($Node_{coupled}$, PB))
12.       endfor
13.       c-list += t-list        // update a partition block
14.     endfor
15.   endwhile
16.   // phase 3: fill empty partition blocks
17.   while e-array contains $PB_{empty}$ do
18.     assignTo(remove($Node_{highest}$, c-list), $PB_{empty}$)
19.   endwhile
```

```
20.     // phase 4: distribute nodes to e-array
21.     while c-list is not empty do
22.        assignTo(remove($Node_{lowest}$, c-list), $PB_{lowest}$)
23.     endwhile
24.     // phase 5: evaluate a new partitioning result
25.     if superiorTo(evaluate(e-array), evaluate(p-array)) then
26.       return e-square-p-partitioning(e-array)
27.     else
28.       retrun p-array
29.     endif
30.   endprocedure
```

Algorithm 3. E^2S partitioning in aggressive lookahead scheme.

In the deferred lookahead scheme, the algorithm consists of seven phases: *initialization*, *identification*, *expansion*, *filling*, *distribution*, *evaluation,* and *k-level lookahead computation,* as shown in Algorithm 4. In the identification phase, a partition block having the highest cost is selected as an expandable partition block (lines 6 – 17). If the expandable partition block contains no coupled node, the second largest partition block is selected to be the expandable block. That process continues until a partition block having a coupled node is found. In the expansion phase, the selected block is expanded (lines 19). In the filling phase, the expanded block is repopulated with a cost node if it becomes empty after expansion with its coupled node. The phases that follow are the same as those in the aggressive scheme (lines 25 – 44).

```
1.    procedure PB[] e-square-s-partitioning(PB[] p-array, k)
2.      // phase 1: initialize e-array and e-partition
3.      e-array =  p-array; e-node = null
4.      e-partition = a $PB_{highest}$ in e-array
5.      // phase 2: identify a $PB_{expandable}$ from e-array
6.      while true do
7.        if e-partition ≡ null then return p-array
8.        else
9.          if e-partition contains $Node_{coupled}$ then
10.           e-node = $Node^{highest}_{coupled}$ in e-partition
11.           if e-node ≠ null then break  else return p-array  endif
12.         else
13.           e-partition = $PB_{highest}$ from e-array excluding
14.                                           previously selected PBs
15.         endif
16.       endif
17.     endwhile
18.     // phase 3: expand e-node and put them into c-list
19.     c-list = expand(remove(e-node, e-partition))
20.     // phase 4: fill the e-partition with $Node_{highest}$
21.     if empty(e-partition) then
22.       assignTo(remove($Node_{highest}$, c-list), e-partition)
23.     endif
24.     // phase 5: distribute nodes to e-array
25.     while c-list is not empty do
26.       assignTo(remove($Node_{lowest}$, c-list), $PB_{lowest}$)
27.     endwhile
28.     // phase 6: evaluate a new partitioning result
29.     if superiorTo(evaluate(e-array), evaluate(p-array)) then
30.       return e-square-p-partitioning(e-array)
31.     else  do
32.     // phase 7 : expand every $Node_{coupled}$ in all PBs upto k level
33.       depth = 0;
34.       while depth++ < k do
35.         for each PB in the p-array do
36.           t-list = null
37.           for each $Node_{coupled}$ in PB do
38.             t-list  +=  expand(remove($Node_{coupled}$, PB))
39.           endfor
40.           PB += t-list
41.         endfor
42.       endwhile
43.       retrun p-array
44.     endif
45.   endprocedure
```

Algorithm 4. E^2S partitioning in deferred lookahead scheme

EXPERIMENTS

We compare GMP-*k* to the *GMP-baseline* algorithm, the *random* partitioning algorithm, and the *ratio-cut* partitioning algorithm, over various partition block sizes and cost patterns. The random partitioning algorithm arbitrarily selects cost nodes and assigns them to a set of partition blocks. The ratio-cut partitioning algorithm cuts a sub tree that has the minimum cost disparity as compared to the average cost of a given cost tree.

The cost patterns used in the experiments are listed in Appendix A. Cost is a real number generated by a cost pattern generator based on a particular probability mass function. All experimental results over various partition block sizes with those patterns are presented in Appendix B. We use the cost disparity between partition blocks as partitioning evaluation measures.

As shown in the Table 1, GMP-*k* produces better partitioning results. In most cases, it produces better partitioning results as *k* increases. However, the larger *k* does not guarantee a better partitioning result.

Table 1. Partitioning results based on the exponential cost pattern

PB	Rand	Ratio	GMP-B	GMP-*1*	GMP-*2*	GMP-*3*
2	90.4	64.1	25.7	15.1	6.7	3.0
3	95.6	75.6	19.2	15.05	7.2	4.4
4	96.0	111.7	21.9	12.8	5.9	4.1
5	114.5	141.6	16.2	12.1	6.6	4.1
6	92.9	124.3	20.5	13.3	7.5	5.4
7	89.9	110.1	22.6	12.9	6.3	4.9
8	88.5	110.0	16.0	11.8	7.3	5.0

PB: partition block size, Rand: random, Ratio: ratio-cut

CONCLUSIONS

A new partitioning algorithm for hierarchical modular DEVS BDs, GMP-*k*, is proposed. It is a concise, generic, and configurable partitioning algorithm; it produces a collection of partition blocks having different partitioning resolutions by controlling lookahead parameter *k*. During the partitioning process, GMP-*k* minimizes model decomposition and produces monotonically improved results. Because it provides a clean separation between domain specific cost analysis of BDs and generic partitioning logic, the new algorithms can be extended to a variety of different partitioning problems.

REFERENCES

[1] T. E. Anderson, D. E. Culler, and D. A. Patterson, “The Berkeley Networks of Workstations (NOW) Project,” Compcon ’95: Technologies for the Information Superhighway, Digest of Papers, March 5-9, 1995.

[2] T. Sterling, D. J. Becker, and D. Savarese, “Beowulf: A Parallel Workstation for Scientific Computation,” Proceedings of the 24th International Conference on Parallel Processing (ICPP), pp. 11-14, Vol. 1, August 1995.

[3] D. S. Milojicic, V. Kalogeraki, et al., “Peer-to-Peer Computing,” HP Lab. Technical Report HPL-2002-57, July 3, 2003.

[4] 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.

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

[6] R. M. Fujimoto, “Parallel and Distributed Simulation Systems,” Wiley Interscience, 1999.

[7] S. Park, "Cost-based Partitioning for Distributed Simulation of Hierarchical Modular DEVS Models", PhD. Dissertation, University of Arizona, May 2003.

[8] A. Pothen, "Graph Partitioning Algorithms with Applications to Scientific Computing," In D. E. Keyes, A. Sameh, V. Venkatakrishnan (eds.), *Parallel Numerical Algorithms*, Kluwer Academic Publishers, pp. 323-368, 1997.

[9] P. Fjallstrom, "Algorithms for Graph Partitioning: A Survey," Linkoping Electronics Articles in *Computer and Information Science*, vol. 3, 1998.

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

[11] 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.

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

[13] 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

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

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

[16] Y. Zhao and G. Karypis, "Evaluation of Hierarchical Clustering Algorithms for Document Datasets," *CIKM 2002*, 2002.

[17] 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.

[18] 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.

[19] H. Vangheluwe, "DEVS as a common denominator for multi-formalism hybrid systems modeling," presented at IEEE International Symposium on Computer-Aided Control System Design, Anchorage, Alaska, 2000.

[20] C. A. Hunt, G. E. P. Ropella, M. S. Roberts, and L. Yan, "Biomimetic In Silico Devices," Computational Methods in Systems Biology 2004 (CMSB), Paris, France, 2004.

[21] S. Park and B. P. Zeigler, "Distributing Simulation Work Based on Component Activity: A New Approach to Partitioning Hierarchical DEVS Models ", CLADE 2003, June 2003.

[22] V. Cardellini, M. Colajanni, and P. S. Yu, "Request Redirection Algorithms for Distributed Web Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, pp. 355-368, 2003.

ACKNOWLEGEMENT

This research has been supported in part by CDH Research Foundation Grant No. R21-CDH-00101, “Fundamentals of Agent-Based Models for Biological Systems.”

BIOGRAPHIES

Sunwoo Park is a postdoctoral fellow of Biopharmaceutical Sciences and Bioengineering at UCSF. His research areas include distributed/parallel modeling and simulation, discrete event systems, systems biology and computational biology, and theoretical computing.

Dr. C. Anthony Hunt has served as a Professor of Biopharmaceutical Sciences and Bioengineering at the University of California, San Francisco (UCSF) for over 20 years. Three of his areas of interest are Systems Biology, Computational Biology and the Pharmaceutical Sciences.

Bernard P. Zeigler is Professor of Electrical and Computer Engineering at the University of Arizona, Director of the Arizona Center for Integrative Modeling and Simulation, an IEEE Fellow, and the 2000 recipient of the Society for Computer Simulation’s McLeod Founder’s Award.

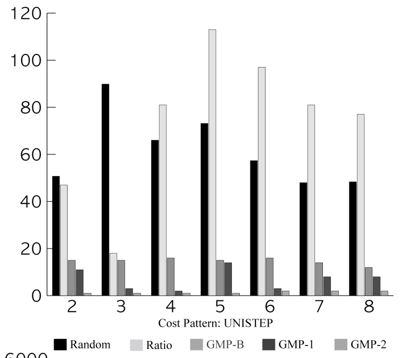
APPENDIX

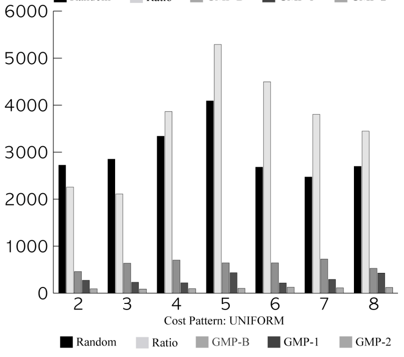
Part A. A collection of cost patterns used for cost generation of DEVS models [22]

Cost Pattern	PMF	Parameters	Distribution
$P_{unitstep}(x)$	$\delta(x)$	None	Unit Step
$P_{uniform}(x)$	1	None	Uniform
$P_{exponential}(x)$	$\lambda e^{-\lambda x}$	$\lambda = 0.05$	Exponential
$P_{invgaussian}(x)$	$\sqrt{\frac{\lambda}{2\pi x^3}}e^{\frac{-\lambda(x-\mu)^2}{2\mu^2 x}}$	$\mu = 3.86$, $\lambda = 9.46$	Inverse Gaussian
$P_{pareto}(x)$	$\alpha k^{\alpha} x^{-\alpha-1}$	$\alpha = 1.245$, $k = 3$	Pareto
$P_{lognormal}(x)$	$\frac{1}{x\sqrt{2\pi\sigma^2}}e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}}$	$\mu = 5.929$, $\sigma = 0.321$	Lognormal

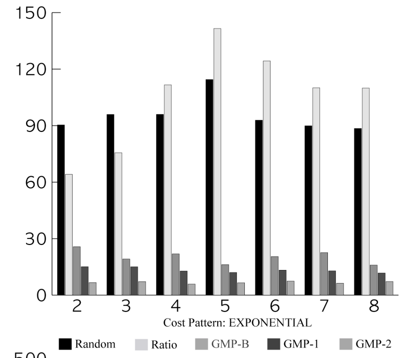
PMF: Probability Mass Function; $\delta(x)$: PMF that returns 1 when $x = a$. otherwise, returns 0

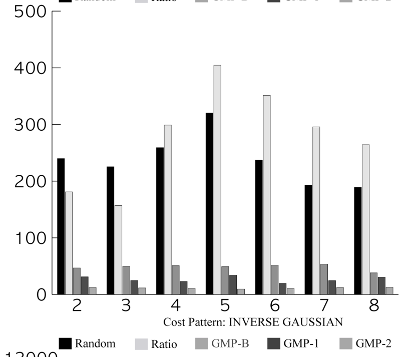
Part B. Partitioning results for various cost patterns and partitioning block sizes

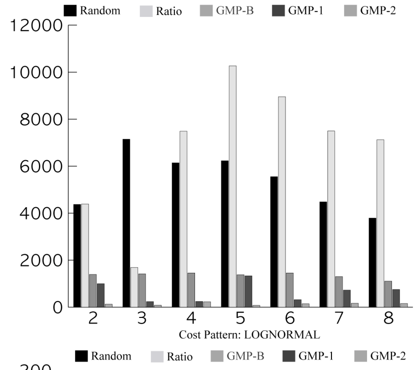


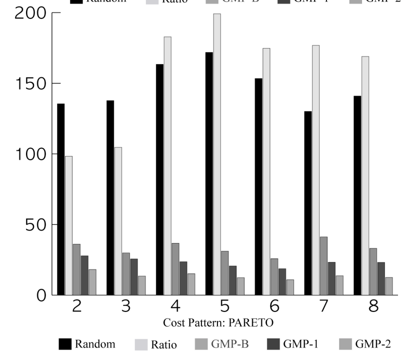


For a DEVS coupled model satisfying T(7, 4, 211), we perform 20 experiments per each cost pattern while changing the partition block size from 2 to 8 and compute average cost disparity between partition blocks. T(7, 4, 211) is the cost tree that consists of the depth, 7, the number of child nodes of a coupled node, 4, and the total number of atomic nodes in the tree, 211. Three different lookahead parameter values (i.e., 1, 2, and 3) are used in GMP-k partitioning with the aggressive mode.


Cost Pattern: EXPONENTIAL
Random
Ratio
GMP-B
GMP-1
GMP-2


Cost Pattern: INVERSE GAUSSIAN
Random
Ratio
GMP-B
GMP-1
GMP-2


Cost Pattern: LOGNORMAL
Random
Ratio
GMP-B
GMP-1
GMP-2


Cost Pattern: PARETO
Random
Ratio
GMP-B
GMP-1
GMP-2