DC-Mesh: A Contracted High-Dimensional Mesh for Dynamic Clustering

Masaru Takesue

Dept. Electronics and Information Engr., Hosei University, Tokyo 184-8584 Japan
takesue@ami.ei.hosei.ac.jp

Abstract. This paper proposes a *DC-mesh* network that allows requesting nodes to be put into clusters while the requests are sent to a target node, as well as is easy to layout on an LSI chip. To organize the DC-mesh, we use the partitioning in the word space based on the Hamming code [1]. We introduce an index scheme, (parity-value,information-value), in the word space, and map it onto a 4-D (dimensional) mesh so that the Hamming distance between the words in each partition is preserved in the Manhattan distance between the corresponding nodes on the mesh; two of the dimensions are contracted for easy wiring. The resultant DC-mesh consists of a number of local 2-D meshes and a single global 2-D mesh; all processing nodes linked to one local mesh are connected to one node of the global mesh via a bus to compensate for the contraction. A subset of the nodes in a partition organizes a dynamic cluster. The diameter equals the greater of the diameters of local and global meshes.

1 Introduction

To reduce communication latency in multiprocessors, a cache hierarchy in a *static cluster*, of which member nodes are fixed in hardware, is a popular technique. However, the static cluster cannot adapt efficiently to the change of communication patterns, due to the contention on per-cluster resources such as the directory for cache coherence, and because of the complexity of cache protocols.

For hypercube-connected systems, a *dynamic cluster* can be organized of which member nodes are determined during the requests are sent to the target node [2], exploiting the partitioning of the n-bit word space based on the n-bit Hamming code [1]; one cluster consists of a subset of nodes in a partition. No per-cluster resource is required for the clustering. The distance between the representative and another nodes in each cluster is less than three.

The hypercube, however, needs long wires to layout, that will lead to an unacceptably long signal-delay in a future LSI chip [3]. This paper addresses such networks that need no long wires, but also can produce dynamic clusters. As a network with such properties, we propose a *DC-mesh* (*Dynamically Clustering mesh*). To organize the mesh, we map the partitions in the word space [1] onto a high-dimensional mesh so that the Hamming distance in the word space is preserved in the Manhattan distance on the mesh as completely as possible.

We start with a 2-D (dimensional) array of the words indexed by their parity and information values. With a reflected Gray code sequence, we map the array onto a 4-D mesh, but contract two of the dimensions for easy layout. This leads to multiple local 2-D meshes and a single global 2-D mesh that are disconnected with each other. To obtain the DC-mesh, we connect all processing nodes linked to a local mesh together with one node of the global mesh via a bus.

Related work: Commercial and research systems adopt static clusters: STiNG [4] and SGI Origin [5] use the ring and an extended hypercube, respectively. The Stanford Dash is configured into the mesh [6]. A research chip, Hydra [7], has 4 processors and exploits two buses between their caches. A chip reconfigurable for several types of applications includes 64 processing nodes, that are connected with each other by the mesh [8].

The Ψ-cube [1] can produce dynamic clusters since it is organized through recursive partitioning based on the Hamming code. But this network will be unacceptable in an LSI chip due to its long buses, though it is much easier to wire than the hypercube. It is possible to organize dynamic clusters in multistage networks if each network switch has a directory [9]. Dynamic clusters are also organized by freezing the memory blocks in a specified cache and allowing the other caches to access the frozen blocks with no cache coherence [10].

A few methods based on the Hamming code and/or other linear codes have been reported to map the resources such as I/O processors onto the hypercube so that each node is adjacent to at least one resource [11] or a specified number of resources [12, 13]. However, those methods exploit none of the properties of Hamming codes exploited in our partitioning.

In the rest of the paper, Section 2 describes the properties of partitions [1]. Section 3 organizes the DC-mesh, and describes the routing method for clustering. Section 4 summarizes the paper and discusses future research.

2 Properties of the Partitions

This section describes the properties of Hamming code-based partitions that are used to organize the DC-mesh and dynamic clusters; for the detail, see [1].

A *codeword* c of the n-bit Hamming code $\psi(n, k)$ has p-bit parity for k-bit information, where the p is the smallest integer that satisfies $(2^p - 1) \geq n$, and $k + p = n$. Assuming no or a single-bit error in a received word w, the *syndrome* $\varepsilon = w \cdot H_n^t$ indicates the erroneous bit position if $\varepsilon \neq 0$, or no error otherwise, where H_n^t is the transpose of the parity-check matrix H_n for $\psi(n, k)$.

For partitioning, we exploit not only single-bit errors, but also *detectable* double-bit errors for which $\varepsilon > n$. Then, of a pair of erroneous bit positions (d, f) $(d + f = \varepsilon,\ d < f)$, we fix position f equal to 2^{p-1}; so $d = \varepsilon \oplus f$. The number N_d of detectable double-bit errors is equal to $(2^p - 1 - n)$.

The *error vector* e_ε for syndrome ε has bit(s) of 1 at position(s) s $(= \varepsilon)$ or (d, f) $(d + f = \varepsilon)$, and bits of 0 in the other positions. Let T_c denote the partition represented by word c, and put a word w with syndrome ε into partition T_c, where $c = w \oplus e_\varepsilon$ and $\oplus$ is the exOR operation. Then the n-bit word space is

partitioned into 2^k *partitions each of* 2^p *words*. The Hamming distance between the *leader* (i.e., the representative) word c and word w is less than 3.

We produce *multiple suits* of codewords from the *original suit* S_0 of codewords for the parity-check matrix H_n. Then the suits organize another set of partitions for the word space, i.e., 2^p *suits each of* 2^k *(code) words*. Partitioned with any suit, the 2^p words in each obtained partition are those in different (2^p) suits. Thus every word belongs to one of the 2^p partitions each obtained with a separate suit, and is the leader when partitioned with the suit including the word.

To avoid traffic congestion on a single leader when clustering the requests for a target node t (see Section 3), we send the request from a requesting node s to *one* (ℓ) *of its* 2^p *leaders* (each in a separate suit). This routing is based on the following property: Let $S_{\ni t}$ be the suit including a word t, and assume that a word ℓ is included in both suit $S_{\ni t}$ and partition T_s. Then word ℓ is unique and is obtained by $\ell = s \oplus e_\varepsilon$, where ε is the syndrome for $s \oplus t$ with suit S_0.

3 DC-meshes

This section presents an indexed word space, the mapping from the space to the node-address space of the DC-mesh, and its structure and routing method.

3.1 Indexed Word Space

We assign the index (i, j) to the word, denoted by $w_{i,j}$, of which parity and information parts have values i and j, respectively; then an array of 2^p rows and 2^k columns is organized. We denote the i^{th} row and j^{th} column by P_i and I_j. Let ψ-*neighbors* of a word $w_{i,j}$ be the non-leaders in the partition $T_{w_{i,j}}$. Recall that N_d number of double-bit error words have incorrect bits in positions (d, f). Let d' denote the position d in the parity part, and $N_{d'}$ refer to the number of double-bit error words with erroneous positions (d', f). Then,

Theorem 1. *Of the* ψ*-neighbors of word* $w_{i,j}$*,* k *words are in row* P_i*,* $(p + N_{d'})$ *words are in column* I_j*, and* $(N_d - N_{d'})$ *words are at the cross-points of row* $P_{i \oplus e_f}$ *and columns* $I_{j \oplus e_d}$ *(*$d \in$ *information-part), i.e., words* $w_{i \oplus e_f, j \oplus e_d}$*.*

Proof. Of the single-bit error words of word $w_{i,j}$, k words have each an error in an information bit, so those words are in row P_i. Likewise, p words each with an error at a parity position are located in column I_j. Moreover, of the double-bit error words, $N_{d'}$ words each with errors at positions (d', f) are in column I_j, so a total of $(p + N_{d'})$ words are in the I_j. Since $(N_d - N_{d'})$ number of double-bit error words each have a pair (d, f) ($d \in$ information-part) of error positions, those words are located in the cross-points of row $P_{i \oplus e_f}$ and columns $I_{j \oplus e_d}$.

3.2 Structure of the DC-meshes

The parity size p does not vary so much (equals 3 or 4) for up to the n of 15, so we map each column I_j onto one *local* 2-D *mesh*, i.e., a *basic block* of the

DC-mesh. Let node (i, j) be the node on which word $w_{i,j}$ is mapped. In parallel with this mapping, we map each row P_i $(0 \leq i < 2^p)$ onto a 2-D mesh, so that it consists of the nodes (i, j) $(j = 0, \ldots, 2^k - 1)$ each of different local meshes for columns I_j. Then a 4-D mesh is obtained.

Since a 4-D mesh is generally difficult to layout, we contract the 2^p number of 2-D meshes for rows into a separate 2-D mesh, called *global mesh*, that is produced by contracting all 2^p nodes (i, j) $(i = 0, \ldots, 2^p - 1)$ in the local mesh for each column I_j into a single node, denoted by $*j$, of the global mesh. Then we obtain 2^k local meshes for the columns and one global mesh for the rows; note that these meshes have no connection with each other.

The DC-mesh is organized as follows: We connect one *processing node* to a node of a local mesh by a direct link, and all processing nodes connected with the local mesh for I_j together to node $*j$ of the global mesh by a bus. Moreover, to preserve the ψ-neighbor relation of the word space on the DC-mesh as completely as possible, we exploit the Gray code as the mapping function (see Definition 1) in the word-to-node mapping described above, since this code allows the adjacency relation in the word space to be preserved on the mesh [14].

Let $i(r_1)$ and $i(c_1)$ be the upper r_1 bits and the lower c_1 $(r_1 + c_1 = p)$ bits of index i. Likewise, the upper r_2, middle c_2, and lower d_2 $(r_2 + c_2 + d_2 = k)$ bits of index j are denoted by $j(r_2)$, $j(c_2)$, and $j(d_2)$, where d_2 equals $(n - 8)$ if $n > 8$ or 0 otherwise, and keeps the size of global mesh small, i.e., less than or equal to 4×4. Let $*(j/2^{d_2})$ denote the set of (words in the) 2^{d_2} columns of which indices j are the same in the r_2-bit and c_2-bit portions, but are different from each other in the d_2-bit portion. Indices (x_g, y_g) and (x_ℓ, y_ℓ) are used for the nodes in the global mesh M^g and the local mesh $M^\ell_{x_g,y_g,z_g}$, respectively. We denote the m^{th} code in the sequence of *len*-bit Gray codes by $G(m, len)$.

Definition 1. *(Mapping Method) We map word $w_{i,j}$ in column I_j on the node (x_ℓ, y_ℓ) of the local $2^{r_1} \times 2^{c_1}$ mesh $M^\ell_{x_g,y_g,z_g}$, and map the set $*(j/2^{d_2})$ of 2^{d_2} columns on the node (x_g, y_g) of the global $2^{r_2} \times 2^{c_2}$ mesh M^g, where $x_\ell = G^{-1}(i(r_1), r_1)$, $y_\ell = G^{-1}(i(c_1), c_1)$, $x_g = G^{-1}(j(r_2), r_2)$, $y_g = G^{-1}(j(c_2), c_2)$, $z_g = j(d_2)$, and G^{-1} is the inverse of G.*

The mapping when $n = 6$ (and hence, $k = p = 3$) is shown in Fig. 1. A mesh node is shown by the rectangle, outside of which the node index is shown (only for $M^\ell_{0,0}$ and M^g, for space). The two-digit integer ij for M^ℓ or $*j$ for M^g inside the node represents the index, (i, j) or $*(j/2^{d_2})$ ($2^{d_2} = 1$, in this case), of the word or column-set mapped on the node. Each column I_j is mapped onto a 2×4 M^ℓ ($r_1 = 1$ and $c_1 = 2$). Word $w_{5,0}$ ($i = 5$) in column I_0, for instance, is mapped on node $(G^{-1}(5(r_1), r_1), G^{-1}(5(c_1), c_1)) = (1, 1)$ of $M^\ell_{0,0}$. The 2×4 M^g ($r_2 = 1$ and $c_2 = 2$) for rows is obtained by mapping the set $*j$ of a single column (since $d_2 = 0$) onto node $(G^{-1}(j(r_2), r_2), G^{-1}(j(c_2), c_2))$. For example, column $*3$ is mapped on node $(0, 2)$ of M^g.

The mapping when $n = 10$ (so $p = 4$ and $k = 6$) is shown in Fig. 2. In this case, $r_1 = c_1 = 2$ and $r_2 = c_2 = d_2 = 2$, so both M^ℓ and M^g are of 4×4; one M^ℓ is shown by the rectangle. Indices x_g and y_g for M^ℓ meshes (and hence, of the M^g

nodes) are shown in the left-most and upper-most parts. Since $2^{d_2} = 4$, the set $*(j/4)$ of four columns, denoted by $*(j/4)z_g$ ($z_g = 0, \ldots, 3$) in the M^ℓ box ($j/4$ is expressed by a hexadecimal number), are mapped on the node $*(j/4)$ with index $(G^{-1}(j(r_2), r_2), G^{-1}(j(c_2), c_2))$ of the M^g; this leads to the quintuplet of four M^ℓs with indices $(G^{-1}(j(r_2), r_2), G^{-1}(j(c_2), c_2), z_g)$ ($z_g = 0, \ldots, 3$) and the M^g node. Index z_g (shown in the parentheses near the two M^ℓs in two quintuplets, for space) of M^ℓ equals 0, 1, 2, and 3 respectively for the upper-left, upper-right, lower-left, and lower-right meshes in each quintuplet. To increase the bandwidth in M^g, adjacent nodes of the M^g are connected by four ($= 2^{d_2}$) links.

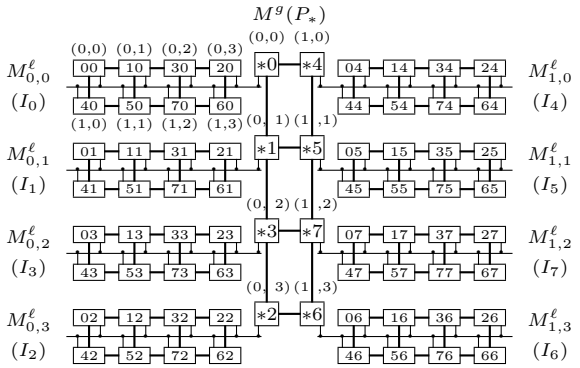

Fig. 1. The 64-node DC-mesh.

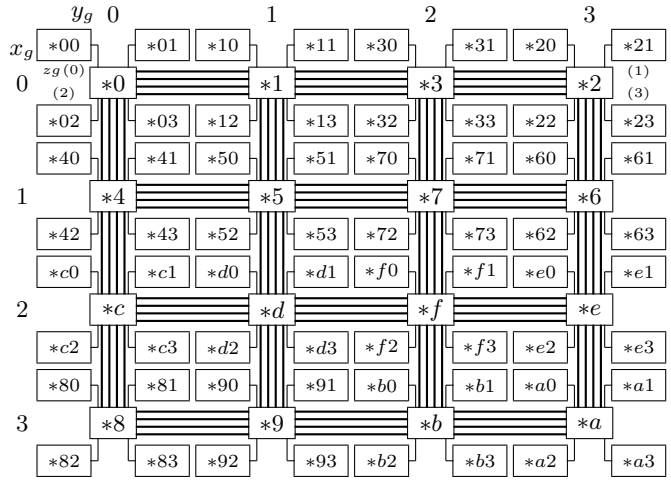

Fig. 2. The 1k-node DC-mesh.

3.3 Routing and Clustering in the DC-mesh

Assume a source and a target addresses, $((x_{g(s)}, y_{g(s)}, z_{g(s)}), (x_{\ell(s)}, y_{\ell(s)}))$ and $((x_{g(t)}, y_{g(t)}, z_{g(t)}), (x_{\ell(t)}, y_{\ell(t)}))$, of a massage in the DC-mesh. Then the message is sent according to the following XY-routing:

Definition 2. *(Routing Method) The message is sent to the target in the local mesh $M^{\ell}_{x_{g(s)}, y_{g(s)}, z_{g(s)}}$ if the global indices of the source and target are the same; $(x_{g(s)}, y_{g(s)}, z_{g(s)}) = (x_{g(t)}, y_{g(t)}, z_{g(t)})$. Otherwise, it is first sent to node $(x_{g(s)}, y_{g(s)})$ of the global mesh M^g via the local bus $z_{g(s)}$ of the source, next to node $(x_{g(t)}, y_{g(t)})$ on the M^g if $x_{g(s)} \neq x_{g(t)}$ or $y_{g(s)} \neq y_{g(t)}$ (this step is not required if $x_{g(s)} = x_{g(t)}$ and $y_{g(s)} = y_{g(t)}$), last to the target $(x_{\ell(t)}, y_{\ell(t)})$ in the local mesh $M^{\ell}_{x_{g(t)}, y_{g(t)}, z_{g(t)}}$ via the target's local bus $z_{g(t)}$.*

Theorem 2. *The Manhattan distance required for a message transfer equals $|x_{\ell(s)} - x_{\ell(t)}| + |y_{\ell(s)} - y_{\ell(t)}|$ if $(x_{g(s)}, y_{g(s)}, z_{g(s)}) = (x_{g(t)}, y_{g(t)}, z_{g(t)})$, $|x_{g(s)} - x_{g(t)}| + |y_{g(s)} - y_{g(t)}|$ if $x_{g(s)} \neq x_{g(t)}$ or $y_{g(s)} \neq y_{g(t)}$, or 0 otherwise.*

Proof. This is clear since the message is sent via the local mesh in the first case, through the global network in the second case, and via the local buses otherwise, assuming that the connection, such as a point-to-point link and a bus, between a network node and a processing node is not counted in the distance.

Thus the diameter of DC-mesh is equal to the greater of the diameters of local and global meshes. For the clustering, we use the value of word $w_{i,j}$ mapped on node (i, j) as its address. The word value is easy to obtain by $i = G(x_\ell, r_1) \circ G(y_\ell, c_1)$, and $j = G(x_g, r_2) \circ G(y_g, c_2) \circ z_g$, where $\circ$ is concatenation. A dynamic cluster is produced of a subset of the nodes in a partition. Let C_x be a dynamic cluster produced from the partition T_x represented by node x; note that node x may not be in the C_x, but we say that it is represented by node x. Recall that $S_{\ni t}$ denotes the suit of words that includes word t. Then,

Theorem 3. *(Dynamic Clustering) If a node s requesting for a service from a target node t sends the request to node $\ell = s \oplus e_\varepsilon$ included in both partition T_s and suit $S_{\ni t}$, then node s is put into the cluster C_ℓ, where ε is the syndrome for $s \oplus t$ with suit S_0. Moreover, node s is put into different clusters, C_{ℓ_1} and C_{ℓ_2}, for separate targets, t_1 and t_2, if they are in different suits, $S_{\ni t_1}$ and $S_{\ni t_2}$.*

Proof. There is a unique node ℓ included in both partition T_s and suit $S_{\ni t}$ (see Section 2). Since *all requesting nodes* send their requests to one of the nodes in suit $S_{\ni t}$, those nodes are partitioned into *clusters* . Particularly, node s is put into cluster C_ℓ. For different target nodes, t_1 and t_2, such that $S_{\ni t_1} \neq S_{\ni t_2}$, node s belongs to separate clusters, C_{ℓ_1} and C_{ℓ_2}, because leader nodes, ℓ_1 $(= s \oplus e_{\varepsilon 1})$ and ℓ_2 $(= s \oplus e_{\varepsilon 2})$, are in different suits, $S_{\ni t_1}$ and $S_{\ni t_2}$, where $\varepsilon 1$ and $\varepsilon 2$ are the syndromes for $s \oplus t_1$ and $s \oplus t_2$ both with suit S_0.

The Hamming distance between node s and leader ℓ is less than 3 since $s \in C_\ell \subseteq T_\ell$ (or $\ell \in T_s$). The type of a request sent from leader ℓ to target t and

its issue timing depend on the applications. For instance, leader ℓ relays the first received request to the target for cache coherence, or produces a request after receiving all requests and sends it to the target for barrier synchronization.

Last we describes how the ψ-neighbor relation is preserved in the partitions (and clusters) produced on the DC-mesh. We denote the Hamming and Manhattan distances between indexes i and i' by $HD(i, i')$ and $MD(i, i')$.

Theorem 4. *(Preservation of the ψ-Neighbor Relation) The ψ-neighbor relation in the partitions of the word space is almost preserved in the partitions on the DC-mesh. Strictly, the Hamming distance of 1 in the word space is preserved in the Manhattan distance on the mesh, while the Hamming distance of 2 is mapped to the Manhattan distance greater than 0.*

Proof. Let's consider the ψ-neighbors of node (i, j). Then the $(p + N_{d'})$ number of ψ-neighbors are in the M^{ℓ} for column I_j (Theorem 1). Each of the p nodes has an index (i', j) (for a single-bit error in the parity part), and hence, $HD(i, i')$ between the nodes (i, j) and (i', j) equals one. So $MD(i, i') = 1$ for those node, owing to the mapping function G. Likewise, each of the $N_{d'}$ number of ψ-neighbors has an index (i'', j) (for an error in double bits both on parity positions), so that $HD(i, i'') = 2$. Generally, $MD(i, i'') \geq 2$ even with the mapping function G (though $MD(i, i'') = 2$ for the mesh of which size is less than or equal to 4×4 such as shown in Figs. 1 and 2). Since each of the k number of ψ-neighbors has an index (i, j') (for a single-bit error in the information part), it is in the M^{ℓ} for column $I_{j'}$. So the distance $HD(j, j')$ of 1 is preserved in the distance $MD(j, j')$ because the message is then sent from node $*j$ to node $*j'$ on the M^g. Each of the $(N_d - N_{d'})$ number of nodes has an index (i', j') (for an error in double bits, one of which is in the information part). Then the HD between nodes (i, j) and (i', j') equals $HD(i, i') + HD(j, j') = 2$, but $MD(i, i') + MD(j, j') = 1$ since $MD(j, j') = 1$ and $MD(i, i') = 0$; the latter is the distance from node $*j'$ of the M^g to node (i', j') via the bus.

4 Conclusions

We have proposed the DC-mesh for the dynamic clustering of nodes, as well as for easy layout on an LSI chip, exploiting the properties of partitioning based on the Hamming code. We first arranged the word space into an array so that the word with the parity value i and the information value j has index (i, j). Next, we mapped the indexed word space onto the node space of a 4-D (dimensional) mesh, according to the inverse of Gray code. Last, we contracted two dimensions of the obtained mesh for an easy layout.

The resultant DC-mesh consists of multiple local 2-D meshes and a single global 2-D mesh; each local mesh is connected to a node of the global mesh by a bus to compensate for the contracted dimentions. The diameter of the DC-mesh is equal to the maximum of the diameters of local and global meshes.

A dynamic cluster for a service in a target node is organized in a partition if its member node sends a request to its leader node, that is determined by

the addresses of the member and target nodes. Since the effective number of dimensions of the DC-mesh is greater than 2, the Hamming distance, that is less than 3, between the leader and non-leader words in each partition of the word space is almost preserved in the Manhattan distance between the corrsponding nodes on the DC-mesh.

To increase the bandwidth between the local and global meshes, we can exploit multiple buses and hence, multiple global meshes, leading to a fat DC-mesh. Another approach to DC networks is to connect the nodes in each partition with each other by a single bus, leading to a bused fat-hypercube (fat due to the connections corresponding to double-bit errors). In any case, we need to evaluate the performance of these DC networks by simulation with real applications.

References

1. M. Takesue: Ψ-Cubes: Recursive Bused Fat-Hypercubes for Multilevel Snoopy Caches. Proc. Int. Symp. on Parallel Architectures, Algorithms, and Networks. IEEE CS Press (1999) 62-67
2. M. Takesue: An Adaptive Local Protocol for Reducing Coherence Latency in Clustered Computations. Proc. ISCA 12th Int. Conf. on Parallel and Distributed Computing Systems (1999) 21–26
3. D. Matzke: Will Physical Scalability Sabotage Performance Gains? IEEE Computer **30** (1997) 37-39
4. T. Lovett and R. Clapp: STiNG: A ccNUMA Computer System for the Commercial Marketplace. Proc. 23th Int. Symp. on Computer Architectures (1996) 308-317
5. J. Laudon and D. Lenoski: The SGI Origin: A ccNUMA Highly Scalable Server. Proc. 24th Int. Symp. on Computer Architectures (1997) 241-251
6. D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam: The Stanford Dash Multiprocessor. IEEE Computer **25** (1992) 63–79
7. K. Olukotun et al.: The Case for a Single Chip Multiprocessor. Proc. 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (1996) 2-11
8. K. Sankaralingam et al.: Exploiting ILP, TLP, and DLP with the Polymorphous Architecture. Proc. 30th Int. Symp. on Computer Architectures (2003) 422-433
9. R. R. Iyer and L. N. Bhuyan: Design and Evaluation of a Switch Cache Architecture for CC-NUMA Multiprocessors. IEEE Trans. on Computers **49** (2000) 779-797.
10. M. Takesue: Schemes for Reducing Communication Latency in Regular Computations on DSM Multiprocessors. Proc. Int. Conf. on Parallel Processing (1998) 164-171
11. A. L. N. Reddy: Parallel Input/Output Architectures for Multiprocessors. Ph.D. dissertation, Dept. of Electrical and Computer Engr., Univ. of Illinois, Urbana (1990)
12. H.-L. Chen and N.-F. Tzeng: Efficient Resource Placement in Hypercubes Using Multiple-Adjacency Codes. IEEE Trans. on Computers **43** (1994) 23-33.
13. N.-F. Tzeng, and G.-L. Feng: Resource Allocation in Cube Network Systems Based on the Covering Radius. IEEE Trans. on Parallel and Distributed Systems **7** (1996) 328-342
14. Y. Saad and M. H. Schltz: Topological Properties of Hypercubes. IEEE Trans. on Computers **37** (1988) 867-872