A New Genetic Algorithm for Scheduling for Large Communication Delays

Johnatan E. Pecero[1], Denis Trystram[1], and Albert Y. Zomaya[2]

[1] LIG Grenoble University, 38330 Montbonnot Saint-Martin, France
[2] The University of Sydney, Sydney, NSW 2006 Australia

Abstract. In modern parallel and distributed systems, 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 such systems. Accounting for these communications is essential for attaining efficient hardware and software utilization. Therefore, we provide in this paper a new combined approach for scheduling parallel applications with large communication delays on an arbitrary number of processors. In this approach, a genetic algorithm is improved with the introduction of some extra knowledge about the scheduling problem. This knowledge is represented by a class of clustering algorithms introduced recently, namely, convex clusters which are based on structural properties of the parallel applications. The developed algorithm is assessed by simulations run on some families of synthetic task graphs and randomly generated applications. The comparison with related approaches emphasizes its interest.

1 Introduction

In modern parallel and distributed systems, the scheduling problem is more difficult not only by the new characteristics of these systems, but also by the need for data transfer among tasks, which in general are heavily communicated. 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 parallel processor system. Accounting for these communications is essential for attaining efficient hardware and software utilization. Given its importance, several heuristic methods have been developed for considering communications in the scheduling problem. The most widely used of the existing heuristics are *list scheduling*, *task clustering* and *genetic algorithms* [1]. Task clustering is an efficient way to reduce relatively the unbalance between communication delays and processing times. It starts by grouping heavily communicating tasks to the same labeled cluster considering unbounded of processors [2]. A post-clustering step is performed to obtain the final scheduling onto a bounded number of processors; however, if the post-clustering stage is performed without any attention it could degrade the overall system's performance when the number of available processors is less than the generated clusters [3].

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 241–252, 2009.

In this paper, we combine task clustering with a meta-heuristic for efficient scheduling of applications with communication costs, especially for large communications, onto arbitrary number of processors. The main contribution is to develop an effective genetic algorithm, which is called *Genetic Acyclic Clustering Algorithm* (mGACA in short), for scheduling tasks through clustering directly. More precisely, it performs the clustering and post-clustering phases in one step. The major feature of the new algorithm is that it takes advantage of the effectiveness of task clustering for reducing communication delays combined with the ability of the genetic algorithms for exploring and exploiting information of the search space of the scheduling problem. The genetic algorithm makes the clustering based on structured properties of the parallel applications. Moreover, the main focus is on a recent class of structured clustering, called *convex*, which has interesting properties [4] and has been recently used in the context of uncertainties in the scheduling problem [5]. Hence, the final clustering generated by the genetic algorithm will also remain acyclic. The mGACA algorithm is assessed by simulations run on some families of traced task graphs representative of equation solver algorithms. Furthermore, the comparison with related approaches emphasizes its interest.

The rest of the paper is organized as follows. In Section 2, we state the problem, provide some related works and briefly discuss genetic algorithms. Section 3 introduces the new genetic algorithm. Experimental results are given in Section 4. Section 5 concludes the paper.

2 Background and Problem Statement

2.1 Notations and Definitions

Although many studies in scheduling have been focusing on heterogeneous systems, scheduling in homogeneous systems are still of concern because of its wide use and relative simplicity of modeling. In this paper, we consider a generic multiprocessor system composed of m identical processors linked by an interconnection network. While computing, a processor can communicate through one or several of its links and communications among tasks executed on the same processor are neglected. This computational model corresponds to the classical *delay model* [6].

As usually, the application is modeled by a *Directed Acyclic Graph* (DAG). It is represented by a precedence task graph $G = (V, E)$, where V is the set of n nodes corresponding to the tasks of the application that can be executed on the available processors. To every task t_j, there is an associated value p_j representing its processing time. $\mu(V) = \sum_{j \in V} p_j$ represents the time it would take to run all the tasks of the DAG on a single processor. 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 $e_{i,j} \in E$ 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. The weight of any edge stands for the communication requirement

among the connected tasks. Thus, to every edge $e_{ij} \in E$ there is an associated value c_{ij} representing the time needed to transfer data from t_i to t_j. When we refer to large communication it means that $max\ p_i < min\ c_{ij}$.

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 $\prec$.*

The considered scheduling objective is to minimize the makespan of a clustering R on the multiprocessor system when all the communication overheads are included. Unfortunately, finding a schedule of minimal makespan is in general a difficult problem, even if there are unbounded number of processors available [2]. 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 to balance the workload. On the other hand, the more nodes are distributed, the more inter-processor communications, which are expensive, are performed.

2.2 Related Works

There exists mainly two classes of task clustering algorithms: the algorithms based on the critical path analysis and those based on decomposition of the precedence task graph. The critical path based clustering algorithms try to shorten the longest execution path (considering both execution and communication costs) in a precedence task graph. The principle is to group tasks if they belong to the longest path and the relative parallel time is not increased [2,7]. The clustering algorithms based on graph decomposition explicitly handle the trade-off between parallelism and communication overhead. They intent to divide the application into appropriate size and number of tasks to balance communication and parallelism so that the makespan is minimized. The principle of the graph decomposition clustering algorithms is to gather tasks into structured properties. McCreary and Gill in [8] developed a recursive canonical decomposition into *clans* (a group of tasks is a clan if all these tasks have the same predecessors and successors). This decomposition is unique and has also found applications in other fields such as a graph drawing.

Lepère and Trystram [4] provided a recursive decomposition based on the following principle: assigning tasks to locations into *convex clusters*. It means that for any path whose extremities are in the same cluster, all intermediate tasks are also in that cluster. The decomposition founded on convex clusters result in interesting properties. For example, considering arbitrary time execution and large communication delay the authors in [5] showed that the convex clustering are *3-Dominant*. It means that there exists a convex clustering whose execution time on an unbounded number of processors is less than three the time of an optimal clustering. Indeed, we expect better solutions on this specific class of structures. The authors in [5] also investigated convex clusters in the context of the scheduling problem with disturbances. In this context, the authors showed the ability of convex clusters to absorb the bad effects of disturbances on the

communications during the schedule without needing a stabilization process. The main assumption is that the resulting clustering founded on convex clusters does not contain any cycle (i.e., remains a DAG). Based on this assumption the authors in [5] claimed that any convex clustering algorithm is intrinsically stable since there are no cumulative effects of disturbances.

2.3 Genetic Algorithms and Clustering

Genetic Algorithms (GAs) are general-purpose, stochastic search methods where elements (called *individuals* or *chromosomes*) in a given set of solutions (called *population*) are randomly combined and modified (these combinations are called *crossover* and *mutation*) until some termination condition is achieved [9]. GAs use global search techniques to explore different regions of the search space simultaneously by keeping track of a set of potential solutions of diverse characteristics. GAs have been widely used for the scheduling problem in number of ways showing the potential of using this class of algorithms for scheduling [10]. Most of them can be classified as methods that use GA to evolve directly the actual assignment and order of tasks into processors, and methods that use GA in combination with other scheduling techniques. Zomaya and Chan [11] developed a genetic based clustering algorithm (henceforth called GA-cluster in short and will be used in the experimental section) which follows the principle of the Simple Genetic Algorithm [9]. GA-cluster takes as input the number of clusters to be generated 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. Traditional genetic operators are used to generate new individuals. Selection uses a proportionate selection scheme.

3 Genetic Acyclic Clustering Algorithm: mGACA

3.1 The Provided Solution

The solution we provide is based on the decomposition of the task graph on convex clusters. That is, mGACA partitions the graph into convex sets of tasks with the property that the final clustering is a DAG (no cycles). The genetic operators could produce individuals with a number of clusters greater than the number of physical processors. Thus, a merging clustering algorithm is applied after the reproduction process until the number of clusters has been reduced to the actual number of processors. Both, genetic operators and merging clusters algorithm has been designed to produce only legal solutions not violating convexity constraints. Before presenting the new genetic algorithm, we introduce the notion of convex clusters following Lepère and Trystram in [4].

Definition 2 (Convex cluster). *A group of tasks $T \in V$ is convex if and only if all pairs of tasks $(t_x, t_z) \in T$, all the intermediate tasks t_y on any path between t_x and t_z belongs to T.*

For each clustering $R = \{V_i, \prec_i\}_i$, we can build a corresponding *cluster-graph* denoted by $G_R^{cluster}$ and defined as follows: the nodes of $G_R^{cluster}$ are the clusters of R. There exists an edge between two distinct cluster nodes V_i and V_j $(i \neq j)$, if and only if there exist two tasks $t_x \in V_i$ and $t_y \in V_j$ such that $(t_x, t_y) \in E$. Moreover, the graph is weighted by $max\ c_{xy}$ on each edge, and each cluster node V_i is weighted by $\nu(V_i) = \sum_{k \in V_i} p_k$.

Definition 3 (Acyclic clustering). *A clustering R is acyclic, if and only if all the clusters are convex and $G_R^{cluster}$ remains a DAG.*

3.2 Algorithm Design

The mGACA scheduling algorithm follows the principle of the Grouping Genetic Algorithms (GGA). GGA is a genetic algorithm heavily modified to suit the structure of grouping problems [12]. Those are the problems where the aim is to find a good partition of a set, or to group together the members of the set, but in most of these problems, not all possible groupings are allowed. GGA manipulates groups rather than individual objects, similarly mGACA manipulates clusters rather than tasks. The pseudo-code of the mGACA algorithm is given below:

String representation. An essential characteristic of GAs is the coding of the variables that describe the problem. In our GA-based approach, each chromosome is represented by a string of length n, where n is the number of tasks in the given DAG. Each gene of a given string in the chromosome is a tuple, such as (t_j, T_k) where the *i-th* location of the string includes the task t_j and its respective cluster T_k. We enriched this representation with a cluster part, encoding the clusters on one gene for one cluster basis. The length of the cluster part representation is variable and depends on the number of clusters (i.e., the physical processors) given in the input of mGACA, it means, the number of clusters is less than or equal to the number of processors. This representation is similar to that used in [12] to encode valid solutions for *the grouping problems*.

Initial population. mGACA uses a *Bottom-level Clustering Algorithm* (BCA) to create the initial population. BCA clusters the DAG according to the *Bottom level* (blevel) value of every task. The blevel of a task t_i is the length of the longest path from t_i to an exit task, including the processing time of t_i [13]. It is recursively defined by Eq.(1).

$$blevel(t_i) = p_i + max_{t_j \in SUCC(t_i)}\{c_{ij} + blevel(t_j)\} \quad (1)$$

The BCA algorithm works as follows: the algorithm starts by randomly selecting the number of clusters nCl to be generated (between 1 and m) for any individual. After that, the clusters' grain size nt_Cl is calculated. Next, compute the *blevel* for each task and sort the tasks in decreasing order according to their *blevel* value (tie-breaking is done randomly). Afterward, assign the sorted tasks in a list. For each cluster T_i assign the nt_Cl tasks in the top of the list and remove them from that. This process is repeated until the number of clusters has been reached.

As mentioned above, any individual is legal if it does not contain any cycle respecting convexity and precedence constraints. The following proposition proves that BCA always generate legal individuals.

Proposition 1. *The BCA algorithm always generates feasible acyclic clustering.*

Proof. The proof is as follows. Assume that $T_1, ..., T_l$ are clusters generated by BCA and tasks $t_i \in T_i$, $t_j \in T_j$, for $i < j$. Since BCA sorts and clusters tasks in sequence, it is obvious that $blevel(t_i) > blevel(t_j)$. From the definition of *blevel*, t_i cannot be a successor of t_j. Thus, tasks in T_i will not depend on any task in T_j. Therefore, the clusters are convex and the resulting clustering is acyclic. □

Fitness Function. We used the fitness function of the GA-cluster algorithm in [11]. To evaluate the fitness of each solution, first the schedule length of the particular solution is determined, then this schedule length is mapped to an initial fitness value (f'_{S_i}) using Eq.(2):

$$f'_{S_i} = \mu(V) - C_{S_i} \tag{2}$$

Where C_{S_i} represents the schedule length of the individual S_i. However, in some cases f'_{S_i} can still be negative since some of the initial clusterings can result in schedules whose makespan is worse than running the tasks on a single processor. So a linear scaling is employed to obtain the final fitness values (Eq.(3)).

$$f_{S_i} = \frac{f'_{S_i} - MIN[f'_{S_i}]}{MAX[f'_{S_i}] - MIN[f'_{S_i}]} \tag{3}$$

Eq.(3) is then applied to all initial fitness values to map them into the final fitness values, and evaluate the individuals in the population. To obtain the schedule length 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.(4):

$$est(t_i) = MAX[\sigma(t_i) + p_i + c_{ij}, (i,j) \in E] \tag{4}$$

Where: $\sigma(t_i)$ is the starting execution time of task t_i. Let us recall that $c_{ij} = 0$ if tasks t_i and t_j are placed on the same processor. Once all tasks have been scheduled, the algorithm uses Eq.(5) to determine the schedule length of the scheduled DAG.

$$C_{S_j} \equiv MAX[\sigma(t_j) + p_j] \tag{5}$$

Stopping Condition. There are three ways mGACA will possibly terminate. Firstly, if it has reached the maximum number of iterations. Secondly, when the best fitness in a population has not changed for a specified number of generations. Finally, when the difference between the average and the best fitness remains constant for some given number of generations.

Selection. mGACA uses the proportionate selection scheme and replaces the entire previous generation. In order to implement proportional selection, mGACA uses the *roulette wheel* principle of selection. In the roulette wheel selection, the probability for selecting an individual S_i is directly proportional to its fitness.

Crossover. Crossover takes two individuals as input and produces new individuals that have some portions of both parent's genetic material. Hence, the offspring keeps some of the characteristics of the parents. We adapted the crossover algorithm given in [12]. Let S_1 and S_2 be individuals which should generate offspring S_1' and S_2'. The algorithm starts by selecting randomly two crossing sites and delimiting the *crossing section*, in each of the two parents. Then, inject the contents of the crossing section of S_2 (that is, the second parent) in the first crossing site of S_1 (the first parent). Recall that the crossover works with the cluster part of the chromosome, so this means injecting some of the *clusters* from S_2 into S_1. After that, eliminate all *tasks* now occurring twice from the clusters they were belonging to of in the first parent. Consequently, some of the "old" clusters coming from the first parent are altered: they do not contain all the same tasks anymore since some of those tasks had to be eliminated. If necessary, *adapt* the resulting clusters, according to the convex cluster constraint. At this stage, a local problem-dependent heuristic has been used. Finally, apply the same procedure to the two parents with their roles permuted in order to generate the second offspring (i.e., the second child).

The proposed crossover algorithm always generate feasible convex clusters since another local heuristic is used to repair the altered clusters, *"if necessary"*. We have developed a simple local heuristic as follows: we remove the task of the altered cluster violating the convex constraint and allocate it and the set of its predecessors into a new cluster. We only select the predecessors belonging to the altered cluster.

Merging. Since crossover operator can generate chromosomes with a number of clusters greater than the available physical processors, we developed an algorithm to merge existing clusters in a pairwise fashion until the number of clusters is reduced to m. The merging clusters algorithm (MCA) works over the cluster-graph $G_R^{cluster}$. Recall that the clusters represent the task nodes in $G_R^{cluster}$. This procedure initially sorts the elementary clusters by $\nu(V_i)$, to allow the smallest cluster node to be considered for merging at each step. For each cluster calculates its blevel. After that, chooses the smallest cluster. If the chosen cluster will be merged with one of its immediate predecessors, then select the predecessor with the smallest blevel (ties are broken by selecting the immediate predecessor which communicates the most) and merge them in one cluster. In other case, if the chosen task will be merged with one of its immediate successors, then select the immediate successor with the greatest blevel value and merge them in one cluster. Repeat this process until the number of cluster nodes is less than or equal to the number of processors m. Let us remark that, if the chosen task is an entry task

(i.e., a task without predecessors) then the algorithm merges it with any of its immediate successors. In other case, if it is an exit task (i.e., a task without successors) then MCA chooses a task among its immediate predecessors and merges them.

Proposition 2. *MCA always merge $G_R^{cluster}$ in feasible acyclic clustering.*

Proof. Let us proof Proposition 2 with an example. Let us consider the acyclic graph $G_R^{cluster}$. Supposse that we have three cluster nodes of $G_R^{cluster}$ (we name them T_x, T_y and T_z, respectively). Let us consider the partial orders $T_x \prec T_y \prec T_z$ and $T_x \prec T_z$ given in the $G_R^{cluster}$ graph. Now, consider that T_z is selected to be merged with another cluster. Thereafter, $T_x, T_y \in PREC(T_z)$ (the set of immediate predecessors). It is clear that $blevel(T_y) < blevel(T_x)$. As MCA merges only two clusters at the same time and selects always the $min(blevel_{\forall i \in PREC(T_z)}[i])$, thus MCA will select T_y to merge with T_z and the resulting clustering still acyclic clustering. As it holds for all $T_y \in PREC(T_z)$ with the $min(blevel_{\forall T_y \in PREC(T_z)}[T_y])$ this completes the proof. □

Mutation. The mutation operator works by changing a task's cluster to a new cluster. First, an individual is randomly chosen. Next, the mutation operator is applied to the selected chromosome with a certain probability p_m. Then, the mutation operator selects a cluster T_i randomly (from the cluster part of the individual). After that, the mutation operator selects a task t_i from the top or the bottom of the selected cluster T_i, creates a new cluster and puts the task t_i in the new cluster. We recall that the tasks in the clusters are sorted by topological order. Thus, if the mutation operator selects any task from the top or the bottom of any cluster and puts it in a new cluster does not violate the convex cluster constraint and the resulting clustering is acyclic. Finally, after mutation, mGACA verifies if the number of clusters is greater than the number of processors m, then apply the merging algorithm MCA.

4 Experiments

In this section, we present the performance comparison of mGACA and related algorithms by extensive experiments. The related algorithms are the well-known standard algorithm *ETF* [14] and *GA-cluster* described in Section 2.3. Experimentations have been performed on a PC with a 1 Ghz dual-core. The following parameter values were adopted by mGACA and GA-cluster for these experiments: population size equal to 300, crossover probability of 0.75, mutation probability of 0.001, a generation limit of 300, and a percentage of convergence of 0.75. We conducted experiments on two classes of instances: synthetic task graphs, and random graphs. We have used the normalized schedule length (NSL) as a metric for comparison. It is defined as the ratio of the schedule length computed by the algorithm and the sum of the processing times of the tasks on a critical path. Average NSL (ANSL) values are used in our experiments.

4.1 Results on Synthetic Task Graphs

For the first comparison, we present the schedule lengths produced by each algorithm for a set of synthetic task graphs representing basic numeric kernels [15]. Synthetic task graphs can be characterized by the size of the input data matrix because the number of nodes and edges in the task graph depends on the size of this matrix. For example, the number of nodes for LU decomposition graph is equal to $(N^2 + N - 2)/2$ where N is the size of the matrix. We conducted experiments on three different families of synthetic graphs: namely LU decomposition, Cholesky factorization, and Jacobi transformation. For each synthetic graph, we have varied the communication to computation cost ratio (CCR). It is defined as the average of weights of edges divided by the average of tasks' computation times. Figure 1 gives the ANSL values of the algorithms for the Cholesky graphs at various matrix sizes when the number of processors is equal to 8 and 32. For each matrix size, there are three different CCR ratios: 5, 10 and 15. We observe that the performance of the mGACA algorithm outperforms on average the related approaches. For example, the ANSL value of mGACA on 8 processors, is shorter than that of the ETF and GA-cluster algorithms by 12.44% and 50% respectively. Moreover, the ANSL value produced by mGACA on 32 processors, is shorter than that of the ETF and GA-cluster by 15.23% and 61% respectively.

The next experiment is with respect to various CCR ratios. We simulated the algorithms on the LU graph with matrix size equal to 19 (189 tasks) and six CCR values 5, 8, 10, 12, 15, 18. Figure 2 shows the performance of mGACA with the result of ETF and GA-cluster by computing a percentage of improvement of the ANSL:

$$gain = \frac{ANSL_{algorithm} - ANSL_{mGACA}}{ANSL_{algorithm}} * 100 \tag{6}$$

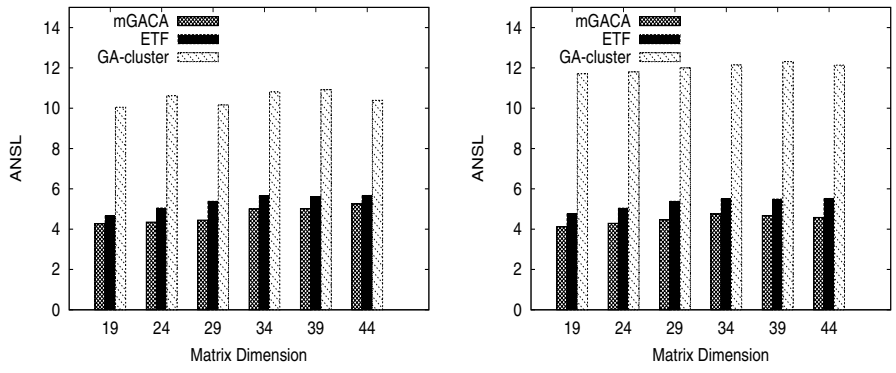

Fig. 1. ANSL for Cholesky factorization graphs with respect to the matrix size on 8 (left) and 32 (right) processors

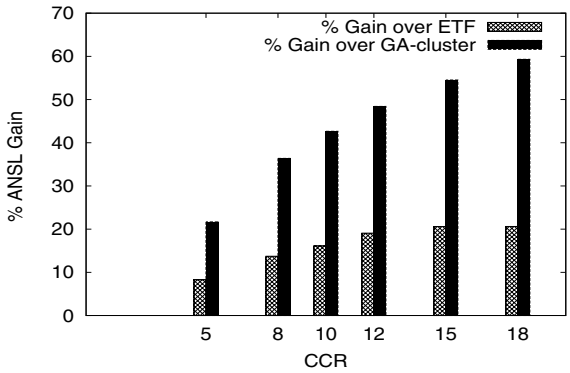

Fig. 2. Percentage improvement in ANSL of mGACA over ETF and GA-cluster

From Figure 2, we notice that mGACA improves the performance of ETF and GA-cluster while increases the CCR. Let us remark that for this experiment mGACA never computed schedules that delayed the makespan (i.e., schedules greater than the sequential time) when scheduling on the LU graphs which is not the case for ETF and GA-cluster.

4.2 Results on Random Graphs

This subsection presents the results obtained on random graphs. We have generated two sets of random graphs: Winkler graphs and Layered random graphs. The Winkler graphs [16] are random graphs representative of multidimensional orders. The three algorithms have been tested on random 2-dimensional orders. To generate 2-dimensional order with n elements, n points are chosen randomly in the $[0;1] \times [0;1]$ square. Each point becomes a node and there is an arc between two points a and b if b is greater than a according both dimensions. To generate the second set of random graphs we have implemented a layered random graph generator such as the one proposed in [7]. The algorithm takes three parameters N, L and p that can be described as follow: N nodes are distributed on L layers and for each couple of successive layers $(l, l+1)$ and each couple $(a+b) \in l \times (l+1)$, there is an arc (a, b) with probability p. No other arcs are present (especially between non successive layers). For generating the two sets of random graphs, we varied two parameters: size and CCR. The size of the graph was varied from 100 to 300 nodes with increment of 50. Five different values of CCR were selected: 2, 5, 10, 15, and 20. There were 30 graphs generated for each combination of those parameters. The processing time was randomly selected from the range $(1, 20)$ with the uniform distribution. The algorithms were scheduling those graphs on three different number of processors (fully connected 8, 16 and 32 processors). In total, there were 2250 different experiment settings. We compared the result of the mGACA with the result of ETF and GA-cluster by computing the percentage of improvement of the ANSL.

Table 1. Percentage gain on the normalized schedule length produced by mGACA over the ETF and GA-cluster

Factor		% gain over ETF			% gain over GA-cluster		
		Best	Avg	Worst	Best	Avg	Worst
CCR	2	−17.25	−19.25	−22.25	−12.02	−11.33	−10.24
	5	−1.32	2.66	6.43	21.585	22.57	23.95
	10	5.23	18.10	25.52	50.02	50.73	51.58
	15	9.71	19.47	28.12	61.34	61.8	62.39
	20	11.77	23.67	32.76	67.90	68.26	68.78
graph size	100	−1.88	7.39	14.60	52.33	54.28	56.02
	150	0.0	10.25	17.11	56.67	57.11	57.80
	200	1.00	11.68	20.37	54.06	54.43	55.01
	250	3.74	15.24	24.09	50.45	50.83	51.33
	300	7.33	14.18	20.64	46.08	46.50	47.13
system	8	3.31	12.03	18.94	48.82	49.88	50.98
	16	3.64	12.43	20.04	52.13	52.73	53.48
	32	5.00	12.57	20.99	53.72	54.08	54.66

We ran each algorithm 20 times on each graph. We compared the best, average and the worst *ANSL*. As the number of experiments is considerable, Table 1 presents only aggregated results. Negative values mean that mGACA gets largest schedules than that computed by the related algorithms. We observe that any increases of the different experiment settings (i.e., CCR, size) the performance of mGACA increases regarding ETF and GA-cluster. We can see that on average mGACA outperforms the related approaches. The most significant improvement is given while increasing communication delays. The mGACA algorithm performs better on graphs with large communications. On the contrary, it is the most important factor that affects the performance of the related approaches. On the one hand, the mGACA algorithm takes advantage of clustering by gathering tasks into appropriate size to balance communication and parallelism while minimizing the makespan. On the other hand, it is well-known that the performance of ETF is $2 - \frac{1}{m} + \rho$ factor of the optimal [14], where ρ is a factor of the communication delays. If communication delays increase the makespan of the ETF algorithm increases as well. With respect to the GA-cluster algorithm, the knowledge-augmented significantly helps mGACA to locate better solutions than GA-cluster.

Regarding the running time, we noticed as expected that mGACA is slower than ETF because the nature of genetic algorithms. However, the running time to compute a solution is admissible in all the experiments, for example, for the Cholesky factorization graph with matrix size of 44 (1034 tasks), mGACA spent 35 seconds on average to compute a solution. On the other hand, the running time of mGACA is smaller than GA-cluster because the knowledge-augmented significantly helps mGACA to explore the search space and locate better solutions spending less time than GA-cluster.

5 Conclusions and Perspectives

We have presented a new genetic algorithm for scheduling tasks through clustering based on the graph decomposition. The new algorithm was improved with the introduction of some knowledge about the scheduling problem. This knowledge is represented by the convex clusters which are based on structural properties of the parallel applications. Extensive experiments have been run for comparing the developed approach to popular related scheduling algorithms and emphasize the interest of this algorithm and convex clusters. The new genetic algorithm seems well suited for new parallel systems like clusters of workstations where the inter-processor communications are much larger than the intra-processor communication costs. One important prospect is to investigate the behavior of the developed algorithm in the context of the scheduling with disturbances on the communication delays.

References

1. Sinnen, O.: Task Scheduling for Parallel Systems. Wiley-Interscience, NJ (2007)
2. Sarkar, V.: Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT Press, Cambridge (1989)
3. Kianzad, V., Bhattacharyya, S.S.: Efficient techniques for clustering and scheduling onto embedded multiprocessors. IEEE TPDS 17(7), 667–680 (2006)
4. Lepère, R., Trystram, D.: A new clustering algorithm for large communication delays. In: Proc. 11th IPDPS 2002, Fort Lauderdale, Florida, April 2002, pp. 68–73 (2002)
5. Mahjoub, A., Pecero, J.E., Trystram, D.: Scheduling with uncertainties on new computing platforms. Journal Comput. Optim. Appl. (to appear)
6. Rayward-Smith, V.J.: Uet scheduling with unit interprocessor communication delays. Discrete Applied Mathematics 18(1), 55–71 (1987)
7. Yang, T., Gerasoulis, A.: Dsc: Scheduling parallel tasks on an unbounded number of processors. IEEE TPDS 5(9), 951–967 (1994)
8. McCreary, C., Gill, H.: Automatic determination of grain size for efficient parallel processing. Comm. of ACM 32(9), 1073–1078 (1989)
9. Goldberg, D.E.: Genetic algorithms in search, optimization, machine learning. Addison-Wesley, Boston (1989)
10. Zomaya, A.Y., Ercal, F., Olariou, S. (eds.): Solutions to parallel and distributed computing problems: Lessons from biological sciences. Wiley, NY (2001)
11. Zomaya, A.Y., Chan, G.: Efficient clustering for parallel tasks execution in distributed systems. In: Proc. NIDISC 2004, IPDPS (April 2004)
12. Falkenauer, E.: Genetic algorithms and grouping problems. John Wiley and Sons Ltd., England (1999)
13. Kwok, Y.K., Ahmad, I.: Benchmarking and comparison of the task graph scheduling algorithms. JPDC 59(3), 381–422 (1999)
14. Hwang, J.J., Chow, Y.C., Angers, F.D., Lee, C.Y.: Scheduling precedence graphs in systems with interprocessor communication times. SIAM Journal on Computing 18(2), 244–257 (1989)
15. Kitajima, J.P., Plateau, B., Bouvry, P., Trystram, D.: Andes: Evaluating mapping strategies with synthetic programs. J. of Syst. Arch. 42(5), 351–365 (1996)
16. Winkler, P.: Random orders. Order 1, 317–331 (1985)