Processor Allocation in k-ary n-cube Multiprocessors

Po-Jen Chuang and Chih-Ming Wu
Department of Electrical Engineering
Tamkang University
Tamsui, Taipei Hsien, Taiwan 25137, R. O. C.

Abstract

Composed of various topologies, the k-ary n-cube system is desirable for accepting and executing topologically different tasks. In this paper, we propose a new allocation strategy to utilize the large amount of processor resources in the k-ary n-cubes. Our strategy is an extension of the TC strategy on hypercubes and is able to recognize all subcubes with different topologies. Simulation results show that with such full subcube recognition ability and no internal fragmentation, our strategy depicts constantly better performance than the other strategies, such as the Free-list strategy on k-ary n-cubes and the Sniffing strategy.

1 Introduction

Composed of various topologies, such as the multi-dimensional toris, meshes and binary cubes, the k-ary n-cube is a generalized form in the multiprocessor system [1,2]. As in real practice the incoming tasks to a system can involve different topologies, the k-ary n-cube becomes very desirable for accepting and executing topologically different tasks. To utilize its large amount of processor resources, processor allocation strategies play quite an important role. When a task arrives dynamically at a k-ary n-cube system requesting processors connected in a specific topology, a good processor allocation strategy makes it possible to find a subsystem swiftly with the right topology and size. As an incoming task has a proper $r(2 \leq r \leq k)$ and $m(0 \leq m \leq n)$, an exact amount of processors with a particular topology must be allocated to it.

A k-ary n-cube, denoted by $H_{k,n}$, comprises k^n nodes, each with $2n$ connection links serving as communication channels directly to $2n$ immediate neighbors. The parameter n is the *dimension* of the cube and k is the *radix* or the number of nodes along each dimension. A 4-ary 3-cube is shown in Fig. 1. A node in the k-ary n-cube can be identified by an n-digit radix-k address $A_{n-1}A_{n-2}\cdots A_1A_0$, where A_i represents the node's position in the ith dimension. Rings, meshes, tori, binary n-cubes (hypercubes), and Omega networks are topologically isomorphic to the family of k-ary n-cubes [2]. The computation pattern (topology) and the size of the corresponding subcube requested by each incoming task must be specified first. The processors in an $H_{k,n}$ must be thus allocated to all such tasks so that processor utilization is maximized (or equivalently, a task is assigned an available subcube whenever it exists). Due to the special structure of the k-ary n-cube, it is nontrivial to detect the availability of a subcube. To accomplish this, several previous strategies have been proposed. The Free-list strategy on k-ary n-cubes [3] is an extension of the Free-list strategy on hypercubes. It can recognize all the free subcubes, but its k is restricted to the power of two. The Sniffing strategy [4] performs an exhaustive search under some subcube representations and is unable to achieve full subcube recognition. The k-ary Partner strategy [5] is based on the concept of "partners" in the k-ary tree representation and can allocate only r-ary subcubes with $r = k$. To achieve full subcube recognition ability with no limitations, we propose a new processor allocation strategy which is an extension from the Tree-Collapsing (TC) strategy on hypercubes [6]. Our new strategy recognizes either cubic or non-cubic subcubes and allocates the exact amount of processors to an incoming task. By allocating the exact amount of processors, internal fragmentation can be substantially eliminated.

2 Our Proposed Strategy

Our strategy is indeed the generalization of the TC Strategy for processor allocation in hypercubes [6]. The TC Strategy uses an n-bit *notation* to represent exactly 2^{n-m} distinct m-dimensional subcubes in an n-dimensional hypercube. For example, a 4-bit notation $\square * \square *$ represents 4 distinct 2-dimensional subcubes $0*0*$, $0*1*$, $1*0*$, and $1*1*$ in a 4-dimensional hypercube. Algorithm G*, the basis of the TC Strategy, is used to generate the complete search space for an incoming request (i.e., all the possible distinct subcubes of the requested subcube dimension, say m) by repeating the process of *right rotating transform* on the n-bit notation starting from the *primary* notation $\square\square\cdots\square * *\cdots *$, with the leftmost $n-m$ bits being the *select* bits and the rightmost m bits being the don't care bits. The *Right Rotating Transform (R-transform)*, as stated in [6], operates on an n-bit notation, say $A = a_0a_1\cdots a_{n-1}$. Specifically, an R-transform from bit i, denoted by ρ_i, performs right rotation on the rightmost $n-i$ bits of A and produces a *rotated notation* $A' = a_0'a_1'\cdots a_{n-1}'$ such that

(i) $a_j' = a_j$, for $0 \leq j < i$,

(ii) $a_j' = a_{j-1}$, for $i < j \leq n-1$, and

(iii) $a_i' = a_{n-1}$.

To give an example, the rotated notation of performing ρ_2 on a 6-bit notation $\square * \square * \square *$ is $\square * * \square * \square$.

Algorithm G* for the TC Strategy is given in the following.

Algorithm G*:

Input: the primary notation (stored in *primary_notation*) and the dimension of a subcube to be allocated, m.

Procedure main(n, m, primary_notation)
begin
if($m = 0$ or $m = n$) then stop;
for $i := n - m - 1$ down to 0
call rotate(i, primary_notation, new_notation);
call subsequent_rotate(i, 0, new_notation);
end for
end;

Procedure subsequent_rotate(pre_level, pre_step, pre_notation)
begin
step := pre_step + 1;
if *step* $\geq m$ then return
else
for $i :=$ pre_level to $n - m - 1$
call rotate(i+*step*, pre_notation, new_notation);
call subsequent_rotate(i, *step*, new_notation);
end for
end if
end;

It is clear that there are $C(n,m) \times 2^{n-m}$ possible distinct m-dimensional subcubes in an n-dimensional hypercube. As Algorithm G* can generate a complete search space for any requested m-dimensional subcubes in an n-dimensional hypercube with the primary notation and the $C(n,m)-1$ rotated notations it produces, we believe the same notation can be generalized and applied to an $H_{k,n}$ ($k \geq 2$) by using the n-bit *notation* (with m *don't care* bits $*$'s and $n-m$ *select* bits $\square$'s) to represent exactly k^{n-m} distinct $S_{k,m}$'s. For the hypercubes, the select bits can be 0 or 1; for the $H_{k,n}$, they can be $0, 1, 2, \cdots, k-1$. It is easy to derive that Algorithm G* can be used to generate the complete search space for any requested $S_{k,m}$ in $H_{k,n}$,i.e., to generate the $C(n,m) \times k^{n-m}$ possible distinct $S_{k,m}$'s.

Our proposed strategy, to be called the extended TC strategy, uses Algorithm G* to generate the complete search space and is able to recognize any $S_{k,m}$'s with radices being the same as the system's in the $H_{k,n}$. To recognize subcubes of the other topologies, such as meshes, hypercubes, and multi-dimensional meshes, our extended TC strategy makes use of *subnotations*, each representing a subset of possible subcubes which may have different dimension radices. To give an example, for notation $\square\square\cdots\square**\cdots*$, the subset of possible subcubes with different dimension radices $(m_{m-1}, m_{m-2}, \cdots, m_0)$ can be represented by subnotation $\square_{n-1}\cdots\square_m(*_{p_{m-1}})_{r_{m-1}}\cdots(*_{p_0})_{r_0}$, where $\square \in \{0, 1, \cdots, k-1\}$; p_i $(0 \leq i \leq m-1)$ represents the starting node position in dimension i, thus $0 \leq p_i \leq k-1$; r_j $(0 \leq j \leq m-1)$ is the requested radices (number of nodes) in dimension j and $2 \leq r_j \leq k$. Assuming the incoming task is a two-dimensional 4×2 mesh and the system is a 4-ary 3-cube, a subset of the possible subcubes (noted by $\square**$) can be represented by subnotation $\square(*_0)_4(*_0)_2$. The starting node of dimension 0 is 0 and the dimension needs two neighboring nodes, say node 0 and node 1. Likewise, dimension 1 can take nodes 0, 1, 2, and 3. A possible 4×2 mesh $0(*_0)_4(*_0)_2$ (that is, when $\square$ =0 in the above subnotation) has 8 nodes (0,0,0), (0,0,1), (0,1,0), (0,1,1), (0,2,0), (0,2,1), (0,3,0) and (0,3,1). If all the 4×2 meshes represented by the subnotation are busy, another subnotation $\square(*_0)_4(*_1)_2$, $\square(*_0)_4(*_2)_2, \cdots$, or $\square(*_3)_4(*_3)_2$ is considered subsequently. (Note that the combinations $00, 01, 02, 03, 10, 11, \cdots, 32, 33$ of all the possible starting node positions in the dimensions with *don'tcare* bits are considered, and beginning from the starting node position, the requested numbers of nodes in those dimensions are taken in a "wraparound" way.) All the possible subcubes (4×2 meshes) represented by each of the above subnotations are to be said in a subset of subcubes *dictated* by the primary notation $\square**$. If neither of the subcubes in any subcube subsets (dictated by the primary notation) satisfies the incoming task, start performing Algorithm G* to get the rotated notations, one by one, from $\square**$. Repeat the above process for each rotated notation until a free requested subcube is found.

The operation of processor allocation using our extended TC strategy is provided below.

Processor Allocation

step 1: Set $m :=$ the dimension and $r_i :=$ the radix for dimension i of the subcube required to accommodate task I_j.

step 2: Set *notation* := primary_notation.

step 3: Search all the subsets (represented by subnotations) of the possible subcubes (with the required dimension and radix in each dimension) dictated by *notation* until an available subcube is found. When found, go to step 5.

step 4: Perform Algorithm G* and do the same search process as Step 3 for one rotated notation (if any) at a time. If an available subcube is found, go to the next step; otherwise, go to Step 6.

step 5: Set the corresponding allocation bits for each node of the available subcube to 1's and allocate the subcube to task I_j. Stop.

Step 6: Attach I_j to the task queue and wait until a subcube is released.

Processor Relinquishment:

Reset those allocation bits for each node of the released subcube to 0's.

3 Performance Evaluation

Extended simulation runs have been conducted to collect performance results for the extended TC strategy, the Free-list strategy on k-ary n-cubes and the Sniffing strategy. (The k-Partner strategy is excluded due to its inability to allocate topologically different tasks.) The simulation is carried out in a 4-ary 3-cube system and task allocation is handled by a task dispatcher independent of the system. All processors are initialized to be free and the 100 tasks are generated and queued at the dispatcher (no new tasks are generated during the course of simulation). Two residence time distributions of uniform(5.0, 10.0) and

uniform(10.0, 15.0) as well as an FCFS scheduling discipline are considered. The dimensions and radices of subcubes requested by the 100 tasks are respectively governed by two distributions, Uniform (U) and Normal (N). Thus there are four combinations of the dimension-radix distribution: U*U, U*N, N*U and N*N, and they are able to form subcubes of various topologies.

Performance measures such as the completion time, processor utilization, external fragmentation and internal fragmentation are collected for analysis and comparison. All the results are the average values of 10 random seeds; given 95% of confidence, they will be correct with the error range falling between ±3.4%. Simulation results of the four performance measures (under the residence time distribution of uniform(5.0, 10.0) and the four dimension-radix distributions with tasks requesting different topologies) are respectively depicted in Figs. 2(a)-(d). As observed, for the U*U distribution, the extended TC strategy depicts higher processor utilization than the Sniffing strategy by 20.56%. This is mainly because our strategy is implemented with full subcube recognition ability and without internal fragmentation, while the Sniffing strategy repeatedly allocates excessive processors to tasks due to its singular internal fragmentation (44.34%, the result of its special subcube representations). Processor utilization for the extended TC strategy is also higher than that of the Free-list strategy (by 7.28%) because the Free-list strategy has to allocate more than needed free processors to a task when the requested radix is not the power of two, resulting in remarkable internal fragmentation (28.66%). Note that both internal and external fragmentations can affect processor utilization in the system but the former is more dominating. In fact, substantial internal fragmentation has enacted obvious impact on processor utilization and completion time for both the Sniffing and Free-list strategies as the simulation results demonstrate.

For the U*N distribution, processor utilization for the extended TC strategy is better than the Sniffing strategy (by 13.66%) and the Free-list strategy (by 16.15% — a significant lift from 7.28% for the U*U distribution). The markdown for the Free-list Strategy is apparently the result of its uprising internal and external fragmentations because when the dimension-radix distribution is U*N, the most frequently requested subcubes will be subcubes whose radices are three. In this case, the internal fragmentation for the Free-list strategy worsens and as a result external fragmentation also takes a turn for the worse.

For the N*U distribution, processor utilization for the extended TC strategy is still higher than the other two strategies (19% higher than the Sniffing strategy and 7.72% higher than the Free-list strategy). As can be seen, the value for the Free-list strategy rises to the same level as in the U*U distribution because the radix distribution becomes uniform again, cutting down the probability of prompting internal fragmentation for the strategy. However, when the dimension-radix distribution turns N*N, processor utilization for the Free-list strategy plunges again due to the same reason noted before for the U*N distribution.

With the same (dimension-radix) distributions, simulation results under the residence time distribution of uniform (10.0, 15.0) are also collected to see the effect of different residence times. As it turns out, the results follow almost the same trend as what is collected for the residence time distribution of uniform (5.0, 10.0). The only difference is: The longer the residence time (10.0, 15.0), the larger the external fragmentation for all the strategies under all the dimension-radix distributions. This is because extending the residence time of each task tends to prompt external fragmentation.

4 Conclusion

In this paper we propose a new processor allocation strategy for the k-ary n-cube with no limitation on the requested dimensions and radices. Our proposed strategy can recognize either cubic or non-cubic subcubes and allocate the exact amount of processors to an incoming task. That is, our strategy is able to achieve full subcube recognition for tasks requesting different topologies. Under such a strategy, internal fragmentation can be substantially eliminated. As simulation results demonstrate, in spite of having larger external fragmentation, our strategy depicts constantly better processor utilization than the other investigated strategies due to its ability to achieve full subcube recognition under different topologies and also due to the absence of internal fragmentation. It is also noted that internal fragmentation plays quite an important role in dominating processor utilization in a system. Hence, besides pursuing full subcube recognition and preventing external fragmentation, a good processor allocation strategy should first concentrate on the elimination of internal fragmentation.

Acknowledgments

This work was supported in part by the National Science Council, Taiwan, R. O. C., under Grant No. NSC85-2213-E-032-006.

References

[1] W. J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. on Computers*, Vol. 39, No. 6, pp. 775–785, June 1990.

[2] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993, pp. 86–87.

[3] H.-L. Chen and C.-T. King, "An Efficient Dynamic Processor Allocation Scheme for K-ary n-cube Multicomputers," *Proc. 1993 Int'l Conf. on Parallel and Distributed Systems*, Dec. 1993, pp. 217–221.

[4] V. Gautam and V. Chaudhary, "Subcube Allocation Strategies in a K-ary N-cube," *Proc. 1993 ISCA Int'l Conf. on Parallel and Distributed Computing Systems*, Oct. 1993, pp. 141–146.

[5] K. Windisch, Virginia Lo, and B. Bose, "Contiguous and Non-Contiguous Processor Allocation Algorithms for K-ary N-cubes," *Proc. 1995 Int'l*

Conf. on Parallel Processing, Aug. 1995, pp. II 164–168.

[6] P.-J. Chuang and N.-F. Tzeng, "A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers," *IEEE Trans. on Computers*, Vol. 41, No. 4, pp. 467–479, Apr. 1992.

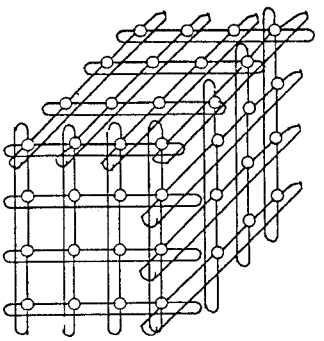

Fig. 1. A 4-ary 3-cube.
(Hidden nodes and connection links are not shown.)

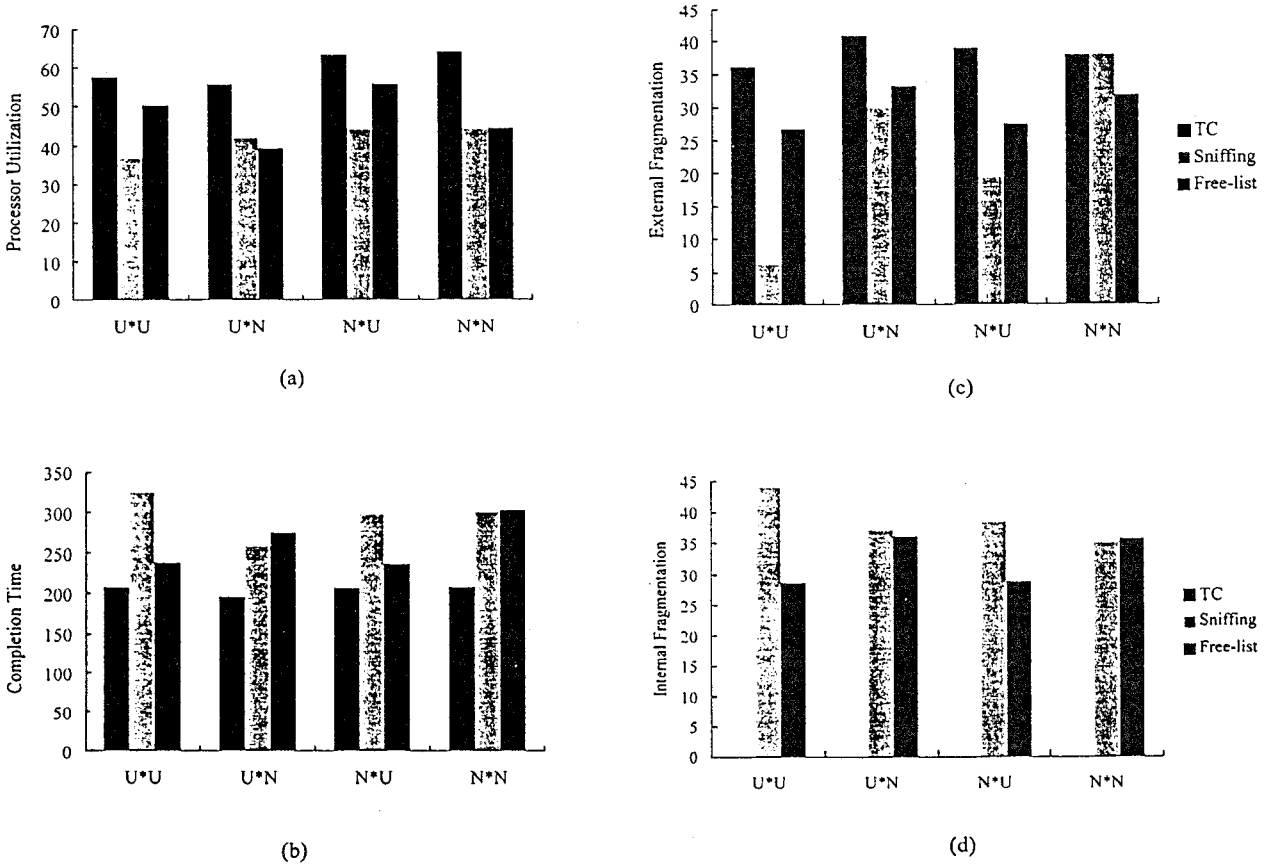

Fig. 2. (a) Processor utilization (b) Completion time (c) External fragmentation (d) Internal fragmentation for different strategies under the residence time distribution of uniform(5.0, 10.0).