Four Algorithms on the Swapped Dragonfly

Richard Draper

Abstract

A Swapped Dragonfly with \( M \) routers per group and \( K \) global ports per router is denoted \( D_3(K,M) \). A Swapped Dragonfly with \( K \) and \( M \) restricted is studied in this paper. There are four cases. In each case the resulting Swapped Dragonfly has a special utility:

- If \( K = L^2 \), an \( LM \times LM \) matrix product can be performed in \( LM \) rounds. An \( n \times n \) matrix product may be performed in \( n^2/LM \) rounds. Each round takes four router hops.
- If \( K = ks \) and \( M = ms \) an all-to-all exchange may be performed in \( KM^2/s \) rounds. An all-to-all exchange of size \( n \geq KM^2 \) may be performed in \( n^2/KM^2 \) rounds.
- If \( K = 2^k \) and \( M = 2^n \), a dilation three emulation of the \((k + 2m)\) dimension Boolean hypercube exists. An ascend-descend algorithm may be performed at twice the cost on the hypercube.
- \( D_3(K,M) \) contains \( M \) adjacent depth four edge-disjoint spanning trees. Equipped with a synchronized source-vector packet header, it can perform \( n \) broadcast in \( 3n/M \) rounds.

The rounds in these algorithms are free of link conflicts. These results are compared with algorithms on a hypercube. Comparisons with the fully populated Dragonfly are also made. These results are more applicable than the special cases because \( D_3(K,M) \) contains emulations of every Swapped Dragonfly with \( J \leq K \) and/or \( L \leq M \).

The Swapped Dragonfly is a new approach to the Dragonfly concept. It behaves much like a three-dimensional mesh. The underlying graph is defined before the network is defined. The underlying graph of a Dragonfly is not defined until ports are identified on routers and links are defined by connections with ports.

Keywords: Swapped Interconnection Network, Matrix Product, All-to-all, Universal Exchange, Boolean Hypercube, Ascend-descend algorithm, Broadcast, Edge-disjoint spanning tree.

1 Introduction

This paper presents algorithms which can be performed on a Swapped Dragonfly interconnection network more efficiently than they can be performed on a fully-populated Dragonfly interconnection network. A knowledge of a Swapped Dragonfly \([1] \) and a fully populated Dragonfly \([11] \) is assumed. (The term “interconnection network” is elided hereafter.) A fully-populated Dragonfly will be called a Maximal Dragonfly. The information about a Maximal Dragonfly pertinent to this discussion is not generally available so explanations will be given when necessary.

The Swapped Dragonfly \([5] \) is an interconnection network having \( KM^2 \) routers. Routers have coordinates \((c \mod K, d \mod M, p \mod M)\). Connectivity is given by:

\[
(c,d,p) \xrightarrow{1} (c,d,p') \quad \text{and} \quad (c,d,p) \xrightarrow{8} (c',p,d)
\]

Note the swap of \( d \) and \( p \). These are referred to as local and global connections, respectively. Conceptually the network is made up of \( K \) cabinets containing \( M \) drawers. Each drawer \( c \) has \( M \) routers. There are \( K \) global ports and \( M - 1 \) local ports on each router. Connections are bidirectional. The local connections connect the routers in a drawer in a complete graph. The Swapped Dragonfly is denoted \( D_3(K,M) \).

The routers of a Maximal Dragonfly and a Swapped Dragonfly are identical. A Maximal Dragonfly has \( KM^2 + M \) routers. Routers have coordinates \((g,p), 0 \leq g < KM + 1 \) and \( 1 \leq p < M \). The network is constructed from groups (a.k.a drawers) consisting of \( M \) routers connected in a complete graph. The \( K \) global ports of the routers are used to connect groups to one-another such that the groups are connected in a complete graph of order \( KM + 1 \). The underlying graph of the network is a graph called a replacement graph \([8] \) of a complete graph of order \( M \) in a complete graph of order \( KM + 1 \). There are an enormous number of ways to construct a replacement graph.
graph [6], but only one way has been used in the design of an actual machine [1], [2]. This is the one to which the term Maximal Dragonfly in this paper refers. It is denoted \( \text{MDF}^2(K,M) \).

\( D^3(K,M) \) is treated as a packet switching network using source-vector routing. At \((c,d,p)\) a source-vector \((\gamma,\pi,\delta)\) produces the path

\[
(c,d,p) \xrightarrow{\delta} (c,d,p+\delta) \xrightarrow{\pi} (c+\gamma,p+\delta,d) \xrightarrow{\pi} (c+\gamma,p+\delta,d+\pi)
\]

The following four properties of \( D^3(K,M) \) were established in [5] and will be referred to in this paper.

1. Simultaneously, every router can send a packet with header \((\gamma,\pi,\delta)\) without link conflicts. The result is a permutation of the routers.

2. The routers of \( D^3(K,M) \) with \( c \) in a set of size \( J < K \) and \( p \) in a set of size \( L < M \) are connected in a closed sub-network isomorphic to \( D^3(J,L) \).

3. If \( \gamma \neq \gamma', \delta \neq \delta', \pi \neq \pi' \) two routers can simultaneously send packets with headers \((\gamma,\pi,\delta)\) and \((\gamma',\pi',\delta')\) without link conflicts.

4. \( D^3(K,M) \) scales linearly in \( K \) and quadratically in \( M \).

This paper examines four special cases of the Swapped Dragonfly: \( D^3(K^2,M), D^3(k^2,ms), D^3(2^k,2^m) \) and \( D^3(K,M) \) with synchronizing header \([v;\gamma,\pi,\delta]\). Each is useful for a particular algorithm. On \( D^3(K^2,M) \) vector-matrix multiply takes one round of four hops with two off-and-ons to calculate products and accumulate sums. The algorithm can be in place or out of place. On \( D^3(k^2,ms) \) an all-to-all exchange takes only \( KM^2/s \) rounds. \( D^3(2^k,2^m) \) emulates a \( k+2m \) Boolean-hypercube with maximal dilation three and average dilation two. \( D^3(K,M) \) with synchronizing header can perform \( M \)-broadcasts in one round of five router hops. \( D^3(2^k,2^m) \) with synchronizing header can emulate \( (k+2m) \) Boolean-hypercube with uniform dilation four.

Property 2 of \((D^3(K,M))\) makes it possible to use these special Swapped Dragonflies "inside" a general \( D^3(K,M) \) for a range of values of \( K \) and \( M \) at a cost of some of the algorithm’s performance.

2 \( D^3(K^2,M) \)

\( D^3(K^2,M) \) may be viewed as a \( K \times K \) array of \( M \times M \) blocks. The index set for such an array is \( \{(s,t,u,v) : 0 \leq s, t < K \text{ and } 0 \leq u, v < M\} \). The index set \((s,t,u,v)\) is assigned to \((c,d,p) = (s+tK,u,v)\). The arithmetic is done mod \( K^2 \) and canonical values are chosen as representatives of equivalence classes. A \( KM \times KM \) matrix stored in this way can be transposed in a single global hop \( \gamma = (s+th-(t+sK)) \).

A row vector \((s,u)\) refers to \((s,*,u,*)\) where * denotes all possible values of the coordinate. The term vector will refer to the index set or to a \( KM\)-vector stored at the index set. A column vector \((t,v)\) refers to \((*,t,*,v)\). Note in \( D^3(K^2,M) \) a row vector \((s,u)\) is stored at \((s+Ku,*,u)\) and a column vector \((t,v)\) is stored at \((*,tK,*,v)\). Note that a vector is stored on \( KM \) routers.

**Theorem 1** On \( D^3(K^2,M) \) a \( KM \times KM \) matrix product takes \( KM \) rounds. Each round takes 4 network hops and two off and ons.

**Proof:** To form the product, \( VA \), of a row vector \( V \) at \((s,u)\) and an \( RM \times RM \) matrix \( A \) on \( D^3(K^2,M) \) two phases are required. The first is to bring the row vector into juxtaposition with the columns of \( A \) so that \( VA_{l,t,v,j,v'} \) can be computed. The second phase is to accumulate these values at the row vector \((s,u)\). The result is an in-place algorithm.

The first phase is done by broadcasting element \((t,v)\) at \((s,u)\) to all locations in row \((t,v)\) of \( A \):

\[
(s+tk,u,v) \xrightarrow{G} (s+Kv,u,v) \xrightarrow{L} (s+K,v,v) \forall(t,v).
\]  

(2.1)

The first broadcast is over all global ports of \((s+tk,u,v)\). The second broadcast is over all local ports of \((t+sK,v,u)\) because \( t \neq t_1 \). The center routers are all distinct. If \( t \neq t_1 \) the center routers are in different drawers. Therefore, \( KM \) broadcasts can occur simultaneously. The first phase is completed in two network hops. That is, row \((s,u)\) can be brought into juxtaposition with every column of \( A \) in two network hops. The juxtaposed pairs hop off, are multiplied, and the product hops on. A path in \((2.1)\) has the form:

\[
(s+tK,u,v) \xrightarrow{G} (t+t'K,v,u) \xrightarrow{L} (t+t'K,v,v') \forall(t,v).
\]  

(2.2)

The accumulation phase of the algorithm requires \((t,t',v,v')\) to map to \((s,t,u,v)\) for all \((t',v')\). The path \((2.2)\) reverses path \((2.2)\).

\[
(t+t'K,v,v') \xrightarrow{L} (t+t'K,v,u) \xrightarrow{G} (s+tK,u,v) \forall(t',v').
\]  

(2.3)

If \((t_1,v_1) \neq (t,v)\) the path

\[
(t_1+t'K,v_1,v') \xrightarrow{L} (t_1+t'K,v_1,u) \xrightarrow{G} (s+tK,u,v),
\]

\(^*\)Isomorphism is used here to mean dilation one emulation.
does not conflict with [2] because the center routers are distinct. Therefore, the path can be followed simultaneously for all \((t, v)\). \(M\) values arrive at \((t + tK, v, u)\) after the local step. They hop off, are accumulated and the partial sum hops on. \(K\) partial sums arrive at \((s + tK, u, v)\) after the global step. They hop off and are accumulated. The value is \(\sum_{t,v} V_{t,v} A_{t',v',u'}\) which is element \((t', v')\) of the vector-matrix product. A matrix multiply takes \(KM\) rounds.

The vector-matrix multiply takes one round consisting of four network hops and two off-and-ons to perform arithmetic. The time is \(4t_w + 2t_s\), where \(t_w\) denotes router latency and \(t_s\) is time for the on-and-off. It is presented as an in-place algorithm. By modifying \(s\) and \(u\) in the last two hops, it can be converted to an out of place algorithm. □

The expected situation for this algorithm is an \(n\)-vector \(V\) and a \(n \times n\) matrix \(A\) with \(n \geq KM\). Let \(X = n/KM\). There is an \(X\) subvector \(V_{t, i, a, u}\) and an \(X \times X\) submatrix \(A_{t', v', u'}\) for all \((t', t, v, v'); (s, u)\) is fixed. The broadcast path is used X times to bring all \(V_{t, i, a, u}\) into juxtaposition with the columns \(A_{t', v', u', i, t'}\). The \(X\) vector \(V_{t, i, a, u}\) hops off the network so that the vector-matrix product \(\sum_{t,v} V_{t,v} A_{t',v',u',v}\) can be computed. (This was a scalar product when \(n = KM\).) The result is \((VA)_{t', i, a, u, v'}\). For each \((t', t, v, v')\) it is an \(X\) vector. These are accumulated using path \(X\) times for each \((t', t, v, v')\). The result is an \(X\) vector at \((s, t, u, v)\). The \(n\) vector \((s, *, u, *)\) of these \(X\) vectors is \(VA\). The vector multiply takes \(n/KM\) rounds. The matrix multiply takes \(n^2/KM\) rounds because the vector multiply must be used \(n\) times. There is a cost for the \(X \times X\) product which is independent of the network cost.

**Theorem 2** On \(D3(K^2, M)\) an \(n \times n\) matrix product with \(n >> KM\) takes \(n^2/KM\) rounds. Each round takes \(4t_w + 2t_s\) time where \(t_w\) is router latency and \(t_s\) is time for off-and-on.

It is possible to transfer this algorithm to a \(KM \times KM\) matrix on \(D3(K, M)\). It requires storing \((s, t, u, v)\) at \((s, u, v)\) ∀. The memory requirements are increased by a factor of \(K\) at every router. A row vector \((s, u)\) is stored as \(K\)-tuples at \((s, u, *)\) and a column vector \((t, v)\) is stored at \((*, *, u)\) but only one entry of the vector is at each \((c, d, v)\). A vector-matrix multiply takes \(K\) rounds and a matrix multiply takes \(K^2M\) rounds.

If \((L + 1)^2 > K > L^2\), it is faster to do a vector-matrix multiply on \(D3(L^2, M)\) than on \(D3(K, M)\) because a vector of length \(KM\) takes \(K/L\) rounds on \(D3(L^2, M)\) and \(K\) rounds on \(D3(K, M)\). This procedure is made possible by property 2 of the Swapped Dragonfly.

The following table presents the network cost of matrix multiplication algorithms. The notation is \(n \times n\) matrices with \(P\) processors. The Cannon algorithm [3] was originally done on a \(\sqrt{P} \times \sqrt{P}\) mesh. The other algorithms are on a Boolean hypercube with \(P\) nodes.

| Algorithm | Network Cost |
|-----------|--------------|
| DNS       | \(2t_wn^2/\sqrt{P}\) |
| HJE       | \(2t_wn^2/\sqrt{P} \log P\) |
| DNS       | \(2t_wn^2/\sqrt{P}\) |
| GS        | \(3t_wn^2/P^{2/3} \log P\) |
| DNS       | \(4t_wn^2/P^{2/3}\) |

3 \(D3(ks, ms)\)

**Theorem 3** On the Swapped Dragonfly, \(D3(ks, ms)\), an all-to-all exchange among \(n \geq KM^2\) nodes can be performed in \(n^2/KM^2\) rounds.

Proof: In \(Z\) mod \(M\), \(s\) generates a subgroup \(G = \{0, s, \cdots, (m-1)s\}\). \(G\) has \(s\) cosets. Denote them \(\{0\}, \cdots, \{s-1\}\). An analogous statement applies to \(K\). An example is instructive. If \(m = 5\) and \(s = 3\) then \(G = \{0, 3, 6, 9, 12\}\) and a coset \([t]\) is \(t + G\);

\[
\begin{align*}
[0] &= \{0, 3, 6, 9, 12\} \\
[1] &= \{1, 4, 7, 10, 13\} \\
[2] &= \{2, 5, 8, 11, 14\}
\end{align*}
\]

Notice that the cosets partition \(Z\) mod \(M\) into three disjoint sets and that the columns partition it into five disjoint sets. The second partition is called a dual partition.

Consider the following array which is called a disagreeable array (DA).

\[
\begin{array}{cccccc}
   i & 0 & 1 & \cdots & s-1 \\
   \gamma & [0] & [1] & \cdots & [s-1] \\
   \pi & [0] & [1] & \cdots & [s-1] \\
   \delta & [0] & [1] & \cdots & [s-1]
\end{array}
\]
Each column contains \(km^2\) vectors \((\gamma, \pi, \delta)\); \(k\) values of \(\gamma\), \(m\) values of \(\pi\), and \(m\) values of \(\delta\). If \(i \neq j\) and \((\gamma, \pi, \delta)\) is a vector from column \(i\) and \((\gamma', \pi', \delta')\) is a vector from column \(j\) then \(\gamma \neq \gamma', \pi \neq \pi'\), and \(\delta \neq \delta'\). Therefore, every router can simultaneously send packets on the vector paths \((\gamma, \pi, \delta)\) and \((\gamma', \pi', \delta')\) without link conflicts by property[3] If one vector is taken from each column of the array, every router can simultaneously send \(s\) packets without link conflict. This is denoted \((lgl)^s\) and is a round of the algorithm being developed here. It takes 3 hops. There are \(km^2\) vectors in each column of the DA. Therefore, there are \(km^2\) rounds \((lgl)^s\) delivering \(km^2s\) packets. Note that each vector is used only once in this process.

Cyclically shifting row \(\pi\) one place to the left produces a new DA. No vector in the new array appeared in the previous array. Suppose \((\gamma, \pi, \delta)\) is in the original array and \((\gamma', \pi', \delta')\) is in this array. If \(\pi = \pi', (\gamma, \delta) \neq (\gamma', \delta')\) because \(\pi\) and \(\pi'\) are in the same coset so \((\gamma, \delta)\) and \((\gamma', \delta')\) are in different cosets. If \((\gamma, \delta) = (\gamma', \delta')\) then \(\pi\) and \(\pi'\) are in different cosets. Therefore, this new array produces \(km^2\) rounds \((lgl)^s\) delivering \(km^2s\) packets to a different set of destinations. A series of \(s^2\) left shifts of the bottom two rows produces \(s^2\) distinct DAs. Each yields \(km^2s\) vectors in \(km^2\) rounds \((lgl)^s\). The entire set is \(km^2s^3 = KM^2\) in \(KM^2/s\) rounds. If \(n = XKM^2, K = ks\) and \(M = ms\), an all-to-all exchange between \(n\) nodes take \(X^2KM^2/s = n^2/M^KM^2/s\) rounds. □

To convert this argument into an algorithm it is necessary to specify the order in which elements are chosen from the cosets in a DA. A single column of the first DA has the form:

\[
\begin{align*}
\gamma & | i & i+s, \cdots, i+(k-1)s \mod K \\
\pi & | j & j+s, \cdots, j+(m-1)s \mod M \\
\delta & | k & k+s, \cdots, k+(m-1)s \mod M
\end{align*}
\]

The left shifts are determined by \(\phi = \mu + vs; 0 \leq \phi < s^2\). Row \(\pi\) of the array is shifted left \(\mu\) times and row \(\delta\) is shifted left \(v\) times. Selecting the entries of a vector amounts to choosing an entry of the partition/dual partition array. The choice made is in the same position for each column of the DA. The following converts this observation to an algorithm. The entries of a vector selected from a DA are determined by \(\lambda = a+bm+c\pi^2; 0 \leq \lambda < km^2\). The vector \((\gamma, \pi, \delta)\) is taken from location \(c\) of the \(\gamma\) row, \(a\) of the \(\pi\) row and \(b\) of the \(\delta\) row. This algorithm is referred to as the doubly-parallel all-to-all algorithm. It has \(KM^2/s\) rounds. If \(K\) and \(M\) are relatively prime it reduces to the all-to-all algorithm in [5] that takes \(KM^2\) rounds.

The doubly parallel algorithm can be employed even if \(K\) and \(M\) are relatively prime by finding \(J < K\) and \(L < M\) for which \(J\) and \(L\) have a common factor. As soon as an all-to-all involves \(X > 1\) items at every router, the cost is multiplied by \(X^2\). If \(K\) and \(M\) are relatively prime, going from \(D3(K, M)\) to \(D3(k, m)\) for \(K > ks\) and \(M > ms\) will produce a doubly-parallel algorithm with fewer than \(km^2\) rounds if \(KM^2 < sJL^2\) because

\[
\frac{JL^2}{s} \left( \frac{KM^2}{JL^2} \right)^2 < KM^2 \text{ iff } \frac{KM^2}{s} \left( \frac{KM^2}{JL^2} \right)^s < KM^2 \text{ iff } \frac{KM^2}{JL^2} < s.
\]

This generally works if \(K - J\) and \(M - L\) are small. For example, if \(K = 7\) and \(M = 16, J = 5, L = 15\), and \(s = 5\) then \(KM^2/JL^2 = 1.59\) so the doubly-parallel algorithm on \(KM^2\) objects runs on \(D3(J, L)\) has 225 \((1.59)^2 = 569\) rounds which is far less than 1792 rounds on \(D3(7, 16)\).

Johnsson and Ho [10] developed an all-to-all on a Boolean hypercube of \(P\) processors with network time \(t_nP/2\). For a set of size \(n \geq P\) the time is \(n^2/P\). On \(D3(k, ms)\) the network time is \(n^2/P\).

The doubly-parallel algorithm can be pipelined in several ways. The round schedules are

\[
\begin{align*}
(l & g) s \\
(l & g) s & (l & g) s \\
(l & g) s & (l & g) s & (l & g) s \\
(l & g) s & (l & g) s & (l & g) s & \cdots
\end{align*}
\]

The first is a cost one schedule, the second is cost 2, and the third is cost 3. It is obvious that Schedules 2 and 3 can be used without link contention. Therefore, an all-to-all algorithm runs in time \(2KM^2/s\) or \(3KM^2/s\) if \(K = ks\) and \(M = ms\).

Clearly, there is a potential for intraround conflicts between every other row in schedule 1. Schedule 1 can be used because of the care with which \(\pi\) and \(\delta\) were chosen. Suppose row \(\pi\) of the original DA is left-shifted \(\mu\) places. Let \(t' = t + \mu \mod s\). Column \(t\) of the new DA contains \(\{t' + 0, t' + s, \ldots, t' + (m-1)s\}\) in row \(\pi\). Suppose \(a\) determines which element is to be selected from each cell of row \(\pi\). After the shift \(t' + as\) is selected from column \(t\). That is, the set of \(s\) values of \(\pi\) being selected for a round of the all-to-all algorithm is \(\{t' + a, t' + a + 1, \ldots, t' + a + s - 1\}\). This is a cell of the dual partition of the coset partition of \(s\) in \(Z \mod M\). This is also true of the
set of $\delta s$ in a round. Therefore, a conflict of $\delta(i)$ with $\pi(i+2)$ in Schedule 1 means that the set of $\pi$'s of round $i$ is equal to the set of $\delta$s of round $i+2$. A single delay eliminates the conflict.

Here is an example of what happens using the earlier example for $\pi$ and $\delta$. $\gamma$ can be ignored because the only possible conflict does not involve $\gamma$. Suppose $(\mu, \nu) = (0, 2)$ and $(a, b) = (1, 2)$. The DA is

$$
\begin{array}{c|c|c}
 i & 0 & 1 & 2 \\
\pi & {0, 3, 6, 9, 12} & {1, 4, 7, 10, 13} & {2, 5, 8, 11, 14} \\
\delta & {2, 5, 8, 11, 14} & {0, 3, 6, 9, 12} & {1, 4, 7, 10, 13} \\
\end{array}
$$

$(a, b)$  3 vectors in round $(\mu, \nu, a, b)$

$$(1, 2) \quad \begin{pmatrix} \pi \\ \delta \end{pmatrix} = \begin{pmatrix} 3 \\ 8 \end{pmatrix} \begin{pmatrix} 4 \\ 6 \end{pmatrix} \begin{pmatrix} 5 \\ 7 \end{pmatrix}$$

$$
(2, 2) \quad \begin{pmatrix} 6 \\ 8 \end{pmatrix} \begin{pmatrix} 7 \\ 6 \end{pmatrix} \begin{pmatrix} 8 \\ 7 \end{pmatrix}$$

$$
(3, 2) \quad \begin{pmatrix} 9 \\ 8 \end{pmatrix} \begin{pmatrix} 10 \\ 6 \end{pmatrix} \begin{pmatrix} 11 \\ 7 \end{pmatrix}$$

$$
(4, 2) \quad \begin{pmatrix} 12 \\ 8 \end{pmatrix} \begin{pmatrix} 13 \\ 6 \end{pmatrix} \begin{pmatrix} 14 \\ 7 \end{pmatrix} \quad b = a + 2 \mod M
$$

Note that $\pi$ in row $(2, 2)$ and $\delta$ in row $(4, 2)$ take the same set of values. In Schedule 1 this causes a conflict, actually 3 conflicts. However, each row of vectors is sent simultaneously so a one hop delay resolves the conflict. The condition $b = a + 2 \mod m$ occurs $m$ times. Therefore, for each DA a delay occurs $km$ times in $km^2$ rounds. There are $s^2$ DA's in the algorithm, so in $km^2s^2 = KM^2/s$ rounds there are $kms^2 = KM$ delays. Some delays will be successive as rows $(1, 2)$ and $(3, 2)$ demonstrate. Schedule 1 can only be used if $s \leq M/2$ because every round uses $2s$ local links.

The preceding discussion has proven the following: If $K = ks$ and $M = ms$, then

Using Schedule 1, if $s \leq M/2$ the doubly-parallel algorithm takes time $((KM^2/s + KM)/s)t_w$. Using Schedule 2 the doubly-parallel algorithm takes $(2KM^2/s)t_w$ and is conflict free. Using Schedule 3, the doubly-parallel algorithm takes time $3KM^2t_w/s$ and is conflict free.

A MDF$(K, M)$ has $(KM + 1)M$ routers. $M$ and $(KM + 1)$ are relatively prime so an algorithm like this is not possible. A partially populated Dragonfly with $KM^2$ groups may be able to exploit the idea.

If $s = 1$ this theorem reduces to an algorithm originally occurring in [5]. The algorithm can be implemented on a fully populated Dragonfly provided the connectivity of groups in the Dragonfly is done properly [6].

## 4 D3$(2^k, 2^m)$ and The Swapped Boolean Hypercube

The Abelian groups used to enumerate routers and ports do not have to be cyclic groups. For example, if $M = 16$ the group can be $\oplus \mathbb{Z} \mod 2$ which is Boolean algebra on the 4 bit quantities. If $c$ and $p$ do not use the same kind of group, the condition that $M$ and $K$ have a common factor is replaced by the condition that the group used for $c$ and the group used for $p$ have subgroups of the same size.

The Swapped Boolean Hypercube, $SBH(k, m)$, is a graph with $2^{k+2m}$ nodes. It’s address space is a set of $k + 2m$ long bit strings that are partitioned into three fields $(c, d, p)$. The field $c$ is the high order $k$ bits, $p$ is the low order $m$ bits and $d$ is the middle $m$ bits. If $p$ and $p'$ differ by one bit $i$, $(c, d, p)$ is connected to $(c, d, p')$. This link is denoted $\pi_i$ and is called a local link. If $c$ and $c'$ differ by bit $i$, there is a link denoted $\gamma$ connecting $(c, d, p)$ to $(c', d, p')$. Note the swap of $p$ and $d$. There is one additional link which connects $(c, d, p)$ to $(c, p, d')$. It is denoted $Z$. If $p = d$ there is no link $Z$. All links are bidirectional. The nodes of $SBH(k, m)$ are of degree $k + m$.

The bit exchange between $(c, d, p)$ and another node depends upon the field the bit is in. The table gives the bit exchanges.

| field  | action |
|--------|--------|
| $c$    | $(c, d, p) \xrightarrow{\gamma} (c', d, p)$ |
| $d$    | $(c, d, p) \xrightarrow{Z} (c, p, d')$ |
| $p$    | $(c, d, p) \xrightarrow{\pi_i} (c, d, p')$ |

If $d = p$, $\gamma$ connects $(c, d, d)$ to $(c', d', d)$ and $Z \circ \pi_i$ connects $(c, d, d)$ to $(c, d', d)$. $SBH(k, m)$ is a dilation three emulation of the hypercube of dimension $k + m$. It’s diameter is $2k + m + 3m = 2k + 4m$. It’s nodes are degree $k + 2m$. The average dilation is less than two. A node attached to $(c, d, p)$ translates a program designed for the $(k + 2m)$ Boolean hypercube into a program on the Swapped Hypercube using the above paths.
If $D3(2^k, 2^m)$ is constructed using Boolean arithmetic, it obviously contains $SBH(k, m)$. $\gamma = c \oplus c'$, $\pi_t = p \oplus p'$ and $Z$ is the 0 global port. On $SBH(k, m)$, these paths are used only for pairs that differ by a single bit. Both directions can occur simultaneously. All three exchanges are vector paths so can occur simultaneously because of property 4 of Swapped Dragonflies. That is why $\pi_t$ is a $glg$ path instead of an $lgf$ path. Therefore, an ascend-descent algorithm can be performed at twice the cost of the algorithm on a $(k + 2m)$ Boolean hypercube because the average elevation of the emulation is two.

If $k$ is even, $2^k$ is a square so linear algebra can be performed efficiently. If $k$ is not even, $k - 1$ is and $D3(2^{k-1}, 2^m)$ can be found inside $D3(2^k, 2^m)$.

Johnsson and Ho [10] developed an all-to-all algorithm on a Boolean hypercube that takes network time $t_w n/2$. The algorithm is parallel over all hypercube links. The algorithm can be run on $SBH(k, m)$ or on $D3(2^k, 2^m)$. On both a hypercube link is one, two, or three network hops. The average is two. The algorithm takes $(2/3)(2^{k+2m}/2)t_w$ time on $SBH(k, m)$.

The object of the following discussion is to prove that the doubly-parallel all-to-all on $D3(2^k, 2^m)$ is faster than the Johnson and Ho all-to-all on $SBH(k, m)$. There is a constraint on the doubly-parallel all-to-all, $s \leq M/2$, because paths on $D3(2^k, 2^m)$ use two local hops. Therefore $s = \min(2^k, 2^{m-1})$ so the doubly-parallel algorithm takes

$$2^{k+2m}/\min(2^k, 2^{m-1}) = \max(2^m, 2^{k+m+1})t_w$$

time on $D3(2^k, 2^m)$. This is less than $(2^{k+2m}/3)t_w$ because $m$ and $k$ are not zero. Therefore, the doubly-parallel algorithm is superior.

5 The Broadcast Swapped Dragonfly

The $M$ routers on a drawer of $D3(K, M)$ contain $M$ depth four edge-disjoint spanning trees

$$(c, d, p) \xrightarrow{G} (c, d, p) \xrightarrow{L} (c, d, p) \xrightarrow{0} (c, d, p) \xrightarrow{L} (c, d, p).$$

Replacing $p$ by $p'$ leads to edge disjoint paths. This fact can be used to do multiple broadcasts from source $(c, d, q)$ by starting with $(c, d, q) \rightarrow (c, d, p)$ directing each $(c, d, p)$ to do a different broadcast. This requires five router hops for each broadcast vs three router hops for the depth three spanning tree

$$(c, d, p) \rightarrow (c, d, *) \rightarrow (c, *, d) \rightarrow (c, *, *)$$

at $(c, d, p)$.

Implementation of this idea requires that routers can be equipped with a program that does not depend upon their position in $[5, 1]$. The program proposed here depends upon packets having a synchronizing header. A Swapped Dragonfly using these headers is called a Broadcast Swapped Dragonfly. The header has four entries $[b; \gamma, \pi, \delta]; b$ is a counter, $\gamma$ is a global port and $\delta$ and $\pi$ are local ports. A router interprets the header in the following way.

- if $b$ is odd, use local port $\delta$ and change $b$ to $b - 1$, $\delta$ to $\pi$ and $\pi$ to 0,
- if $b$ is even, use global port $\gamma$ and change $b$ to $b - 1$ and $\gamma$ to 0.

A packet has arrived at an edge router when $b = 0$. This program is independent of where the packet is in $[5, 1]$. If routers cannot duplicate packets, the header becomes part of the packet. It is interpreted by a node attached to the router at each hop of the path. Here are evolutions of paths when $b = 3$ and 4;

\[
\begin{align*}
(c, d, p) & \rightarrow (c, d, p + \delta) & \rightarrow (c + \gamma, p + \delta, d) & \rightarrow (c + \gamma, p + \delta, d + \pi) \\
[3; \gamma, \pi, \delta] & \rightarrow [2; \gamma, 0, \pi] & \rightarrow [1; 0, 0, \pi] & \rightarrow [0; 0, 0, 0] \\
(c, d, p) & \rightarrow (c + \gamma, p, d) & \rightarrow (c + \gamma, p, d + \delta) & \rightarrow (c + \gamma, p + \delta, d + \pi) & \rightarrow (c + \gamma, d + \delta, p + \pi) \\
[4; \gamma, \pi, \delta] & \rightarrow [3; 0, \pi, \delta] & \rightarrow [2; 0, 0, \pi] & \rightarrow [1; 0, 0, \pi] & \rightarrow [0; 0, 0, 0] \\
\end{align*}
\]

The synchronized header is part of the packet header. It will be necessary for the packet header to have a broadcast bit to distinguish the packet from a point-to-points packet.

If routers can duplicate packets, $M$ broadcast take time $t_r + 5t_w$ where $t_r$ is the time required to delegate broadcasts from $(c, d, p)$ to its neighbors and $t_w$ is router latency. If routers cannot duplicate packets, the time for $M$ broadcasts is $5t_w$. Using the level three spanning tree at $(c, d, p)$, the time for $M$ broadcasts is proportional to $M$.

If a large number $X$ of broadcasts are needed the comparison is $5X/M$ to $3X$ network hops which is clearly a win for the depth-four trees. However, chaining may change the calculation. The following is an analysis of performing $X >> M$ broadcasts using pipe-lining of the level three algorithm and the level five algorithm.

The depth three tree pipe-line.

\[
\begin{align*}
(c, d, p) & \xrightarrow{l} (c, d, *) \xrightarrow{g} (c, d, *) \xrightarrow{g} (c, d, *) \\
(c, d, p) & \xrightarrow{l} (c, d, *) \xrightarrow{g} (c, d, *) \xrightarrow{g} (c, d, *) \\
(c, d, p) & \xrightarrow{l} (c, d, *) \xrightarrow{g} (c, d, *) \xrightarrow{g} (c, d, *) \\
(c, d, p) & \xrightarrow{l} (c, d, *) \xrightarrow{g} (c, d, *) \xrightarrow{g} (c, d, *) \\
\end{align*}
\]

\footnote{This idea originated with Johnsson and Ho [9].}
is free of conflict if \( p \neq d \) so the cost is \( X \) router hops.

Pipe-lining the \( M \) depth-four spanning tree is more problematic; the first local hop is the delegation step.

\[
(c, d, p) \xrightarrow{t} (c, d, q) \xrightarrow{t} (*, q, d) \xrightarrow{t} (*, q, *) \xrightarrow{0} (*, *, q) \xrightarrow{t} (*, *, *)
\]

\[
(c, d, p) \xrightarrow{t} (c, d, q) \xrightarrow{g} (*, q, d) \xrightarrow{t} (*, q, *) \xrightarrow{0} (*, *, q) \xrightarrow{t} (*, *, *)
\]

\[
d \neq p \\
(c, d, p) \xrightarrow{t} (c, d, q) \xrightarrow{g} (*, q, q) \xrightarrow{t} (*, *, *)
\]

So chaining in pairs gives the following:

\[
(c, d, p) \xrightarrow{t} (c, d, q) \xrightarrow{g} (*, q, d) \xrightarrow{t} (*, q, *) \xrightarrow{0} (*, *, q) \xrightarrow{t} (*, *, *)
\]

\[
(c, d, p) \xrightarrow{t} (c, d, q) \xrightarrow{g} (*, q, d) \xrightarrow{t} (*, q, *) \xrightarrow{0} (*, *, q) \xrightarrow{t} (*, *, *)
\]

delivers 2 broadcasts every 6 router hops for a cost of \( 3X/M \).

In broadcast mode the level three broadcast at \((c, d, p)\) has header \([3; *, *, *]\) and the level four broadcast has header \([4; *, *, *]\). The evolution of the headers in a broadcast is:

\[
(c, d, p) \rightarrow (c, d, *) \rightarrow (*, *, d) \rightarrow (*, *, *)
\]

\[
(c, d, p) \rightarrow (*, p, d) \rightarrow (*, p, ye) \rightarrow (*, *, p) \rightarrow (*, *, *)
\]

respectively. In the level four path, the global port 0 is used by \( KM \) routers. Note that the header \([2; 0, 0, *]\) compels a router to send point-to-point over global port 0 and \([1; 0, 0, *]\) compels a local broadcast.

The Broadcast Swapped Dragonfly \( D3(2^k, 2^m) \) enables the emulation of the \((k + 2m)\)-Boolean hypercube with uniform dilation four. The path \( c, d, \) and \( p \) are given by

\[
c \quad 4; \gamma, 0, 0 \\
d \quad 4; 0, 0, \delta \\
p \quad 4; 0, \pi, 0
\]

These are all four-path but they have the advantage that all paths of a given type can be followed concurrently, and also paths of different type can be followed concurrently without link conflict. The presence of a dilation four hypercube in \( D3(2^k, 2^m) \) means that algorithms designed for hypercubes may be compared with the algorithms designed here and the faster algorithm used. If \( K \) and \( M \) are not powers of 2, \( D3(K, M) \) contains an emulation of \( D3(2^k, 2^m) \) with \( k = \log K \) and \( m = \log M \).

6 Conclusion

It has been shown that there are three constraints on the parameters \( K \) and \( M \) that lead to useful algorithms:

1. On \( D3(K^2, M) \) an \( n \times n \) matrix product can be computed in network time \( 4(n^2/KM)T_w \).

2. On \( D3(ks, ms) \) an all-to-all exchange can be performed in network time \( (n^2/KM^2)S \).

3. \( D3(2^k, 2^m) \) contains a dilation three average two emulation of the \((k + 2m)\) Boolean hypercube.

4. Additionally, it has been shown that equipping \( D3(K, M) \) with a synchronizing counter enables \( n \) broadcasts in network time \( (3n/M)T_w \).

Result 2 is the only algorithm to do an all-to-all on \( P \) processors in less than \( P/2 \) network time. Result 3 implies that an ascendant-descendant algorithm can be done on \( D3(K, M) \) at twice the cost of doing the algorithm on a Boolean hypercube. The first three cases may apply to \( D3(J, N) \) with \( J \geq K \) and \( N \geq M \) because \( D3(J, N) \) contains an emulation of \( D3(K, M) \).

Source-vector routing is used to define the algorithms in 1, 2, and 4. It leads to algorithms devoid of interround conflicts. Source-vectors can be defined on a Maximal Dragonfly. However, a vector \((\gamma, \pi, \delta)\) generally leads to a link conflict at the third hop when it is used by two routers. This produces interround conflicts which lead to hotspots in an application. On a Dragonfly, the algorithms studied here would be used in a defectile routing environment. Vectors would be converted to destinations.

Note that \( D3(2^{2k}, 2^{2m}) \) is both a \( D3(k^2, M) \) and a \( D3(ks, km) \). It can do a \( n \times n \) matrix product in \( n/\sqrt{P} \) time, an all-to-all exchange in time \( n^2/\sqrt{P} \min(2^k, 2^m) \) and an ascendant-descendant algorithm at a factor two penalty over the cost on a hypercube of the same size. Clearly, Swapped Dragonflies of the type \( D3(2^k, 2^m) \) or of type \( D3(K, 2^m) \) with \( K \) only slightly larger than \( 2^k \) can be versatile networks. The emulation of a \((k + 2m)\) Boolean hypercube in \( D3(2^k, 2^m) \) means that algorithms designed for hypercubes may be compared with the algorithms designed here and the faster algorithm used.
References

[1] B. Alverson, E. Froese, L. Kaplan, and D. Roweth. *Cray XC© Series*. www.cray.com.

[2] A. Ariimilli, V. Chung, W. Densel, B. Durup, T. Hoefler, J. Jones, J. Lewis, J. Li, N. Ni, and R. Rajamoni. The PERCS High Performance Interconnect. *18th Symposium on High Performance Interconnects* DOI 10.1109/HOT1.10, 2010

[3] L.E. Cannon. A cellular computer to implement the Kalman Filter Algorithm, *Technical report, Ph.D. Thesis*, Montana State University, 14 July 1969

[4] E. Dekel. Nassimi, D., and Sahni, S. Parallel Matrix and graphs algorithms *SIAM J. Computing* 10:657-673. 1982

[5] R. Draper. The Swapped Dragonfly , ArXiv for Computer Science:2202.01843.

[6] R. Draper. The Fine Structure of Dragonflies, rndrape@gmail.com, rndrape@super.org

[7] H. Gupta, P. Sadayappan. Communication Efficient Matrix-Multiplication on Hypercubes, dbpubs.stanford.edu

[8] S. Hoory, N. Linial, A. Wigderson. Expander Graphs and Their Applications, Bulletin of the American Mathematical Society. 43 (4). 439-562. doi 10.1090/S0273-0979-06-01126-8, 2008

[9] C.T. Ho, S.L. Johnsson, and A. Edelman. Matrix multiplication on hypercubes using full bandwidth and constant storage, In *Proceedings of the Sixth Distributed Memory Computing Conference* (1991) 447-451.

[10] S.L. Johnsson and C.T. Ho. Optimum broadcasting and personalized communication in hypercubes, *IEEE Transactions on Computers*, 38(9) (1989) 1249-1268.

[11] J. Kim, W.J. Dally, S. Scott, and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. *International Symposium on Computer Architecture*. 77-88, 2008.