Research Journal of Applied Sciences, Engineering and Technology 13(9): 682-689, 2016
DOI:10.19026/rjaset.13.3341
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2016 Maxwell Scientific Publication Corp.
Submitted: November 3, 2014 Accepted: April 1, 2015 Published: November 05, 2016

Research Article

Shape-and Orientation-independent 2D-Buddy Processor Allocation Strategy in 2-D Mesh-based Multicomputers

Othman Jabir, Saleh Oqeili and Sulieman Bani-Ahmad
Prince Abdullah Bin Ghazi Faculty of Information Technology, Al-Balqa Applied University, Main Campus, Salt, Jordan

Abstract: We propose and evaluate a new processor allocation strategy in two-dimensional Multicomputers. The proposed strategy is comparatively evaluated against a set of well-known allocation strategies from the literature, namely; the 2D buddy System (Juang *et al*., 1997), the Multiple Buddy System and the Paging non-contiguous processor allocation strategies. Our experimental results show that the proposed allocation strategy has solved a number of flaws and drawbacks that have been observed in previously proposed strategies. Further, we observed that our proposed allocation strategy is scalable. We refer to the newly proposed strategy by the "Flexible 2D Buddy System" or F2DBS for short. In our experimental results, we also demonstrated that the F2DBS strategy is more flexible and applies to any 2D mesh of any shape and orientation.

Keywords: Multicomputers, processor allocation, 2D buddy system, 2-D mesh, F2DBS

INTRODUCTION

Parallel computing in the present day acquired the largest space in a technology and the computer field (Rauber and Rünger, 2010; El-Rewini and Abd-El-Barr, 2005; Bani-Ahmad, 2011b). In fact, parallel computing and high performance computing in general, has become the dominant paradigm in computer architecture today.

Parallel computing uses multiple processing elements (CPUs basically) simultaneously to solve a problem (Bani-Ahmad, 2013). The problem in hand is thus broken down into multiple and (hopefully) independent parts which such that each processing element can execute its part of the problem simultaneously with others (Grama *et al*., 2003).

In Multicomputers, when job *A* requests group of processors, the request of this job is processed following the same set of steps regardless of the underlying processor allocation strategy used. First, if no enough processors can be found for the job then this job must enter a waiting queue. The waiting queue follows a certain scheduling mechanism, which deals with these jobs and selects the next job to serve (Feitelson and Rudolph, 1996). Assuming job *A* is scheduled, the processor allocation module of the system tries to allocate processors for the job (Dhotre, 2009).

Performance and efficiency of processor allocation strategies is of utmost importance in parallel computing. A good and high-efficiency allocation strategy tries to exploit the processor's network in the best way without any problems and delays and as we mentioned earlier that supercomputer needs to process the operations very quickly (Penrose and Wade, 2003).

An allocator is responsible for booking a free processor to incoming job, while job scheduling or (scheduler) is responsible for controlling the order of waiting jobs for processing.

There are two main types for processor allocation:

- Contiguous Allocation strategies which involve booking processors as one block
- Non-contiguous Allocation strategies that do not depend on booking processors as one block.

LITERATURE REVIEW

Processors allocation strategies and scheduling mechanisms are the main topics in parallel computing and these topics are linked together and affect each other. And its efficiency and performance are reflected on system and user. Researchers try to develop new processor allocation strategies that are efficient and of better performance. Studies show that the processor's allocation strategy used can significantly affect the

Corresponding Author: Othman Jabir, Prince Abdullah Bin Ghazi Faculty of Information Technology, Al-Balqa Applied University, Main Campus, Salt, Jordan
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

efficiency of parallel computing systems (Parhami, 2005).

Processors allocation strategies: Processors allocation strategies are divided into two main types (Contiguous processor Allocation strategies and non-Contiguous processor allocation strategies) and recently there is a new type appears combining the previous types as in Bani-Ahmad (2011a, 2013).

Contiguous allocation strategies:

Frame sliding: Examines the first candidate frame from the lower left most available processor and candidate frame is slid horizontally or vertically by the stride of the width or height of the requested submesh, respectively until an available frame is found (De Rose *et al*., 2007; Windisch *et al*., 1995a). It has high allocation overhead, O (*n*).

First fit: Scans the mesh for a free rectangular submesh large enough for the requested submesh and allocates from the first one found. (Bani-Mohammad *et al*., 2007; Windisch *et al*., 1995a). In First Fit algorithm allocation and deallocation both have a complexity of *O* (*n*).

Best fit: Scans the mesh for free rectangular submeshes large enough for the requested submesh and allocates from the one closest in size to the request (Bani-Mohammad *et al*., 2007; Windisch *et al*., 1995a). Best Fit algorithm both have allocation and deallocation overhead of *O* (*n*).

Non-contiguous allocation strategies:

Random: Randomly selects processors from the set of free processors to satisfy the request (Windisch *et al*., 1995b, 1995a). The complexity of both allocation and deallocation is *O* (*k*), where *k* is the number of processors for satisfying the process (job).

Paging (i): Based on a static partitioning of the mesh into pages of size $2^i \times 2^i$ repeatedly allocates pages to the job in row major order until enough processors have been allocated to satisfy the request. Variations of paging use scan patterns other than row major (Lo *et al*., 1997; Windisch *et al*., 1995a). The complexity of allocating is *O* (*k*) and the complexity of deallocation is *O* (*n*), where *k* is a number of processors for satisfying the process (*job*).

Multiple buddies: Allocates multiple square blocks whose side lengths are powers of two until the job has received exactly the number of processors requested (Liu *et al*., 1994; Li and Cheng, 1990; Windisch *et al*., 1995a). The complexity of both allocation and deallocation for MBS Algorithm is *O* (*n*).

Scheduling mechanisms:

First Come First Served (FCFS): "As the name suggests, in FCFS scheduling, the processes are executed in the order of the arrival in the ready queue, which means the process that enters the ready queue first, get the CPU first" (Sargunar, 2011).

Last Come First Served (LCFS): As in the FCFS scheduling mechanism with a difference, where the process that enters the ready queue last, get the CPU first.

Shortest (Longest) Service Demand (SSD, LSD): The job with short (long) service demand will first processing and service demand multiplied by the number of processors are job neediest. The service demand is calculated as the estimated service time multiplied by the number of processors needed.

Shortest (Longest) Hold Time (SHT, LHT): The jobs with short (long) hold time will first processing. Hold time is the time the job spends in waiting queue.

Smallest (Largest) Job first: These mechanisms depend on size of job and the number of processors the job need.

THE PROPOSED PROCESSOR ALLOCATION STRATEGY

In modern processor allocation strategies, we have noticed many researchers tend to divide the Mesh to enhance the performance and we found a set of algorithms with good performance such as the Multiple Buddy Systems (MBS), 2-D Buddy System (2DBS) and Paging.

In this study we developed a flexible strategy that gave a range of options in the division, unlike other strategies, which restricted the mesh division. For example, in paging (i) allocation strategy the mesh division has been restricted by the formula ($2^i \times 2^i$) where the division will be restricted by the number of 2 multiplied only (1×1, 2×2, 4×4, 8×8, 16×16 … etc.) other dimensions are forbidden in this strategy for example (1×2, 3×3, 5×5).

In a 2-D Buddy System situation, the dimensions of mesh subdivisions are not restricted to number 2 multiplied, the only restriction is that the two dimensions of the submeshes be equal.

In Multiple Buddy System, the division starts from 2^i until 2° and we note that the division, again, is restricted to be multiple of the number 2.

Our new algorithm significantly reduces those restrictions, the groundwork for the new algorithm is to control the division of mesh and make it more flexible. We note in the previous algorithms that the division is static and restricted and some algorithms control the mesh size too. In the new strategy, we will be having flexibility in dividing the mesh.

The F2DBS algorithm:

Step 1: Let M (w, h) be a 2-D mesh with size wxh, J (Js) be a job of size Sj to be allocated.

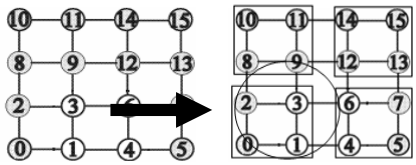

Fig. 1: Demonstrating the proposed algorithm through an example

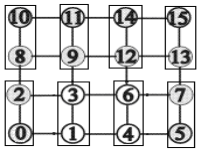

Fig. 2: Dynamically reconstructing processing unit blocks in the proposed approach

Note: A pre-scheduling step is to subdivide the system mesh in hand into blocks each of which is of a given dimension (e.g., let D (w', h') be the mesh block dimensions). This dimension is decided such that it is suitable for the shape and orientation of the mesh.

Step 2: If the current job size is smaller than or equal to the total number of idle processing units, then;
Give one idle block of size S to the current job
If no blocks of that size exist, reduce the size S
If the allocated block is enough for the job
Done, return success
ELSE
Assign another idle block until the allocation is successful.

Step 3: Else Allocation Failed, Put the Job back to the wait queue.

Scenario (Fig. 1): Assume having a (4×4) 2-D mesh, with processors 0, 2, 5, 7 and processors from 8 to 15 be busy. Assume further that we received a new task that seeks a block of size (2×2). The 2DBS algorithm divides the mesh into ($2^1 \times 2^1$) submeshes and save it to list.

With the 2DBS applied, four submeshes exist and can be allocated for this request: those are (0, 1, 2, 3/4, 5, 6, 7/8, 9, 10, 11/12, 13, 14 and 15). We note that we have 2×2 submesh available in the system (1, 3, 4, 6) but we can't use it because it is not in the list that the 2DBS algorithm makes. Our new allocation strategy makes it more flexible and allows the job to use every processor available in the mesh.

We note in the previous division that we face a problem and the (2DBS) algorithms do not give us any other option, but to wait and that affects the efficiency. If we take the previous scenario according to the new strategy (F2DBS), we will choose the new dimension for the mesh division let (2×1), so the mesh division will be as shown in Fig. 2.

Now we have eight sub meshes according to the new division. The free sub meshes are (1, 3) and (4, 6) and we can use them to allocate the job (2×2) easily. We note that the new strategy gives us multiple options for booking the job in our 2D mesh.

Note: This dimension is considered suitable for the shape and orientation of the mesh.

In the previous scenario, we discussed one strategy, which is (2-D buddy system). And it turns that our new proposed allocation strategy (Flexible 2-D Buddy System) was able to solve the problem that appeared in the scenario (4.1) that by changing the division dimension. And this is the flexibility we are talking about in our new proposed allocation strategy. Note that the solution is very simple but very effective.

EXPERIMENTAL SETUP

We use ProcSimity v3.4 Simulator in our experiments. As mentioned earlier, we must comparatively evaluate processor allocation strategies.

Allocation strategies implemented in ProcSimity: There are set of allocation strategies defined in the simulator as follow: Random*, Multiple Body System (MBS), Paging, First Fit (FF), Best Fit (BF) and Frame Sliding. Details about each of these algorithms can be found in Windisch *et al*. (1995a).

Communication pattern implemented in ProcSimity: A method used to communicate between processors.

There are set of communication pattern defined in the simulator as follow: No Communication, All-to-All, N-Body, One-to-All Random, FFT, NAS Multigrid Benchmark, Divide and Conquer Binomial Tree Algorithm, NAS Kernel CG Benchmark. Details about each of these communication patterns can be found in Windisch *et al*. (1995a).

ProcSimity implement a number of communication patterns. The most important of which are:

- **All-to-All communication pattern:** Each processor in a job sends message to other processors in the same job
- **One-to-All communication pattern:** Randomly select a processor in the job and this processor sends message to processors in the same job
- **Random communication pattern:** Each processor in a job repeatedly sends message to another processor randomly
- **N-body communication pattern:** "In the n-body pattern, the processors assigned to a job form a virtual ring, for a job using n processors, every processor sends a message to its successor processor in the ring in each of (n/2) ring subphases and then sends a message to the processor halfway across the ring during a single chordal subphase" (Bunde *et al*., 2004), as shown in Fig. 3.

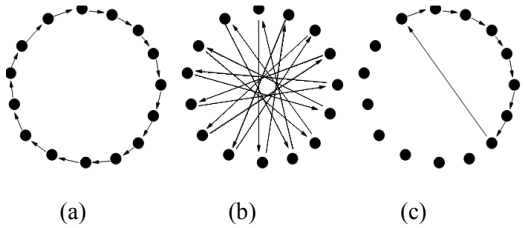

Fig. 3: Messages sent during an N-body communication pattern (Bunde *et al*., 2004)

Scheduling mechanisms implemented in ProcSimity: The method used to organize the jobs and determine the processing priority. There are set of scheduling mechanism defined in the simulator as follow: First Come First Serve (FCFS), Last Come Last Serve (LCFS), Shortest Service Demand First (SSD), Shortest Hold Time First (SHT), Smallest Job First, Longest Service Demand First (LSD), Longest Hold Time First (LHT), Largest Job First and Scan Up. Details about each of these communication patterns can be found in Windisch *et al*. (1995a).

EXPERIMENTAL RESULTS AND OBSERVATIONS

Comparison and discussion: First, we will identify the basic simulator inputs that we use in the comparison between our new allocation strategy and the other allocation strategies.

Mesh Dimension (width×height): We select the group sizes that we will deal with, that is, will be (16×32), (32×32) and (30×15).

Number of jobs: It will be 1000 jobs in every run.

Number of runs: It will be 10 runs in every simulation run. Next we compare our new allocation strategy with other allocation strategies that mentioned before. And we show if our new proposed allocation strategy improves performance and raise efficiency in the processor allocation in 2-D mesh-based multicomputer.

Comparison phase II: After determining the best communication pattern and the best scheduling mechanism that gave us the best performance and high efficiency with our new strategy, here comes the phase of comparison with other allocation strategy algorithms.

Scenario (1):
Mesh width = 16, Mesh Height = 32
Page width = 2, Page Height = 4
Communication pattern: N-Body
Scheduling mechanism: Shortest Hold Time (SHT)

We note that (2DBS) algorithm does not work in this case, because it requires that the mesh height and width must be equal.

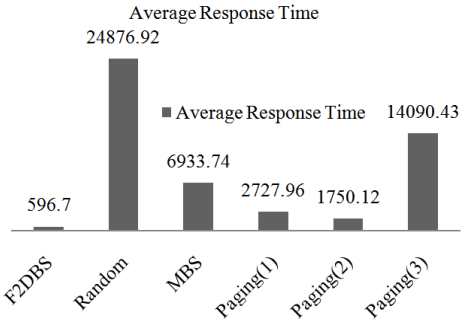

Fig. 4: Average response time for N-Body in multi allocation strategies with Mesh (16×32)

Now for F2DBS algorithm and the rest of algorithms, note that our algorithm achieved readings with high efficiency compared with the rest of the algorithm, for example, the average response time, our algorithm achieved a result of 596.7 and closer algorithm is Paging (2) algorithm achieved 1750.12, knowing that the average response time is the principal criterion for the comparison between the allocation strategies and it was the best proof of the preference of our algorithm on the rest of the algorithms. However, also note the priority of our algorithm on the rest of the algorithms in all the readings, we note its superiority in average service time and system load and an average finish time.

Observations on Fig. 4:
- The best allocation strategy is our new proposed allocation strategy (F2DBS) with result 596.7.
- Note the preference for paging (2) allocation strategy on the rest of the other strategies with result 1750.12.
- Note that the best strategy paging (2) achieved a result 1750.12 and this is too far from what our new proposed strategy achieved 596.7 and this demonstrates the high efficiency of our new allocation strategy.
- The worst allocation strategy is Random Allocation strategy.

Observations on Fig. 5:
- The best allocation strategy is our new proposed allocation strategy (F2DBS) with result 417.57, where there is convergence between the results compared to paging (2) that achieved 657.55.
- The worst allocation strategy is Random Allocation strategy.

Observations on Fig. 6:
- The best allocation strategy is our new proposed allocation strategy (F2DBS) with result 34.93.
- There is convergence between paging (2) allocation strategy and paging (3) allocation strategy.

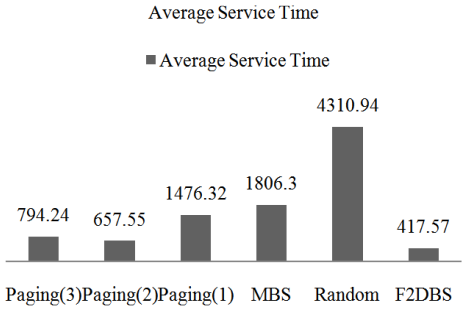

Fig. 5: Average service time for N-Body in multi allocation strategies with Mesh (16×32)

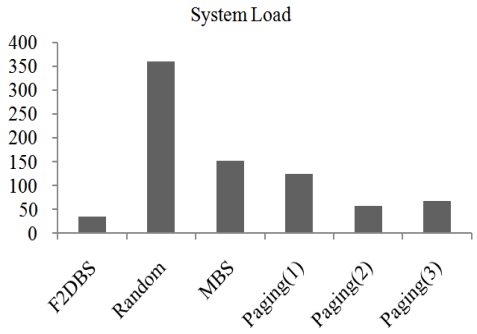

Fig. 6: System load for N-Body in multi allocation strategies with Mesh (16×32)

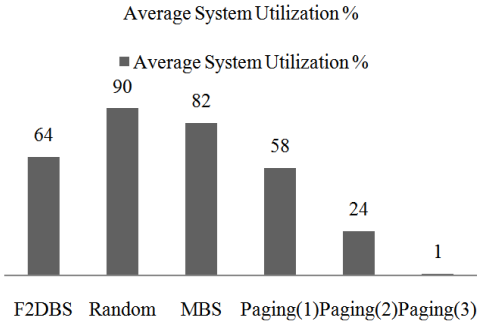

Fig. 7: Average system utilization for N-Body in multi allocation strategies with Mesh (16×32)

- The worst allocation strategy is Random Allocation strategy.

Observations on Fig. 7:

- The best allocation strategy is paging (3) allocation strategy achieved 1% and then paging (2) achieved 24%.
- Our new allocation strategy (F2DBS) achieved 64% and this is not the best result in average system utilization.
- The worst allocation strategy is Random Allocation strategy.

Observations on Fig. 8:

- The best allocation strategy is our new proposed allocation strategy (F2DBS) with result 15243.72.

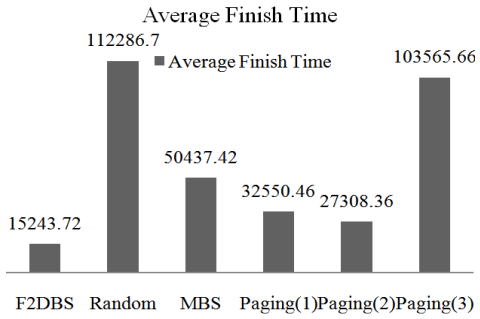

Fig. 8: Average finish time for N-Body in multi allocation strategies with Mesh (16×32)

- There is convergence between paging (2) allocation strategy and paging (3) allocation strategy.
- The worst allocation strategy is Random Allocation strategy.

Summary of observations: When we use (16×32) 2-D mesh and N-Body communication pattern and Shortest Hold Time (SHT) scheduling mechanism we can extract some points:

- The best allocation strategy it is our new proposed strategy flexible 2-D buddy system (F2DBS).
- Compared with other allocation strategies, except our new proposed allocation strategy, paging (2) is the best allocation strategy.
- The worst allocation strategy is Random Allocation strategy.
- In average system utilization the paging (3) achieved the best result comparing with other allocation strategies.

Scenario (2):
Mesh width = 30, Mesh Height = 15
Page width = 5, Page Height = 5
Communication pattern: N-Body.
Scheduling mechanism: Shortest Hold Time (SHT)

We note that (2DBS) and (Paging(i)) algorithms do not work, that's because (2DBS) requires that mesh height and width must be equal in mesh and (Paging(i)) requires that mesh dimension must be from number 2 multiplied such as (2, 4, 8, 16, 32, … etc.).

Here, there is no need to compare between the readings of our algorithm and other existing algorithms.

For example, the average response time, our algorithm achieved 2430.27 and the closer one is (MBS) achieved 10871.28 and this is a huge difference when compared with each other and this applies to the rest of the readings as our algorithm achieved reading with very high efficiency compared to the rest of the algorithms.

Summary of observations for Fig. 9 to 13: When we use (30×15) 2-D mesh and N-Body communication

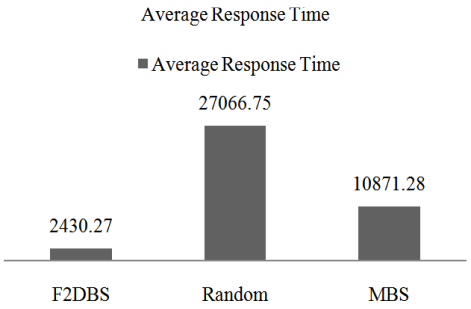

Fig. 9: Average response time for N-Body in multi allocation strategies with Mesh (30×15)

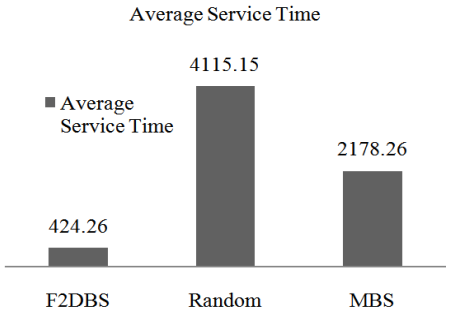

Fig. 10: Average service time for N-Body in multi allocation strategies with Mesh (30×15)

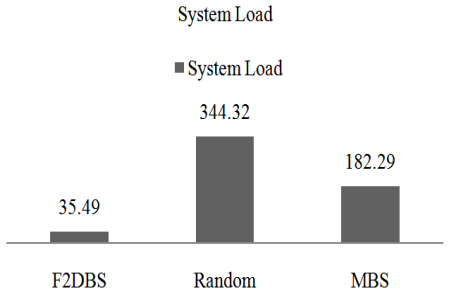

Fig. 11: System load for N-Body in multi allocation strategies with Mesh (30×15)

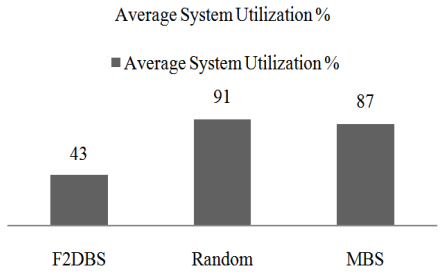

Fig. 12: Average system utilization for N-Body in multi allocation strategies with Mesh (30×15)

pattern and Shortest Hold Time (SHT) scheduling mechanism we can extract some points:

- The best allocation strategy it is our new proposed strategy Flexible 2-D Buddy System (F2DBS)

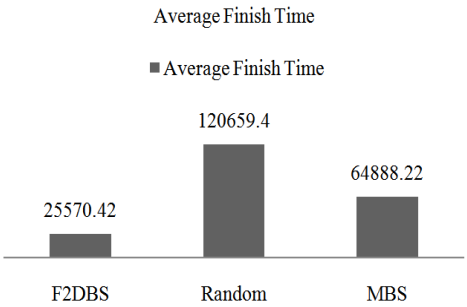

Fig. 13: Average finish time for N-Body in multi allocation strategies with Mesh (30×15)

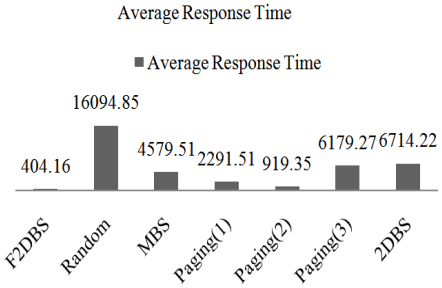

Fig. 14: Average response time for N-Body in multi allocation strategies with Mesh (32×32)

- The worst allocation strategy is Random Allocation strategy.
- In this case shows the benefit of our new proposed allocation strategy where (F2DBS) avoided the drawbacks of other allocation strategy with good performance and high efficiency

Scenario (3):
Mesh width = 32, Mesh Height = 32
Page width = 4, Page Height = 4
Communication pattern: N-Body
Scheduling mechanism: Shortest Hold Time (SHT)

This case is the most famous case in allocation strategy's comparison. It is the basic case and supported by many researchers. In this case, we note that there is a preference for our new algorithm for all other algorithms, Average response time, Average service time, System load, Average system utilization) and Average finish time, all of these standards our new algorithm has high-efficiency readings and this guide to improve the performance of the processor allocation.

Observations on Fig. 14:

- The best allocation strategy is our new proposed allocation strategy (F2DBS) with result 404.16, with a preference for paging (2) allocation strategy on the rest of the other strategies with result 919.35.
- Note that the best strategy paging (2) achieved a result 919.35 and this is the double of what our

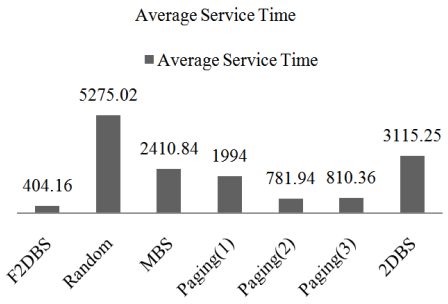

Fig. 15: Average service time for N-Body in multi allocation strategies with Mesh (32×32)

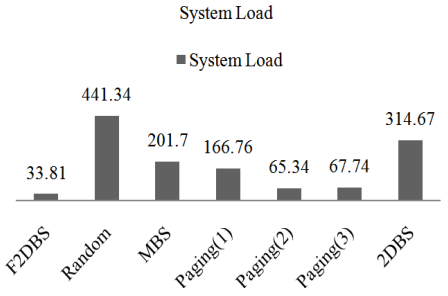

Fig. 16: System load for N-Body in multi allocation strategies with Mesh (32×32)

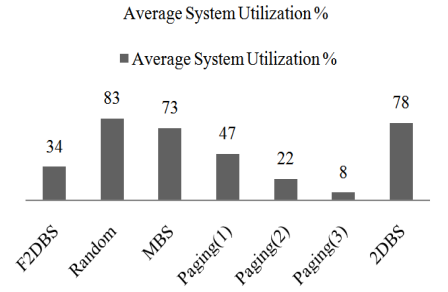

Fig. 17: Average system utilization for N-Body in multi allocation strategies with Mesh (32×32)

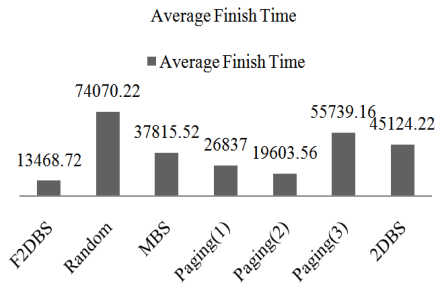

Fig. 18: Average finish time for N-Body in multi allocation strategies with Mesh (32×32)

new proposed strategy achieved 404.16. And this demonstrates the high efficiency of our new allocation strategy.

- The worst allocation strategy is Random Allocation strategy.

Observation on Fig. 15:

- The best allocation strategy it is our new proposed allocation strategy (F2DBS).
- The worst allocation strategy is Random Allocation strategy.
- There is convergence between the paging (2) allocation Strategy and paging (3) allocation strategy.

Observation on Fig. 16:

- The best allocation strategy it is our new proposed allocation strategy (F2DBS).
- The worst allocation strategy is Random Allocation strategy.
- There is convergence between the paging (2) allocation Strategy and paging (3) allocation strategy.

Observation on Fig. 17:

- In system, load criterion the best allocation strategy is paging (3).
- The worst allocation strategy is Random Allocation strategy.
- Our proposed allocation strategy achieved 34% but paging (2) allocation strategy achieved 22% and paging (3) allocation strategy achieved 8% and these results had better than our strategy's result.

Observation on Fig. 18:

- The best allocation strategy it is our new proposed allocation strategy (F2DBS).
- The worst allocation strategy is Random Allocation strategy.

Summary of observations: When we use (32×32) 2-D mesh and N-Body communication pattern and Shortest Hold Time (SHT) scheduling mechanism, we can extract some points:

- The best allocation strategy it is our new proposed strategy flexible 2-D buddy system (F2DBS) except system load criterion.
- In system load criterion, paging (3) is the best allocation strategy.
- The worst allocation strategy is Random Allocation strategy.

CONCLUSION

In this study a new processor allocation strategy in 2-D mesh-based multicomputers was proposed. The new allocation strategy successes avoid the drawbacks shown in the other processor allocation strategies such as (2D Buddy System, the Multiple Buddy System and the Paging non-contiguous Processor allocation strategies), with good performance and high efficiency.

After multiple stages of comparison through ProcSimity simulator, we infer the following set of advantages about the proposed processor allocation strategy "Flexible 2-D Buddy System (F2DBS)":

- The best communication pattern with the proposed allocation strategy (F2DBS) is (N-Body) communication pattern and the worst one is (One-to-All) communication pattern.
- The best scheduling mechanism with the proposed allocation strategy (F2DBS) is Shortest Hold Time (SHT).
- The stability of some readings and not being different from one to another. And this is a good indicator of the possibility of adopting any type of scheduling mechanism (Flexibility in choosing scheduling mechanism in some cases).
- Our new proposed allocation strategy could solve problems in some other allocation strategies like (2DBS) and (Paging (i)), where we could get a reading with high efficiency, whether mesh has equal dimension or dimensions were not multiplied of 2.
- Our new proposed allocation strategy outperformed the rest of the algorithms where it is readings very efficient compared to the other allocation strategy's readings.
- The proposed allocation strategy applies to any 2-D mesh of any shape and orientation.

REFERENCES

Bani-Ahmad, S., 2011a. Bounded gradual-request-partitioning-based allocation strategies in 2D-mesh multicomputers. Int. J. Digital Content Technol. Appl., 5(1).

Bani-Ahmad, S., 2011b. Processor allocation with reduced internal and external fragmentation in 2D Mesh-based multicomputers. J. Appl. Sci., 11(6): 943-952.

Bani-Ahmad, S., 2013. Submesh allocation in 2D-Mesh multicomputers: Partitioning at the longest dimension of requests. Int. Arab J. Inf. Techn., 10(3): 245.

Bani-Mohammad, S., M. Ould-Khaoua and I. Ababneh, 2007. A new processor allocation strategy with a high degree of contiguity in mesh-connected multicomputers. Simul. Model. Pract. Th., 15(4): 465-480.

Bunde, D.P., V.J. Leung and J. Mache, 2004. Communication patterns and allocation strategies. Proceeding of the 18th International Parallel and Distributed Processing Symposium.

De Rose, C.A.F., H.U. Heiss and B. Linnert, 2007. Distributed dynamic processor allocation for multicomputers. Parall. Comput., 33(3): 145-158.

Dhotre, I.A., 2009. Operating Systems. 7th Edn., Technical Publication Pune, India.

El-Rewini, H. and M. Abd-El-Barr, 2005. Advanced Computer Architecture and Parallel Processing. John Wiely and Sons Inc., Hoboken, N.J.

Feitelson, D.G. and L. Rudolph, 1996. Job scheduling strategies for parallel processing. Proceeding of the IPPS'96, Honolulu, Hawaii.

Grama, A., A. Gupta, G. Karypis and V. Kumar, 2003. Introduction to Parallel Computing. 2nd Edn., Addison Wesley, Boston, Massachusetts.

Juang, T.Y.T., Y.C. Tseng and Y.S. Chen, 1997. An enhanced 2D buddy strategy for submesh allocation in mesh networks. Proceeding of the 3rd International Conference on Algorithms and Architectures for Parallel Processing. Melbourne, Vic., pp: 345-352.

Li, K. and K.H. Cheng, 1990. A two dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system. Proceeding of the ACM Annual Conference on Cooperation (CSC '90). New York, pp: 22-27.

Liu, W., V. Lo, K. Windisch and B. Nitzberg, 1994. Non-contiguous processor allocation algorithms for distributed memory multicomputers. Proceeding of the ACM/IEEE Conference on Supercomputing (Supercomputing '94). New York, pp: 227-236.

Lo, V., K.J. Windisch, W. Liu and B. Nitzberg, 1997. Noncontiguous processor allocation algorithms for mesh-connected multicomputers. IEEE T. Parall. Distr., 8(7): 712-726

Parhami, B., 2005. Computer Architecture: From Microprocessors to Supercomputers. Oxford University Press, New York, pp: 575.

Penrose, D.E.M. and E. Wade, 2003. Interconnection Networks: An Engineer Approach. Elsevier Science, USA.

Rauber, T. and G. Rünger, 2010. Parallel Programming: For Multicore and Cluster Systems. Spring-Verlag, Berlin.

Sargunar, J., 2011. Introduction to Computer Science. 2nd Edn., Dorling Kindersley, India.

Windisch, K., J.V. Miller and V. Lo, 1995a. ProcSimity: An experimental tool for processor allocation and scheduling in highly parallel systems. Proceeding of the 5th Symposium on the Frontiers of Massively Parallel Computation. Washington, DC, USA.

Windisch, K., V. Lo and B. Bose, 1995b. Contiguous and non-contiguous processor allocation algorithms for k-ary n-cubes. IEEE T. Parall. Distr., 6: 414-421