Planning And Allocation of Tasks in a Multiprocessor System as a Multi-Objective Problem and its Resolution Using Evolutionary Programming*

Apolinar Velarde Martinez
Department of Computer Science
Technological Institute El Llano Aguascalientes
El Llano, Aguascalientes México

Juan Antonio Nungaray Ornelas
Department of Computer Science
Technological Institute El Llano Aguascalientes
El Llano, Aguascalientes México

Eunice Ponce de León Sentí
Department of Computer Science
Universidad Autónoma de Aguascalientes
Aguascalientes, México

Juan Alejandro Montañez de la Torre
Department of Computer Science
Technological Institute El Llano Aguascalientes
El Llano, Aguascalientes México

***Abstract*—the use of Linux-based clusters is a strategy for the development of multiprocessor systems. These types of systems face the problem of efficiently executing the planning and allocation of tasks, for the efficient use of its resources. This paper addresses this as a multi-objective problem, carrying out an analysis of the objectives that are opposed during the planning of the tasks, which are waiting in the queue, before assigning tasks to processors. For this, we propose a method that avoids strategies such as those that use genetic operators, exhaustive searches of contiguous free processors on the target system, and the use of the strict allocation policy: First Come First Serve (FIFO). Instead, we use estimation and simulation of the joint probability distribution as a mechanism of evolution, for obtaining assignments of a set of tasks, which are selected from the waiting queue through the planning policy Random-Order-of-Service (ROS). A set of conducted experiments that compare the results of the FIFO allocation policy, with the results of the proposed method show better results in the criteria of: utilization, throughput, mean turnaround time, waiting time and the total execution time, when system loads are significantly increased.**

Keywords—Multicomputer system; Evolutionary Multi-objective Optimization; First Input First Output; Random-Order-of-Service; Estimation of Distribution Algorithms; Univariate Distribution Algorithm

I. INTRODUCTION

Multi computer systems with architectures and mesh topologies using 2D and 3D interfaces, designed for commercial and research purposes, have been two of the most common networks in research and industrial environments because of their simplicity, scalability, structural regularity and ease of implementation [1, 2, 3]. Examples of such systems are the IBM BlueGene / L [4] and the Intel Paragon [5]. Some of the commercial Multi Computer systems are Multiple Instruction Multiple Data (MIMD) systems with architectures that enable partitions of processor submeshes, and have the advantage of supporting multiple parallel (multi-tasks) jobs [1, 2, 3, 6]. Parallel jobs are usually represented by a Directed Acyclic Graph (DAG), the nodes express the particular tasks partitioned from an application and the edges represent the inter-task communication [7]. The tasks can be dependent or independent; independent tasks, can be executed simultaneously to minimize processing time, and dependent tasks are cumbersome and must be processed in a pre-defined manner, to ensure that all dependencies are satisfied [6]. In an SIMD mesh, that processes parallel jobs, tasks are planned in the queue by a planning policy (usually being First Come First Serve (FCFS)) [2, 3, 8, 9], they are then assigned to the mesh processor, where they remain until they finish their implementation [7]. Planning of resources in the mesh, through hardware partitioning involves two components: a scheduler and dispatcher to the mesh [2, 3, 8, 9]. The function of the scheduler is to choose the next task, or the following tasks in the queue that will be assigned to a sub-mesh, of free processors for execution. The function of the submesh allocator is to locate free submeshes, which are to be assigned to the selected tasks by the scheduler. The allocator uses a contiguous and/or of noncontiguous assignment method. When a contiguous allocation method is used, the tasks partitioned from an application can only be assigned to adjacent processors, unlike a noncontiguous allocation method, where tasks can be assigned in a scattered form across the mesh wherever free processors are located [2,3,8,9]. To maximize the use of resources in the target system, current computer systems opt to use non-contiguous allocation methods, applying wormhole routing and free submesh recognition techniques.

Some examples of this are: the frames processor that uses windows traveling the length and width of the grid [10]; iterative processes that divide submeshes in equal-sized partitions [11], the use of the free-lists approach, [12, 13] among others [22-26]. During the processing of tasks extracted from the queue, we look to optimize a set of objectives that are generally found to be opposite. Upon finalizing the total processing of tasks running on the target system, we seek to optimize a set of proper criteria, from the multiprocessor systems. In the following paragraphs, we list the objectives and

This investigation is sponsored by Tecnológico Nacional de México, and developed in Instituto Tecnológico el Llano, Aguascalientes, México.

exemplify the form in which they are opposed, as well as the list of criteria that is optimized.

The objectives sought to be optimized, for processing tasks using the scheduler and proposed in [14] are:

1) Reduce the waiting time of tasks in the queue, assigning more tasks to the mesh of processors once the allocator reports, the number of processors in the free submeshes.

2) Reduce task starvation, that is, avoid discrimination in the allocation of tasks that require a lot of processors (great tasks), caused by the continued allocation of tasks requiring a lesser amount of processors (small tasks).

3) Minimize external fragmentation, that is, minimize the percentage of free processors, after the allocation algorithm places one or more tasks in the processor mesh.

4) Minimize the communication overhead or network contention [15], through contiguity between processors (as close as possible to assign the set of free processors), in order to decrease the distance in the communication path, and avoid interference between the processing elements (searches for the best way to accommodate tasks in the free processors). This point is then identified as the quadratic dynamic allocation of tasks.

Upon complete processing of all tasks in the system, we then seek to optimize the system utilization criteria, throughput, response rate, mean turnaround time and overall waiting time [15, 16].

A simple example of the contrast of the previous four objectives occurs, when we look to minimize external fragmentation by using a noncontiguous allocation method. The largest number of jobs in the free processors is allocated regardless of their location in the mesh, resulting in the maximization of system utilization; however, the opposite effect is produced upon maximizing the communication overhead, between tasks if not assigned contiguously along the length and width of the grid. Thus, in seeking to maximize or minimize some of the objectives, in order to optimize your results, usually the result of another objective is degraded, producing contrasting results between themselves, enabling oneself to view the problem of task planning and allocation as a multi-objective problem.

A multi-objective problem, involves optimizing a number of targets simultaneously, and its solution with or without the presence of constraints, results in a set of interchangeable optimal solutions called the search space, popularly known as Pareto-optimal solutions. For an adequate solution in this, evolutionary optimization algorithms are utilized (EOA), which use a population focuses in their search procedure [17]. The EOA's possess several characteristics that are desirable for problems, involving multiple conflicting objectives and intractably large and highly complex search spaces [18].

In this paper, a hybrid method is proposed to address the problem of planning and allocation of multiple parallel jobs in a multiprocessor system, as a multi-objective problem. In this manner, it makes use of the scheduler and allocator to achieve the best assignments in the processor grid, which optimize the resources of the target system, during processing and completion of tasks. This method uses a static task scheduling, defined as a scheduling at compile time [19].

The proposed method is evaluated with two task selection policies from the queue: FIFO and ROS [20]. This method connects the planner and the dispatcher to conduct the process of task selection, from the queue randomly and makes the best assignment in the processor grid, by evaluating a set of conflicting objectives. The work that the scheduler and dispatcher does is divided into five steps as follows: first, the dispatcher reports the number of free processors that the grid has in time *t*, in the second step, by means of the Random-Order policy-of-service the scheduler selects the same number of tasks, with subtasks from the queue that the allocator previously reported, regardless of the location of the processors across the grid. This set of selected tasks, is considered a feasible solution of the search space, to which three disjointed objectives are evaluated: the waiting time from the rest of the tasks that remain in the queue, the starvation of the tasks of the queue (if occurs), the external fragmentation, and the communication overhead. In the third step, the process of dynamic selection of tasks by the planner continues until to the stop criterion is fulfilled. Lastly, for the set of feasible solutions, the joint probability distribution can be appreciated using the algorithm UMDA (Univariate Marginal Distribution Algorithm), to obtain the best allocation to the processor grid. After finalizing the total execution of tasks, the following criteria is evaluated and compared: system utilization, throughput, response rate, mean waiting time and turnaround time with different workloads in the target system; the effectiveness of the proposed method is compared with the most widely used task planning method: FCFS.

This paper is organized as follows: In section 2, we discuss a classification of methods that throughout the years have been proposed for the planning and allocation of tasks using heuristics techniques and geometric models. In section 3, a stopping criterion is performed using a definition of the objectives, and the form in which they are opposed during the execution of tasks. Section 4, describes the functionality proposed in this research method. Section 5, describes the experiments conducted by the method. In section 6, the future work to develop after this research is described, and section 7, conclusions, describes the findings of this research.

II. Related Works

In [19], two classes or categories for scheduling are specified, a) list scheduling and b) clustering; in this paper, related jobs are classified depending on 1) planner use and heuristic techniques, and 2) allocator use in conjunction with geometric patterns, and free submesh searches throughout the grid.

A. Task scheduling methods that make use of the planner and heuristic techniques

Heuristic methods base their functionality in genetic algorithms (GA), which are global search techniques that explore different regions of the search space simultaneously, by keeping track of sets of potential solutions called a population [21]. Over the years, different methods have been

proposed based on this search technique. In this section we can observe how a set of these investigations show the similarities in the operators used.

In [16], the multiprocessor scheduling problem is based on the deterministic model, and the precedence relationship among the tasks is represented by an acyclic directed graph. This method uses a representation based on the schedule of the tasks, in each individual processor. Several lists of computational tasks represent the planner, respecting the order of precedence of the tasks.

Each list can be further viewed as a specific permutation of the tasks in the list. Crossover operators, reproduction and mutation are applied to the created lists, in order to optimize the finishing time of the schedule. Similar research is presented in [22], where the scheduling problem is formulated in a genetic search framework based on the observation, that if the tasks of a parallel program are arranged properly in a list, an optimal schedule may be obtained by scheduling the tasks, one by one according to their order in the list. These lists are codified in chromosomes, which represent feasible solutions in the search space; genetic search operators are applied to these chromosomes, such as crossover and mutation, as well as an additional operator called "an investment operator". Chromosomes are manipulated by genetic operators, in order to determine an optimal scheduling list, leading to an optimal schedule. To improve convergence time of the proposed algorithm, the connected synchronous island model is used. In [23], the proposed genetic algorithm minimizes the schedule length of a task graph, to be executed on a multiprocessor system. It uses processes that evolve candidate solutions by the use of a set of operators, such as fitness-proportionate reproduction, crossover and mutation; doing so through a traditional method of genetic algorithms. This algorithm does not consider the communication time between tasks. In [24], an initial chromosome, consisting of genes is generated, where each gene will use the priority of node in a directed acyclic task graph (DAG); trade, crossover and mutation operators are applied to the chromosome in order maximize the makespan of the *k-th* chromosome, using an evaluation function. Communication costs are not considered in this paper. In [25], a Modified List Scheduling Heuristic (MLSH) and hybrid approach composed of genetic algorithms, and MLSH for task scheduling in multiprocessor systems is used; this method uses three new different types of chromosomes: task list, list processor and a combination of both types. In order to maximize the finishing time of schedule, the genetic operators: crossover and mutation, used in the chromosomes are the selection. The main features of this type of method are the use of a set of genetic operators (parameters), which seek to optimize a single objective function (maximize execution times of the tasks). Nevertheless, if a researcher does not have experience in using this type of an approach for the resolution of a concrete optimization problem, then the choice of suitable values for the parameters can be converted into an optimization problem [26]. Similarly, in these methods as the complexity of the task graphs and the proposed solution are increased, the number of operators, that must manipulate the algorithm to try to find the best solutions in the search space also increases; best case scenario being the possibility that the algorithm will land in the least amount of local minimums.

B. Methods with geometric models for scheduling

Alternately to heuristic methods, other methods seek to solve the problem of task planning and allocation by looking for free processors, that are contiguous to the length and width of the grid; ensuring that the tasks assigned during implementation remain as close together as possible.

In [27], a submesh reservation strategy for incoming tasks is used, this method combines a submesh reservation technique with a priority technique as follows: an incoming task requests a number of processors, a reservation will occur if these cannot be assigned to of a set of processors constituted in a sub-mesh, as long as it does not exceed the threshold established within the parameter FREE_FRAC. The priority of waiting tasks is handled through a "no_supercede" parameter, which allows you to suspend allocations if the threshold in the parameter MAX_PRI is exceeded, and it also prioritizes tasks that have aged in the waiting queue. In [28], the approach contains a list of allocated submeshes, sorted in a non-increasing order by the second coordinate in their upper right corner. This list serves two purposes: first, it determines the nodes that cannot be used as a basis for new free submesh applications and second, it identifies nodes that are located on the right edge of the assigned submeshes, in order find the nodes that could be used as a basis in finding free submeshes. When a parallel job is selected to be assigned, a search is performed to locate a suitable sub-mesh, if this does not occur; the assignment is made with longest free submesh, whose length of sides does not exceed the requested submesh. Through a search process, the free submesh that best fits the application is located. Other current techniques, through an initial strategy, look to make the allocation of tasks to the mesh, but if resulted in failure, a second allocation strategy is activated to replace the first in order achieve the assignment. For example in [29], the First Fit technique (FF) proposed in [30], that searches for free submeshes best suiting the application (to find the maximum adjacency between processors while reducing communication latency between tasks), is used in conjunction with the Best Fit (BF) technique proposed in [31]. This technique searches for the exact number of processors that the task requires in the free submeshes; thus, in [29], if a task requests a 4x4 sub-mesh and the request cannot be granted, the request size is reduced by a multiple of 2, then a 2x2 grid will be requested and so on until the request is the minimum number of processors, 1 X 1 in this case. When the first technique fails, the second technique BF is enabled, and through this, a search is performed within the free submeshes which best fit, that is, with the exact number of processors that the task requires [31].

III. Basic Concepts

This section describes the concepts and the evolutionary algorithm used in this research.

A. Definitions

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

Definition 2: A 2D mesh, which is referenced as *M (W, L)* consists of *W X L* processors, where *W* is the width of the mesh and *L* is the height of the mesh. Each processor is denoted by a pair of coordinates *(x, y)*, where: $0 \leq x < W$ and $0 \leq y < L$. A processor is connected by a bidirectional communication link to each of its neighbors. For each 2D mesh 2D $a=P_{ij}$.

Definition 3: In a 2D mesh, *M (W, L)*, a sub-mesh: *S (w, l)* is a two-dimensional mesh belonging to *M (W, L)* with width *w* and height *l*, where $0 < w \leq W$ and $0 < l \leq L$. *S (w, l)* are represented by the coordinates *(x, y, x', y')*, where *(x, y)* is the lower left corner of the submesh and *(x', y')* is the upper right corner. The node in the lower left corner is called the base node of the sub-mesh, 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 x l* processors.

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

Definition 5: The correspondence of a task or subtask to a free processor in the mesh is defined as the following: if ϑ is a set of system tasks, and $\vartheta = J_1, J_2, \ldots, J_n$ where *n* is the number of tasks in time *t* and ϑk is a set of sub-tasks of task *k* where: $\vartheta_k = j_{k1}, j_{k2}, \ldots, j_{kf}(k)$ and *f(k)* is the total number of sub-tasks of task *j*. For each task *j* and each sub-task $f(k) \in j$ has a processor in $m_i \in P$ in which it is to execute each task j and each sub-task $j_{kf}(k)$, consuming an uninterrupted time $t \in N$.

Definition 6: Given two matrices size *n x n*: a flow matrix *F* whose *(i, j) -th* elements represent flows between tasks *i* and *j* and an arrangement of distances D, whose *(i, j) -th* elements represent the distance between sites *i* and *j*. An assignment is represented by vector *p*, which is a permutation of the numbers 1, 2,…, *n* and *p(j)* is where the task *j* is assigned. Thus, the quadratic task assignments can be written as:

$$\min_p \varepsilon \sum_{i=1}^{n} \sum_{i=1}^{n} f_{ij} dp(i) p(j) \tag{1}$$

Definition 7: An optimization problem is one whose solution involves finding a set of candidate alternative solutions that best meet the objectives. Formally, the problem consists of the solution space *S* and function objective *f*. Solving the optimization problem *(S, f)* consists of determining an optimal solution, namely, a feasible solution $x^* \in S$ such that $f(x^*) \leq f(x)$, for any $x \in S$. Alternative solutions can be expressed by assigning values to some finite set of variables $X = \{X_i: i = 1,2, ..., n\}$. If U_i is denoted the domain or universe (set of possible values) of each of these *n* variables. The problem consists of selecting the value x_i that is assigned to each variable X_i from domain U_i that when subjected to certain restrictions, optimizes an objective function *F*. The universe of solutions is identified with the set $U = \{x = (x_i: i=1, 2, \ldots, n): x_i \in U_i\}$. The problem constraints reduce the universe of solutions to a subset of $S \subseteq U$ called feasible space.

A performance evaluation of a parallel system, upon finalizing the processing of all running tasks, is evaluated on the following criteria [1]:

Definition 8: The utilization is defined as the fraction of time in which the system was used, and is given by:

$$U_G = W_G / (C_G * m_G) \tag{2}$$

Where: W_G is the amount of work that the system performs, C_G is the completion time of execution of all tasks in the system and m_G is the total number of processors in the system.

Definition 9: Throughput. The number of completed tasks per unit of time in the system is given by:

$$n / C_G \tag{3}$$

Where: *n* is the total number of jobs in the system.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

Definition 10: Mean turnaround time. The average time it takes all tasks from entering the local queue until their execution is finalized. Calculated as:

$$\frac{1}{n} \sum_{j=1}^{n} t_t^j \tag{4}$$

Where:

$$t_t^j = c^j - r^j$$

c^j is the completion time of the task and r^j is the delivery time of task *j*.

Definition 11: Waiting time, defined as the average waiting time before starting the task execution. Calculated as:

$$\frac{1}{n} \sum_{j=1}^{n} t_w^j \tag{5}$$

Where:

$$t_w^j = t_s^j - r^j$$

t_s^j is the start time of execution of task j.

Definition 12: Response ratio, defined as the coefficient response average of all tasks. Defined as:

$$\frac{1}{n} \sum_{j=1}^{n} (t_w^j + p^j) / p^j \tag{6}$$

Where: p^j is the runtime and t_w^j is the waiting time of task j.

B. UMDA for dynamic quadratic assignment to model the problem of task scheduling

The EDA (Distribution Evolutionary Algorithm) uses estimation and simulation, from the joint probability distribution as a mechanism of evolution, instead of, directly manipulating the individuals that represent solutions to the

problem [26]. An EDA begins by randomly generating a population of individuals, which represent solutions to the problem, iteratively performs three types of operations on the population: a subset of the best individuals of the population is generated; a learning process from a probability distribution model from selected individuals is performed, and new individuals that simulate the obtained distribution model are generated. The algorithm stops when a certain number of generations are reached, or when the performance of the population fails to significantly improve; an UMDA is used to estimate the joint distribution in each generation from selected individuals. Thus, the joint probability distribution is factorized as a product of independent univariate distributions, i.e.:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i \mid D_{l-1}^{S_e})}{N}$$

Where:

$$\delta_j(X_i = x_i \mid D_{l-1}^{S_e}) = \begin{Bmatrix} 1 \, si \, en \, el \;\; j-esimo \; caso \; de D_{l-1}^{S_e}, X_i = x_i \\ 0 \, en \, otro \, caso \end{Bmatrix}$$

The pseudocode for an UMDA algorithm is as follows:

Generate M individuals (the initial population) randomly

Repeat for l = 1,2, .. until the stop criterion:

$D_{l=1}^{S_e} \leftarrow$ *Select* $N \leq M$ *individuals of* $D_{l=1}$ *in accordance to the selection method*

$$p_l(x) = p(x \mid D_{l-1}^{S_e}) = \prod_{i=1}^{n} p_l(x_i) = p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i \mid D_{l-1}^{S_e})}{N}$$

Estimate the joint probability distribution.

Dl $\leftarrow$ *Sample M individuals (the new population) from* $p_l(x)$

IV. Statement of the Proposed Method

This section is structured as follows: Section 4.1 shows three instantiations of the manner in which the objectives are opposed during the planning and allocation of tasks. In section 4.2 the functionality of the proposed method is explained in detail.

A. Contraposition of the objectives during job processing

The way that the objectives are opposed during job processing is shown in [32], it is explained through three examples in the following sections. In Figure 1, an 8x8 2D processor mesh is shown; the 35 occupied processors are shown in closed circles and the 29 free processors, with unfilled circles. In the queue, a set of 6 dependent tasks partitioned from an application wait for execution: task T_0 with 4 subtasks, task T_1 with 3 subtasks, task T_2 with 4 subtasks, task T_3 with 3 subtasks, task T_4 with two subtasks and task T_5 with 25 subtasks, supposing that the planning method can choose more than one task to be assigned in the processor mesh with noncontiguous allocation method.

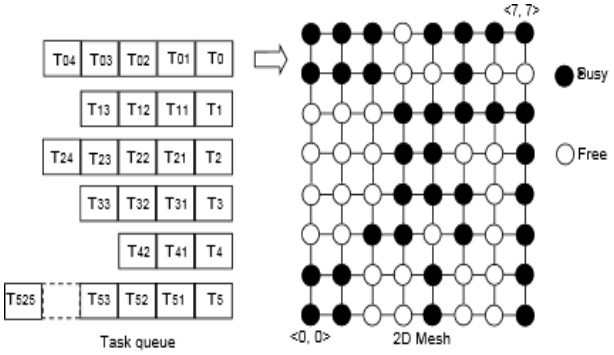


Fig. 1. System structure for task execution on a Multicomputer 2D mesh system

Example 1: Consider that in time *t*, the allocator reports the 29 free processors, with this data the scheduler determines that the set of 5 tasks: T_0, T_1, T_2, T_3 and T_4 are candidates to occupy 21 processors in the mesh, or assign task T_5 requiring 26 processors and task T_4 requesting 3 processors.

By assigning the set of the 5 tasks, the same number of positions in the queue are released allowing the entry of new tasks, and the number of accesses to the queue is decreased in order perform more task searches. The previous procedure allows for more than one task to enter the mesh, and decreases task waiting time at the head of the queue; but in opposition to each other, the assigning of these 5 tasks generates an external fragmentation of 8 processors and produces starvation of task T_5 in this assignment. The result that is had is a contrast between objectives 1 and 2.

Objective 1 seeks to minimize the number of assignments to the mesh of processors in order minimize task waiting time, and objective 2 seeks to maximize the use of the processors in the mesh and minimize starvation of the large tasks. Now, if tasks T_4 and T_5 are assigned, neither starvation nor external fragmentation occurs, but a smaller number of tasks can be accepted in the queue, and so the number of assignments to the mesh increases therefore, also increasing the time tasks must wait to enter the processor mesh.

Example 2: In order to illustrate the contrast between objectives 3 and 4, consider Figure 1. Objective 3 seeks to maximize the use of the processors in the mesh, avoiding external fragmentation, and Objective 4 seeks to minimize overhead communication through minimizing the adjacency of processors that are assigned to a task. The assumed set of the 5 selected tasks are: T_0, T_1, T_2, T_3 and T_4, and are allocated in contiguous processors as follows: T_0 task is assigned in submesh <4,0> <5,2> regardless of the processor in position <4,2>, task T_1 is assigned to the sub-mesh <2,0> <3,1>, task T_2 is assigned the submesh in <0,5> <2,6> regardless of the processor in position <2,5>, task T_3 is assigned in submesh <0,2> <1,3>, and task T_4 is assigned in submesh <6,3> <7,4> regardless of the processor in position <7,4>. This allocation maximizes the adjacency between processors, and produces an external fragmentation of 8 processors. Now if the system

assigns task T_5, together with task T_1 or T_3, all of the free processors will be used, and in opposition to the allocation of the 5 tasks, external fragmentation will be minimized. Thus, the contrast of goals 3 and 5 is produced.

Example 3: Exemplification of the contrast between the objectives of the minimization of task, residence time in the queue and the maximization of communication overhead (objectives 1 and 4), is shown when a large number of tasks are sought to be assigned in the processor mesh, and processors to which tasks are to be assigned are not close enough together or contiguous. This is done to avoid producing very high communication costs. As an example, consider allocating the 5 task set: T_0, T_1, T_2, T_3 and T_4. The number of allocations made to the mesh is minimized, but if the allocator does not consider assignment of disjoint processors by a previous calculation method of communication overhead, tasks will be assigned disjoint in the mesh, causing adjacency to be minimal and communication costs between tasks to be very high.

B. Functionality of the proposed method

Proposal: In time *t* a 4X4 processor mesh is had, whose status array is shown in Figure 2, where the number 1 represents the occupied processors that were assigned to a task at time *t-1*, and the number 0 represents the free processors that have not been assigned to a task or sub-task.

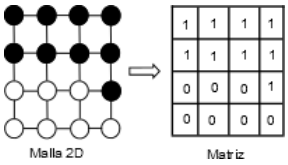


Fig. 2. 4X4 processor mesh represented by a matrix

Symmetrical distances between processors are given in Table I (due to space constraints, only half of the table is shown), these distances represent the "jumps" that a message must execute in order to achieve communication between two processors.

TABLE I. SYMMETRICAL DISTANCES BETWEEN PROCESSORS IN A 4X4 MESH FROM FIGURE 5

	1	2	3	4	5	6	7	8	9
1	0	1	2	3	1	2	3	4	2
2	1	0	1	2	2	1	2	3	3
3	2	1	0	1	3	2	1	2	4
4	3	2	1	0	4	3	2	1	5
5	1	2	3	4	0	1	2	3	1
6	2	1	2	3	1	0	1	2	2
7	3	2	1	2	2	1	0	2	3
8	4	3	2	1	3	2	1	0	4
9	2	3	4	5	1	2	3	4	0

Table 2 shows the waiting queue containing 4 pending execution tasks; said tasks are waiting to be executed in the mesh.

TABLE II. WAITING QUEUE IN TIME T WITH 4 TASKS EACH CONTAINING 4 SUBTASKS

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

Previous knowledge is had of the extent of the degree of communication (communication costs), between the main task and sub-tasks that is composed of all tasks that are found in the waiting queue, and the relationship between the same sub-tasks. Table 3, shows the matrix of communication costs for tasks T_1 and T_2; Table 4, shows the matrix of communication costs for T_3 and T_4 tasks; communication costs are established between the main task and subtasks and between subtasks. For example, the communication cost between task T_1 and subtask T_{11} is 3.

TABLE III. MATRIX COMMUNICATION COSTS FOR TASKS T_1, T_2

	T_1	T_{11}	T_{12}	T_{13}	T_2	T_{21}	T_{22}
T_1	0	3	0	3	0	0	0
T_{11}	2	0	1	4	0	0	0
T_{12}	0	1	0	2	0	0	0
T_{13}	3	5	3	0	0	0	0
T_2	0	0	0	0	1	3	0
T_{21}	0	0	0	0	1	3	0
T_{22}	0	0	0	0	2	0	4

TABLE IV. MATRIX COMMUNICATION COSTS FOR T_3, T_4 TASKS

	T_3	T_{31}	T_{32}	T_{33}	T_4	T_{41}
T_3	0	1	3	2	0	0
T_{31}	1	0	1	2	0	0
T_{32}	4	5	0	1	0	0
T_{33}	2	5	2	0	0	0
T_4	0	0	0	0	0	3
T_{41}	0	0	0	0	2	0

To illustrate the relationship between task and subtasks, consider that you have task T_1 with three sub-tasks T_{11}, T_{12} and T_{13}, (as shown in Figure 3). The lines show the transfer of messages, thus task T_1 can send and receive messages from their sub-tasks; in turn sub-tasks can do the same with the main task and each other.

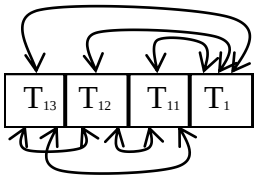


Fig. 3. Message Path between tasks; task with 3 sub-tasks

The functionality of the proposed method is divided into five stages: *a) communication between the scheduler and dispatcher, b) dynamic selection of tasks in queue, c) aptitude evaluation of created solutions, d) generation of new populations, and e) allocation of the best individual to the processor mesh.* The following sections explain each stage, together with the proposed example.

Communication between the planner and the allocator: Once the allocator counts the number of processors available in the mesh, it reports this amount to the planner; using the example from Figure 2, the allocator will inform the planner of 7 available processors. In our example we have shown a case in which all free processors appear totally adjacent, but if they are found to be disjoint, the process that the method follows is the same.

Dynamic selection of tasks waiting in the queue and the dynamic quadratic assignment of tasks to the processor mesh: With the number of available processors on the grid, the scheduler performs the following three steps to the pre-selection of a set of tasks: 1) dynamically selects a task queue using the ROS. In this method, all the tasks have the same probability of selection [20]. 2) Verifies that number of processors required by the task is less than or equal to the number of processors detected by the allocator; if the condition is met, the number of available processors is reduced by the amount of processors required by the task; if the condition is not met, another task will be randomly selected from the queue. 3) Every time that a task is accepted, three checks are made to the prompt completion of tasks: 1) if whether or not the number of available processors is 0, 2) if the stop condition is true, and 3) if all tasks in the queue have been selected at least once. The Random-Order-of-Service policy, allows the tasks that effectively fit the number of available processors in the net to be selected.

In the example, the first task prompt occurs: a random number is generated, based on the number of jobs in the queue which in this case is 4; if the task can be chosen in the sub-mesh or free submeshes, then it will be considered for calculating its allocation and cost of message transfer; if not, it will choose another task. For this exemplification case, the 2 randomly selected tasks are: T_1 and T_2; their placement in the mesh with respective subtasks is shown in Table 5.

TABLE V. LOCATION OF TASKS T_1, T_2 AND T_3 THAT REPRESENT THE FIRST ASSIGNMENT IN THE MESH

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

The second assignment, randomly generates a new allocation in the free sub-mesh corresponding to the assignment of tasks T_3 and T_4 (as shown in Table 6).

TABLE VI. TASK ALLOCATION MATRIX ACCORDING TO THE STATE MATRIX OF THE MESH AT TIME T REPRESENTING A SECOND SOLUTION OF THE QUADRATIC DYNAMIC ALLOCATION PROBLEM

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

Generation of the third allocation shown in Table 7 produces the assignment of tasks T_2 y T_4 to the mesh.

TABLE VII. MATRIX ASSIGNMENTS ACCORDING TO THE STATE MATRIX OF THE MESH AT TIME T, REPRESENTING A THIRD SOLUTION OF THE QUADRATIC DYNAMIC ALLOCATION PROBLEM

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

In the fourth generation, the dynamic task selection produces tasks T_2 and T_3 to the mesh. Produced allocation shown in Table 8.

TABLE VIII. TASK ASSIGNMENT MATRIX ACCORDING TO THE STATE MATRIX OF THE MESH, AT TIME T REPRESENTING A QUARTER SOLUTION OF THE DYNAMIC QUADRATIC ASSIGNMENT PROBLEM

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

Aptitude evaluation of created solutions. Evaluation of created solutions suitability: At this stage the pre-selected set of tasks in the previous step is evaluated with three different objectives: a) the percentage of external fragmentation (*ef*) produced after allocation in order to minimize the number of idle processors in the mesh. For the example case, the first assignment produces an *ef*=0%, the second allocation 0.14%, the third allocation 0.28% and the fourth allocation 0%. b) The number of tasks that the phenotype assigns to the mesh of processors: In the example, the four allocations manage to position two tasks in the processor mesh. c) Communication Overhead or network contention: The allocation cost is calculated for each task, based on communication costs between tasks and the distances between processors, given the message path from one processor to the other and vice versa.

When considering message passing between processors, one must calculate the cost of transference, from the source to the destination and vice versa. In the exemplified case, the transfer rate from task T_1 to subtask T_{11} is different than that from sub-task T_{11} to task T_1, although both measurements can be equal, the values of the distances between processors remain unchanged.

The values to be calculated are given in the operations shown in Table 9 for task T_1, in Table 10 for the task T_2, in Table 11 for the task T_3, and in Table 12 for the task T_4. The total of the respective individuals are summed in order to obtain the total solution cost, with a total of 35 for task T_1 (as shown in Table 9) and 17 for task T_2 (shown in Table 10). The representation of the above calculations is given by equation (1).

TABLE IX. CALCULATION OF MESSAGE TRANSFER COST FOR TASK T_1.

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

TABLE X. CALCULATION OF MESSAGE TRANSFER COST FOR TASK T_2.

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

TABLE XI. CALCULATION OF MESSAGE TRANSFER COST FOR TASK T_3.

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

TABLE XII. CALCULATION OF MESSAGE TRANSFER COST FOR TASK T4.

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

The totals obtained from each calculation, add up to make an individual assessment by the value obtained in the objective functions. This step allows the individuals with the best values in each objective function, to be obtained and selected from the population.

Generation of new populations: Once a population has been obtained, the best individuals iteratively build new populations of individuals from which to extract those that best fit, with these, the probabilistic model is estimated.

Estimating the probabilistic model: in this part we will use the simplest probabilistic model, in which all variables describing the problem are independent. We calculate the frequency of task occurrence from a part of the population, containing the best individuals in each empty cell of the mesh at time *t*, through truncation selection along with the percentage of the truncation. In this case, the frequency of occurrence can be shown in Table 13, due to space constraints, only the frequencies for processor 0 are shown.

TABLE XIII. FREQUENCY OF OCCURRENCE OF EACH TASK IN EACH CELL.

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

Allocation of the best individual to the processor mesh (Determination of the best individual). This step shows the task or tasks that produce the best allocation representing the most feasible solution, and which is assigned to the mesh.

V. EXPERIMENTS

In this section we explain the experiments conducted with the proposed method, against those of the strict FCFS allocation policy; most of the proposed works use this policy (FCFS) during task planning. At the end of the workload execution in the waiting queue, the five criteria components that are sought to be optimized, in multiprocessor systems are evaluated: utilization, throughput, mean turnaround time, waiting time and total execution time. The parameters that the algorithm uses for its normal operation, that do not need to be optimized are:

1) The size of the 2D mesh: This sets the size of the mesh and therefore the number of processors on the target system.

2) Number of tasks: the total number of tasks that the system processes also called the overall system load.

3) Number of subtasks for each task.

4) Time of execution for each task: the parameter that defines the number of seconds, the task will remain within the mesh, constituted by the sum of seconds of each of the subtasks that make up the task.

5) Capacity of the queue: the number of tasks that the waiting queue accepts to be processed, and the number of subtasks that each task may contain.

6) Number of tasks that the system will seek to enter into the waiting queue: defined as the number of tasks that the algorithm searches for, in the waiting queue using the ROS planning method. The number of tasks is determined by the conditions of the stopping algorithm, whether or not, the tasks waiting in the queue have been selected at least once, or if the number of processors available at time t was already covered.

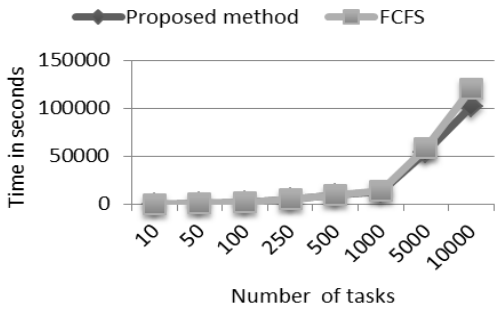
7) Number of phenotypes or individuals per population that will be created: the parameter that defines the number of individuals, within each one of the populations that constructs the algorithm to determine the best individual (set of tasks assigned to the mesh of processors).

8) Number of Populations, that will be created: defined as the number of stocks, that the system generates to extract the best individuals and estimate the probabilistic model.

These are the normal operating parameters of the algorithm. The execution of tests was carried out in the cluster of Liebres InTELigentes servers, consisting of four servers: HP Proliant Quad core with the Linux operating system.

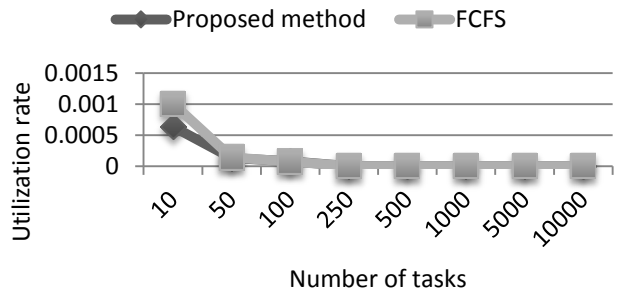
Experiment 1: For the first experiment an 8X8 mesh is used with different queue capabilities: from 10 to 10.000 tasks (as shown in Charts 1 and 5). The number of subtasks per task for this experiment was set 1 to 10 (light load). The task execution time is 1 to 100 seconds. The number of tasks that the system will seek into the waiting queue, once free submeshes are produced in the mesh, is dynamic, and corresponds to the stopping method set in the algorithm as well as the number of phenotypes.

Total execution time: This approach is shown in graph 1. The behavior of both methods, when loads are light, is very similar in this approach. FCFS is a policy free of starvation and non-discriminatory in nature, allowing the task at the head of the queue waiting to be served, once the number of solicited processors is released into the mesh; due to the fact that with light loads, a large number of processors are not required and requests can be quickly met.



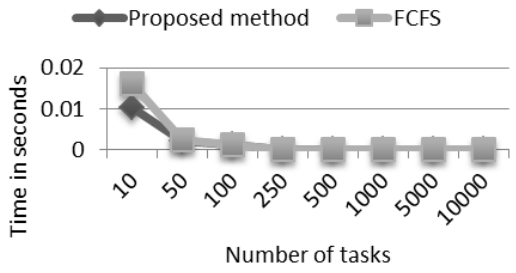
Graph 1. Total execution time

Utilization: For this experiment, system utilization is measured by each total workload that the system processes. When comparing system utilization, the behavior of both methods is practically the same as illustrated in graph 2. A previous allocation planning is not synonymous with better system utilization when light loads are processed.



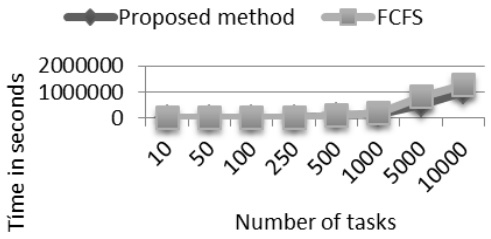
Graph 2. Utilization

Throughput: Due to the acceleration that occurs in the allocation, the number of completed tasks per unit of time in the system (when the FIFO allocation policy is used), produces times very similar to the proposed method. That is, the generation of a set of tasks with a small amount of subtasks, upon finalizing execution, enables new tasks to be entered into the mesh without waiting until a large number of processors have been released. The results of this test appear in graph 3.



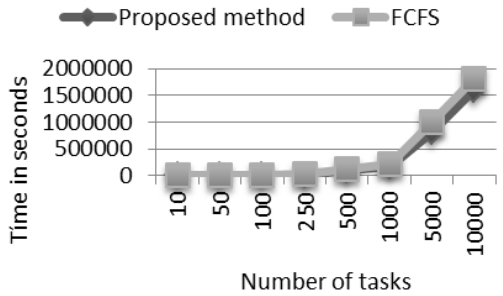
Graph 3. Throughput

Mean turnaround time: graph 4 shows how the proposed policy exceeds the FIFO method, when there is an increase of more than 250 tasks in the system load. The significant improvement in time is produced by the response factor, to the task that is at the head of the queue.



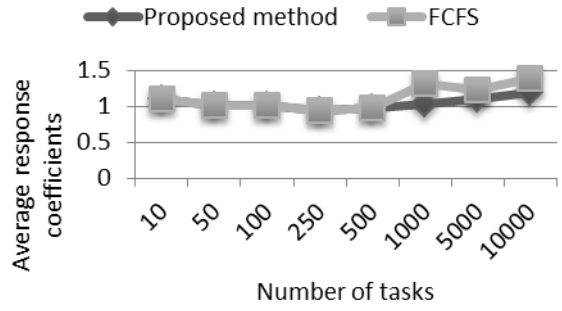
Graph 4. Mean turnaround time

Waiting time: The average waiting time of tasks before starting its execution, is significantly improved with the proposed method when the number of tasks in the waiting queue increases (as shown in graph 5). It is assumed that, the improvement in times of this criterion is due, to the utilized ROS planning that does not consider the immediate assignment of the task that is in the head of the waiting queue, but the task search that best suits the free processors.



Graph 5. Waiting time

Response ratio: System performance remains constant in both methods when the number of tasks is less than or equal to 500, and varies when loads are increased in the waiting queue (as shown in graph 6). The system performance is considered an important criterion in multiprocessor systems, because it shows the constant and proper use of resources in the target system, or in certain cases, processor waste generated in the target system.

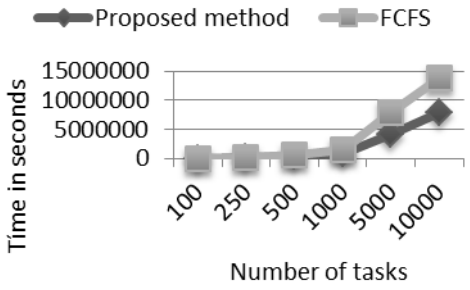


Graph 6. Response ratio

Experiment 2: Graphs 7 to 12 show the results of the second experiment. In this experiment, the number of subtasks per task is significantly increased, but the creation of tasks with few subtasks is also allowed. The objective of this experiment is to have a mixture of tasks: tasks with many processor requirements and tasks with little processor requirements. This

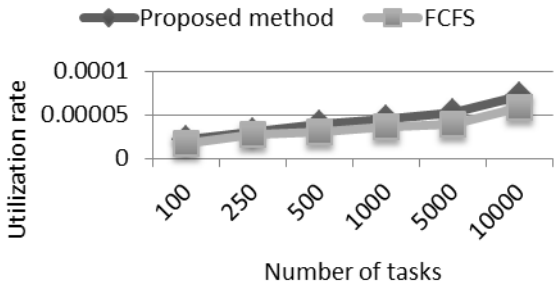
mix allows us to produce external fragmentation of the mesh, and observe the behavior of the algorithm in order to solve the dynamic quadratic assignment problem in the mesh.

Total execution time: When the proposed method plans a large quantity of tasks in the waiting queue, you can choose randomly from a variety of task requirements, causing the total execution time to be reduced drastically (as shown in graph 7).



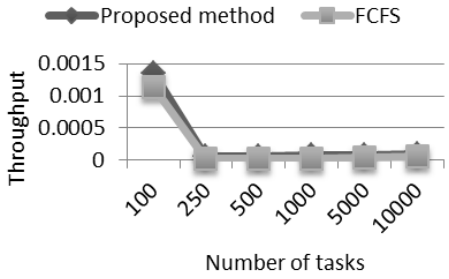
Graph 7. Total execution time

Utilization: A better percentage of system utilization is reflected in graph 8, upon using the proposed method due to the fact that in each assignment, all total free processors are assigned to tasks (external fragmentation is decreased). When using a high percentage of processors, network latency increases exponentially because of message passing between tasks.



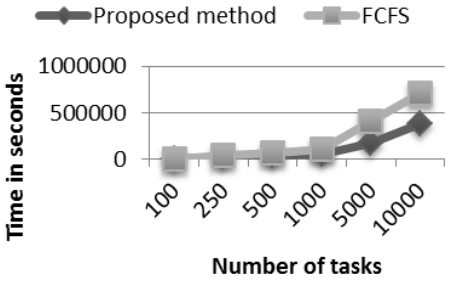
Graph 8. Utilization

Throughput: Although the proposed method outperforms the FCFS policy in number of completed tasks per unit time, both provide similar behavior when there is an increase in the number of subtasks per task (as shown in graph 9).



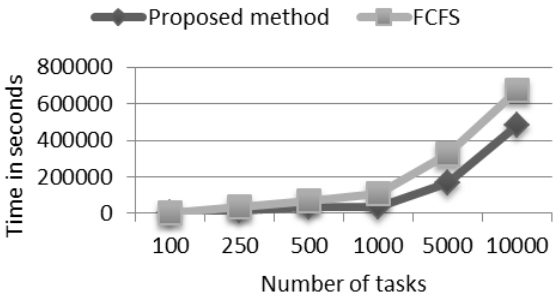
Graph 9. Throughput

Mean turnaround time: graph 10 shows the results of mean turnaround time; the proposed method provides shorter times in responses to tasks with a heavy workload.



Graph 10. Mean turnaround time

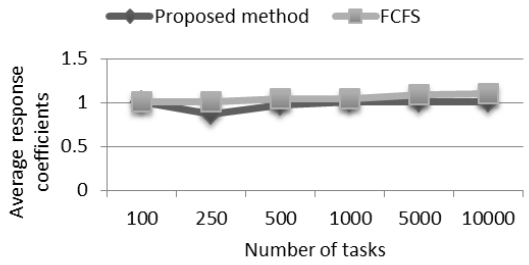
Waiting time: The observation in this approach (shown in graph 11), is the reduction in the waiting time of tasks in the waiting queue with the proposed method; the FCFS offers longer waiting times for tasks. Here we observe, the wait that is generated for the tasks with large requirements in the waiting queue; these tasks, must wait until the number of free processors required, to achieve their entry into the mesh are available.



Graph 11.Waiting time

Another important factor is the external fragmentation that occurs when the task at the head of the waiting queue is assigned, and the next task at the head of waiting queue can no longer be allocated due to the number of remaining free processors after the assignment (this being different from the number of processors that is required). Unlike the aforementioned, the proposed method does not assign tasks sequentially in the waiting queue but randomly looks for tasks that best suit the free processors (thereby minimizing task waiting times).

Response ratio: As a consequence of the obtained results in the waiting time criterion, system utilization is significantly improved producing better results in the response rate criterion with the proposed method (as shown in graph 12). Achieving a maximum utilization of free processors through planning, every time an assignment is made, yields better results in system utilization.



Graph 12. Response ratio

The goal of both experiments was to show the behavior comparison of the FCFS scheduling policy, which is the most widely, used policy in experiments carried out with the proposed task planning methods in multiprocessor systems. The results show variations in response times of both methods. The main objective of the proposed method is to achieve a pre-planning to the allocation through the optimization of 3 targets, that at the end of workload processing achieves improvement in the established criterion for its evaluation.

VI. Future Works

Considered the basis of this work, future research that arises, is a parallel evaluation of the objectives that are opposed in the planning and allocation of tasks, in a multiprocessor system using multi-core programming. This research is being carried out in a server cluster.

VII. Conclusions

This paper, through the joint operation of the task planner and the processor dispatcher considering all the parameters involved in task planning and allocation, presents a strategy that yields a more efficient use of computing resources in a multiprocessor system.

The main objective of this research, is to achieve a pre-planning to the allocation that considers the evaluation of three objective functions that lead to obtaining, a structured assignment avoiding a unique planning of task lists, based on genetic operators or based on the exhaustive search of free submeshes, using geometric models; the proposed method uses the planning policy ROS, whose random behavior allows all tasks to have the same probability of selection, each time the scheduler selects a set of tasks to be assigned to the mesh.

Similarly, the method looks for the best position of the tasks in the processors using a dynamic quadratic assignment, which is determined by estimating and simulating the joint probability distribution, as a mechanism of evolution in order to reduce communication overhead in mesh processors.

The experiments carried out in our work, use the FCFS against the method proposed in this paper. What happens when a planning task with a strict FCFS policy is compared against a totally random policy?

The set of conducted experiments, show the results of the five criteria that are evaluated in multiprocessor systems upon finalizing total execution of workloads, unlike other researches that only seek to optimize a single evaluated criterion. When system loads are light, both planning policies have a similar behavior, and they manage to locate tasks quickly enough in the mesh, but upon increasing the processor requirements with a larger number of subtasks per task, the proposed method has better results in the following evaluated criteria: utilization, throughput, mean turnaround time, waiting time and the total execution time.

The positivity of the proposed method lies in three key areas: 1) that all tasks have the same probability to be served once a set of tasks are selected for assignment, 2) actively maintain a noncontiguous allocation strategy, which allows it to confront the dynamic quadratic assignment for positioning tasks on processors and 3) avoid producing communication overhead in the processor mesh.

References

[1] Grama A., Gupta A., Karypis G., and Kumar V. Introduction to Parallel Computing. Second Edition Addison Wesley, January 16, 2003 ISBN: 0-201-64865-2

[2] Ababneh I., Bani-Mohammad S. “A new window-based job scheduling scheme for 2D mesh Multicomputers”. Simulation Modeling Practice and Theory 19 (2011) Pp. 482-493

[3] Ahmad S. E. “Processors Allocation with Reduced Internal and External Fragmentation in 2D Mesh-based Multicomputers”. Journal of Applied Science 11 (6) 943-952, 2011 ISSN 1812-5654 2011 Asian Network for Scientific Information

[4] Adiga NR., Almasi G., Almasi GS., Aridor Y., et al. “An Overview of the BlueGene/L Supercomputer”. Team IBM and Lawrence Livermore National Laboratory

[5] Bokhari. S. H. “Communication Overhead on the Intel PARAGON, IBM SP2 & MEIKO CS-2”. In http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19960004071.pdf

[6] Hussain H. Ur Rehman S., Hammed A., Ullah S., et al. “A survey on resource allocation in high performance distributed computing system”. Parallel Computing System & Applications 39 ELSEVIER Pp. 709-736. 2013

[7] Leung J. Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 1st ed., CRC Press Inc., Boca Raton, FL, USA, 2004

[8] Walker P., Bundle D., and V. Leung. “Faster high-quality processor allocation”. In Proceedings of the 11th LCI International Conference on High-Performance Clustered Computing 2010

[9] Yoo S., Youn H., and Choo H. “Dynamic Scheduling and Allocation in Two-Dimensional Mesh-Connected Multicomputers for Real-Time Tasks”. IEICE TRANS. INF & SYST., VOL. E84-D, No. 5 May 2001

[10] Chuang P., and Tzeng N. “Allocation precise submesh in mesh connected systems”. IEEE Transactions on Parallel and Distributed Systems 5(2) (1994) Pp. 211-217

[11] Chuang P. and Tzeng N. “An Efficient Submesh Allocation Strategy for Mesh Computer Systems”, Proc. Int’l Conf. on Distributed Computing Systems, May 1991, Pp.256-263

[12] Liu T., Hwang K., Lombardi F., and Bhuyan L. “A Submesh Allocation Scheme for Mesh-Connected Multiprocessor System”, Proc. 1995 Int’l Conf. Parallel Processing, vol II, 1995 Pp. 159-163

[13] Ababneh I. “On submesh allocation for 2D mesh multicomputers using the free-list approach: Global placement schemes”. Performance Evaluation 66(2009) ELSEVIER Pp. 105-120

[14] Velarde A., Ponce de Leon E., and Diaz E. “Planning and Allocation Tasks in a Multicomputer System as a Multi-objective Problem”. Advances in Intelligent Systems and Computing 227. EVOLVE 2013. International Conference Held at Leiden University, July 10-13, 2013. Leiden, The Netherlands. Springer

[15] Feitelson D., Rudolph L., Schwiegelshohn U., Sevcik K., and Wong P. Theory and Practice in Parallel Job Scheduling. Report in http://www.cs.huji.ac.il/~feit/parsched/jsspp97/p-97-1.pdf

[16] Hou E., Ansari N. and Ren H. “A Genetic Algorithm for Multiprocessor Scheduling”. IEEE Transactions On Parallel and Distributed Systems, Vol 5, No. 2, February 1994 Pp. 113-120

[17] Deb K. Multi-objective Optimization Using Evolutionary Algorithm. Wiley 2001.

[18] Zitzler E., Laumanns M., and Bleuler S. A Tutorial on Evolutionary Multiobjective Optimization In Metaheuristics for Multiobjective Optimization. 2003. In http://citeseerx.ist.psu.edu/viewdoc/

[19] Sinnen O. Tasks Scheduling for Parallel Systems. Wiley Series. 2007

[20] Rogiest W., Laevens K., Walraevens J., and Bruneel H. "When Random-Order-of-Service outperforms First-Come-First-Served". Operations Research Letters 43. Elsevier. (2015) Pp. 504-506

[21] Goldberg D. E. "Genetic Algorithms in Search, Optimization and Machine Learning". Addisson –Wesley, Reading, MA, 1989

[22] Kwok Y. K. and Ahmad I. "Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm". Journal of Parallel and Distributed Computing 47, (1997) Pp. 58-77

[23] Bohler M., Moore F., and Pan Y. "Improved Multiprocessor Task Scheduling Using Genetic Algorithms" MAICS-99 Proceedings. (1999)

[24] Hwang R., Gen M., and Katayama H. "A Performance Evaluation of Multiprocessor Scheduling with Genetic Algorithm". Asia Pacific Management Review 11(2) 2006. Pp 67-72.

[25] Mohamed M. and Awadalla M. "Hybrid Algorithm for Multiprocessor Task Scheduling". International Journal of Computer Science Issues, Vol 8, Issue 3, No. 2, May 2011

[26] Larrañaga P. "A Review on Estimation of Distribution Algorithms". Estimation of Distribution Algorithms Vol. 2, Kluwer Academic Publishers. 2002. Pp. 57-100

[27] Das D. and Pradhan D. K. "Job Scheduling in Mesh Multicomputers". IEEE Transactions on Parallel and Distributed Systems, VOL. 9, NO. 1, JANUARY 1998

[28] Bani-Mohamad S., Ould-Khaoua M., I Ababneh., and Mackenzie L. "Non-contiguous Processor Allocation Strategy for 2D Mesh Connected Multicomputers based on Sub-meshes Available for Allocation". Proceeding of the 12th International Conference on Parallel and Distributed Systems (ICPADS'06) 2006 IEEE

[29] Bani A. S., "Submesh Allocation in 2D Mesh multicomputers: Partitioning at the Longest Dimension of Request". The International Arab Journal of Information Technology, Vol. 10, No.3, May 2013. Pp. 245 252.

[30] Y. Zhu. "Efficient processor allocation strategies for mesh-connected parallel computers". Journal of Parallel and Distributed Computing. ELSEVIER. Volume 16, Issue 4, December 1992, Pages 328-337

[31] G. Kim. "On submesh allocation for mesh multicomputers: a best-fit allocation and a virtual submesh allocation for faulty meshes". Parallel and Distributed Systems, IEEE Transactions on. Volume: 9, Issue: 2. 1998. Pp. 175-185.

[32] Velarde A."Objective Analysis in Task Planning and Allocation of Multicomputer Systems". RCS Research in Computer Science. Vol 104. ISSN 1870-4069. (2015) Pp. 23-39