Identifying Parallel Jobs for Multi-Physics Simulators Scheduling

Renata M. de Carvalho, Ricardo M. F. Lima, Adriano L. I. de Oliveira
Center of Informatics
Federal University of Pernambuco
Recife, Pernambuco, Brazil
rwm@cin.ufpe.br, rmfl@cin.ufpe.br, alio@cin.ufpe.br

Felix Christian Guimarães Santos
Department of Mechanics
Federal University of Pernambuco
Recife, Pernambuco, Brazil
fcgs@demec.ufpe.br

Abstract—**Real problem simulations involving physic phenomena can demand too much execution time. To improve the performance of these simulations it is necessary to have an approach to parallelize the processes that compose the simulation. MPhyScaS (Multi-Physics and Multi-Scale Solver Environment) is an environment dedicated to the automatic development of simulators. Each MPhyScaS simulation demands a great amount of time. To parallelize MPhyScaS simulations, the approach used should define a hierarchical parallel structure. The aim of the work herein presented is to identify parallel jobs and dependent ones. The presented model is based on Coloured Petri Nets (CPN). This information will be input to a scheduling algorithm.**

Index Terms—**Parallel jobs, dependencies, Coloured Petri Nets, scheduling.**

I. Introduction

MPhyScas (Multi-Physics Multi-Scale Solver Environment) is a computational system dedicated to the automatic development of simulators based on the finite element method [1]. These simulators models the behavior of a set of interacting phenomena in space and time. These phenomena are usually of different nature (deformation of solids, heat transfer, etc.) and may be defined in different scales of behavior (macro and micro mechanical behavior of materials). During the simulation, part of one phenomenons data may depends on information from other phenomenon. Such dependent phenomena are said to be coupled. The dependency between phenomena may occur in any geometric part, where both phenomena are defined. The number of phenomena may be very high and the dependencies between them is usually extremely complex. Hence, this kind of problems tend to be very costly. In particular, identify the dependency between coupled phenomena is not a trivial task. At the same time, discovery these dependencies is essential to develop efficient parallel simulators. The sequential version of MPhyScas tool does not provide an automatic mechanism to determine how different phenomena interact nor what data they share or depend upon. Therefore, a programmer interested in create a parallel version of a MPhyScas' simulator needs to manually analyze the simulator to extract such information.

In this paper, we explore the modular layered architecture of MPhyScas to automatically identify data dependencies between phenomena in multi-physics and multi-scale simulators. The automation of this task may dramatically reduce the costs to develop parallel simulators. Moreover, it contributes to remove errors caused by imprecise representation of data dependencies in the parallel simulator.

We adopted a formal approach, based on Coloured Petri Nets (CPN) [2], to model the simulator structure. In MPhyScas, the simulator is represented using a XML file format. We developed a compiler to transverse the XML representation of the simulator and automatically translate it into the corresponding CPN model. The generated model is then simulated using the CPNTools [3] to extract the dependency graph, which is used to assess the data dependency in the simulator structure.

As a final contribution of this work, we proposed a genetic algorithm to calculate a schedule for parallel simulator in a multiprocessor environment. To calculate the schedule, the algorithm uses data dependence information obtained through the CPN model and the configuration of multiprocessor environment, including communication costs and processors' speed. A case study is conducted to evaluate the performance of parallel simulators produced using our genetic algorithm.

This paper is organized as follows. Section II presents some of the related works. Section III details the MPhyScaS definition and its properties. The methodology used in the hole work is presented in Sections IV. Section V and VI describe a model definition based on CPN to represent MPhyScaS data and how the model is analyzed in order to identify dependencies. Section VII describes how the dependencies information is used as input to a GA algorithm. Sections VIII and IX show the experiments and results obtained, respectively. And the conclusions of the proposed work are presented in Section X.

II. Related Work

In this section we discuss some meta-heuristic proposed to deal with the scheduling problem.

Xu *et al.* [4] used Simulated Annealing (SA) for fixed job scheduling problem. Their objective was to minimize assignment cost of the sequential network model. In [5], Kang *et al.*proposed a discrete variant of Particle Swarm Optimization (PSO) to solve job scheduling problems which all tasks are non-preemptive and independent of each other. Thus, there is no communication cost to be worried about. Chong *et al.* [6] relied on nectar collection of honey bee colonies to create an algorithm for solving scheduling problem.

978-1-4244-6588-0/10/$25.00 ©2010 IEEE

They consider only the makespan function to compare its algorithm. The algorithms cited above are of the improvement type. They start out with a complete schedule, which may be selected arbitrarily, and then try to obtain a better schedule by manipulating the current schedule.

Among these methods, the Genetic Algorithm (GA) has emerged as a tool that is beneficial for a variety of study fields. Kim [7] proposed a permutation-based elitist genetic algorithm that used serial schedule generation scheme for solving a large-sized multiple resource-constrained project scheduling problem. Kim consider the number of resources, but do not consider waiting time and communication costs. Moattar *et al.* [8] proposed a GA based algorithm that finds schedules where jobs are partitioned between processors in which total finishing time and waiting time are minimized. As we did, they used a fitness function based on aggregation to optimize two criteria simultaneously, but they did not aggregate the communication cost to their function. Yang *et al.* [9] also relies on GA for solving scheduling problem, but they have a different optimization criteria: close the gap between the specification of concurrent, communicating processes and the heterogeneous processor target without compromising required real-time performance and cost-effectiveness.

III. MPhyScaS

MPhyScaS (Multi-Physics and Multi-Scale Solver Environment) is an environment dedicated to the automatic development of simulators based on the Finite Element Method (FEM [10]). The term multi-physics is a qualifier to a set of phenomena that interact in time and space. A multi-physics system can also be called a system of coupled phenomena. These phenomena are of different natures and behavior scale.

Usually, simulators based on FEM can be organized in a layers architecture [11]. In the top layer global iterative loops can be found, corresponding to overall scenery of the simulation. The second layer contains what is called solution algorithms which dictates the way linear systems are built and solved. The third layer contains the solvers for linear systems and all the machinery for operating with matrices and vectors. The last layer is the phenomenon layer, which is responsible for computing local matrices and vectors at the finite element level and assembling them into global data structures.

The MPhyScaS architecture establishes a computational representation for the computational layers using patterns (Figure 1). Kernel level represents the global scenery level, the level of the solution algorithms is represented by Block level, the level of solvers is represented by Group level, and the phenomena level is represented by Phenomenon level.

The original architecture of MPhyScaS provides support to the automatic building of sequential simulators only. For instance, it does not have abstractions that could automatically define the distribution of data and procedures and their relationships across a cluster of PC's. The MPhyScaS parallel architecture (called MPhyScaS-P) satisfy a number of new requirements, including the support of parallel execution of the simulators in clusters of PC's.

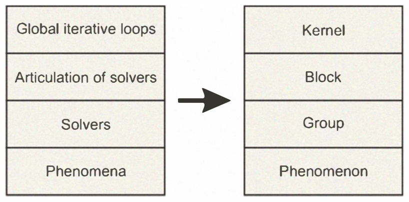


Fig. 1. Computational representation for the layers of the simulator.

The topology of the procedures in the workflow of MPhyScaS-S is implemented in MPhyScaS-P in a hierarchical form with the aid of a set of processes, which are responsible for the procedures synchronization. There are three types of leader processes (see Figure 2):

- **Cluster Rank Process:** It is responsible for the execution of the Kernel and to synchronize the beginning and the end of each one of its level's tasks, which requires demands to the lower level process. In a simulation there is only one ClusterRank process;
- **Machine Rank Process:** One process is chosen among all processes running in an individual machine to be its leader. It is responsible for the execution of procedures in the Block level and to synchronize the beginning and the end of each one of it's level's tasks, which requires demands to lower level processes;
- **Process Rank Process:** It is responsible for the execution of the procedures in the Group level. The ClusterRank and all MachineRank processes are also ProcessRank processes.

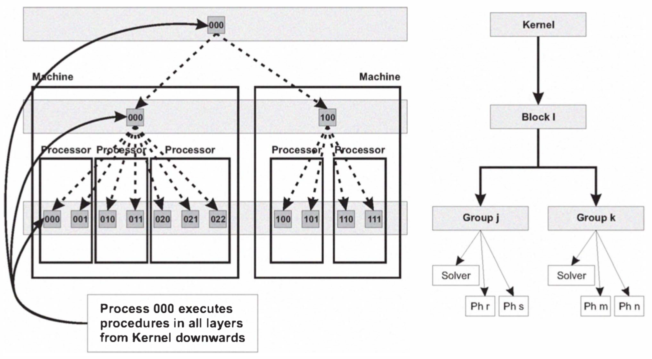


Fig. 2. Layers with procedures executed by ClusterRank in MPhyScaS-P.

IV. Methodology

In this section, we describe the methodology used to identify the dependencies between MPhyScaS processes. The methodology defines an automatic process whose result is given as input to our scheduling algorithm. Figure 3 illustrates such a process.

The first step of the methodology automatically creates a Petri net model to represent the data structure of a given MPhyScaS application. In particular, we use a high level Petri net called Coloured Petri Nets (CPN). This kind of Petri net has strongly typed tokens. Thus, depending on its type, a token

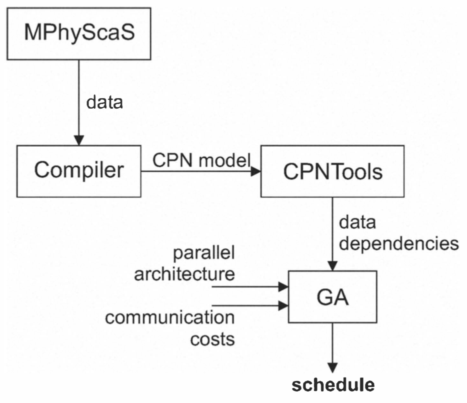


Fig. 3. An overview of the proposed methodology.

might store a complex data type. We explore this feature during the model simulation to store the identification of tasks being executed as well as their timestamps. The model is simulated through the CPNTools [3]. The simulation output exposes the data dependencies between MPhyScaS' processes. This information is given as input to the Genetic Algorithm (GA) to generate the schedule. The complexity of the problem relies on the data dependence because the size of some required data might be very large, increasing communication costs. The GA also takes into account the architecture where the application will execute as well as the communication costs involved.

V. Petri Net Model

The CPN model is automatically generated from the XML file extracted from the MPhyScaS framework. Such a XML file contains a complete description of a particular MPhyScaS simulator. Examples of the information extracted from the XML file to produce the CPN model are: the data produced and consumed by each process, the size of each data processed by the simulator, and the position of each process in the MPhyScaS hierarchy. The MPhyScaS hierarchical structure restricts the way each process can communicate with others. Therefore, this information is implicitly used to generate the CPN model for a particular MPhyScaS simulator.

In our CPN model, each object of MPhyScaS architecture is represented by a place, and each request from an object to another is represented by a transition. The transitions have guards, so that only tokens with a particular value can enable a transition to fire. During the model simulation tokens store the identification of tasks being executed as well as their timestamps.

The relationship between a layer i and the layer above it (layer $i+1$) is formally described in Definition 5.1.

Definition 5.1 (CPN model for processes communication): In the CPN model created to represent the communication between the layer i and the layer $i+1$ (the layer above i), $T_{i,i+1}$ be a set of transitions modeling the communication between the layers i and $i+1$, $P_i = O(T_{i,i+1})$ be a set of places representing the processes at level i and $P_{i+1} = I(T_{i,i+1})$ be a set of places representing the processes at level $i+1$, the following restrictions must hold:

1) $\#P_i = \#T_{i,i+1}$
2) let $p_{ia}, p_{ib} \in P_i, \quad p_{ia} \neq p_{ib}$,
$\nexists t_{i,i+1} \in T_{i,i+1} \quad | \quad (p_{ia} \in {}^{\bullet}t_{i,i+1}) \wedge (p_{ib} \in {}^{\bullet}t_{i,i+1})$
3) let $p_{i+1a}, p_{i+1b} \in P_{i+1}, \quad p_{i+1a} \neq p_{i+1b}$,
$\nexists t_{i,i+1} \in T_{i,i+1} \quad | \quad (p_{i+1a} \in t_{i,i+1}{}^{\bullet}) \wedge (p_{i+1b} \in t_{i,i+1}{}^{\bullet})$

This definition describes how one layer communicate to the above layer. For process communication, more than one object of a layer can belong to the same object of the above layer. MPhyScaS architecture has already defined that, in general, this can only occur if each communication between an object and the object of the layer above is represented separately. This also makes easy to identify the entire path of a task execution.

In Figure 4, we use the Block level to illustrate the level i and the Kernel one to the level $i+1$. Figure 4(a) is an example of a valid part of the Petri net based on Definition 5.1 which is an example of valid relationships between three Blocks and a Kernel. Similarly, Petri net presented in Figure 4(b) violates rule 2 of Definition 5.1. Beside of Petri net in Figure 4(b) does not represent a request from the Kernel (because in this Petri net the Kernel asks to Blocks 2 and 3 using only one request for executing something), the way represented in this Petri net would not permit us to analyze each path that one request goes through.

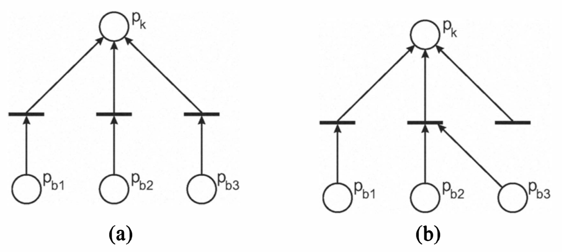


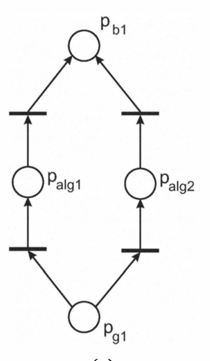
Fig. 4. Graphical representation of a valid and an invalid part of Petri net based on Definition 5.1.

MPhyScaS architecture defines only four layers. Some of these layers are divided into sub-layers:

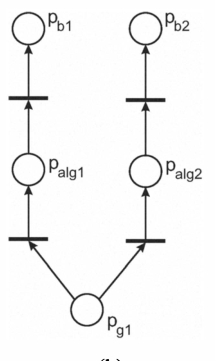
- The Block layer is divided into Block and Algorithm;
- The Group layer is divided into Group, QuantityTask, and GroupTask;
- The Phenomenon layer is divided into Phenomenon and Quantity.

Another restriction shows up when there is communication between two layers and this communication occurs between the first sub-layer of a layer i and the last sub-layer of $i+1$. In this case, more than one objects of the last sub-layer of $i+1$ can communicate to the same object of first sub-layer of i. Following MPhyScaS architecture definition, this can only occur if the objects of last sub-layer of $i+1$ belongs to the same object of first sub-layer of $i+1$.

In Figure 5, we use Group level as the first sub-layer of i, Algorithm level as the last sub-layer of $i+1$, and Block level as the first sub-layer of $i+1$. Figure 5(a) is an example of a valid Petri net that follows the new restriction which is an example of valid relationships between a Group and two Algorithms that belongs to the same Block. Similarly, Petri net presented in Figure 5(b) is an example of this new restriction violation. In this last Petri net, two different Algorithms (Algorithms 1 and 2) belonging to different Blocks (Blocks 1 and 2 respectively) ask for the execution of the same Group (Group 1).



(a)



(b)

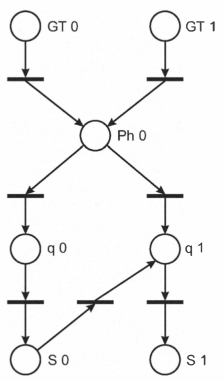
Fig. 5. Graphical representation of a valid and an invalid part of Petri net considering the new restriction.

VI. Model Analysis

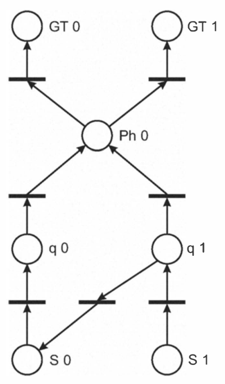
To assist our modeling we use the tool CPNTools [3], which is a mature and well tested tool that supports editing, simulation, and analysis of CPN.

To illustrate the analysis done, Figure 6(a) shows a part of a Petri net that represents the execution of two GroupTasks. GroupTask 0 (GT 0) requires Phenomenon 0 (Ph 0) to execute Quantity 0 (q 0) which assembles the Global State 0 (S 0). In the same Petri net, GroupTask 1 (GT 1) requires Phenomenon 0 (Ph 0) to execute Quantity 1 (q 1) which needs that Global State 0 (S 0) is already assembled to be able to assemble the Global State 1 (S 1).

In order to make it easier the model analysis, we invert the direction of each arc in the Petri net as shown in Figure 6(b). This helps us to identify the entire path which a requirement of a task execution passed through. In this way, the analysis begins in the execution of a task (State level) and goes through the levels following a bottom-up direction until it reaches the top level (Kernel level).



(a)

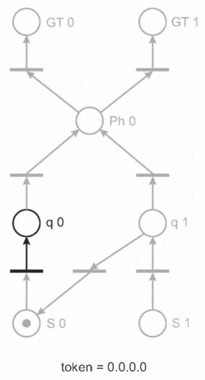


(b)

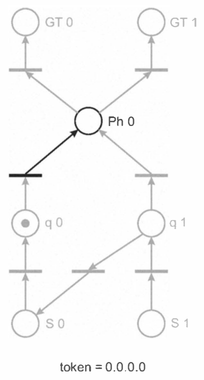
Fig. 6. An example of a Petri net that represents the execution of two GroupTasks and the same Petri net with arcs direction inverted.

The first task to execute is represented by a token with 0.0.0.0 value. This token is in the place that represents the Global State 0 (S 0) because this is the global state assembled by the task 0.0.0.0 (see Figure 7(a)). This marking enables the transition that makes token go to the place that represents the Quantity 0. This is the quantity that assembles Global State 0.

When the token is in the place that represents the Quantity 0 (q 0), as depicted in Figure 7(b), it enables the transition that makes the token go to place that represents the Phenomenon 0. The same occurs when the token is in the place that represents Phenomenon 0 (Ph 0). A transition is enable to fire, moving the token to the place that represents GroupTask 0.



(a)



(b)

Fig. 7. Petri net in the execution of the first task of the simulation.

The analysis continues until Kernel level is reached. When this occurs, the token assumes the value of the next task of the simulation. In the case of the example, the token assumes 1.1.0.1 value. It is placed in the Global State 1 which is the next global state to be assembled in the simulation. Figure 8(a) shows the token in the place that represents Global State 1 (S 1). In this case, the token enables the transition that makes the token go to the place that represents Quantity 1.

When the transition is fired and the token is in the Quantity 1 place, the token enables two transitions to fire. The first transition is the one which makes the token go to place that represents Phenomenon 0. The second transition is the one which makes the token go to place that represents the Global State 0. Figure 8(b) depicts this situation.

The dependencies are identified when a toke enables two transition. In this case, we verify all state space constructed from this moment. The analysis search for objects that belong to levels that execute something. These levels are Kernel, Algorithm, QuantityTask, GroupTask, and Quantity levels. The other levels only forward a requirement, executing nothing. There is a dependence between the objects found of the same level. The dependence occurs so that the object that came from an already executed task must be executed before the other one. In the example, the analysis finds two dependences: Quantity 1 depends on the Quantity 0; and GroupTask 1 depends on the GroupTask 0 for executing.

The restriction of each transition is important because the analysis do not identify dependences if an object depends on

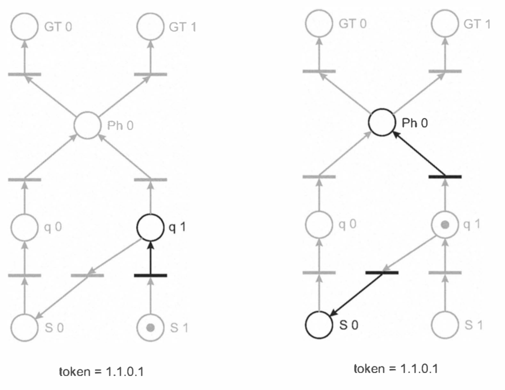


Fig. 8. Petri net in the execution of the second task of the simulation.

another one and this last one depends on another one. In other words, the analysis do not identify dependence between the first object and the third one.

The model analysis is summarized in Pseudo-Code 1. We use the occurence graphs method (also called state spaces or reachability graphs) to realize the analysis.

Pseudo-Code 1: Model analysis pseudo-code.

```
1 foreach node of the state Space do
2     boolean dependence = Verify
      Dependence(node) //Verify if there
      are more than one tokens and if they
      are at the same level
3     if dependence then
4         Write In File(node)
5     end
6 end
```

Line 1 of Pseudo-Code 1 shows that each node of the state space, *i.e.* each reachable system state will be analyzed. In line 2 it is verified if there is any dependence in the node. For this, it is verified if there are more than one token in the net and if the tokens are at the same level of MPhyScaS architecture. If a dependence exists, the node indicating the places that contains the tokens will be written in a file (lines 3 to 5).

VII. Scheduling Parallel Jobs

A. Genetic Algorithms

Genetic Algorithms are inspired by the mechanism of natural selection where stronger individuals are likely the winners in a competing environment. Here, GA uses a direct analogy of such natural evolution. Throughout a genetic evolution, the fitter chromosome has a tendency to yield good quality offspring which means a better solution to any problem [12].

In each cycle of genetic operation, a subsequent generation is created from the chromosomes in the current population (parents). The genes of the parents are mixed and recombined for the production of offspring in the next generation. It is expected that from this process of evolution, the better chromosome will create a large number of offspring, and thus has a higher chance of surviving in the subsequent generation, emulating the survival-of-fittest mechanism in nature [13].

B. An Algorithm Based on GA for Scheduling Parallel Jobs

Analyzing the CPN model, we can automatically generate a DAG structure, whose nodes are processes and edges represents the data dependences between processes. The DAG is then provided as input for the proposed genetic algorithm to find schedules for parallel MPhyScaS applications.

MPhyScaS' tasks are non-preemptive. Moreover, we know in advance the worst-case execution time (WCT) of each task as well as the communication costs involved. Eventually, we assume a platform with fixed number of homogeneous processors. Considering this scenario, we propose to apply a genetic algorithm to generate an offline schedule for the parallel execution of MPhyScaS simulations.

The GA individual is represented by a sequence of processes for the execution. This sequence is divided into blocks that have their size equals to the number of processors. In our scheduling algorithm, the genetic operators affect only the processes. They modify where the processes will be executed, and at what execution time it will occur.

The genetic algorithm proposed herein applies a proposed crossover operator to generate two new individuals into the new generation. It is a two-level crossover consisting of getting genes of the parents and swapping repeated information. The mutation operator guarantees the modification in communication cost values without violating the precedence of processes.

Our scheduling algorithm wants to find a trade off between the execution of processes and the time necessary for these processes to communicate with each other. Besides that, we want to minimize the idle time. In other words, our scheduling algorithm aims to optimize three criteria together by aggregation. The fitness function used is

$$\gamma = \omega_1 \cdot C_{max} + \omega_2 \cdot T + \omega_3 \cdot \sum_{j=1}^{n} WT_j$$

where C_{max} is the makespan function, T represents the total time spent with communication, $\sum_{j=1}^{n} WT_j$ is the sum of idle times for all processors, and ω_1, ω_2, and ω_3 are weights for giving importance to each function.

VIII. Experiments Setup

The experiment explores two parallel architectures composed of fixed number of homogeneous processors. The difference is the number of processors in each architecture: the first one has 3 processors, and the second, has 6 processors.

We compare our algorithm against three scheduling algorithms: List Scheduling [14], Longest Processing Time (LPT) [15], and Shortest Processing Time (SPT) [16]. In list scheduling algorithm we use the MPhyScaS architecture for determining the priority of the processes. Processes in the same level have the same priority.

We calculate the arithmetic mean and standard deviation of each simulated scenario. From each of 24 scenarios (combining three simulators, two parallel architecture, and four scheduling algorithms), we collected data from 30 simulations. This estimation was based on the number of samples required to calculate the mean value for the fitness function assuming a confidence interval of 95%. We used the data of ten simulations, and we found that we need about nine samples.

We use a benchmark composed of three different MPhyScaS' simulators, which are represented through DAGs specifying the processes and their dependences.

For sequential simulation of the problems, the communication cost and the waiting time functions have their values nulled, and the makespan function has its maximum value. The results for sequential execution will also be presented.

In the experiments, we set the parameters to values:

- Makespan (ω_1): 60%
- Communication cost (ω_2): 30%
- Waiting time (ω_3): 10%

These parameters are based on the characteristics of MPhyScaS simulators. One might change them to assign a distinct level of importance for each optimization criteria, generating a different scheduling as result.

We evaluate the proposed GA algorithm using two population sizes: one with 50 indiviuals; and another with 100 individuals, using cross probability equals to 0.9 and the mutation rate equals to 0.1.

IX. RESULTS

In this section, we will present the results for the three different MPhyScaS simulators, which are represented through DAGs specifying the processes and their dependences.

A. The First Simulator

The DAG that represents this simulator has 50 nodes (processes) and 61 edges (dependences). These dependences were found by analyzing the Petri net model that corresponds to this DAG. A maximum of 8 levels of nesting is found in dependences of this graph.

The value obtained using the fitness function for the sequential execution of the first problem is equal to 1446.6. Table I shows the results to list scheduling, LPT and SPT for both parallel architectures (composed by 3 processors and 6 processors). The results found by the proposed genetic algorithm for 500, 1000, and 2000 iterations are presented in Table II. One can note that we also present in both tables the percentage gain obtained based on the sequential execution. For the GA algorithm the percentage gain presented is calculated to the result obtained after 2000 iterations.

The improvement of the proposed algorithm is depicted in Figure 9. Figure 9(a) shows the convergence for the architecture with 3 processors, while the one for the architecture with 6 processors is shown in Figure 9(b).

B. The Second Simulator

The second DAG has 126 nodes (processes) and 223 edges (dependences), which were found by the Petri net model. This graph has 16 levels of nesting as its maximum.

TABLE I
RESULTS FOR LIST SCHEDULING, LPT AND SPT

3 processors			
	List Scheduling	LPT	SPT
Mean	969.9200	1045.5100	1017.1700
S.D.	36.7211	37.2302	40.0182
Gain (%)	32.9517	27.7264	29.6854
6 processors			
	List Scheduling	LPT	SPT
Mean	1066.7400	1104.3200	1132.8300
S.D.	114.8803	120.1449	122.5676
Gain (%)	26.2588	23.6610	21.6902

TABLE II
RESULTS FOR GA

		Fitness	
3 processors			
Iterations		50 individuals	100 individuals
500	Mean	842.2633	833.4733
	S.D.	31.0516	21.0644
1000	Mean	839.9333	829.4833
	S.D.	29.8499	24.1853
2000	Mean	837.4533	828.8533
	S.D.	29.3756	24.0546
Gain (%)		42.1088	42.7033
6 processors			
Iterations		50 individuals	100 individuals
500	Mean	707.2600	697.2100
	S.D.	37.0818	36.6040
1000	Mean	689.2900	686.1900
	S.D.	38.0706	35.5946
2000	Mean	682.0100	680.0800
	S.D.	36.3665	32.2251
Gain (%)		52.8543	52.9877

The value obtained using the fitness function for the sequential execution of the second problem is equal to 3510.0. The list scheduling, LPT and SPT results are presented in Table III for both parallel architecture considered. Table IV shows the results obtained for the same scenarios using the proposed GA algorithm.

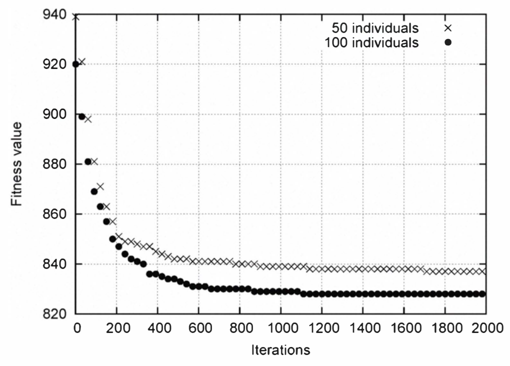
TABLE III
RESULTS FOR LIST SCHEDULING, LPT AND SPT

3 processors			
	List Scheduling	LPT	SPT
Mean	2592.1100	2580.9700	2618.6800
S.D.	157.4361	154.3924	159.9462
Gain (%)	26.1507	26.4681	25.3937
6 processors			
	List Scheduling	LPT	SPT
Mean	2944.8700	2817.8200	2842.7300
S.D.	158.5764	155.0172	160.9108
Gain (%)	16.1006	19.7202	19.0105

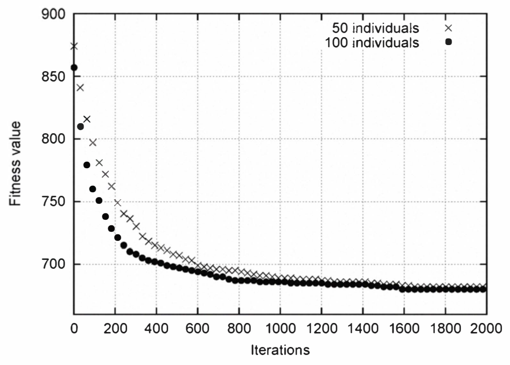
The convergence during all iterations of the latest genetic algorithm results presented can be seen in Figure 10.

C. The Third Simulator

The DAG that represents the third simulator has 150 nodes (processes) and 237 edges (dependences). These dependences were found by analyzing the Petri net model that corresponds to this problem. A maximum of 12 levels of nesting is found in dependences of this graph.

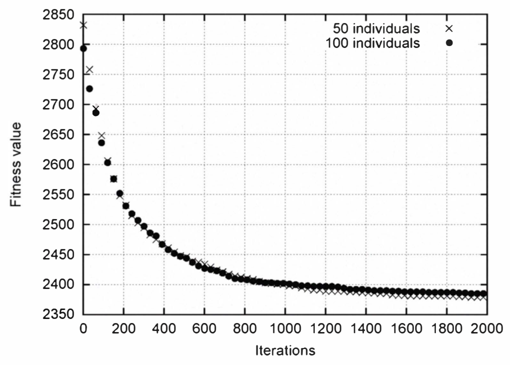


(a)

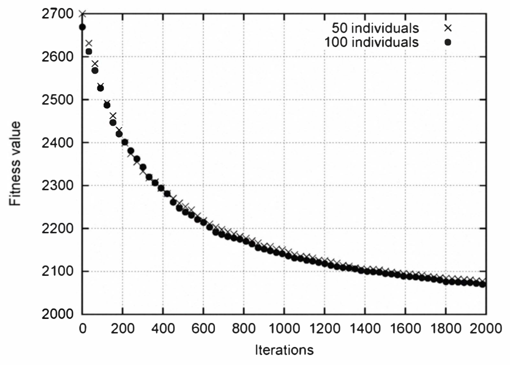


(b)

Fig. 9. The convergence of the proposed genetic algorithm for 50 and 100 individuals: (a) using first architecture; (b) using second architecture.



(a)



(b)

Fig. 10. The convergence of the proposed genetic algorithm for 50 and 100 individuals: (a) using first architecture; (b) using second architecture.

TABLE IV
RESULTS FOR GA

		Fitness	
3 processors			
Iterations		50 individuals	100 individuals
500	Mean	2447.6730	2445.8930
	S.D.	62.3827	61.3071
1000	Mean	2399.9230	2401.9930
	S.D.	59.5612	74.9665
2000	Mean	2379.2830	2385.7330
	S.D.	63.3114	70.7054
Gain (%)		32.2141	32.0304
6 processors			
Iterations		50 individuals	100 individuals
500	Mean	2254.2800	2242.7870
	S.D.	74.0134	62.4255
1000	Mean	2149.6600	2140.3770
	S.D.	66.9043	65.4855
2000	Mean	2077.6200	2068.6470
	S.D.	67.2494	59.3870
Gain (%)		40.8085	41.0642

The value obtained using the fitness function for sequential execution of the third problem is equal to 5680.8. The results to list scheduling, LPT and SPT are presented in Table V considering both parallel architecture. Table VI shows the results also for both architectures considered found by the proposed genetic algorithm for 500, 1000 and 2000 iterations.

Figure 11 depicts the convergence for the presented results. The convergence related to the architecture with 3 processors is depicted in Figure 11(a) and the one related to the architecture with 6 processors, in Figure 11(b).

TABLE V
RESULTS FOR LIST SCHEDULING, LPT AND SPT

3 processors			
	List Scheduling	LPT	SPT
Mean	5464.2300	5356.8000	5495.4900
S.D.	256.9011	255.3976	257.7374
Gain (%)	3.8123	5.7034	3.2620
6 processors			
	List Scheduling	LPT	SPT
Mean	6727.8600	6727.8000	6982.8900
S.D.	259.8337	258.9764	260.4492
Gain (%)	−18.4315	−18.4305	−22.9209

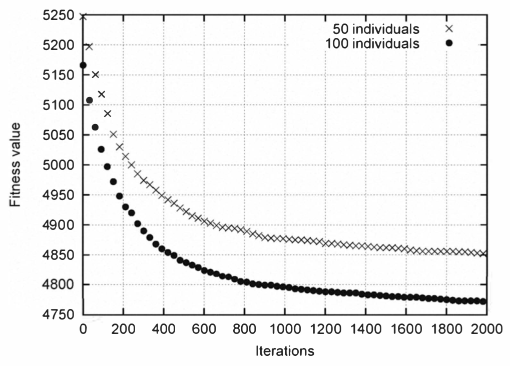
Looking at the results for the second architecture, one can note that only GA algorithm gets some improvement. The other algorithms do not get improvement because of the communication cost and waiting time which spent more time than the saved one.

X. CONCLUSIONS

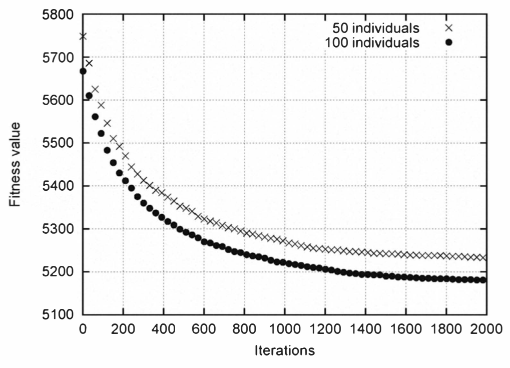
This work explored an important factor related to the parallelization of simulators based on the Finite Element Method (FEM [10]), *i.e.* multi-physics simulators. In this kind of simulators, there is a large set of relations between objects within the architecture. Distinguishing what could be a dependence or a simple relation is an arduous and high complex task. In

TABLE VI
RESULTS FOR GA

		Fitness	
3 processors			
Iterations		50 individuals	100 individuals
500	Mean	4925.2800	4838.9400
	S.D.	101.4804	111.0909
1000	Mean	4877.0000	4796.0000
	S.D.	96.3841	117.8795
2000	Mean	4852.7400	4772.9200
	S.D.	102.7384	111.5300
Gain (%)		14.5765	15.9816
6 processors			
Iterations		50 individuals	100 individuals
500	Mean	5350.2400	5294.6800
	S.D.	147.6926	107.5116
1000	Mean	5269.4200	5221.0000
	S.D.	148.4255	107.9098
2000	Mean	5232.9200	5181.1800
	S.D.	154.3999	106.3545
Gain (%)		7.8841	8.7949



(a)



(b)

Fig. 11. The convergence of the proposed genetic algorithm for 50 and 100 individuals: (a) using first architecture; (b) using second architecture.

this paper, we propose a model based on Coloured Petri Nets which makes the task of classifying dependencies automated.

It was also shown in the experiments that the model is capable in identifying all existing dependencies in the simulators associated to each presented scenario. So that, it cancels the possibility of classification error. Hence, the proposed model achieved success in its main proposal, *i.e.* to identify all dependencies between processes.

We also proposed a scheduling algorithm, based on genetic algorithms, to explore the dependencies in order to define a schedule near the optimal one. This process can be very complex when considering other effects: the architecture where the simulations will run; the communication cost between jobs; and the waiting time of each processor. We evaluate our approach against three well known algorithms and three different multi-physics simulators. The results demonstrated that our approach was able to find excellent schedule for the parallel multi-physics simulators in the benchmark.

As a future work, we will adapt the algorithm to consider parallel platforms with heterogeneous processors. We will also compare our approach to other meta-heuristics.

REFERENCES

[1] F. C. G. Santos, J. M. A. Barbosa, J. M. Bezerra, and E. R. R. J. Brito, "An architecture for the automatic development of high performance multi-physics simulators," *Fourth International Conference on Advanced Computational Methods in Engineering (ACOMEN 2008)*, 2008.

[2] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Monographs in Theoretical Computer Science a Series of Eatcs)*. 2, 1997.

[3] CPNGroup, "Cpntools: Computer tool for coloured petri nets," 2009, url: http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[4] J. Xu, H. Sun, and W. Yang, "Heuristic algorithm for fixed job scheduling problem," in *ICNC '07: Proceedings of the Third International Conference on Natural Computation (ICNC 2007)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 698–701.

[5] Q. Kang, H. He, H. Wang, and C. Jiang, "A novel discrete particle swarm optimization algorithm for job scheduling in grids," in *ICNC '08: Proceedings of the 2008 Fourth International Conference on Natural Computation*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 401–405.

[6] C. S. Chong, A. I. Sivakumar, M. Y. H. Low, and K. L. Gay, "A bee colony optimization algorithm to job shop scheduling," in *WSC '06: Proceedings of the 38th conference on Winter simulation*. Winter Simulation Conference, 2006, pp. 1954–1961.

[7] J.-L. Kim, "Permutation-based elitist genetic algorithm using serial scheme for large-sized resource-constrained project scheduling," in *WSC '07: Proceedings of the 39th conference on Winter simulation*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 2112–2118.

[8] E. Moattar, A. Rahmani, and M. Derakhshi, "Job scheduling in multi processor architecture using genetic algorithm," *Innovations in Information Technology, 2007. Innovations '07. 4th International Conference on*, pp. 248–251, Nov. 2007.

[9] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, "Energy-aware runtime scheduling for embedded-multiprocessor socs," *IEEE Des. Test*, vol. 18, no. 5, pp. 46–58, 2001.

[10] D. L. Logan, *A First Course in the Finite Element Method*, 3rd ed. Pacific Grove, CA, USA: Brooks/Cole Publishing Co., 2002.

[11] F. C. G. Santos, E. R. R. J. Brito, and J. M. A. Barbosa, "Dealing with coupled phenomena in the finite element method," *XXVII Latin American Congress on Computational Methods in Engineering*, 2006.

[12] W. Banzhaf, P. Nordin, R. Keller, and F. Francone, *Genetic Programming - An Introduction*. San Francisco, CA: Morgan Kaufmann, 1998.

[13] K. F. Man, K. S. Tang, and S. Kwong, *Genetic Algorithms: Concepts and Designs with Disk*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[14] B. Simion, C. Leordeanu, F. Pop, and V. Cristea, "A hybrid algorithm for scheduling workflow applications in grid environments (icpdp)," in *OTM Conferences (2)*, 2007, pp. 1331–1348.

[15] K. Altendorfer, B. Kabelka, and W. Stocher, "A new dispatching rule for optimizing machine utilization at a semiconductor test field," *Advanced Semiconductor Manufacturing Conference, 2007. ASMC 2007. IEEE/SEMI*, pp. 188–193, June 2007.

[16] J. Leung, L. Kelly, and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, Inc., 2004.