Exploiting Symmetry for Partitioning Models in Parallel Discrete Event Simulation

Jan Lemeire[1], Bart Smets[1], Philippe Cara[2], Erik Dirkx[1]
[1]*Parallel Systems lab,* [2]*Department of Mathematics*
Vrije Universiteit Brussel (VUB)
Brussels, Belgium
Email: {jlemeire,bsmets,pcara,erik.dirkx}@vub.ac.be

Abstract

We investigated the benefit of exploiting the symmetries of graphs for partitioning. We represent the model to be simulated by a weighted graph. Graph symmetries are studied in the theory of permutation groups and can be calculated in polynomial time with the nauty algorithm [15]. We designed an algorithm to extract useful symmetries from the automorphism group, which can be used to create partitions derived from the graph's structure. Our approach is focused on composite graphs, for which identical subgraphs reoccur in the graph. If these identical subgraphs can be mapped onto each other by symmetries, the subgraphs are replaced by equivalent multivertices, resulting in a 'natural' aggregation of vertices. This approach is applied to parallel simulation of a detailed IP-switch with a conservative synchronous algorithm. The experimental results show that even for good partitions, global and temporal load imbalances are inevitable.

1. Introduction

We study the problem of finding good model partitions for Parallel Discrete Event Simulation (PDES). This problem is equivalent to graph partitioning, where the model is represented by a weighted graph. The weight of an edge represents its amount of communication and the weight of a node its amount of computation. Graph partitioning is an important problem that has extensive applications in many other areas, including scientific computing, VLSI design, data mining and grid applications. The goal is to partition the vertices of a graph in roughly *p* equal parts to balance the graph, such that it minimizes the number of edges connecting vertices in different parts. These are the edge-cuts, representing the communication between the different parts.

We investigated how symmetries can be used to partition the graph. This work started with the work Bart Smets [16]. Naturally, symmetry is not guaranteed and symmetric partitions are no guarantee for efficient partitions. However, models in PDES, like network or electronic components, are often highly structured. They often consist of reoccurring *components* with more communication inside the components then between them. These structural properties of the model result in what we called *composite graphs* and can be exploited to get an 'natural' partition.

1.1. Parallel Discrete Event Simulation

In *discrete event simulation,* the device under study is modeled by interconnected *logical processes*, in which state changes happen at discrete times by *events*.

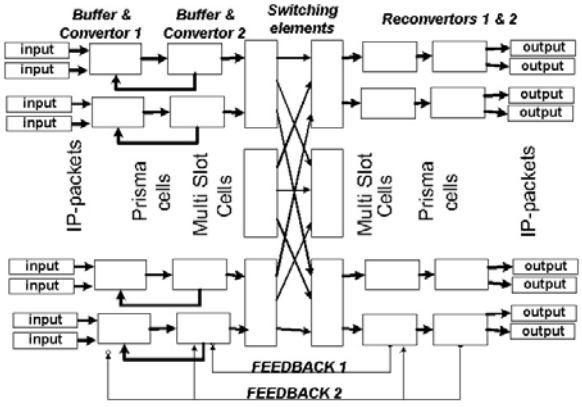


Figure 1. IP-switch model

Figure 1 shows the model of a detailed IP-switch [8], in which the IP-packets are first converted into prisma cells, then into multi-slot cells before going through the switching elements. After this, the original IP-packets are reconstructed and sent to the output. Feedback signals try to prevent packet loss due to congestion in the switching elements. Packets are therefore buffered into the input stages. The switch contains 32 inputs, resulting in more than 4000 processes.

In *Parallel Discrete Event Simulation* (PDES) [5,6] the model is partitioned among the available processors and simulated with a parallel algorithm that synchronizes

the different partitions. We will explain our approach with the simplified model of the switch (Figure 2), whereas the experimental results are gathered from the simulation of the detailed model.

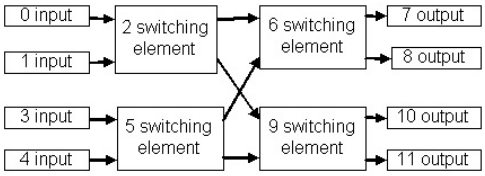


Figure 2. Simplified switch model

1.2. Related Work

The partitioning task is known to be NP-complete in general [7]. However, when the graph exhibits certain regularities, good partitions can be found in polynomial time. An example is METIS [13], a multilevel coarsening algorithm, in which highly interconnected nodes are successively grouped (step 1) to become a much smaller graph with lower partitioning complexity. After partitioning of the aggregated graph (step 2), the partitions are gradually refined when the multivertices are expanded to reconstruct the original graph (step 3). In [14], the assumptions on the graph for the success of the algorithm are analyzed.

Graph partitioning is defined in section 2 and graph symmetries in section 3. Section 4 discusses the symmetry tree that is used in section 5 for partitioning. Finally, section 6 shows the experimental results.

2. Graph Partitioning

A graph contains the relational information of an object. It is defined as a structure $G=<V, E>$ in which V is a finite set of *vertices* (or *nodes*) and $E \subset V \times V$ is a finite set of *edges* (unordered pairs or ordered pairs for a directed graph). Two vertices u and v are called *adjacent* if $(u,v) \in E$. A *weighted graph* associates a number $|v|$ to each vertex and a number $|e|$ to each edge.

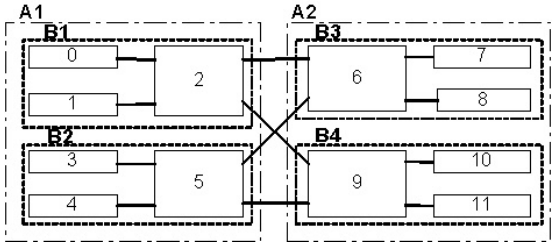


Figure 3. A composite graph

We define a *composite graph* is a graph that is composed of subgraphs that occur multiple times in the graph. Identical subgraphs can then be replaced by a single *multivertex*. Structured models often contain such subgraphs, as shown in Figure 3, in which nodes can be replaced by the multivertices A_1, B_1, etc. In fact, the vertices of Figure 3 are on their turn composed out of subgraphs of the detailed IP switch model of Figure 1. Furthermore, each process of Figure 1 can again be described in more detail, as shown in Figure 4, which is the detailed description of the buffer&convertor1 process of Figure 1. This is called the *level of abstraction* of the model. We study the graphs for which these subgraphs are symmetric. Note that we are now working with undirected graphs, since the direction of the communication is irrelevant for the partitioning.

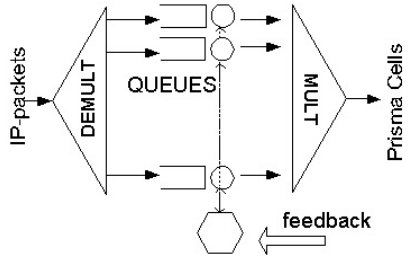


Figure 4. The buffer&convertor1 process in detail

Graph partitioning is defined as follow: given a graph $G=<V,E>$, partition V into p disjoint subsets $V_1 \ldots V_p$ such that the following objectives are optimized:

1. The sum of the weights $|v|$ of all v of V_i for each i $(1 \ldots p)$ is the same.
2. The sum of the weights of the cut edges (edges for which the nodes belong to a different partition) is minimized.

3. Graph Symmetries

Symmetries are investigated in mathematics by Group Theory [12]. A *symmetry* of an object is defined as a transformation that leaves the essential features of the object unchanged [10]. The set of all symmetries of an object form a *group*. This means that symmetries can be composed, an identity transformation exists and each transformation has an inverse transformation.

A *graph symmetry* is a permutation f of the vertex set V that preserves adjacency:

$$\forall v_1, v_2 \in V : (v_1, v_2) \in E \Leftrightarrow (f(v_1), f(v_2)) \in E \quad (1)$$

The group of all such permutations together with composition is called the automorphism group *Aut(G)* of the graph [3]. A permutation is a bijective transformation. It can be described with the *cyclic notation*. For example, the permutation (a_1, a_2, a_3) carries a_1 to a_2, a_2 to a_3 and a_3

to a_1. The permutation $(a_1, a_2)(b_1, b_2)$ carries a_1 to a_2, a_2 to a_1 and simultaneously b_1 to b_2 and b_2 to b_1. Some of the symmetries of the graph of Figure 2 are:

(0, 1)
(3, 4)
(0, 3) (1, 4) (2, 5)
(7 8)
(10 11)
(7 10)(8 11)(6 9)
(0 7)(1 8)(2 6)(3 10)(4 11)(5 9)

Note that all other symmetries can be composed out of these, therefore they are called a set of *generators* for *Aut(G)*.

An efficient and widely used algorithm for finding the symmetries in a graph is *nauty* [15, see also http://cs.anu.edu.au/~bdm/nauty]. The average performance is polynomial, with a degree no bigger than 4 and in practice less than 3 [15]. A handy tool for finding and manipulating graph symmetries, using the nauty algorithm, is GAP [http://www.gap-system.org].

We will analyze the symmetries in composite graphs. Note that reoccurring multivertices are called *isomorphic* subgraphs in group theory. We define a *multivertex symmetry* as a permutation that maps multivertices onto each other and preserves adjacency of these. It is clear that the multivertices of Figure 3 are symmetric. These multivertex symmetries are interesting for optimizing the partitioning task. Therefore we have to extract them from *Aut*(*G*) and describe them into a practicable representation.

4. The Composite Symmetry Tree

We represent the composite graph by a hierarchical tree of reoccurring multivertices, where each multivertex is refined in the next level. Figure 5 shows the composite symmetry tree of the switch in Figure 2. At each level, equivalent multivertices are represented by the same symbol, they are mutually interchangeable by a symmetry.

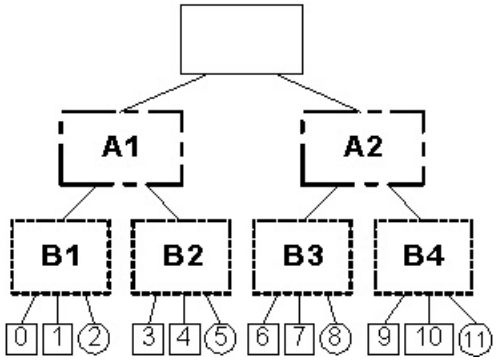


Figure 5. The composite symmetry tree of the simplified switch model

In the work of Smets [16], the definition of the tree is extended for *ring symmetries*. This is represented by connecting the nodes which are neighbors in the ring transformation.

4.1. Additional Definitions

For the construction of the tree, we need some concepts from the theory of permutation groups [9]. The *orbit* of a vertex $v \in V$ is defined as: $orb(v) = \{g(v): g \in Aut(G)\}$, the set of all vertices on which v can be mapped by the automorphisms of the graph. If a vertex $v' \in orb(v)$, then v is also an element of the $orb(v')$. In this way, the orbits partition the vertices of the graph into disjoint subsets (Figure 6).

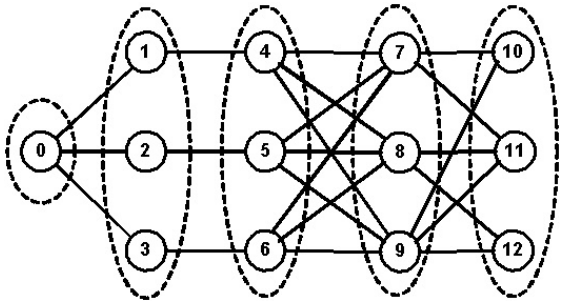


Figure 6. Orbits of a graph

The *stabilizer* of a subset V' of V, are the symmetries that keep all elements of V' unchanged:

$$Aut(G)_{V'} = \{g \in Aut(G) \mid \forall v \in V' : g(v) = v\} \quad (2)$$

A *block of imprimitivity* is a set of vertices $B \subseteq V$ for which:

$$\forall g, g' \in Aut(G) : g.B = g'.B \quad or \quad g.B \cap g'.B = 0 \quad (3)$$

A block of imprimitivity is completely mapped onto itself by the symmetries *or* onto a disjoint set of vertices. The other set will necessarily also be blocks of imprimitivity. A *maximal block of imprimitivity* of a vertex set V' is an imprimitivity block $B \subset V'$ of maximal size, but non-trivial ($B=V'$). These concepts can be calculated by GAP/nauty.

4.2. Construction Algorithm

The intuitively defined symmetric components are clearly blocks of imprimitivity. We will however only aggregate vertices if an objective choice exists and if the

number of connections between the different multivertices is low.

The principle of the algorithm is to aggregate vertices of different orbits. For the graph of Figure 6, the goal is to aggregate the vertices 1 and 4 into a multivertex, that is isomorphic to multivertices {2,5} and {3,6}. However, we will only group vertices that are adjacent to exactly one vertex of another orbit. In other cases, when all or multiple vertices of one orbit are connected to the same vertex, aggregation makes no sense. When all vertices are interconnected, as for 2, 5 and 8 with 0, and {4,5,6} with {7,8,9} in Figure 6, both orbits can permute independently and there is no objective reason to group vertices. If only some, but more than one, vertices are adjacent, as for the vertices of orbit {7,8,9} with orbit {10,11,12} in Figure 6, vertices of both orbits cannot transform independently. By permuting some of the vertices of one orbit, some of the other should also permute. Then, there are multiple equivalent choices in grouping the vertices of the different orbits. Moreover, grouping such vertices would cause a lot of communication between the multivertices. Hence, we will also not aggregate vertices in such cases.

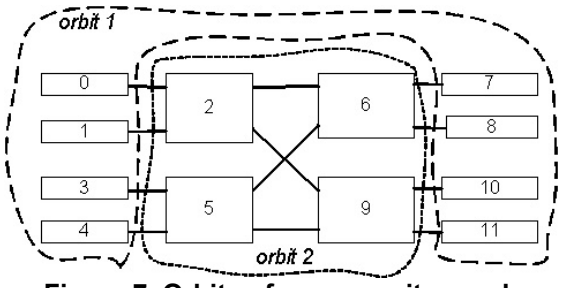


Figure 7. Orbits of a composite graph

For complex composite graphs, a 'hierarchy' can be identified inside orbits, as in Figure 7. To apply the aggregation method, we should start with grouping the *maximal imprimitivity blocks* of the orbits, shown in Figure 8. By applying the above algorithm, block *1a* can be aggregated with block *2a*, and block *1b* with *2b*. They form the top-level components A_1 and A_2 of the symmetry tree of Figure 5.

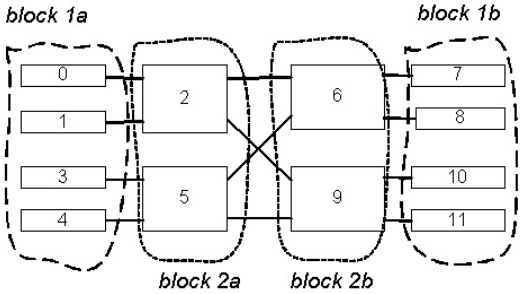


Figure 8. Maximal blocks of imprimitivity of the orbits

The symmetry tree is recursively built in a top-down way. For deeper investigation of the symmetries of a multivertex, all other multivertices are *stabilized*. The new automorphism group $Aut'(G)_{V'}$ is then used to find the subcomponents with the same aggregation method. Since components are isomorphic, all subcomponents found in one component also appear in the other components. Applied onto the graph of the switch (Figure 3), the symmetric supervertices A_1 and A_2 can be refined into B_1, B_2, B_3 and B_4. This gives us the desired symmetry tree of Figure 5.

4.3. Performance

The experimental performance results on a 1.8GHz Pentium 4 processor are shown in Table 1. We compared the processing time of the algorithm for a random graph, the simplified switch of Figure 2 and the detailed switch of Figure 1, for a varying number of vertices.

Table 1. Processing time (in *seconds*) of symmetry tree construction in function of the number of vertices

#vertices	**1000**	**2000**	**3000**	**4000**
Random graph	0.014	0.091	0.165	0.242
Simplified switch	9.35	132	480	1331
Detailed switch	8.87	116	413	983

It is known that nauty quickly detects that graphs have few symmetries, namely in polynomial time of degree 2 [15]. This is confirmed by the experiments that give a quadratic processing time for a random graph without symmetries. For the simplified and detailed switch we get a third degree polynomial dependency.

5. Exploiting the Symmetry Tree for Partitioning

With the Composite Symmetry Tree, proposals for graph partitions can easily be extracted. The simplified switch can be partitioned along 2 axes, as shown in Figures 9 and 10. A switch with more inputs will have axes parallel with Ax 2 and partitioning along these axes will give good results.

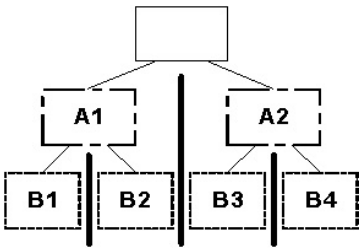


Figure 9. Partitioning with the symmetry tree

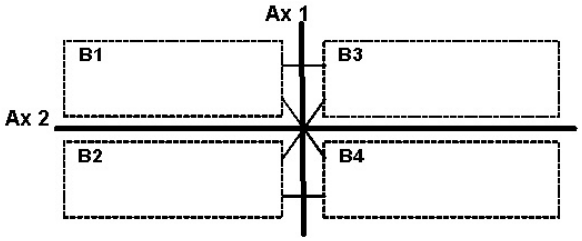


Figure 10. The resulting model partitioning

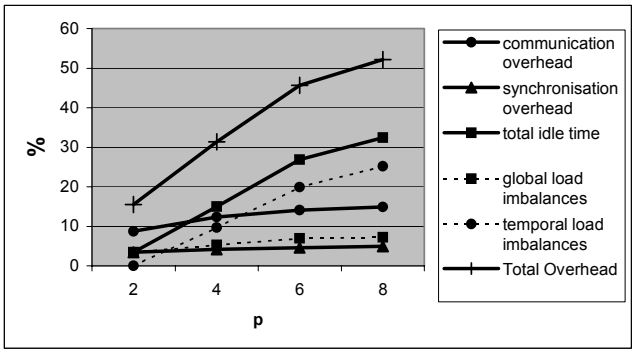
The partitions result from the structure of the graph, so it is a kind of 'natural' partitioning. The symmetries guarantee an equal distribution of the workload and by construction, the communication is minimized.

Note that the approach resembles the aggregation method of METIS and is a kind of 'natural' aggregation of the vertices according to symmetries.

6. Simulation of IP Switch

The illustrating experiment is the simulation of the detailed IP-switch [8] of Figure 1 on a cluster of 8 Pentium II processors of 333MHz connected by a 100Mb/s non-blocking switch. For parallel simulation, we use a conservative synchronous algorithm [1], based on the time window algorithms [5]. It consists of cycles of independent simulation on each processor, alternated with synchronization between the different processors by communicating the events traveling through the cut-edges. The cycle length or window size equals the *lookahead*.

Table 2. Experimental results for the parallel overheads



The switch has 32 in- and output channels and is fed with random TCP/IP traffic according to a normal distribution. It puts an average load of 62% on the switching elements. It is partitioned along the horizontal symmetry axes (parallel with Ax 2 in Figure 10). The experimental results in function of the number of processors are shown in Table 2. The values for the speedups are 1.7 for p=2, 3.0 for p=*4*, 4.1 for p=*6* and 5.2 for p=8.

The parallel simulation time with 6 processors of 1s real time was 1683s, resulting in a speedup of 4.06. The total overhead of 45.6% of the parallel runtime was mainly caused by communication overhead overlapping with computation (14.1%) and load imbalances (26.9%). The synchronization overhead of our algorithm only took 4.6% of the parallel runtime. The relatively high communication overhead is caused by the packets traveling through the edge-cuts. This intercommunication is unavoidable. On the other hand, the high load imbalances suggest a bad partitioning, but this is not the case. The total idle time T_{idle} is measured as the sum over all cycles that the processors have to wait for the slowest processor, since the simulation happens synchronously in cycles:

$$T_{idle} = T_{gli} + T_{tli} = p.\sum_{k \in cycles} (SimT^k_{\max} - SimT^k_{avg}) \quad (4)$$

where $SimT^k$ is the simulation time of cycle k, the maximal and average respectively. We identify 2 different types of load imbalances that cause idling: the *global* and *temporal load imbalances*. The global load imbalance T_{gli} is caused by unequal total simulation time *TotalSimT* between the processors:

$$T_{gli} = p.(TotalSimT_{\max} - TotalSimT_{avg}) \quad (5)$$

The temporal load imbalances T_{tli} on the other hand, are caused by fluctuations of the load imbalances between the different cycles and can be calculated as follow:

$$T_{tli} = T_{idle} - T_{gli} \quad (6)$$

The slowest processor that causes the idle time at each synchronization point can differ from cycle to cycle. The resulting global load imbalance, which is an average of the load imbalance over all cycles, can thus be low (here: 7%), whereas temporal fluctuations are generating a much higher idling of the processors (here:19.9%).

Since our model is completely symmetric, with the same average, but stochastic, load in each of the symmetric components, no better partitioning is available. However, the deviation from the average load leads to a global imbalance and load fluctuations generate high temporal imbalances as an effect of using a synchronous parallel simulation algorithm. Table 2 shows how the temporal imbalances increase rapidly with the number of processors and cause the main limitation on the growth of the speedup.

7. Conclusions

We argued that if a graph has symmetries, these should be exploited for partitioning. We constructed an algorithm that uses the theory of permutation groups to find *multivertex symmetries* in composite graphs. These useful symmetries are represented in a *symmetry tree*, from which proposals for partitions can easily be constructed. The application of this approach to our highly structured models yields promising results.

We want to extend the algorithm to other types of symmetries and to quasi-symmetries. Moreover, symmetrical properties are useful in reducing the complexity of algorithms in general and should be reflected in the visualization of such models and their graphs.

8. References

[1] W. Brissinck, "Tuneable Granularity Parallel Discrete Simulation." *PhD thesis*, Vrije Universiteit Brussel, Brussels, May 1999.

[2] R.E. Bryant, "Simulation of Packet Communications Architecture Computer Systems." *Technical Report* MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[3] Peter J. Cameron, "Automorphisms of Graphs", Queen Mary University of London, 2001.

[4] K.M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", *IEEE Trans. on Softw. Eng.* SE-5, 5 , 440-452, Sept 1997.

[5] A. Ferscha, "Parallel and Distributed Simulation of Discrete Event Systems", *Handbook of Parallel and Distributed Computing,* McGraw-Hill, 1995.

[6] R.M. Fujimoto, "Parallel Discrete Event Simulation", *Communications of the ACM*, 33, pp 29-53, Oct 1990.

[7] M. Garey, D. Johnson and L. Stockmeyer, "Some simplified NP-complete graph problems", *Theoretical Computer Science*, 1976.

[8] S. Geudens, "Quantitative Study of a Highly Formant Network Switch with Distributed Simulation", *Master Thesis*, Vrije Universiteit Brussel, Brussels, 2000.

[9] M. Jr. Hall, "The Theory of Groups", Macmillan, New York, 1976.

[10] R. Howlett, "Aspects of Symmetry", *Lecture Notes*, Sydney (http://www.maths.usyd.edu.au:8000/u/bobh/2gwhole.pdf).

[11] G. Israel and M. Wilhelm, "Groups and Their Graphs", The Mathematical Asociation of America,1965

[12] John F. Humphreys, "A Course in Group Theory", Oxford University Press, 1996.

[13] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *Technical Report* 95-035, Dept. Computer Science, Univ. Minnesota, Minneapolis, Minnesota, 1995.

[14] G. Karypis and V. Kumar, "Analysis of Multilevel Graph Partitioning". *Proceedings of Supercomputing'95*, San Diego, 1995.

[15] B. McKay, "Practical Graph Isomorphism", *Congressus Numerantium* 30, 45-87, 1981.

[16] B. Smets, "Symmetrie-onderzoek van Graffen", *Master thesis*, Vrije Universiteit Brussel, Brussels, 2003.