Dual of a Complete Graph as an Interconnection Network *

S.Q. Zheng
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803
zheng@bit.csc.lsu.edu

Jie Wu
Department of Electrical & Computer Engineering
Florida Atlantic University
Boca Raton, FL 33431
jie@sunrise.cse.fau.edu

Abstract

A new class of interconnection networks, the hypernetworks, has been proposed recently. Hypernetworks are characterized by hypergraphs. Compared with point-to-point networks, they allow for increased resource-sharing and communication bandwidth utilization, and they are especially suitable for optical interconnects. In this paper, we propose a scheme for deriving new hypernetworks using hypergraph duals. As an example, we investigate the dual, K_n^*, of the n-vertex complete graph K_n, and show that it has many desirable properties. We also present a set of fundamental data communication algorithms for K_n^*. Our results indicate that the K_n^* hypernetwork is a useful and promising interconnection structure for high-performance parallel and distributed computing systems.

Key Words: algorithm, communication, hypernetwork, interconnection network, optical interconnect, parallel and distributed computing.

1 Introduction

The interprocessor communication performance is one of the most critical aspects of high-performance parallel and distributed computing systems. Designing high bandwidth, low latency and scalable interconnection networks is a great challenge faced by architecture designers. It is well known that VLSI computing systems are wire limited [4, 12, 14]. The connecting devices take up most chip area, and dictate the cost of the system. The system performance is limited by interconnection delays caused by resistance, capacitance, and inductance. Point-to-point networks, which are characterized by graphs, have been extensively investigated. High-dimensional point-to-point networks (e.g. the hypercube) have communication performances better than that of low-dimensional point-to-point networks under the constant-delay model that has no restriction on the bisection-size (and therefore no restriction on wire density). However, they are hard to implement and have excessively high cost. In a low-dimensional network (e.g. the 2-D mesh) the number of links is relatively small, the diameter large, and consequently, more congestions tend to occur. However, it is easier to implement. Dally [4] showed that, under the assumption of same wire bisection, low-dimensional networks outperform high-dimensional networks because they provide lower latency, less contention, higher hot-spot throughput and more resource (mainly wires) sharing. In recent years, we have seen the trend of seeking interconnection alternatives that combine the best features of low-dimensional networks, such as lower wire densities and higher wire sharing, and best features of high-dimensional networks, such as smaller network diameters and higher potential scalability. Evidently, such alternatives are no long pure point-to-point networks.

One of the major driving forces of these changes is the advance of optical interconnection technologies. Photons are non-charged particles, and do not naturally interact. Consequently, there are many desirable characteristics of optical interconnects: high speed (speed of light), increased fanout, high bandwidth, high reliability, supporting longer interconnection lengths, exhibiting low power requirements, and immunity to EMI with reduced crosstalk. These characteristics have significant system configuration and complexity implications [6, 7, 8]. For example, multiple-bus configurations with increased scalability are possible because of relaxed fanout and distance constraints. The optical fanout (which is the maximum number of processors that can be attached to an optical connecting device) is not bound by capacitance but by the power that must be delivered to each receiver to maintain a specified bit-error-rate, referred to as optical power budget. Processors can be arranged at increased physical distances. Resource sharing, achieved by multiple accesses of optical interconnect devices using time-division multiplexing (TDM), wavelength division multiplexing (WDM), code division multiplexing (CDM), space division multiplexing (SDM), or hybrid multiplexing [9, 10], is a fundamental advantage of optical networks.

Realizing that conventional graph theory is no longer adequate for the design and analysis of the new generation interconnection structures based on optical interconnect devices, a new class of interconnection networks, the hypernetworks, was proposed recently [15]. The class of hypernetworks is a generalization of point-to-point networks, and it contains point-to-point networks as a subclass. In

*Work supported in part by NSF grant ECS-9626215 and Louisiana grant LEQSF(1996-99)-RD-A-16.

a hypernetwork, the physical communication medium (a hyperlink) is accessible to multiple processors. The relaxation on the number of processors that can be connected by a link provides more design alternatives so that greater flexibilities in trade-offs of contradicting design goals are possible. The underlying graph theoretic tool for investigating hypernetworks is hypergraph theory [3]. Interested readers may refer to [15, 16, 17] for more justifications, design issues, implementation aspects, and some hypernetwork design examples.

In this paper, we propose a scheme for constructing a new hypernetwork from an existing one using the concept of dual graph in hypergraph theory. We show that the dual H^* of any given hypergraph H is a hypergraph that have some properties related to the properties of H. Thus, based on the properties of H, one can investigate the properties of H^*. Since the structure of H and its dual H^* can be drastically different, finding hypergraph duals can be considered as a general approach to the design of new hypernetworks. We demonstrate this approach by investigating the structure of the dual K_n^* of an n-vertex complete point-to-point network K_n. We discuss topological, fault-tolerance and data communication aspects of the K_n^* hypernetwork. The scalability and expandability of the K_n^* are also addressed. Our results indicate that the K_n^* hypernetwork is a useful and promising interconnection network for high-performance parallel and distributed computing systems.

2 Preliminaries

Hypergraphs are used as underlying graph models of hypernetworks. A *hypergraph* [3] $H = (V, E)$ consists of a set $V = \{v_1, v_2, \cdots, v_N\}$ of vertices, and a set $E = \{e_1, e_2, \cdots, e_m\}$ of hyperedges such that each e_i is a non-empty subset of V and $\cup_{i=1}^{m} e_i = V$. An edge e contains a vertex v if $v \in e$. If $e_i \subseteq e_j$ implies that $i = j$, then H is a *simple hypergraph.* When the cardinality of an edge e, denoted as $|e|$, is 1, it corresponds to a selfloop edge. If all the edges have cardinality 2, then H is a graph that corresponds to a point-to-point network. In this paper, we only consider simple hypergraphs (and graphs).

For a subset J of $\{1, 2, \cdots, m\}$, we call the hypergraph $H'(V', E')$ such that $E' = \{e_i | i \in J\}$ and $V' = \cup_{e_i \in E'} e_i$ the *partial hypergraph of H generated by the set J.* For a subset U of V, we call the hypergraph $H''(V'', E'')$ such that $E'' = \{e_i \cap U | 1 \leq i \leq m, e_i \cap U \neq \phi\}$ and $V'' = \cup_{e \in E''} e$ the *sub-hypergraph induced by the set U.*

The degree $d_H(v_i)$ of v_i in H is the number of edges in V that contain v_i. A hypergraph in which all the vertices have the same degree is said to be *regular.* The *degree of hypergraph H*, denoted by $\Delta(H)$, is defined as $\Delta(H) = \max_{v_i \in V} d_H(v_i)$. A regular hypergraph of degree k is called *k-regular hypergraph.* The *rank* $r(H)$ and *antirank* $s(H)$ of a hypergraph H is defined as $r(H) = \max_{1 \leq j \leq m} |e_j|$ and $s(H) = \min_{1 \leq j \leq m} |e_j|$, respectively. We say that H is a *uniform hypergraph* if $r(H) = s(H)$. A uniform hypergraph of rank k is called *k-uniform hypergraph.* A hypergraph is *vertex* (resp. *hyperedge*) *symmetric* if for its any two vertices (resp. hyperedges) v_i and v_j (resp. e_i and e_j) there is an automorphism of the hypergraph that maps v_i to v_j (resp. e_i to e_j).

In a hypergraph H, a path of length q is defined as a sequence $(v_{i_1}, e_{j_1}, v_{i_2}, e_{j_2}, \cdots, e_{j_q}, v_{i_{q+1}})$ such that (1) $v_{i_1}, v_{i_2}, \cdots, v_{i_{q+1}}$ are all distinct vertices of H; (2) $e_{j_1}, e_{j_2}, \cdots, e_{j_q}$ are all distinct edges of H; and (3) $v_{i_k}, v_{i_{k+1}} \in e_{j_k}$ for $k = 1, 2, \cdots, q$. A path from v_i to v_j, $i \neq j$, is a path in H with its end vertices being v_i and v_j. A hypergraph is *connected* if there is a path connecting any two vertices. We only consider connected hypergraphs. A hypergraph is *linear* if $|e_i \cap e_j| \leq 1$ for $i \neq j$, i.e., two distinct hyperedges share at most one common vertex. For any two distinct vertices v_i and v_j in a hypergraph H, the distance between them, denoted by $dis(v_i, v_j)$, is the length of the shortest path connecting them in H. Note that $dis(v_i, v_i) = 0$. The *diameter* of a hypergraph H, denoted by $\delta(H)$, is defined by $\delta(H) = \max_{v_i, v_j \in H} dis(v_i, v_j)$. More concepts in hypergraph theory can be found in [3].

A *hypernetwork M* is a network whose underlying structure is a hypergraph H, in which each vertex v_i corresponds to a unique processor P_i of M, and each hyperedge e_j corresponds to a *connector* that connects the processors represented by the vertices in e_j. A connector is loosely defined as an electronic or a photonic component through which messages are transmitted between connected processors, not necessarily simultaneously, in constant time. We call a connector a *hyperlink.* We restrict our discussions on hyperlink implementations to the optical domain.

The simplest implementation of a hyperlink is by a bus. Basically, there are three bus configurations: bidirectional bus, dual-bus and folded bus. Dual-bus and folded bus are unidirectional; they are particularly suitable to be implemented in optical domain. In a duel-bus system, every processor is connected to two unidirectional buses, and one bus attachment consists of a pair of transmitter (e.g. laser diode) and receiver (e.g. photo diode). The two buses transmit in opposite directions so that there is a path from every processor to every other processor in the system. In a folded bus system, each processor is attached to the bus twice, one attachment for reading and the other for writing. The bus is divided into two portions, the up-stream for processors to send data, and the down-stream for processors to receive data. With TDM or CDM, the performance of dual-bus and folded bus can be improved. A hyperlink can also be implemented by a ring which transmit data in pipelined fashion. Switches, which implement SDM, can be considered as hyperlinks. Free-space optical or optoelectronic switching device such as a spatial light modulator (SLM) [10] belongs to this class of hyperlinks. A star coupler [10, 13], which uses WDM, can be considered either as a generalized bus structure or a photonic switch, is another implementation of a hyperlink. Similarly, an ATM switch, which uses TDM, is also a hyperlink. In rest of this paper, the following pairs of terms are used interchangeably: (hyper)edges and (hyper)links, vertices and processors, point-to-point networks and graphs, and hypernetworks and hypergraphs.

The problem of designing efficient interconnection networks can be considered as a constrainted optimization problem. For example, the goal of designing point-to-point networks is to find well-structured graphs (whose ranks are fixed, as a constant 2) with small degrees and diameters. In hypernetwork design, the relaxation on the number of processors that can be connected by a hyperlink (i.e. the rank of the hyperlink) provides more design alternatives so that greater flexibilities in trade-offs of contradicting design goals are possible.

3 Dual Hypernetworks and K_n^* Hypernetwork

The *dual* of a hypergraph $H = (V, E)$ with vertex set $V = \{v_1, v_2, \cdots, v_N\}$ and hyperedge set $E = \{e_1, e_2, \cdots, e_m\}$ is a hypergraph $H^* = (V^*, E^*)$ with vertex set $V^* = \{v_1^*, v_2^*, \cdots, v_m^*\}$ and hyperedge set $E^* = \{e_1^*, e_2^*, \cdots, e_N^*\}$ such that v_j^* corresponds to e_j with hyperedges $e_i^* = \{v_j^* | v_i \in e_j \text{ in } H\}$.

Proposition 1 *H is r-uniform if and only if H^* is r-regular.*

Proposition 2 *The dual of a linear hypergraph is also linear.*

Proposition 3 *A hypergraph H is vertex symmetric if and only if H^* is hyperedge symmetric.*

Proposition 4 *The dual of a sub-hypergraph of H is a partial hypergraph of the dual hypergraph H^*.*

Since $(H^*)^* = H$, all the above propositions still hold after interchanging H with H^*.

Proposition 5 $\delta(H) - 1 \leq \delta(H^*) \leq \delta(H) + 1$.

Propositions 1 - 5 show that some properties of the dual hypergraph H^* of a given hypergraph H can be derived from properties of H. For example, if H is a ring, then H^* is isomorphic to H. However, in general, the structures of H and its dual H^* can be drastically different. Finding hypergraph duals can be considered as a general approach to the design of new hypernetworks.

We consider using the duals of point-to-point graphs as hypernetworks. Among all the hypergraphs derived from duals of point-to-point graphs, the dual, K_n^*, of the n-vertex complete graph K_n has the smallest m/N ratio when N is fixed, and smallest diameter, where m and N are the number of hyperedges and vertices, respectively. Properly labeling the vertices and hyperedges in a hypergraph can greatly simplify its use as a communication network. Vertex labels are used as processor addresses. Similarly, hyperedge labels are used as the unique names of hyperlinks. There are many ways to label the vertices and hyperedges of K_n^*. Although all different labeling schemes of K_n^* are equivalent because the symmetries of K_n^* (Proposition 3), we choose to define the K_n^* hypernetwork using an interesting scheme by which the connectivity of K_n^* can be concisely derived.

Definition 1 *Let $N_n = n(n-1)/2$ for $n > 0$. The K_n^* hypernetwork, $n \geq 3$, is a hypergraph that consists of N_n vertices, $v_1, v_2, ..., v_{N_n}$, and n hyperlinks, $e_1, e_2, ..., e_n$. The connectivity of K_n^* can be recursively defined as follows:*

(1) *K_3^* consists of three vertices v_1, v_2, and v_3, and three hyperlinks $e_1 = \{v_1, v_2\}$, $e_2 = \{v_1, v_3\}$, and $e_3 = \{v_2, v_3\}$.*

(2) *K_n^* is constructed from K_{n-1}^* by adding $n-1$ more vertices $v_{N_{n-1}+1}, v_{N_{n-1}+2}, ..., v_{N_{n-1}+n-1} = v_{N_n}$, and one more hyperlink e_n such that all the newly added $n-1$ vertices are connected to e_n and $v_{N_{n-1}+m}$ is connected to hyperlink e_m, $1 \leq m \leq n-1$.*

For a vertex v_i in K_n^*, we use i as its vertex label. Similarly, we use j as the label of hyperedge e_j of K_n^*. By a simple induction on n, it is easy to show that $(K_n^*)^*$ is a complete graph of n vertices. By the properties of K_n and above Propositions, we observe the following fact:

Fact 1 *K_n^* is 2-regular, $(n-1)$-uniform, linear, and vertex and hyperedge symmetric; the diameter of K_n^* is 1 if $n = 3$, and 2 if $n > 3$.*

In the following alternative definition, the connectivity of K_n^* hypernetwork is explicitly specified.

Definition 2 *Let $N_n = n(n-1)/2$ for $n > 0$. The K_n^* hypernetwork, where $n \geq 3$, is a hypergraph that consists of N_n vertices, $v_1, v_2, \cdots, v_{N_n}$, and n hyperlinks, $e_1, e_2, \cdots, e_n$. For any two distinct vertices v_i and v_j, let $u_i = \min\{r | N_r \geq i\}$, $u_j = \min\{s | N_s \geq j\}$, $l_i = i - N_{u_i - 1}$, and $l_j = j - N_{u_j - 1}$. v_i and v_j are connected by a hyperlink if and only if one of the following conditions holds: (1) $u_i = u_j$; (2) $u_i = l_j$; (3) $l_i = u_j$; or (4) $l_i = l_j$. Furthermore, if (1) or (2) holds then $v_i, v_j \in e_{u_i}$, and if (3) or (4) holds then $v_i, v_j \in e_{l_i}$.*

By a simple induction on n, one can easily see that Definitions 1 and 2 use the same vertex and hyperedge labeling schemes and they are equivalent. It is easy to verify that any vertex v_i of K_n^* is connected to exactly two hyperedges e_l and e_u, where $u = \min\{r | N_r \geq i\}$, and $l = i - N_{u-1}$. We call hyperedges e_l and e_u the *lower* and *upper hyperedge* of v, respectively. For any l and u such that $1 \leq l < u \leq n$, there is a unique vertex v_i that is connected to hyperedges e_l and e_u, and furthermore, $i = N_{u-1} + l$. Therefore, a vertex v_i of K_n^* can be uniquely identified by an ordered pair $\langle l, u \rangle$, $1 \leq l < u \leq n$.

The notion of $\langle l, u \rangle$ can be interpreted in another way. If we group those vertices that share the same upper hyperlink, $n-1$ groups (also called *blocks*) are formed. The k-th

($k > 0$) block contain k vertices. Vertices within each block are labeled based on the location of their lower hyperlinks in the block. Given vertex $\langle l, u \rangle$, $u-1$ is the block number of of the block it resides, and l is the rank of this vertex within the block. As shown in the next section, being able to address processors by hyperlinks is a useful property of the K_n^* hypernetwork for the design and analysis of parallel algorithms. Figure 1 shows the bus implementation of the K_6^* hypernetwork, whose corresponding K_6 is shown in Figure 2.

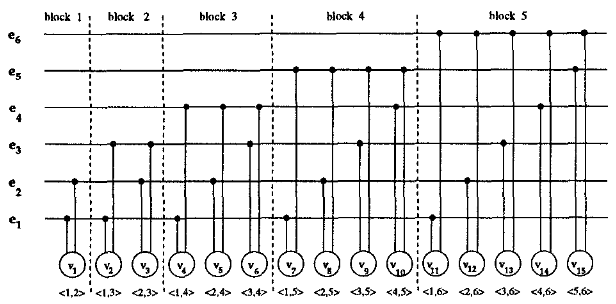

Figure 1: Bus implementation of K_6^*.

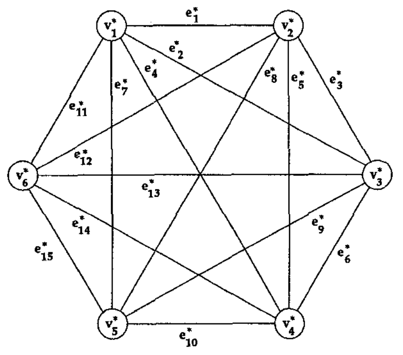

Figure 2: Complete graph K_6 corresponding to K_6^*.

The uniformity (i.e. all hyperlinks consist of the same number of processors), regularity (i.e. all the processors are included in the same number of hyperlinks), and linearity (i.e. no two hyperlink share more than one processor) of the K_n^* hypernetwork have important implications. Consider the bus-based implementations of hypernetworks. Here, uniformity and linearity imply that the bus loads are evenly distributed and minimized, and regularity implies simplified processor design since all the processors have the same interface circuitry. Vertex (hyperedge) symmetry is important for a hypergraph to be used as a hypernetwork, since it allows for all the processors (hyperlinks) to be treated as identical. Both Definitions 1 and 2 can be used to expand an existing K_n^* hypernetwork to a K_{n+1}^* hypernetwork without modifying the connections in K_n^*. The property that a larger hypernetwork can be easily constructed using smaller hypernetworks in the same class, when enhancement is desired, is call the the *expandability* of a hypernetwork. Clearly, the K_n^* hypernetwork is easy to expand. The incremental expandability of K_n^* is discussed in Section 5. Proposition 4 indicates that K_n^* can be partitioned into several smaller hypernetworks in the K_n^* family. This property is useful in designing parallel algorithms for K_n^* using the divide-and-conquer paradigm.

Consider the fault tolerance aspect of the K_n^* hypernetwork. We say that a hypernetwork H is *x-processor fault-tolerant* (resp. *y-hyperlink fault-tolerant*) if it remains connected when no more than any x processors (resp. y-hyperlinks) are removed. We have the following claim.

Theorem 1 *K_n^* is $(2n-3)$-processor fault-tolerant and 1-hyperlink fault-tolerant.*

The K_n^* hypernetwork may become infeasible when n is large. To improve scalability, we can use K_n^* as a building block to construct more complicated hypernetworks. For example, we may arrange $N = n^2(n-1)^2/4$ processors as an $[n(n-1)/2] \times [n(n-1)2]$ grid, and connect each row and column as a K_n^*. The resulting "two-dimensional" hypernetwork is regular, uniform, and linear. Both the degree and diameter of this hypernetwork are 4, and the rank of this hypernetwork is $n-1$, which is $O(N^{1/4})$. Similarly, we can construct a "three-dimensional" regular, uniform and linear hypernetworks of N processors with degree and diameter 6, and rank $O(N^{1/6})$. Compared with the K_n^* hypernetwork, these "multidimensional" hypernetworks have decreased processor failure tolerance and improved hyperlink failure tolerance.

4 Data Communication Algorithms for the K_n^* Hypernetwork

In this section, we demonstrate how to use the vertex and hyperedge labels to design data communication algorithms for the K_n^* hypernetwork. For simplicity, we assume bidirectional bus implementation of hyperlinks. We also assume that transmitting a word between two processors connected by a bus takes constant time. Since a bus is shared by all its connected processors, at most one pair of processors can communicate at any time instance. Bus communications can be either synchronous and asynchronous. In asynchronous mode communication, arbiters are needed to allocate the bus to processors in an online fashion. We assume a synchronous mode communication. Bus allocations, although operated dynamically, are predetermined by an off-line scheduling algorithm. This bus operational mode has been used in [5] for analyzing a multiple-bus interprocessor connection structure. We consider four types communication operations: one-to-one communications, one-to-many communications, many-to-one communications and many-to-many communications. We show that the performances of our algorithms are either optimal (*ROUTE* and *BROADCAST*) or optimal within a constant factor (*PERMUTATION*, *REDUCTION*, *TOTAL_EXCHANGE* and *PREFIX*). These communication algorithms constitute a powerful set of tools for designing parallel algorithms on the K_n^* hypernetwork.

4.1 One-to-One Communications

We consider two fundamental one-to-one communication operations, shortest path routing between two processors, and data exchange using a permutation.

4.1.1 Shortest path routing

The following algorithm can be used for data routing from $v_i = \langle l_i, u_i \rangle$ to $v_j = \langle l_j, u_j \rangle$ in K_n^*.

```
procedure ROUTE($\langle l_i, u_i \rangle$, $\langle l_j, u_j \rangle$)
begin
  if $u_i = u_j$ or $u_i = l_j$ then
    $\langle l_i, u_i \rangle$ sends the message to $\langle u_i, l_j \rangle$ using
    hyperlink $e_{u_i}$
  else if $l_i = u_j$ or $l_i = l_j$ then
    $\langle l_i, u_i \rangle$ sends the message to $\langle l_j, u_j \rangle$ using
    hyperlink $e_{l_i}$
  else
  /* $\langle l_i, u_i \rangle$ and $\langle l_j, u_j \rangle$ do not share a hyperlink */
    $l = \min\{l_i, l_j\}$;
    if $l_i = l$ then
      $\langle l_i, u_i \rangle$ sends the message to $\langle l_j, u_j \rangle$ through the
      path $(\langle l_i, u_i \rangle, e_{l_i}, \langle l_i, u_j \rangle, e_{u_j}, \langle l_j, u_j \rangle)$
    else $\langle l_i, u_i \rangle$ sends the message to $\langle l_j, u_j \rangle$ through
      the path $(\langle l_i, u_i \rangle, e_{u_i}, \langle l_j, u_i \rangle, e_{l_j}, \langle l_j, u_j \rangle)$
end
```

Theorem 2 *For any given pair of processors v_i and v_j in the K_n^* hypernetwork, algorithm ROUTE routes a message from v_i to v_j, or vise verser, along a shortest path.*

Example 1 *For $v_i = v_3 = \langle 2, 3 \rangle$ and $v_j = v_{14} = \langle 4, 6 \rangle$ in K_6^*, ROUTE finds a path $(v_3, e_3, v_{13} = \langle 3, 6 \rangle, e_6, v_{14})$.* □

4.1.2 Permutation

Permutation is a bijection on the set of processors in K_n^*. In a permutation communication operation, each processor $\langle a, b \rangle$ sends a message to another processor $\langle a', b' \rangle$, and each processor receives a message from exactly one processor. We use a set of N_n ordered processor pairs $(\langle a, b \rangle, \langle a', b' \rangle)$ to represent an permutation. In each pair $(\langle a, b \rangle, \langle a', b' \rangle)$, $\langle a, b \rangle$ and $\langle a', b' \rangle$ are called the source processor and destination processor of the pair, respectively. We use $A_{(\langle a,b \rangle, \langle a',b' \rangle)}$ to denote a message to be sent from $\langle a, b \rangle$ to $\langle a', b' \rangle$. A permutation is called a *total permutation* if $\langle a, b \rangle \neq \langle a', b' \rangle$ for all pairs; otherwise, it is called a *partial permutation.* We only consider total permutations, since a partial permutation can be carried out using a total permutation by masking out those processors which are mapped to themselves.

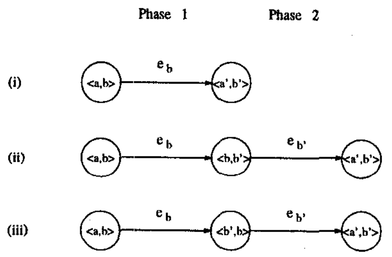

Figure 3: Routing paths used by algorithm *PERMUTATION* for messages $A_{(\langle a,b \rangle, \langle a',b' \rangle)}$. (i) $b = b'$, (ii) $b < b'$, and (iii) $b > b'$.

We present an algorithm *PERMUTATION* which performs a permutation operation efficiently. Depending on the values of b and b', algorithm *PERMUTATION* routes messages $A_{(\langle a,b \rangle, \langle a',b' \rangle)}$ along different paths of length at most 2. There are three cases: (i) $b = b'$, (ii) $b < b'$ and (iii) $b > b'$. For each of these three cases, algorithm *PERMUTATION* routes the messages strictly along paths shown in Figure 3. Based on these path patterns, we call a message a two-step message if it follows a path of length 2 (cases (ii) and (iii)) ; otherwise, it is called a one-step message (case (i)). Note that the source and destination processors of a two-step message may be distance 1 apart. Algorithm *PERMUTATION* consists of two phases. In the first phase, all one-step messages are sent to their destinations, and all two-step messages are routed to the intermediate processors of their routing paths. In the second phase, all two-step messages are sent to their destinations. In each phase, a hyperlink may be used to transmit more than one message. By our synchronous mode assumption, we know that the multiple accesses of the same hyperlink can be scheduled before the execution of the algorithm. Hence, transmissions on different hyperlinks can be performed in parallel, and transmissions on the same hyperlink are performed sequentially following a fixed schedule. Note that we can achieve the same effect if arbiters are used to allocate hyperlinks in an on-line fashion.

```
procedure PERMUTATION
begin
  /* Phase 1 */
  for all hyperlinks $e_k$ do in parallel
    Use $e_k$ to sequentially transmit those messages with
      $e_k$ assigned for their first step
  endfor
  /* Phase 2 */
  for all hyperlinks $e_k$ do in parallel
    Use $e_k$ to sequentially transmit the two-step
      messages with $e_k$ assigned for their second step
  endfor
end
```

Theorem 3 *Assuming bus hyperlinks of K_n^*, algorithm PERMUTATION carries out a permutation in at most $2(n-1)$ parallel message transmission steps, which is optimal for K_n^* within a constant factor. If transmitting a message on a hyperlink takes $O(1)$ time, the time complexity of PERMUTATION is optimal.*

Careful readers may notice that hyperlink e_1 is not used in *PERMUTATION*. If we let e_1 to share some communication load, the permutation performance can be slightly improved. In fact, by evenly distributing the communication load among hyperlinks, the performances of all algorithms presented in this paper, excluding *ROUTE* and *BROADCAST*, can be slightly improved. However, the modified algorithms will be more complicated.

4.2 One-to-Many Communication

Consider the following algorithm for broadcasting a message from any processor $v = \langle l, u\rangle$ to all the other processors in K_n^*.

```
procedure BROADCAST($\langle l,u\rangle$)
begin
  $\langle l,u\rangle$ broadcasts the message to all the processors
     connected by $e_u$;
  for all the processors $\langle a,b\rangle$ such that $a = u$ or $b = u$
  do in parallel
     if $a = u$ then $\langle a,b\rangle$ broadcasts the message to
        processors in $\{\langle i,b\rangle | i > a\}$ using $e_b$;
     if $b = u$ then $\langle a,b\rangle$ broadcasts the message to
        processors in $\{\langle a,j\rangle | j \neq b\}$ using $e_a$
  endfor
end
```

Theorem 4 *Algorithm BROADCAST broadcasts a message from any processor to all the other processors in the K_n^* hypernetwork in two steps, which is optimal. Furthermore each destination processor receives one and only one copy of the broadcast message.*

4.3 Many-to-One Communication

A reduction (or census, or fan-in) function is defined as a commutative and associative operation on a set of values, such as finding maximum, addition, logic or, etc. It can be carried out using a many-to-one communication operation. The following is an algorithm for performing a reduction operation specified by the *operator* + on a set of N_n values $A_1, A_2, \cdots, A_{N_n}$ stored in $v_1, v_2, \cdots, v_{N_n}$, and putting the final result in v_1. We assume that each processor v_i has a working register B_i (which is initialized to A_i). Again, we use an ordered pair $\langle l, u\rangle$ of hyperlinks to represent a processor. $A_{\langle l,u\rangle}$ and $B_{\langle l,u\rangle}$ represent A and B values associated with $\langle l, u\rangle$, respectively. Given any processor $\langle l, u\rangle$, procedure *TRANSFORM* is used to transform $\langle l, u\rangle$ to $\langle 1, 2\rangle$ and all the other $\langle a, b\rangle$ in K_n^* to $\langle a', b'\rangle$.

```
procedure TRANSFORM ($\langle l,u\rangle$)
begin
  for all $\langle a,b\rangle$ do in parallel
    if $a = 1$ and $b = 2$ then $\langle a',b'\rangle := \langle l,u\rangle$
    else if $a = 1$ then $\langle a',b'\rangle := \langle \min\{l,b\}, \max\{l,b\}\rangle$
    else if $a = 2$ then $\langle a',b'\rangle := \langle \min\{u,b\}, \max\{u,b\}\rangle$
    else if $a = l$ then $\langle a',b'\rangle := \langle 1,b\rangle$
    else if $b = u$ then $\langle a',b'\rangle := \langle 2,a\rangle$
  endfor
end
```

By the symmetry of the K_n^* hypernetwork, we know that the new identities $\langle a', b'\rangle$ assigned to processors of K_n^* satisfy the connectivities of K_n^*. We use $v_1 = \langle 1, 2\rangle$ to collect the final result.

```
procedure REDUCTION ($\langle 1,2\rangle$, +)
begin
  for all $\langle 1,j\rangle$ such that $j \geq 3$ do in parallel
      $\langle 1,j\rangle$ receives $A_{\langle 2,j\rangle}$ from $\langle 2,j\rangle$ using $e_j$ and
          performs $B_{\langle 1,j\rangle} := B_{\langle 1,j\rangle} + A_{\langle 2,j\rangle}$
  endfor;
  for $k = 3$ to $n$ do in parallel
      for all $\langle 1,j\rangle$ do in parallel
          if $j = 2$ then $\langle 1,j\rangle$ receives $B_{\langle 1,k\rangle}$ from $e_j$ and
          performs $B_{\langle 1,j\rangle} := B_{\langle 1,j\rangle} + B_{\langle 1,k\rangle}$
          else if $j > k$ then $\langle 1,j\rangle$ receives $A_{\langle k,j\rangle}$ from $e_j$
               and performs $B_{\langle 1,j\rangle} := B_{\langle 1,j\rangle} + A_{\langle k,j\rangle}$
               else do nothing
      endfor
  endfor
end
```

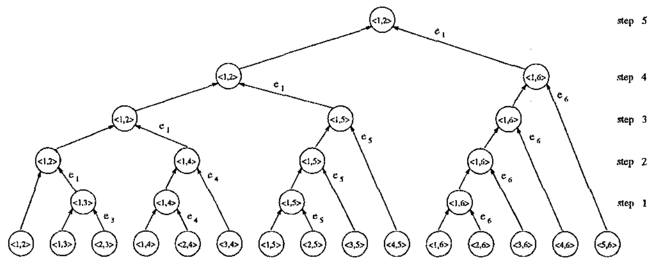

Figure 4: Communication pattern used by *REDUCTION* on K_6^*.

Theorem 5 *Assuming bus hyperlinks of K_n^*, the data communication of algorithm REDUCTION takes $n-1$ parallel communication steps. which is optimal for K_n^* within a constant factor. Assuming each + operation takes constant time, the time complexity of algorithm REDUCTION is $O(n)$, which is optimal for K_n^*.*

Similar performance can be achieved by a two-dimensional mesh point-to-point network of the same size.

However, the number of links in such a mesh M is a quadratic function of the number of hyperlinks in K_n^*, the degree of a M is 2 times the degree of K_n^*, and the data broadcasting time in M is $O(n)$ times that of K_n^*.

4.4 Many-to-Many Communication

We consider two cases: all-to-all communication and prefix computation. In all-to-all communication, each processor sends a message to all the other processors. It is also called the *total exchange* operation. The prefix computation can be considered as a many-to-many operation since many results are computed using many operands.

4.4.1 All-to-all communication

We can obtain an all-to-all communication by modifying the algorithm *REDUCTION*. The *operator* used is set union. After $n-1$ steps, v_1 receives all messages. Then, using two additional steps, v_1 broadcast all the N_n messages to all processors in K_n^*. A drawback of this algorithm is that each step transmits $O(N_n)$ messages along a hyperlink. We give another algorithm with improved performance. This algorithm consists of two phases: the first phase performs total-exchange within each block (intra-block) and the second phase performs total-exchange between blocks (inter-block).

```
procedure TOTAL_EXCHANGE
begin
  /* Phase 1:  intra-block total-exchange */
  for $j = 3$ to $n$ do in parallel
      for $i = 1$ to $j-1$ do
        Processor $\langle i,j\rangle$ broadcasts its message to
        processors in $\{\langle a,j\rangle | a \neq i\}$
        using $e_j$
      endfor
  endfor;
  Denote the set of messages processor $\langle i,j\rangle$ has by $S_{\langle i,j\rangle}$;
  /* Phase 2:  inter-block total-exchange */
  for $i = 2$ to $n$ do
    $\langle 1,i\rangle$ broadcasts $S_{\langle 1,i\rangle}$ to processors
    in $\{\langle 1,b\rangle | b \neq i\}$ using $e_1$;
    for all processors in $\{\langle 1,b\rangle | b \neq i\}$ do in parallel
      $\langle 1,b\rangle$ broadcasts $S_{\langle 1,i\rangle}$ it received to processors
      in $\{\langle a,b\rangle | a \neq 1\}$ using $e_b$
    endfor
  endfor
end
```

Theorem 6 *Assuming bus hyperlinks of K_n^*, the algorithm TOTAL_EXCHANGE requires $3(n-1)$ parallel communication steps, each hyperlink is used to transmit at most $n-1$ messages per step. If all the messages have the same length, this performance is optimal for K_n^* within a constant factor.*

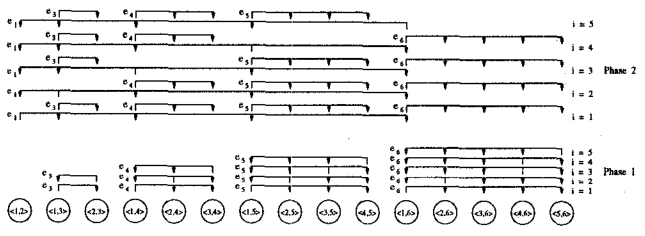

Figure 5: Communication patterns used by *TOTAL_EXCHANGE* on K_6^*.

4.4.2 Prefix Computation

Given a sequence $S = (a_1, a_1, \cdots, a_N)$ of N elements in a domain D, and an associative operation $\otimes$ on D, the prefix problem is to compute $z_i = a_1 \otimes a_2 \otimes \cdots \otimes a_i$ for $1 \leq i \leq N$. The prefix computation is a fundamental problem in parallel computing. It has a wide range of applications such as processor allocation, data distribution and alignment, data compaction, job scheduling, sorting, packet routing, matrix computation, linear recurrence, polynomial evaluation, graph algorithms, and general arithmetic formulae [11]. We use A_i to denote the operand value a_i initially stored in processor v_i.

```
procedure PREFIX ($\otimes$)
begin
  /* Phase 1 */
  for $j = 3$ to $n$ do in parallel
    for $i = 1$ to $j-1$ do
      Processor $\langle i,j\rangle$ broadcasts $A_{\langle i,j\rangle}$ to processors
      in $\{\langle a,j\rangle | a > i\}$ using $e_j$
    endfor
  endfor
  Each processor $\langle a,b\rangle$ performs $\otimes$ operation on all the
  $A$-values received, including its own $A$-value, and let
  the result be $X_{\langle a,b\rangle}$;
  /* Phase 2 */
  for $j = 2$ to $n-1$ do in parallel
    Processor $\langle j-1,j\rangle$ broadcasts $X_{\langle j-1,j\rangle}$ to
    processors in $\{\langle j-1,b\rangle | b > j\}$ using $e_{j-1}$
  endfor
  Let the value received by processor $\langle a,b\rangle$ is $Y_{\langle a,b\rangle}$;
  /* Phase 3 */
  for $j = 3$ to $n$ do in parallel
    for $i = 1$ to $j-2$ do
      Processor $\langle i,j\rangle$ broadcasts $Y_{\langle i,j\rangle}$ to processors
      in $\{\langle a,j\rangle | a \neq i\}$ using $e_j$
    endfor
  endfor
  for all the processors $\langle a,b\rangle$ do in parallel
    $\langle a,b\rangle$ performs $\otimes$ operation on $X_{\langle a,b\rangle}$ and all the
    $Y$-values it received
  endfor
end
```

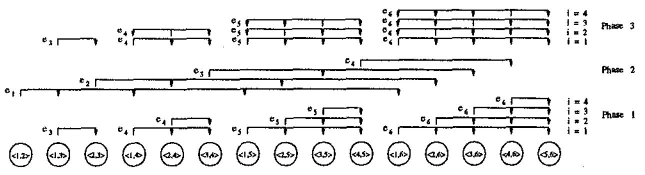

Figure 6: Communication patterns used by *PREFIX* on K_6^*.

Theorem 7 *Assuming bus hyperlinks of K_n^*, algorithm PREFIX carries out a prefix computation on K_n^*, with each processor initially contains one operand, in $2n-3$ parallel communication steps, which is optimal for K_n^* within a constant factor. Assuming each $\otimes$ operation takes constant time, the time complexity of algorithm PREFIX is $O(n)$, which is optimal for K_n^*.*

5 Incomplete K_n^* Hypernetwork

We observe that the gap, $N_n - N_{n-1} = n-1$, between K_{n-1}^* and K_n^* is not a constant. It is desirable that hypernetworks can be expanded with incremental size increases. For any given N such that $N_{n-1} < N < N_n$, we can construct a sub-hypergraph H of K_n^* such that $|V(H)| = N$ and $|E(H)| = n$. Such a sub-hypergraph is called an *incomplete K_n^** hypergraph.

Definition 3 *The incomplete K_n^* hypernetwork, where $n > 3$, of N vertices such that $N_{n-1} < N < N_n$ is the sub-hypergraph of K_n^* induced by vertex set $\{v_1, v_2, \cdots, v_N\}$.*

The vertices in an incomplete K_n^* can be divided into $n-1$ blocks. The i-th block has i vertices for $1 \le i < n-1$ as in K_n^*, and the $(n-1)$-th block has at least one vertex and at most $n-2$ vertices. For convenience, we call the $(n-1)$-th block of an incomplete K_n^* its *incomplete block.* We use k_n to denote the number of vertices in the incomplete block of an incomplete K_n^*. An incomplete K_n^* is linear and 2-regular, but it is not uniform, and not (vertex and hyperedge) symmetric. It is not difficult to prove that the diameter of incomplete K_n^* hypernetwork, where $n > 3$, is 2.

It is easy to verify that the shortest path routing communication algorithm *ROUTE* and data broadcasting algorithm *BROADCAST* presented in the previous section can be directly used for the incomplete K_n^* hypernetwork. Therefore, Theorem 2 and Theorem 4 hold for shortest path routing and data broadcasting operations on an incomplete K_n^* hypernetwork, respectively.

Algorithm *PERMUTATION* cannot be used for permutation operations on an incomplete K_n^*. We present a modified algorithm *PERMUTATION_INC.* Depending on the values of b and b', algorithm *PERMUTATION_INC* routes messages $A_{(\langle a,b\rangle,\langle a,b'\rangle)}$ along different paths of length at most 3. There are seven cases: (i) $b = b'$, (ii) (ii) $b < b' \neq n$, (iii) $b' < b \neq n$, (iv) $b' = n$ and $b \le \lceil k_n/2 \rceil$, (v) $b = n$ and $b' \le \lceil k_n/2 \rceil$, (vi) $b' = n$ and $b > \lceil k_n/2 \rceil$, and (vii) $b = n$ and $b' > \lceil k_n/2 \rceil$. Any message $A_{(\langle a,b\rangle,\langle a',b'\rangle)}$ in a total permutation satisfies one and only one of these seven conditions. For each of these cases, algorithm *PERMUTATION_INC* routes the messages strictly along paths shown in Figure 7. Algorithm *PERMUTATION_INC* is similar to *PERMUTATION.* It consists of three phases. In the first phase, all one-step messages are sent to their destinations, and all two-step and three-step messages are routed to the next processors on their routing paths. In the second phase, all two-step messages are sent to their destinations, and all three-step messages are sent to the third processors on their routing paths. Then, in the third phase, all three-step messages reach their destinations. As in algorithm *PERMUTATION*, in each phase of algorithm *PERMUTATION_INC*, messages are transmitted on different hyperlinks in parallel, and messages are transmitted on the same hyperlink sequentially. In the next theorem, we give a simple upper bound for the number of parallel message transmission steps of *PERMUTATION_INC.* Its actual performance can be much better.

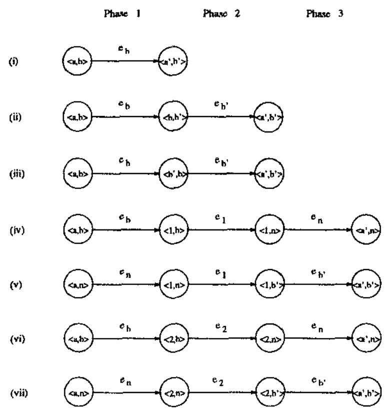

Figure 7: Routing paths used by algorithm *PERMUTATION_INC* for messages $A_{(\langle a,b\rangle,\langle a',b'\rangle)}$. (i) $b = b'$, (ii) (ii) $b < b' \neq n$, (iii) $b' < b \neq n$, (iv) $b' = n$ and $b \le \lceil k_n/2 \rceil$, (v) $b = n$ and $b' \le \lceil k_n/2 \rceil$, (vi) $b' = n$ and $b > \lceil k_n/2 \rceil$, and (vii) $b = n$ and $b' > \lceil k_n/2 \rceil$.

Theorem 8 *Assuming bus hyperlinks of an incomplete K_n^*, algorithm PERMUTATION_INC carries out a permutation in at most $n-2+\max\{n-2, k_n+1\}+k_n \le 3n-5$ parallel message transmission steps, where k_n is the number of processors in the incomplete block. The performance of this algorithm is optimal for the incomplete K_n^* within a constant factor. If transmitting a message on a hyperlink takes $O(1)$ time, the time complexity of PERMUTATION_INC is optimal.*

Consider the reduction operation. Since an incomplete K_n^* is not symmetric, we cannot use procedure *TRANSFORM* to relabel the processors. We adapt *REDUCTION* given in the previous section to an incomplete K_n^* by adding one operation: send the final result from $\langle 1,2\rangle$ to the final destination $\langle l,u\rangle$ using hyperlinks in at most two steps.

Corollary 1 *Assuming bus hyperlinks of the N-processor incomplete K_n^* hypernetwork, where $N_{n-1} < N < N_n$, algorithm REDUCTION can be extended to carry out a reduction operation on an incomplete K_n^* hypernetwork in $n+1$ parallel communication steps, which is optimal for the incomplete K_n^* within a constant factor. Assuming each $+$ operation takes constant time, the time complexity of this extended REDUCTION algorithm is $O(n)$, which is optimal for the incomplete K_n^*.*

It is simple to verify that the all-to-all data communication algorithm *TOTAL_EXCHANGE* presented in the previous section can be used for the incomplete K_n^* hypernetwork. This is done by treating all the processors v_j such that $j > N$ in the K_n^* hypernetwork as dummy processors that do not participate in communications.

Corollary 2 *Algorithm TOTAL_EXCHANGE carries out an all-to-all communication on an incomplete K_n^* hypernetwork in $3(n-1)$ parallel communication steps, each hyperlink is used to transmit at most $n-2$ messages per step. If all the messages have the same length, this performance is optimal for the incomplete K_n^* within a constant factor.*

The algorithm *PREFIX* given in the previous section also cannot be directly applied to an incomplete K_n^* hypernetwork. We have to modify it to obtain an algorithm with similar performance.

```
procedure PREFIX_INC (⊗)
begin
  /* Phase 1 */
  Same as the Phase 1 of PREFIX;
  /* Phase 2 */
  for $j = 2$ to $n-2$ do in parallel
      Processor $\langle j-1,j\rangle$ broadcasts its $X$-value to
      processors in $\{\langle j-1,b\rangle | b > j\}$ using $e_{j-1}$
  endfor
  Assume that the value received by processor $\langle a,b\rangle$
  is $Y_{\langle a,b\rangle}$;
  /* Phase 3 */
  for $j = 3$ to $n-1$ do in parallel
      for $i = 1$ to $j-2$ do
          Processor $\langle i,j\rangle$ broadcasts its $Y$-value to
          processors in $\{\langle a,j\rangle | a \neq i\}$ using $e_j$
      endfor
  endfor
  for all the processors $\langle a,b\rangle$ such that $b \neq n$
  do in parallel
      $\langle a,b\rangle$ performs $\otimes$ operation on $X_{\langle a,b\rangle}$ and all the
      $Y$-values it received, and let the result be $Z_{\langle a,b\rangle}$
  endfor
  /* Phase 4 */
  Use ROUTE to send $Z_{\langle n-2,n-1\rangle}$ from $\langle n-2,n-1\rangle$ to
  $\langle 1,n\rangle$;
  $\langle 1,n\rangle$ broadcasts $Z_{\langle n-2,n-1\rangle}$ to processors in $\{a,n | a \neq 1\}$
  using $e_n$;
  for all the processors $\langle a,b\rangle$ such that $b = n$
  do in parallel
      $Z_{\langle a,b\rangle} := X_{\langle a,b\rangle} \otimes Z_{\langle n-2,n-1\rangle}$
  endfor
end
```

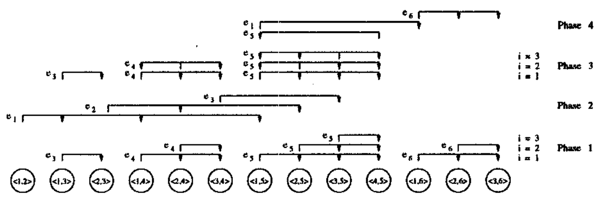

Figure 8: Communication patterns used by *PREFIX_INC* on an incomplete K_6^* of 13 processors.

Theorem 9 *Assuming bus hyperlinks of K_n^*, algorithm PREFIX_INC carries out a prefix computation on an incomplete K_n^* hypernetwork, with each processor initially contains one operand, in $2n-2$ parallel communication steps, which is optimal for the incomplete K_n^* within a constant factor. Assuming each $\otimes$ operation takes constant time, the time complexity of algorithm PREFIX_INC is $O(n)$, which is optimal for the incomplete K_n^*.*

6 Discussions

We say that a linear hypernetwork is *non-trivial* if it has at least 4 vertices, at least 2 hyperlinks, and each hyperlink contains at least 2 vertices. Let $H^+ = \{H | H$ is a non-trivial linear, regular hypergraph of degree 2 $\}$. We call the hypernetworks in H^+ the class of *degree-2 linear, regular hypernetworks.* For any H in H^+, its dual H^* is a point-to-point network. However, the dual of a point-to-point network may not be in H^+. For example, if a vertex v of a point-to-point network G has degree 1, then its corresponding hyperlink e^* in G^* contains exactly one vertex, and consequently, G^* is not in H^+. Let $G^+ = \{G | G$ is a point-to-point network such that it has at least 4 edge, and each vertex of G has degree greater than 1 $\}$. Clearly, for any point-to-point network G in G^+, we can obtain a non-trivial, linear, degree-2 hypernetwork H in H^+ by applying the dual operation to G. It is easy to prove that for any hypernetwork H in H^+, $\delta(H) \geq 2$. Those degree-2 hypernetworks that are excluded from H^+ are not interesting.

Among all the hypergraphs derived from duals of point-to-point graphs, the dual, K_n^*, of the n-vertex complete graph K_n has the smallest m/N ratio when N is fixed and smallest diameter, where m and N is the number of hyperedges and vertices, respectively. We have discussed the K_n^* hypernetwork in much detail. Between the high cost/performance of fully connected network K_n and low cost/performance of linearly connected network (a ring) are a set of point-to-point networks that constitute a wide range of trade-offs in cost and performance. For example, H can be point-to-point networks such as hypercubes, star graphs [1], chordal rings (including barrel shifters) [2], etc. The duals of these point-to-point networks also constitute a wide range of trade-offs in cost and performance.

In any point-to-point network, the number of links is at least equal to the number of processors (except a tree, in which the number of links is one less the number of processors). A trivial lower bound on the time complexity of parallel algorithm on a point-to-point network is the best sequential time divided by the number of processors. But in a hypernetwork, it is desirable that the number of hyperlinks is less than the number of processors due to cost-effectiveness consideration. In such a situation, the number of hyperlinks, the rank of hyperlinks and the hypernetwork degree are important factors in determining the lower bounds of time complexities of parallel algorithms, as demonstrated in our algorithm analysis. If we replace each bus by a crossbar switch, more efficient algorithms for the communication and computing problems we considered are possible. For example, using crossbar switches as hyperlinks of K_n^*, reduction and prefix operations can be implemented in $O(\log n)$ time, which is optimal. The $O(n^2)$ time complexity of total-exchange operation on the K_n^* hypernetwork cannot be improved because of the constant degree of K_n^*. We do not know if the time complexity of permutation operation on the K_n^* hypernetwork with crossbar switch hyperlinks can be reduced to $O(\log n)$.

Most discussions in this paper are restricted to constant degree (more specifically, degree 2) linear hypernetworks. Our approach can be easily generalized to the design and analysis of variable-degree and/or non-linear hypernetworks. Such networks may have better communication and fault-tolerance performances. Hypernetwork design is formulated as a constrainted hypergraph construction optimization problem. Hypergraph theory plays a central role in hypernetwork design and analysis. Simple hypergraph theory concepts, such as Steiner triple systems and hypergraph duals, have led to several interesting hypernetwork topologies as demonstrated in [17] and this paper. We would like to point out that hypernetwork designs are also related to block design problems in combinatorial mathematics, which in turn are related to algebra and number theory. We expect many new hypernetwork designs under the guidance of these theories.

References

[1] S. Akers, D. Harel, and B. Krishnamurthy, The Star Graph: an Attractive Alternative to the n-Cube, *Proceedings of 1987 International Conference on Parallel Processing*, pp. 393-400, 1987.

[2] B.W. Arden, and H. Lee, Analysis of Chordal Ring Network, *IEEE Transactions on Computers*, **30**, pp. 291-295, 1981.

[3] C. Berge *Hypergraphs*, North-Holland, 1989

[4] W.J. Dally, Performance Analysis of k-ary n-cube Interconnection Networks, *IEEE Trans. on Computers*, Vol. 39, pp. 775-785, 1990.

[5] O.M. Dighe, R. Vaidyanathan, and S.Q. Zheng, The Bus-Connected Ringed Tree: A Versatile Interconnection Network, to appear in *Journal of Parallel and Distributed Computing.*

[6] P.W. Dowd, Wavelength Division Multiple Access Channel Hypercube Processor Interconnection, *IEEE Trans. on Computers*, Vol 41, No. 10, pp. 1223-1241, 1992.

[7] M.R. Feldman, S.C. Esener, C.C. Guest and S.H. Lee, Comparison Between Optical and Electrical Interconnects Based on Power and Speed Considerations, *Appl. Opt.*, Vol. 27, pp. 1742-1751, 1988.

[8] M.R. Feldman and C.C. Guest, Interconnect Density Capabilities of Computer Generated Holograms for Optical Interconnection of Very Large Scale Integrated Circuits, *Appl. Opt.*, Vol. 28, pp. 3134-3173, 1989.

[9] P. E. Green, Jr., *Fiber Optical Networks*, Prentice Hall, 1993.

[10] J. Jahns and S.H. Lee (editors), *Optical Computing Hardware*, Academic Press, Inc., 1994.

[11] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercube* , Morgan Kaufmann Publishers, Inc., 1992, pp. 78-82, 239-244.

[12] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[13] C. Partridge, *Gigabit Networking*, Addison-Wesley, 1994.

[14] C.L. Seitz, Concurrent VLSI Architectures, *IEEE Trans. on Computers*, Vol. 33, pp. 1247-1265, 1984.

[15] S.Q. Zheng, Hypernetworks - A Class of Interconnection Networks with Increased Wire Sharing: Part I - Part IV, Technical Reports, Department of Compute Science, Louisiana State University, Baton Rouge, LA 70803, Dec., 1994.

[16] S.Q. Zheng, Hypercube Hypernetworks : Implementations of Hypercube with Increased Wire Sharing, *Proc of the 8th International Conf. on Parallel and Distributed Computing Systems*, pp. 452-457, 1995.

[17] S.Q. Zheng, Sparse Hypernetworks Based on Steiner Triple Systems, *Proc of 1995 International Conf. on Parallel Processing*, pp. I.92 - I.95, 1995.