AI Game Playing Approach for Fast Processor Allocation in Hypercube Systems using Veitch diagram

ABSTRACT
This paper proposes 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 much less complexity than the other existing allocation policies such as Buddy, Gray Code (GC), Modified Buddy, Modified Gray Code, Free List, Heuristic Processor Allocation (HPA) and Tree Collapsing (TC). The basic idea of our strategy is to find out a free subcube that can fit the Veitch diagram also called as the Karnaugh map (K-map). This is aided with an AI Game playing approach to ensure optimality. The cells in the Veitch diagram attribute the processors. The proposed strategy can be implemented efficiently by using a graph coloring approach with a resultant penalty factor computation. It deals with cubic as well as non-cubic allocation and is not only statically optimal but also optimal in a dynamic environment. Extensive performance analyses were carried out and outcomes are discussed comparatively with other allocation strategies. It is shown that our approach proves to be better in allocation and deallocation costs. The algorithm also utilizes memory efficiently and minimizes the system fragmentation. Simulation results illustrate that the AIPA significantly improves the performance.

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

1. INTRODUCTION

Baring copious fascinating properties [i], hypercube topology has been a major appeal for the 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 requiring a set of processors is allocated and upon completion, these processors are now released for future requests. Former process is ascribed as "allocation" and 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 is prevailing as a challenging task. A subcube is a subgraph of a hypercube that preserves the properties of the hypercube [i]. Major effort has been pinned on to minimize "internal fragmentation" [k]. However, this is not the only focus of researchers but there are as well other major parameters such as efficient processor utilization, memory overhead etc. The processor allocation and deallocation have become important topics due to the above-mentioned parameters that has to be utilized efficiently.

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.

The Veitch diagram [b] is used to represent the hypercube system in our proposed scheme. Representation of this kind leads to low 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. The proposed algorithm in this paper shows complexities of $O(2^{n/2})$ for allocation and $O(2^n)$ for deallocation.

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

The rest of the paper is organized as follows. Section II gives the preliminaries about the paper. 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 are given 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.

![3-cube system](image1)

Figure 1. 3-cube system.

DEFINITION 1. The Hamming Distance between two subcubes $a = a_1a_2…a_n$ and $b = b_1b_2…b_n$; where $a_i, b_i \in \Sigma$ and $\Sigma$ $\forall$ $i \in [1, n]$; can be defined as $H(a,b) = \sum_{i=1}^{n} h(a_i, b_i) = 1$ if $a_i \neq b_i$ and $a_i, b_i \in \{0, 1\}$, and 0 otherwise. For example, $H(0x1, 10x) = 1$ whereas $H(0x1, 10x) = 0$.

DEFINITION 2. The Exact Distance between the two subcubes $a$ and $b$ above, can be defined as $E(a, b) = \sum_{i} e(a_i, b_i)$ 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 $\{S_1, S_2, …, S_m\}$, $(1 \leq m \leq n)$ with dimensions $d_1, d_2, …, d_m$ respectively.

2) $H(S_i, S_j) = 1 \forall 1 \leq i, j \leq m, i \neq j.$
3) \( E(S_i, S_j) = d_i - d_j + 1 \) for all \( 1 \leq i \leq j \leq m \).

\( d_i \) is the dimension and \( d = \sum_{i=0}^{m} 2^d_i \) 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 paper 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.

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>Y</td>
<td>X</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>101</td>
<td>111</td>
<td>101</td>
</tr>
</tbody>
</table>

![Figure 2. Karnaugh map (K-map) representation.](image)

![Figure 3. 3-cube system showing adjacency.](image)

**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].

**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 4. Allocation in a 2-cube system.](image)

**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. 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. AI GAME PLAYING APPROACH FOR FAST PROCESSOR ALLOCATION

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 appropriate 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**

<table>
<thead>
<tr>
<th>Input</th>
<th>Requested No. of Processors, D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Allocates required number of cells in the K-map for the requested task</td>
</tr>
<tr>
<td>Variables</td>
<td>Color Table, CT; Task, T; Color Table Pointer, CTptr; Penalty Factor of T, PF; Task Shape, TS 1 ≤ i ≤ v where v is the number of possible task shapes.</td>
</tr>
<tr>
<td>Method</td>
<td>Begin</td>
</tr>
<tr>
<td></td>
<td>For each CTptr in CT begin</td>
</tr>
<tr>
<td></td>
<td>For i=1 to v begin</td>
</tr>
<tr>
<td></td>
<td>PF = Trace (CTptr, CTptr, TS_i)</td>
</tr>
<tr>
<td></td>
<td>Choose (First_minval (PF_i))</td>
</tr>
<tr>
<td></td>
<td>Proc_allocated = 1 ∨ Allocated Processors</td>
</tr>
<tr>
<td></td>
<td>Reset (Visited Flags)</td>
</tr>
<tr>
<td></td>
<td>End for</td>
</tr>
<tr>
<td></td>
<td>Coloring ( )</td>
</tr>
<tr>
<td></td>
<td>End for</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
</tbody>
</table>

This function is used to allocate a set of processors to the given task. This algorithm 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**

<table>
<thead>
<tr>
<th>Input</th>
<th>Subcube, SC_i, 1 ≤ i ≤ n where n is the number of incomplete subcubes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Nil</td>
</tr>
<tr>
<td>Variables</td>
<td>Processor in Subcube SC_i, PE; Address, A; Processor, Pi; Width of Pi, W_i; Color Table CT; Height of Pi, H_i; Color, C; Color stack CS</td>
</tr>
<tr>
<td>Method</td>
<td>Begin</td>
</tr>
<tr>
<td></td>
<td>For each PE in SC_i begin</td>
</tr>
<tr>
<td></td>
<td>Alloc-Status (PE) =False</td>
</tr>
<tr>
<td></td>
<td>End for</td>
</tr>
<tr>
<td></td>
<td>For each C in CT begin</td>
</tr>
<tr>
<td></td>
<td>Push (C, CS)</td>
</tr>
<tr>
<td></td>
<td>End for</td>
</tr>
<tr>
<td></td>
<td>Coloring ( )</td>
</tr>
<tr>
<td></td>
<td>For each Pi in hypercube begin</td>
</tr>
<tr>
<td></td>
<td>Recompute H_i &amp; W_i of Pi</td>
</tr>
<tr>
<td></td>
<td>Hval (Pi)</td>
</tr>
<tr>
<td></td>
<td>End for</td>
</tr>
</tbody>
</table>

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.
**Function: Hval**

**Input:** Processor, \( P_i \); \( 1 \leq i \leq n \) where \( n \) is the number of processors in the hypercube.

**Output:** Heuristic Value, \( HV_i \)

**Method:**

Begin

\[
HV_i = 0
\]

If (Alloc-status (Top (\( P_i \))) = True) then

\[
HV_i = HV_i + 1
\]

End if

If (Alloc-status (Bottom (\( P_i \))) = True) then

\[
HV_i = HV_i + 1
\]

End if

If (Alloc-status (Left (\( P_i \))) = True) then

\[
HV_i = HV_i + 1
\]

End if

If (Alloc-status (Right (\( P_i \))) = True) then

\[
HV_i = HV_i + 1
\]

End if

End

3.1 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 (Blue and Pink).

![Figure 6. A 6-cube system.](image)

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 7 (2+3+2) at the address \{10111X1, 111101\} (Figure 8).

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

![Figure 8. After allocation of task9.](image)

![Figure 9. K-map representation consisting of 5 allocated tasks.](image)
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.

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 latest Heuristic Value at each processor after deallocation. Updated Heuristic values are in the free processors. (Figure 11)

Figure 10. After deallocation of task 4.

![Figure 10. After deallocation of task 4.](image1)

Figure 11. Updated k-map.

![Figure 11. Updated k-map.](image2)

4. PERFORMANCE COMPARISON

The proposed strategy (AIPA) 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 complexities are calculated with the help of [j].

![Graph - 1](image3)

According to the Graph-1, the allocation cost of this algorithm is much lesser when compared to most of the existing strategies. The Buddy List strategy [m], despite having a better allocation cost than the AIPA approach, fails in terms of full recognition whereas the AIPA approach achieves it with optimal results.

In the case of deallocation cost, our approach shows a better performance than the Free List [h], Modified Buddy [m], MSS [n] and other approaches as shown in Graph-2. The deallocation performance of this
method is similar to that of the Tree Collapsing strategy [d]. As per the Graph-2, Buddy List [m] again seems to have better optimal cost for deallocation than AIPA, but as mentioned earlier, it does not achieve full recognition and also has a search limitation of $2^{n-k}$. 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 AIPA is only with respect to the K-map.

4.1 Simulation Results

Our strategy is compared with others via simulation to verify the performance improvement and reduced allocation / deallocation time. 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]:

- $T_s$: Total search time in time interval $T$
- $J$: Number of requests that can be satisfied in time interval $T$
- $U$: Total utilization of processors by requests in time $T$

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

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

$E$: Efficiency of the strategy

$$E = \frac{U}{2^n T}$$

Table 1. Efficiency of processor allocation strategies ($E$).

<table>
<thead>
<tr>
<th>Dim</th>
<th>Uniform</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buddy</td>
<td>Gray</td>
<td>HPA</td>
</tr>
<tr>
<td>5</td>
<td>90.42</td>
<td>90.84</td>
</tr>
<tr>
<td>6</td>
<td>81.17</td>
<td>81.14</td>
</tr>
<tr>
<td>7</td>
<td>80.49</td>
<td>80.94</td>
</tr>
<tr>
<td>8</td>
<td>82.25</td>
<td>82.14</td>
</tr>
<tr>
<td>9</td>
<td>82.28</td>
<td>82.31</td>
</tr>
<tr>
<td>10</td>
<td>82.88</td>
<td>83.09</td>
</tr>
</tbody>
</table>

Table 1 shows that the AIPA strategy performs better than Buddy [m], GC [m] and HPA [g] strategies in terms of the efficiency. This result comes from the fact that AIPA strategy can recognize more subcubes than the other strategies. The results shown in Table 2 represent that the AIPA strategy also serves more input tasks than the other compared strategies.

5. 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 paper, 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 shows that the AIPA has a time complexity of $O(2^{n/2})$ for allocation and $O(2^n)$ for deallocation. The space complexity of our algorithm is also $\Omega(2^n)$. This analysis ultimately implies that the proposed AIPA policy is better than the existing allocation algorithms on an overall basis. In addition, the complexity of the algorithm is less in terms of implementation. 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


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


