Dynamic Quadratic Assignment to Model Task Assignment Problem to Processors in a 2D Mesh

A. Velarde M.[1], E. Ponce de Leon S.[2], E. Díaz D.[2] and A. Padilla D.[2]

[1] NSS Softtek, Unix Developer
[2] Universidad Autónoma de Aguascalientes, Mexico

Abstract. Mesh multicomputers systems are a viable option for parallel computing. The process of executing a task in such systems is to assign a set of $n+1$ mesh-free processors to the task at the head of the queue consisting of n subtasks. We assume that a task has a parent process and child processes that are called subtasks. These allocations are based on methods that seek to maintain the execution of the task in adjacent processors, and methods that avoid maintaining free sub-meshes in the mesh caused by tasks that have completed their execution. Assign N facilities to a number N of sites or locations where it is considered a cost associated with each of the assignments can be seen as a quadratic assignment problem whose solution combinations grow exponentially, an NP-complete problem. This paper describes a method for modelling dynamic quadratic assignment problem of assigning tasks to processors in 2D mesh multicomputers system, using the simplest class of the Estimation of Distribution Algorithm (EDA), the Univariate Marginal Distribution Algorithm (UMDA) which aims to enable calculation of a distribution joint probability from the selected tasks in the queue that are susceptible to initiate enforcement in the mesh. Experiments are based on a comparison the proposed method with two more methods of allocation of tasks to processors: Hilbert curves and linear assignment. The results show better time allocation, and better utilization of free processors but more times during the process of full recognition of the mesh. The denser workload down to 256 with 256 subtasks tasks each, 16×16 mesh size, that correspond to 256 processors.

Keywords: Parallel computing, mesh multicomputer, estimation of distribution algorithm, univariate marginal distribution algorithm.

1 Introduction

Distributed computing has been considered as the future of high performance computing [1]. The study of partitioning of different distributed computing systems such as multiprocessor systems shared memory multicomputers systems with transfer messages and wide-area distributed systems have been extensively studied in order to get more computing power, and thus permit tasks that are inherently parallel [2], run on shorter time. Examples of these tasks that require more computing power than a single node can provide are: access to distributed

data in different groups of computers [3], the graphics processing tasks [2] and in fields of science and engineering.

Parallel computing consists of a set of processors which cooperate with each other to find a solution to a given problem [4]. The processor communication can be based on shared memory or distributed memory model. In shared memory architectures also known as multiprocessor systems, processors communicate via shared memory. However, in parallel computers with distributed memory also known as multicomputers the processors communicate is by exchanging messages through interconnection network [5].

To solve a problem using parallel computing should break down problem into smaller subproblems that can be solved in parallel. The results should be efficiently combined to obtain the end result of the main problem, but is not easy to break a problem due to data dependency that exists in it and then. Due to data dependency that exists in a problem, it is not easy to divide it into subproblems because the load of communication when the problem is running in parallel is very high. The important point here is the time taken for communication between two processors compared to the processing time. Due to this factor communication scheme should be well planned to get a good parallel algorithm [4].

There are several approaches to implement parallel computing: through meshes [6], squares (grids) [3], and heterogeneous platforms [2], using certain performance metrics, heuristics such as simulated annealing [1] or the greedy algorithm [3], and alternative methods such as measuring the workload, where the workload is generated not by discovery but dynamically by user models that interact with the system and whose behaviour in simulation is similar to the behaviour of users in reality [8].

The vast majority of research converge on the approach that two structures are to be developed in software for coexistence of multiple processors executing tasks in parallel: the allocation Processor and Task Scheduler [9].

Some of the objective functions that is desirable to minimizing or maximizing in the research work are: reduce the starvation of jobs, decrease internal fragmentation, reduce external fragmentation, reduce costs communication within local networks to reduce communication costs in wide area networks, reduce the number of jobs in the queue, maximize the number of jobs running in parallel, maximizing the time response to users to maintain their satisfaction and motivate them to bring more jobs to the system [8].

The optimization problems that model a physical system which involves one objective function perform the task of finding an optimal solution is called a single objective optimization, and when the problem optimization involves more than one objective function, the task of finding one or more optimal solutions is known as multi-objective optimization, also known as Multiple Criterion Decision-Making (MCDM) [10].

In evolutionary computation have emerged parallel multi-objective evolutionary algorithms their application to solve problems with long performance, excessive memory requirements to solve complex problems, decrease the like-

lihood of falling into local optima, find solutions domains simultaneously and work with multi-objective [11].

The objective of this paper is to describe a method for modelling dynamic quadratic assignment problem of assigning tasks to processors in 2D mesh multicomputers system, using the simplest class of the estimation of distribution algorithm (EDA), the univariate marginal distribution algorithm (UMDA)which aims to enable calculation of a distribution joint probability from the selected tasks in the queue that are susceptible to initiate enforcement in the mesh, considering two structures that are necessary to execute tasks in parallel: the Allocation Processor and Task Scheduler.

2 Allocation Algorithms

Efficient allocation of processors and scheduling tasks are two critical processes if the computational power of large-scale multicomputers want to effectively use [12]. Allocation Processor is responsible for finding, selecting and assigning all processors on which a parallel job will run, while the Task Scheduler is responsible for determining the order in which the jobs are selected for execution using a scheduling policy [13].

If a job arrives and can not be immediately executed within the system due to lack of free processors, or the existence of other jobs running, it is sent to the queue. Once the processors are assigned to a task, they remain allocated exclusively to it until the task ends. When the task ends and leaves the system, processors are released and made available to Allocation Processor. One of the main goals of parallel execution is to minimize the time that a job waits a set of free processors in the mesh are assigned to it, so it is important to develop algorithms with efficient allocation strategies processors, that minimize waiting time of tasks within the mesh.

Is necessary to consider the following definitions:

Definition 1. *An n-dimensional mesh has* $k_0 \times k_1 \times \ldots \times k_{n-2} \times k_{n-1}$ *nodes, where* k_i *is the number of nodes along the ith dimension and* $k_i \geq 2$*.Each node is identified by n coordinates,* $\rho_0(a)$, $\rho_1(a)$, ... , $\rho_{n-2}(a)$, $\rho_{n-1}(a)$, *where* $0 \leq \rho_i(a) < k_i$ *for* $0 \leq i < n$*. Two nodes a and b are neighbours if and only if* $\rho_i(a) = \rho_i(b)$ *for all dimensions except for one dimension j, where* $\rho_j(b) = \rho_j(a) \pm 1$*. Each node in a mesh refers to a processor and two neighbours are connected by a direct communication link.*

Definition 2. *A 2D mesh, which is referenced as* $M(W, L)$ *consists of* $W \times L$ *processors, where W is the width of the mesh and L is the length. Each processor is denoted by a pair of coordinates* (x, y)*, where* $0 \leq x < W$ y $0 \leq y < L$*. A processor is connected by a bidirectional communication link to each of its neighbours.*

Definition 3. *In a 2D mesh, $M(W, L)$, a sub-mesh $S(w, l)$ is a sub-two-dimensional mesh nodes belonging to $M(W, L)$ with width w and length l, where $0 < w \leq W$ y $0 < l \leq L$. $S(w, l)$ is represented by the coordinates (x, y, x', y'), where (x, y) is the lower left corner of the sub-mesh and (x', y') is the upper right corner. The node of the lower left corner is called the base node of the sub-mesh node and the upper right corner is the end node. In this case $w = x' - x + 1$ and $l = y' - y + 1$. The size of $S(w, l)$ is $w \times l$ processors.*

Definition 4. *In a 2D mesh $M(W, L)$, an available sub-mesh $S(w, l)$ is a free sub-mesh that satisfies the conditions: $w \geq \alpha$ y $w \geq \beta$ assuming that the assignment of $S(\alpha, \beta)$ requested, where the assignment refers to selecting a set of processors for a task arrival.*

2.1 Processor Allocation Strategies

Two strategies have been developed for the allocation of processors [12], and their classification reflects the type of recognition is performed on the mesh of processors: *contiguous processor allocation strategies* and *non contiguous processor allocation strategies*. In this section we briefly summarize processor allocation research, paying special attention to request partitioning based strategies used in our study.

Strategies contiguous processor allocation, appears when you have a partial recognition of the system and can be assigned for the execution of work, only contiguous sub-mesh of processors to jobs that request. The figure 1 shows a contiguous allocation strategy of 4 processors in a mesh of size 4×4.

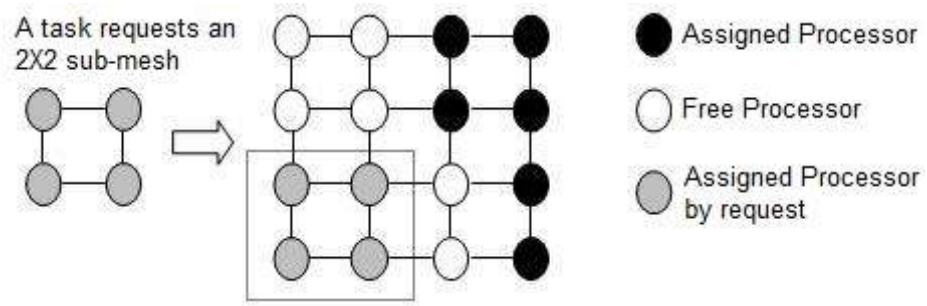

Fig. 1. Contiguous allocation of 4 processors in a mesh of size 4×4.

Such a strategy implies that even with the number of free processors in the system, it is not possible to assign if the sub-meshes are not contiguous. Suppose a task T that requests a sub-mesh 2×2 as shown in figure 1, there is still a number of free processors containing the sub-meshes that are not contiguous, so the task should be put into the queue system where it remains until a sub-mesh size requested is available. This causes a 4-processor external fragmentation.

The fragmentation of processors can be of two types: internal and external [14]. Internal fragmentation occurs when more processors are assigned to a job that really requires, while external fragmentation occurs when there are enough free processors to satisfy outstanding allocation requests but can not be assigned because it is not contiguous, figure 1 exemplifies this kind of fragmentation.

Contiguous processor allocation strategies have been developed by different research for 2D mesh connected multicomputers, examples of these are the Two Dimensional Buddy System (2DBS) [15], Frame Sliding (FS) [16], Adaptive Scan [17] , First Fit (FF) and Best Fit (BF) [18]. 2DBS strategy only applies to square mesh systems by leading to external processor fragmentation. The FS strategy is applicable to a mesh of any size and shape but made âĂŃâĂŃno acknowledgement of all sub-nets free, so it produces external fragmentation. The FS technique applies an operation that allows a frame to slide across the screen. AS strategy improves system performance by applying an exchange procedure orientation of the application when it can not be accommodated in the original orientation, for example if a job requests a sub-grid is $\alpha \times \beta$ not available, can be assigned to a sub-mesh $\beta \times \alpha$ however, the allocation time is high compared with FS because the search of processors in the mesh is at a distance of a processor in a vertical direction. FF and BF strategies detected all free sub-mesh big enough, but lack the ability to complete detection of sub-nets because they do not exchange the orientation of the applications.

To reduce the fragmentation of contiguous processor allocation produced, *non-contiguous processor allocations strategies* have been proposed [19]. In the *non-contiguous allocation* techniques jobs can run on multiple disjoint sub-meshes, avoiding a wait of one sub-mesh size and shape required. Figure 1 when a job requests allocation of a sub-grid of size 2×2) contiguous allocation fails because a sub-mesh the number of processors available is not contiguous. However, the four free processors (drawn with white circles into the figure 1) can be assigned to work when we adopt a non-contiguous allocation. Although non-contiguous allocation can increase the transfer of messages on the network, improving the contiguity to reduce external processor fragmentation and increase usefulness.

The widespread adoption of wormhole routing [19], whose main characteristic is the latency of messages, less sensitive to the distance and have the ability to moderate heavy traffic conditions in practical systems, has consider non-contiguous allocation for multicomputers that use long distance communication. The method used to respond to requests for partitioned allocation has a significant impact on the performance of non-contiguous allocations, so you should go to maintain a high degree of contiguity among processors assigned to a parallel to the overhead of communication reduced without affecting overall system performance [19].

The non-contiguous allocation strategies have been developed are: Random [14], Paging [14], Multiple Buddy Strategy MBS [14], Adaptive Scan Multiple Buddy and ANCA [19], Adaptive Multiple Scan Buddy and AS & MB [21], and variants of page Recent [24]. At Random [14], internal and external fragmentation is eliminated but there is a high interference of communication between

jobs. In the Paging method [14], there is a degree of contiguity among processors assigned to a parallel, which increases if you use more pages size. However there may be internal fragmentation of the processor when pages are allocated to large jobs do not require complete. MBS [14], improves the performance compared to previous strategies but it presents problems when assigning a sub-mesh contiguous free processors, so you can increase the communication overhead. ANCA [19] divide the application in 2^i equal parts in the i^{th} iteration and requires partitioning and assignment occur in the same iteration, which causes in a previous iteration are not allocated sub-meshes to a large part of the application, which can increase the communication overhead. The performance of AS & MB [21], the response time and service are identical the MBS [14], however AS & MB has high overhead allocation for large meshes. In the variants paging unit allocation is a single processor, which requires more time to make a decision allocation in large mesh, while in MBS [14] and ANCA [19], allocation unit increases so it takes a long time to make an assignment in large systems.

3 Task Assignment Problems in a Computer Environment Distributed

The problems of assignment, have evolved along with the architecture of distributed and parallel systems, this evolution has brought some problems with performance, on which is:

- All resources reside under a single domain.
- The resource set is invariant.
- Applications and data reside on the same site or the data collection is a highly predictable.

The scheduling in multicomputers Systems (Job Scheduling Task Scheduler) is the selection and assignment of a partition of processors to a parallel task, with the inherent objective of maximizing the performance of a running workflow [6].The task and its subtasks that are distributed among the processors to be executed communicate with each other to synchronize or to exchange any partial or final result [5].

In multicomputers system once a task or process has assigned to a multiple nodes, can be use any scheduling local algorithm. However, precisely because it has very little control when assigns a process to a node, it is important to the decision of which process should be in which node. Therefore worth considering how to allocate effectively processes to nodes. The algorithms and heuristics used to make such assignment is called processor allocation algorithms [7] performed the schedule Job [7]. There are processor allocation algorithms have been proposed over the years, considering the local assignment in which the task is assigned on one processor and those that allow assignment to a subset of multicomputers system processor, which will be explained in detail in the following sections. Both types of algorithms seek to maximize the clock cycles [7] to avoid and the waste of CPU due to the lack of local labour, minimize the total bandwidth of

communication and ensure equity for users and processes.The differences lie in what is known and what is to be achieved. Among the properties of a process that could be known are: the needs CPU time, memory consumption and the amount of communication with all other processes.

4 Metaheuristics Applied to the Problem of Scheduling Tasks

As defined above, the allocation of tasks is a scheduling problem of tasks that should be assigned to a set of machines. Considering the variation of the job shop (JS) applicable to this research project, then describe some metaheuristics applied to this particular planning problem, in the knowledge that for every aspect of the problem of planning there are different uses of heuristics.

In [25] used branch and bound techniques and present the known graph-directed search method, with which proceeds to traverse a tree, whose nodes are sequences of operations manufacturing in line. They do not consider lines with penalties for delays or for the steps of the tasks from one machine to another, hence the direct applicability of a method sweep of trees in the instrument selection process and trajectories.

In [26] is incorporated two new concepts in the development of an algorithm GRASP (Greedy Randomized Adaptive Search Procedures) for the JSP standard: a strategic escalation procedure (another form of probability distribution) to create candidates for technical solution and a POP (its acronym in English Proximate Optimality Principle) also in the construction phase. Both concepts have already been applied to solve quadratic assignment problem. Genetic algorithms are applied in [27]. In this paper, the representation of chromosomes is based on random charges of elitism, the authors seek to mimic the behaviour of a variable work environment in a flexible production line. On the other hand in [28], is the division of tasks in very small chromosomes so that the populations generated are as varied as possible. Tabu Search is used in [29], in this work are as neighbourhoods for the implementation of the tabu list to the movement of a candidate operation i-ésima task to program a machine to another machine, with the Tabu list that matrix of operations and machines which run, other interesting works can be found in [30] considers a flexible route, which is defined as one where there are machines that can run more than one type of operation, extending the definition of the JSP, in [31] apply the concepts of JSP on distributed computing techniques in programming tasks.

5 The Quadratic Assignment Problem

In Quadratic Assignment Problem (QAP), we have a set of n places and n tasks, and assign to each place a task, so there $n!$ possible assignments. To measure the cost of each possible allocation, multiply the flow between each pair of tasks by the distance between the assigned locations and all pairs are added. Our goal

is to find the allocation that minimizes the costs of this process. Considering the example of assigning the tasks of the input queue of figure 2,where you have 3 tasks with 4 different machines, then you have 12 possible assignments to perform 3 tasks, otherwise, if you have a list of tasks that must be assigned glued to a mesh of processors should consider the distances between processors which will be assigned tasks and subtasks, as well as costs of communication between them [29].

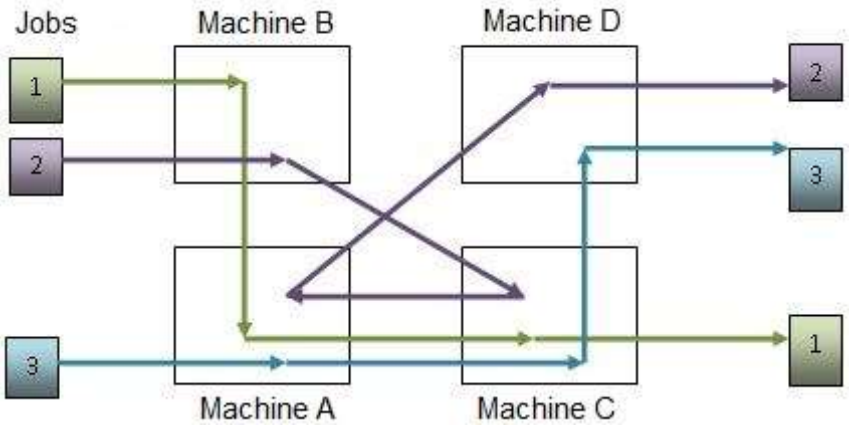

Fig. 2. Three tasks with four different machines

Mathematically we can formulate the problem by defining two matrices of size $n \times n$: a matrix flow F whose $(i,j)-ith$ elements represents the flows between tasks i and j and an array of distances D whose $(i,j) - ith$ elements represent the distance between sites i and j. An assignment is represented by the vector p, which is a permutation of the numbers *1, 2, ... , n* . $p(j)$ is the place where the task j is assigned. With this definition, the quadratic assignment problem can be written as:

$$min_{p\in} \sum_{i=1}^{n} \sum_{i=1}^{n} f_{ij} dp(i)p(j)$$

QAP is NP-complete. It is seen as the problem of NP-complete combinatorial optimization more difficult. Troubleshooting larger than 30 (eg over 900 0-1 variables) is computationally impractical. Among the algorithms used to solve the QAP the Branch and Bound has been the most successful. However, the loss of a lower limit is one of the greatest difficulties, because it is inaccurate or the time required to calculate it is computationally impractical.

6 Statement of Dynamic Quadratic Assignment Problem to model the assignment of tasks to processors in a 2D mesh multicomputers System

Given the gaps in the mesh, the chance to swap tasks at time t is given by the minimization function:

$$min\ e(\pi^t) = (d_{ij})(m_{ij}) + ... + (d_{ik,jk})(m_{ik,jk})$$

Looking for the smallest value found in the set of all possible assignments at time t whose domain is the possible permutations of tasks within a sub-grid.

6.1 Description of the Modelling Problem

The assumption that we have a priori in the Modelling Problem is the knowledge of the degree of communication between the main task and subtasks of all tasks that are in the queue, and the relationship between those subtasks, for example if you have a task *T1* with three subtasks *S11, S12 and S13* the interaction that can occur between them is illustrated in figure 3 through lines showing the message transfer.

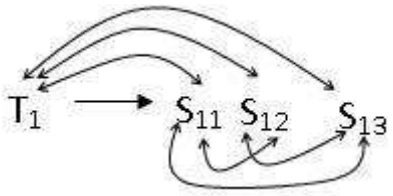

Fig. 3. Message passing between tasks, a task with 3 subtasks.

There is also a symmetric matrix of distances between processors that specifies the hops that a message must be made between a processor and another. Given this process of allocation of processors is based on carrying out a calculation of allocation of processors based on their availability in the Grid and Tasks in the queue.

6.2 Example

Consider the following example. At a time t we have a mesh processor array of size 4×4 whose status is shown in Table 6.2, where 1 represents a processor busy which was assigned to a task at time *t-1*, and 0 is not a free processor has been assigned to a task or subtask, the symmetrical distance between processors are given by a the distance matrix where every value is calculated for each pair of

processors and the distance between them is defined as the length of the shortest path that connects a processor with the other on the mesh. In the queue are 4 Tasks pending execution in the order presented in Table 6.2, these tasks are waiting for the mail run and a matrix of communication costs of each task with its subtasks and among subtasks in Table 6.2.

Table 1. State matrix of the mesh in time t.

1	1	1	1
1	1	1	1
0	0	0	1
0	0	0	0

Table 2. Input queue of tasks in a time t.

T_1	S_{11}	S_{12}	S_{13}	0
T_2	S_{21}	S_{22}	0	0
T_3	S_{31}	S_{32}	S_{33}	0
T_4	S_{41}	0	0	0

To generate the first assignment we take the state matrix of the mesh by a random assignment of tasks taken from the queue and will fit in the free submeshes. Each individual in the population represents an allocation of tasks and subtasks in the gaps as shown in Table 6.2 represents the matrix of assignments according to the state matrix at time t, in this way would generate an initial population random size 2, consisting of tasks T_1 and T_2.

To obtain the first value of the objective function, calculates the cost of the allocation for each task based on the communication costs between tasks and the distances between processors, given the passage of messages from one processor to another and vice versa . When considering message passing between processors must calculate the cost of their transfer, from source to destination and vice versa, i.e. for the case of the transfer rate exemplification of T_1 to S_{11} is different from S_{11} to T_1, but may be that both weights are equal, but the values âĂŃâĂŃof the distances remain the same. So to calculate the values is given in the operations shown in Table 6.2 for the task T_1, and Table 6.2 for the task T_2. The totals of the respective individuals are added to obtain the cost of the solution shown in Table 6.2, which is 35.

This is given by:

C_{ij} is the communication cost of i y j.

i is the task or subtask of the task.

j is the task or subtask of the task.

Table 3. Matrix communication costs between tasks.

	T_1	S_{11}	S_{12}	S_{13}	T_2	S_{21}	S_{22}	T_3	S_{31}	S_{32}	S_{33}	T_4	S_{41}
T_1	0	3	0	3	0	0	0	0	0	0	0	0	0
S_{11}	2	0	1	4	0	0	0	0	0	0	0	0	0
S_{12}	0	1	0	2	0	0	0	0	0	0	0	0	0
S_{13}	3	5	3	0	0	0	0	0	0	0	0	0	0
T_2	0	0	0	0	1	3	0	0	0	0	0	0	0
S_{21}	0	0	0	0	2	0	4	0	0	0	0	0	0
S_{22}	0	0	0	0	4	3	0	0	0	0	0	0	0
T_3	0	0	0	0	0	0	0	0	1	3	2	0	0
S_{31}	0	0	0	0	0	0	0	1	0	1	2	0	0
S_{32}	0	0	0	0	0	0	0	4	5	0	1	0	0
S_{33}	0	0	0	0	0	0	0	2	5	2	0	0	0
T_4	0	0	0	0	0	0	0	0	0	0	0	0	3
S_{41}	0	0	0	0	0	0	0	0	0	0	0	2	0

Table 4. Task allocation matrix according to state of the mesh at time t, which represents a first solution of the dynamic quadratic assignment problem.

1	1	1	1
1	1	1	1
S_{11}	S_{12}	S_{21}	1
T_1	S_{13}	T_2	S_{22}

Table 5. Calculating the cost of transferring messages to the task T_1.

$T_1 \rightarrow S_{11}$	$S_{11} \rightarrow T_1$	$(3+2)*1$	5
$T_1 \rightarrow S_{12}$	$S_{12} \rightarrow T_1$	$(0+0)*2$	0
$T_1 \rightarrow S_{13}$	$S_{13} \rightarrow T_1$	$(3+3)*1$	6
$S_{11} \rightarrow S_{12}$	$S_{12} \rightarrow S_{11}$	$(1+1)*1$	2
$S_{11} \rightarrow S_{13}$	$S_{13} \rightarrow S_{11}$	$(4+5)*2$	18
$S_{12} \rightarrow S_{13}$	$S_{13} \rightarrow S_{12}$	$(2+3)*1$	5
		Total	35

Table 6. Calculating the cost of transferring messages to the task T_2.

$T_2 \rightarrow S_{21}$	$S_{21} \rightarrow T_2$	$(1+2)*1$	3
$T_2 \rightarrow S_{22}$	$S_{22} \rightarrow T_2$	$(3+4)*1$	7
$S_{21} \rightarrow S_{21}$	$S_{22} \rightarrow S_{21}$	$(4+3)*1$	7
		Total	17

C_{ij} is the distance between processors which is assigned the task and subtask or subtasks.

The second assignment is generated by assigning tasks T_3 and T_4 to the mesh as shown in table 6.2.

Table 7. Task Allocation Matrix according to the State of the mesh in time t, which represents a second solution of the dynamic quadratic assignment problem.

1	1	1	1
1	1	1	1
S_{31}	S_{33}	0	1
T_3	S_{32}	T_4	S_{41}

It calculates the second value of the objective function in the same way as above, the values âĂŃâĂŃare given in Table 6.2 for Task T_3, and Table 6.2 for task T_4.

Table 8. Calculating the cost of transferring messages for task T_3.

$T_3 \rightarrow S_{31}$	$S_{31} \rightarrow T_3$	$(1+1)*1$	2
$T_3 \rightarrow S_{32}$	$S_{32} \rightarrow T_3$	$(3+4)*1$	7
$T_3 \rightarrow S_{33}$	$S_{33} \rightarrow T_3$	$(2+2)*2$	8
$S_{31} \rightarrow S_{32}$	$S_{32} \rightarrow S_{31}$	$(1+5)*2$	12
$S_{31} \rightarrow S_{33}$	$S_{33} \rightarrow S_{31}$	$(2+5)*1$	7
$S_{32} \rightarrow S_{33}$	$S_{33} \rightarrow S_{32}$	$(1+2)*1$	3
		Total	39

Table 9. Calculating the cost of transferring messages for task T_4.

$T_4 \rightarrow S_{41}$	$S_{41} \rightarrow T_4$	$(3+2)*1$	5
		Total	5

The generation of the third and fourth assignment shown in Table 6.2 occurs when assigning tasks to the mesh T_2 and T_4 with values âĂŃâĂŃobtained through the calculations in the first and second generation are applied in this, for the third value of the objective function.

In the fourth generation are assigned tasks T_2 and T_3 to the mesh to obtain a fourth value of the objective function. The assignment produced is shown in Table 6.2.

Table 10. Task Allocation Matrix according to the State of the mesh in time t, which represents a third solution to dynamic quadratic assignment problem.

1	1	1	1
1	1	1	1
0	0	T_{21}	1
T_4	S_{41}	T_2	S_{22}

Table 11. Task Allocation Matrix according to the State of the mesh in time t, which represents fourth solution of dynamic quadratic assignment problem.

1	1	1	1
1	1	1	1
S_{31}	S_{33}	T_{21}	1
T_3	S_{32}	T_2	S_{22}

7 UMDA for Dynamic Modeling Quadratic Assignment Problem to Model the Problem of Assigning Tasks to Processors in a 2D Mesh

The behaviour of the algorithms Evolutionary Computation most common (genetic algorithms and evolutionary strategies) depend on various parameters associated with them, crossover and mutation operators, crossover and mutation probabilities, population size, number of generations, replacement rate generation, etc. If you have no experience in the use of evolutionary algorithms in the optimization problem to be resolved, the determination of appropriate values âĂŃâĂŃfor the above parameters in itself becomes an optimization problem [54]. For this reason, together with the fact that predicting the movements of the population of individuals in the search space is extremely difficult, has led to the birth of a type of algorithms known as estimation algorithm distributions (EDAs).

In contrast to the genetic algorithms, EDA do not require operators crossover or mutation. The new population of individuals are obtained by simulating a probability distribution, which is estimated from a database containing selected individuals in generation above. For further reference the reader can consult [54].

In this type of algorithms used to estimate the model in each generation, the joint probability distribution from the selected individuals, $p_l(x)$ is as simple as possible. In fact, the joint probability distribution is factored as a product of independent univariate distributions. That is:

$$p_l(x) = p(x|D_{l-1}^{Se} = \prod_{i=1}^{n} p_l(x_i)$$

Each univariate probability distribution estimated from the frequencies marginal:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_l - 1_e^S)}{N}$$

where:

$$\delta_j\left(X_i = x_i | D_{l-1}^{Se}\right) = \begin{cases} 1 \; if \; in \; the \; j-th \; event \; of \; D_{l-1}^{Se}, \; X_i = x_i \\ 0 \qquad\qquad\qquad\qquad in \; another \; case. \end{cases}$$

The pseudo code for the UMDA, see Table 12.

Table 12. Pseudo code for the UMDA.

UMDA
$D_0 \leftarrow$ Generate M individuals (initial population) randomly

Repeat for $l = 1, 2, ...$ until the stopping criterion check

$D_{l-1}^{Se} \leftarrow$ Select $N \leq M$ individuals
of D_{l-1} according to a selection method

$p_l(x) = p(x|D_{l-1}^{Se} = \prod_{i=1}^{n} p_l(x_i) =$

$\prod_{i=1}^{n} \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_l - 1_e^S)}{N} \leftarrow$
To estimate the joint probability distribution

$D_l \leftarrow$ Sample M Sample m individuals, the new population from $p_l(x))$

Reading Parameters: Once you have completed the first assignment of tasks to the grid of processors, tasks start their execution so that at time t begin to vacate the sub-meshes. producing sets of processors that wait for tasks to be performed, as shown in Table 6.2. Based on the queue of tasks (see section), the planner performs a search of the tasks that can fit in such sub-meshes unoccupied. Once this selection is necessary to generate the initial population.

Initial population generation. Generating the initial population to generate each individual (allocation). The pseudo code for the generation of initial population is shown in Table 13.

Evaluate Population. As explained in Section 6.2, we obtain the total generated in Tables 6.2, 6.2, 6.2 y 6.2. The totals obtained are added to the assessment of an individual by the value obtained by the same objective function. The pseudo code for the evaluation of the population shown in Table 14.

Estimating the Probabilistic Model. In this part we will use the simplest probabilistic model, in which all the variables describing the problem are independent, so we calculate the frequency of apparition of a task in each empty cell of the mesh at time t of a part of the population who are the best individuals through a selection by truncation and the percentage of truncation. In this case the frequency of occurrence can be shown in Table 7, the pseudo code for estimating the probabilistic model shown in Table 16.

Generate the population from the Probabilistic Model. Pseudo code for generation the population from the probabilistic model is shown in Table 17, where

Table 13. Pseudo code for the generation of initial population.

```
Initial Population

Repeat for Tasks = 1, 2, ... until the end of the queue of tasks

    if TasksNumber in the position Tasks =
        Number of free processors in the submeshes

        Storage Allocation

    if not

        Consider the following sub-mesh empty
    end if

Report the number of tasks and identification
```

Table 14. pseudo code for the evaluation of the population.

```
Evaluate Population

Repeat for Tasks = 1, 2, ... until the end of the queue of tasks
    that can be admitted to free submeshes.

    Make the sum of the amounts obtained in each
    calculation of the cost of transferring messages for tasks

Report the total value of the objective function
```

if you generate a random number between 0 and 4 and if it is between 0 and 1 the task T_1 is assigned.

Save the best individual. This process is accomplished by taking each population generated the best individual at the time of ordering the current population. The process of ordering from low to high is because the search is performed is minimized.

8 Experimental Design and Results

Experiments are based on a comparison the proposed method with two more methods of allocation of tasks to processors: *linear assignment* and *Hilbert curves*. *Linear assignment* is the most used method in the allocation of processors, the ease of implementation allows tasks to be quickly placed into the free mesh but presents difficulties when they begin to release sub-meshes and these are not contiguous. The nature of the method allows assignments of the bottom left of the mesh to the right. The recognition that the algorithm makes is based on a linear path of the mesh. The problems with this method are that we can not assign

Table 15. Frequencies of occurrence of each task in each cell.

	P(0,0)	P(0,1)	P(0,2)	P(0,3)	P(1,0)	P(1,1)	P(1,2)
T_1	1	0	0	0	0	0	0
S_{11}	0	0	0	0	1	0	0
S_{12}	0	0	0	0	0	1	0
S_{13}	0	1	0	0	0	0	0
T_2	0	0	3	0	0	0	0
S_{21}	0	0	0	0	0	0	3
S_{22}	0	0	0	3	0	0	0
T_3	2	0	0	0	0	0	0
S_{31}	0	0	0	0	2	0	0
S_{32}	0	2	0	0	0	0	0
S_{33}	0	0	0	0	0	2	0
T_4	1	0	1	0	0	0	0
S_{41}	0	1	0	1	0	0	0
$\sum$	4	4	4	4	3	3	3

Table 16. Pseudo code to Estimate the probabilistic model.

Estimating probabilistic model

Repeat for $AssignedTasks = 1, 2, ...$ until the end of the tables that contain the tasks that can be allocated to submeshes Free.

Verify that the net position of each task is assigned, 1 posted on each assignment and stored in the matrix of the estimated probabilistic model

Report the values obtained in the frequency allocation

free sub-meshes in a different order, and presents difficulties when attempting to take free sub-meshes assignment by the linearity of procedure, this produces a high segmentation in the tasks and increasing the transfer of messages on the mesh.

Hilbert curves method is a very interesting method for assign tasks into the mesh. The Hilbert curve is a space filling curve that visits every point in a square grid with a size of 2×2, 4×4, 8×8, 16×16, or any other power of 2. It was first described by David Hilbert in 1892. Applications of the Hilbert curve are in image processing: especially image compression and dithering. It has advantages in those operations where the coherence between neighbouring pixels is important. The Hilbert curve is also a special version of a quad tree; any image processing function that benefits from the use of quad trees may also use a Hilbert curve.

The results obtained in tests arise in three important ways:

The waiting time processes in the input row. Both methods of assignment Gilbert curves and linear allocation establish a philosophy based on FIFO, both

Table 17. Pseudo code to generate the population from the probabilistic model.

```
Generation the population from the probabilistic model

if (random ≤ Dif[0]) then P(0,0)=Sym[0]
else
    if (random ≤ Dif[1]) then P(0,0)=Sym[1]
else
    if (random ≤ Dif[2]) then P(0,0)=Sym[2]
```

do not compete for the assignment early into the mesh of processors so that a null starvation guaranteed, but the waiting times in the row are proportional to the times of the processes. Both methods ensure the allocation of tasks with subtasks in free submeshes close but as shown in the chart when the number of tasks, subtasks and their sizes increase, the waiting times are increased by the same proportionality.Gilbert curves during the process of free submeshes shows contiguous allocation, but due to allocation method used does not allow use them, but shows a tendency to use cluster of processors, which ensures that as the tasks increase also increases the processor usage across the grid, avoiding leaving idle processors.

The proposed method allows for competition on the waiting list. Every time a part of the mesh is indicated as free processor allocator is responsible to perform a search for tasks that can fit in the sub-grid. There is a greater tendency towards starvation of processes, when small submeshes are unoccupied small tasks are arranged, when small submeshes are unoccupied small tasks are arranged, when large tasks release large number of processors there are occupation by a small tasks too, tasks that require lots of processes waiting indefinitely until the requirements of small tasks are handled so that the waiting time for certain tasks tends to be higher.

The number of occupied meshes. In the two methods that use a FIFO allocation free submeshes processors can remain idle, even if there are jobs in the queue that require equal or lesser number of processors, these tasks wait until the shift assignment will be valid. In the proposed method dispatcher will search tasks that required number of processors is equal to or less than the number of processors in the sub-grid free. As shown in Figure 2, the proposed method allows greater utilization of processors in the allocation of tasks, reducing the idleness of processors.

9 Future Works

The works are planned to develop in the future are: Hardware implementation multicomputers system. Hardware implementation multicomputers system consisting of a number n of dedicated computers interconnected through a medium that allows message passing. Operate within the hardware structure with evolutionary algorithms to enable an evaluation of allocation of processors time,

processes starvations and percentage of idle processors vs percentage of processors used. Allow a real evaluation of evolutionary algorithms in a real scenario. Perform the evaluation with other evolutionary algorithms that include the three aspects that are being evaluated.

Acknowledgements The work was financed by Instituto para el Desarrollo de la Sociedad del Conocimiento del Estado de Aguascalientes México and under the proyect PIINF10-2 of the UAA.

References

1. Jian Chen and Valerie E. Taylor: Mesh Partitioning for Efficient Use of Distributed Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 1, January (2002)
2. Pierre-Francois Dutot, Tchimou N.Takpe, Frederic Suter: Scheduling Parallel Task Graphs on (Almost) Homogeneous Multicluster Platforms. IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 7, July (2009)
3. Alessandro Amoroso, Keith Marzullo: Multiple Job Scheduling in a Connection-Limited Data Parallel System. IEEE Transactions on Parallel and Distributed Systems, Vol. 17, No. 2, February (2006)
4. C. Xavier and S. S. Iyengar: Introduction to Parallel Algorithms. Editorial: Wiley-Interscience, New York USA (1998)
5. A. Yassin Al-Dubai, M. Ould-Khaoua, L. M. Mackenzie: An Efficient Path-Based Multicast Algorithm for Mesh Networks, ipdps, pp.283, International Parallel and Distributed Processing Symposium (IPDPS'03) (2003)
6. Debendra Das Sharma and Dhiraj K. Pradhan: Job Scheduling in Mesh Multicomputers. IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 1, January (1998)
7. T. Srinivasan, Jayesh Seshadri, Arvind Chandrasekhat and J. B. Siddharth Jonathana: Minimal Fragmentation Algorithm for Task Allocation in Mesh-Connected Multicomputers Proceedings. IEEE International Conference on Advances in Intelligent Systems Theory and Applications AISTA 2004 in conjunction with IEEE Computer Society, IEEE Press (2004)
8. Edi Shmueli and Dror G. Feitelson. On Simulation of Parallel-Systems Schedulers: Are We Doing the Right Thing? IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 7, July (2009)
9. A. Velarde M., E. E. Ponce de Leon S., E. Diaz, A. Padilla: Planning and Allocation of processors in 2D meshes. Doctoral Consortium. Mexican Internacional Conference on Artificial Intelligence MICAI 2010 Pachuca Hidalgo, México (2010)
10. Kalyanmoy Deb: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, LTD. New York USA (2001)
11. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison CHRISTIAN BLUM Université Libre de Bruxelles AND ANDREA ROLI Universitá degli Studi di Bologna
12. Saad O. Bani Mohammad: Efficient Processor Allocation Strategies for Mesh-Connected Multicomputers. Tesis Doctoral. Faculty of Information and Mathematical Sciences University of Glasgow UK (2008)

13. B.S. Yoo and C.-R. Das: A Fast and Efficient Processor Allocation Scheme for Mesh-Connected Multicomputers, IEEE Transactions on Parallel & Distributed Systems, vol. 51, no. 1, pp. 46-60 (2002)
14. V. Lo, K. Windisch, W. Liu, and B. Nitzberg: Non-contiguous processor allocation algorithms for mesh-connected multicomputers, IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 7, pp. 712-726 (1997)
15. K. Li and K.H. Cheng: A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System, Journal of Parallel and Distributed Computing, vol. 12, no. 1, pp. 79-83 (1991)
16. P.J. Chuang and N.F. Tzeng: Allocating precise submeshes in mesh connected systems, IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 2, pp. 211-217 (1994)
17. J. Ding and L.-N. Bhuyan: An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems, Proceedings of the 1993 International Conference on Parallel Processing, vol. 2, pp. 193-200 1(993)
18. Y. Zhu: Efficient processor allocation strategies for mesh-connected parallel computers, Journal of Parallel and Distributed Computing, vol. 16, no. 4, pp. 328-337 (1992)
19. C.Y. Chang and P. Mohapatra: Performance improvement of allocation schemes for mesh-connected computers, Journal of Parallel and Distributed Computing, vol. 52, no. 1, pp. 40-68 (1998)
20. J. Mache, V. Lo, and K. Windisch: Minimizing Message-Passing Contention in Fragmentation-Free Processor Allocation, Proceedings of the 10th International Conference on Parallel and Distributed Computing Systems, pp. 120-124 (1997)
21. K. Suzaki, H. Tanuma, S. Hirano, Y. Ichisugi, C. Connelly, and M. Tsukamoto: Multi-tasking Method on Parallel Computers which Combines a Contiguous and Non-contiguous Processor Partitioning Algorithm. Proceedings of the 3rd International Workshop on Applied Parallel Computing, Industrial Computation and Optimization, Lecture Notes in Computer Science, Springer, London, pp. 641- 650 (1996)
22. S. Bani-Mohammad, M. Ould-Khaoua, and I. Ababneh: A New Processor Allocation Strategy with a High Degree of Contiguity in Mesh-Connected Multicomputers, Journal of Simulation Modelling, Practice & Theory, vol. 15, no. 4, pp. 465-480 (2007)
23. P.Liu, Ch. Ch. Hsu and J. J. Wu: I/O Processor Allocation for Mesh Cluster Computers, IEEE Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems ICPADS-05 (2005)
24. D. P. Bunde, V. J. Leung and J. Mache: Communication Patterns and Allocation Strategies, Sandia Technical Report SAND2003-4522 (2004)
25. P. Brucker, B. Jurisch, and B. Sievers: A branch and bound algorithm for the job-shop scheduling problem. Journal of Discrete Applied Mathematics. No. 49, pp.105-127 (1994)
26. S. Binato, W. Hery, D. Loewenstern, and M. Resende: A GRASP for Job Scheduling. Technical Report No. 00.6.1 AT& T Labs Research (2000)
27. J. Goncalves, J. Magalhaes, M. Resende: A Hibrid Genetic algorith for the Job Shop Scheduling. AT& T Labs Research Technical Report TD-5EAL6J (2002)
28. L. Davis: Job shop scheduling with genetic algorithms. First International Conference on Genetic Algorithms and their Applications, pp. 136-140 (1985)
29. E. Taillard: Parallel Taboo Search Technique for the Job shop Scheduling Problem. Journal on Computing Science, No. 6 pp. 108-117 (1994)

30. J. Chambers, W. Barnes: Taboo Search for the Flexible-Routing Job Shop Problem. Department of Computer Sciences, University of Texas-USA, Reporte Técnico (1997)
31. V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan: Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests. Department of Computer and Information Science of Ohio State University, USA. Disponible http://www.gridforum.org (2003)