Efficient Fast Hartley Transform Algorithms for Hypercube-Connected Multicomputers *

Cevdet Aykanat, member, IEEE, and **Argun Derviş**
Department of Computer Engineering, Bilkent University
06533 Bilkent, Ankara, Turkey

Abstract

Although Fast Hartley Transform (FHT) provides efficient spectral analysis of real discrete signals, the literature that addresses the parallelization of FHT is extremely rare. FHT is a real transformation and does not necessitate any complex arithmetics. On the other hand, FHT algorithm has an irregular computational structure which makes efficient parallelization harder. In this paper, we propose an efficient restructuring for the sequential FHT algorithm which brings regularity and symmetry to the computational structure of the FHT. Then, we propose an efficient parallel FHT algorithm for medium-to-coarse grain hypercube multicomputers by introducing a dynamic mapping scheme for the restructured FHT. The proposed parallel algorithm achieves perfect load-balance, minimizes both the number and volume of concurrent communications, allows only nearest-neighbor communications and achieves in-place computation and communication. The proposed algorithm is implemented on a 32-node iPSC/2 [1] *hypercube multicomputer. High efficiency values are obtained even for small size* FHT *problems.*

Index Terms *: Digital Signal Processing, Fast Hartley Transform, parallel computing, multicomputer, hypercube, load balance, nearest-neighbor communication.*

1 Introduction

Digital Signal Processing (*DSP*) of real-time signals has gained importance with recent advances in digital computer technology. Digital signal processors, digital computers specializing in signal processing, are in development and available on the market. All of this growth is for massive amounts of computations in various *DSP* applications. One way to satisfy the performance requirement of *DSP* applications is to choose clever algorithms or expand the processor performance or both of them. *DSP* applications are characterized by computations that are massive but fairly straightforward and simple. Furthermore, these computations exhibit orderly structures. Besides, *DSP* algorithms are very efficient. These algorithms are optimized and improved several times until now. However, it is still not enough for most of the *DSP* applications. Performance of conventional computers are still very limited in cases where extensive number crunching computations are required. *Discrete Fourier Transform* (*DFT*) and *Discrete Hartley Transform* (*DHT*) are such examples.

The *DFT* of an input sequence $\{f(i)\colon i=0,1,\ldots,N-1\}$ of length N is

$$F(k) = \sum_{i=0}^{N-1} (f(i)[Cos(2\pi ki/N) - jSin(2\pi ki/N)]) \tag{1}$$

*This work is partially supported by Intel Supercomputer Systems Division under Grant SSD100791-2 and Turkish Scientific and Technical Research Council under Grant EEEAG-5

[1] iPSC/2 is a registered trademark of Intel Corporation

for $k = 0, 1, \ldots, N-1$. *DFT* provides a method for spectral analysis of discrete signals. Thus, Cooley and Tukey providing a more efficient algorithm [3], named as *Fast Fourier Transform* (*FFT*), made possible many applications concerning the computation of *DFT* to be realizable because of performance problems.

Beyond the highly accepted usage of *FFT*, it is a complex transformation. That is, both *DFT* and *FFT* include complex arithmetic even if the input signal consists of real numbers only. Hence, *FFT* contains redundancy if the signals in the time domain are real. *DHT* is developed for a more efficient and faster transformation [4]. The *DHT* of an input sequence $\{h(i)\colon\ i = 0, 1, \ldots, N-1\}$ of length N is

$$H(k) = \sum_{i=0}^{N-1} (h(i)[Cos(2\pi ki/N) + Sin(2\pi ki/N)]) \tag{2}$$

for $k = 0, 1, \ldots, N-1$, where the input sequence $h()$ is constrained to real numbers only. Hartley transform does not necessitate any complex arithmetics. This important feature of Hartley transform increases the performance of *DHT* by a factor of two, while decreasing the memory requirements again by a factor of two at the same time. Computational complexities of both schemes are $O(N^2)$. *FFT* reduces this time to $O(N \lg_2 N)$ [3]. As well as *FFT*, *DHT* has also a fast formulation called *Fast Hartley Transform* (*FHT*) [1, 2] with computational complexity $O(N \lg_2 N)$. *FHT* provides efficient spectral analysis of real discrete signals.

The purpose of this paper is to investigate the efficient parallelization of one-dimensional *FHT* algorithms on medium-to-coarse grain multicomputers implementing the hypercube interconnection topology. Computational load balance and communication overhead are two crucial factors that determine the efficiency of a parallel algorithm. In a multicomputer with high communication latency (start-up time), both the number and the volume of communications should be minimized in order to reduce the communication overhead. The communication structure of the parallel algorithm is also a crucial issue. In a multicomputer, each adjacent pair of processors can concurrently communicate with each other over the communication links connecting them. Such communications are referred as *single-hop* communications. However, non-adjacent processors can communicate with each other by means of software or hardware routing. Such communications are referred as *multi-hop* communications. Multi-hop communications are usually routed in a static manner over the shortest paths of links between the communicating pairs of processors. In software routing, the cost of multi-hop communications is substantially greater than that of the single-hop messages since all intermediate processors on the path are intercepted during the communication. The performance difference between an individual multi-hop and single-hop communication is relatively small in hardware routing. However, a number of concurrent multi-hop communications may congest the routing network thus resulting in substantial performance degradation. Hence, achieving concurrent communications between adjacent pairs of processors is a valuable asset in designing efficient parallel algorithms. Moreover, in almost all commercially available multicomputer architectures, interprocessor communications can only be initiated from/into contiguous local memory locations. Hence, communications from/into scattered memory locations may introduce considerable overhead to the parallel program. In this work, all these points are considered in designing an efficient parallel *FHT* algorithm for hypercube-connected multicomputers.

Although there is a substantial amount of literature on the parallelization of the *FFT*, the literature that addresses the parallelization of *FHT* is extremely rare. This situation can be attributed to the following reasons: **(i)** wide popularity of the *FFT* algorithm in the computer science literature, **(ii)** irregular computational structure of *FHT* compared to the symmetrical and regular computational structure of the *FFT*, **(iii)** feasibility of indirect computation of *FHT* through *FFT*. However, direct computation of *FHT* is much more efficient compared to any indirect computation of *FHT*.

To our knowledge, only Hou [6] and Lin [8] investigated the parallelization of *FHT* on hypercubes. Hou's algorithm is a fine-grain algorithm which considers the parallelization of N-point *FHT* on a hypercube with $P = N$ processors, where each processor is assigned a single *FHT* point. In this work, we briefly describe an extension of Hou's fine-grain algorithm to medium-to-coarse grain parallelism, where $N \geq 4P$. This algorithm uses only single-hop communications. The number and volume of concurrent communications required by this scheme are $3d-3$ and $\approx (3d-3)M$ *FHT* points, respectively, where $M = N/P$. The dynamic mapping scheme proposed by Lin [8] reduces the number of concurrent communications to d, each with a volume of N/P *FHT* points. Concurrent communication volume overhead of Lin's algorithm is $Md - M/2$ *FHT* points on the *Hartley graph*. However, in a hypercube implementation of Lin's algorithm, $d-2$ concurrent exchange communication steps involve multi-hop communications since Hartley graph can not be embedded with dilation-one onto the hypercube graph. Hence, concurrent communication volume overhead of Lin's algorithm will be much higher on the hypercube topology due to the congestion during these d–2 concurrent exchange communications. Although these two algorithms are successful attempts to reduce the communication overhead, neither of them achieves perfect load balance for the simplified butterfly scheme. Lin's algorithm, which is originally proposed for the basic butterfly scheme, achieves perfect load balance only for this scheme. However, basic butterfly scheme requires $\approx 60\%$ more floating point operations than the simplified butterfly scheme.

In this work, we propose an efficient restructuring for the sequential *FHT* algorithm which brings regularity and symmetry to the computational structure of the *FHT*. The restructured algorithm does not involve any computational overhead compared to the original algorithm. Then, we propose an efficient parallel *FHT* algorithm for medium-to-coarse grain hypercube multicomputers by introducing a dynamic mapping scheme for the restructured *FHT*. The proposed parallel algorithm has the following nice features for the implementation of an N-point *FHT* on a d-dimensional hypercube with $P = 2^d \leq N/4$ processors: **(i)** achieves perfect load-balance for the simplified butterfly scheme, **(ii)** allows only nearest-neighbor communications, **(iii)** minimizes the number of concurrent communications to d by eliminating fragmentary message passing, **(iv)** minimizes the total concurrent communication volume to $dM/2$ by minimizing the volume of communication in each concurrent exchange step to $M/2 = N/2P$ *FHT* points, **(v)** achieves in-place computation and communication.

The sequential *FHT* is presented in Section 2. In Section 3, parallelization of the presented *FHT* scheme is discussed. Section 3.1 presents the proposed restructuring of the *FHT* algorithm for an efficient parallelization. The dynamic mapping scheme proposed for the restructured *FHT* algorithm is presented in Section 3.2. Section 4 presents the experimental results on Intel's iPSC/2 hypercube multicomputer.

Figure 1: Computational steps in ($N = 32$)-point Fast Hartley Transform.

2 Sequential *FHT* Algorithm

Different strategies exist for the computation of *FHT* and some include Radix-2 Decimation-in-Time *FHT*, Radix-2 Decimation-in-Frequency *FHT*, Radix-4 *FHT*, Split Radix *FHT*, Recursive *FHT* and Vector *FHT* [5, 7, 9, 10]. Computational steps for a 32-point, radix-2, decimation-in-time *FHT* algorithm [7] is illustrated in Fig. 1. This tabular representation is proposed in [10]. The input in this scheme is N real numbers in *bit-reversed* order. The output is N real numbers in *normal* order. The Ci and Si factors in Fig. 1 represent $Cos(2\pi i/N)$ and $Sin(2\pi i/N)$, respectively. As is seen in Fig. 1, each level of *FHT* algorithm takes a set of N real numbers and transforms them into another set of N real numbers. This process is repeated $n = \lg_2 N$ times, resulting in the *in-place* computation of the desired Hartley transform in normal order. However, the tabular representation is not sufficient for a detailed analysis of the computational interdependencies which is crucial for an efficient parallel algorithm design. In this work, computational flow graph for the *FHT* algorithm is derived in order to explore the computational interdependencies.

A close examination of Fig. 1, reveals that *FHT* computations at each level resemble basic *FFT* butterfly computations. The first level ($\ell = 0$) consists of 2-point *FFT* like butterflies. However, the remaining levels ($\ell = 1, 2, \ldots, n-1$) consist of 4-point *FHT* butterflies. There are two types of basic *FHT* butterflies which will be referred here as *type-1* and *type-2* basic *FHT* butterflies. Fig. 2 illustrates the computational flow graphs for type-1 and type-2 basic *FHT* butterflies at level ℓ in an N-point *FHT*. Each type of *FHT* butterfly is identified by an ordered 4-tuple $\{p, r, q, s\}$. Note that both types of basic butterflies consist of two stages.

In the first stage of a type-1 basic *FHT* butterfly, the (q,s) pair is involved in two butterfly type of computations to generate 4 intermediate results. Each butterfly computation involves the multiplication of q and s points by Cos/Sin and Sin/Cos factor pairs, respectively, and pairwise addition of these 4

Figure 2: Computational flow graphs for (a) type-1 and (b) type-2 basic *FHT* butterflies.

multiplication results. Hence, the first stage of a type-1 basic butterfly involves 8 multiplications and 4 additions. In the first stage of a type-2 basic butterfly, both q and s points are multiplied by $Cos{+}Sin$ (CS) factor pairs to generate 4 intermediate results. Hence, the first stage of a type-2 basic butterfly involves 4 multiplications. In the second stages of both type-1 and type-2 basic butterflies, these 4 intermediate results are individually added to their p, r, q, s points to update these values for the next level. The second stages of both types of basic *FHT* butterflies involve 4 additions.

A careful analysis of type-1 basic butterfly computation reveals that angles of Cos and Sin factor pairs multiplied by the q and s points are mutually π radians away from each other, since $2\pi(i + \frac{N}{2})/N$ = $2\pi i/N + \pi$. Hence, type-1 basic *FHT* butterfly (Fig. 2(a)) can be simplified as shown in Fig. 3(a). This simplification reduces the total number of floating point operations in the first stages of type-1 butterflies to 6 (from 8 multiplications and 4 additions to 4 multiplications and 2 additions) as follows:

$$qtemp := Ci \times H[q] + Si \times H[s]; \tag{3a}$$
$$stemp := Cj \times H[s] + Sj \times H[q]; \tag{3b}$$
$$H[q] := H[p] - qtemp; \tag{3c}$$
$$H[s] := H[r] - stemp; \tag{3d}$$
$$H[p] := H[p] + qtemp; \tag{3e}$$
$$H[r] := H[r] + stemp; \tag{3f}$$

The resulting *FHT* butterfly will be referred here as type-1 *simplified FHT* butterfly. A similar analysis can also be applied to type-2 basic *FHT* butterfly to reduce the number of multiplications involved in the first stage from 4 to 2. Furthermore, a detailed analysis shows that $Cos{+}Sin$ factors multiplied by the q and s points are always 1. Hence, the remaining two multiplications can also be omitted. Fig. 3(b) illustrates the computational flow-graph for a type-2 *simplified FHT* butterfly. Note that multiplications with $Cos{+}Sin$ factors are shown in Fig. 3(b) for the sake of completeness. Hence, the computations involved in a type-2 simplified *FHT* butterfly are as follows:

$$qtemp := H[q]; \tag{4a}$$
$$stemp := H[s]; \tag{4b}$$
$$H[q] := H[p] - qtemp; \tag{4c}$$
$$H[s] := H[r] - stemp; \tag{4d}$$
$$H[p] := H[p] + qtemp; \tag{4e}$$
$$H[r] := H[r] + stemp; \tag{4f}$$

Figure 3: Computational flow graphs for (a) type-1 and (b) type-2 simplified *FHT* butterflies.

In the rest of the paper, simplified *FHT* butterflies will be referred as butterflies for the sake of simplicity, unless otherwise stated.

Each *FHT* point in an N-point *FHT* is assumed to have an n-bit binary representation where $n = \lg_2 N$. For example, f_n (binary string of length n) denotes the binary representation of an *FHT* point q, where q denotes its decimal index in the bit-reversed ordering. In both types of butterflies, *FHT* points in both (p,q) and (r, s) pairs differ only in the ℓth bit of their n-bit binary representation at level ℓ such that $q = p + 2^\ell$ and $s = r + 2^\ell$. That is, ℓth bits of the binary representations of both q and s indices are “1”, whereas ℓth bits of both p and r indices are “0”. Note that the least significant bit of a binary number is referred here as its $0th$ bit. Hence, *FHT* points in (p, q) and (r, s) pairs are separated by 2^ℓ at level ℓ.

In a type-1 butterfly at level ℓ, two *FHT* points of each (q, s) pair differ only in the least significant ℓ bits of their n-bit binary representations. This difference is such that, least significant ℓ bits of the binary representations of the q and s indices are mutually 2’s complement of each other. Hence, the separation between q and s indices of a type-1 butterfly varies between 2 and $2^\ell - 2$ at level ℓ for $\ell \geq 2$. In a type-2 butterfly at level ℓ, q and s points only differ in the $(\ell - 1)^{th}$ bit of their binary representations such that q is a power of 2^ℓ, and $s = q + 2^{\ell-1}$. That is, q and s indices of a type-2 butterfly are separated by $2^{\ell-1}$ at level ℓ. Hence, type-2 butterflies at level ℓ can easily be identified by the 4-tuples $\{p, r, q, s\} = \{p, p{+}2^{\ell-1}, p{+}2^\ell, p{+}3 \times 2^{\ell-1}\}$ where p is a multiple of $2^{\ell+1}$ (i.e., least significant $(\ell + 1)$-bits are all 0’s). These observations can be summarized by the following definitions.

Definition 1: For any binary strings b_k and $f_{\ell-1} \neq \phi_{\ell-1}$ (where $k = n - \ell - 1$), the 4-tuple

$$\left\{ b_k 00 f_{\ell-1}, b_k 01 f^c_{\ell-1}, b_k 10 f_{\ell-1}, b_k 11 f^c_{\ell-1} \right\}$$

constitutes a $type$-1 *FHT* butterfly at level ℓ ($\ell = 2, \ldots, n-1$) in an (N=2^n)-point *FHT*. Here, subscripts denote the lengths of the respective binary strings, $\phi_{\ell-1}$ denotes a string consisting of $\ell - 1$ zeros, and $f^c_{\ell-1}$ denotes $(\ell - 1)$-bit 2’s complement of $f_{\ell-1}$. Note that $(1 f^c_{\ell-1})$ and $(0 f_{\ell-1})$ are ℓ-bit 2’s complement of each other since $f_{\ell-1}$ contains at least one 1.

Figure 4: Computational flow graph for the 32-point *FHT* and its static tiled mapping on a 3-dimensional hypercube.

Definition 2: For any binary string b_k (where $k = n - \ell - 1$), the 4-tuple

$$\left\{b_k 00 \phi_{\ell-1}, b_k 01 \phi_{\ell-1}, b_k 10 \phi_{\ell-1}, b_k 11 \phi_{\ell-1}\right\}$$

constitutes a type-2 *FHT* butterfly at level ℓ ($\ell = 1, 2, \ldots, n-1$) in an (N=2^n)-point *FHT*.

Fig. 4 illustrates the proposed computational flow-graph for the (N=32)-point *FHT* algorithm using the simplified butterfly scheme. As is seen in Fig. 4, first level ($\ell = 0$) is a special level which consists of two-point butterflies without any Cos/Sin factor multiplications. That is, only addition/subtraction operations are performed in two-point butterflies. Each level ℓ of the following $n-1$ levels consist of $N/2^{\ell+1}$ consecutive blocks where each block contains $2^{\ell+1}$ consecutive *FHT* points. For example, at level $\ell = 3$, a 32-point *FHT* contains $32/2^{3+1} = 2$ blocks, $B_3^0 = \{0-15\}$ and $B_3^1 = \{16-31\}$, where each block consists of $2^{3+1} = 16$ consecutive *FHT* points. First, second, third and fourth quarters of each block contain $2^{\ell-1}$ p, r, q and s points of the $2^{\ell-1}$ butterflies confined to that block. The first points of successive quarters of each block constitute the p, r, q, s points of the only $type$-2 butterfly involved in that block. As is seen in Fig 4, $\{16,20,24,28\}$ is the only type-2 butterfly involved in block $B_3^1 = \{16-31\}$, whereas $\{17,23,25,31\}$, $\{18,22,26,30\}$ and $\{19,21,27,29\}$ constitute the type-1 butterflies in that block. Hence, the number of type-1 and type-2 butterflies at level ℓ are

$$N_{T1}^{\ell} = N/4 - N/2^{\ell+1} \tag{5a}$$
$$N_{T2}^{\ell} = N/2^{\ell+1} \tag{5b}$$

respectively. Note that $N_{T1}^{\ell} + N_{T2}^{\ell} = N/4$ *FHT* butterflies exist at each level for $\ell = 1, 2, \ldots, n-1$. Also note that level $\ell = 1$ consists of only $N/4$ type-2 butterflies and the number of type-2 butterflies decreases by one half in the following $n-2$ levels and reduces to 1 at the last level ($\ell = n-1$).

Fig. 5 illustrates the pseudo-code for the sequential *FHT* algorithm. In this algorithm, N real inputs $\{h(i)$: $i = 0, 1, \ldots, N-1\}$ are assumed to be stored in *bit-reversed* order in one-dimensional H-array. Computations are performed in-place and the results are obtained in the H-array in *normal* order. As is seen in Fig. 5, the first outer *for-loop* performs the computations associated with the 2-point butterflies in the first level ($\ell = 0$). The second outer *for-loop* performs the computations associated with the remaining $n-1$ levels. The first inner *for-loop* iterates $N/2^{\ell+1}$ times to identify the $N/2^{\ell+1}$ consecutive *FHT* blocks at each level. The innermost *for-loop* iterates $2^{\ell-1} - 1$ times to identify and perform the computations involved in the $2^{\ell-1} - 1$ type-1 butterflies in each block. In Fig. 5, $p1, r1, q1, s1$ and $p2, r2, q2, s2$ refer to the p, r, q, s points of type-1 and type-2 butterflies, respectively. The total number of type-1 and type-2 FHT butterflies are

$$\sum_{\ell=2}^{n-1} N_{T1}^{\ell} = \frac{N}{4} \lg_2 N - \frac{3N}{4} + 1 \tag{6a}$$
$$\sum_{\ell=1}^{n-1} N_{T2}^{\ell} = \frac{N}{2} - 1 \tag{6b}$$

respectively. Recall that type-1 and type-2 simplified butterflies require 10 and 4 floating-point operations, respectively, and that first level ($\ell = 0$) involves only N floating point addition/subtraction operations. Hence, the sequential execution time of an N-point *FHT* computation can be modeled as

$$T_{seq} = \Big(2.5N \lg_2 N - 4.5N + 6\Big)\, t_{calc} \tag{7}$$

where t_{calc} is the time taken by the floating-point multiplication, addition and subtraction operations. The computation of Cos/Sin factors are not involved in the given complexity analysis.

In most of the real time *DSP* applications, N-point *FHT* is applied consecutively, for a fixed N, to N-point input data sets. Hence, in general, $N/2$ coefficient values are computed once, as the value of the N is fixed, and stored in a table. These coefficients are then accessed by a simple table-lookup procedure during successive *FHT* computations.

```
/* Input in bit-reversed order in H[0 ... N-1]   */
/* Output in normal order in H[0 ... N-1]     */

for i := 0 to N/2−1 do
    temp := H[2i + 1];
    H[2i + 1] := H[2i] − temp;
    H[2i] := H[2i] + temp;

for ℓ := 1 to n − 1 do
    for i := 0 to N/2^(ℓ+1) − 1 do
        p2 := i × 2^(ℓ+1);      q2 := p2 + 2^ℓ;
        r2 := p2 + 2^(ℓ−1);    s2 := q2 + 2^(ℓ−1);
        qtemp := H[q2];   stemp := H[s2];
        H[q2] := H[p2] − qtemp;
        H[s2] := H[r2] − stemp;
        H[p2] := H[p2] + qtemp;
        H[r2] := H[r2] + stemp;

        for j := 1 to 2^(ℓ−1) − 1 do
            p1 := p2 + j;         q1 := p1 + 2^ℓ;
            r1 := p2 + 2^ℓ − j;  s1 := r1 + 2^ℓ;
            qtemp := Cfac1 × H[q1] + Sfac1 × H[s1];
            stemp := Cfac2 × H[s1] + Sfac2 × H[q1];
            H[q1] := H[p1] − qtemp;
            H[s1] := H[r1] − stemp;
            H[p1] := H[p1] + qtemp;
            H[r1] := H[r1] + stemp;
```

Figure 5: Sequential ($N = 2^n$)-point *FHT* algorithm

3 Parallel FHT Algorithm

There are strong computational dependencies in the *FHT* algorithm. These computational dependencies exist between successive levels confined within the butterflies. As is seen in Fig. 3(a) and Fig. 4, stage-2 computations in type-1 butterflies depend on the results of the stage 1 computations. The computation of *qtemp* and *stemp* values ((3a) and (3b), respectively) in the first stage necessitates bidirectional interdependency between q and s points, which will be referred here as $q{\leftrightarrow}s$ *interactions*. Note that first stages of type-2 butterflies involve no computations and interactions. Type-2 butterflies are also modeled as two stage computations just for the sake of completeness. The update of p, r, q and s points in the second stages of all butterflies (for $\ell = 1, 2, \ldots, n-1$) necessitate bidirectional interdependencies between the p and q, and r and s points, which will be referred here as $p{\leftrightarrow}q$ and $r{\leftrightarrow}s$ interactions. The $p{\leftrightarrow}q$ and $r{\leftrightarrow}s$ interactions are very regular in nature since p and q, and r and s points are separated by 2^ℓ at level ℓ for $\ell \geq 1$. In fact, this regularity in the $p{\leftrightarrow}q$ and $r{\leftrightarrow}s$ interactions makes hypercube topology very suitable for the parallelization of *FHT*. However, the $q{\leftrightarrow}s$ interactions complicates the parallelization because of the irregular spacing between q and s points of type-1 butterflies.

This paper investigates the parallelization of ($N = 2^n$)-point *FHT* on a d-dimensional hypercube with $P = 2^n$ processors, where the number of 4-point *FHT* butterflies is an integer (power of 2) multiple of the number of processors (i.e., $N \geq 4P$). A straightforward parallelization can be achieved by adopting a static *tiled* mapping. The first processor in the decimal ordering is assigned the first $M = N/P$ *FHT* points, the second processor is assigned the next M points and so on. Successive processors in the decimal ordering are assigned the consecutive slices of *FHT* points with each slice containing equal number of M consecutive *FHT* points. This mapping prevents the fragmentation of *FHT* butterflies and (q,s) pairs during the first $n{-}d$ and $n{-}d{+}1$ levels, respectively. Both (p,q) and (r,s) pairs of butterflies are fragmented across processor pairs which are neighbors over channel $c = \ell - n + d$ at level ℓ for $\ell = n-d, \ldots, n-1$. Here, channel c denotes the set of $P/2$ communication links between processor pairs whose d-bit binary representations differ only in their cth bit. Hence, these pairwise exchanges due to the $p{\leftrightarrow}q$ and $r{\leftrightarrow}s$ interactions can be accomplished by performing a concurrent single-hop exchange communication over channel $c = \ell - n + d$ at level ℓ for $\ell = n-d, \ldots, n-1$. Unfortunately, the nature of fragmentation of (q,s) pairs, and hence the nature of the communications due to the $q{\leftrightarrow}s$ interactions are very irregular and complicated because of the irregularity in these interactions. A careful analysis reveals that the $q{\leftrightarrow}s$ interactions necessitate concurrent exchange communications, each with a volume of $M-1$ *FHT* points, at each level of the last $d-1$ levels, plus concurrent exchange communications, each with a volume of single *FHT* point, at each level of the last d–2 levels. All former type of exchange communications are single-hop communications at level $\ell = n-d+1$ and multi-hop communications with distances $2, \ldots, d-1$ during the last $d-2$ levels $\ell = n-d+2, \ldots, n-1$ respectively. All latter type of communications are single-hop communications at level $\ell = n-d+2$ and mostly multi-hop communications with maximum distances $2, \ldots, d-2$ during the last $d-3$ levels $\ell = n-d+3, \ldots, n-1$ respectively. Multi-hop exchange communications during the last d–2 levels will introduce drastic performance degradation due to the congestion.

The fine-grain algorithm proposed by Hou [6] considers the parallelization of N-point *FHT* on a hypercube with $P = N$ processors, where each processor is assigned a single *FHT* point. Here, we will briefly describe an extension of Hou's fine-grain algorithm to medium-to-coarse grain parallelism. A *tiled* decomposition scheme is adopted for the initial mapping. This initial mapping is maintained during the first $n-d+2$ levels $\ell = 0, 1, \ldots, n-d+1$. The tiled mapping scheme already confines the *FHT* butterflies to 1-dimensional and 2-dimensional subcubes over channels $c = 0$ and $c = 0, 1$ at levels $\ell = n-d$ and $\ell = n-d+1$, respectively. Hence, the second stages of levels $\ell = n-d$ and $\ell = n-d+1$, and the first stage of level $\ell = n-d+1$ necessitate concurrent single-hop exchange communications over channels $c = 0, 1$ and $c = 0$ due to the $p{\leftrightarrow}q$, $r{\leftrightarrow}s$ and $q{\leftrightarrow}s$ interactions, respectively. Then, at the end of each level $\ell = n-d+1, \ldots, n-2$, those processor pairs which exchanged their local $M-1$ or M q or s points during the first stage of that level, exchange the further responsibilities of these local *FHT* points. These mapping exchange operations performed at the end of each level ℓ, for $\ell = n-d+1, \ldots, n-2$, confine the *FHT* butterflies to 2-dimensional subcubes over successive channels $\ell - n + d$ and $\ell - n + d + 1$, at the following level $\ell + 1$. The d-bit binary representations of 4 processors in each subcube differ only in their cth and $(c{-}1)th$ bits such that these two successive bits are "00", "01", "10" and "11" in the first, second, third and fourth processors, respectively. The fragmentation of *FHT* butterflies across these subcubes during the last d–1 levels is such that first, second, third and fourth processors in each subcube hold M p, r, q and s points, respectively, of the M butterflies confined to that subcube. Hence, each level ℓ of the last d–1 levels require 3 concurrent single-hop exchange communications, each with a volume of M (or $M-1$) *FHT* points, over channels $c-1$, c and $c-1$, respectively, where $c=\ell-n+d$. The first and second exchange communications are information exchange operations due to the $q\leftrightarrow s$, and $p\leftrightarrow q$, $r\leftrightarrow s$ interactions in the first and second stage computations, respectively. The third exchange communication is a mapping exchange operation due to the nonlocal $q\leftrightarrow s$ swaps. Note that level $\ell=n-d$ necessitates only one concurrent single-hop exchange communication over channel $c=0$, and the mapping exchange communication at the last level may not be necessary. Thus, the number and volume of concurrent communications required by this scheme are $3d-3$ and $\approx (3d-3)M$ *FHT* points, respectively.

The dynamic mapping scheme proposed by Lin [8] reduces the number of concurrent communications to d. The initial mapping avoids the fragmentation of two-point butterflies at level $\ell = 0$ by assigning consecutive *FHT*-point pairs to successive processors in a cyclic manner. This initial mapping scheme can be considered as a *scattered* mapping of consecutive *FHT*-point pairs, where *FHT* point pair ($2i$, $2i+1$) is assigned to processor i mod P. The dynamic mapping during the following d levels confines the *FHT* butterflies to processor pairs which are neighbors on the *Hartley graph* during levels $\ell = 1, 2, \ldots, d$, and prevents the fragmentation of butterflies during the last $n-d-1$ levels. At level $\ell=1, 2, \ldots, d$, processor pairs whose least significant $\ell-1$ bits are all $0's$ hold $M/2$ type-2 butterflies, whereas all other processor pairs hold $M/2$ type-1 butterflies. Former and latter types of processor pairs will be referred here as type-2 and type-1 processor pairs, respectively. The fragmentation of level-ℓ butterflies (for $\ell=1, 2, \ldots, d$) across each processor pair is such that ith local *FHT*-point pairs in the first and second processors, whose $(\ell-1)th$ bits are 0 and 1, correspond to the (p,r) and (s,q) pairs of the butterflies, respectively, confined to that processor pair, for $i=0, 1, \ldots, M/2-1$. The first and second processors of type-1 pairs are responsible for updating the (p,s) and (r,q) pairs, respectively, or vice-versa, depending on their ℓth bits. The first and second processors of type-2 pairs are responsible for updating the (p,q) and (r,s) pairs, respectively. Hence, type-1 processor pairs need to exchange all of their local *FHT* points at the beginning of each level $\ell = 2, \ldots, d$. However, type-2 processor pairs need to exchange only half of their local *FHT* points at the beginning of each level $\ell = 1, 2, \ldots, d$. These exchanges will be referred here as type-1 and type-2 exchanges, respectively. One half of the M local *FHT* points involved in each type-1 exchange is a mapping exchange, whereas the other half is exchanged because of the computational interdependencies. Type-2 exchanges are both mapping and information exchanges.

All $P/2$ processor pairs are type-2 pairs at level $\ell = 1$, and the number of type-2 processor pairs decreases by one half in the following $d-1$ levels, thus reducing to 1 at level $\ell = d$. Thus, the communication volume of type-1 exchanges determines the concurrent communication volume during levels $\ell=2, 3, \ldots, d$. Hence, concurrent communication volume overhead of Lin's algorithm is $Md-M/2$ *FHT* points on Hartley graph. Unfortunately, Hartley graph cannot be embedded with dilation one onto the hypercube graph as is also indicated in [8]. In a hypercube implementation of Lin's algorithm, type-2 exchanges are single-hop communications over channel $c = \ell-1$ at level ℓ for $\ell = 1, 2, \ldots, d$. Type-1 exchanges at level $\ell=2$ are single-hop communications over channel $c=1$. Hence, all exchanges can be concurrently performed over channels $c = 0$ and $c = 1$ at levels $\ell = 1$ and $\ell = 2$, respectively. However, type-1 exchanges at levels $\ell=3, \ldots, d$ are mostly multi-hop communications with maximum distances of

$\ell - 1 = 2, \ldots, d-1$. Hence, concurrent communication volume overhead of Lin's algorithm will be much higher on the hypercube topology due to the congestion during these $d-2$ levels.

Although these two algorithms are successful attempts to reduce the communication overhead, neither of them achieves perfect load balance for the simplified butterfly scheme. Consider the coarse-grain extension of Hou's algorithm. The tiled mapping scheme, which is maintained during the first $n-d+2$ levels, achieves perfect load balance during the first $n-d$ levels, since it assigns equal number of unfragmented butterflies to each processor during these levels. However, load balance is disturbed during the first stage computations of the last d levels. At levels $\ell = n{-}d$ and $\ell = n{-}d{+}1, \ldots, n{-}1$, processors can be considered as divided into 2 and 4 groups, each containing $P/2$ and $P/4$ processors, respectively. At level $\ell = n-d$, each processor in the first and second halves of the hypercube holds and updates $M/2-1$ (p,r) and (q,s) pairs of type-1 butterflies, respectively. Hence, at level $\ell = n-d$, one half of the processors holding q and s points concurrently perform $3M-6$ floating point operations while the processors in the other half wait idle for receiving these *qtemp* and *stemp* results corresponding to the first stage computations of type-1 butterflies. At levels $\ell = n{-}d{+}1, \ldots, n{-}1$, each processor in the first, second, third and fourth quarters of the hypercube holds and updates either $M-1$ or M p, r, q and s points of type-1 butterflies, respectively. Hence, at levels $\ell = n{-}d{+}1, \ldots, n{-}1$, one half of the processors holding q or s points concurrently perform $3M$ or $3M-3$ floating point operations while the processors in the other half wait idle for receiving these *qtemp* or *stemp* results corresponding to the first stage computations of type-1 butterflies. Note that this algorithm cannot achieve perfect load balance even for the basic butterfly scheme during the first stage computations of last $d-1$ levels because of the 4-way computational fragmentation of *FHT* butterflies during these levels. Here, 4-way computational fragmentation refers to the situation in which 4 different processors compute the 4 different points of the same *FHT* butterfly.

Lin's algorithm, which is originally proposed for the basic butterfly scheme, achieves perfect load balance only for this scheme. This algorithm achieves perfect load balance during levels $\ell = 0$ and $\ell = d+1, \ldots, n-1$, both for the basic and simplified butterfly schemes, by assigning equal number of unfragmented butterflies to each processor during these $n{-}d$ levels. The 2-way fragmentation during levels $\ell = 1, 2, \ldots, d$ achieves perfect load balance for the basic butterfly scheme during these d levels. Consider the performance of this algorithm for the simplified butterfly scheme during $d-1$ levels $\ell = 2, \ldots, d$. After the exchange operations during these levels, $M/2$ type-1 butterflies are duplicated in each type-1 processor pair. However, each processor in type-1 pairs is responsible for updating either the (p,s) or (r,q) pairs of the respective $M/2$ butterflies. Hence, both processors in each type-1 pair should compute the same *qtemp* and *stemp* values for all $M/2$ butterflies local to that processor pair, because these two values are needed in the second stage computations of both (p,s) and (r,q) pairs. This redundancy during the first stage computations of type-1 butterflies reduces the performance of the algorithm to that of the basic butterfly scheme. This redundancy can be avoided if the first and second processors in each type-1 pair compute the *qtemp* and *stemp* values, or vice-versa, and then exchange these results. This approach attains the performance of the simplified butterfly scheme with perfect load balance at the expense of $d-1$ extra single-hop exchange communications each with a volume of $M/2$ *FHT* points.

Figure 6: Computational mappings of *FHT* butterflies to processors (a) coarse-grain extension of Hou's algorithm ($\ell = n-d+1, \ldots, n-1$), (b) Lin's algorithm ($\ell = 1, 2, \ldots, d$), (c) proposed algorithm ($\ell = n-d, \ldots, n-1$), during the respective levels which involve communications.

Figures 6(a) and 6(b) clearly illustrate the 4-way and 2-way computational fragmentation of 4-point *FHT* butterflies in coarse-grain extension of Hou's algorithm and Lin's algorithm, respectively, during the indicated levels which involve communications. Note that the 2-way fragmentation at level $\ell = n-d$ of the coarse-grain extension of Hou's algorithm is not shown in the figure since it is an exceptional level of this algorithm. In this figure, p_i, r_i, q_i and s_i represent the p, r, q and s points of the same butterfly, respectively. Circles represent processors and solid lines indicate the adjacency of the respective processor pairs in the hypercube topology. The square represents a 2-dimensional subcube over channels $c-1$ and c. Dashed line indicates the adjacency of the respective processor pair in the Hartley graph. The orderings in the lists indicate the local orderings of the *FHT* points in the H arrays of the respective processors.

In the following section, we propose and describe a restructuring which brings regularity to the $q \leftrightarrow s$ interactions, without disturbing the regularity of the $p \leftrightarrow q$ and $r \leftrightarrow s$ interactions. Then, we will propose a dynamic mapping scheme for the restructured algorithm which totally avoids the computational fragmentation of *FHT* butterflies, as is illustrated in Fig. 6(c).

3.1 Restructuring

The computational interdependencies between the successive levels of the *FHT* algorithm should be closely examined in order to achieve a suitable restructuring for an efficient parallelization. Two consecutive blocks B_ℓ^{2i} and B_ℓ^{2i+1} at level ℓ constitute the block $B_{\ell+1}^{i}$ at the next level $\ell+1$, for $i = 0, 1, \ldots, 2^{n-\ell-2}-1$. For example, in a 32-point *FHT* (see Fig. 4), two consecutive *FHT* blocks $B_2^2 = \{16-23\}$ and $B_2^3 = \{24-31\}$ at level $\ell = 2$ constitute the *FHT* block $B_3^1 = \{16-31\}$ at the next level $\ell = 3$. The $(\ell+1)th$ bits of the indices of all *FHT* points in even and odd numbered blocks B_ℓ^{2i} and B_ℓ^{2i+1} at level ℓ are 0 and 1, respectively. We can deduce the following two theorems by considering the butterfly pairs ($T_\ell^0 \in B_\ell^{2i}$, $T_\ell^1 \in B_\ell^{2i+1}$), where $T_\ell^1 - T_\ell^0 = 2^{\ell+1}$. Here, $T_\ell^1 - T_\ell^0 = 2^{\ell+1}$ denotes that, $p_\ell^1 - p_\ell^0 = r_\ell^1 - r_\ell^0 = q_\ell^1 - q_\ell^0 = s_\ell^1 - s_\ell^0 = 2^{\ell+1}$ where $T_\ell^1 = \{p_\ell^1, r_\ell^1, q_\ell^1, s_\ell^1\}$ and $T_\ell^0 = \{p_\ell^0, r_\ell^0, q_\ell^0, s_\ell^0\}$. That is, ($T_\ell^0$, T_ℓ^1) denotes the set of $2^{\ell-1}$ butterfly pairs in consecutive *FHT* blocks B_ℓ^{2i} and B_ℓ^{2i+1} at level ℓ such that the indices of the p, r, q and s points of the two butterflies in each pair differ only in their $(\ell+1)^{th}$ bits. For example, in a 32-point *FHT* (see Fig. 4), ($T_2^0 \in B_2^2$, $T_2^1 \in B_2^3$) denotes two butterfly pairs ($\{16, 18, 20, 22\}, \{24, 26, 28, 30\}$) and ($\{17, 19, 21, 23\}, \{25, 27, 29, 31\}$).

Figure 7: The combination structures of (a) type-1, (b) type-2 *FHT* butterfly pairs.

Theorem 1: Each level-ℓ ($\ell \geq 2$), type-1 *FHT* butterfly pair ($T1_\ell^0 \in B_\ell^{2i}, T1_\ell^1 \in B_\ell^{2i+1}$) constitute the type-1 butterfly pair $(FT1_{\ell+1}, ST1_{\ell+1}) \in B_{\ell+1}^i$ at the next level $\ell+1$, where

$$FT1_{\ell+1} = \{p_{\ell+1}^F, r_{\ell+1}^F, q_{\ell+1}^F, s_{\ell+1}^F\} = \{p_\ell^0, s_\ell^0, p_\ell^1, s_\ell^1\}$$
$$ST1_{\ell+1} = \{p_{\ell+1}^S, r_{\ell+1}^S, q_{\ell+1}^S, s_{\ell+1}^S\} = \{r_\ell^0, q_\ell^0, r_\ell^1, q_\ell^1\}$$

Proof: Since $T1_\ell^0$ and $T1_\ell^1$ are type-1 butterflies at level ℓ and $T1_\ell^1 - T1_\ell^0 = 2^{\ell+1}$, we have

$$\begin{array}{ll}
FT1_{\ell+1} & ST1_{\ell+1} \\
p_\ell^0 = b_k000f_{\ell-1} = b_k00g_\ell; & r_\ell^0 = b_k001f_{\ell-1}^c = b_k00g_\ell^c \\
s_\ell^0 = b_k011f_{\ell-1}^c = b_k01g_\ell^c; & q_\ell^0 = b_k010f_{\ell-1} = b_k01g_\ell \\
p_\ell^1 = b_k100f_{\ell-1} = b_k10g_\ell; & r_\ell^1 = b_k101f_{\ell-1}^c = b_k10g_\ell^c \\
s_\ell^1 = b_k111f_{\ell-1}^c = b_k11g_\ell^c; & q_\ell^1 = b_k110f_{\ell-1} = b_k11g_\ell
\end{array}$$

where $k = n-\ell-2$. Here, $g_\ell = 0f_{\ell-1} \neq \phi_\ell$, and $g_\ell^c = 1f_{\ell-1}^c \neq \phi_\ell$ since $f_{\ell-1} \neq \phi_{\ell-1}$ by Defininition 1. Hence, proof follows by Definition 1. □

Theorem 2: Each level-ℓ ($\ell \geq 1$), $type$-2 *FHT* butterfly pair ($T2_\ell^0 \in B_\ell^{2i}$, $T2_\ell^1 \in B_\ell^{2i+1}$) constitute the butterfly pair $(FT2_{\ell+1}, ST2_{\ell+1}) \in B_{\ell+1}^i$ at the next level $(\ell+1)$, where

$$FT2_{\ell+1} = \{p_{\ell+1}^F, r_{\ell+1}^F, q_{\ell+1}^F, s_{\ell+1}^F\} = \{p_\ell^0, q_\ell^0, p_\ell^1, q_\ell^1\}$$
$$ST1_{\ell+1} = \{p_{\ell+1}^S, r_{\ell+1}^S, q_{\ell+1}^S, s_{\ell+1}^S\} = \{r_\ell^0, s_\ell^0, r_\ell^1, s_\ell^1\}$$

are type-2 and type-1 butterflies, respectively.

Proof: Since $T2_\ell^0$ and $T2_\ell^1$ are $type$-2 butterflies at level ℓ and $T2_\ell^1 - T2_\ell^0 = 2^{\ell+1}$, we have

$$\begin{array}{ll} FT2_{\ell+1} & ST1_{\ell+1} \\ p_\ell^0 = b_k000\phi_{\ell-1} = b_k00\phi_\ell & r_\ell^0 = b_k00(1\phi_{\ell-1}) \\ q_\ell^0 = b_k010\phi_{\ell-1} = b_k01\phi_\ell & s_\ell^0 = b_k01(1\phi_{\ell-1}) \\ p_\ell^1 = b_k100\phi_{\ell-1} = b_k10\phi_\ell & r_\ell^0 = b_k10(1\phi_{\ell-1}) \\ q_\ell^1 = b_k110\phi_{\ell-1} = b_k11\phi_\ell & s_\ell^0 = b_k11(1\phi_{\ell-1}) \end{array}$$

where $k = n-\ell-2$. Proof follows by Definitions 2 and 1 since $0\phi_{\ell-1} = \phi_\ell$ and ℓ-bit 2's complement of $(1\phi_{\ell-1})$ is equal to itself. □

Fig. 7 illustrates the combination structures of type-1 and type-2 butterfliy pairs. As is seen in Fig. 4, in a 32-point *FHT*, the type-1 butterfly pair ($\{1,7,9,15\}\in B_3^0$, $\{17,23,25,31\}\in B_3^1$) at level $\ell=3$, constitutes the type-1 butterfly pair ($\{1,15,17,31\}$, $\{7,9,23,25\}$)$\in B_4^0$ at the next level $\ell = 4$. Similarly, the type-2 butterfly pair ($\{0,4,8,12\}\in B_3^0$, $\{16,20,24,28\}\in B_3^1$) at level $\ell=3$, constitute the (type-2, type-1) butterfly pair ($\{0,8,16,24\}$, $\{4,12,20,28\}$)$\in B_4^0$ at the next level $\ell=4$.

In the discussions given so far, p, r, q and s labels were used both to identify different points of *FHT* butterflies and the decimal indices of the corresponding *FHT* points in the H-array. However, for the sake of clarity of further discussions, p, r, q and s labels will be used only to identify different points of *FHT* butterflies, whereas i and j labels will be used to identify their decimal indices in the H-array. For example, we will consider the combination structures of the butterfly pairs (T_ℓ^0, T_ℓ^1) where

$$\begin{aligned} T_\ell^0 &= \{p_\ell^0, r_\ell^0, q_\ell^0, s_\ell^0\} = \{i_1, i_2, i_3, i_4\} \\ T_\ell^1 &= \{p_\ell^1, r_\ell^1, q_\ell^1, s_\ell^1\} = \{j_1, j_2, j_3, j_4\}. \end{aligned}$$

Note that i and j indices satisfy the same relations previously defined for p, r, q and s points. That is, $i_3 = i_1 + 2^\ell, i_4 = i_2 + 2^\ell, j_3 = j_1 + 2^\ell, j_4 = j_2 + 2^\ell, j_1 - i_1 = j_2 - i_2 = j_3 - i_3 = j_4 - i_4 = 2^{\ell+1}, \ldots, etc.$ In this notation, Theorems 1 and 2 can be re-stated as follows: level-(ℓ+1) ($FT1_{\ell+1}$, $ST1_{\ell+1}$) and ($FT2_{\ell+1}$, $ST1_{\ell+1}$) pairs generated by type-1 $(T1_\ell^0, T1_\ell^1)$ and type-2 $(T2_\ell^0, T2_\ell^1)$ pairs will have the following structure in the H-array;

$$\begin{array}{ll} FT1_{\ell+1} = \{i_1, i_4, j_1, j_4\} & ST1_{\ell+1} = \{i_2, i_3, j_2, j_3\} \\ FT2_{\ell+1} = \{i_1, i_3, j_1, j_3\} & ST1_{\ell+1} = \{i_2, i_4, j_2, j_4\}, \end{array}$$

respectively.

Theorems 1 and 2 reveal that regularly separated (by powers of 2's) butterfly pairs at a particular level constitute scrambled butterfly pairs at the following level. The scrambled combination of the butterfly pairs is the main reason for the irregular spacing between q and s points of type-1 butterflies in the following levels. However, this scrambling between butterfly pairs can be avoided by a clever re-ordering while storing the computational results of each butterfly into the H-array. This internal re-ordering will be different for type-1 and type-2 butterflies since the combination structures of these two types of butterfly pairs are different from each other. Combination structure of type-2 FHT butterfly pairs is also investigated since they generate a single type-1 butterfly at the following level.

The scrambled combination of type-1 butterfly pairs are avoided by swapping r and s points type-1 butterflies while storing their updated values into the H-array. The scrambled combination of type-2 butterfly pairs are avoided by swapping r and q points of type-2 butterflies while storing their updated values into the H-array. In this scheme, the results of type-1 $(T1_\ell^0, T1_\ell^1)$ and type-2 $(T2_\ell^0, T2_\ell^1)$ pairs will have the following order in the H-array at the completion of level-ℓ computations;

$$T1_\ell^0 = \{i_1, i_4, i_3, i_2\} \qquad T1_\ell^1 = \{j_1, j_4, j_3, j_2\}.$$
$$T2_\ell^0 = \{i_1, i_3, i_2, i_4\} \qquad T2_\ell^1 = \{j_1, j_3, j_2, j_4\},$$

respectively. Hence, in the proposed scheme, the generated type-1 $(FT1_{\ell+1}, ST1_{\ell+1})$ and (type-2, type-1) $(FT2_{\ell+1}, ST1_{\ell+1})$ pairs will have the following structure in the H-array:

$$\begin{aligned} FT1_{\ell+1} &= \{p_\ell^0, s_\ell^0, p_\ell^1, s_\ell^1\} & ST1_{\ell+1} &= \{r_\ell^0, q_\ell^0, r_\ell^1, q_\ell^1\} \\ &= \{i_1, i_2, j_1, j_2\} & &= \{i_4, i_3, j_4, j_3\} \\ FT2_{\ell+1} &= \{p_\ell^0, q_\ell^0, p_\ell^1, q_\ell^1\} & ST1_{\ell+1} &= \{r_\ell^0, s_\ell^0, r_\ell^1, s_\ell^1\} \\ &= \{i_1, i_2, j_1, j_2\} & &= \{i_3, i_4, j_3, j_4\} \end{aligned}$$

according to Theorems 1 and 2, respectively. Fig. 8 illustrates the alignment operations during the computation of restructured *FHT* butterflies, and the combination structures of the restructured butterfly pairs. The computations involved in a restructured type-1 simplified *FHT* butterfly are:

$$qtemp := Ci \times H[q] + Si \times H[s]; \tag{8a}$$
$$stemp := Cj \times H[s] + Sj \times H[q]; \tag{8b}$$
$$H[q] \ := H[p] - qtemp; \tag{8c}$$
$$H[s] \ := H[r] + stemp; \tag{8d}$$
$$H[p] \ := H[p] + qtemp; \tag{8e}$$
$$H[r] \ := H[r] - stemp; \tag{8f}$$

The computations involved in a restructured type-2 simplified *FHT* butterfly are:

$$qtemp := H[q]; \tag{9a}$$
$$stemp := H[s]; \tag{9b}$$
$$H[q] \ := H[r] + stemp; \tag{9c}$$
$$H[s] \ := H[r] - stemp; \tag{9d}$$
$$H[r] \ := H[p] - qtemp; \tag{9e}$$
$$H[p] \ := H[p] + qtemp; \tag{9f}$$

Comparison of (8) with (3), and (9) with (4) reveals that the proposed restructuring does not introduce any computational overhead. The proposed restructuring has the following nice features. The combination structures of both types of butterfly pairs are very similar. Consider both type-1 and type-2 level-ℓ butterfly pairs (T_ℓ^0, T_ℓ^1) that combine to constitute the $(FT_{\ell+1}, ST_{\ell+1})$ butterfly pairs at the next level $\ell+1$. The first (last) two *FHT* points of T_ℓ^0 followed by the first (last) two *FHT* points of T_ℓ^1 will constitute $FT_{\ell+1}$ $(ST_{\ell+1})$ respectively, at the next level. The only difference is the reverse allocation of the *FHT* points of the (p,r) and (q,s) pairs of the second $ST_{\ell+1}$ butterfly in the H-array when (T_ℓ^0, T_ℓ^1) is a $type$-1 butterfly pair. Note that the proposed restructuring avoids the scrambled combination structure between butterfly pairs at successive levels. Furthermore, in the proposed scheme, p, r points and q, s points of both $FT_{\ell+1}$ and $ST_{\ell+1}$ will be allocated to the consecutive locations of the H-array if p, r points and q, s points of both T_ℓ^0 and T_ℓ^1 are initially allocated to the consecutive locations of the H-array. This structure is valid for both types of butterfly pairs in the proposed restructuring scheme, since (p, r) and (q, s) pairs of $FT_{\ell+1}$

Figure 8: The combination structures of (a) type-1, (b) type-2 restructured *FHT* butterfly pairs.

and $ST_{\ell+1}$ constitute the first two and last two points of both types of T_ℓ^0 and T_ℓ^1 butterflies. That is, if

$$T1_\ell^0 = \{i_1, i_1+1, i_3, i_3+1\}$$
$$T1_\ell^1 = \{j_1, j_1+1, j_3, j_3+1\}$$

then we will have

$$FT1_{\ell+1}^1 = \{i_1, i_1+1, j_1, j_1+1\}$$
$$ST1_{\ell+1}^1 = \{i_3+1, i_3, j_3+1, j_3\}.$$

Similarly, if

$$T2_\ell^0 = \{i_1, i_1+1, i_3, i_3+1\}$$
$$T2_\ell^1 = \{j_1, j_1+1, j_3, j_3+1\}$$

then we will have

$$FT2_{\ell+1}^1 = \{i_1, i_1+1, j_1, j_1+1\}$$
$$ST1_{\ell+1}^1 = \{i_3, i_3+1, j_3, j_3+1\}.$$

This important feature of the proposed restructuring scheme will be exploited to avoid the fragmentation of the (q, s) pairs of type-1 butterflies during the parallelization.

In the original *FHT* algorithm, 4-point butterfly computations start at level $\ell = 1$ which contains only type-2 butterflies. Note that p, r points and q, s points of all type-2 butterflies at level $\ell = 1$ are already allocated to the consecutive locations of the H-array. Hence, if the proposed re-structuring is applied starting from level $\ell = 1$, then p, r points and q, s points of all butterflies at the following levels will be allocated to the consecutive locations of the H-array. Fig. 9 illustrates the computational flow-graph for the restructured 32-point *FHT* algorithm. As is seen in Fig. 9, the type-1 butterfly pair ({18,19,22,23}, {26,27,30,31}) at level $\ell = 2$ constitute the type-1 butterfly pair, ({18,19,26,27}, {23,22,31,30}) at the following level $\ell = 3$. Similarly, type-2 butterfly pair ({16,17,20,21}, {24,25,28,29}) at level $\ell = 2$ constitute the (type-2, type-1) butterfly pair ({16,17,24,25}, {20,21,28,29}) at the following level $\ell = 3$. As is also seen in Fig. 9, the proposed restructuring does not disturb the block structure of the original *FHT* algorithm. Furthermore, the proposed restructuring brings regularity and symmetry to the in-block allocation structure of the *FHT* butterflies. The following paragraph explains the regular allocation structure of $2^{\ell-1} = 2^{\ell+1}/4$ butterflies in each block at level ℓ for $\ell = 1, 2, \ldots, n-1$.

In each block, $2^{\ell-1}$ consecutive *FHT*-point pairs in the first and second halves constitute the (p, r) and (q, s) pairs, respectively, of the butterflies involved in that block. Consecutive *FHT*-point pairs in each half are ordered regularly such that ith pairs in the first and second halves constitute the (p, r) and (q, s) pairs of the same butterfly, respectively, for $i = 0, 1, \ldots, 2^{\ell-1}$. The first pairs ($i = 0$) in each half constitute the only type-2 butterfly involved in that block. The following $2^{\ell-1} - 1$ consecutive pairs ($i = 1, 2, \ldots, 2^{\ell-1} - 1$) in each half constitute $(2^{\ell-1} - 1)$ type-1 butterflies involved in that block. However, the last $(2^{\ell-2} - 1)$ consecutive pairs ($i = 2^{\ell-2} + 1, \ldots, 2^{\ell-1} - 1$) in each half hold the *FHT* points of (p, r) and (q, s) pairs in the reverse order (i.e., as $\{r, p\}$ and $\{s, q\}$). These reverse ordered (p, r) and (q, s) pairs belong to the second type-1 butterflies generated from type-1 butterfly pairs in the previous level.

For example, in a 32-point restructured *FHT* algorithm (See Fig. 9), the 4-tuples $\{0,1,8,9\}$, $\{2,3,10,11\}$, $\{4,5,12,13\}$, $\{7,6,15,14\}$ constitute the $2^{3-1} = 4$ *FHT* butterflies involved in block $B_3^0 = \{0-15\}$ at level $\ell = 3$. Note that the first butterfly $\{0,1,8,9\}$ is the only type-2 butterfly involved in B_3^0. Also note that (p, r) and (q, s) pairs of only the last type-1 butterfly $\{7,6,15,14\}$ are hold in reverse order in the H-array since $2^{3-2} - 1 = 1$. As is seen in Fig. 9, this type-1 butterfly is the second butterfly generated by the type-1 butterfly pair ($\{2,3,6,7\}$, $\{10,11,14,15\}$) in the previous level ($\ell = 2$).

Fig. 10 illustrates the pseudo-code for the restructured *FHT* algorithm. Note that this algorithm has a very similar structure compared to standard algorithm given in Fig. 5 since both programs exploit the block structure of the *FHT* computations at each level. However, the assignment statements for p, r, q, s indices are different due to the restructuring. Furthermore, (8) and (9) are used instead of (3) and (4), respectively, in order to realize the internal alignment operations for the restructured butterfly computations. The last $2^{\ell-2} - 1$ iterations of the innermost *for-loop* for $\ell \geq 3$ need extra attention since *FHT* points of the last $(2^{\ell-2} - 1)$ (p, r) and (q, s) pairs of each block are hold in reverse order in the H-array during these levels. A careful analysis of (3) reveals the symmetry between the computations of p and r points, and q and s points of type-1 butterflies. That is, correct values for the type-1 butterflies will also be computed if we interchange p with r, q with s, and i with j in (3). In this case, *qtemp* will hold the correct value of *stemp* and vice versa. This symmetry in type-1 butterfly computations is exploited in the restructured *FHT* algorithm as follows. The first two lines in the innermost *for-loop* computes the indices of the p, r, q, s points of type-1 butterflies involved in a particular block assuming a proper ordering of the *FHT* points in (p, r) and (q, s) pairs. Hence, during the first $2^{\ell-2}$ iterations, $p1, r1, q1, s1$ variables refer to the correct *FHT* points p, r, q, s, respectively, in the H-array. However, during the last $2^{\ell-2} - 1$ iterations, $p1, r1, q1, s1$ indices refer to r, p, s, q points, respectively, in the H-array. Thus, this scheme implicitly achieves the interchange of p with r, and q with s. The interchange of the Cos/Sin factors (i.e., interchange of i and j) is also achieved implicitly during construction of the Cos/Sin factor index tables prior to the execution of the program. As is seen in Fig. 9, at level $\ell = 4$, i/j indices of the last

Figure 9: Computational flow graph for a 32-point restructured *FHT* and its tiled mapping on a 2-dimensional hypercube. The nonlocal alignment operations during the last two levels $\ell = 2$ and $\ell = 3$ correspond to mapping exchanges of the respective *FHT* points

$2^{4-2} - 1 = 3$ Cos/Sin factor pairs appear in reverse order (as j/i; 9/7, 10/6, 13/3). Hence, the last four statements of the innermost *for-loop* effectively computes the correct values for the s, p, r, q points of type-1 butterflies, and stores them into $H[q1]$, $H[s1]$, $H[p1]$, $H[r1]$, respectively. Thus, the updated values of the s, p, r, q points of type-1 butterflies are effectively stored into their s, q, r, p locations, respectively. Hence, p and q points of type-1 butterflies are effectively swapped, instead of r and s points, during these iterations.

The implementation scheme proposed in Fig. 10 modifies the combination structure of the last $2^{\ell-2}-1$ type-1 butterfly pairs ($T1_\ell^0$, $T1_\ell^1$) in each block pair (B_ℓ^{2i}, B_ℓ^{2i+1}), at levels $\ell \geq 3$. We need to examine the combination structure of these *reverse* butterfly pairs in order to show that the implementation scheme in Fig. 10 does not disturb the regularity and symmetry of the proposed restructuring. Consider the reverse type-1 ($T1_\ell^0$, $T1_\ell^1$) butterfly pairs, where

$$T1_\ell^0 = \{i_1 + 1, i_1, i_3 + 1, i_3\}$$
$$T1_\ell^1 = \{j_1 + 1, j_1, j_3 + 1, j_3\}.$$

```
/* Input in bit-reversed order in H[0 ... N-1]   */
/* Output in normal order in H[0 ... N-1]     */
for i := 0 to N/2−1 do
    temp := H[2i + 1];
    H[2i + 1] := H[2i] − temp;
    H[2i] := H[2i] + temp;

for ℓ := 1 to n − 1 do
    for i := 0 to N/2^(ℓ+1) − 1 do
        p2 := i × 2^(ℓ+1);      q2 := p2 + 2^ℓ;
        r2 := p2 + 1;           s2 := q2 + 1;
        qtemp := H[q2];   stemp := H[s2];
        H[q2] := H[r2] + stemp;
        H[s2] := H[r2] − stemp;
        H[r2] := H[p2] − qtemp;
        H[p2] := H[p2] + qtemp;

        for j := 1 to 2^(ℓ−1) − 1 do
            p1 := p2 + 2 × j;      q1 := p1 + 2^ℓ;
            r1 := p1 + 1;          s1 := q1 + 1;
            qtemp := Cfac1 × H[q1] + Sfac1 × H[s1];
            stemp := Cfac2 × H[s1] + Sfac2 × H[q1];
            H[q1] := H[p1] − qtemp;
            H[s1] := H[r1] + stemp;
            H[p1] := H[p1] + qtemp;
            H[r1] := H[r1] − stemp;
```

Figure 10: Restructured sequential ($N = 2^n$)-point *FHT* algorithm

The algorithm in Fig. 10 effectively swaps p and q points of reverse type-1 butterfly pairs during the alignment operation. Hence, reverse type-1 butterfly pairs will have the following allocation structure

$$T1_\ell^0 = \{i_3 + 1, i_1, i_1 + 1, i_3\}$$
$$T1_\ell^1 = \{j_3 + 1, j_1, j_1 + 1, j_3\}.$$

in the H-array just after the alignment operations. Thus, according to Theorem 1, type-1 ($FT1_{\ell+1}$, $ST1_{\ell+1}$) pairs generated by the reverse type-1 butterfly pairs will have the following structure:

$$FT1_{\ell+1} = \{p_\ell^0, s_\ell^0, p_\ell^1, s_\ell^1\} = \{i_3 + 1, i_3, j_3 + 1, j_3\}$$
$$ST1_{\ell+1} = \{r_\ell^0, q_\ell^0, r_\ell^1, q_\ell^1\} = \{i_1, i_1 + 1, j_1, j_1 + 1\}$$

in the H-array. For example, type-1 ({7,6,15,14}, {23,22,31,30}) butterfly pair at level $\ell = 3$ constitutes the type-1 ({15,14,31,30}, {6,7,22,23}) butterfly pair at the next level $\ell = 4$. It is clear that, ($ST1_{\ell+1}, FT1_{\ell+1}$) butterfly pairs generated during the last $2^{\ell-2} - 1$ iterations will have the same spatial structure compared to the ($FT1_{\ell+1}, ST1_{\ell+1}$) butterfly pairs generated during the first $2^{\ell-2}$ iterations of the innermost *for-loop*. Hence, the scheme proposed in Fig. 10 maintains the regular and symmetrical features of the restructured *FHT* algorithm without disturbing the simplicity and regularity of programming.

As is seen in Fig. 9, the order of the output results is scrambled in the proposed restructured *FHT* algorithm. However, in most of the *DSP* applications a sequence of *DSP* blocks are applied consecutively on a set of input data. A proper output/input interface between successive *DSP* blocks can always be maintained, if the output or input data order of a particular *DSP* block is disturbed for the sake of efficiency. Hence, the order of input and output data of individual *DSP* blocks does not bring any inefficiency to the overall application.

3.2 Dynamic Mapping

Consider the performance of the tiled mapping scheme for the parallelization of the restructured *FHT* algorithm. The internal alignment operations for the restructured butterflies will correspond to simple local swap operations during the first $n-d$ levels since the tiled mapping prevents the fragmentation of butterflies during these levels. However, these alignment operations will necessitate mapping exchange communications after the second stage computations of the last d levels because of the fragmentation of butterflies during these levels. The nonlocal alignment operations performed at the end of each level ℓ, for $\ell = n-d, \ldots, n-2$, confine the *FHT* butterflies of the next level $(\ell+1)$ to 1-dimensional subcubes over channel $c = \ell - n + d + 1$. The d-bit binary representations of the two processors in each subcube differ only in their cth bit such that this bit is "0" and "1"in the first and second processors of the subcube, respectively. The fragmentation of *FHT* butterflies across these subcubes is such that first and second processors in each subcube hold and are responsible for computing $M/2$ (p, r) and (q, s) pairs, respectively, of the M butterflies confined to that subcube. Hence, each level ℓ of the last d levels require two concurrent single-hop exchange communications both over channel $c = \ell - n + d$. The first concurrent exchange communication, of volume M *FHT* points, is due to the $p \leftrightarrow q$ and $r \leftrightarrow s$ interactions. The second concurrent exchange communication, of volume $M/2$ *FHT* points, is a mapping exchange operation due to the nonlocal alignment operations. Thus, the proposed restructuring reduces the number and volume of concurrent communications to $2d$ and $3dM/2$ *FHT* points, respectively. Although this scheme achieves perfect load balance for the basic butterfly scheme it doesn't achieve perfect load balance for the simplified butterfly scheme because of the fragmentation of butterflies during the last d levels.

In this section, we propose a dynamic mapping scheme for the restructured *FHT* algorithm which prevents the fragmentation of *FHT* butterflies. Starting with the initial tiled mapping, alignment operations in the restructured *FHT* algorithm do not fragment the butterflies during the first $n-d$ levels, and confines the butterflies to 1-dimensional subcubes during the last d levels. The first and second processors in each subcube hold (p, r) and (q, s) pairs of the butterflies confined to that subcube. In the proposed scheme, at the beginning of each level ℓ during the last d levels, first and second processors in each subcube exchange the appropriate halves of their local (p, r) and (q, s) pairs, respectively, such that each processor gathers $M/4$ unfragmented butterflies. This exchange communication is a mapping exchange operation which effectively exchanges the responsibility of further computations associated with those exchanged *FHT* points. The $M/2$ butterflies fragmented across the two processors of each subcube are evenly divided between these two processors after the mapping exchange communication. Hence, this scheme achieves perfect load balance both for the basic and simplified butterfly schemes, since it gathers and assigns equal number of unfragmented butterflies to each processor at each level. These mapping exchange operations are the only communication requirement of the proposed scheme since they gather and assign unfragmented butterflies to all processors at each level of the last d levels. Hence, in this scheme, each level ℓ of the last d levels require only one concurrent single-hop exchange communication, of volume $M/2$ *FHT* points, over channel $c = \ell - n + d$. Thus, the proposed scheme reduces the number and volume of concurrent communications to d and $dM/2$ *FHT* points, respectively. In this scheme, the alignment

Figure 11: Dynamic mapping of a 32-point restructured *FHT* on a 2-dimensional hypercube

operations associated with the restructured *FHT* butterflies remain as simple local swaps during the last d levels. These local alignment operations maintain the regularity of local *FHT* computations, enable *in-place* local computations and communications.

Fig. 11 illustrates the proposed *dynamic* mapping scheme for a 32-point restructured *FHT* on a 2-dimensional hypercube. Fig. 12 illustrates the pseudo-code for the node program of the proposed parallel *FHT* algorithm. The pseudo-code is given only for the last d levels, since the pseudo-code for the node program is very similar to the restructured sequential *FHT* algorithm (Fig. 10) for the first $(n-d)$ levels. As is seen in Fig. 11, the computational flow graphs for the local *FHT* computations performed by processors during the first $(n-d)$ levels are exactly same as the computational flow graph for the M-point *FHT* algorithm. That is, P processors can be considered as concurrently computing P independent M-point *FHT* (using proper Cos/Sin factors for the N-point *FHT*) during the first $n-d$ levels, Hence, the pseudo-code of the node program for the first $n-d$ levels of the parallel algorithm can easily be obtained by replacing variables N and n in Fig. 10 with M and $m = \lg_2 M$, respectively.

In the first inner *if-then-else* statement of Fig. 12, each processor identifies itself either as the first or the second processor in the respective 1-dimensional subcube by simply checking the cth bit of its processor index. Here, *mynode* is assumed to be a d-bit binary number representing the index of the respective processor. The variable c denotes the channel over which the mapping exchange operation is to be performed at that level. Then, each first processor exchanges the second half of its local H-array with the first half of the local H-array of the respective second processor, and vice-versa. Hence, first processors effectively exchange their local $M/4$ (p, r) pairs with the local $M/4$ (q, s) pairs of the respective second processors, and vice-versa. The pr and qs indices used inside the first *if-then-else* statement identify the nature of the *FHT* points being sent and received.

```
/* Computations over the last d bits */
/* d concurrent exchange communication phase */
/* H : a real array of size M = N/P */

for ℓ := n−d to n−1 do
    c := ℓ − (n − d);
    dnode := mynode ⊕ 2^c;

    if (cth bit of mynode is 0) then do
        send from (H[pr]: pr = M/2, ..., M−1)  to  dnode;
        recv into (H[qs]: qs = M/2, ..., M−1) from dnode;
    else
        send from (H[qs]: qs = 0, ..., M/2−1)  to  dnode;
        recv into (H[pr]: pr = 0, ..., M/2−1) from dnode;

    if (mynode mod 2^(c+1) = 0) then do
        p2 := 0;          r2 = 1;
        q2 := M/2;        s2 = q2 + 1;
        qtemp := H[q2];   stemp := H[s2];
        H[q2] := H[r2] + stemp;
        H[s2] := H[r2] − stemp;
        H[r2] := H[p2] − qtemp;
        H[p2] := H[p2] + qtemp;
    else
        p1 := 0;          r1 = 1;
        q1 := M/2;        s1 = q1 + 1;
        qtemp := Cfac1 × H[q1] + Sfac1 × H[s1];
        stemp := Cfac2 × H[s1] + Sfac2 × H[q1];
        H[q1] := H[p1] − qtemp;
        H[s1] := H[r1] + stemp;
        H[p1] := H[p1] + qtemp;
        H[r1] := H[r1] − stemp;

    for i := 1 to M/4−1 do
        p1 := 2 × i;         r1 := p1 + 1;
        q1 := p1 + M/2;      s1 := q1 + 1;
        qtemp := Cfac1 × H[q1] + Sfac1 × H[s1];
        stemp := Cfac2 × H[s1] + Sfac2 × H[q1];
        H[q1] := H[p1] − qtemp;
        H[s1] := H[r1] + stemp;
        H[p1] := H[p1] + qtemp;
        H[r1] := H[r1] − stemp;
```

Figure 12: Parallel ($N = 2^n$)-point restructured *FHT* algorithm with dynamic mapping for a d-dimensional hypercube with $P = 2^d$ processors (last d levels).

The proposed parallel *FHT* algorithm does not necessitate any extra send or receive buffers. All communications are initiated from/into contiguous locations of the local H arrays thus avoiding any scatter/gather type of local operations for communications. Note that first and second processors at a particular level use the second and first halves of their local H-arrays, respectively, as contiguous *send* and *receive* buffers for the exchange communication operations. Hence, the proposed scheme has a very regular *in-place* communication structure. In Fig. 12, *send* and *recv* denote *synchronous* (*blocking*) send and receive primitives. Synchronous send/receive operations are used to prevent the contamination of the message to be sent with the incoming message since the same half of the local H-array is used both as send and receive buffers at a particular level.

Fig. 13 illustrates the computation and communication structure of the proposed parallel algorithm for a 32-point *FHT* on a 3-dimensional hypercube. First level $\ell = 0$ is not shown in Fig. 13 since it neither involves communications nor local alignment operations. Circles indicate processors and numbers inside the circles indicate the indices of the respective processors. Each processor is associated with two lists (of length $M = 4$) at each level. Upper and lower lists denote the order of the local *FHT* points before and after the restructured butterfly computations. Wide and narrow crosses in this figure represent the local alignment operations for type-1 and type-2 restructured butterflies, respectively. Solid lines represent the communication links over which the concurrent mapping exchange communication occurs at a particular level. Processor pairs connected with solid lines represent the 1-dimensional subcubes discussed earlier. The sublists (of length 2) at the tails of arrows represent the *FHT* points transmitted in the respective directions during a particular concurrent exchange communication.

At the beginning of each level during the last d levels, the ith local *FHT*-point pairs in the first and second processors of each 1-dimensional subcube correspond to the (p, r) and (q, s) pairs of the same butterfly, for $i = 0, 1, \ldots, M/2-1$. Hence, after the mapping exchange communication, the ith *FHT*-point pairs in the first and second halves of each processor correspond to the same butterfly, for $i = 0, 1, \ldots, M/4-1$. Thus, as is also seen in Fig. 12, each processor performs *simplified FHT* butterfly computations on local (p, r) and (q, s) pairs separated by $M/2 = N/2P$. The proposed parallel *FHT* algorithm has a very regular *in-place* computational structure and hence can also be implemented on *SIMD* type hypercubes efficiently.

Although butterflies are partitioned evenly among processors throughout the algorithm, the type of butterflies the processors compute during the last d levels, are not. *FHT* block sizes increase as $2, 4, \ldots, 2^{n-d}$ during the first $n-d$ levels. Thus, each processor computes equal number of *FHT* blocks during the first $n-d$ levels, since tiled mapping assigns consecutive $M = N/P = 2^{n-d}$ *FHT* points in blocks to processors. Recall that each *FHT* block at a particular level $\ell \geq 1$ contains one type-2 and $2^{\ell-1}-1$ type-1 butterflies Hence, type-1 and type-2 butterflies are partitioned evenly among processors at each level $\ell = 1, \ldots, n-d-1$. That is, each processor computes $M/2$ 2-point butterflies, $M/2^{\ell+1}$ type-2 butterflies, and $M/4 - M/2^{\ell+1}$ type-1 butterflies during the first $n-d$ levels. Hence, there is no deviation from the perfect load balance during the first $n-d$ levels. However, the number of type-2 butterflies is $P/2$ at level $\ell = n-d$, decreases by one half during the following $d-1$ levels, and reduces to 1 at the last level. Hence, even distribution of type-2 butterflies is not possible during the last d levels. After

Figure 13: Computation and communication structure of the proposed parallel algorithm for a 32-point *FHT* on a 3-dimensional hypercube (first level ℓ=0 is not shown).

the mapping exchange operation at each level ℓ of the last d levels, the first butterfly of $M/4$ butterflies in each processor is a type-2 butterfly if least significant $c+1$ bits of the prorocessor are all $0's$, where $c = \ell - n + d$. Otherwise, it is a type-1 butterfly as well as the remaining $M/4-1$ butterflies. So, at each level ℓ of the last d levels, $P/2^c$ processors compute one type-2 and $M/4-1$ type-1 butterflies, while the others compute $M/4$ type-1 butterflies, where $c = \ell - n + d$. As is seen in Fig. 12, this difference in local computations is resolved simply by the second *if-then-else* statement before the inner *for-loop*.

The parallel execution time of the proposed *FHT* algorithm can be modeled as

$$T_{par} = \frac{1}{P} T_{seq}^{n-d} + 10\frac{N}{4P} t_{calc} + \left(t_{su} + \frac{N}{2P} t_{tr}\right) \lg_2 P \tag{10}$$

where t_{su} is the message startup time overhead and t_{tr} is the time taken for the transmission of a floating-point word (4-bytes). The first and second terms in (10) represent the parallel execution times of the first $n-d$ and last d levels, respectively. Note that bottleneck processors which compute only type-1 butterflies during the last d levels determine the parallel execution times of these levels. In the first term, T_{seq}^{n-d} represents the sequential execution time of the first $n-d$ levels. The expression for T_{seq}^{n-d} can be derived by using (5) as follows

$$\begin{aligned} T_{seq}^{n-d} &= \big(N + 10 \sum_{\ell=2}^{n-d-1} N_{T1} + 4 \sum_{\ell=1}^{n-d-1} N_{T2}\big) t_{calc} \\ &= \big(2.5N \lg_2 \frac{N}{P} - 4.5N - 6P\big) t_{calc} \end{aligned} \tag{11}$$

Substituting (11) into (10) we obtain

$$T_{par} = \left(\frac{2.5N}{P} \lg_2 N - 4.5\frac{N}{P} + 6\right) t_{calc} + \left(t_{su} + \frac{N}{2P} t_{tr}\right) \lg_2 P \tag{12}$$

Comparing the first term of (12) with the expression given for the overall sequential execution time T_{seq} in (7), we can rewrite (12) as

$$T_{par} = \frac{1}{P} T_{seq} + \left(t_{su} + \frac{N}{2P} t_{tr}\right) \lg_2 P + \frac{6(P-1)}{P} t_{calc} \tag{13}$$

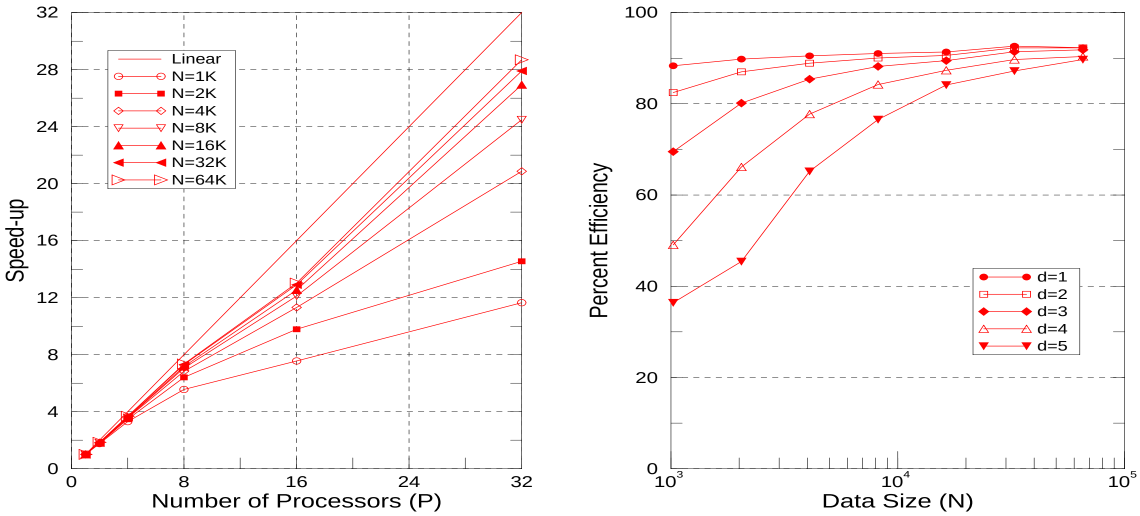
The first two terms in (13) represent the parallel execution time under perfect load balance conditions. The last term in (13) represents the slight deviation from the perfect load balance as a parallel computational overhead term. This overhead, which is always smaller than the machine specific constant $6t_{calc}$, can be neglected for sufficiently large N/P or N values.

4 Experimental Results

All programs introduced in this work (Figs. 5, 10, and 12) are coded in C language and run on an Intel's *iPSC/2* hypercube with 32 processors for various $N = 2^n$ data sizes, $128 \leq N \leq 64K$. The performance of the original and the restructured sequential *FHT* algorithms (Figs. 5 and 10, respectively) are observed to be the same, as is expected. Parallel *FHT* algorithm with static tiled mapping is not implemented for reasons of losing load balance, high number and volume of communications as well as multi-hop communications. The coarse grain extension of Hou's algorithm is also not implemented for similar reasons;

Table 1: Parallel execution times (in msecs) of *Lin's* and the proposed (*Our*) parallel algorithms for various size *FHTs* on 3 and 4 dimensional hypercubes during the d exchange communication phase.

N	$P = 8$			$P = 16$		
	Exec. time			*Exec. time*		
	Lin's	*Our*	Lin/Our	*Lin's*	*Our*	Lin/Our
1K	11	8	1.38	9	8	1.13
2K	19	14	1.36	14	11	1.27
4K	34	25	1.36	25	19	1.32
8K	65	47	1.38	46	33	1.39
16K	131	91	1.44	90	65	1.38
32K	267	181	1.48	178	126	1.41
64K	521	332	1.57	353	226	1.56



(a) (b)

Figure 14: (a) speed-up and (b) efficiency curves for the proposed parallel *FHT* algorithm

losing load balance, large number and volume of concurrent communications. The performance of the proposed parallel restructured *FHT* algorithm with dynamic mapping (Fig. 12) is evaluated in comparison with Lin's [8] algorithm. Table 1 illustrates the parallel performance comparison of Lin's and the proposed algorithms. As is described earlier, the parallel computational performance of Lin's algorithm reduces to that of the basic butterfly scheme. Recall that $t_{basic}/t_{simp} = 1.6$ where t_{basic} and t_{simp} denote the computational complexity of type-1 basic and simplified butterflies, respectively. As is seen in Table 1, the experimental performance ratio of the proposed algorithm to Lin's algorithm approaches to this ratio with increasing *FHT* size. Larger communication volume overhead of Lin's algorithm does not introduce significant decrease in its relative performance on *iPSC/2* compared to the proposed algorithm because of the small $t_{tr}/t_{calc} \approx 0.25$ value. Furthermore, index computation overhead of Lin's algorithm is less than that of the proposed algorithm (2 versus 4 per butterfly). The experimental performance ratio values do not exceed the value 1.6 because of the above mentioned reasons. However, the relative performance of the proposed algorithm compared to Lin's algorithm is expected to be much higher on hypercubes with larger t_{tr}/t_{calc} values. The relative performance is also expected to increase with increasing hypercube dimension since Lin's algorithm introduces congestion during the last $d-2$ levels of the d concurrent exchange communication phase due to the multi-hop messages during these levels.

Fig. 14 displays the speed-up and efficiency curves for the proposed parallel *FHT* algorithm. As is seen in Fig. 14, nearly linear speed-up is achieved for large N. As is also seen in Fig. 14, efficiency remains over 85% when $N/P \geq 512$ *FHT* points are mapped to an individual processor of the hypercube. Relatively small efficiency values for small size problems on large dimensional hypercubes are due to the high communication latency ($t_{su} \gg t_{calc}$) value of the *iPSC/2* architecture.

5 Conclusion

The Fast Hartley Transform which is a promising alternative to the Fast Fourier Transform is parallelized for hypercube-connected multicomputers. The proposed restructured sequential *FHT* algorithm brings regularity and symmetry to the computation of *FHT*. The proposed parallel *FHT* algorithm which exploits this restructuring and uses the dynamic mapping scheme achieves both perfect load-balance and nearest-neighbor communications, requires only d concurrent exchange communications by eliminating fragmentary message passing, and has a concurrent communication volume of $N/2P$ *FHT* points per exchange step. The proposed parallel algorithm also achieves *in-place* computation and communication. The proposed parallel *FHT* algorithm is implemented on an Intel's *iPSC/2* hypercube multicomputer with 32 processors. High efficiency values are obtained even for small size problems.

References

[1] R. N. Bracewell, "The Fast Hartley Transform," in *Proc. of IEEE*, vol. 72, no. 8, Aug. 1984.

[2] O. Buneman, "Conversion of FFT's to Fast Hartley Transforms", *SIAM J. Sci. Stat. Comput.*, vol. 7, No. 2, pp. 624-638, April 1986.

[3] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297-301, April 1965.

[4] R. V. L. Hartley, "A more symmetrical Fourier analysis applied to transmission problems", *Proc. of IRE*, vol. 30, pp. 144-150, March 1942.

[5] H.S. Hou, "The Fast Hartley Transform Algorithm", *IEEE Trans. Comput.,*, vol. C-36, no.2, pp. 147-156, Feb. 1987.

[6] H.S. Hou, "Hypercube architecture for singular value decomposition and other fast transforms", *IEEE Trans. Comput.*, vol. C-36, no.2, Feb. 1987.

[7] T. Le-Ngoc and M. Tue Vo, "Implementation and Performance of The Fast Hartley Transform," *IEEE Micro*, pp. 20-27, Oct. 1989.

[8] X. Lin, T. F. Chan and W. J. Karplus, "The fast Hartley transform on the hypercube multiprocessors," in *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications* vol. I. Pasadena, CA: *Association for Computing Machinery*, Jan. 1988, pp. 1451-1454.

[9] H. V. Sorensen, D. L. Jones, C. S. Burrus and M. T. Heideman, "On computing the discrete Hartley transform," *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. ASSP-33, no. 4, Oct. 1985.

[10] J. D. Villasenor and R. N. Bracewell, "Vector Hartley transform," *Electronics Letters*, vol. 25, no. 17, pp. 1110-1111, Aug. 17, 1989,