Scheduling Parallel Jobs for Multiphysics Simulators

Renata M. Carvalho, Ricardo M. F. Lima, Adriano L. I. Oliveira, and Felix C. G. Santos

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 improve the performance of clusters in the processing of MPhyScaS simulations which are composed by a set of dependent tasks by scheduling them. The presented model is based on Genetic Algorithms (GA) to schedule the parallel tasks following MPhyScaS architecture dependence restrictions. The communication between processors must also be considered in this scheduling. Therefore, a trade off must be found between the execution of processes and the time necessary for these processes to communicate with each other.**

I. Introduction

Scheduling is a decision-making process that is used on a regular basis in many systems. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. Scheduling plays an important role in most systems as well as in most information processing environments [1]. For this work, the resources are the processors available for executing, and the tasks are the processes that must be executed.

An effective way to extract the most performance and scalability out of the parallel application is to exploit both task parallelism and data parallelism [2]. With this approach, an application consists of tasks organized in a Directed Acyclic Graph (DAG) in which each edge corresponds to a precedence relation between two tasks, implying possible data communication. The common approach to schedule DAG's is the task parallel paradigm, which assigns one task to one processor. The scheduling consists on the distribution of the DAG nodes among the machine nodes, so that the makespan is minimum.

Early scheduling algorithms [3], [4], [5] do not take communication into account, but due to the increasing gap between computation and communication performance of parallel systems, the consideration of the communication became more important [6].

MPhyScaS (Multi-Physics and Multi-Scale Solver Environment) is an environment dedicated to the automatic development of simulators. It simulates real problems involving a set of different physic phenomena. Each MPhyScaS simulation demands a great amount of time. To improve the performance of MPhyScaS simulation it is necessary to have an approach to parallelize the processes that compose the simulation. This approach should define the distribution of processes, and their relationship across a cluster of PC's. It also should make use of the concept of layers already used in MPhyScaS in order to define a hierarchical parallel structure.

Each object of MPhyScaS architecture represents one non-preemptive process. In MPhyScaS, the computation time of each process and the communication cost between processes due to the size of required data are known. Thus, we use an off line scheduling approach. This approach groups processes that communicate large data with each other in the same processor. This decrease the data volume through the network, decreasing the communication cost of a simulation. In our approach we also consider a finite number of processors.

Evolutionary algorithms have the ability of finding globally competitive optima in large and complex search spaces efficiently [7]. Thus, we based on Genetic Algorithms (GA) technique to search for a schedule that improve the performance of MPhyScaS simulation. The use of GA technique was possible due to the flexibility of GA to fit any problem. Due to the same reason, we could extend the fitness function in order to consider more effects, such as the communication cost and the waiting time of the processors.

This paper is organized as follows. Section II presents some of the related works used to compose the proposed work. Section III details the MPhyScaS definition and its properties. The methodology used in the hole work is presented in Section IV. Section V describes a model definition based on CPN to represent MPhyScaS data which will help to identify dependencies. Section VI proposes an algorithm based on GA that consider some new effects in scheduling process, worried in solving MPhyScaS scheduling problem. Sections VII and VIII shows the experiments and results obtained, respectively. And the conclusions of the proposed work are presented in Section IX.

II. Related Work

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

Xu *et al.* [8] used Simulated Annealing (SA) for fixed job scheduling problem. Their objective was to minimize assignment cost of the sequential network model. In [9], 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.* [10]

Renata Medeiros de Carvalho, Ricardo Massa Ferreira Lima, and Adriano Lorena Incio de Oliveira are with the Center of Informatics, Federal University of Pernambuco, Brazil (email: rwm@cin.ufpe.br, rmfl@cin.ufpe.br, alio@cin.ufpe.br).

Felix Christian Guimaraes Santos is with the Department of Mechanics, Federal University of Pernambuco, Brazil (email: fcgs@demec.ufpe.br).

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

relied on nectar collection of honey bee colonies to create an algorithm for solving scheduling problem. 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. Several studies have been done to solve the scheduling problem using GA. Kim [11] 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.* [12] 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.* [13] 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 [14]). 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 the Finite Element Method can be organized in a layers architecture [15]. In the top layer global iterative loops can be found, corresponding to the overall scenery of the simulation. The second layer contains what is called the solution algorithms. Each solution algorithm 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 (see Figure 1). The Kernel level represents the global scenery level, the level of the solution algorithms is represented by the Block level, the level of solvers is represented by the Group level, and the phenomena level is represented by the 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.

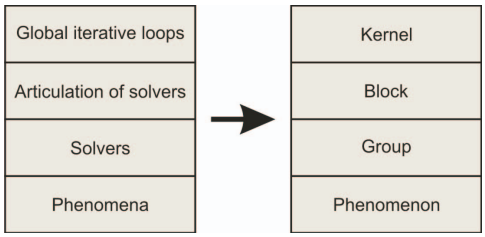


Fig. 1. Computational representation for the layers of the simulator.

Whenever the architecture of a computational system allows a hierarchy of procedures, it may be a good idea to define a hierarchy of processes in such a way that few of them would accumulate some very light management tasks. Benefits for this strategy include [16]: (i) procedures can be hierarchically synchronized, reducing management concerns and increasing correctness; (ii) since locality concerns change along the hierarchy levels, memory management can become more specialized from top to bottom; (iii) the hierarchy allows the encapsulation of services and procedures, making easier the components exchange.

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.

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

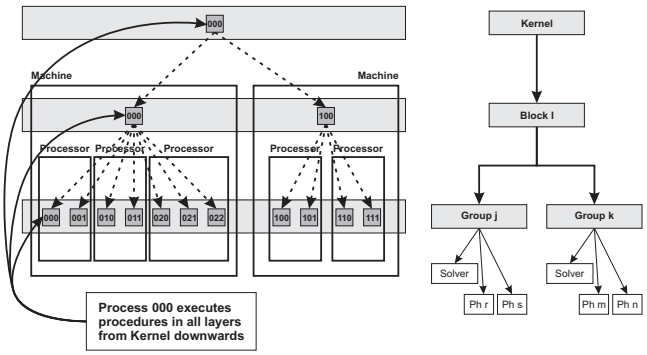


Fig. 2. Layers with procedures executed by ClusterRank in MPhyScaS-P.

MPhyScaS application. In particular, we use a high level Petri net called Coloured Petri Nets (CPN). This kind of Petri net has strongly typed tokes. Thus, depending on its type, a token 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 [17]. The simulation output exposes the data dependencies between MPhyScaS' processes. This information is given as input to the proposed 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.

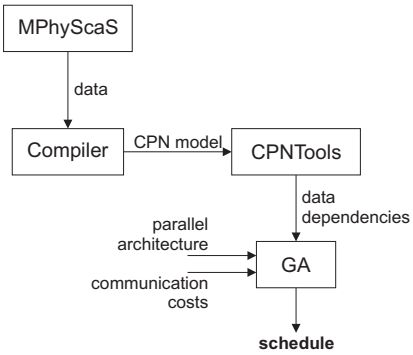


Fig. 3. An overview of the proposed methodology.

V. Coloured Petri Net Model

A coloured Petri net (CPN) [18] is a bipartite directed graph, consisting of two types of vertices: (i) places (drawn as circles), and (ii) transitions (drawn as bars). Places model the states, and transitions the events of the system. In CPN, a transition is able to fire (known as enabled) when: (i) it has one token of the proper type on each of its input arcs, and (ii) the guard (boolean expression) attached to the transition holds. An enabled transition can fire and thus remove tokens from its input places and generate tokens for its output places. This kind of Petri net has strongly typed tokes. Thus, depending on its type, a token might store a complex data type. CPNs allow to model hierarchical structures. The basic idea is to allow the construction of a large model by using a number of small models. These small models are called pages, and are connected to each other by places called ports. Such places can be input or output types.

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 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.

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 GA proposed to find schedules for parallel MPhyScaS applications.

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. If a transition firing enables other transition to fire, we mark the enabled transition as been dependent on the fired transition. This is a causal dependency, indicating that the enabled transition can only execute after the fired transition. When the token reaches the last layer of the architecture, its value is modified to represent the next process of the simulation. This is repeated until the entire CPN model is covered and all dependencies are identified.

To assist our modeling we use the tool CPNTools [17], which is a mature and well tested tool that supports editing, simulation, and analysis of CPN.

VI. Genetic Algorithm for Scheduling Parallel MPhyScaS Processes

This section describes the application of the proposed Genetic Algorithm (GA) to generate schedules to improve the performance of parallel MPhyScaS simulations.

A. Target Application Settings

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.

B. Genetic Algorithms

Genetic algorithms are a class of global optimization algorithm based on the theory of natural selection. In GA, the

individuals of a population of potential solutions to a problem having good genetic characteristics have greater survival and reproduction possibilities. As a result, the individuals less adapted to the environment tend to disappear. Thus, GAs favor the combination of the individuals most apt, i.e., the candidates most promising for the solution of the problem [19]. GAs use a random strategy of parallel search, directed to the search of points of highest fitness, i.e., points in which the function to be minimized or maximized has relatively low or high values, respectively.

C. Individual Representation

Each individual in our genetic algorithm is represented by a sequence of processes. This sequence is divided into blocks whose sizes are equal to the number of processors. Figure 4 is an example of an individual.

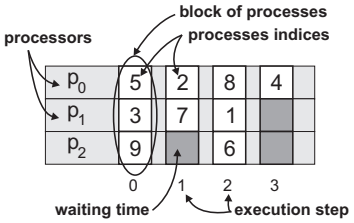


Fig. 4. An example of a scheduling individual.

A block represents the processes that will be executed in parallel, each process in a processor. Each block also has an index that indicates an execution step (see Figure 4). Note that, a block can contain less processes than its size. This is due to the processes dependencies. In the example, processes 1, 4, 6 and 8 depend on processes 2 and 7. This period without process execution is referred to as *wait time*. It is an important factor observed in our algorithm, which tries to minimize the number of waiting times.

D. Selection and Elitism

In our problem, the individuals are selected through a fitness-based process, the proportionate-selection method [7]. This method is used to select both individuals that will be affected by the crossover and the mutation operators. Each of these operators selects a proportion of the population; and the values of the proportions is a parameter given in the experiments.

E. Crossover

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 (i) getting genes of the parents, and (ii) swapping repeated information. In the stage of getting the genes, the processors of the parents are divided into two partitions: even processors and odd processors. Each partition is given to a new offspring so that the first new offspring gets the information of even processors of the first parent and odd processors of the second parent; the second new offspring gets the information of odd processors of the first parent and even processors of the second parent. This process is shown in Figure 5.

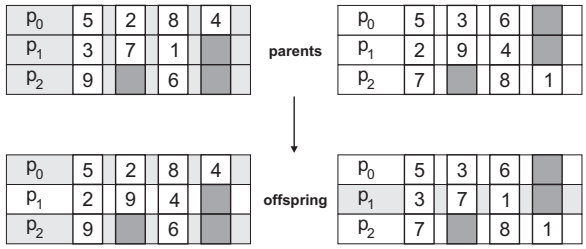


Fig. 5. First step of crossover individuals.

In the swapping repeated information stage, repeated jobs associated to an offspring are searched, and the repeated jobs at the same position in the two new offspring will be considered, as shown in Figure 6. The two new offspring swap these repeated jobs (Figure 6 also shows the resulting of this swapping based on the result obtained in Figure 5). This stage is important to increase the chance of producing a feasible offspring .

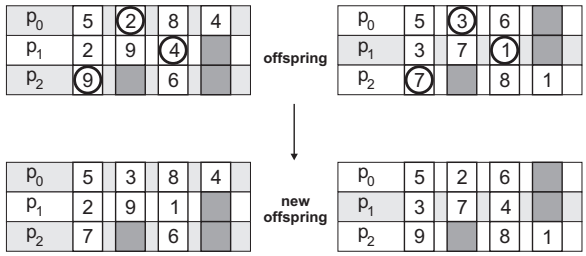


Fig. 6. Second step of crossover individuals.

Although this does not occur every time, when an offspring is not feasible, we discard it in order to have a new generation containing only feasible individuals. An example that shows a crossover that generates unfeasible offspring is depicted in Figure 7, which shows the first step of the crossover. The second step of the crossover is depicted in Figure 8.

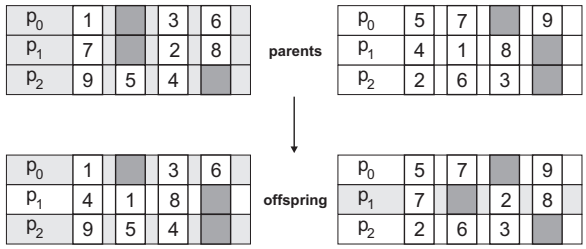


Fig. 7. First step of crossover individuals that generates unfeasible offspring.

One can note that in Figure 8 it is possible to change processes 4 and 7, since they appear in both the first and the second offspring. Processes 1 and 2 also appear in the first and second offspring, yet they cannot be changed because they do not occupy the same position. Thus, the offspring generated after the second step of the crossover are not feasible.

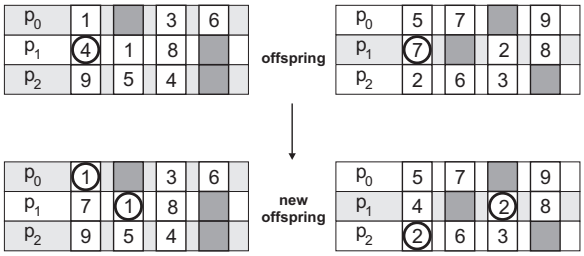


Fig. 8. Second step of crossover individuals that generates unfeasible offspring.

F. Mutation

The uniform mutation was applied as follows: for each individual selected for this operation, an integer random number r is generated and next two random processes are swaped in the rth execution step. These operators guarantee the modification in the values of communication costs without violating the precedence of processes.

Figure 9 shows an example of this process. Here, we suppose that the integer random number generated is $r = 3$, so that the third execution step is the one which will be mutated. Next, we suppose that we selected randomly the numbers 0 and 2 which indicate that the processes that will change are the one in the third execution step of the processor 0 and the one in the third execution step of the processor 2. These processes are marked in Figure 9, which also shows the result after the mutation.

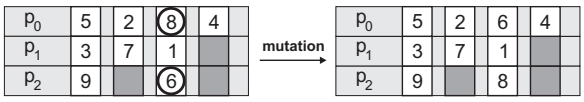


Fig. 9. An example of the proposed mutation.

G. Fitness Function

We defined metrics to evaluate the scheduling algorithm proposed. Such metrics simultaneously evaluate three optimization criteria: i) the total execution time (makespan); ii) the total idle time of each processor (waiting time); iii) the time spent with communication between processes.

The first criterion to be optimized is makespan (total finishing time), which means the maximum execution time of the last job. The makespan function, represented by $C_{max} = max\{C_i | i = 1, ..., n\}$, where C_i is the finishing execution time of the job i. The second criterion is the communication cost. Here, we represented this function by T, where $T = \sum t_{ij}, \forall i, j$, assuming that i, j are two processes that need to communicate and that t_{ij} is the time spent in the communication. The third criterion is the waiting time, which corresponds to the total idle time of the processors. The waiting time function is represented by $\sum_{j=1}^{n} WT_j$, which means the sum of the waiting times of each processor j.

Although in the MPhyScaS problem the optimization criterion consists of the minimization of these three functions, we put these criteria together in the following function

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

where ω_1, ω_2 and ω_3 are weights for giving different relevance to each function.

VII. Experiment Setup

This section describes the scenarios explored in the experiment conducted to evaluate our scheduling algorithm. We use a benchmark composed of three different MPhyScaS' simulators, which are represented through DAGs specifying the processes and their dependences.

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 [20], Longest Processing Time (LPT) [21], and Shortest Processing Time (SPT) [4].

Combining the three simulators, the two parallel architecture, and the four scheduling algorithms give us 24 execution scenarios. We used the data of ten simulations to estimate the number of samples required, we found that we need about nine samples to calculate the mean value for the fitness function, assuming a confidence interval of 95%. Based on this estimation, we collected data from 30 simulations of each scenario. We calculate the arithmetic mean and the standard deviation of each scenario.

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. The Kernel level has the highest priority and the Quantity level (a sublevel of the Phenomenon level) has the lowest priority.

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.

VIII. 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.

TABLE I
RESULTS FOR LIST SCHEDULING, LPT AND SPT

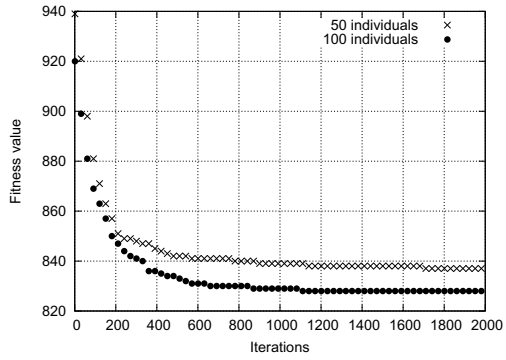
3 processors			
	List Scheduling	LPT	SPT
Mean	969.9200	1045.5100	1017.1700
S.D.	36.7211	37.2302	40.0182
Gain (%)	32.9517	27.7264	29.6854
6 processors			
	List Scheduling	LPT	SPT
Mean	1066.7400	1104.3200	1132.8300
S.D.	114.8803	120.1449	122.5676
Gain (%)	26.2588	23.6610	21.6902

TABLE II
RESULTS FOR GA

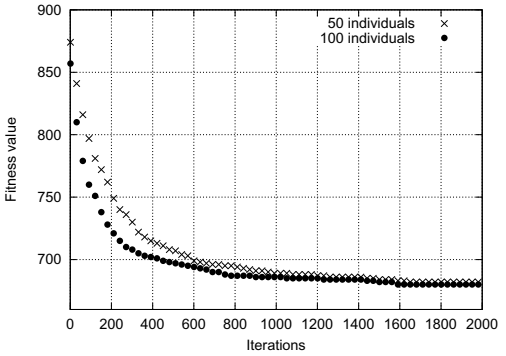
		Fitness	
3 processors			
Iterations		50 individuals	100 individuals
500	Mean	842.2633	833.4733
	S.D.	31.0516	21.0644
1000	Mean	839.9333	829.4833
	S.D.	29.8499	24.1853
2000	Mean	837.4533	828.8533
	S.D.	29.3756	24.0546
Gain (%)		42.1088	42.7033
6 processors			
Iterations		50 individuals	100 individuals
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 improvement of the proposed algorithm is depicted in Figure 10. Figure 10(a) shows the convergence for the architecture with 3 processors, while the one for the architecture with 6 processors is shown in Figure 10(b).

Considering the first architecture, at the 500th iteration the 100-individual population had already presented better schedules than the 50-individual population at the end of 2000 iterations. Note that in both situation the 100-individual population converged faster than the 50-individual population.



(a)



(b)

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

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.

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 11.

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

One can note that this second problem is more difficult to solve than the previous one so that both population size behaved almost in the same way. For this problem, the fact of having more individuals did not contribute to the population with 100 individuals due to the complexity of the problem.

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.

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

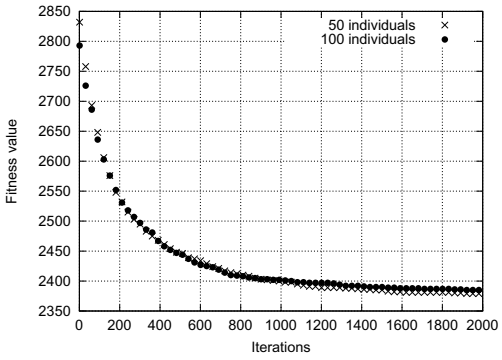
TABLE V

RESULTS FOR LIST SCHEDULING, LPT AND SPT

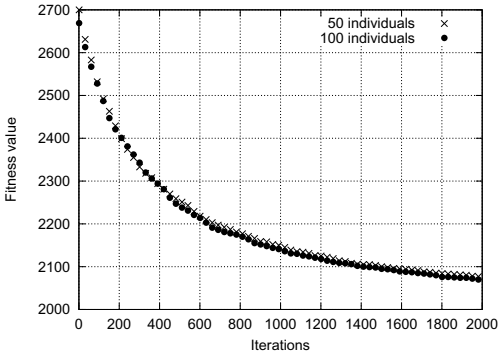
3 processors			
	List Scheduling	LPT	SPT
Mean	5464.2300	5356.8000	5495.4900
S.D.	256.9011	255.3976	257.7374
Gain (%)	3.8123	5.7034	3.2620
6 processors			
	List Scheduling	LPT	SPT
Mean	6727.8600	6727.8000	6982.8900
S.D.	259.8337	258.9764	260.4492
Gain (%)	−18.4315	−18.4305	−22.9209

Figure 12 depicts the convergence for the presented results. The convergence related to the architecture with 3 processors is depicted in Figure 12(a) and the one related to the architecture with 6 processors, in Figure 12(b).

Looking at the results for the second architecture, one can note that only GA algorithm gets some improvement.



(a)



(b)

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

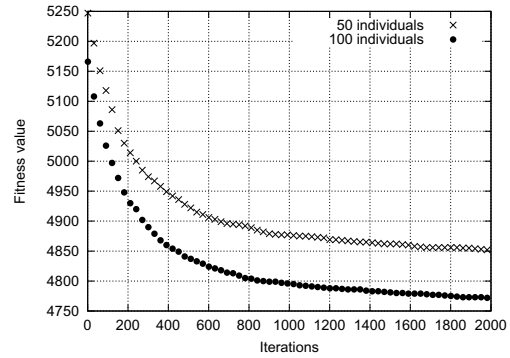
TABLE VI

RESULTS FOR GA

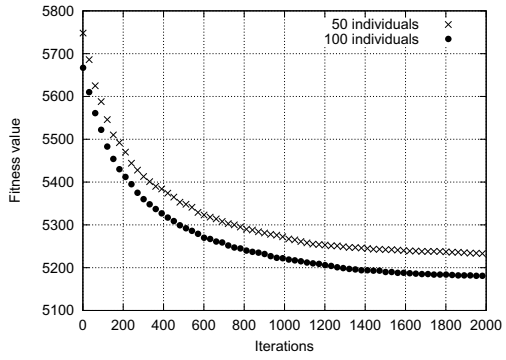
		Fitness	
3 processors			
Iterations		50 individuals	100 individuals
500	Mean	4925.2800	4838.9400
	S.D.	101.4804	111.0909
1000	Mean	4877.0000	4796.0000
	S.D.	96.3841	117.8795
2000	Mean	4852.7400	4772.9200
	S.D.	102.7384	111.5300
Gain (%)		14.5765	15.9816
6 processors			
Iterations		50 individuals	100 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

The other algorithms do not get improvement because of the communication cost and waiting time which spent more time than the saved one.

Using the data of 30 samples of each simulated scenario, we found that only 0.8% of the generated individuals were unfeasible, assuming a confidence interval of 95%.



(a)



(b)

Fig. 12. The convergence of the proposed genetic algorithm for 50 and 100 individuals: (a) using first architecture; (b) using second architecture.

IX. Conclusions

This work explored an important factor related to the parallelization of simulators based on the Finite Element Method (FEM [14]), i.e. multi-physics simulators. We proposed a scheduling algorithm, based on genetic algorithms, to explore the job dependencies in order to define a schedule near to the optimal one. The existent algorithms usually consider only the makespan (total completion time) function to obtain the schedule. 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. Considering other effects leads us to create a complex environment composed of great quantity of variables. This makes the simple scheduling algorithms obsolete. The algorithm proposed in this paper considers three optimization criteria: i) the total execution time (makespan); ii) the total idle time of each processor (waiting time); iii) the time spent with communication between processes. It also takes into account the architecture where the simulations will be run.

We evaluated our approach against three well known algorithms and three multi-physics simulators. The results demonstrated that our approach was able to find excellent schedule for the parallel 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] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008.

[2] K. Aida and H. Casanova, "Scheduling mixed-parallel applications with advance reservations," in *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. New York, NY, USA: ACM, 2008, pp. 65–74.

[3] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook on Scheduling: Models and Methods for Advanced Planning (International Handbooks on Information Systems)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

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

[5] A. W. J. Kolen, A. H. G. R. Kan, C. P. M. van Hoesel, and A. P. M. Wagelmans, "Sensitivity analysis of list scheduling heuristics," *Discrete Appl. Math.*, vol. 55, no. 2, pp. 145–162, 1994.

[6] O. Sinnen and L. Sousa, "Comparison of contention aware list scheduling heuristics for cluster computing," in *ICPPW '01: Proceedings of the 2001 International Conference on Parallel Processing Workshops*. Washington, DC, USA: IEEE Computer Society, 2001, p. 382.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

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

[18] E. Roubtsova, "Property specification for coloured petri nets," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 3, Oct. 2004, pp. 2617–2622 vol.3.

[19] A. P. Engelbrecht, *Computational Intelligence: An Introduction*. NJ: Wiley Publishing, jan 2007.

[20] 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.

[21] 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.