

A Cellular Genetic Algorithm For Scheduling Applications And Energy-Aware Communication Optimization

Mateusz Guzek
University of Luxembourg,
L-1359 Luxembourg-Kirchberg
mateusz.guzek@uni.lu

Bernabé Dorronsoro
University of Luxembourg,
L-1359 Luxembourg-Kirchberg
bernabe.dorronsoro@uni.lu

Johnatan E. Pecero
University of Luxembourg,
L-1359 Luxembourg-Kirchberg
johnatan.pecero@uni.lu

Pascal Bouvry
University of Luxembourg,
L-1359 Luxembourg-Kirchberg
pascal.bouvry@uni.lu

Samee U. Khan
Department of Electrical and Computer Engineering,
North Dakota State University,
Fargo, ND 58108
samee.khan@ndsu.edu

ABSTRACT

In modern parallel and distributed systems, inter-processor communications are a crucial factor of performance. The time for exchanging data is usually larger than that for computing elementary operations. Consequently, these communications slow down the execution of the application scheduled on the computing platform. Accounting for these communications is essential for attaining efficient hardware and software utilization. Moreover, energy dissipation due to the transfer of data between processing elements has become a major concern. Therefore, in this paper we develop an energy-aware static algorithm, which intrinsically optimizes the energy consumption due to the transfer of data in a distributed system. This is achieved by properly allocating and scheduling the tasks that constitute the applications on the processing elements, minimizing inter-processor communications. The proposed algorithm is a new Cellular Genetic Algorithm based on task clustering techniques. That is, the genetic operators work considering groups of tasks instead of applying them directly on the tasks. Simulation results showed that this algorithm is very compelling in terms of application completion time, inter-processor communication and energy communication dissipation.

KEYWORDS: Optimization, meta-heuristics, distributed systems, scheduling, energy-aware.

1. INTRODUCTION

New parallel and distributed systems are based on inter-connections of a large number of processing elements, and they are characterized by many new features that should be taken into account for optimizing their performance. The efficient execution of the applications is much more difficult because of the new characteristics of these systems. The difficulty is growing with the complexity of the interactions between more and more architectural constraints and the diversity of the applications. The design of efficient scheduling algorithms has been reconsidered under the influences of new parameters of such platforms. For most of the existing parallel and distributed systems *inter-processor communications* are a critical factor of performance. Generally, the time for exchanging data is usually larger than that for computing elementary operations. Consequently, these communications slow down the execution of the application scheduled on the computing system, and this is even more important for parallel and distributed systems such as cluster and grid computing. Accounting for these communications is essential for attaining efficient hardware and software utilization. Furthermore, with the introduction of a new generation of business based grid computing (i.e. Cloud computing) dealing with communications during the scheduling is crucial not only in terms of system efficiency but also in terms of monetary cost-benefits, since communications in this computing system

is particularly expensive [1].

Due to the importance of the inter-processor communications on parallel and distributed systems, a large range of scheduling heuristic methods has been developed. Most efforts have focused on two issues, the minimization of application completion time (i.e. throughput maximization) and the time complexity. However, recent research has focused on developing energy-aware scheduling algorithms. Various techniques including dynamic voltage scaling, dynamic power management, or resource hibernation have been investigated [2, 3, 4, 5]. Previous efforts addressing the energy dissipation optimization mainly concentrated on the energy consumption of processing elements. The reason is perhaps that the processor is the main consumer of energy, and it also offers the most flexible options for energy management. Processor energy consumption has been reduced dramatically and it is now a smaller percentage of the overall system energy. However, in parallel and distributed systems with interacting components, communication interfaces additionally contributes to the overall energy consumption and are now consuming high share of the system energy. Consequently, energy dissipation due to the transfer of data between processing elements has become a major interest raising various monetary, environmental, and system performance concern [6].

In this paper, we provide a new energy-aware communication static scheduling algorithm. It intrinsically optimizes the energy communication dissipation by properly allocating and scheduling precedence-constrained applications on the processing elements; reducing application completion time and inter-processor communications at the same time. Because of inter-processor communications costs, an important step in scheduling is task clustering. It is the process of aggregating several basic tasks that will be executed sequentially on each processor in order to obtain tasks with a larger size. By clustering the tasks, we are able to reduce the unbalance between communication and computation times so that the total execution time or *makespan* is minimized.

The main contribution is to develop an effective *Cellular Genetic Algorithm* (we call it EACS, standing for Energy-Aware Communications Scheduler) for scheduling precedence-constrained applications with communications through task clustering techniques. That is, we have designed the genetic operators (e.g. recombination, mutation) considering groups of tasks instead of applying them directly on the tasks. The rationale is that in task clustering the clusters are the meaningful building blocks, i.e. the smallest piece of a solution, which can convey information on the expected quality of the solution they are part

of. A salient feature of the proposed algorithm is that it takes advantage of the good exploration/exploitation balance performed by the cellular genetic algorithm on the search space, improved with the effectiveness of task clustering for reducing inter-processor communication. Simulation results showed that this algorithm is very compelling in terms of application completion time, inter-processor communication and energy communication dissipation.

The remainder of the paper is organized as follows: Section 2 describes the models and states the problem. Section 3 briefly discusses Cellular GAs. The proposed solution is described in Section 4. Experimental results are given in Section 5. Section 6 concludes the paper.

2. PROBLEM MODELIZATION

2.1. System Model

The target execution support that we consider in this work is a cluster computing system composed by a collection of a set of m homogeneous processing elements (i.e. simple PC machines). For sake of simplicity and without any loss of generality, we assume that all the processing elements are fully connected through network resource. That is, every processor has a direct link to any other processor. The studied network is considered to be *nonblocking*; that is, the communication of two processors does not block the connection of any other two processors in the network [7]. Additionally, we assume that the links have the same data transfer speed Bw —the transmission rate of a link connecting two processors without contention measured by Mega bits per second (Mbps)—, and the same energy consumption rate ce measured in terms of Joule/Mb. We also consider that a message can be transmitted from one processor to another while a task is being executed on the receiver processor. This is possible in many systems. Finally, communications between tasks executed onto the same processor are neglected. This computational model corresponds to the classical delay model [8].

2.2. Application Model

As it is common, we represent the application or parallel program by a *Directed Acyclic Graph* (DAG), also called the macro-dataflow graph. It is represented by a precedence task graph $G = (V, E)$, where V is the set of n nodes corresponding to the tasks t_j of the parallel program (see Figure 2a). There is an associated value p_j to every task t_j , representing its processing time. E is the set of directed edges between the tasks that maintain a partial order among them. Let \prec be the partial order of the tasks in G , the partial order $t_i \prec t_j$ models precedence constraints. That is, if there is an edge $e_{ij} \in E$ then task t_j cannot start

its execution before task t_i completes. Hence, the results of task t_i must be available before task t_j starts its execution. Each edge e_{ij} has an associated parameter c_{ij} , measured in Mb , which is the volume of transferred data, as well as a data transfer time or inter-processor communication cost, defined as: $Tc_{ij} = \frac{c_{ij}}{Bw}$. We can define a clustering of the tasks in graph G for this application model as:

Definition 1 (clustering) A clustering $R = \{V_i, \prec_i\}_i$ of G is a mapping of the graph onto groups of tasks associated to a total linear order extension of the original partial order \prec .

2.3. Energy Model

The energy communication consumption in this work is computed based on a model derived from findings reported in the literature [9]. The communication energy model adopted in this work is defined as:

$$E = ce \sum_{i=1}^n \sum_{j=1, j \neq i}^n c_{ij}, \quad (1)$$

where the term $\sum_{i=1}^n \sum_{j=1, j \neq i}^n c_{ij}$ is the total of the active and effective communication or volume of data transmission for the current schedule.

2.4. Scheduling Model

The considered scheduling objective is to find a clustering R of G with both a minimum execution time (i.e. makespan) and minimum energy requirements for the inter-processor communications. Using the standard 3-fields notation of scheduling problems, it is denoted by $P_m | prec, p_i, c_{i,j} | C_{max}$. This scheduling problem is NP-hard [10]. An optimal schedule is a trade-off between high parallelism and low inter-processor communication. On the one hand, tasks should be distributed among the processors in order to balance the workload. On the other hand, the more nodes are distributed, the more interprocessor communications, which are expensive, are performed. A natural idea is therefore to determine first which nodes should always be executed on the same processor before the actual scheduling. Task clustering is a technique that follows this idea [10, 11], and it is often used for scheduling tasks on an unlimited number of processors [12, 13, 14]. Nevertheless, it is proposed as an initial step in scheduling when the number of processors is bounded [7]. The basic rule of task clustering is that every task in a cluster must be executed in the same processor. Thus, the communication cost between tasks in a cluster can be ignored.

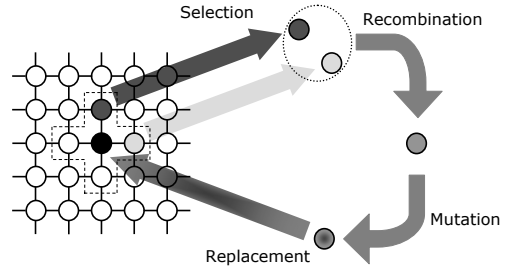


Figure 1. In Cellular GAs, Individuals Are Only Allowed To Interact With Their Neighbors

3. CELLULAR GAs

Cellular GAs [15, 16, 17], are structured population algorithms with a high explorative capacity. The individuals composing their population are (usually) arranged in a two dimensional toroidal mesh. This mesh is also called grid. Only neighboring individuals (i.e. the closest ones measured in Manhattan distance) are allowed to interact during the breeding loop (see Figure 1). This way, we introduce a form of isolation in the population that depends on the distance between individuals. Hence, the genetic information of a given individual spreads slowly through the population (since neighborhoods overlap). The genetic information of an individual will need a high number of generations to reach distant individuals, thus avoiding premature convergence of the population. By structuring the population in this way, we achieve a good exploration/exploitation trade-off on the search space. This improves the capacity of the algorithm to solve complex problems [15, 18].

A canonical CGA follows the pseudo-code of Algorithm 1. In this basic CGA, the population is usually structured in a regular grid of d dimensions ($d = 1, 2, 3$), with a neighborhood defined on it. Each individual in the grid is iteratively evolved (line 2). A generation is the evolution of all individuals of the population. Individuals may only interact with individuals belonging to their neighborhood (line 3),

Algorithm 1 Pseudo-code For a Canonical CGA (asynchronous)

```

1: while ! StopCondition() do
2:   for all ind in population do
3:     neigh ← get_neighborhood(ind);
4:     parents ← select(neigh);
5:     offspring ← recombine(p_comb, parents);
6:     mutate(p_mut, offspring);
7:     evaluate(offspring);
8:     replace(ind, offspring);
9:   end for
10: end while

```

so parents are chosen among the neighbors (line 4) with a given criterion. Recombination and mutation operators are applied to the individuals in lines 5 and 6, with probabilities p_{comb} and p_{mut} , respectively. Afterwards, the algorithm computes the fitness value of the new offspring individual (or individuals) (line 7), and replaces the current individual (or individuals) in the population (line 8), according to a given replacement policy. This loop is repeated until a termination condition is met (line 1), for example: the total elapsed processing time or a number of generations.

The CGA described by Algorithm 1 is called asynchronous, because the population is updated with next generation individuals immediately after their creation. These new individuals can interact with those belonging to their parent's generation. Alternatively, we can place all the offspring individuals into an auxiliary population, and then replace all the individuals of the population, with those from the auxiliary population, at once. This last version is referred to as the synchronous CGA model. As it was studied in [15, 19], the asynchronous CGAs converge the population faster than the synchronous CGAs.

4. THE CELLULAR GA-BASED TASK CLUSTERING

In this section we describe the proposed solution, called EACS.

4.1. Proposed Solution

Commonly touted for their ability to solve nonlinear and combinatorial problems, CGAs typically perform well on problems in which the objective and/or search space combine both discrete and continuous variables. They are also often noted for searching large, multi-modal spaces effectively since they operate on a population of solutions rather than on one individual and use no gradient or other problem-specific information. A great advantage of CGAs is that they make no assumptions about the problem space they are searching and that they are continuously maintaining a population set of search points from the solution space which are explored.

In traditional schedule optimization methods (e.g. list scheduling, task clustering, and various types of mathematical programming algorithms), the search algorithm is tightly coupled to the schedule generator. These methods operate in the problem space; they require information about the schedule in order to search for better schedules. CGAs operate in the representation space. They care only about the structure of a solution, not about what that structure represents. The performance of each solution is the

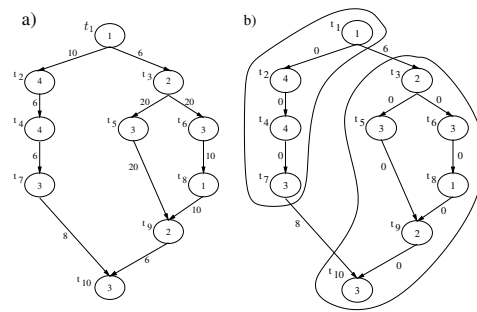


Figure 2. a) Example Of A DAG. b) A Possible Clustering Of The DAG

only information CGA needs to guide its search. For example, a typical heuristic scheduler requires information about the resources and constraints in order to decide which task should be scheduled next in order to build the schedule. The EACS algorithm, on the other hand, only needs to know how good a schedule is and how to combine two schedules to form another one.

4.2. Algorithm Design

In the proposed EACS algorithm, the population is structured in a bidimensional grid of individuals and the neighborhood is defined as a L5 neighborhood shape [15]. As we already mentioned, the aim of EACS is to gather the tasks on clusters (one cluster for one physical processor) and schedule them efficiently onto m processors. The number of clusters generated by EACS is less than or equal to the number of available physical processors in the computing system.

4.2.1. String Representation

Solutions are represented as a string of integers of length n , where n is the number of tasks in the given DAG. The value in the i -th position in the string represents the cluster to which task i is scheduled. The maximum number of clusters is bounded to the amount of processors in the system. Due to the specification of our problem, namely large communication delays, it is important to avoid too fine-grained scheduling. We have enriched this representation with a group part (T_i), encoding the clusters on one gene for one cluster basis. The group part is a string of integers, a random permutation of clusters used in the scheduling. The length of the group part representation is variable. The genetic operators are applied to the group part of the chromosomes, the standard task part of the chromosomes identify which task actually form which cluster.

Figure 3 depicts a string representation for the clustering of Figure 2 b). Two clusters compose the clustering. This representation is enriched by the cluster part. For this ex-

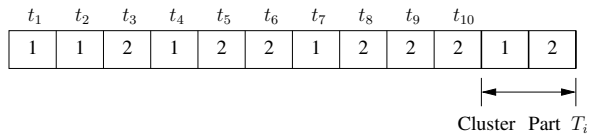


Figure 3. A String Representation Of The Clustering Depicted In Figure 2 b)

ample, cluster 1 is composed by tasks $\{t_1, t_2, t_4, t_7\}$ and cluster 2 is composed by tasks $\{t_3, t_5, t_6, t_8, t_9, t_{10}\}$.

4.2.2. Redundancy

In this type of clustering encoding (i.e. group-number encoding), there are possible multiple representations of the same solutions: as all processors in the system are identical, then if all tasks are rescheduled from processor j to processor k , and all tasks from processor k are rescheduled to processor j , the resulting clustering will be equivalent. However, the difference in labeling hides this similarity, so that the redundancy prevents EACS from finding similarities among the solutions and increases the difficulty of finding good “building blocks”. To avoid that, we adopt the *reordering* procedure used in [20].

4.2.3. Initial Population and Fitness Function

The first population of solutions is initialized randomly. The fitness of each individual S_i is a measurement of performance of the design variables as defined by the objective function and the constraints of the problem. First, makespan SL_{S_i} is calculated for each individual of the population. Hence, the fitness function is Eq. (2).

$$f_{S_i} = \frac{1}{SL_{S_i}} . \quad (2)$$

To obtain the makespan for a particular clustering, each task is scheduled in decreasing order priority. That is, to schedule a task into the first slot available after the specified earliest start time on the assigned processor. The earliest start time $est(t_i)$ for task t_i is calculated using Eq. (3):

$$est(t_i) = MAX[\sigma(t_i) + p_i + Tc_{ij}, (i, j) \in E] , \quad (3)$$

where: $\sigma(t_i)$ is the start execution time of task t_i . Let us recall that $Tc_{ij} = 0$ if tasks t_i and t_j are placed on the same processor. Once all tasks have been scheduled, the algorithm uses Eq. (4) to determine the makespan of the scheduled DAG.

$$SL_{S_i} \equiv MAX[\sigma(t_i) + p_i] \quad (4)$$

To reduce the computation required, it only checks the sink tasks, since by definition they should finish after any of their predecessors.

4.2.4. Stopping Condition and Selection

EACS stops the execution when it reaches the maximum number of iterations. The chosen selection operator is tournament best of two.

4.2.5. Recombination

The most important operator for finding new clusterings is recombination or crossover. It is based on the grouping genetic algorithm crossover operator [21]. This operator uses the group part of the parents to change the structure of the clustering. When recombination occurs, it randomly chooses one crossover point in each group part of both parent solutions. Then, considering creation of the first new solution, it copies the assignments of tasks from the clusters of the second parent whose identifiers are on the right side of crossover point. The rest of the tasks assignments is copied from the first parent. To create the other new solution, analogical procedure takes place, with switched roles of parents. Because of described behavior, the resulting solutions may have fewer clusters than the initial ones.

4.2.6. Mutation

First, an individual is randomly chosen. Next, the mutation operator is applied to the selected chromosome with a certain probability p_{mut} . The mutation operator randomly changes one task in the scheduling to another different cluster. The new cluster is chosen from the range of all possible clusters. It means that when not all clusters are used, mutation of one task may produce a new cluster. Therefore, it has opposite influence on the number of used clusters than the recombination operator.

5. EXPERIMENTAL RESULTS

In this section, we compare the proposed solution with related approaches by simulations. The performance comparison is made in terms of solution quality of the makespan and the energy dissipated during the scheduling due to the inter-processor communication. We compare the EACS algorithm with a GA based on clustering called GAC developed in [20]. GAC follows the principle of the Simple Genetic Algorithm [22]. It takes as input the number of clusters to be generated (i.e. the number of processors) and the schedule is obtained directly. A chromosome is represented as an array of integers of length equal to the number of tasks. Each entry in the array corresponds to the cluster for a task. The initial population is randomly created. Standard single point crossover is used to generate new individuals. Mutation works by changing the cluster a task belongs to any other cluster different than the one where it was originally associated. Selection uses a proportionate selection scheme.

The EACS algorithm used the following parameters. A bi-dimensional grid of 100 individuals (10×10) has been chosen for structuring the population. As we already mentioned, the neighborhood used is composed of L5 individuals or NEWS neighborhood (the considered individuals plus those located at its North, East, West, South). The number of generations is 200, recombination probability of 0.9, mutation probability of 0.001. The same parameters were adopted by GAC for the generation limit, population size, recombination and mutation probabilities.

We have selected for our simulations task graphs from the *Standard Task Graph Set* (STG) [23]. The STG Set provides task graphs modeled from actual application programs. *SPEC fpppp*, *sparse matrix solver*, and *robot control application* were adopted as targeted benchmark. Table 1 describes the characteristics for these applications. For each graph, we have varied the *communication to computation cost ratio* (CCR), which is the average of communication cost to the average computation times of tasks. We have generated five CCRs (0.5, 1, 2, 5, 10) for each graph.

Table 1. Employed Benchmarks From STG And Their Main Characteristics

Application	# of Tasks	# of Edges
SPEC fpppp	334	1196
Sparse matrix solver	96	128
Robot control	88	131

The algorithms have been compared in terms of both the makespan of the solution and the energy consumed in the inter-processor communications. We show in figures 4 and 5 boxplots with the results we obtained with EACS and GAC with $m = 16$ and 32 available processors, respectively. The line in the middle of the boxes represents the median of the results, and the limits of the boxes are the lower and upper quartiles. The whiskers are the result at position 1.5 multiplied by the positions of the lower and upper quartiles. Crosses represent the values out of the whiskers. If the notches of two boxes are not overlapped then we can assume that there are statistically significant differences between the algorithms with 95% confidence. We can take from figures 4 and 5 the same conclusions in the comparison of EACS and GAC, both in terms of makespan and energy consumption. Our algorithm outperforms GAC in all the cases in terms of makespan (with the only exception of sparse problem with 16 processors) and energy consumed, with significant differences in most cases. Additionally, we can see that, in general, the higher the CCR is, the larger the differences between EACS and GAC are. This means that EACS is performing better when the communication times become more important in the

problem. Indeed, we have experimentally checked that the difference between EACS and GAC is even larger for CCR = 10.

The fact that AECS outperforms GAC could be expected when comparing them in terms of the energy consumption in the communications, since it is precisely designed to minimize it. However, it stands out that by reducing the communications the algorithm is able to also lower the makespan compared to GAC, which was designed to look for efficient makespan solutions.

If we compare the results obtained for the problem instances with 16 and 32 processors, we obtained that the two algorithms improved the makespan of solutions with statistical significance after duplicating the resources, as it could be expected. In terms of energy, GAC is providing worse solutions (with statistical confidence in most cases) when increasing the available resources, therefore the use of more resources induces a higher inter-processor communication degree. However, this is not true for EACS. Indeed, the energy consumed in the communications is lower for the 32 processors instances, being these differences significant for the fpppp and sparse problems. We consider this is an outstanding feature of our algorithm, since it is able to reduce the inter-processor communications even after doubling the number of processors used. This result highlights the effectiveness of the proposed representation and genetic operators based on clusters of tasks.

6. CONCLUSION

In this paper, we have proposed a new genetic algorithm for scheduling applications on distributed memory parallel architectures. We focused on reducing the energy consumed during the inter-processor communication in applications with large communications. For that, we used in our genetic algorithm a solution representation and variation operators that are promoting the organization of tasks into clusters, reducing this way the inter-processor communications.

The new algorithm, called EACS, outperformed the GAC algorithm (selected from the literature) both in terms of makespan and energy consumption. Additionally, we found that the inter-processor communications energy consumption of the solutions obtained by EACS is lower for the problems with 32 processors than in the case of 16, with statistical significance in most cases. On the contrary, the solutions reported by GAC were in all cases using more energy in the case of the 32 processors problems.

In future work, we plan to study the behavior of the algo-

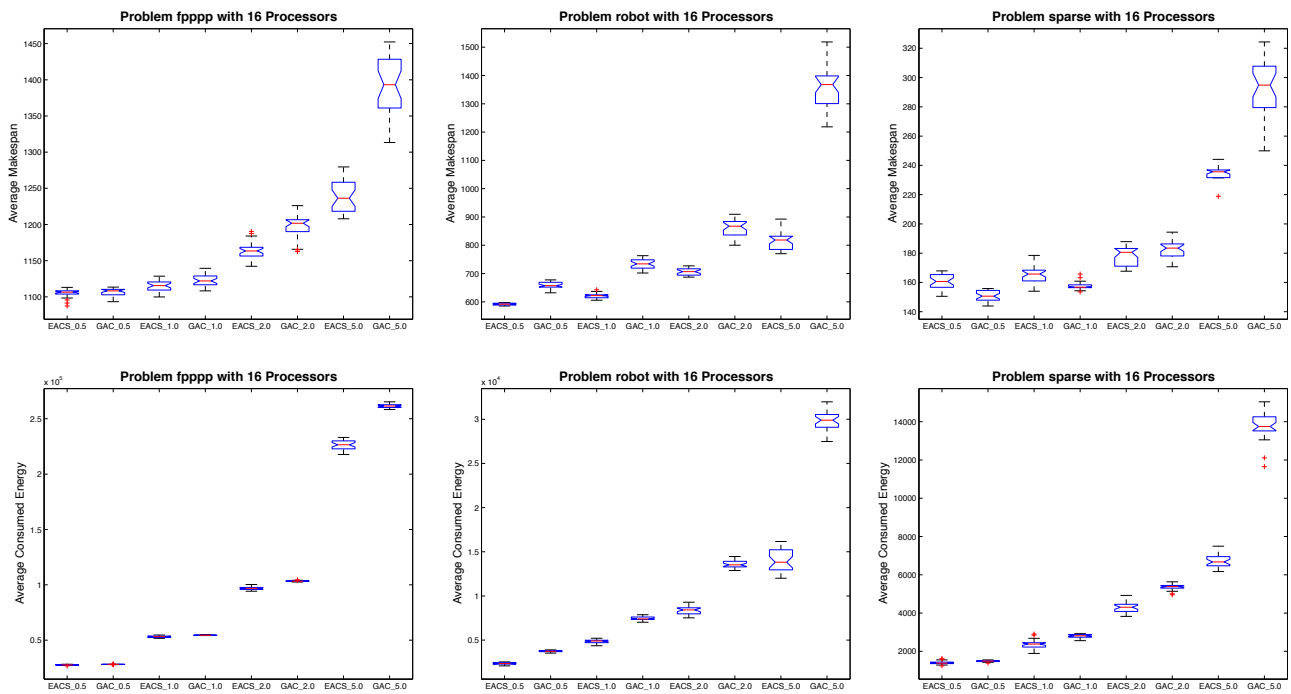


Figure 4. Makespan And Energy Consumed For The Solutions Obtained By EACS And GAC For All The Studied Problems Considering 16 Available Processors

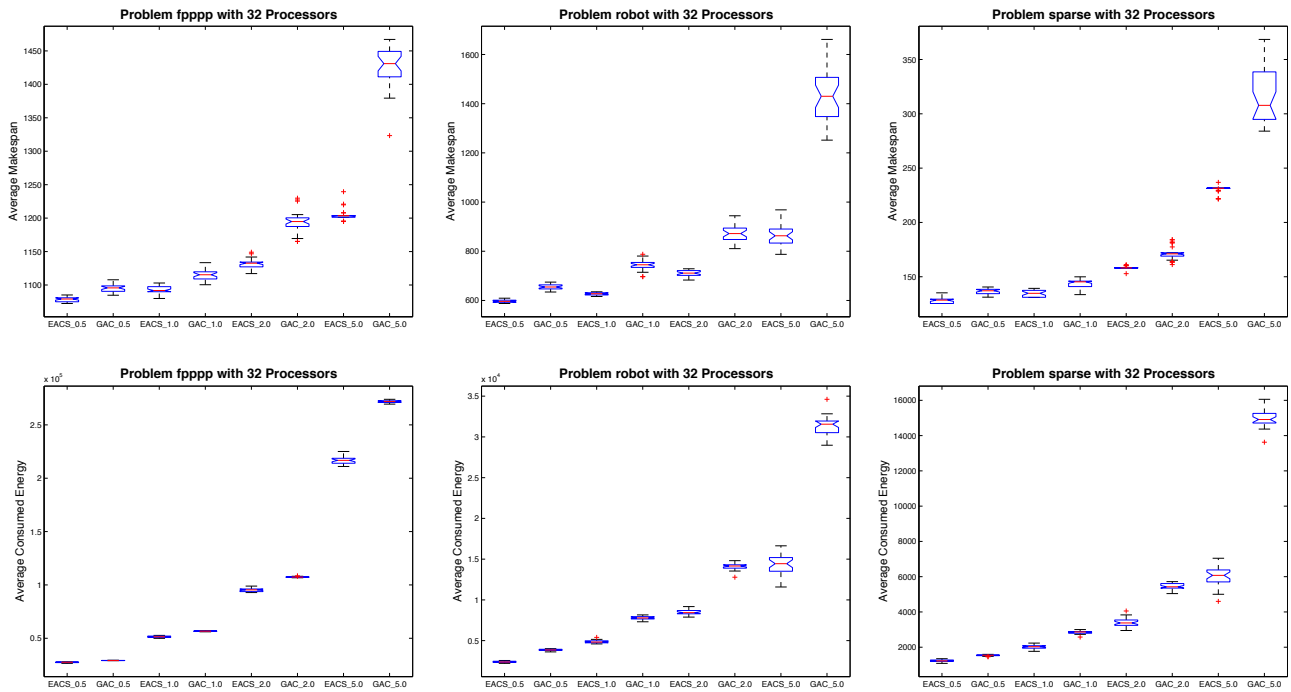


Figure 5. Makespan And Energy Consumed For The Solutions Obtained By EACS And GAC For All The Studied Problems Considering 32 Available Processors

rithms for a larger number of problem instances. Additionally, we are considering increasing the size of the studied instances to deal with more realistic Grid sizes. For that, it could be useful to parallelize the algorithm. Another important prospect is to consider computing systems with heterogeneous link network resources and voltage scalable links. Moreover, we plan to investigate our algorithm assuming systems based on Networks-on-Chips (NoC) technologies, where static partitioning and scheduling applications is especially recommended for NoC where communication overhead could impose significant delays if performed at runtime.

ACKNOWLEDGEMENTS

This work is supported by the National Research Fund (FNR) of Luxembourg through project Green-IT no. C09/IS/05 and the Luxembourg research grant scheme (AFR) through the grant no. PDR-08-010.

REFERENCES

- [1] D. Kondo, B. Javadi, P. Malecot, F. Capello, D. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in Proc. 18th HCW'09 workshop, Italy, 2009.
- [2] S. Irani, S. Shukla, R. Gupta, "Online strategies for dynamic power management in system with multiple power-saving states," *ACM Trans on Embedded Computing Systems*, Vol. 2, No. 3, pp. 325–346, 2003.
- [3] L. Benini, G. Micheli, DYNAMIC POWER MANAGEMENT: DESIGN TECHNIQUES AND CAD TOOLS, Kluwer Academic Publisher, 1998.
- [4] M.T. Schmitz, B.M. Al-Hashimi, P. Eles, SYSTEM-LEVEL DESIGN TECHNIQUES FOR ENERGY EFFICIENT EMBEDDED SYSTEMS, Springer US, 2005.
- [5] S.U. Khan, I. Ahmad, "A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids," *IEEE Trans on Parallel and Distributed Systems*, Vol. 21, No. 4, pp. 537–553, 2009.
- [6] Y.C. Lee, A.Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in Proc of the 9th CCGRID '09. Washington, DC, 2009, pp. 92–99.
- [7] O. Sinnen, TASK SCHEDULING FOR PARALLEL SYSTEMS. Wiley-Interscience, Hoboken, NJ, 2007.
- [8] V.J. Rayward-Smith, "UET scheduling with unit interprocessor communication delays," *Discrete Applied Mathematics*, Vol. 18, No. 1, pp. 55–71, 1987.
- [9] T. Xie, X. Qin, M. Nijim, "Solving energy-latency dilemma: Task allocation for parallel applications in heterogeneous embedded systems," in Proc of the Int Conf on Parallel Processing (ICPP'06), Washington, DC, pp. 12–22, 2006.
- [10] V. Sarkar, PARTITIONING AND SCHEDULING PARALLEL PROGRAMS FOR MULTIPROCESSORS, MIT Press, Cambridge, MA, USA, 1989.
- [11] M.A. Palis, J.C. Liou, D.S.L. Wei, "Task clustering and scheduling for distributed memory parallel architectures," *IEEE Trans on Parallel and Distributed Systems*, Vol. 7, No. 1, pp. 46–55, 1996.
- [12] R. Lepère, D. Trystram, "A new clustering algorithm for large communication delays," in Proc. 11th IPDPS, Florida, USA, pp. 68–73, 2002.
- [13] M. Hakem, F. Butelle, "Critical path scheduling parallel programs on an unbounded number of processors," *Int Journal of Foundations of Computer Science*, Vol. 17, No. 2, pp. 287–301, 2006.
- [14] A. Mahjoub, J.E. Pecero, D. Trystram, "Scheduling with uncertainties on new computing platforms," *Journal Comput Optim Appl*, 2010.
- [15] E. Alba, B. Dorronsoro, CELLULAR GENETIC ALGORITHMS, ser. Operations Research/Computer Science Interfaces, Springer-Verlag Heidelberg, 2008.
- [16] B. Manderick, P. Spiessens, "Fine-grained parallel genetic algorithm," in Third Int Conf on Genetic Algorithms (ICGA), pp. 428–433, 1989.
- [17] D. Whitley, "Cellular genetic algorithms," in Fifth Int Conf on Genetic Algorithms (ICGA), S. Forrest, Ed. California, CA, p. 658, 1993.
- [18] E. Alba, M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 5, pp. 443–462, 2002.
- [19] E. Alba, B. Dorronsoro, M. Giacobini, M. Tomassini, *Handbook of Bioinspired Algorithms and Applications*. CRC Press, ch. Decentralized Cellular Evolutionary Algorithms, pp. 103–120, 2006.
- [20] A.Y. Zomaya, G. Chan, "Efficient clustering for parallel tasks execution in distributed systems," in Proc. of Workshop NIDISC 04, New Mexico, USA, pp. 167–177, 2004.
- [21] E. Falkenauer, GENETIC ALGORITHMS AND GROUPING PROBLEMS, John Wiley and Sons Ltd, England, 1999.
- [22] D.E. Goldberg, GENETIC ALGORITHM IN SEARCH, OPTIMIZATION, MACHINE LEARNING, Addison-Wesley, Boston, MA, USA, 1989.
- [23] T. Tobita, H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J Sched*, Vol. 5, Issue. 5, pp. 379–394, 2002.