IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 1, pp. 57-72
ISSN: 1646-3692

AI GAME PLAYING APPROACH FOR FAST PROCESSOR ALLOCATION IN HYPERCUBE SYSTEMS USING VEITCH DIAGRAM

Srinivasan T
Department of Computer Science and Engineering
Sri Venkateswara College of Engineering, Sriperumbudur, India - 602105

Srikanth PJS
Department of Computer Science and Engineering
Sri Venkateswara College of Engineering, Sriperumbudur, India - 602105

Praveen K
Department of Computer Science and Engineering
Sri Venkateswara College of Engineering, Sriperumbudur, India - 602105

Harish Subramaniam L
Department of Computer Science and Engineering
Sri Venkateswara College of Engineering, Sriperumbudur, India - 602105

ABSTRACT

In this journal, we present a method called "AI Game Playing Approach for Fast Processor Allocation in Hypercube Systems using Veitch diagram (AIPA)" which achieves a fast and complete subcube recognition with a complexity that is far less than that of Gray Code (GC), Buddy, Modified Buddy, Modified Gray Code, Free List, Heuristic Processor Allocation (HPA), Tree Collapsing (TC) and other existing allocation policies. The crux of the strategy is to identify a free subcube that can fit the Veitch diagram (also called as the Karnaugh map or K-map). The cells in the Veitch diagram attribute the processors. An AI Game playing approach is applied to ensure optimality along with a graph coloring approach with a resultant penalty factor computation, for effective implementation of the strategy. The algorithm deals with cubic as well as non-cubic allocation and is not only statically optimal but also optimal in a dynamic environment. Extensive performance analysis has been carried out with outcomes discussed comparatively with other allocation strategies. It is shown that our approach supersedes many others in terms of allocation and deallocation costs. The algorithm is also efficient in memory utilization and minimization of system fragmentation. Moreover, the simulation results illustrate that the AIPA strategy significantly improves performance.

KEYWORDS

External / Internal Fragmentation, Graph Coloring, Hypercube, Incomplete subcube, Veitch diagram, Penalty Factor, Processor Allocation / Deallocation, AI Game Playing.

1. INTRODUCTION

With its fascinating properties [i], hypercube topology has been a major appeal for researchers of different fields in recent years. A hypercube, also known as a binary n-cube is a parallel computer with 2^n processors (represented as cells in the Veitch diagram). Each cell corresponds to one of the 2^n vertices of the n-cube. In general, an incoming task is allocated a set of required processors and upon completion, the processors are released for future requests. The former process is ascribed as "allocation" and the latter as "deallocation". Several processor allocation schemes have been proposed in the literature [c], [d], [e], [g], [h], [k], [l], [m], [n].

An attempt to realize complete subcube recognition prevails as a challenging task. A subcube is a subgraph of a hypercube that preserves the properties of the hypercube [i]. Tremendous efforts have been pooled in to minimize "internal fragmentation" [k]. However, this is not the only concern for researchers; there are other critical parameters such as efficient processor utilization, memory overhead etc. Processor allocation and deallocation have become important issues due to the need for efficient utilization of the parameters mentioned above.

One way to enhance processor utilization is to identify all subcubes that are available in the system. Internal fragmentation occurs when the processor allocation scheme allocates more number of processors than what is requested. External fragmentation [c] on the other hand, occurs when a subcube large enough to house the incoming task may not be satisfied since the processors are scattered as subcubes of lower dimension.

In our proposed scheme, the Veitch diagram [b] is used to represent the hypercube system in an attempt to lessen memory overhead. Incomplete subcubes are maintained as a catalog of colors. Each incomplete subcube asserts a color that is not the same as that of its neighbors.

Performance analysis reflects that the proposed scheme surpasses the existing strategies in terms of cost incurred in allocation, deallocation and fragmentation at higher workloads.

The rest of the journal is organized as follows. Section II presents the preliminaries. Section III describes our strategy. Section IV delineates the theoretical analysis and performance comparison along with simulation results to contrast the proposed strategy with others. Finally concluding remarks unfold in section V.

2. PRELIMINARIES

We consider a n variable Karnaugh map [b] to represent a n-cube system where individual nodes or subcubes are represented by a n-bit string of ternary symbols from $\Sigma=\{0,1,X\}$ where X denotes “don’t care” [a]. For example, in a 3-cube system (Figure 1), 0XX denotes the nodes 000,001,010,011 and XXX denotes all the eight nodes.

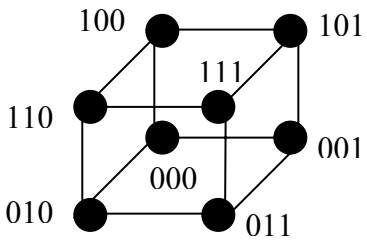


Figure 1. 3-cube system.

DEFINITION 1. The **Hamming Distance** between two subcubes $a = a_1a_2\ldots a_n$ and $b = b_1b_2\ldots b_n$; where $a_i \varepsilon \Sigma$ and $b_i \varepsilon \Sigma \; \forall \; i \varepsilon [1, n]$; can be defined as $H(a,b) = \sum_{i=1}^{n} h(a_i, b_i) = 1$ if $a_i \neq b_i$ and $a_i, b_i \varepsilon \{0, 1\}$, and 0 otherwise. For example, H(00x, 1xx) = 1 and H(1x0, x0x) = 0.

DEFINITION 2. The **Exact Distance** between the two subcubes a and b above, can be defined as $E(a, b) = \sum_{i=1}^{n} e(a_i, b_i)$ where $e(a_i, b_i) = 0$ if $a_i = b_i$ and 1 otherwise. For example, E(0x1, 10x) = 3 whereas H(0x1, 10x) = 1.

DEFINITION 3. An **Incomplete Subcube** (ISC) [d] S can be defined as follows:

1) It consists of a group of disjoint subcubes $\{S_1, S_2, \ldots, S_m\}$, $(1 \leq m \leq n)$ with dimensions $d_1, d_2, \ldots, d_m$, respectively.
2) $H(S_i, S_j) = 1 \; \forall \; 1 \leq i, j \leq m, i \neq j$.
3) $E(S_i, Sj) = d_i - d_j + 1$ for all $1 \leq i \leq j \leq m$.

d_i is the dimension and $d = \sum_{i=0}^{m} 2^{di}$ the size of ISC S. S_1 is called the head of the ICS S.

DEFINITION 4. The **Karnaugh map (K-map)** is a matrix of cells (squares). Each square represents a minterm and in this journal, they refer to a processor in a hypercube. Combination of adjacent cells represents a subcube. The address of the adjacent processors (cells) in the K-map differs exactly by 1 bit [b].

EXAMPLE 1. (Figure 2 and Figure 3) Consider a 3-cube system. This is represented in K-map as a 2 x 4 matrix. Processors X and Y are adjacent and thus differ by 1 bit.

	00	01	11	10
0	000	Y	X	010
1	100	101	111	101

Figure 2. Karnaugh map (K-map) representation.

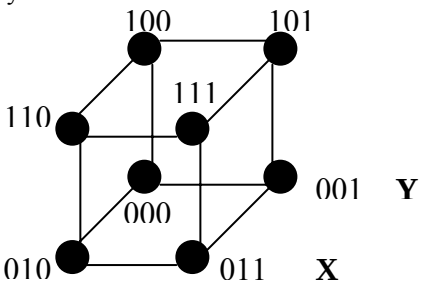


Figure 3. 3-cube system showing adjacency.

DEFINITION 5. **Internal Fragmentation** occurs in case of cubic allocation when the actual required processors for the task are not in the form of 2^k, where k is the task dimension.

DEFINITION 6. **External Fragmentation** occurs when a sufficient number of free processors cannot form an incomplete subcube of the required size. The physical fragmentation problem is similar to that of the memory fragmentation and may result from the sequence of incoming and outgoing tasks or simply a **"bad"** allocation [c].

Fragmentation Due To Bad Allocation **(a)**

No Fragmentation **(b)**

Figure 4. Allocation in a 2-cube system.

EXAMPLE 2. Consider the 2-cube system shown in Figure 4. If an incoming task requesting one node each is allocated as {00,11} (Figure 4a) instead of {00,01} (Figure 4b), a subsequent request for a 2-node cannot be allocated.

Figure 5. Tetris Game.

DEFINITION 7. **Graph Coloring**
The Graph Coloring is a technique of assigning same color to all the free adjacent processors in the incomplete free subcube. A different color is assigned to each incomplete subcube.

DEFINITION 8. **Tetris Game**
The AI game built-in is the Tetris Game (Figure 5). In this game, one must fit the falling pieces to form full lines. One can rotate and translate the falling pieces. The game ends when no more pieces can fall i.e. when incomplete lines reach the top of the board.

3. LITERATURE REVIEW

3.1 Free List Strategy

This strategy maintains a list of free subcubes available in the hypercube, with one list for a dimension. The free list consists of n+1 independent lists, where the i^{th} list corresponds to dimension i, for $i \leq 0 \leq n$. The elements in the list are represented by their unique address (a sequence of n ternary symbols). A n-cube is represented as a sequence of n "x"s initially.

An incoming request for dimension k gets allocated by assigning the first element in the list of dimension k i.e., when there is a request for a k-cube, for $k \leq n$, one of the nearest higher dimension subcubes is decomposed from the most significant bit side for finding a k-cube.

Although the allocation steps are simple, the strategy involves a quite complicated deallocation process. Specifically the deallocation process has to do three steps. First, merge the released subcube with any other subcube to form a bigger cube, or guarantee another available cube of the same dimension. Next, it searches all subcubes of newly produced cube and removes them from their corresponding lists. Finally, repeat the first two steps until nothing can be done further.

3.2 Buddy Strategy

In this strategy, 2^n allocation bits are used to keep track of the availability of the nodes in a hypercube of dimension n. A value 0 (1) in the allocation bit indicates the availability (unavailability) of the corresponding node. Typically, this strategy involves a logical representation of hypercube in the form of a binary tree structure. In the binary tree structure, every leaf node represents the processor and every intermediate node attributes the subcube of some dimension of hypercube. The dimension of the subcube decreases as we choose a node in the tree from root to leaf.

The allocation of requested subcube is done by a search made in the tree representation of hypercube. The search initiates from the root and extends till the leaf node. So once a node in the tree for requested dimension is found to be free, its allocation bit is set so as to declare those set of processors to be in allocated state.

The deallocation procedure is simply the reverse of allocation procedure. In the same fashion as that of allocation method, the tree is traversed and respective node's allocated bit is reset.

3.3 Gray Code Strategy

This strategy works in similar fashion to that of Buddy strategy. The difference lies in the way the nomenclature is done to address the processors in the hypercube. The addressing is done based on Gray Code (GC).

3.4 Modified Buddy Strategy

Modified Buddy Strategy eliminates the disadvantage of buddy strategy. Typically, it searches one level further in the tree structure. In other words, if there is no free subcube of dimension, this strategy tries to merge subcube of smaller dimension. This strategy works similar to that of buddy strategy where 2^n allocation bits are used to keep track of the availability of all nodes. An integer α represented bits is regarded as free if $(\alpha^m)^{0\backslash 1}$ and $(\alpha^m)^{1\backslash 1}$ are free. For example, an integer three in two bits, i.e., 11, is free if integers six and seven in 3 bits are free. This notation implies the free subcubes of small dimension.

The allocation scheme is similar to that of Buddy strategy as long as there is a free subcube of requested dimension. If one such does not exists, then unlike Buddy strategy, this strategy traces one level down the tree structure and tries to find two free subcubes at that level. These two subcubes together constitute the subcube of requested dimension. Such processes could be carried out only if the hamming distance between the two subcubes chosen is 1. This process extends until the request is processed or till the tree is traced till the leaf node.

The deallocation procedure is simply the reverse of allocation procedure. The tree is traversed and respective node's allocated bit is reset.

3.5 Tree Collapsing Strategy

Tree collapsing strategy can recognize and assign subcubes based on requested task size instead of any arbitrary size. This strategy has much less complexity than others in generating search space.

This strategy involves collapsing the binary tree representation of a hypercube successively so that the nodes which form a subcube but are distant would be brought close to each other for recognition. The strategy can be implemented efficiently by using right rotating operations on the notations of the set of subcubes corresponding to the nodes at a certain level of binary tree representations.

4. AI GAME PLAYING APPROACH FOR FAST PROCESSORALLOCATION (AIPA)

AI Game Playing approach for Fast Processor Allocation (AIPA) is developed with an idea of bringing about a full recognition among the 2^n processors in the hypercube. It also has a better efficiency than most of the current existing allocation policies. This strategy is applicable for both cubic as well as non-cubic allocation [c]. The allocation algorithm uses a heuristic function [g] to identify the apposite subcube to be allocated and the Tetris Game approach for mapping the tasks onto the K-map. The allocation and deallocation algorithm uses the Graph Coloring technique [f], which keeps track of all the incomplete subcubes with the help of a color table. Each incomplete subcube has a unique color and each processor in that subcube has the same color.

```
Algorithm: Allocation
Input       : Requested No. of Processors, D.
Output        : Allocates required number of cells in the K-map for the
              requested task.
Variables     : Color Table, CT; Task, T; Color Table Pointer, CTptr;
              Penalty Factor of Ti, PFi; Task Shape, TSi 1 ≤ i ≤ v where v
              is the number of possible task shapes.
Method :
Begin
              For each CTptr in CT begin
                     For I = 1 to v begin
                            PFi = Trace (CTptr, CTptr, TSi)
                            Choose (First_minval (PFi))
                         Proc_allocated = 1 ∀ Allocated Processors
                     Reset (Visited Flags)
                     End for
              Coloring ( )
              End for
End
```

The allocation algorithm is used to allocate a set of processors to the given task. It takes care of both cubic as well as non-cubic tasks. The **"Trace"** function moves the Task Shape TS_i over the Karnaugh map K using the Tetris movement. The Task Shape TS_i with minimum Penalty Factor is chosen. If more than one task shape has the same value, then the one with a minimum height along with an additional priority of minimum width on placement in the K-map is chosen. The **"Coloring"** function assigns a color to all unallocated and uncolored processors in the hypercube.

```
Algorithm: Deallocation
Input         : Subcube, SCi, 1 ≤ i ≤ n where n is the number of
incomplete
               subcubes.
Output        : Nil
Variables     : Processor in Subcube SCi, PE; Address, A; Processor,
Pi;
               Width of Pi, Wi; Color Table CT; Height of Pi, Hi ; Color,
               C; Color stack CS;
Method :
Begin
              For each PE in SCi begin
                     Alloc-Status (PE) = False

              End for
             For each C in CT begin
                    Push (C, CS)
              End for
              Coloring ( )
             For each Pi in hypercube begin
                   Recompute Hi & Wi of Pi

                     Hval (Pi)
              End for
End
```

In the deallocation algorithm, the height of each processor is the number of free continuous processors, which are vertically above this processor. The width of each processor is the number of free continuous processors, which are horizontally to the right of this processor. The Color Table keeps track of the colors in the K-map. Once the processors are freed, the "**Coloring**" function will be invoked.

4.1 Example

Assume at a given instance, tasks T1, T2, T3, T4, T5, T6, T7, T8 are allocated as shown in the Figure 6. A Color Table has a pointer to the incomplete subcubes in this 6-cube system represented by the two colors.

	000	001	011	010	110	111	101	100
000	T1	3	3	T7	T7	3	4	3
001	T1	3	3	T7	T7	3	4	3
011	T1	3	4	2	T6	3	4	3
010	T1	3	4	3	T6	T6	2	T8
110	1	4	4	4	1	T5	2	T8
111	T4	3	4	3	T4	T5	2	T8
101	1	4	4	3	T4	2	3	T8
100	T3	3	4	3	T4	3	4	2

Color Table
Blue
Pink

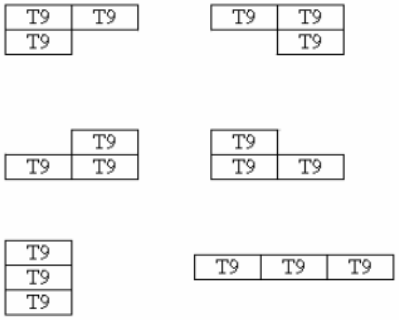


Figure 6. A 6-cube system.

Figure 7. Possible task shapes for a task size of 3 (non-cubic).

Suppose now, if a task of size 3 is requested, all possible task shapes for these 3 processors are generated. (Figure 7). Traversal through the free processors is shown in the Figure 6 by dashed lines. Each task shape is made to fit during the traversal along the dashed lines. The task shape that fits with a minimum penalty factor is then considered finally. In this case, upon traversal, the minimum penalty factor is found to be 6 (2+2+2) at the address {X10101, 111101} (Figure 8). The deallocation algorithm involves releasing of processors by accepting the address of any one of them. Now consider a K-map as follows. In this case five tasks have already been allocated (Figure 9). The freed processors are assigned colors using graph-coloring scheme. Since an incomplete subcube {0X1X} is not adjacent to the incomplete subcube {110X, 1001}, each of them is assigned a different color.

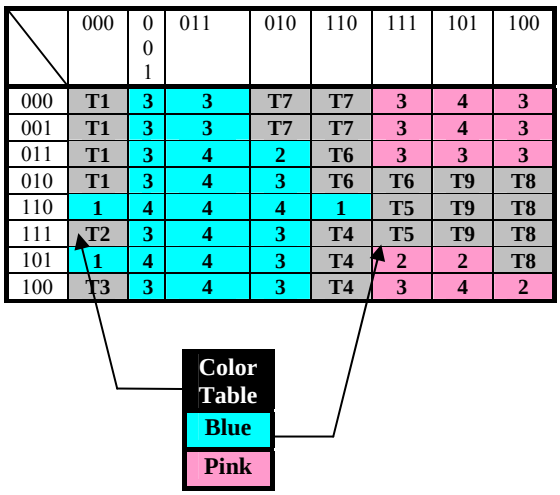


Figure 8. K-map representation consisting of 5 tasks.

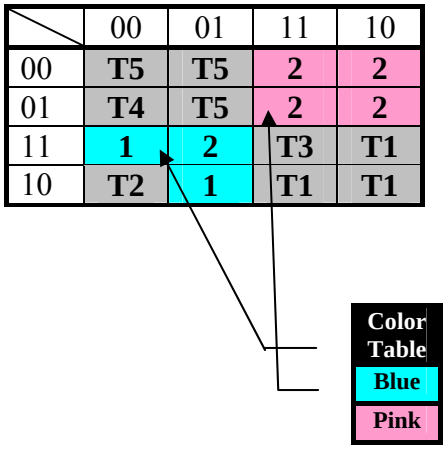


Figure 9. After allocation of task 9.

STEP 1: RESET THE ALLOCATION FLAG TO Nil

Upon the deallocation of task 4, the processor at 0100 is deallocated by setting its allocation flag to nil.

STEP 2: COLORING

Reset all the deallocated processors, by setting allocated flag to nil and calling coloring function, which colors the k-map. If there are two incomplete subcubes then K-map on coloring has two colors. (Figure 10)

STEP 3: HEURISTIC FUNCTION VALUE UPDATES

Compute the Heuristic Value of all the processors adjacent to the deallocated processors.

Updated Heuristic values are shown in the Figure 11.

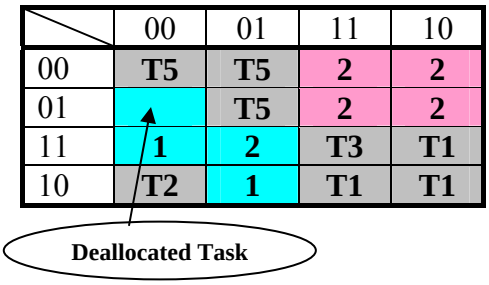


Figure 10. After deallocation of task 4.

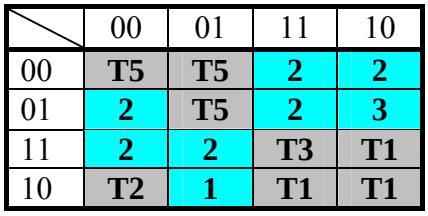


Figure 11. Updated k-map.

5. PARALLEL AIPA

Parallel AI Game Playing approach for Fast Processor Allocation (Parallel AIPA) which is developed with an idea of utilizing all the free processors in the hypercube can be extended to the AIPA strategy, hence bringing down the time complexity to the O(n).

```
Algorithm: Allocation
Begin
            Flood the Veitch Diagram & Task Size to be allocated among
                                                              all processors
           For each Processor in Parallel
                For each Task Shape for a task
                   If task shape can be placed taking Processor Position
                                    as origin in the Veitch Diagram
                           Compute the Penalty Factor
                End For
                Find the Minimum Penalty Factor among these task shapes
               If (Processor is a Client)
                   Receive the Minimum Penalty Factor of all its
                                                            Neighbours
                   Find the Minimum among all these Penalty Factors
                Else If (Processor address is one's complement of Client
                                                              address)
                   Send its Minimum Penalty Factor to its Neighbours
                Else
                   Receive the penalty factor for its children
                   Send the Minimum Penalty Factor among them to its
                                                              Parents
           End For
           Invoke the Coloring Mechanism
           Recompute the Heuristic Value
End
```

```
Algorithm: Deallocation
Begin
            Flood the Task to be deallocated
           Reset the Color Table
           For each Processor in Parallel
                If (Task is executed in the Processor)
                     Allocation_flag = 0;
           End For
           Invoke the Coloring Mechanism
           Recompute the Heuristic Value
End
```

6. PARALLEL AIPA SIMULATION STRATEGY

The simulation process of the Parallel AIPA strategy is divided into four modules namely the Input module, Scheduler module, Allocation strategy module and the Performance Analysis module.

In the Input module, the various workloads consisting of a set of jobs, where a triplet of job defines a job size, hold time and arrival time. The job size (n) is the number of processors

requested by a job to be allocated. This parameter is generated by using one of the four distributions namely Normal, Uniform, Geometric and Hypergeometric. To model practical systems more realistically, we assumed a Poisson job arrival process. A third parameter about hypercube workloads concerns job hold time which is generated using the exponential distribution. In the Scheduler module, the tasks are ordered in the ready queue for execution. We considered a non-preemptive scheduler namely Scan Up. The output of the scheduler module is the scheduled workload. The output of the scheduler module serves as an input to the Allocation strategy module. In this module, processor requests of the scheduled workloads are processed by the Parallel AIPA allocation and deallocation algorithms. In the Performance Analysis module, the performance of the Parallel AIPA strategy is compared with various other existing strategies in terms of system utilization, mean response time, allocation and deallocation time complexities and memory overhead.

6.1 Process Execution

The process execution involves a set of stages. When the tasks arrive, the system is set in ready state. A ready queue is maintained to track the tasks that have reported. Once the tasks are subjected to processing, the system switches to execution state. An execution queue is maintained to list those processes that are under processing. The above issue is described in the following diagram (Figure 12).

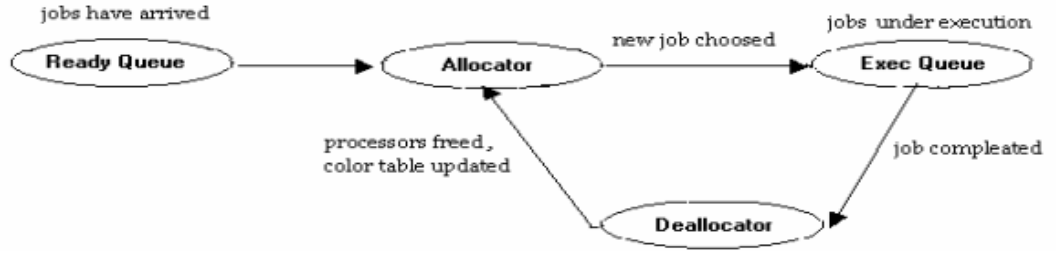


Figure 12. States involved during the execution.

Once the task is serviced, the system calls the deallocator to free the processors involved. Only then, will the next request be serviced upon calling the allocator.

6.2 Simulation Details

Simulation was carried out with the help of a Parallel JAVA [JOPI] simulator. The Java object-passing interface (JOPI) provides an infrastructure for a parallel programming environment in Java. It yields the necessary functionality to write object-passing parallel programs. The main features of JOPI are: (1) It is suitable for clusters and distributed heterogeneous systems, (2) It utilizes the object-oriented programming paradigm for parallel programming thus simplifying the development process.

JOPI provides an MPI- like interface that can be used to exchange objects among processes. Using objects to exchange information is advantageous because it facilitates passing complex structures and enables the programmer to isolate the problem space from the

parallelization problem. Software agents were used to provide the necessary functionality on the participating processors.

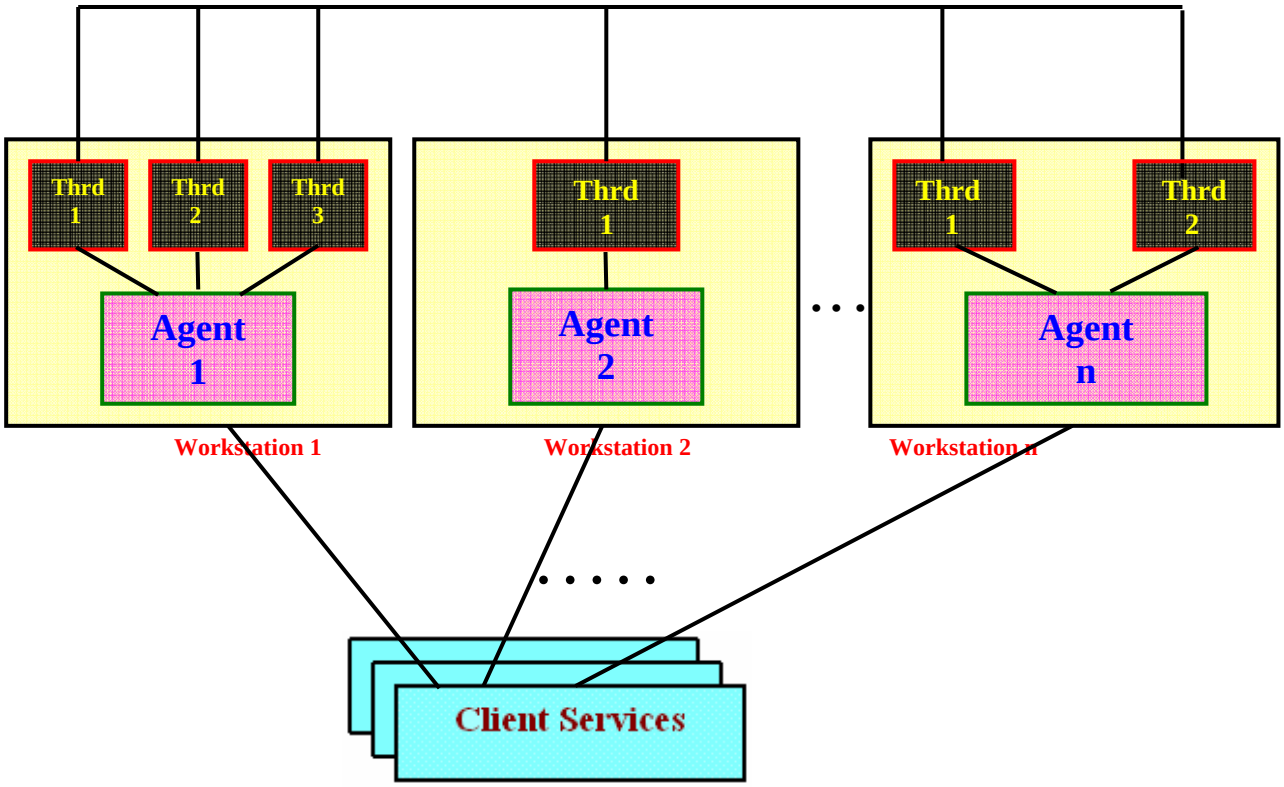


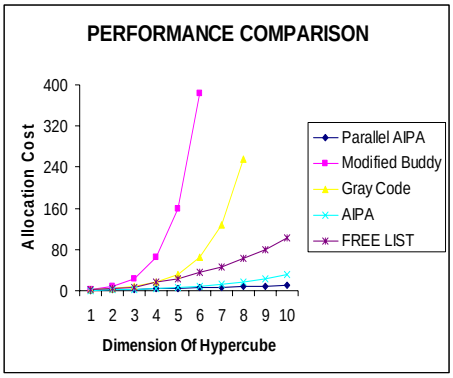
Figure 13. JOPI Run-Time Environment.

This system facilitates high performance computing in Java (parallel Java) for clusters or heterogeneous systems. JOPI class library provides the parallel programming APIs. The environment as shown in Figure 13 helps in automatic scheduling and deploying of parallel JOPI processes on remote machines and JOPI client services provides users with environment monitoring and control commands. Software agents were used to provide the necessary functions that support the JOPI on clusters and distributed environments. Some of the benefits of using agents are: portability, expandability, flexibility, security, and resources management. In addition, the system is written completely in standard Java and can be used on any machine that has a Java virtual machine (JVM).

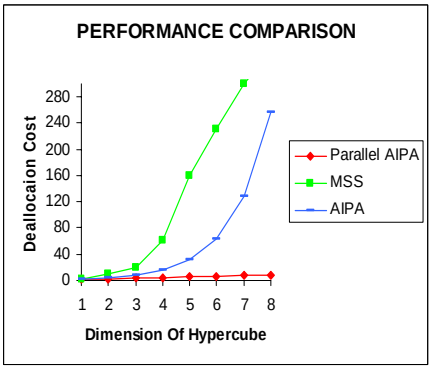
7. PERFORMANCE COMPARISON

The proposed AIPA strategy has the allocation and the deallocation complexities of $O(2^{n/2})$ & $O(2^n)$ respectively and a space complexity of $\Omega(2^n)$ for both allocation as well as deallocation strategy. The Parallel AIPA strategy, however has the allocation and the deallocation

complexities of O(n) & O(n) respectively. Hence the basic difference between these two strategies lies here. The complexities of the two strategies are calculated with the help of [j].



Graph – 1



Graph - 2

According to the Graph-1, the allocation cost of the AIPA strategies is much lesser when compared to most of the other existing strategies.

In the case of deallocation cost, our approach shows a better performance than the MSS [n] as shown in Graph-2. In the above graph, few strategies are not shown, since their cost is very high when compared to our proposed strategy.

Another major performance parameter to be dealt with is fragmentation. The main purpose of Penalty Factor in our approach is to reduce the external fragmentation in the hypercube. On the other hand, the non-cubic allocation [c] that is possible in our strategy eliminates the internal fragmentation. The space complexity of this approach is $\Omega(2^n)$ since the memory required in the AIPA strategies is only with respect to the K-map.

7.1 Simulation Results

Our strategy is compared with others via simulation to verify the performance improvement. Simulation results, however does not consider the strategies that has a greater complexity such as Free List [h] and Modified Buddy [m]. Most of the assumptions used in this simulation are the same as in [m].

Under the simulation conditions defined in [m], the performance of strategies are measured in terms of T_s, E and J, which are averaged over 100 independent runs, and defined as follows [g]:

J : Number of requests that can be satisfied in time interval T

U : Total utilization of processors by requests in time T

$$U = \sum_{i=1}^{J} 2^{|I_i|} t_i$$

where $|I_i|$ is the dimension of requested subcube and t_i is the residence time until T of the request I_i

E : Efficiency of the strategy $\quad E = \frac{U}{2^n T}$

Table 1. Efficiency of processor allocation strategies (E)

Dim	UNIFORM				NORMAL			
	BUDDY	GRAY CODE	HPA	**Parallel AIPA**	BUDDY	GRAY CODE	HPA	**Parallel AIPA**
5	82.42	82.84	83.17	**90.12**	79.85	80.25	80.92	**87.23**
6	81.17	81.14	81.46	**89.67**	79.66	79.70	79.81	**87.12**
7	80.49	80.94	81.43	**89.03**	78.89	78.79	79.33	**86.66**
8	82.25	82.14	82.36	**91.23**	79.83	79.75	80.49	**8.82**
9	82.28	82.31	82.78	**91.84**	78.14	78.02	78.63	**86.13**
10	82.88	83.09	83.18	**92.27**	77.40	77.33	77.82	**85.42**

Table 2. The allocation time (T_s)

Dim	UNIFORM				NORMAL			
	BUDDY	GRAY CODE	HPA	**Parallel AIPA**	BUDDY	GRAY CODE	HPA	**Parallel AIPA**
5	11.2	14.9	10.2	**9.3**	10.4	14.1	10.2	**9.1**
6	16.1	21.7	16.4	**15.2**	15.0	20.7	15.1	**12.6**
7	26.3	36.5	27.4	**23.6**	26.7	34.5	25.5	**21.7**
8	46.9	63.8	46.2	**42.6**	46.5	60.3	43.8	**42.8**
9	89.5	110.9	84.9	**74.5**	83.7	106.2	79.3	**75.3**
10	168.0	212.9	164.1	**149.2**	166.9	213.2	148.9	**146.3**

Table 1 shows that the Parallel AIPA strategy performs better than Buddy [m], Gray Code [m] and HPA [g] strategies in terms of the efficiency. This result comes from the fact that AIPA strategies can recognize more subcubes than the other strategies. It can be observed from Table 2 that the allocation time of the Parallel AIPA strategy in time T is less than the Buddy, Gray Code and HPA schemes.

8. FUTURE WORKS

On further analysis of AIPA Algorithms, it proved to be promising enough to be extended to other two similar interconnection network topologies like 2-D Mesh & 2-D Toroid. The 2-D Mesh topology as shown in the Figure 14 is a generalization of the hypercube, in which there are more than two nodes along a dimension. Each interior processor has 4 adjacent processors and processors along the edges have two to three adjacent processors. The veitch diagram can be replaced by a grid structure (for mesh) whose dimensions are inherited from the 2D Mesh. Each cell in the grid assembles a processor in the mesh.

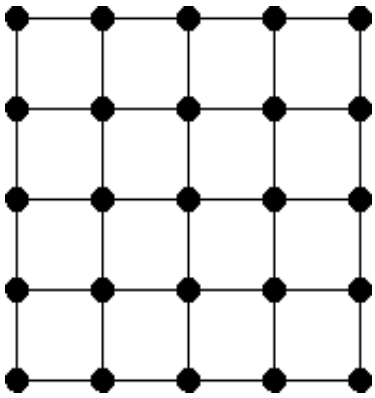


Figure 14. 2-D Mesh.

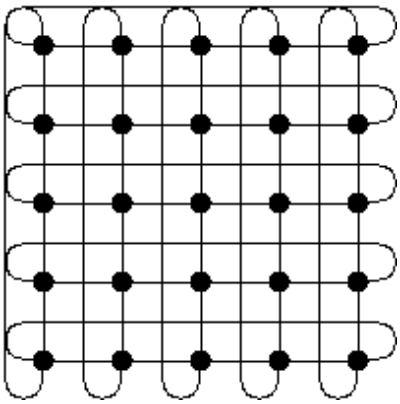


Figure 15. 2-D Toroid.

The 2-D Toroid, on the other hand is merely the topology achieved by wrapping the ends of the mesh. The 2-D Toroid resembles the veitch diagram in its structure as shown in the Figure 15 but varies in its dimension from the conventional toroid. The proposed AIPA strategies can easily be applied to these topologies by redefining the dimensions of the veitch diagram to the corresponding topology. Also the strategies can now be applied for the allocation and deallocation process in both these topologies. However, a further study is required for these topologies in terms of efficiency and performance.

9. CONCLUSION

High performance on a multiprocessor system is achieved by using an efficient processor allocation scheme. To maximize the processor utilization and minimize the time between the starting of a task and its completion, processor allocation and deallocation strategies have become important topics. In this journal, a processor allocation scheme based on the AIPA policy has been introduced for the hypercube computers.

Using this policy, a requested set of processors (both cubic and non-cubic) is allocated to the Veitch Diagram [b] that represents the hypercube system by representing the tasks in the form of Tetris Game blocks and trying to fit these blocks into the Diagram. Quick recognition of the incomplete subcubes is accomplished by using Graph Coloring technique [f]. Each incomplete subcube had a unique color. Since our proposed strategy also deals with non-cubic allocation [c], the problem of internal fragmentation is eliminated.

The performance analysis implies that the proposed AIPA policies are better than the existing allocation algorithms on an overall basis. Furthermore, by incorporating temporal parallelism in our strategy, the time complexity is improvised. The simulation results corroborate the statement above, by showing that our proposed strategy gives a better processor utilization compared to the previous bottom-up schemes, such as the Buddy strategy [m], GC strategy [m] and HPA strategy [g]. Moreover, the AIPA strategy has a less search time than the other existing strategies.

REFERENCES

[a] Michael J. Quinn, 1994. *Parallel Computing – Theory and Practice.* 2nd ed. Singapore: McGraw-Hill.

[b] M.Morris Mano, 2001. *Digital Logic and Computer Design.* Twenty Fourth Reprint Prentice-Hall, Inc., Englewood Cliffs, N.J., USA.

[c] Debendra Das Sharma and Dhiraj K. Pradhan, 1995. Processor Allocation in Hypercube Multicomputers: Fast and Efficient Strategies for Cubic and Non-cubic Allocation. *In IEEE Trans. n Parallel and Distributed Systems*, vol. 6, No.10.

[d] P.O.-Jen and Nian-Feng Tzeng,1992. Fast Recognition-Complete Processor Allocation Strategy for Hypercube computers. *In IEEE Trans. on Computers*, vol. 41, No.4.

[e] Suresh Rai. et al, 1995. Processor Allocation in Hypercube Multiprocessors. *In IEEE Trans. on Parallel and Distributed Systems*, vol. 6, No.6.

[f] Narsingh Deo. Graph Theory with Applications to Engineering and Computer Science. Fifth Reprint Prentice-Hall, Inc.

[g] S.Y.Yoon. et al, 1991. Heuristic Processor Allocation Strategy in Hypercube Systems. *In IEEE Trans. on Parallel and Distributed Systems.*

[h] Jong Kim. et al, 1991. A Top-Down Processor Allocation Scheme for Hypercube Computers. *In IEEE Trans. On Parallel and Distributed Systems,* vol. 2, No.1.

[i] Y.Saad and M.H Schultz, 1988. Topological properties of Hypercubes. *In IEEE Trans. on Computers,* vol 37, pp. 867-872.

[j] Thomas H.Cormen. et al, 2002. *Introduction to Algorithms*, 2nd Edition, Prentice Hall of India.

[k] Moonsoo Kang. et al, 2003. Isomorphic strategy for processor allocation in k-ary n-cube systems. *In IEEE Trans. On Computers,* Vol. 52, Issue: 5, pp. 645 – 657.

[l] B. Bose, A.Al-Dhelaan, 1989. A new strategy for processor allocation in an N-cube multiprocessor. *Proc. Int'l Phoenix Conf. Computing Comm*., pp.114-118.

[m] M.S. Chen and K.G. Shin, 1987. Processor allocation in an N-cube multiprocessor using Gray codes. *In IEEE Trans. on Computers,* Vol. C-36, pp.1396-1407.

[n] S. Dutt and J.P. Hayes, 1991. Subcube allocation in hypercube computers. *In IEEE Trans. on Computers*, Vol. C-40, pp.341-351.

[o] Srinivasan T. et al, 2004. Tetris-Mapping for Processor Allocation in Hypercube Computers using Veitch Diagram (TMPA). *Proceedings of Asia Pacific Conference On Parallel & Distributed Computing Technologies - ObCom 2004.* Vellore, India, pp. 655-672.

[p] Srinivasan T. et al, 2005. AI Game Playing Approach for Fast Processor Allocation in Hypercube Systems using Veitch diagram. *IADIS International Conference on Applied Computing 2005.* Algarve, Portugal, pp. 65-72.

[q] Srinivasan T. et al, 2005. Parallel AI Game Playing Approach for Faster Processor Allocation in Hypercube Systems using Veitch diagram. *11th IEEE International Conference on Parallel and Distributed Systems - ICPADS 2005, IEEE Computer Society.* Fukuoka, Japan, pp. 536-542.