International Journal of Distributed and Cloud Computing
9 (1) 2021, 36-48
http://www.publishingindia.com/ijdcc

Processor Allocation Two-Dimensional Mesh-Based Multi-Computer Systems: Protecting against Over-Partitioning and Maintaining Certain System Utilization Level

Sulieman A. Bani-Ahmad
Professor of Data Science, Department of Intelligent Systems, Faculty of Artificial Intelligence Al-Balqa Applied University, Jordan. Email: sulieman@bau.edu.jo

Abstract: Gradual Request-Partitioning-Based (GRPB) processor allocation strategies try to remedy the problem internal fragmentation by having requests get be allocated non-contiguously in the form of gradually produced blocks in case contiguous allocation is not possible. In this article, we experimentally show that GRPB techniques suffer from the problem of over-partitioning, and thus, negatively affects key performance indicators of multi-computer systems. This paper also proposes a framework for using fuzzy logic to control specific key performance levels in two-dimensional mesh-based multi-computer systems. We demonstrated that it is possible to implement a fuzzy-based feedback control system to control a number of performance indicators in the parallel system. The proposed allocation scheme prevents over-splitting of parallel jobs by forcefully limit to the maximum number of blocks that can be assigned to any parallel job. This maximum number is referred to as the partitioning-bound. Preventing parallel processes from getting over-partitioned is vitally important as this (i) reduces the distance of inter-processor source-to-destination communication and (ii) helps in avoiding the "inter-process interference"; the main cause of communication contention. Fuzzy-based allocation provides a mechanism to dynamically control the partitioning-bound. Instead of pre-setting and having a fixed partitioning-bound level, we provide a way to set this value at run time taking into consideration the current status to multi-computer system.

Keywords: Fuzzy-control, Gradual request-partitioning, Internal fragmentation, Multi-computer systems, Non-contiguous processor allocation.

I. Introduction

Two-Dimensional Mesh-based Multi-computer consists of a set of processing elements connected through a 2D mesh-like "interconnection network". In a typical mesh interconnection network the processing elements are arranged as the nodes of a two-dimensional grid. In this grid, direct point-to-point links exist connecting each node to its adjacent nodes. Mesh multi-computers are suitable for computer problems that require parallel computing. This involves subdividing large computer problems into smaller ones, each executing in a different process. In such case, the execution of processes is carried out simultaneously each on a dedicated set of processing elements. Examples of such computer applications are matrix computations, image processing and problems [1, 2].

Processor allocation in a multi-computer system is concerned with the dedication of the requested frame of processing elements for a specific computer job in hand. Three categories of processor allocation schemes are studies in literature, those are; contiguous, non-contiguous and mixed.

One issue with contiguous processor allocation strategies in 2D mesh-based multi-computer systems is its low system utilization as they fail to eliminate the effect of external and internal fragmentation. Hence, these strategies provide noticeably low and limited performance, mainly; system utilization and response time [3, 4, 5].

In non-contiguous processor allocation, jobs that fail to be allocated in a contiguous manner are subdivided into smaller sub-blocks that are individually allocated in non-contiguous manner instead of waiting until a submesh of the requested size and orientation is available in the system. In practice, lifting the condition of contiguity helps reducing processor fragmentation, and pre-execution waiting delay, and significantly increases system utilization. However, this non-contiguity results in increasing the overall communication overhead due to message contention increase. The Contention is the situation where multiple processes attempt to transmit a message across one shared link simultaneously. The extra communication overhead load depends on how dispersed is a partitioned parallel request (referred to as the degree of non-contiguity) [5, 1].

Gradual Request-Partitioning-Based (GRPB) processor allocation strategies try to tackle this problem on non-contiguity by gradually partitioning job requests and non-contiguously allocating them in the form of blocks (in case contiguous allocation is not possible). Studies demonstrate that GRPB allocation strategies demonstrate having the advantages of both contiguous allocation (e.g., low communication latency) and non-contiguous allocation strategies (e.g., high system utilization and throughput). This is achieved through preserving a *relatively acceptable* level of contiguity between allocated blocks of a parallel job.

Two groups of GRPB allocation strategies are studies in literature; (i) the "Adaptive Non-Contiguous Allocation" (ANCA) and (ii) the "Bounded-Gradual-Partitioning" (BGP) allocation. Those allocation schemes attempt to tackle the problem of internal and external fragmentation through allowing non-contiguity allocated submeshes [6, 7]. In ANCA, for instance, this is done by splitting the requested frame into two sub-frames of *equal sizes* (if possible). Splitting in this scheme is best applied to the longest dimension of the two dimensions of the request. In BGP, however, this is achieved by recursive partitioning the requested frame into two sub-frames: one is *large* and the other is *small*. BGP scheme prevents over-splitting of parallel jobs by limiting the maximum job dispersal represented by preventing the number of non-contiguous blocks assigned to any served parallel job from exceeding specific maximum value. This maximum value is referred to as the *partitioning-bound*.

Preventing parallel processes from getting over-partitioned is vitally important in shortening communication source-to-destination path and in avoiding the inter-process interference. In this work, we aim at providing mechanisms to control the partitioning-bound. Instead of pre-setting and having a fixed partitioning-bound level, we intend to provide a way to set this value at run time taking into consideration the current status to multi-computer system. For example, one way is to consider the current (or cumulative) system utilization to dynamically and automatically decide this value.

Consider the following 2D mesh multi-computer in Fig. 1. In this figure, the multi-computer consists of a mesh of size 5 x 6; a total of 30 processing units. Black nodes represent busy elements (already allocated and occupied). White nodes are available ones. Assume further that a process of size 2 x 2 has arrived and is waiting to be served. Under this scenario, contiguous processor allocation strategies fail to allocate this job despite the fact that enough free-processes are available in the system. BGP allocation strategies lift the contiguity condition and allocate multiple blocks. In this case, two blocks of size 1 x 2 each can be successfully allocated.

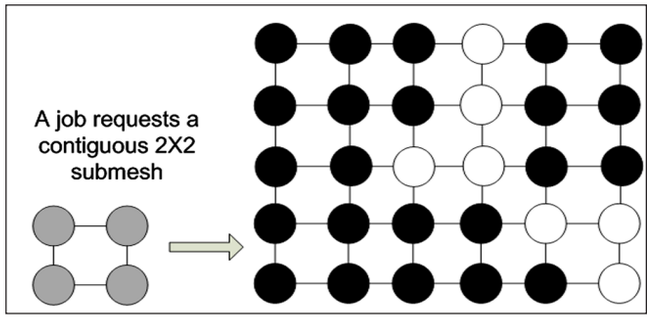

Fig. 1: A Multi-Computer System with 5 x 6 2D Mesh

One issue with BGP allocation strategies is that they may aggressively partition or over-partition parallel jobs by assigning them many sub-blocks. Preventing this phenomenon is important as it reduces the degree of non-contiguity. As a result, the distance of inter-processor communication path becomes shorter and the inter-process interference that creates communication contention reduces.

Our goal in this study is to provide mechanisms to control the partitioning-bound. Instead of pre-setting the partitioning-bound level, we would like to decide this value at run time. The proposed mechanisms take into consideration the current (or cumulative) system utilization and probably other performance metrics to decide this value.

The rest of this paper is organised as follows: Literature review is presented in Section II. The proposed allocation strategy is explained in Section III. In Section IV and V the experimental settings and results are presented. Section V compares the proposed approach to the already presented work in literature. Section VI is the conclusion.

II. Literature Review

A. Contiguous Processor Allocation

Contiguous allocation strategies work by allocating a single and contiguous block of idle processing elements for the execution of the parallel job in hand [8, 3, 4, 5, 9, 10, 11]. In fact, contiguity of processing elements minimizes the source-to-destination distance of inter-processor communication paths and, thus, in tackles the phenomenon of inter-process interference that causes "communication contention". Further, contiguity help reduce the power consumption of multiprocessor architectures, e.g., 2D mesh multi-computers [12, 13].

Other advantages of this category of allocation strategies are:

- Allocated subframes preserve the same exact shape (i.e., topology) as the underlying multi-computer system.

- The number of multi-computers allocated to a particular parallel job exactly matches that parallel job request with no internal fragmentation [14]. Thus, in mesh-connected multi-computers, parallel jobs are allocated to submeshes [14, 3, 4, 5, 9, 10].

Next is a list of well-known contiguous processor allocation strategies:

Two-Dimensional Buddy: In this allocation mechanism, the parallel system is assumed to be square-shaped with the two side-lengths be equal to a power of two (e.g., 2^i). In this case, the size of incoming parallel requests is rounded-up to a square-shape with side-lengths as the nearest power of two. Consequently, this allocation mechanism suffers from the problem of internal fragmentation. Rounding up the sides of parallel requests results in allocating submeshes larger than the actually requested submeshes [9].

In [10], the Frame Sliding (FS) allocation mechanism is proposed to solve the problem of internal fragmentation. This is achieved by allowing parallel requests of any arbitrary shape and size to be allocated. That is; parallel requests are viewed as frames of rectangular shapes. The Frame Sliding algorithm works by sliding each of the requested frames across the system's 2-D or 3-D mesh to examine for an unallocated submesh in the system [10].

In [11], the author has proposed The Best-Fit (BF) and First-Fit (FF) the allocation schemes in order to guarantee the recognition of a unallocated sub-meshes taking into consideration the orientation factor. Each of the 2 approaches works by scanning the entire system's mesh. Identified and allocated submeshes follow the same size, shape, and orientation of the requested frame. In case a frame with the same shape and size exists in the parallel system but the orientation is different, these two strategies fail. To solve this issue, the Adaptive-Scan (AS) scheme is proposed in [15]. The adaptive-scan changes solve this problem by rotating and changing the orientation of the requested submesh being searched for in case the required sub-mesh with the requested orientation is not available.

B. Non-Contiguous Processor Allocation

Studies show that allocation schemes with high *recognition ability* for idle sub-meshes can significantly improve the probability of successfully assigning parallel requests into the system and, consequently, reduce the job pre-execution waiting delay [16].

Further performance-related studies show that performance of already existing contiguous allocation strategies a significant improvement cannot be improved [4, 14, 17]. In practice, the system utilisation performance indicator can significantly degrade because of external fragmentation problem [4, 14]. External fragmentation occurs when parallel jobs fail to be served while enough unallocated processing elements in the parallel system exist.

Hardware advances in 2D-mesh interconnection network, such as the use of wormhole routing technique and the availability of high-performance switching techniques, have resulted in decreasing the communication latency and in making this parameter be less dependent on the distance between the communicating nodes [4, 7, 14]). This has resulted in making non-contiguous allocation plausible in parallel systems. By elimination of the restriction of contiguity in processor allocation, the pre-execution waiting time becomes significantly less. The "Multiple Buddy System" (MBS) and the Paging allocation strategies are examples of non-contiguous allocation algorithms proposed in the literature.

In the MBS strategy introduced in [18], the 2-D mesh of the parallel system is divided into a set of disjoint sub-meshes of square-shape with widths and heights are of powers of two. The MBS algorithm works as follows: the number of required processors is factorized into base-4blocks. In case the requested frame is not available, MBS recursively breaks-down large requests into 4blocks until it produces sub-blocks of the desired overall original size of the parallel job.

In the Paging algorithm that is introduced in [19], the entire system's mesh is sub-divided into "virtual" pages in the form of square frames of side-lengths of 2^i, where i is a parameter referred to as the index parameter. The index parameter is a pre-assigned positive integer.

C. Request-Partitioning-Based (RPB) Allocation Schemes

Partially non-contiguous allocation strategies out-perform totally non-contiguous strategies in terms of jobs dispersal [6, 19]. In Paging algorithm, for example, a reasonable degree of contiguity is gained depending on the indexing scheme used. To illustrate, job dispersal is less when decreasing the index parameter. Increasing the index parameter, however, produces higher internal processor fragmentation [19].

Request-Partitioning-Based allocation reduces the problems of (i) internal and external fragmentation, (ii) low average system utilization (ASU), and (ii) low system throughput. This is achieved by permitting parallel requests to be gradually partitioned and allocated in a non-contiguous manner in case contiguous allocation is not successful [20]. Notice that, in general, the probability of successfully allocating small sub-frames is high.

In literature, two groups of RPB allocation schemes can be found; (i) the "Adaptive Non-Contiguous Allocation" (ANCA) [21] and the (ii) Bounded-Gradual-Partitioning (BGP) allocation [16, 22].

The ANCA algorithm works as follows: First, the algorithm attempts to do contiguous allocation, if this step fails, allocation is achieved by recursively splitting the requested frames into two sub-frames of equal sizes (roughly). Splitting is performed at the longest dimension of the two dimensions; the width and the height [14, 6].

In BGP, however, allocation of parallel jobs is achieved by gradually splitting the original frame into two sub-frames; one large and another small sub-frame of processing elements ([20]0a; [20]0b).

In [2], the authors proposes the Minimum Interference Paging (MIP) non-contiguous processor allocation strategy which attempts to reduce the distances among processors allocated using a paging variant that chooses a set of processors with the lowest distance between the first and last allocated processors, where the distance is the number of processors between the first allocated node and last allocated node. MIP is comparatively evaluated with Paging (0), Multiple Buddy System (MBS) and Adaptive Non-contiguous Allocation (ANCA). The results show that MIP exhibits superior performance in terms of average turnaround time of jobs.

In [21], the author suggests a new noncontiguous allocation strategy called All Request Shapes Greedy Available Busy List (ARSGABL for short). ARSGABL considers all possible request shapes when attempting allocation for a job request experimental simulation-based results confirm that the ARSGABL strategy improves system performance in terms of the average turnaround times of jobs.

D. Unbounded vs Bounded Gradual Request Partitioning (GRP)

GRP-based allocation schemes partitions requests at the longest dimension. This approach works as follows: first, it attempts to find a single and idle submesh using any contiguous allocation strategy (First-Fit or Best-Fit for instance). If contiguous allocation fails and enough number of free processing elements is available in the system, the request is splitted into two sub-frames that are allocated based on the same contiguous allocation approach mentioned above. These steps are repeated until the job request is satisfied. The two new sub-requests are reclusively served until success (the best-fit algorithm is used in the experimental part of this research). This recursive procedure continues until the current job's request is fulfilled. Notice that this approach places *no limit to* the job dispersal.

A pseudo code for the "unbounded GRP-based allocation strategy" is sketched in Fig. 2. The key idea in here is to try to allocate the largest possible contiguous sub-meshes.

```
Procedure GRP-BF(a, b):
Begin
    JobSize = a × b
    If (number of free processors < JobSize) Return failure
    List AllocatedPIDs={}; // PIDs allocate to the job
    Return GRP-BFAllocate (a, b, &AllocatedPIDs);
End

Procedure GRP-BFAllocate (a, b, AllocatedPIDs)
Begin
    S(x, y) = FIND_BF (S(a, b);
    If (S(x, y)  != null)
    {
        Add the PIDs of S to the list AllocatedPIDs;
        Return success;
    }
    Else
    {
        If(a>=b)
            α1= a-1;  β1=b; α2= 1;  β2=b;
        else
            α1= a;  β1=b-1; α2= a;  β2=1;
            Return GRP-BFAllocate (α1, β1, &AllocatedPIDs);
            Return GRP-BFAllocate (α2, β2, &AllocatedPIDs);
    }
End
```

Fig. 2: Pseudo Code for the Unbounded GRP-BF Allocation Strategy

Experiments show that "unbounded GRP-based" allocation strategies may cause jobs get highly dispersed due to *over-partitioning* depending on the temporal state of the multi-computer system mesh. Theoretically, a request for n processors can be partitioned into n blocks of size one processing element each! The negative effect of this phenomenon appears more at heavy intra-process communication loads [VE03]. To remedy this drawback, the *Bounded* GRP-BF is proposed in [20, 22]. This processor allocation algorithm is sketched in Fig. 3. The key difference between Fig. 2 and Fig. 3 is that we put an upper bound on the "number of blocks" into which requests can be subdivided into. This helps prevention of over-partitioning requests.

```
BlockCount; // block count for the current job request
MaxBlockCount; // maximum allowed blocks per job
Procedure BGRP-BF(a, b):
Begin
    JobSize = a × b
    If (number of free processors < JobSize) Return failure
    List AllocatedPIDs={}; // PIDs allocate to the job
    BlockCount=0;
    Return BGRP-BFAllocate(a, b, &AllocatedPIDs);
End

Procedure BGRP-BFAllocate (a, b, AllocatedPIDs)
Begin
    S(x, y) = FIND_BF (S(a, b);
    If (S(x, y)  != null)
    {
    Add the PIDs of S to the list AllocatedPIDs;
    BlockCount= BlockCount+1;
    Return success;
    }
    Else
    {
        If (BlockCount > MaxBlockCount) Return failure
        If(a>=b)
            α1= a-1;  β1=b; α2= 1;  β2=b;
        else
            α1= a;  β1=b-1; α2= a;  β2=1;
            Return BGRP-BFAllocate (α1, β1, &AllocatedPIDs);
            Return BGRP-BFAllocate (α2, β2, &AllocatedPIDs);
    }
End
```

Fig. 3: Pseudo Code for the Bounded GRP-BF Allocation Strategy

III. Proposed Allocation Strategy: Bounded Gradual Request Partitioning with Controlled Job Dispersal

One observation on the pseudo code presented in Fig. 3 is that the upper bound *MaxBlockCount* is preset in the code. The user decides upon this value and the value remains unchanged regardless of performance indicators of the multi-computer system. In this work, we propose mechanisms that help dynamically decide upon this parameter in order to maintain specific targeted system performance level(s). For that, a new procedure is added to the pseudo code of Fig. 3. Fig. 4 represents the pseudo code with one more procedure added to the strategy, that is the Refresh MaxBlockCount() procedure. This procedure is responsible for calculating the new value of the *MaxBlockCount* variable. The new value of this variable should take into consideration the current status of the multi-computer system by calculating some key performance indicators that will be presented shortly. This procedure should be invoked after each successful allocation decision.

```
1  BlockCount; // block count for the current job
2  MaxBlockCount; // maximum allowed blocks per job
3
4  Procedure RefreshMaxBlockCount()
5  Begin
6      . . .
7  End
8
9  Procedure BGRP-BF(a, b):
10 Begin
11     JobSize = a × b
12     If (number of free processors < JobSize) Return failure
13     List AllocatedPIDs={}; // PIDs allocate to the job
14     BlockCount=0;
15     Return BGRP-BFAllocate(a, b, &AllocatedPIDs);
16 End
17
18 Procedure BGRP-BFAllocate (a, b, AllocatedPIDs)
19 Begin
20     S(x, y) = FIND_BF (S(a, b);
21     If (S(x, y)  != null)
22     {
23         Add the PIDs of S to the list AllocatedPIDs;
24         BlockCount= BlockCount+1;
25         RefreshMaxBlockCount();
26         Return success;
27     }
28     Else
29     {
30         If (BlockCount > MaxBlockCount) Return failure
31         If(a>=b)
32             α1= a-1;  β1=b; α2= 1;  β2=b;
33         else
34             α1= a;  β1=b-1; α2= a;  β2=1;
35             Return BGRP-BFAllocate (α1, β1, &AllocatedPIDs);
36             Return BGRP-BFAllocate (α2, β2, &AllocatedPIDs);
37     }
38 End
```

Fig. 4: Pseudo Code for the Bounded GRP-BF Allocation Strategy with the MaxBlockCount Parameter Automatically Determined and Calculated at Run Time

A. Multi-Computer System's Key Performance Indicators

The following is a list of performance indicators that we may intend to control in the multi-computer system. Those indicators should contribute to our decision of determining the value of the MaxBlockCount parameter in the allocation strategy code.

- *"Mean Response Time" (MRT):* RT is the time elapsed from the arrival of a job until the first real output produced from that job.
- *"Mean Job Service Time" (MJST):* The ST of a job is the time elapsed from the successful allocation of the job until the moment the job releases the system resources after it finishes.
- *(Mean or Temporal) "Percent-System-Utilization" (PSU):* Percent-system-utilization of a multi-computer system is defined as the percentage of the busy or allocated processors within a system, this value between 0 and 1 (normalized by dividing by the system's mesh size).
- *"Mean-Packet-Blocking-Time" (MPBT):* For a specific message, this performance indicator is defined as the total time the head of the communication messages (using wormhole routing) is blocked at each and every station over the source-to-destination path.
- *"Mean Packet Latency" (MPL):* In wormhole routing, each message is subdivided into a number of packets. MPL for a specific message is defined as the average of the time that all packets within a parallel job will be sent between processors (From source processor to distention processor).
- *Mean-Number-of-Blocks-Per-Job (MBPJ):* MBPJ is defined as the number of disjoint sub-frames assigned to the parallel jobs that were served since the system is put into service.

Next is the nature of relationship between the *MaxBlockCount* parameter in Fig. 3 and each of the six performance indicators listed before.

Table I: The Nature of Relationship between the MaxBlockCount Parameter and Keyparallel System Performance Indicators

Performance Indicator	Relationship with the *MaxBlockCount* Parameter
MRT	Indirect
MJST	Indirect
PSU	Direct
MPBT	Direct
MPL	Direct
MBPJ	Direct

Notice that, in general, as the *MaxBlockCount* parameter increases the response time and service time of parallel jobs decrease. The reason is that the pre-execution waiting time of jobs significantly decreases because they get allocated early due to lifting the condition of contiguity.

Comparatively, increasing the *MaxBlockCount* parameter increases job dispersal, this causes the number of Blocks Per Job, packet latency time, and packet blocking time all

increase as well. This creates a direct relationship between the *MaxBlockCount* parameter value and those three performance indicators. Notice that, increasing the *MaxBlockCount* parameter value causes the Percent-system-utilization performance indicator to increase as well.

B. Fuzzy Control Systems: Basics

Fuzzy logic is a technique used to handle lexically imprecise propositions with elastic constraints [23]. Practically, fuzzy logic provides a framework for handling sets of rules expressed in an imprecise form. One very basic concept of fuzzy logic is the linguistic variables. Intrinsically, fuzzy logic uses the whole interval of real numbers between zero (pure False) and one (pure True). For example, expressing temperature is possible through a linguistic variable that takes values such as high, low and quite low. Such linguistic variables are naturally embedded in the fuzzy rules of a fuzzy controller and in our nature as humans. In fact, with fuzzy logic processing expressions from natural language, such as 'small', 'large', and 'approximately equal' becomes possible in computers [23].

C. Definitions: Fuzzy Logic Basic Terms

i) Fuzzy Sets and Degree of Memberships

A fuzzy set is defined as a set whose member elements have degrees of membership. In classical sets, the membership of member elements in a given set is assessed in binary terms (either an object is a member of a set or not). Fuzzy sets are different, in the set of "young" people for example, a new born baby will is definitely a member of this fuzzy set, similarly, a 100 years old human is not a member of this set, humans of ages 20, 30, or 40 years partially members of this set with decreasing levels of membership. The grade of membership for all members of a fuzzy set describes that fuzzy set.

An object's grade of membership is a real number in the interval between 0 and 1 (Fig. 5). The Grade of membership is often denoted by the Greek letter μ. Higher number implies higher membership.

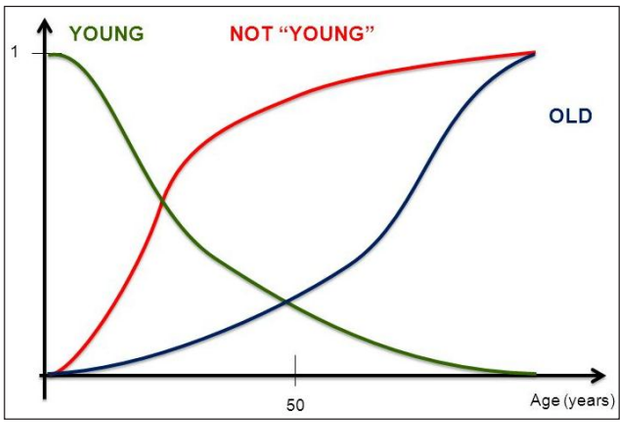

Fig. 5: The Sets More or "Young", "Not Young", and "Old"

Objects within a fuzzy set are taken from a 'universe of discourse' or 'universe' for short. A universe contains all objects that can come into consideration which, depends on the context of course. In describing 'taste' for instance, the elements of this term are taken from a universe such as {bitter, sweet, sour, …}.

Every object or element in the universe of discourse belongs to fuzzy set to some grade as mentioned before. The function that relates a number to each element x of the universe of discourse is called the membership function and is denoted by μ(x).

ii) Continuous or Discrete Membership Functions

Two types of membership functions can be used: continuous or discrete. In the continuous form, the membership function is basically a mathematical function. A membership function is for example bell-shaped (also called a π-curve), s-shaped (called s-curve), or z-shaped (called z-curve or reverse s-curve). All membership functions in Fig. 5 are of the second type. A π-curve membership function in the same context of Fig. 5 would be the "middle age" function. In the discrete type of membership function and the universe are discrete members or points in a list (i.e., vector). Depending on the application, sometimes it can be more convenient to use discrete membership function with sampled (discrete) representation [24].

iii) Representing Continuous Membership Functions

Cosine functions can be used to represent a variety of s-shaped membership functions as follows:

$$s(x_l, x_r, x) = \left\{ \begin{array}{ll} 0 & , x < x_l \\ \frac{1}{2} + \frac{1}{2}\cos\left(\frac{x - x_r}{x_r - x_l}\pi\right) & , x_l \le x \le x_r \\ 1 & , x > x_r \end{array} \right\}$$

Where x_l is the left breakpoint, and x_r is the right breakpoint. The z-shaped curve is just a vertical reflection of this curve as follows:

$$z(x_l, x_r, x) = \left\{ \begin{array}{ll} 1 & , x < x_l \\ \frac{1}{2} + \frac{1}{2}\cos\left(\frac{x - x_l}{x_r - x_l}\pi\right) & , x_l \le x \le x_r \\ 0 & , x > x_r \end{array} \right\}$$

The π-shaped cure can be constructed as a combination of the s-shaped and z-shaped curves such that the peak of the new curve is flat in the interval $[x_3, x_4]$ as follows:

$$\pi(x_1, x_2, x_3, x_4, x) = \min(s(x_1, x_2, x), z(x_3, x_4, x))$$

Where x_1 is the left breakpoint, and x_2 is the right breakpoint.

A fuzzy set is normalized if any membership value in it is less than or equals to 1. To normalize a fuzzy set membership function (or vector), we divide each membership value by the largest membership in the complete set.

D. Fuzzy Control

The process of how fuzzy logic control systems work is shown in Fig. 6A and 6B. Firstly, a crisp set of input data are gathered

from the system to be controlled. In our case, the crisp input is the value of the performance indicator to be measured. After that, this crisp input is converted to a proper fuzzy set through the fuzzification step. Fuzzification implies using fuzzy linguistic variables, fuzzy linguistic terms and membership functions. Inference rules are then used to produce fuzzy outputs that are mapped to a crisp output, again using the proper membership functions, through the defuzzification step. Those steps are presented in Fig. 6 that shows the block diagram of a typical fuzzy control system. Notice that Fuzzy control systems use statements rather than equations. Those statements are represented as empirical rules [27, 28].

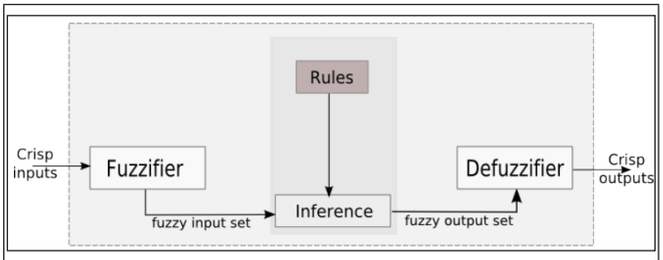

Fig. 6A: A Simple Fuzzy Control System

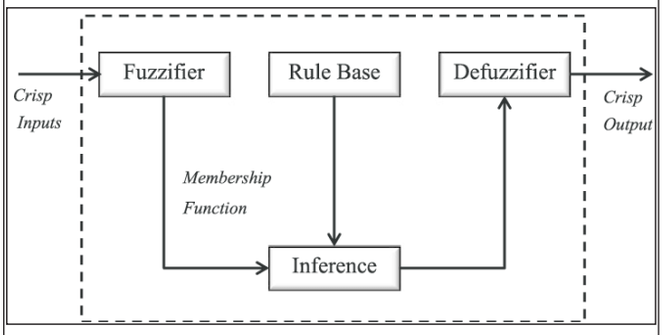

Fig. 6B: A Block Diagram of a Fuzzy Control System

Fig. 6

The fuzzy control system basically compares the detected crisp input values from the system environment and compares it to the targeted key performance indicators to compute the "steady-state error". The computed error (Fig. 7) is then used to modify specific system parameters that have a direct or indirect impact on the targeted performance indicators in the system in hand. The parameter of interest in our context is the maximum number of partitions allowed (*MaxBlockCount* in Fig. 3 above) [27, 28].

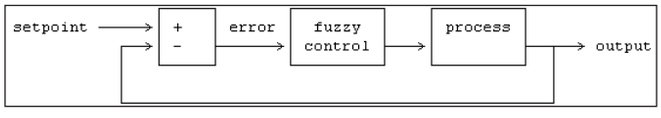

Fig. 7: Steady State Error for Closed-Loop Feedback Control System

IV. Experimental Setting

Every processor allocation mechanisms studied in this research is implemented in C programming language and included into the unix-based parallel simulation tool termed ProcSimity and introduced in [25, 26]. Table II lists the basic experimental setting of the simulated system:

Table II: The Nature of Relationship between the *MaxBlockCount* Parameter and Keyparallel System Performance Indicators

Simulator Setting	Value
Number of simulated jobs per run	1000
Number of runs	10
Job run-mode	Run to completion (no time-sharing)
Confidence level of calculated performance indicators	95%
Relative error of calculated performance indicators	5%
Simulated parallel jobs communication pattern	All-to-all communication pattern [18]
Scheduling mechanism used	First-Come First-Serve, FCFS
Simulated parallel system size	32 x 32 computing units (a total of 1024 computing elements).
Simulated parallel job size distribution	The uniform distribution with 1 to 10 units for each dimension
The set of performance factors considered	MRT, JST, the Simulation "Finish Time" (FT), PSU, MPBT, MPL, and MBPJ

V. Experimental Results and Observations

In this section, we report the experimental results of the proposed allocation strategies after we implemented them. This section is of three parts. In the first part, we comparatively evaluate the proposed techniques with the already available allocation strategies from literature. In this part, we do our experiments in a loaded system (with mean inter-arrival time between jobs be 1). In the second part, we present our observations on the impact of the temporal system load on the key performance indicators of the proposed and the present strategies from literature. In the third part, we present our observations on the impact of the communication load (i.e., # of messages/job) on the key performance indicators of the proposed and the present strategies from literature.

A. Part I: Performance in Loaded System

In this section, we comparatively evaluate the proposed techniques with the already available allocation strategies from literature. To this end, we do our experiments in a loaded system (with mean inter-arrival time between jobs be 1). In all experiments of this part, the number of messages per parallel job is fixed at 30 messages.

In the following table, we present some clarifications about the allocation strategies under study and their abbreviated names in the figures.

TABLE III: THE SET OF ALLOCATION STRATEGIES UNDER STUDY AND THEIR ABBREVIATED NAMES IN DEPICTED GRAPHS

Allocation Strategy	Information about the Strategy
"BGP(1,0)"	The Best-Fit allocation strategy [11]
"BGP(2,0)"	The Bounded Gradual Partitioning allocation strategy [6, 20, 22] MaxBlockCount is fixed at 2
"BGP(4,0)"	The Bounded Gradual Partitioning allocation strategy [6, 20, 22] MaxBlockCount is fixed at 4
"BGP(8,0)"	The Bounded Gradual Partitioning allocation strategy [6, 20, 22] MaxBlockCount is fixed at 8
"BGP(20,0)"	The Bounded Gradual Partitioning allocation strategy [6, 20, 22] MaxBlockCount is fixed at 20
"BGP(60,1)"	The proposed Bounded Gradual Partitioning allocation strategy MaxBlockCount is dynamically decided to make the temporal Percent-system-utilization be 60%.
"BGP(80,1)"	The proposed Bounded Gradual Partitioning allocation strategy MaxBlockCount is dynamically decided to make the temporal Percent-system-utilization be 80%.

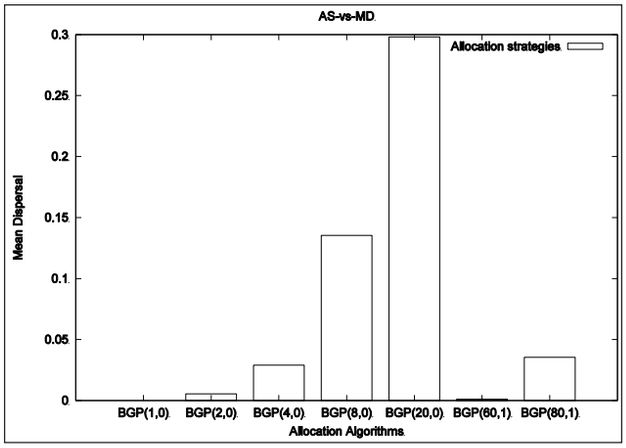

Fig. 8: Allocation Strategies versus Mean Dispersal

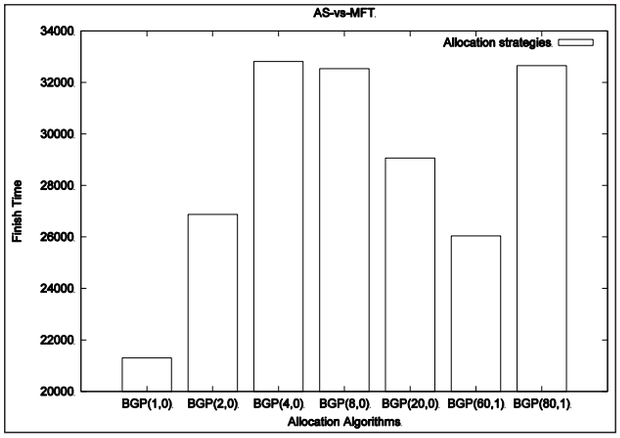

Fig. 9: Allocation Strategies versus Mean Finish Time

Fig. 8 depicts allocation strategies versus mean dispersal in a loaded system. The Bounded Gradual Partitioning allocation strategy where *MaxBlockCount* is fixed at 20 has shown the worst (highest) job dispersal. While the proposed allocation strategies (where *MaxBlockCount* is dynamically decided to make the temporal Percent-system-utilization be 60% and 80%) showed relatively good performance compared to the best allocation strategy in terms of this performance indicator; namely, the Best-Fit allocation strategy. Notice that the job dispersal of the Best-Fit strategy is zero as it assigns one single block for the parallel job.

Fig. 9 depicts allocation strategies versus mean simulation finish time in a loaded system. The Bounded Gradual Partitioning allocation strategy where *MaxBlockCount* is fixed at 20 has shown high mean simulation finish time. The same observation applies to all BGP allocation strategies. This is expected because the simulator simulates 1000 parallel jobs per run for a total of ten runs. Since BGP strategies increases Percent-system-utilization and since the job scheduler uses the FCFS scheduling mechanism, the simulator takes long finish time as relatively large parallel jobs has to remain in the system's waiting queue before they can be successfully allocated and served. This causes the RT and the ST of individual parallel jobs become relatively high. This is clear from Fig. 10 and 11 (shown next). Fig. 10 and 11 shows MJRT and MJST depicted against allocation strategies.

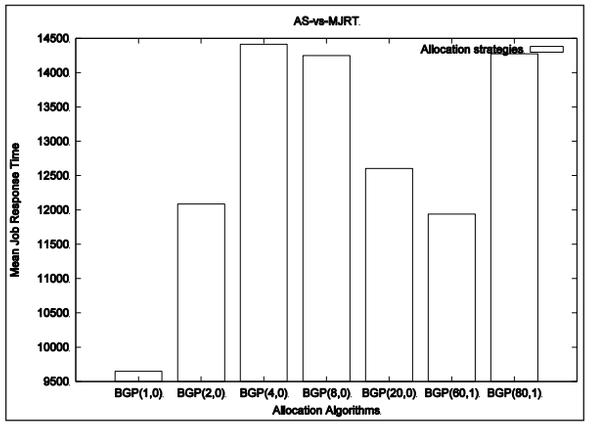

Fig. 10: Allocation Strategies versus MJRT

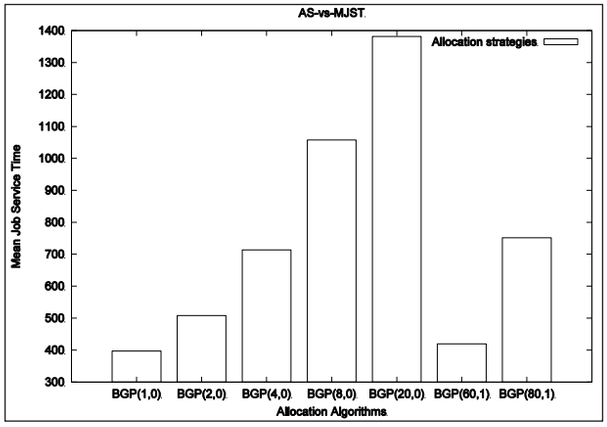

Fig. 11: Allocation Strategies versus Mean-Job-Service-Time

Fig. 12 and 13 depicts the allocation strategies under study versus MPBT and MPL in a loaded system. For a specific message, MPBT is computed as the total time the head of the communication messages (using wormhole routing) is blocked at each and every station over the source-to-destination path. Notice that in wormhole routing, each message is subdivided into a number of packets. MPL for a specific message is computed as the average of the time that all packets within a parallel job will be sent between processors (From source processor to distention processor). The Best Fist allocation strategy has shown the best (lowest) packet blocking time and packet latency of all allocation strategies. This is expected as this allocation strategy does not disperse parallel jobs. The worst allocation strategies according to these two performance indicators are the Bounded Gradual Partitioning strategies where *MaxBlockCount* is fixed at 8 and 20. The proposed allocation strategies have showed better performance and in the case of having the *MaxBlockCount* be dynamically determined to maintain 60% Percent-system-utilization has shown comparable performance to the Best-Fit allocation strategy.

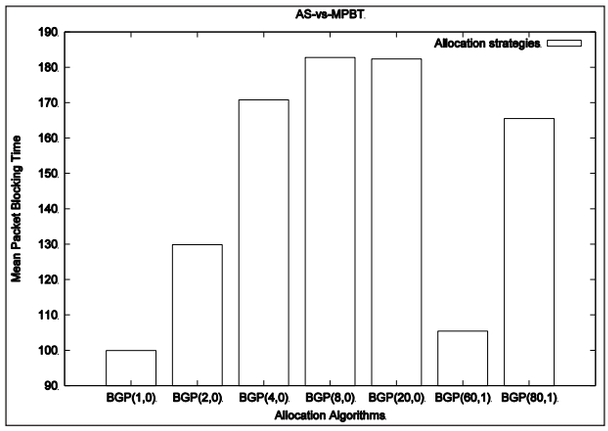

Fig. 12: Allocation Strategies versus MPBT

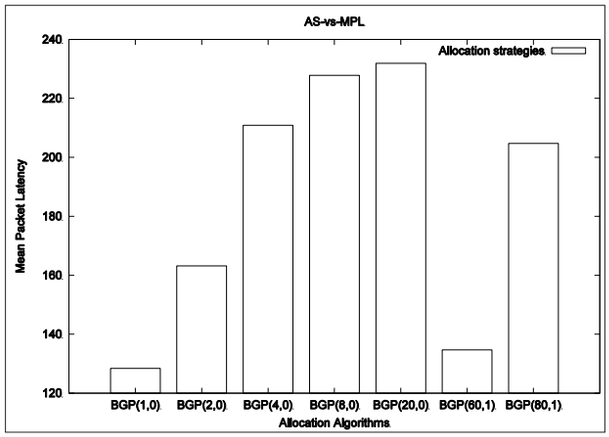

Fig. 13: Allocation Strategies versus Mean Packet Latency

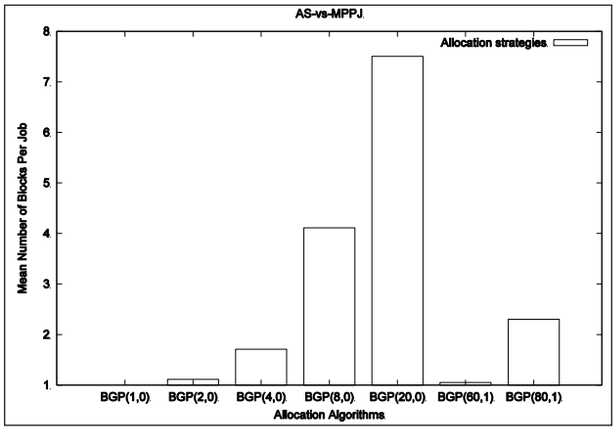

Fig. 14: Allocation Strategies versus MBPJ

Fig. 14 depicts allocation strategies versus MBPJ. MBPJ is computed as the number of *disjoint sub-frames* assigned to the parallel jobs that were served since the system is put into service. As discussed earlier, the Best-Fit allocation strategy do not disperse allocated parallel jobs, each job is assigned one single block. However, the worst allocation strategies according to this factor are the Bounded Gradual Partitioning strategies where *MaxBlockCount* is fixed at 8 and 20. The proposed allocation strategies, on the other hand, it has shown better performance and in the case of having the *MaxBlockCount* be dynamically determined to maintain 60% PSU has shown comparable performance to the Best-Fit allocation strategy.

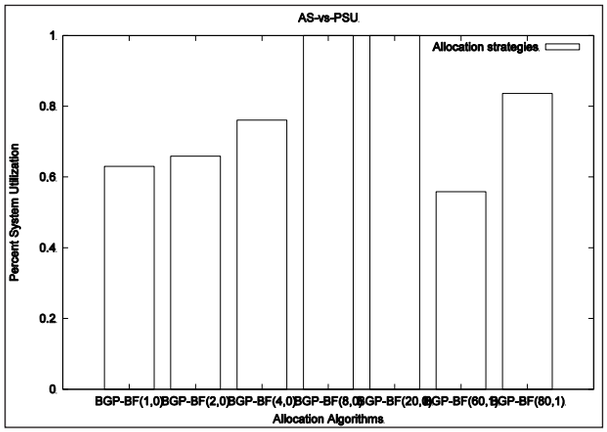

Fig. 15: Allocation Strategies versus Percent-System-Utilization

B. Part II: Performance over System Load

Fig. 16 depicts the system load versus mean number of blocs per job. Notice that as the system gets more loaded, the number of blocks assigned to served parallel jobs increases as the temporal system utilization of the multi-computer system increases. This makes it difficult to successfully allocate parallel jobs without dispersing them. This is clear in Fig. 17 that depicts System load versus mean dispersal. This, in turns, causes the ST of the job to increase. This observation can be noticed in Fig. 18 that depicts system load versus MJST.

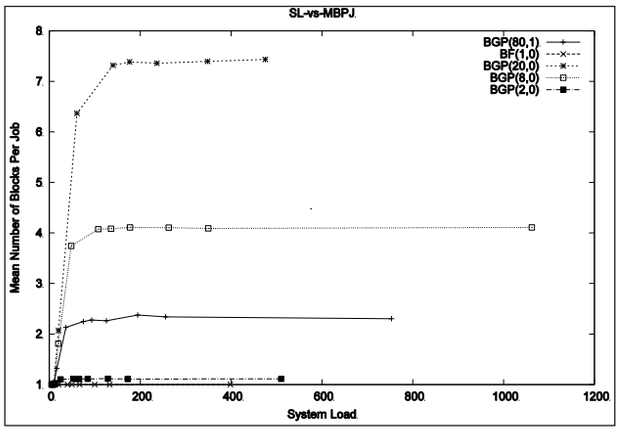

Fig. 16: System Load versus MBPJ

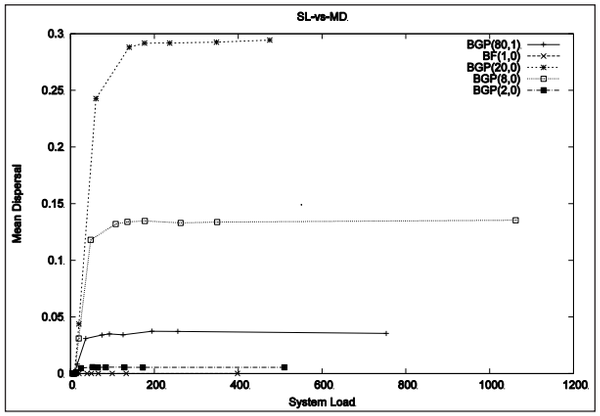

Fig. 17: System Load versus Mean Dispersal

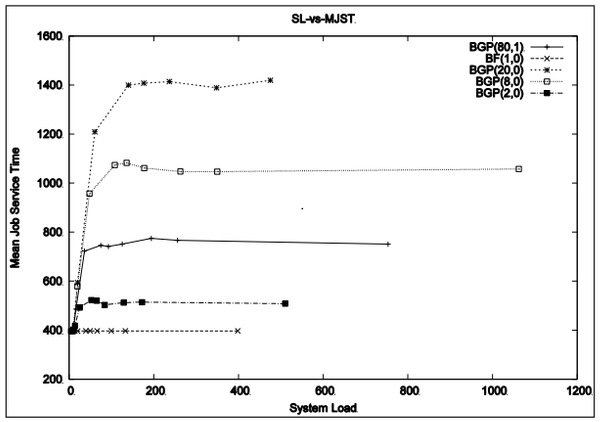

Fig. 18: System Load versus Mean-Job-Service-Time

C. Part III: Performance over Communication Load

In this section, we study the communication load impact on the proposed and examined allocation strategies in this study. In all experiments in this section, we use the well-known all-to-all communication pattern. This communication pattern is used as it produces high message contention. In all-to-all communication or broadcasting, each processing element serves as a source as well as a destination. A parallel process sends the same m-word-size message to every other process allocated to that particular parallel job.

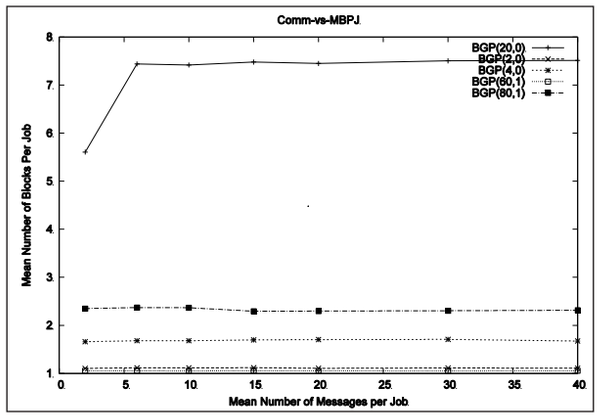

Fig. 19: Communication Load versus MBPJ

Fig. 19 depicts the communication load (represented by the number of messages per parallel job) versus MBPJ. And Fig. 20 depicts the communication load versus mean dispersal level of parallel jobs. We observe that the proposed allocation strategies have shown relatively good and comparable performance compared to the Bounded Gradual Partitioning strategies where *MaxBlockCount* is fixed at 8 and 20.

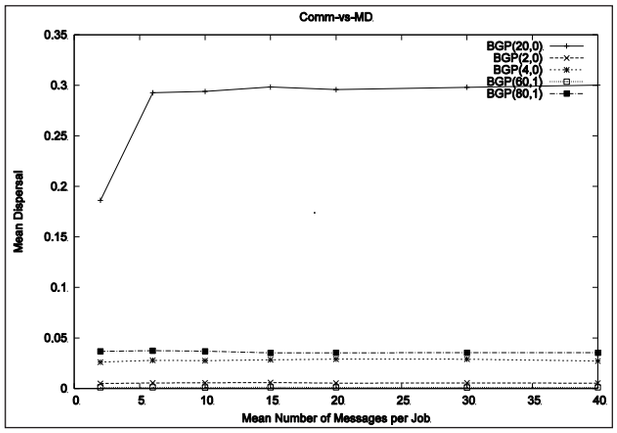

Fig. 20: Communication Load versus Mean Dispersal

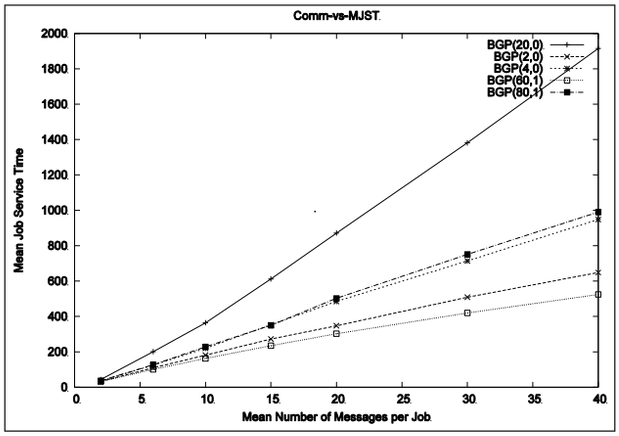

Fig. 21: Communication Load versus MJST

Fig. 19 depicts the communication load versus MPBT. Again, we notice that the proposed allocation strategies have shown relatively good and comparable performance compared to the Bounded Gradual Partitioning strategies where *MaxBlockCount* is fixed at 8 and 20. Those later strategies disperse parallel jobs and, thus, increase the packet latency and the packet blocking times due to message contention. This can also be observed in Fig. 20 that depicts communication load versus MPL.

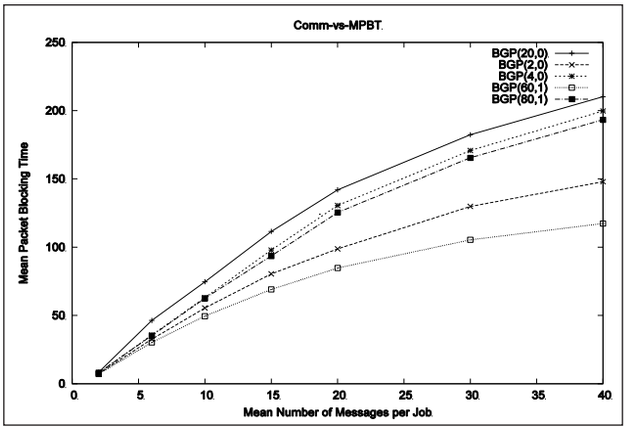

Fig. 22: Communication Load versus MPBT

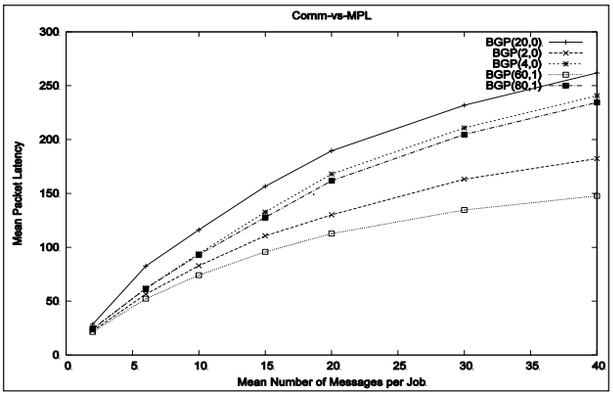

Fig. 23: Communication versus MPL

VI. Conclusion

In this paper, we proposed a framework for using fuzzy logic to control specific key performance levels in two-dimensional mesh-based multi-computer systems. We demonstrated that it is possible to implement a fuzzy-based feedback control system to control a number of performance indicators in the parallel system. We showed that those indicators can be controlled through increasing and decreasing the *MaxBlockCount* parameter in the newly proposed Bounded GRP-BF processor allocation strategy. In this context, the *MaxBlockCount* parameter automatically determined and calculated at run time through a fuzzy-based feedback control system.

The proposed allocation scheme prevents over-splitting of parallel jobs by preventing jobs get over-splitted by controlling the *partitioning-bound*. Fuzzy-based allocation provides a

mechanism to dynamically control the partitioning-bound. Instead of pre-setting and having a fixed partitioning-bound level, we provide a way to set this value at run time taking into consideration the current status to multi-computer system.

As a future work, we will be working on adding a tuning parameter(s) to the proposed approach that grants the operating system (and the user) to control how different performance indicators contribute to the decision of the partitioning-bound value. To this end, a number of tuning methods of controller can be utilized. Thus, we can comparatively evaluate them.

References

[1] S. Bani-Mohammad, I. Ababneh, and M. Yassen, "A new compacting non-contiguous processor allocation algorithm for 2D mesh multicomputers," *JITR*, vol. 8, no. 4, pp. 57-75, 2015.

[2] S. Bani-Mohammad, and I. Ababneh, "Improving system performance in non-contiguous processor allocation for mesh interconnection networks," *Simulation Modelling Practice and Theory*, vol. 80, pp. 19-31, ISSN: 1569-190X, 2018.

[3] G. M. Chiu, and S. K. Chen, "An efficient submesh allocation scheme for two-dimensional meshes with little overhead," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 5, pp. 471-486, May 1999.

[4] Ababneh, "An efficient free-list submesh allocation scheme for two-dimensional mesh-connected multicomputers," *Journal of Systems and Software*, vol. 79, no. 8, pp. 1168-1179, Aug. 2006.

[5] Ababneh, and J. Davis, "Program-based static allocation policies for highly parallel computers," *Proc. IPCCC 95, IEEE Computer Society Press,* Scottsdale, AZ, USA, 28-31 Mar. 1995, pp. 61-68.

[6] S. Bani-Ahmad, "Processor allocation with reduced internal and external fragmentation in 2D mesh-based multi-computers," *Journal of Applied Sciences*, vol. 11, no. 6, pp. 943-952, 2011, doi: 10.3923/jas.2011.943.952.

[7] S. Bani-Mohammad, M. Ould-Khaoua, I. Ababneh, and L. Machenzie, "A fast and efficient processor allocation strategy which combines a contiguous and non-contiguous processor allocation algorithms," Technical Report: TR-2007-229, DCS Technical Report Series, Department of Computing Science, University of Glasgow, Jan. 2007.

[8] B. S. Yoo, and C. R. Das, "A fast and efficient processor allocation scheme for mesh-connected multi-computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 51, no. 1, pp. 46-60, Jan. 2002.

[9] 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, May 1991.

[10] P. J. Chuang, and N. F. Tzeng, "Allocating precise submesh in mesh-connected systems," *IEEE Transactions Parallel and Distributed Systems*, pp. 211-217, Feb. 1994.

[11] 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.

[12] D. Rupanetti, and H. Salamy, "Task allocation, migration and scheduling for energy-efficient real-time multiprocessor architectures," *Journal of Systems Architecture*, ISSN: 1383-7621, vol. 98, pp. 17-26, 2019.

[13] L. E. Rubio-Anguiano, A. C. Trabanco, J. L. B. Velasco, and A. Ramírez-Treviño, "Maximizing utilization and minimizing migration in thermal-aware energy-efficient real-time multiprocessor scheduling," *IEEE Access*, vol. 9, pp. 83309-83328, 2021, doi: 10.1109/ACCESS.2021.3086698.

[14] 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, Jul. 1998.

[15] J. Ding, and L. N. Bhuyan, "An adaptive submesh allocation strategy for two-dimensional mesh connected systems," *Proc. Int. Conf. Parallel Process. II*, Aug. 1993, pp. 193-200.

[16] S. Bani-Mohammad, I. M. Ababneh, and M. Yassen, "A new compacting non-contiguous processor allocation algorithm for 2D mesh multi-computers," *Journal of Information Technology Research (JITR)*, vol. 8, no. 4, 2015.

[17] P. Krueger, T. Lai, and V. A. Radiya, "Job scheduling is more important than processor allocation for hypercube computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, pp. 488-497, May 1994.

[18] V. Lo, K. Windisch, W. Liu, and B. Nitzberg, "Non-contiguous processor allocation algorithms for mesh-connected multi-computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 7, pp. 712-726, Jul. 1997.

[19] T. Liu, K. W. Huang, F. Lombardi, and L. N. Bhuyan, "A submesh allocation scheme for mesh-connected multiprocessor systems," *Proc. Int. Conf. Parallel Process. II*, Aug. 1995, pp. 159-163.

[20] S. Bani-Ahmad, "Submesh allocation in 2D-mesh multicomputers: Partitioning at the longest dimension of requests," *The International Arab Journal of Information Technology (IAJIT)*, vol. 10, no. 3, May 2013.

[21] S. Bani-Mohammad, "An efficient all shapes busy list processor allocation algorithm for 3D mesh multicomputers," *IJCAC*, vol. 7, no. 2, pp. 10-26, 2017.

[22] S. Bani-Ahmad, "Intra-job communication contention and request-partitioning-based allocation strategies in 2D-mesh multi-computers," *Third International Symposium on Parallel Architectures, Algorithms and Programming (PAAP'10),* Dalian, LiaoNing, P. R. China, 18-20 Dec. 2010.

[23] R. Stobart, "Tutorial on fuzzy control," IEE Colloquium on Two Decades of Fuzzy Control - Part 1, London, 1993, pp. 1/1-1/6.

[24] H. J. Zimmermann, *Fuzzy Set Theory and its Applications*. Kluwer Academic, Dordrecht, 1991.

[25] K. Windisch, J. V. Miller, and V. Lo, "ProcSimity: An experimental tool for processor allocation and scheduling in highly parallel systems," *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95)*, Washington, USA, 6-9 Feb. 1995, pp. 414-421.

[26] ProcSimity V4.3 User's Manual, University of Oregon, 1997.

[27] J. Jantzen, Design of Fuzzy Controllers. Tech. Report No. 98-E 864 (design), Technical University of Denmark, Department of Automation, 30 Sep. 1999.

[28] J. Jantzen, Tutorial on Fuzzy Logic. Tech. Report No. 98-E 868, Technical University of Denmark, Department of Automation, 19 Aug. 1998.