computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

A MODIFIED A* ALGORITHM FOR ALLOCATING TASK IN HETEROGENEOUS DISTIRBUTED COMPUTING SYSTEMS

Nirmeen A. Bahnasawy [1], Gamal M. Attiya [2], Mervat Mosa [1], Magdy A. Koutb [2]

[1] Dept. of Computer Science and Engineering, Faculty of Engineering, Menoufia University
nirmeen_a_wahab@hotmail.com, nirmeen23875@yahoo.com
[2] Dept. of Automatic Control Engineering, Faculty of Engineering, Menoufia University

Abstract: *Distributed computing can be used to solve large scale scientific and engineering problems. A parallel application could be divided into a number of tasks and executed concurrently on different computers in the system. This paper provides an optimal task assignment algorithm under memory constraints to minimize required time of finishing a parallel application. The proposed algorithm is based on the optimal assignment sequential search (OASS) of the A* algorithm with additional modifications. This modified algorithm yields optimal solution, lower time complexity, reduces the turnaround time of the application and considerably faster compared with the sequential search algorithm.*

Keywords: *parallel processing, task assignment, distributed computers, heterogeneous processors.*

1. INTRODUCTION

A distributed computing system is characterized by a topology, availability of computers and communication resources. This system consists of heterogeneous computers interconnected through a communication network. Each computer has computation facilities, its own memory, communication capacity and propagation delay between two computers [5]. Such a system can be employed to solve large scale scientific and engineering problems, where, an application could be divided into a number of tasks and executed concurrently on different computers in the system. The distribution of tasks onto different computers of the system is called *allocation problem*. Such a problem is known to be NP-hard, except in a few special cases with strict assumptions.

Several approaches were by many researchers to solve the allocation problem. In [3,8], approaches are developed for allocating and scheduling tasks of a parallel application among computing sites of distributed computing system to achieve some objectives under defined constraints. Shen and Tsia [18] first used the A*algorithm for the task-assignment problem. They ordered the tasks considered for assignment simply starting with task 1 at the tree's first level, task 2 at the second and so on. Kafel [10] considered the assignment problem to minimize the parallel program completion time, they proposed algorithm based on A* algorithm called "Optimal Assignment Sequential Search" (OASS). Also an optimal static scheduling of an arbitrary structured task graph to an arbitrary number of homogeneous processors is discussed on the A*search technique with a computationally efficient cost function and number of state-space pruning technique [16]. Note that, A* is a best-first search algorithm, which has been used extensively in artificial intelligence problem solving. Programmers can use the algorithm to search a tree or a graph.

The problem of minimizing the total cost subject to both memory and processing capacity constraint is discussed in [14] by taking into account the limited capacities of the resources that constitute the communication network (LANs, WANs, and direct link). Task allocation problem is discussed also in [1] where, A* algorithm (A*RS) can be implemented to improve the performance of the earlier algorithm then allocate the multiple tasks that requires more time and space for the solution.

In [5], a simulated annealing technique is developed to quickly find a near optimal solution to make the system more reliable in inter processor communication times under conditions imposed by both the application and system resources. In [12], a genetic algorithm (GA) is developed to finding approximate solutions for problems with very large decision spaces by applying it on the task matching problem independent task. In [16, 17], a new hybrid genetic algorithm is used to solve the task scheduling problem in heterogeneous computing

system which is guarantee that every feasible schedule is reachable with some probability. Many algorithms are compared, evaluated, and get analyzed to present the scheduling task problem [7]. Heuristic algorithm [14] showed that the most effective non-evolutionary known method for scheduling independent tasks in heterogeneous environment is the min-min heuristic.

In this paper, the main objective is to study task assignment problem in distributed systems comprising networked heterogeneous computers and then develop a new technique for obtaining optimal solution to the given problem to minimize the turn around time of the application A comparison is done between the (OASS) and the modified algorithm in assigning a given tasks to a network of processors to minimize the required time for program completion.

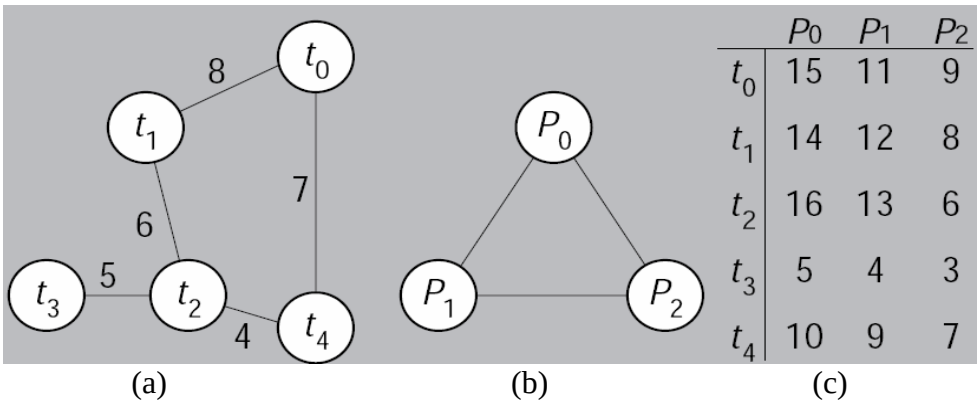
Note that, minimizing the turn around time of a parallel application can be achieved only when the work load is balancing distributed on the different processors of the system

The rest of this paper is organized as follow. Section 2 defines the task assignment problem. Section 3 describes task assignment using optimal sequential search of A* algorithm. Section 4 presents the modified algorithm. Section 5 presents the simulation result, and finally section 6 presents the conclusions.

2. PROBLEM DEFINITION

Task allocation problem may be stated informally as the problem of allocating tasks of a parallel application onto computers of distributed computing system to optimize some performance measures as finishing time [4,6].

In solving the allocation problem, we use the *task interacting graph* model, in which the parallel program is represented by an undirected graph: $G_T = (V_T, E_T)$, where V_T is the set of vertices, $\{t_1, t_2, ..., t_m\}$, and E_T is a set of edges labeled by the communication requirements among tasks. We can also represent the network of processors as an undirected graph, where vertices represent the processors, and the edges represent the processors' communication links. We represent the interconnection network of n processors, $\{p_1, p_2, ..., p_n\}$, by an $n \times n$ link matrix **L**, where an entry $\mathbf{L}_{ij}$ is 1, if processors i and j are directly connected, and 0 otherwise. We do not consider the case where i and j are not directly connected. We can execute a task t_1 from the set V_T on any one of the system's n processors. Each task has an associated execution cost on a given processor. A matrix **X** gives task execution costs, where $\mathbf{X}_{ip}$ is the execution cost of task i on processor p. Two tasks, t_i and t_j, executing on two different processors, incur a communications cost when they need to exchange data. Task mapping will assign two communicating tasks to the same processor or to two different, directly connected processors. A matrix **C** represents communication among tasks, where $\mathbf{C}_{ij}$ is the communication cost between tasks i and j, if they reside on two different processors.



	P_0	P_1	P_2
t_0	15	11	9
t_1	14	12	8
t_2	16	13	6
t_3	5	4	3
t_4	10	9	7

(a) (b) (c)

Fig. 1 – An Example (a) a task graph (b) processor Network (c) Execution cost matrix

A processor's load comprises all the execution and communication costs associated with its assigned tasks. The time needed by the heaviest-loaded processor will determine the entire program's completion time. The task-assignment problem must find a mapping of the set of m tasks to n processors that will minimize program completion time. Task mapping, or assignment to processors, is given by a matrix **A**, where $\mathbf{A}_{ip}$ is 1, if task i is assigned to processor p, and 0 otherwise. The following equation then gives the load on p:

$$Z = \sum_{i=1}^{m} X_{ij} \bullet A_{ip} + \sum_{\substack{q=1 \\ (q \neq p)}}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} (C_{ij} A_{ip} A_{jq} L_{pq})$$

The first part of the equation is the total execution cost of the tasks assigned to p. The second part is the communication overhead on p. A_{ip} and A_{jq} indicate that task i and j are assigned to two different processors (p and q), and L_{pq} indicates that p and q are directly connected. To find the processor with the heaviest load, you need to compute the load on each of the n processors. The optimal assignment out of all possible assignments will allot the minimum load to the heaviest-loaded processor.

3. SEQUENTIAL SEARCH ALGORITHMS (OASS)

Task assignment using the A* algorithm will be occurred as follows. For a tree search, it starts from the root, called the start node (usually a null solution of the problem). Intermediate tree nodes represent the partial solutions, and leaf nodes represent the complete solutions or goals. A cost function f computes each node's associated cost. The value of f for a node n, which is the estimated cost of the cheapest solution through n, is computed as: $f(n) = g(n)+ b(n)$, where $g(n)$ is the search-path cost from the start node to the current node n and $b(n)$ is a lower-bound estimate of the path cost from n to the goal node (solution). To expand a node means to generate all of its successors or children and to compute the f value for each of them.

The nodes are ordered for search according to cost; that is, the algorithm first selects the node with the minimum expansion cost. The algorithm maintains a sorted list, called OPEN, of nodes (according to their f values) and always selects a node with the best expansion cost. Because the algorithm always selects the best-cost node, it guarantees an optimal solution.

For the task-assignment problem under consideration,

- The search space is a tree;
- The initial node (the root) is a null assignment node—that is, no task is assigned;
- Intermediate nodes are partial-assignment nodes—that is, only some tasks are assigned; and
- A solution (goal) node is a complete assignment node—that is, all the tasks are assigned.

```
Generate a random solution
Let S_Opt be the cost of this solution
Build the initial node s and insert it into the list OPEN
Set f(s) = 0
Repeat
    Select the node n with smallest f value.
     Repeat
     Make memory test step
      If (n is true)
    if (n is not a Solution )
       Generate successors of n
        for each successor node n' do
            if (n' is not at the last level in the search tree)
                  f(n') = g(n') + b(n')
           else   f(n') = g(n')
           if (f(n') <= S_Opt)
                 Insert n' into OPEN
         endif
       end for
      else select (n+1)
    end if
    if (n is a Solution)
      Report the Solution and stop
Until (n is a Solution) or (OPEN is empty)
```

Fig. 2 – The Optimal Assignment with Sequential Search algorithm (OASS)

To compute the cost function, $g(n)$ is the cost of partial assignment *(A)* at node n the load on the heaviest-loaded (p); this can be done using the equation from the previous section. For the computation of $b(n)$, two sets T_p (the set of tasks that are assigned to the heaviest-loaded p) and U (the set of tasks that are unassigned at this stage of the search and have one or more communication link with any task in set T_p) are defined. Each task t_i in U will be assigned either to p or any other processor q that has a direct communication link with p. So, associate two kinds of costs with each t_i's assignment: either X_{ip} (the execution cost of t_i on p) or the sum of the communication costs of all the tasks in set T_p that have a link with t_i. This implies that to consider t_i's assignment, we must decide whether t_i should go to p or not (by taking these two cases' minimum cost). Let cost (t_i) be the minimum of these two costs, then we compute $b(n)$ as:

$$b(n) = \sum_{t_i \in U} \cos t(t_i)$$

The A* algorithm for the task-assignment problem has been used early. The tasks are ordered considered for assignment simply by starting with task 1 at the tree's first level, task 2 at the second, and so on.

OASS algorithm [10] uses the A* search technique, but with two distinct features. First, it generates a random solution and prunes all the nodes with costs higher than this solution during the optimal-solution search see Figure 2. This is because the optimal solution cost will never be higher than this random-solution cost. Pruning unnecessary nodes not only saves memory, but also saves the time required to insert the nodes into OPEN. Second, the algorithm sets the value of *f(n)* equal to *g(n)* for all leaf nodes, because for a leaf node n, *b(n)* is equal to 0. This avoids the unnecessary computation of *b(n)* at the leaf nodes.

4. THE MODIFIED ALGORITHM

In this section a new task assignment algorithm is presented. The basic idea is to choose the task to be assigned at each level. The assignment problem is represented as A*algorithm for traversing the tree nodes searching for an optimal solution.

The proposed algorithm handles tasks at the tree levels according to the task of higher connectivity, i.e., the task with the largest number of neighbors and then test the memory constrains, This constrain shows; for an assignment A, the total memory required by all tasks assigned to a processor p must be less than or equal to the available memory capacity of the processor p. Let m_i denotes the amount of memory required for processing a task i and M_p defines the available memory capacity at the processor p, then the following inequality must hold at each processor p in the system:

$$\sum_i m_i A_{ip} \le M_p \ \forall p$$

i.e., before distribution, the program compares memory capacities of the chosen task to be run on the chosen processor ; if their memory capacities are equal or memory capacity of task is less than the memory capacity of processor or memory capacity of processor permits to run that task, the program will complete, and if the memory capacity of task is larger than the memory capacity of processor, the program will search for another one so, the chosen task will choose the next processor to make the same test and neglect that one which is not fit then generate node, and so on until the program complete; this step is called memory constrain step (1) as illustrated in Figure 3.

The algorithm starts by reordering the application tasks according to the task of more connectivity, i.e., calculate the sum of the communication lines for all tasks then choose the highest one, if two tasks have the same connectivity degree; it considers the task of higher communication requirements, then it generates a random solution and prunes all the nodes with costs higher than this solution during the optimal-solution search, this is because the optimal solution cost will never be higher than this random-solution cost. Pruning unnecessary nodes not only saves memory, but also saves the time required to insert the nodes into OPEN, also the algorithm sets the value of *f(n)* equal to *g(n)* for all leaf nodes, because for a leaf node n, *b(n)* is equal to 0, this avoids the unnecessary computation of *b(n)* at the leaf nodes.

Our proposed algorithm is effective in reducing the number of nodes generated (and that are expanded) without sacrificing the optimality criteria. This property allows to reduce mathematical computations such as computing *f(n)* for latest nodes.

The proposed algorithm will be applied on the previous example to study in details the comparison between it and the sequential search algorithm.

Consider (Figure 1), by applying the idea of the proposed algorithm in (Figure 3). The new order of tasks list is obtained as follow: $\{t_2,t_0,t_1,t_4,t_3\}$, the number of resulting node expansion is reduced compared with sequential search algorithm.

```
Generate a random solution
 Let S_Opt be the cost of this solution
For (i=0;i<=n;++)
    Compute connectivity degree and communication requirements.
    If (ti connectivity degree>ti+1 connectivity degree)
       Set ti in ordered task list
          If (ti connectivity degree= ti+1 connectivity degree)
            Compute the communication requirements.
                If (ti communication requirements>ti+1communication requirements)
                    Set ti in ordered task list
                Else
                    If (ti communication requirements=ti+1 communication requirements)
                     Set ti in ordered task list
          Else
              Set ti+1in ordered task list
    Else
       Set ti+1in ordered task list
End for
 Build the initial node s and insert it into the list OPEN
 Set f(s) = 0
 Repeat
     Select the node n with smallest f value.
    Make memory test step
     If(n is true)
     if (n is not a Solution )
        Generate successors of n
         for each successor node n' do
              if (n' is not at the last level in the search tree)
                    f(n') = g(n') + b(n')
             else   f(n') = g(n')
             if (f(n') <= S_Opt)
                    Insert n' into OPEN
```

```
    Else select n+1
       end for
     end if
     if (n is a Solution)
       Report the Solution and stop
 Until (n is a Solution) or (OPEN is empty)
```

Fig. 3 – The Proposed Algorithm

Consider t_2 with three communication links, t_0 which has two communication links like: t_0, t_1, t_4, then the highest communication requirements is chosen, and so on. We get 11 nodes are generated by applying the modified algorithm When OASS algorithm is applied on the same example, 14 nodes are generated compared with these results yield the same optimal solution. This reduction in results saves memory, increase speedup and the performance of program. The modified algorithm has considerably fewer memory requirements than OASS algorithm.

5. SIMULATION REULTS

To test the performance of the algorithm first, simulated by C# ver.5.1and then, a simulation environment in acer core due processor with 1.73 speedup.

To test the modified algorithm the data collect for task graphs of 2 to 30 nodes and processor graphs of 2, 4, 8 nodes. Figure 4 shows node generated of A*O, OASS and modified algorithm on 4 processor nodes. As a result, the number of generating nodes decreases, the running time of program decreases, so the system required memory decreases and then memory efficiency increases in Malg.

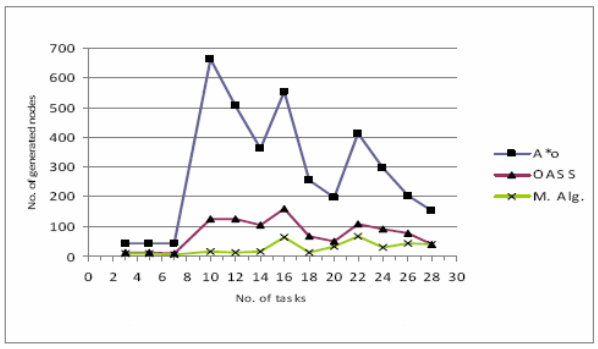


Fig. 4 – Generated nodes case of 4 processors

We also obtain a comparison between the same three algorithms in the running time parameter. The modified algorithm shows a substantial improvement over (OASS) algorithm. Figure 5 shows collecting data for task graph of 3 to 28 tasks and processor graph of four nodes, the results of running times in seconds are obtained, when running time decreased, the speedup of program increased so, the modified algorithm behaves better performance than (OASS) algorithm.

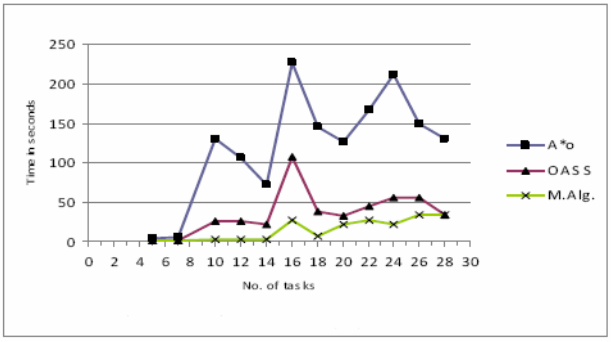


Fig. 5 – Runing times using 4processors

Figures 6 (a, b, c) show generated nodes of A*O, OASS and modified algorithm on (2, 4, 8) processor nodes respectively, the modified algorithm shows a substantial improvement over (OASS) algorithm and A*O algorithm.

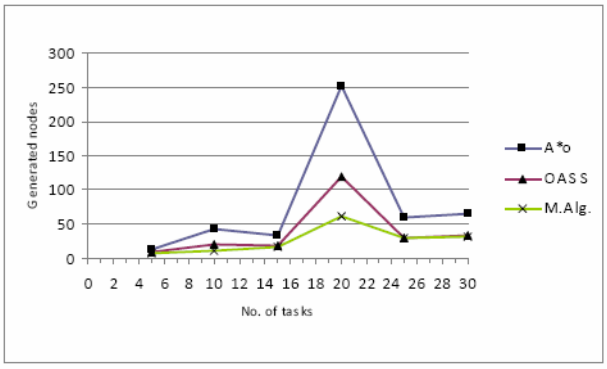


Fig. 6(a) – Generated nodes caseof 2 processors

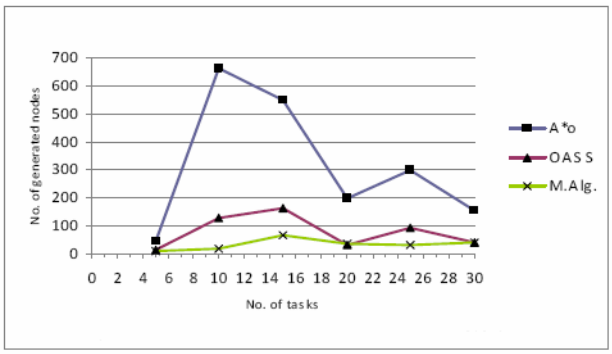


Fig. 6(b) – Generated nodes case of 4 processors

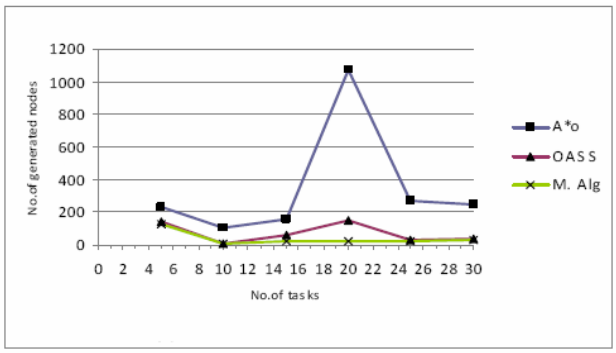


Fig. 6(c) – Generated nodes case of 8 processors

In addition, the improvement in the modified algorithm performance, it efficiently uses the system memory as shown in Figure (7).As the total number of nodes decreases, the required memory saving decreases. Note that, the saving rate is defined as:

$$(1-(G_{(Malg.or\ OASSalg)} / G_{A^*alg})),$$

Where, $G_{(M.alg.or\ OASS\ alg)}$ is the actually number of nodes generated by the two algorithms with respect to A* algorithm.

Figures 7 (a, b, c) show the memory saving rates on (2, 4, 8) processor nodes respective

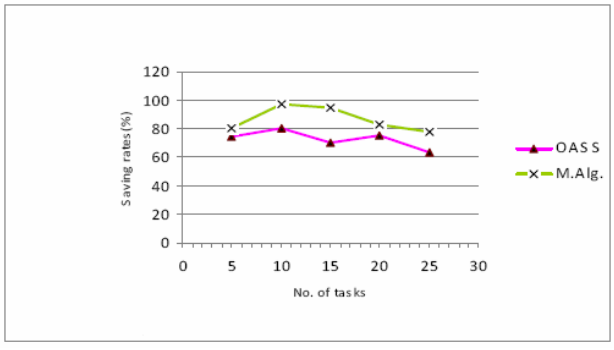


Fig. 7(a) – Memoryrates case of 2 processors

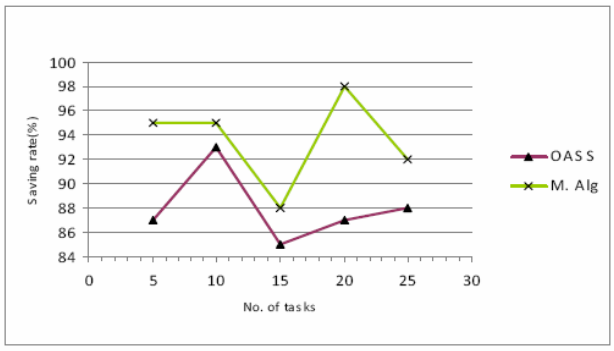


Fig. 7(b) – Memory rates case of 4processors

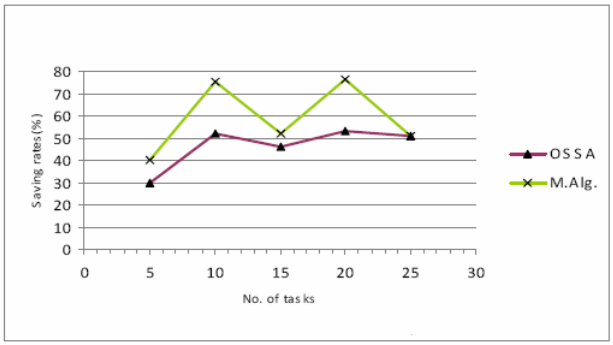


Fig. 7(c) – Memory rates case of 8 processors

Figure (8) presents simulation results of node generated from the previous three algorithms on 4 processor nodes taking into account the memory constrains, if it is included or excluded.

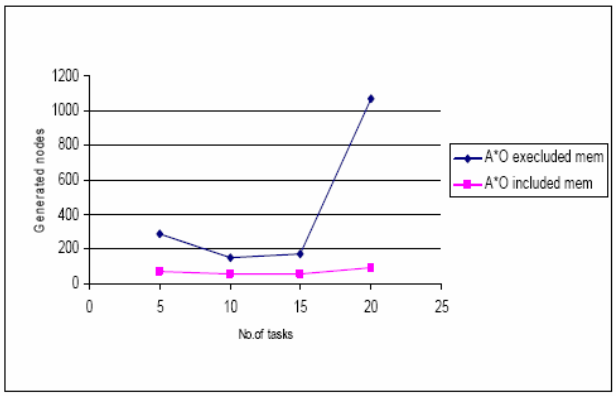


Fig. 8(a) – Simulation results of containing memory conditions of A*O

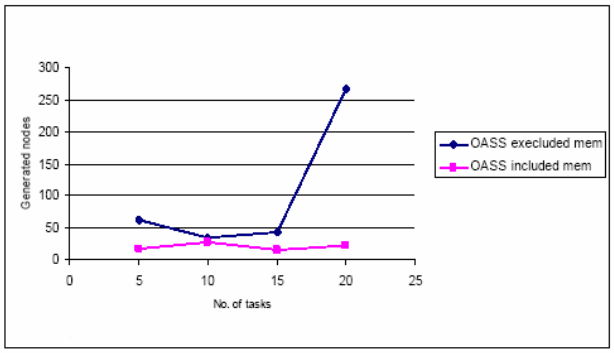


Fig. 8(b) – Simulation results of containing memory conditions of OASS

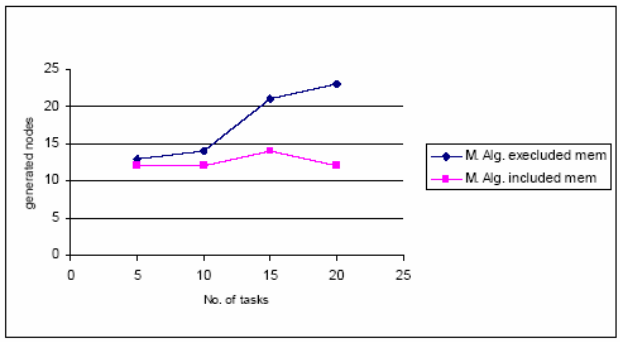


Fig. 8(c) – Simulation results of containing memory conditions of M.Alg.

6. CONCLUSIONS

In this paper, the task allocation problem is studied and a new algorithm for allocating application graphs on to a system of heterogeneous processors is presented. The performance of the proposed algorithm has been investigated in terms of memory efficiency, running time, and saving memory. This has been carried out by using a set of randomly generated application graphs under different conditions like; the communication data rates, the capacities memory of various processors and tasks, and the weight cost of running tasks.

Based on the experimental study, the simulation results show, the proposed algorithm is superior in terms of efficiency and quality in using the memory and also speedup compared with (OASS) which are most important performance measures of evaluating a parallel computer system.

7. REFERENCES

[1] A. K. Tripathi, B. K. Sarker, N. Kumar and D. P. Vidyanhi, "*Multiple Task Allocation with Load Considerations*", Int. Journal of Information and Computing Science (IJICS), Vo1. 3, No, pp.3634, 2000.

[2] Gamal Attiya and Yskandar Hamam, "*Optimal Allocation of Tasks onto Networked Heterogeneous Computers Using Minimax Criterion*", Proceedings of International Network Optimization Conference (INOC,03), pp. 25-30, Evry/Paris, France, 2003.

[3] Gamal Attiya and Yskandar Hamam, "*Hybrid Algorithm for Mapping Parallel Applications in Distributed Systems*", Fifth International Conference on PRAM, Poland, 7-10 September 2003.

[4] Gamal Attiya and Yskandar Hamam, "*Performance Oriented Allocation in Heterogeneous Distributed Systems*", European Simulation and Modeling (ESM 2003) Conference, University of Naples II, Naples, Italy, pp. 27-29 October 2003.

[5] Gamal Attiya and Yskandar Hamam, "*Task Allocation for Maximizing Reliability of Distributed Systems: A simulated Annealing Approach*", J. Parallel Distributed Computer, 66, pp. 1259-1266, 2006.

[6] Haluk Topcuoglu, Salim Hariri, and Min-You Wu, "*Performance- Effective and Low-Complexity Task Scheduling for Heterogeneous Computing*", (IEEE Transactions on Distributed Systems, vol. 13. no. 3 march 2002.

[7] I. Ahmed, Y Kwak, and M.-Y. Wu, "Analysis, *Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors*", 2nd Int'l Symposium on Parallel architectures, Algorithms, and Networks, I-SPAN. Beijing. China. Proc. Of the 1996.

[8] Kamer Kaya a, Bora Ucar a, Cevdet Aykanat, Murat Ikinci, "*Task assignment in heterogeneous computing systems*" J. Parallel Distributed Computers. 66, pp. 32 – 46, (2006).

[9] L. W. Dowdy, E.Rosti, E. Smirni, G. Serazzi, and K. C. Sevcik, "*Processor Saving Scheduling Policies for Multiprocessor Systems*" IEEE Trans. Comput., vol. 47, no. 2, Feb 1998.

[10] M. Kafil, "A*n Optimal Task Assignment Algorithms in Distributed Computing Systems*", IEEE Concurrency, 98, pp 42-49, September 1998.

[11] M. Mezmaz, N. Melab, and E.-G. Talbi, "*An Efficient Load Balancing Strategy for Grid-Based Branch and Bound Algorithm*", Parallel Computing 33, pp. 302-313. February 2007.

[12] S. Woo, S. Yang, S. Kim, and T. Han, "*Task Scheduling in Distributed computing Systems with a Genetic Algorithm*", IEEE, New York, 1997.

[13] Sih, E.A.Lee, "*A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures*", IEEE Trans. Parallel Distributed Systems4, pp. 175-186, February 1993.

[14] W. Boyer, G. Hura, "*A New Min-Span Heuristic Algorithm for Task Scheduling in Heterogeneous Systems*" Proceedings of the Sixth Biennial World conference On Integrated Design and Process Technology, 23. pp. 69, June 2002.

[15] Yskandar Hamam, Khalil S. Hindi, "*Assignment of Program Modules to Processors: A Simulated Annealing Approach*", European Journal of Operational Research 122, pp. 509-513. 2000.

[16] Yu-Kwong Kwok, Ishfaq Ahmad, "*On Multiprocessor Task Scheduling Using Efficient State Space Search Approaches*", J. Parallel Distributed Computing 65, pp. 1515-1532, (2005).

[17] Yi-Wen Zhong, Jian-Gang Yang, and Heng-Nian, "*A Hybrid Genetic Algorithm for Tasks Scheduling in Heterogeneous Computing Systems*" Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, PP. 26-29, August 2004.

[18] Z.-C. Shen and W.-H. Tsai, "*A Graph Matching Approach to Optimal Task Assignment in Distributed Computing System Using a Minimax Criterion*", IEEE Trans. Computers, Vol. C-34, No. 3, Mar., pp. 197–203 1985.

Nirmeen A. Wahab Al Bahnasawy *is a PhD candidate in the Dept. of Computer Science and Engineering, Faculty of electronic Engineering, Menoufia University, Egypt. Her research interests include parallel processing, distributed computing, task allocation and scheduling.*

Mervat Mosa *is PhD in the Dept. of Computer Science and Engineering, Faculty of electronic Engineering, Menoufia University, Egypt. Her research interests include parallel processing and optical computing.*

Gamal M. ATTIYA *graduated in 1993 and obtained his MSc degree in computer science and engineering from the Menufiya University, Egypt, in 1999. He received PhD degree in computer engineering from the University of Marne-La-Vallée, Paris-France, in 2004. He is currently Lecturer at the department of Computer Science and Engineering, Faculty of Electronic Engineering, Menufiya University, Egypt. His main research interests include distributed computing, task allocation and scheduling, computer networks and protocols, congestion control, QoS, and multimedia networking.*

Magdy A. Koutb *received the Eng. Degree from the university of Menofiya, Egypt, in 1977 and M.Sc.degree in 1981 from the university of AlAzhar, Egypt, and the Ph.D. degree from the university of Silesia, Poland in 1985. He has been a Professor since 1997 at the Faculty of Electronic Eng., Menoufiya university. In 2003 he was appointed as Vice-Dean of Post-Graduate Studies and Cultural affairs of the same faculty. He has extensive experience in Computer Engineering and Industrial Electronics. His main research interests include distributed systems, network security and intelligent control systems.*