Implementation of Processor Allocation in an N-Cube Multiprocessor on Macintosh

by

Jinyue Liu

A research project submitted to Oregon State University in partial fulfillment of the Degree of Master of Science

Department of Computer Science
Oregon State University
Corvallis, Oregon

April, 1990

Abstract

The processor allocation in an n-dimensional hypercube multiprocessor using buddy strategy, gray code strategy, and a new strategy is implemented on Macintosh. Our implementations show that when processor relinquishment is not considered (i.e. static allocation), all these strategies are optimal in the sense that each incoming request sequence is always assigned to a minimal subcube. Our implementations also show that the gray code strategy outperforms the buddy strategy in detecting the availability of subcubes, and furthermore, when some faulty processors are considered or processors are allocated dynamically, the new strategy does better than the buddy strategy or gray code strategy in subcube recognition.

Table of Contents

1. Introduction

This project implements the processor allocation in an N-cube multiprocessor using three strategies, i.e. buddy strategy, gray code strategy[4], and the new strategy proposed by Al-Dhelann and Bose[1].

What is an hypercube? An hypercube is a network of loosely coupled processors connected in such a way that two processors are linked if and only if their binary representation differ in exactly one bit position, i.e. the indices of neighboring processors differ by a power of 2.

A n-dimensional hypercube, denoted as n-cube or Q_n, is a hypercube with 2^n processors and is defined recursively as: A 0-dimensional hypercube, Q_0, is a single processor, and an n-dimensional hypercube is two (n-1)-dimensional hypercube with links between corresponding processors in each of them. Fig.1.1 and Fig.1.2 show a Q_4 and a Q_5, hypercube respectively.

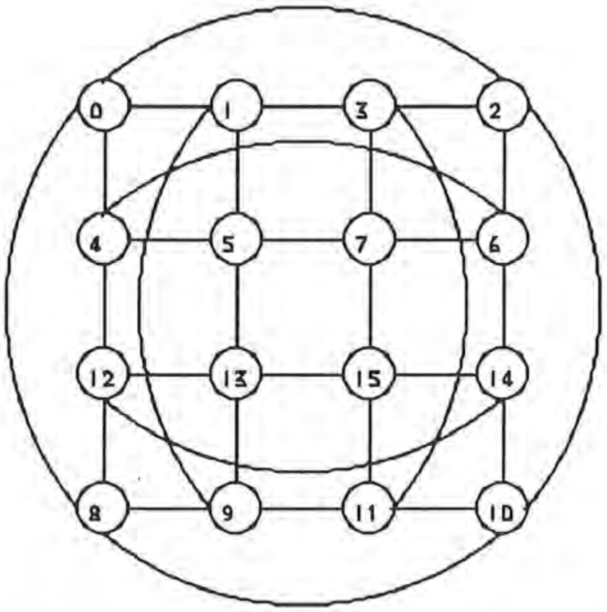


Fig.1.1 4-dimensional hypercube, Q_4

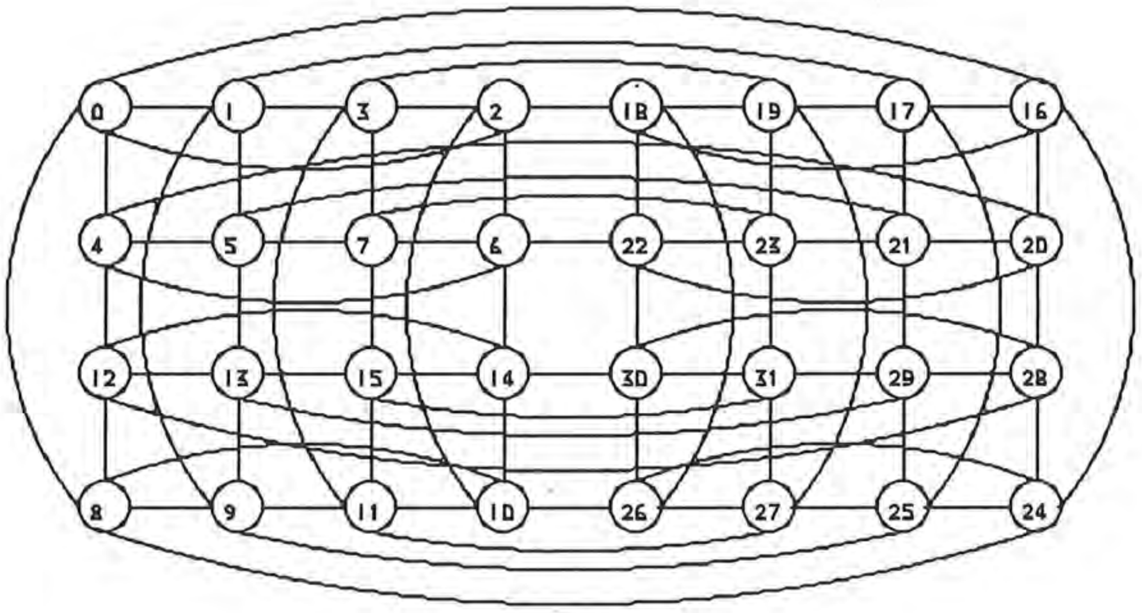


Fig.1.2 5-dimensional hypercube, Q_5

A task arriving at a hypercube multiprocessor --- this is called an "incoming request" --- can be specified in graph form and must be assigned "optimally"(in some sense) to a subcube in the multiprocessor for execution. Upon completion of execution, the subcube used for the task must be released(relinquished or deallocated) for later use. Efficient allocation and/or deallocation of node processors in a hypercube multiprocessor is a key to its performance and utilization. Each incoming task to an n-cube multiprocessor is described by a graph, in which each node denotes a module of the task and each link represents inter-module communication. Each module must be assigned to a subcube so as to preserve node adjacencies in the associated task graph. Thus, processor allocation in an n-cube multiprocessor consists of two sequential steps :

1) Determination of the dimension of the subcube required to accommodate all the modules of each incoming task and allocation of modules of the task to nodes within the subcube.

2) Recognizing and locating each subcube that can accommodate the incoming task.

In this project, we are only concerned the second step in which several processor allocation strategies are used to implement the recognition and location of the subcubes in the n-cube system, assuming that all the incoming task graphs have already been embedded into subcubes[5].

2. Processor Allocation Strategies

In an N-cube multiprocessor, processors must be allocated to incoming tasks in a way that will maximize the processors utilization and minimize the system fragmentation. In order to achieve this goal, it is necessary to detect the availability of a subcube of required size and merge the released small cubes to form a larger ones. Following are the processor allocation strategies to be used in implementation later.

• **Algorithm 1 (Buddy System Strategy)**

Processor Allocation:

Step1. Set *k* to the dimension of a subcube required to accommodate the request.

Step2. Determine the least integer α, $0 \leq \alpha \leq 2^{n-k} - 1$, such that all βth allocation bits, $\alpha 2^k \leq \beta \leq (\alpha+1)2^k - 1$, are 0's. Set all these bits to 1's.

Step3. Allocate processors with addresses $B_n(\beta)$ to the request, where

$$\alpha 2^k \leq \beta \leq (\alpha+1)2^k - 1.$$

Processor Relinquishment:

Reset every *p*th allocation bit to 0, where $B_n(p)$ is used in the subcube released.

Note: $B_n(p)$ above is the binary representation of an integer p with n bits. e.g. $B_3(1)=001$, $B_4(3)=0011$.

- **Algorithm 2 (Gray Code Strategy)**

Processor Allocation:

Step1. Set *k* to the dimension of a subcube required to accommodate the request.

Step2. Determine the least integer a, $0 \leq \alpha \leq 2^{n-k+1} - 1$, such that all ($\beta$ mod 2^n)th allocation bits are 0's, where $\alpha 2^{k-1} \leq \beta \leq (\alpha+2)2^{k-1} - 1$. Set all these bits to 1's.

Step3. Allocate nodes with addresses $G_n(b \bmod 2^n)$ to the request, where

$$\alpha 2^{k-1} \leq \beta \leq (\alpha+2)2^{k-1} - 1.$$

Processor Relinquishment:

Reset every *p*th allocation bit to 0, where $G_n(p)$ is used in the subcube released.

Note: $G_n(m)$ is the BRGC (Binary Reflected Gray Code) representation of m. e.g. $G_4(3) = 0010$. Following formula can be used to translate the gray code to its corresponding binary representation and vice versa:

$g_i = b_i$ **xor** $b_{i+1} \quad i \neq n$

$g_n = b_n$

• **Algorithm 3 (New Strategy)**

Processor Allocation:

Step1. Set k to the dimension of a subcube required to accommodate the request.

Step2. Determine the least integer α, $0 \leq \alpha \leq 2^{n-k+1} - 1$, such that $B_{n-k+1}(\alpha)$ is free and it has a pth, $0 \leq p \leq n-k$, partner $B^p_{n-k-1}(\alpha)$ which is also free. Take p as small as possible.

Step3. Allocate these processors to the request, and set their allocation bits to 1.

Processor Relinquishment:

Reset the allocation bits of all the processors that correspond to the descendants of the nodes $B_{n-k+1}(\alpha)$ and $B^p_{n-k-1}(\alpha)$ to 0.

Note:

1) The αth partner of $a_{k-1}a_{k-2}\ldots a_{\alpha+1}a_{\alpha}a_{\alpha-1}\ldots a_0$ for any $0 \leq \alpha \leq k-1$ is defined as

$a_{k-1}a_{k-2}\ldots a_{\alpha+1}1a_{\alpha-1}\ldots a_0$	if $a_{\alpha} = 0$
undefined	if $a_{\alpha} = 1$

We denote the pth partner of $B_k(i)$ as $B^p_k(i)$.

2) For any integer α, $0 \leq \alpha \leq 2^{n-k+1} - 1$, the node $B_{n-k+1}(\alpha)$ is free if and only if all of its descendants are free. e.g. for $n = 4$ and $k = 2$,

the node 000 is free if and only if the processors 0000 and 0001 are free.

3. User Interface

Fig.3.1 shows the screen for the main menu for the hypercube application environment.

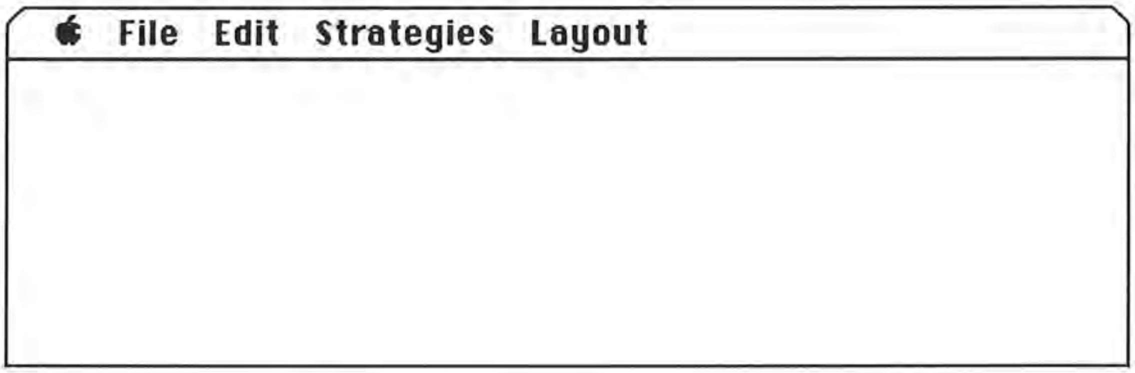


Fig.3.1 The main menu of the Hypercube application

3.1. Apple Menu (Fig.3.2)

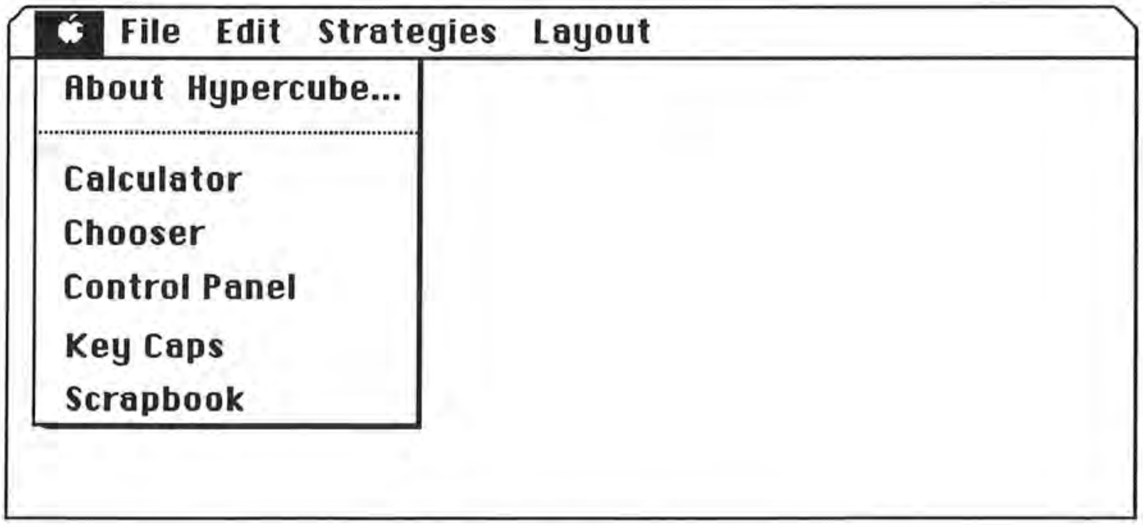


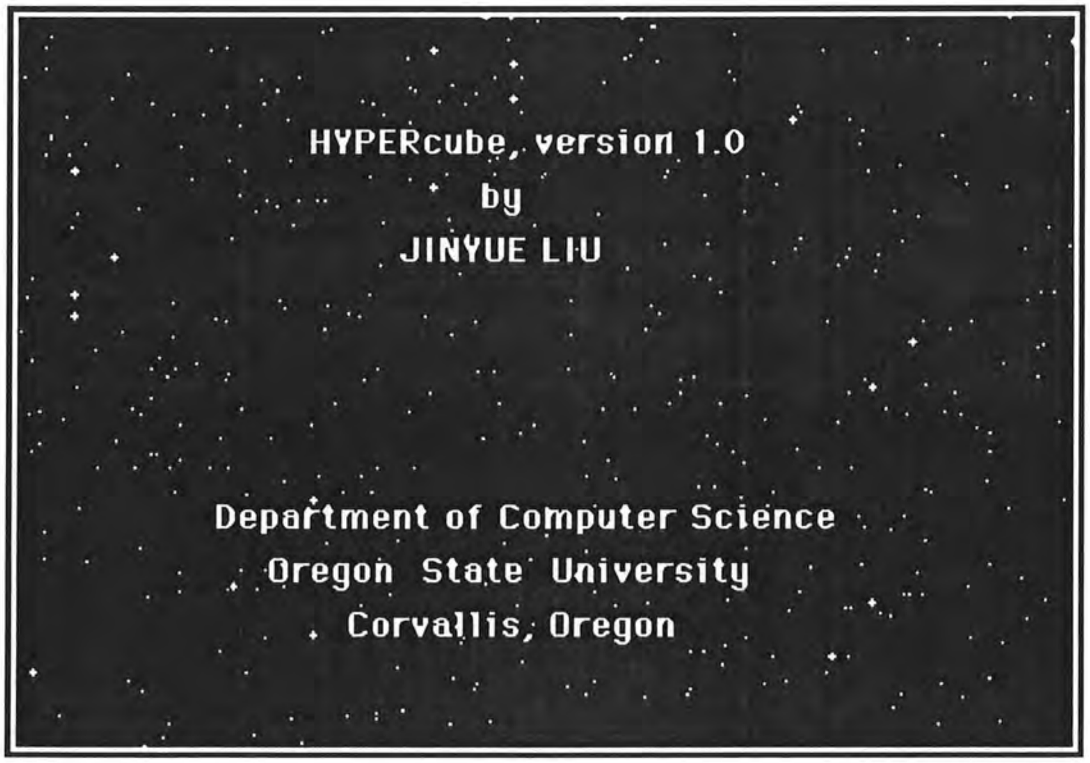
Fig.3.2 Apple menu

• **About Hypercube...**

This item gives an About dialog (Fig.3.3) that displays author information, project information, and version information. The main purpose is to provide user quick information about the application.

• **Desk Accessories**

The other items include desk accessories which depend on your Macintosh system file.



HYPERcube, version 1.0
by
JINYUE LIU

Department of Computer Science
Oregon State University
Corvallis, Oregon

Fig.3.3 About dialog

3.2 File Menu (Fig.3.4)

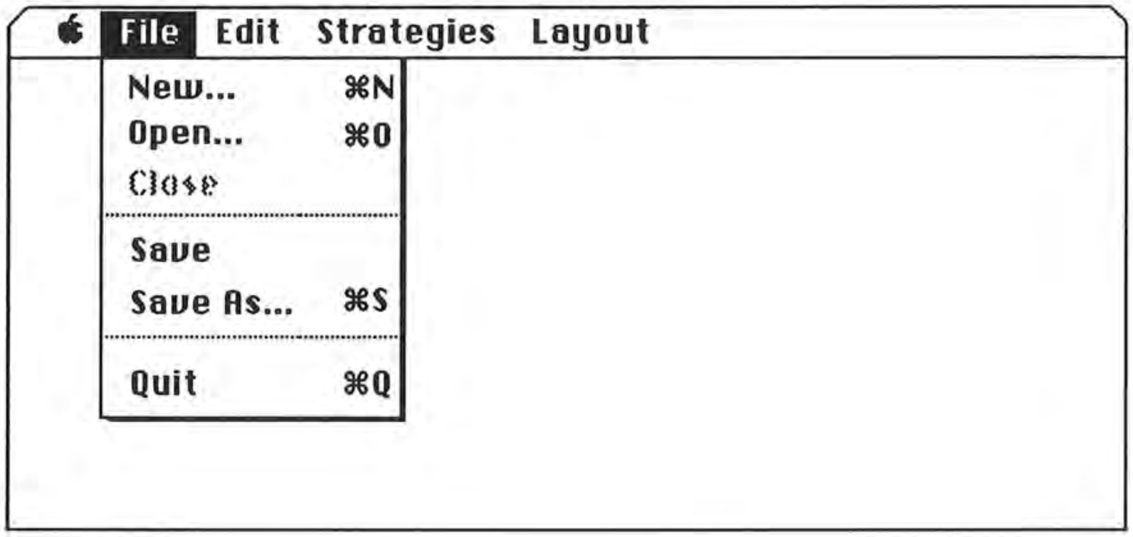


Fig.3.4 File menu

- **New**

  This command is intended to set up a diagram window with a hypercube in it whose size is given by users.

- **Open**

  This command is used to load an already existing diagram in memory.

- **Close**

  This command is used for closing any window that appears on the screen.

- **Save**

  This command is used to save the diagram with its existing name.

- **Save As...**

  This command is used for saving the diagram with another name.

- **Quit**

  This command as usual is used for quiting the application.

3.3 Edit Menu (Fig.3.5)

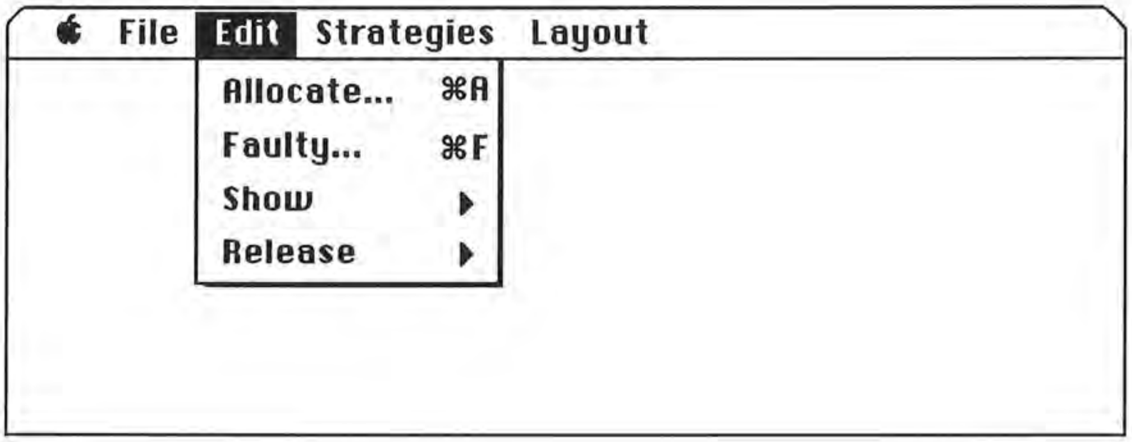


Fig.3.5 Edit menu

- **Allocate...**

  This command allocates the available processors to the incoming task according to some strategy chosen previously by users.

- **Faulty...**

  This command allows users to give the faulty processors.

- **Show**

  This is a submenu. It includes all the existing tasks that can be showed.

- **Release**

  This is a submenu. It includes all the existing tasks that can be released.

3.4 Show Submenu

This menu allows users to show whatever the existing task he wants (Fig.3.6).

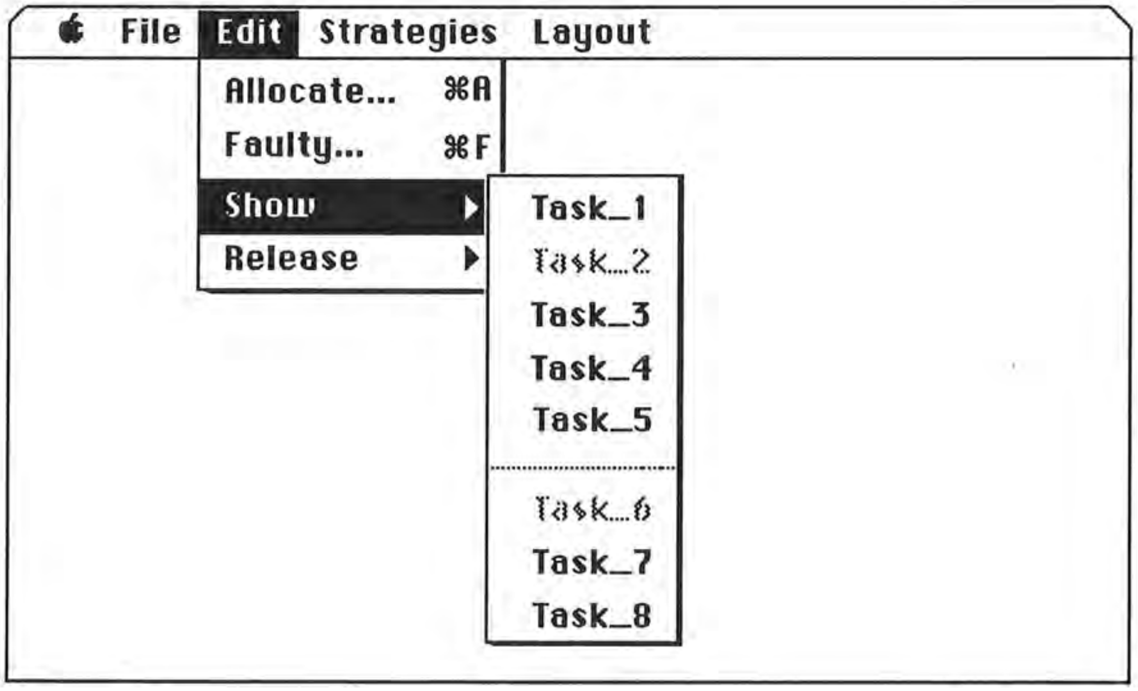


Fig.3.6 Show submenu

Here we use the new menu feature: hierarchical menu, which is used for lists of related items, such as tasks, to keep them simplicity and clarity. A hierarchical menu is a logical extension of the current menu metaphor: another dimension is added to a menu, so that a menu item can be the title of a submenu. When the user drags the pointer through a hierarchical menu item, a submenu appears after a brief delay.

Hierarchical menu items have an indicator (a small black triangle pointing to the right, to indicate "more") at the edge of the menu, as illustrated in Fig.3.6 and Fig.3.7.

3.5 Release Submenu

This command allows users to release whatever the existing task he wants (Fig.3.7). Once a task is released, it can not be showed or released again and the correponding menu item is dimmed(as is Task_2 and Task_6 in Fig.3.7).

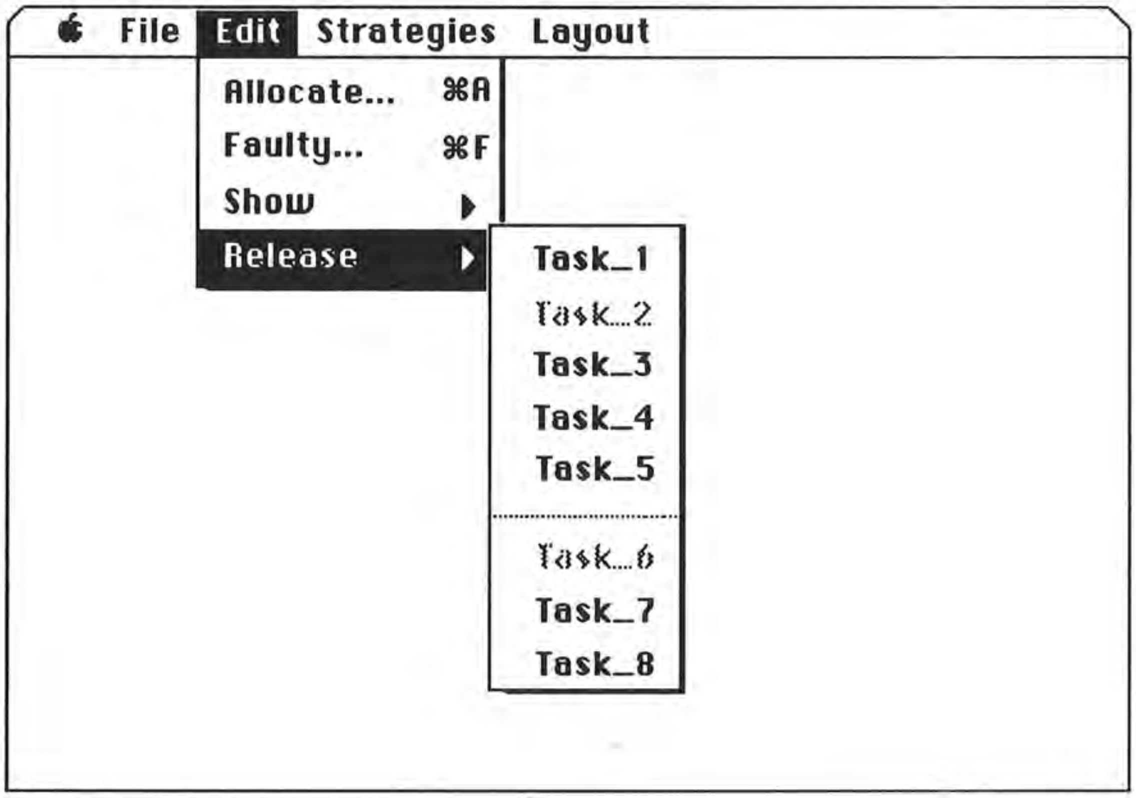


Fig.3.7 Release submenu

3.6 Strategies Menu (Fig.3.8)

This menu consists of three menu items representing three strategies which can be chosen by users.

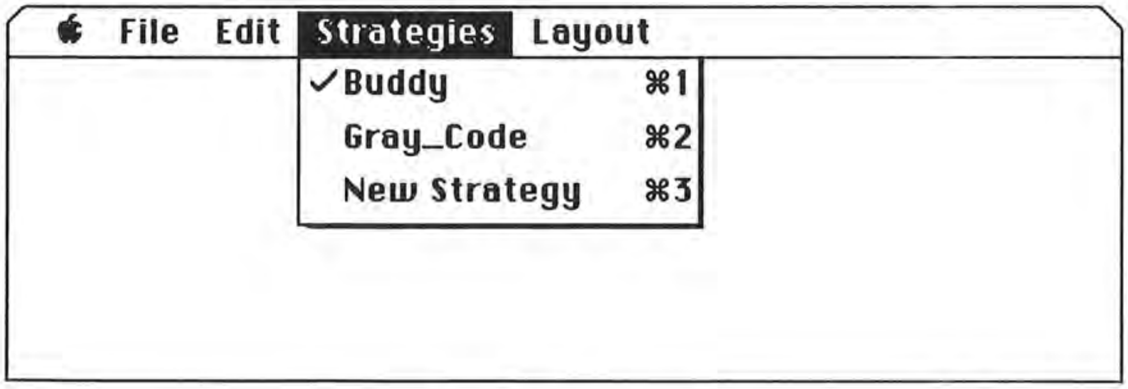


Fig.3.8 Strategies menu

3.7 Layout Menu (Fig.3.9)

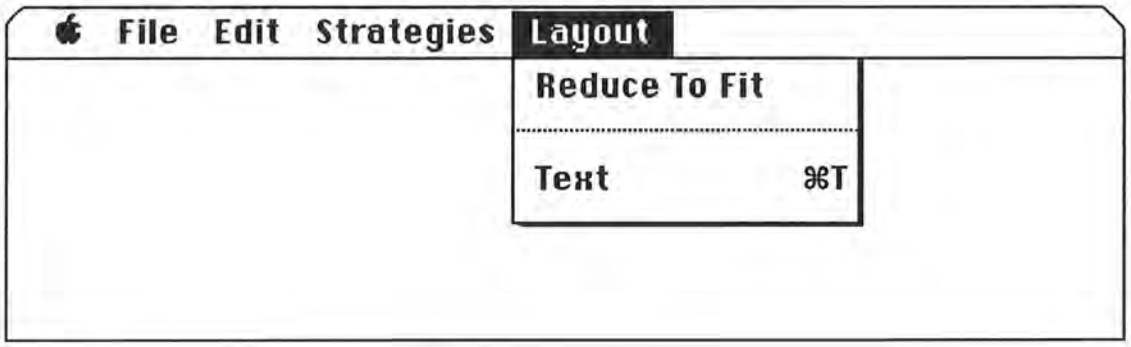


Fig.3.9 Layout menu

- **Reduce To Fit**

  This command is used for viewing entire hypercube in reduced size.

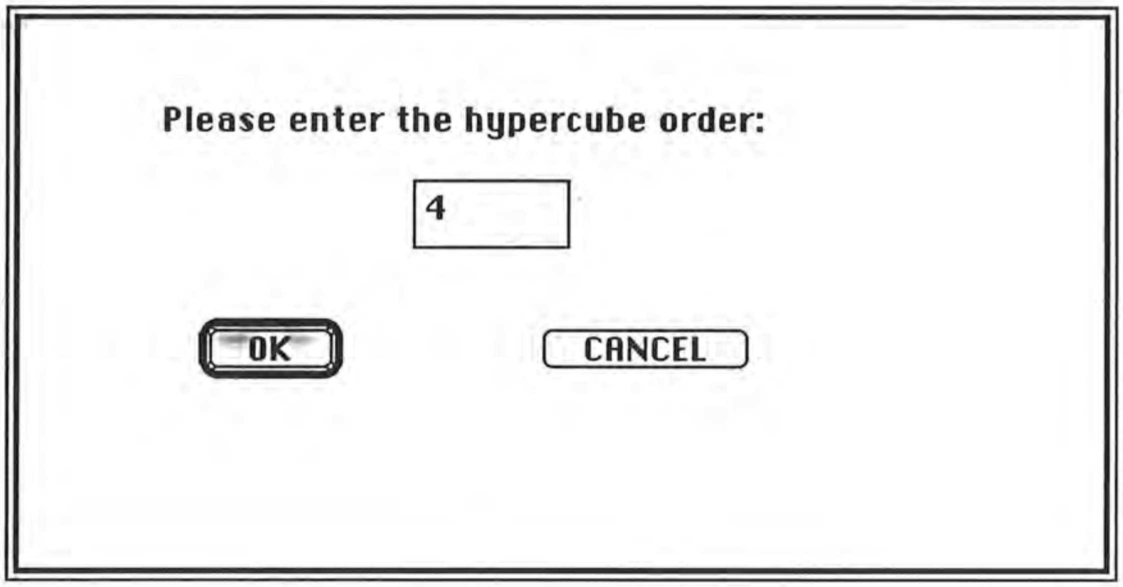
- **Text**

  This command lists all the existing tasks and the faulty processors in binary representation in a text window.

4. Implementation and Implementation Results

4.1. Implementation

The user can create a new hypercube by selecting the New command in File menu. First, a dialog will come up allowing users to give the hypercube order (Fig.4.1).


Please enter the hypercube order:

4

Fig.4.1 Enter hypercube order

After we give the hypercube order and click OK button, a diagram window with hypercube in it will be set up (Fig.4.2).

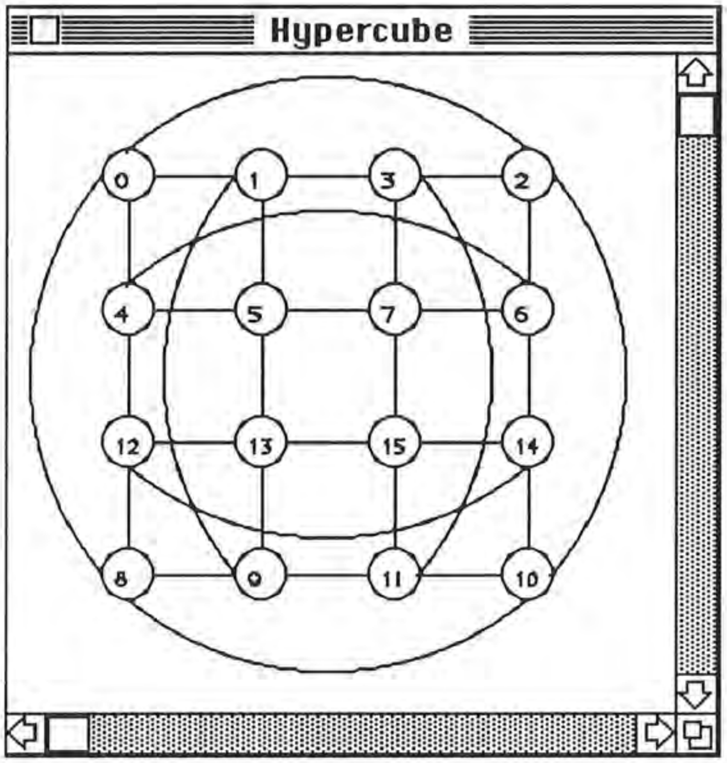
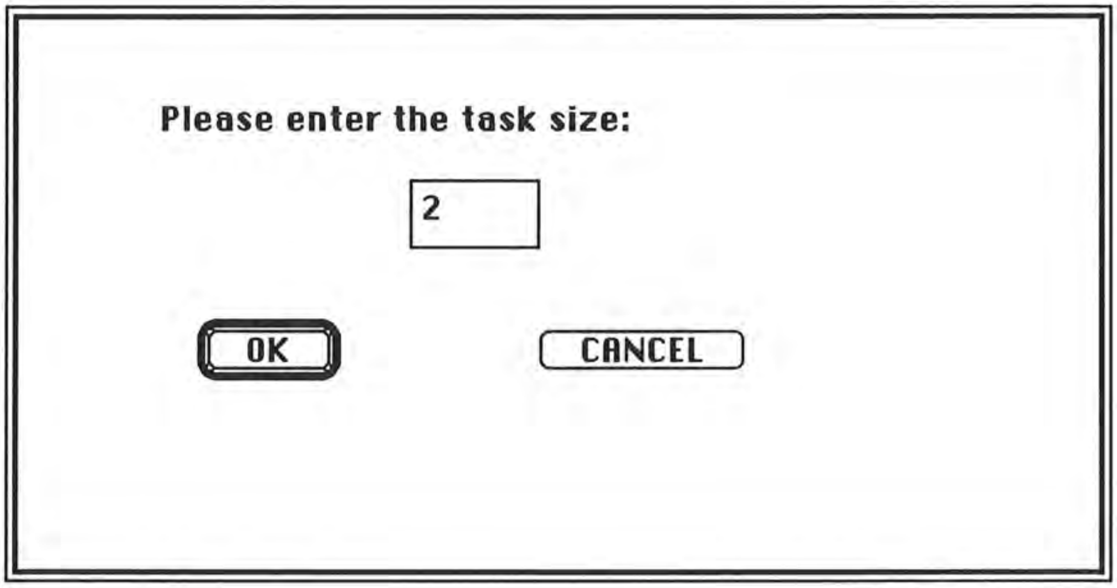


Fig.4.2 4-dimensional hypercube, Q_4

Here each circle stands for a processor and processors connected by line or arc are neighbors. In a two dimensional plane, to draw an n dimension hypercube will result in many line or arc crosses among node connections. In order to reduce the crosses, we design a very special way to arrange the processor location.

The next step is to allocate the processors to the incoming task. This can be achieved by choosing the Allocate command in Edit menu. A dialog will pop up allowing users to enter the task size or subcube order (Fig.4.3). But remember to pick up the strategy you want before you allocate processors. The default strategy will be the buddy strategy when a new hypercube is created. After you enter the task size and click OK button, those processors allocated to this task will be shaded-in (Fig.4.4).


Please enter the task size:

2

Fig.4.3 Enter task size

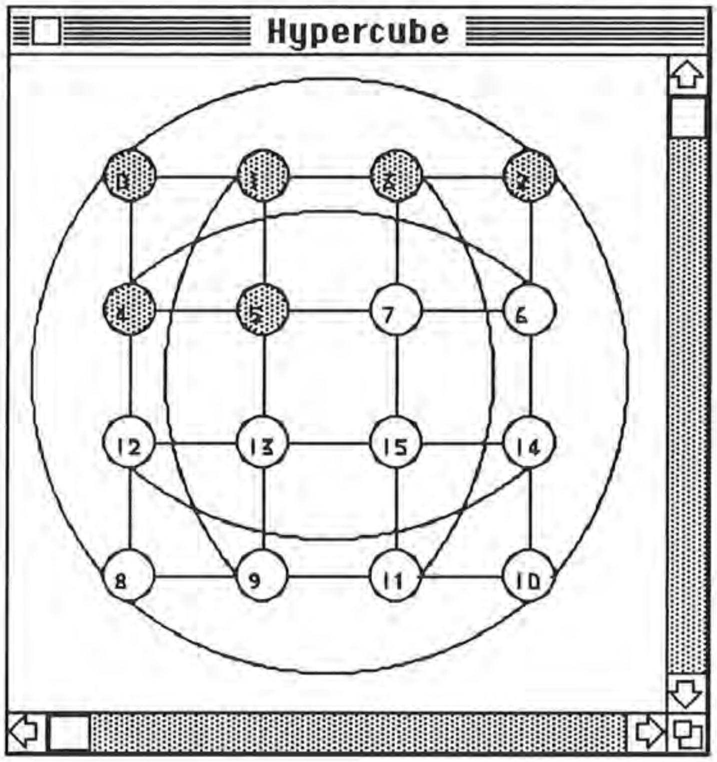


Fig.4.4 Processor allocation with requests $I_1 = Q_2$, $I_2 = Q_1$ using Buddy Strategy

The tasks are given sequentially in **Show** and **Release** submenu. A user can display whatever task he wants by selecting the task in Show submenu. Those processors or nodes belonging to this task we choose will be blinking until the mouse is pressed down. Once the task is finished, these processors belonging to the task may be relingquished. Release submenu can do this by choosing the task in the Release submenu. The processors belonging to the task we choose will be blanked after release and can be allocated to incoming task later.

This application also provides users an opportunity to give some faulty processors. The faulty processor could be assumed by users giving the processor number in a dialog which will be brough up when selecting Faulty command in Edit menu. The faulty processor is represented by a black node and it can be either the busy processor or the idle processor (Fig.4.5). But the faulty processor can not be recovered once it is assumed fault.

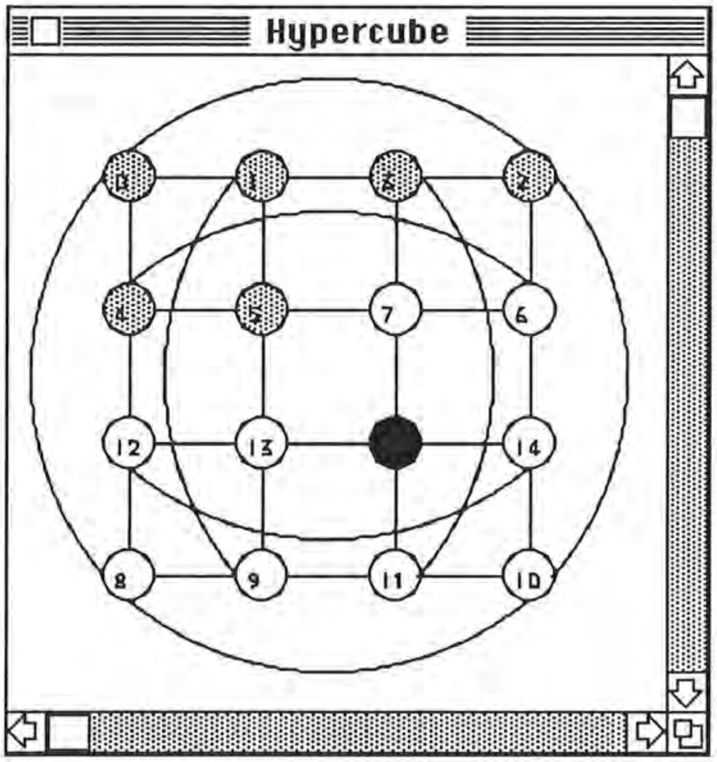


Fig.4.5 Hypercube with a faulty node 15

An entire hypercube can be viewed by the **Redoce To Fit** menu choice available in the Layout menu (Fig.4.6). This is useful especially when the hypercube order is large and the user can not see the entire hypercube on the screen in the normal size. The application also provides the user with a text window that shows all the existing tasks with the processors represented in binary and all faulty processors (Fig.4.7). The text window can be opened by selecting the text item in the layout menu and can be hidden by clicking the close box. The contens of the text window is keeping updated automatically every time the diagram is changed.

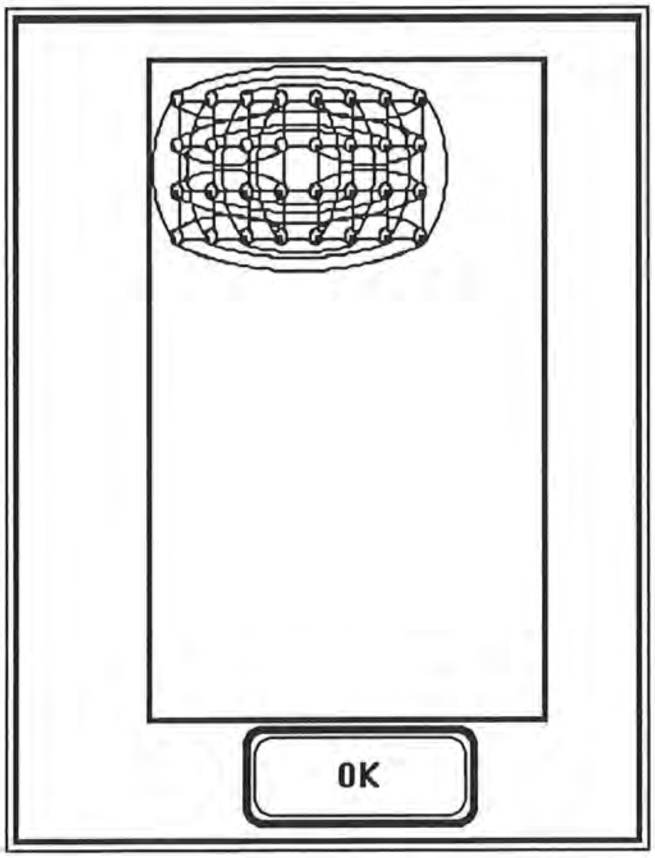
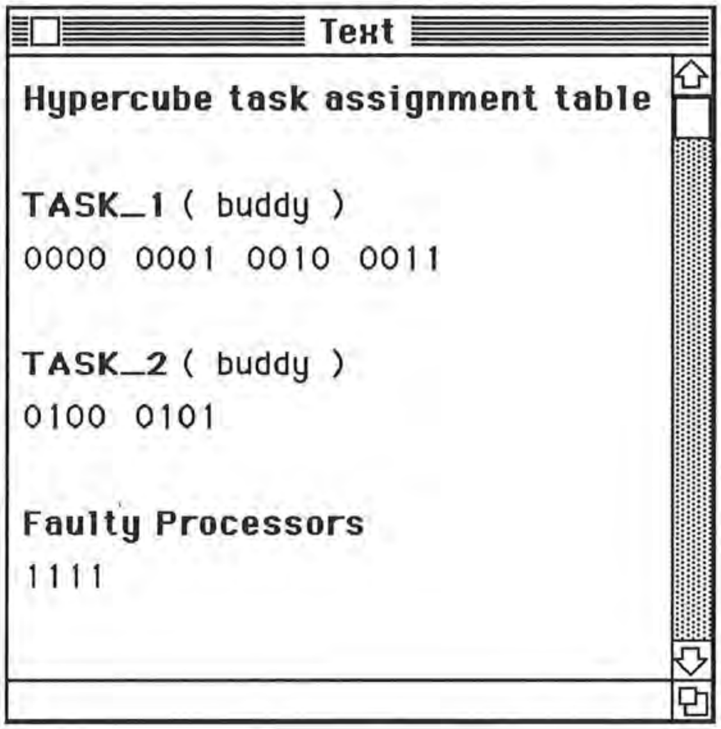


Fig.4.6 Reduced view of the 5-Cube



```
Hypercube task assignment table

TASK_1 ( buddy )
0000  0001  0010  0011

TASK_2 ( buddy )
0100  0101

Faulty Processors
1111
```

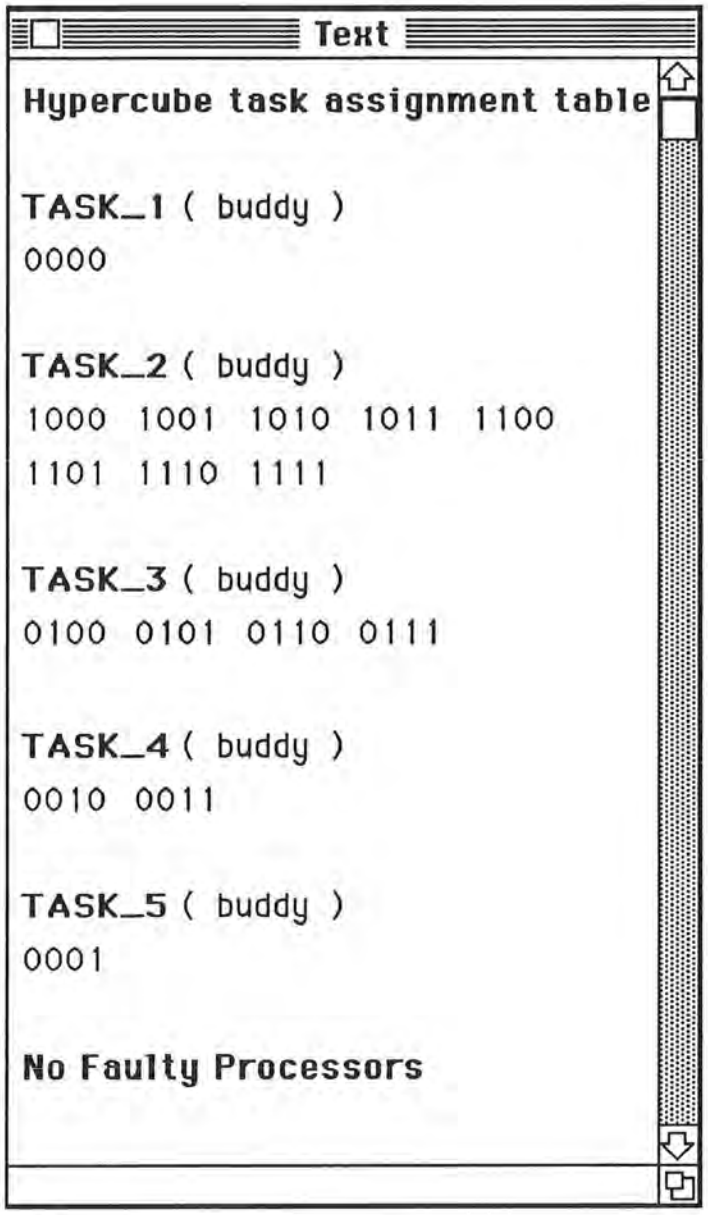
Fig.4.7 Text window corresponding to Fig.4.5

4.2. Implementation Results

i) Static Allocation Test

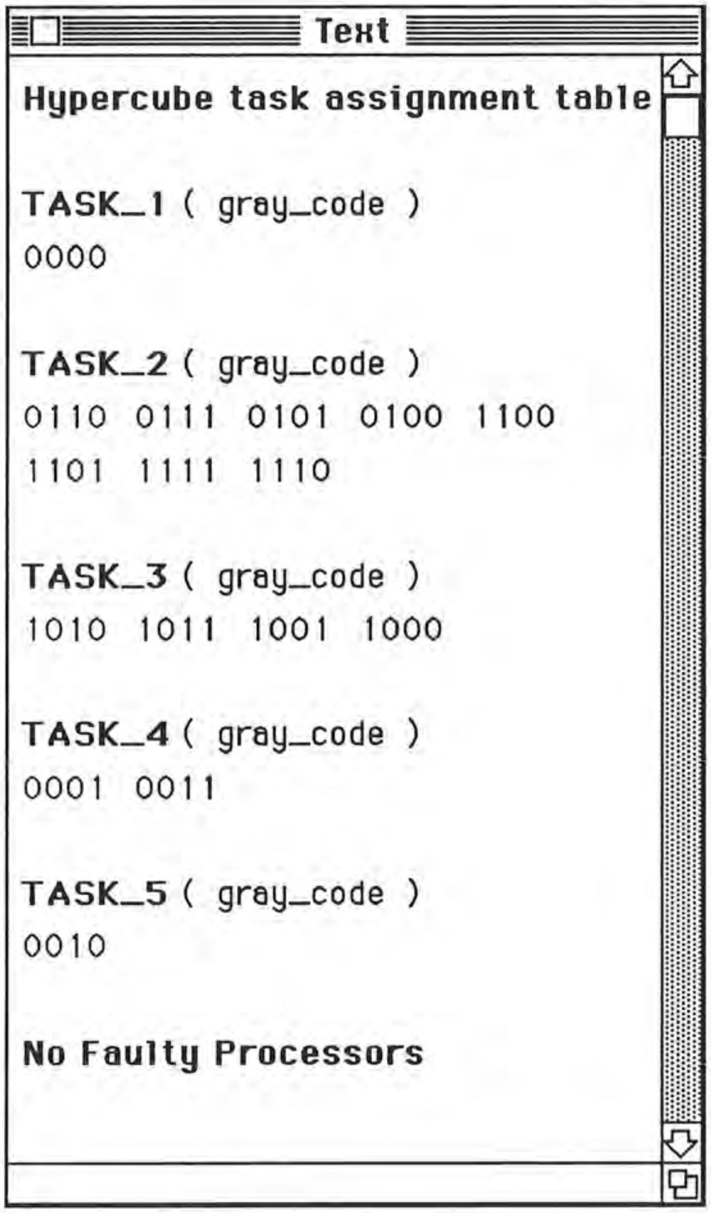
A test of the static processor allocation using three strategies in a 4-cube multiprocessor is given in Fig.4.8. Following are the incoming request numbers I_i's and their sizes Q_j 's:

$I_1 = Q_0$, $I_2 = Q_3$, $I_3 = Q_2$, $I_4 = Q_1$, $I_5 = Q_0$.



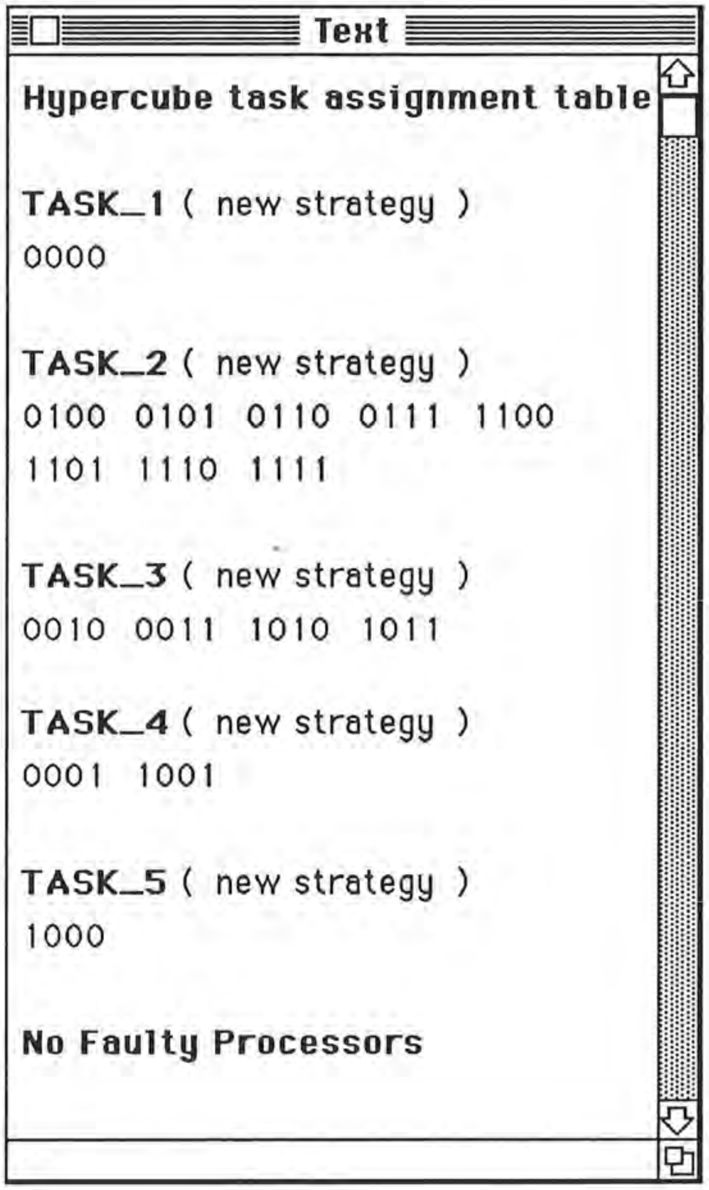
```
Hypercube task assignment table

TASK_1 ( buddy )
0000

TASK_2 ( buddy )
1000  1001  1010  1011  1100
1101  1110  1111

TASK_3 ( buddy )
0100  0101  0110  0111

TASK_4 ( buddy )
0010  0011

TASK_5 ( buddy )
0001

No Faulty Processors
```

Fig.4.8.a Processor allocation using buddy system



```
Hypercube task assignment table

TASK_1 ( gray_code )
0000

TASK_2 ( gray_code )
0110  0111  0101  0100  1100
1101  1111  1110

TASK_3 ( gray_code )
1010  1011  1001  1000

TASK_4 ( gray_code )
0001  0011

TASK_5 ( gray_code )
0010

No Faulty Processors
```

Fig.4.8.b Processor allocation using gray code system



```
Hypercube task assignment table

TASK_1 ( new strategy )
0000

TASK_2 ( new strategy )
0100  0101  0110  0111  1100
1101  1110  1111

TASK_3 ( new strategy )
0010  0011  1010  1011

TASK_4 ( new strategy )
0001  1001

TASK_5 ( new strategy )
1000

No Faulty Processors
```

Fig.4.8.c Processor allocation using new strategy

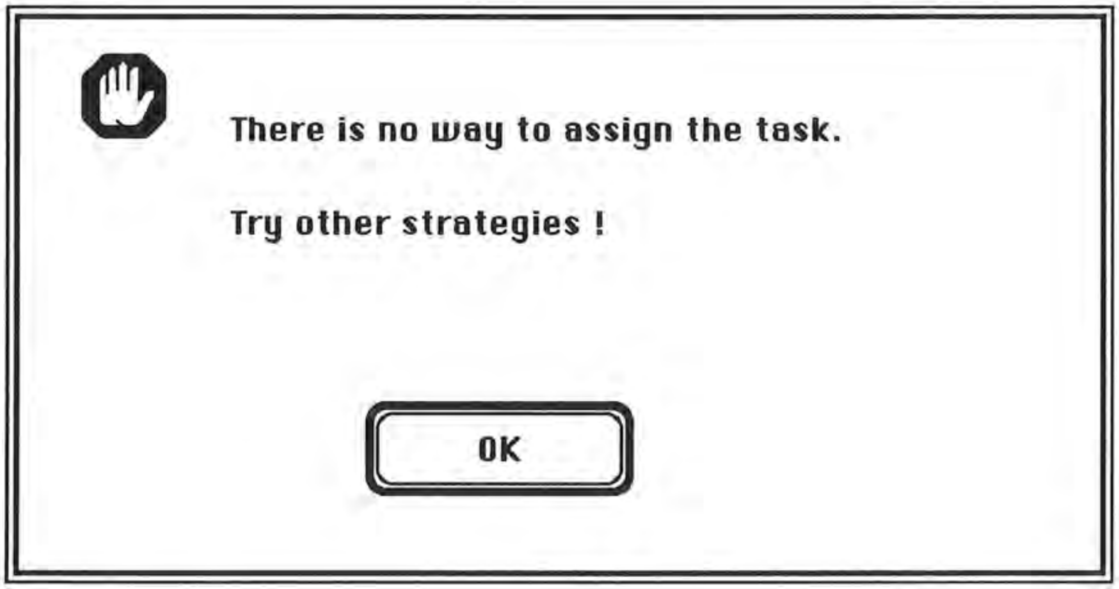
The results show that all these three strategies are optimal in the sense that each incoming request sequence is always assigned to a minimal subcube, and furthermore, from Fig.4.8.c we can see that the new strategy compacts things to the left which result in less system fragmentation; thus the new strategy recognizes more subcubes.

ii) Dynamic Allocation Test

Following are a series of requests:

$I_1 = Q_2$, $I_2 = Q_2$, $I_3 = Q_2$, $I_4 = Q_2$.

After allocation, release tasks I_1 and I_3. Now try allocating processors to incoming task $I_5 = Q_3$. A stop alert dialog will suggest that using other strategies may satisfy this (Fig.4.9).


There is no way to assign the task.

Try other strategies !

Fig.4.9 Stop alert dialog

Our results show that the buddy strategy will not satisfy this request, but gray code strategy and new strategy will satisfy this. This means that gray code strategy and new strategy outperform the buddy strategy in detecting the availability of subcube.

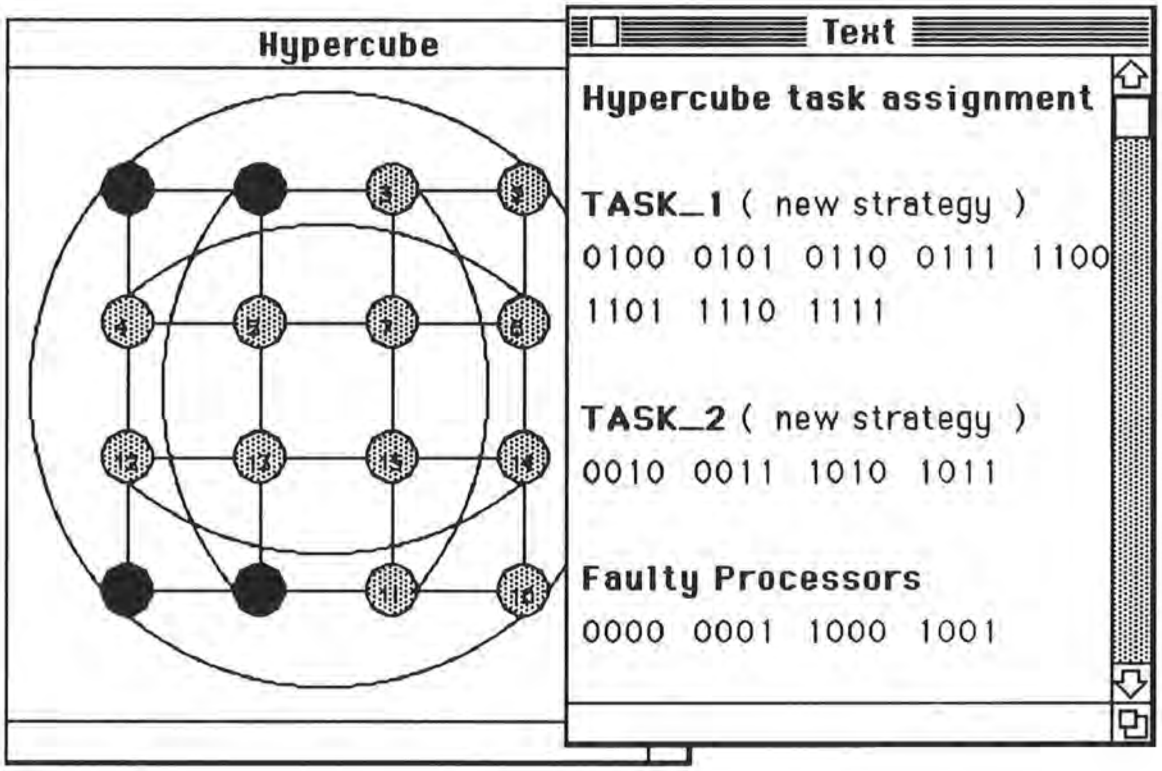
Another series of requets is

$I_1 = Q_1$, $I_2 = Q_2$, $I_3 = Q_1$, $I_4 = Q_3$.

After allocation, release tasks I_1 , I_3 and reallocate the processors to in coming request $I_5 = Q_2$, then both buddy strategy and gray code strategy will not be satisfied. But the new strategy will satisfy this which means that the new strategy outperforms both buddy strategy and gray code strategy.

iii) Fault Tolerance

In a 4-cube multiprocessor with the processors 0000,0001,1000, and 1001 faulty, consider the requests { $I_1 = Q_3$, $I_2 = Q_2$ }. Our result shows that neither buddy system allocation nor gray code strategy will be able to satisfy this. But the new strategy will satisfy this (Fig.4.10). This means that when some faulty processors are cosidered or processors are allocated dynamically the new strategy does better than buddy strategy or gray code strategy.



Hypercube task assignment

TASK_1 (new strategy)
0100 0101 0110 0111 1100
1101 1110 1111

TASK_2 (new strategy)
0010 0011 1010 1011

Faulty Processors
0000 0001 1000 1001

Fig.4.10 Processor allocation using new strategy with processors 0, 1, 8, 9 faulty

5. Summary

We have presented the design and implementation of the processor allocation in an n-cube multiprocessor using several processor allocation strategies on Macintosh microcomputer. We design an easy way to draw the n-dimension hypercube in the 2-dimension plane and users can easily simulate the processor allocation in an n-cube multiprocessor using some processor allocation strategy. A text window is also provided for users to look at the task assignment textually.

References

[1] A. Al-Dhelaan and B. Bose, "A New Strategy for Processors Allocation in an N-Cube Multiprocessor," Computer Science Department, Oregon State University, 1988.

[2] Apple Computer, Inc., Inside Macintosh, Vol. I, II, and V, Addison-Wesley Publishing Company, Inc.

[3] Stephen Chernicoff, Macintosh Revealed, Vol. I and II.

[4] M. Chen and K. G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," *IEEE Trans. Comput.*, Vol. C-36, Dec. 1987, pp. 1396-1407.

[5] M. Chen and K. G. Shin, "Embedment of interesting task modules into a hypercube multiprocessor," in Proc. Second Hypercube Conf., Oct. 1986, pp. 121-129.

[6] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Trans. Comput.*, Vol. C-37, July 1988, pp. 867-872.