J Sign Process Syst (2008) 53:367–382
DOI 10.1007/s11265-008-0239-x

Mapping of Discrete Cosine Transforms onto Distributed Hardware Architectures

Rafael A. Arce-Nazario · Manuel Jiménez · Domingo Rodríguez

Received: 26 April 2007 / Revised: 17 February 2008 / Accepted: 28 April 2008 / Published online: 3 June 2008

Abstract We present an algorithmically-aware, high-level partitioning methodology for discrete cosine transforms (DCT) targeted to distributed hardware architectures. The methodology relies on the exploration of alternate DCT formulations as part of the partition optimization process. To the best of our knowledge, no previously proposed DCT algorithm exists that is capable of consistently producing alternate regular formulations for an *n*-size DCT. Hence, a new Cooley-Tukey-like DCT factorization algorithm was developed to allow exploration of alternate formulations as part of the partitioning optimization process. The use of our factorization mechanism along with a greedy strategy to explore the space of equivalent DCT formulations yielded partitioning solutions with as much as 18% reduction in latency and 83% reduction in run-time as compared to previously proposed regular DCT formulations.

R. A. Arce-Nazario (✉)
Department of Physics and Electronics,
University of Puerto Rico, Humacao Campus,
Humacao 00681-4300, Puerto Rico
e-mail: rafael.arce@upr.edu

M. Jiménez · D. Rodríguez
Electrical and Computer Engineering Department,
University of Puerto Rico, Mayagüez Campus,
Mayagüez 00681-5000, Puerto Rico

M. Jiménez
e-mail: mjimenez@ece.uprm.edu

D. Rodríguez
e-mail: domingo@ece.uprm.edu

Keywords Discrete cosine transforms · Distributed hardware architecture · Partitioning methodology

1 Introduction

The discrete cosine transform (DCT) is an essential component in many current multimedia compression algorithms such as MPEG and JPEG. Most literature for the hardware implementation of the DCT concentrates on the 8-point one-dimensional and the 8×8 two-dimensional transforms, since they are widely used in multimedia applications [1, 2]. The hardware density of current FPGAs allows the implementation of small to moderate DCT sizes within a single device, while meeting some performance criteria. As with other widely used transforms, meeting performance requirements for larger DCTs frequently requires implementation onto distributed hardware architectures (DHAs), such as multi-FPGA boards. In these situations, a systematic partitioning method is needed to effectively map the DCT functionality across the devices of a distributed architecture.

Several general-purpose partitioning methodologies for DHAs have been proposed [3–5]. They commonly use discrete signal transforms (DSTs) such as the DFT and DCT as benchmarks. Nevertheless, they treat these transforms as just any other algorithm, missing the opportunity to use algorithmic level properties and particular graph features (e.g. regularity) that are typical of DSTs. We have developed a methodology called DST mapping using algorithmic and graph interaction and computation (DMAGIC) for partitioning DSTs to

DHAs that uses DST features to improve exploration time and partitioning results. DST-features are introduced in two abstraction levels as part of our methodology: at the graph level and the algorithmic-level. At the graph partitioning level, a series of DST-specific structural considerations are proposed and evaluated to improve the graph partitioning heuristic. At the algorithm-level, an equivalent formulation exploration is conducted in search of formulations that are more suitable to the target topology. Partitioning schemes based on algorithmic-level reformulations have been successfully applied on the Fast Fourier Transform (FFT), using the Cooley-Tukey (CT) factorization rule to obtain alternate formulations [6]. These exploit the regularity of FFTs to devise efficient partitioning schemes. These methods, however, do not necessarily apply to the DCT due to its less regular structure and the lack of CT-like rules to generate regular formulations.

In order to extend formulation-exploration strategies to DCTs, a new systematic way of obtaining regular DCT formulations must be devised. To this end, we studied several regular DCT formulations and analyzed their potential for distributed implementation. This analysis resulted in the development of a factorization of Nikara's perfect shuffle DCT (NPS-DCT) formulation [1] that can be used to explore alternate DCT algorithms. The use of this new factorization allowed DMAGIC to obtain results with over 21% latency improvement as compared to the best among the previously existing formulations.

In this paper, we present an overview of the DMAGIC methodology, highlighting the considerations that make it effective for the partitioning of DSTs to DHAs. The need for a regular factorization DCT scheme is justified within the context of the methodology. Then, the studied regular DCT formulations are reviewed with emphasis on their advantages for distributed implementation. This is followed by the derivation of the new regular factorization method. Finally, the results of using the new factorization method within the DMAGIC methodology are compared against existing DCT regular formulations.

2 A Generalized DST Partitioning Methodology

Figure 1 shows a conceptual map of the proposed partitioning methodology, called DMAGIC. The inputs on the top are a DST specified as a Kronecker Products Algebra formulation, parameterized at least by the resolution of its points, and a high-level specification of the target architecture, which includes the number and logic capacity of the devices and their connection topology. The *Kronecker to Graph* process converts the algorithmic formulation into a dataflow graph (DFG) whose nodes denote functional primitives, i.e. small computational components that are common throughout the formulation and have been identified as efficient procedures on the target devices. The DFG is partitioned using a deterministic graph partitioning/placement heuristic that has been enhanced to handle DST structures. The quality of results from the partition/placement (P/P) process is used by a heuristic formulation-exploration engine to guide exploration onto further formulations. Based on the partitioning results of the current formulation, factorization rules are used to generate a new formulation, which, hopefully, improves the previous results. The conversion/partitioning/reformulation process continues until no considerable improvements are detected, at which point the methodology outputs the best P/P scheme encountered obtained throughout the exploration.

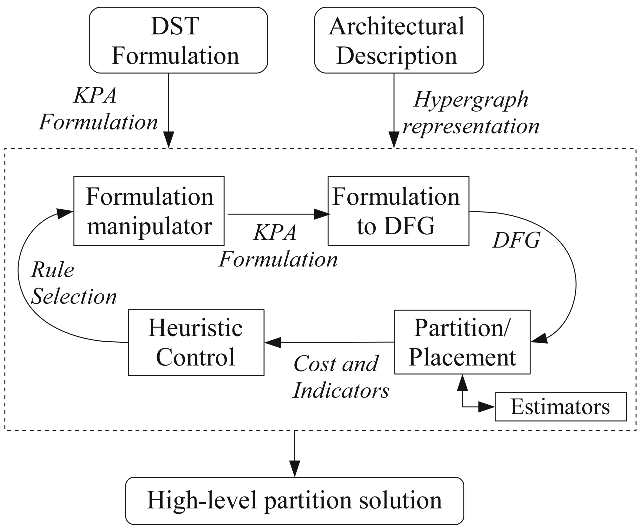


Figure 1 Block diagram for the DMAGIC methodology.

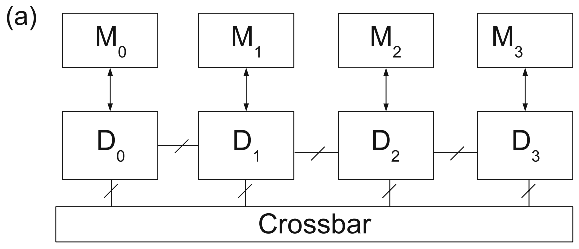
Two main aspects distinguish the proposed methodology as being specially oriented for DSTs. First, a series of DST-structure aware considerations have been incorporated into the partitioning/placement heuristic. These considerations help the partitioning/placement heuristic to conduct a faster exploration and maintain the regularity of the original expression, thus obtaining results that can be efficiently mapped onto hardware structures [6, 7]. Second, the methodology uses rules specific to the particular DST at hand to conduct an exploration of alternate formulations that might be more suitable for partitioning to the given topology. In this sense, DSTs have an advantage over other algorithms, since they a have considerable amount of properties to be used for such purposes.

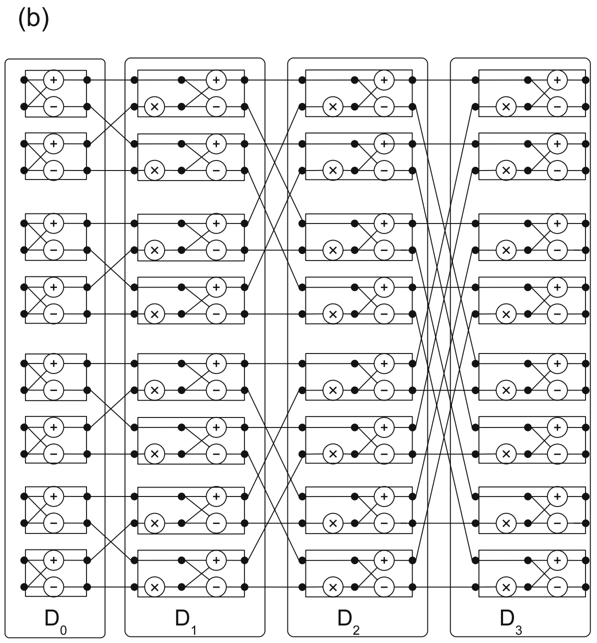
2.1 Graph P/P Process

DMAGIC's P/P algorithm, KL-H, is a *k*-way partition heuristic for heterogeneous-channel topologies [6]. Given that communication channels typically represent the most constrained resource in DHAs, the objective of this algorithm is to best distribute communications among the available channels, while minimizing the amount of communications. KL-H complements the basic Kernighan-Lin bipartition heuristic with several considerations derived from DST flow-graph characteristics, in an effort to improve optimization convergence and solution quality. The following are the three fundamental considerations:

1) **Linear partitioning**: Fast DST algorithms have traditionally been partitioned vertically or horizontally because of the regularity of their dataflow and inter-stage data dependence [2, 8]. *Vertical* partitioning maps computation as in architectural-level pipelines, where one or more complete DST computational stages are assigned to each hardware device. In *horizontal* partitioning, each device carries out all stages of computation for a data subset, similar to single-instruction-multiple-data (SIMD) processing. A pipeline implemented onto a DHA requires all data points to cross all inter-device communication channels. On the other hand, in a SIMD scheme, even a naïve linear partitioning can reduce data communications requirements in half. These two situations are illustrated in Fig. 2.

For the sake of simplicity, assume that a 16-point FFT is being partitioned to an architecture consisting of 4 FPGAs connected in a linear array topology with width of 1 data point and with a crossbar serving as a common connection, as shown in Fig. 2a. Assume also that each FPGA is capable





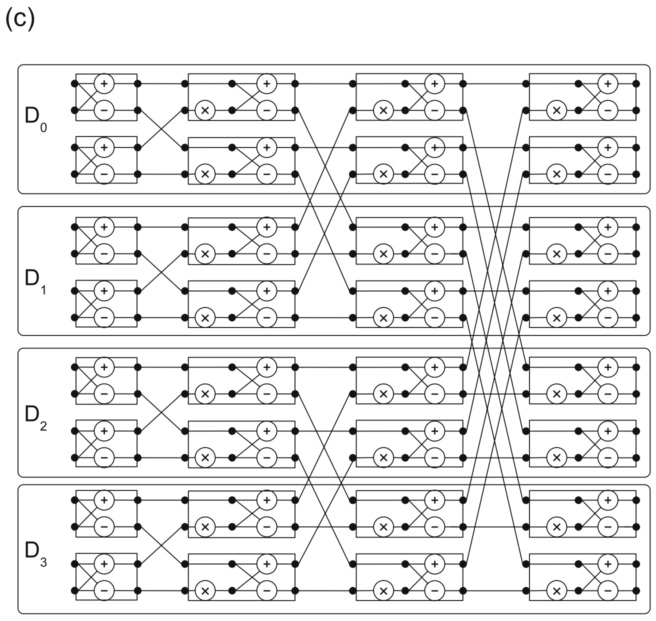


Figure 2 Comparison of vertical and horizontal partition schemes. **a** Target architecture. **b** Vertical partition scheme. **c** Horizontal partition scheme.

Algorithm 1 Produce linear horizontal partitions.

```
Input: DFG $G = (V, E, f_L)$ , $k$: the number of partitions
Output: Horizontal linear partition $\{Q_0, Q_1, .., Q_{k-1}\}$, where $Q_i \in V$ and $Q_0 \cup Q_1 \cup \ldots \cup Q_{k-1} = V$
1.  $Q_0 \leftarrow \emptyset, Q_1 \leftarrow \emptyset, \ldots Q_{k-1} \leftarrow \emptyset$
2.  For every computational column $C \in V$
    2.1. Sort $v \in C$ in order of increasing node levels $(f_L(v))$.
    2.2. $W_{CC} = \sum_{c \in C} w(c)$
    2.3. Determine indexes $(p_0, \ldots, p_{k-2})$ such that $\sum_{i=0}^{p_0} w(c_i) \approx \sum_{i=p_0+1}^{p_1} w(c_i) \approx \ldots \approx \sum_{i=p_{k-2}+1}^{|C|-1} w(c_i) \approx \frac{W_{CC}}{k}$
    2.4. $Q_0 = Q_0 \cup \bigcup_{i=0}^{p_0} c_i, \quad Q_1 = Q_1 \cup \bigcup_{i=p_0+1}^{p_1} c_i, \quad \ldots,$
         $Q_{k-1} = Q_{k-1} \cup \bigcup_{i=p_{k-2}+1}^{|C|-1} c_i$
3.  End For
```

of implementing one butterfly-multiplication (BM) kernel and the necessary permutation logic. A vertical partition, illustrated in Fig. 2b will vertically fold each computational column into the instanced BM kernel. A horizontal partition will fold the two rows assigned to each device into an instanced BM kernel, as illustrated in Fig. 2c. In each case, the instanced kernel will perform the same amount of computations (i.e. 8 BM computations) however; the performance bottleneck will be the communication channels. In the vertical partitioning scheme, every data point will pass through each channel. Meanwhile, in the horizontal partitioning, some of the data points do not need to cross between partitions during the computation. In the example, the vertical partitioning scheme will have a latency of 30 cycles, and will complete each subsequent data set every 16 cycles. The horizontal partitioning scheme (as is, without optimization) will have a latency of 25 cycles, and will complete further data sets every 16 cycles. Swapping the partitions of several BMs in the last column results in a horizontal partitioning with a latency of 18 cycles and completed data sets every 8 cycles. For the type of architecture being targeted, horizontal partitioning is expected to provide lower latencies than the vertical scheme, as it shall significantly reduce data communications. For this reason KL-H limits its partition search to horizontal linear partition solutions.

2) **Balanced initial partitions**: The concept of balanced linear horizontal partitions is illustrated in Fig. 2c. The initial linear horizontal partitions are obtained by using Algorithm 1. The algorithm is given a DST's data-flow graph, where each node is annotated with its *level*; a positive integer that indicates the node's level within the computational topology. For example, a matrix $\left[(I_2 \otimes A_3) \oplus (I_3 \otimes B_2)\right]$ would have two A_3 nodes with levels 0 and 1, and three B_2 nodes with levels 2, 3, and 4. In essence, the initial partition scheme is obtained by dividing the structure horizontally into k equally weighted partitions. Common DST formulations have corresponding DFGs which cluster highly interconnected subgraphs in such a way that a balanced linear horizontal partition represents a good solution. For this reason, in KL-H uses this method rather than randomly obtained initial solutions. When used in KL-H to partition DSTs, balanced initial partitions resulted in reductions of up to 31% on cutsize and up to 67% in iterations vs. average results with random initial partitionings. Reductions in iterations are specially important considering that the graph partitioning heuristic is called upon numerous times throughout the DMAGIC partitioning process.

3) **Schedule compactness**; A common graph structure found in fast DST algorithms is the butterfly network (BN) [9]. Regular DCT formulations commonly contain BN structures as part of their DFGs [1, 2, 10]. Schedule-wise, the BN is a completely rigid structure since all paths of its computational stages are isometric and a delay in the computation of any node means a delay to the completion of processing as a whole. KL-H takes this into account as part of the partitioning process by only considering solutions that exchange nodes from the same computational stage. This consideration entails a more focused partition solution exploration since it limits the number of possible node swaps during each KL-H iteration. DST partitioning experiments were used to confirm the effect of this consideration on partition quality and run time. Run time is reduced by as much as 87% when stage-constrained swapping is used, as compared to non-constrained swapping. The effect on latency is negligible; being less than 1% improvement on average.

Our P/P process estimates solution latency using a cost function which measures the impact of communications. The cost of a solution is represented by a vector:

$$P = \langle \Phi_0, \Phi_1, \ldots \Phi_{M-1} \rangle \,, \tag{1}$$

where Φ_i is the cost of communications through channel i. Let P_1 and P_2 be costs, we say $P_1 < P_2$ if the non-increasing ordering of P_1 is lexicographically smaller than the ordering of P_2. For example, $P_1 = \langle 1, 3, 1, 4 \rangle$ is smaller than $P_2 = \langle 0, 2, 3, 4 \rangle$, since, lexicographically 4,3,1,1 is smaller than 4,3,2,0. The communication cost Φ_i is obtained as:

$$\Phi_i = W(c_i) \sum_{e \in E} R(e, c_i), \tag{2}$$

where the *weight* $W(c_i)$ of a channel is a function $W : C \to \mathbb{Z}^+$, whose value is a relative measure of the impact on system latency of communicating a data point through c.

The communication flag $R(e, c_i)$ of an edge e through a channel c_i determines if the communication represented by e will be done through c_i.

$$R(e, c_i) = \begin{cases} 1 & \text{if } c_i = \mu(e), \\ 0 & \text{else.} \end{cases} \tag{3}$$

Let $a(u)$ and $a(v)$ represent the partitions to which nodes u and v have been assigned, respectively. Let J_e, where $e = \langle u, v \rangle$, be the set of channels that can be used to communicate data from $a(u)$ to $a(v)$. The minimum weight channel $\mu(e)$ is the channel $c_i \in J_e$ with the minimum weight.

2.2 Formulation Exploration Process

DMAGIC uses algorithmic-level rules to explore alternative formulations of a DST. This is done with the objective of finding a formulation whose structural features can be better exploited by the partitioning methodology for the given target topology. Since the number of equivalent formulations grows exponentially with DST size, exhaustive search of the formulation space is impractical. For instance, the SPIRAL code generation program can consider a total of 1,639,236,012 formulations for a size-32 DCT [11]. Clearly, a more effective search of the equivalent-formulation space is needed.

To develop the DMAGIC's formulation-exploration heuristic, we began by assessing the impact of different formulation-level rules on partitioning quality. Initial assessment experiments focused mostly on partitioning FFTs due to the availability of algebraic transformations that systematically generate equivalent, yet structurally diverse formulations, e.g. CT factorization and permutation rules. Of the evaluated transformations, the decomposition strategies obtained using the Cooley Tukey factorization rule (CTFR) had the most evident and predictable impact on partitioning quality [6]. Salient observations from the assessment were incorporated into an algorithm for greedy exploration of the formulation space using CTFR decomposition. This algorithm obtained improved results in terms of implementation latency and exploration time as compared to other general-purpose high-level partitioning methods.

CTFR has two essential features that allow its use for breakdown strategy exploration. First, it is capable of decomposing a size $n = mp$ FFT into the combination of *arbitrary* sized FFTs sized m and p. This allows the generation of multiple breakdown strategies, each corresponding to a unique formulation and having characteristics that enable its improved partitioning for a given DHA. Second, regardless of the decomposition factors, the resulting formulation can still be implemented using the same basic functional primitives as the original, i.e. size-2 butterfly-twiddles. Functional primitive regularity is essential for effective hardware implementation since it simplifies control and allows for a more effective use of hardware resources for computation.

Fast DCT formulations have a structure that resembles that of the FFT, with somewhat less regularity. In order to perform formulation exploration for the DCT, a method is needed which can systematically yield different, yet equivalent formulations, without loss of structure regularity. Several regular fast DCT algorithms have been reported over the years, yet none of them inherently comply with both of the features that make CTFR desirable for FFTs. Püschel, et al. proposed several CT like algorithms for the DCT which factor a DCT size n into a combination of size-m and size-p terms [12]. However, the resulting structures do not naturally encourage hardware implementation, since structural irregularities are introduced. Other algorithms, such as those reported by Wang, Takala, Hsiao, and Nikara imply effective hardware structures, but lack the arbitrary decomposition capability [1, 2, 10, 13]. As proposed, each of these algorithms provides a single formulation for each DCT size, which highly limits their use in a formulation-exploration environment.

As part of the search for a regular CT-like decomposition algorithm for DCTs, several previously proposed regular DCT formulations were studied, as well as their suitability for distributed implementation. Of the studied formulations, NPS-DCT structure allowed us to derive a decomposition scheme that can systematically decompose a DCT into arbitrarily-sized functional primitives, while maintaining regularity. This scheme significantly broadens the DCT formulation space, and allows DMAGIC to find improved partitioning solutions.

3 DCT Regular Algorithms

The N-point 1-D DCT type-II transform matrix is defined as:

$$\left[\mathrm{DCT}_n^{\mathrm{II}}\right]_{mn} = \sqrt{\frac{2}{N}}\left[b_m \cos\left(\frac{m\left(n+\frac{1}{2}\right)\pi}{N}\right)\right],\ m,n = 0,1,\ldots,N-1\ , \tag{4}$$

where b_m is a scaling factor defined as:

$$b_m = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } m=0 \text{ or } m=N \\ 1, & \text{if } m \neq 0 \text{ and } m \neq N \end{cases} \tag{5}$$

The redundancies found in the DCT matrix are exploited to obtain algorithms which reduce its computational complexity. For hardware implementation, it is essential that the fast DCT algorithm not only have a reduced number of expensive operations, such as multiplications, but must also have an overall regular structure. Regular computational structures facilitate the mapping of DCT operations to the limited resources available in hardware, meanwhile keeping a simplified control structure. In general, DCT algorithms have not obtained the regularity found in CT FFT algorithms. However the regularity of some proposed DCT algorithms is enough to implement efficient hardware pipeline structures that meet performance requirements [1, 2]. The following sections discuss the four candidate formulations that were evaluated for their potential use within the DMAGIC methodology. Section 3.1 presents Puščhel's CT-like DCT algorithm, which allows arbitrary decomposition at the expense of structural irregularities. Sections 3.2, 3.3, and 3.4 discuss three regular fast DCT algorithms, emphasizing their suitability for distributed implementation.

3.1 Püschel, et al. CT-Like DCT Algorithms

Püschel, et al. reported several algorithms where DCTs of size $N = M \cdot P$ can be synthesized as the composition of DCTs size M and P along with interfacing permutations and additions. Equation 6 shows one of the proposed algorithms for the DCT-II.

$$\mathrm{DCT}_n^{\mathrm{II}}(r\pi) = C_{n,k} L_{n,k}\left(I_m \otimes \mathrm{DCT}_k^{\mathrm{II}}(r\pi)\right) \times \left(\bigoplus_{0\le i<k} \mathrm{DCT}_m^{\mathrm{II}}(r_i\pi)\right)^{L_{n,k}} R_{n,m} \tag{6}$$

where $L_{n,k}$ is a size-n stride-k stride permutation matrix,

$$C_{n,m} = \begin{bmatrix} I_k & Z_k & & \\ & I_k & \ddots & \\ & & \ddots & Z_k \\ & & & I_k \end{bmatrix}, \tag{7}$$

$$Z_n = \begin{bmatrix} & & & 0 \\ & & 0 & 1 \\ & \iddots & \iddots & \\ 0 & 1 & & \end{bmatrix}, \tag{8}$$

and

$$R_{n,m} = (I_k \oplus J_k \oplus I_k \oplus J_k \oplus \ldots)\ . \tag{9}$$

Here J_k is I_k with the order of the columns reversed.

In particular,

$$DCT_2^{(II)}(r_i\pi) = \mathrm{diag}\left(1, {}^{1}\!/\!\left(2\cos\left({}^{r_i\pi}\!/\!_{2}\right)\right)\right) \cdot F_2 \tag{10}$$

$$F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{11}$$

The $DCT_n^{II}(r_i\pi)$ term is called a *skew* DCT of type II, where r_i has an effect on the multiplication coefficients but not on the structure of the DCT flowgraph. The essential limitation of this algorithm when targeting a distributed implementation resides in matrix C. This matrix represents a varying number of additions and permutations depending on its indexes.

Figure 3 shows the DFGs of the two alternate formulations that can be generated by Eq. 6 for $n = 8$. Repetitive functional primitives can be seen in each of the graphs. However, since different C matrices may be used throughout each decomposition; the functional primitives will differ from stage to stage, making difficult its hardware implementation through uniform modules.

Experimental results using our partitioning heuristics revealed latency increases of up to 100% when comparing formulations containing heterogenous C matrices to formulations using a unique C matrix. Thus,

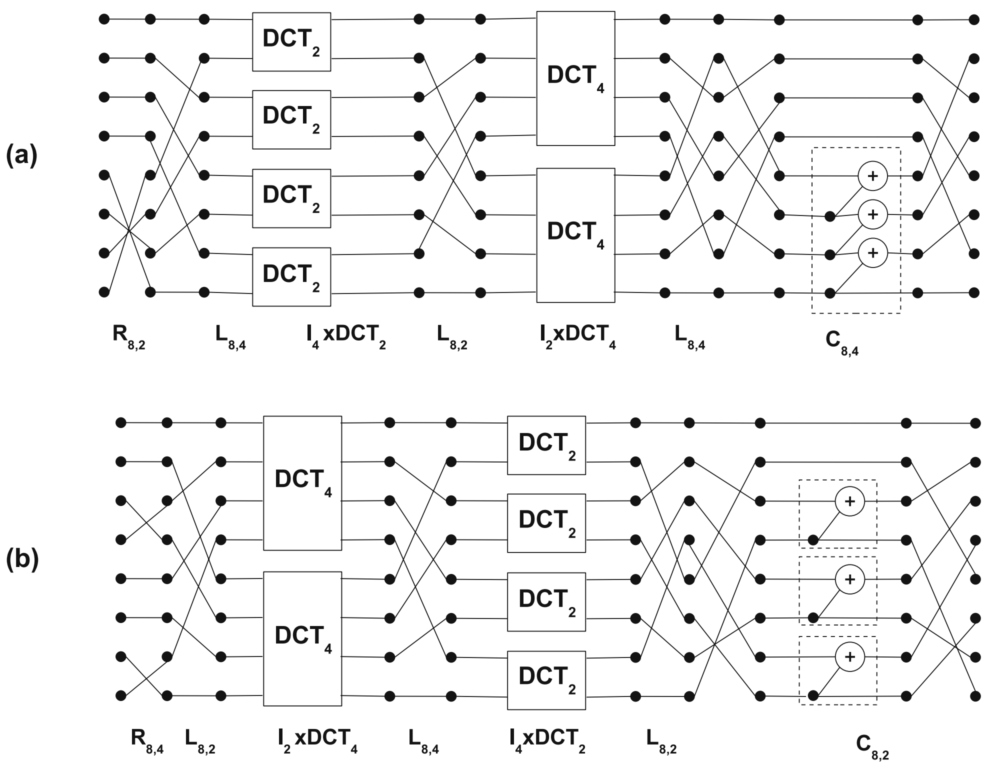


Figure 3 DFGs for the two alternate formulations generated by Eq. 6 for n=8 (**a**, **b**).

practical instances for hardware implementations using Eq. 6 could be obtained as factorizations that produce C matrices of the same type throughout all stages. This represents a very limited set of formulations. For example, Fig. 4 shows the split trees of the DCT formulations for size 64.

3.2 Hsiao and Tseng's DCT Algorithm

Hsiao and Tseng reported a 1-D DCT decomposition algorithm (HT-DCT) that computes a size $N = 2^n$ transform as n stages of a butterfly-with-multiplication (BM) stages followed by $n - 1$ post processing (PP) stages, where some of the results from the BM stages are added [2]. Figure 5 shows a DFG for the type-II formulation of an 8-point 1D HT-DCT. The regularity obtained throughout both stages allowed its implementation as resource-efficient VLSI pipeline consisting of variations of two kernels: one to implement each of the butterfly stages and another for each of the post-processing stages. The explicit separations into two stages that perform distinct operations make it ineffective to merge both functionalities into one functional primitive, since this would lead to a rather non-regular structure, as shown in Fig. 6. This discourages the development of an arbitrary factorization scheme based on HT-DCT.

3.3 Morikawa's Simple Structured Fast DCT Algorithm

As part of the development for his Pruning DCT algorithm, Wang presented Morikawa's simple structured fast DCT (SS-FCT) algorithm [10]. SS-FCT is similar to HT-DCT in that it involves both butterfly-multiply and adding structures. However, as shown in Fig. 7, SS-DCT

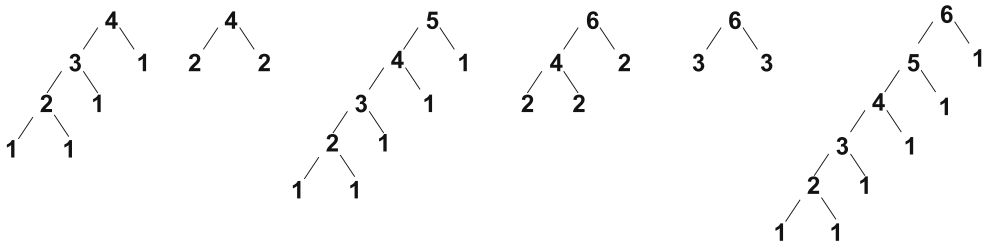


Figure 4 Practical split trees for 16, 32 and 64-point DCT when using Eq. 6 for hardware implementation.

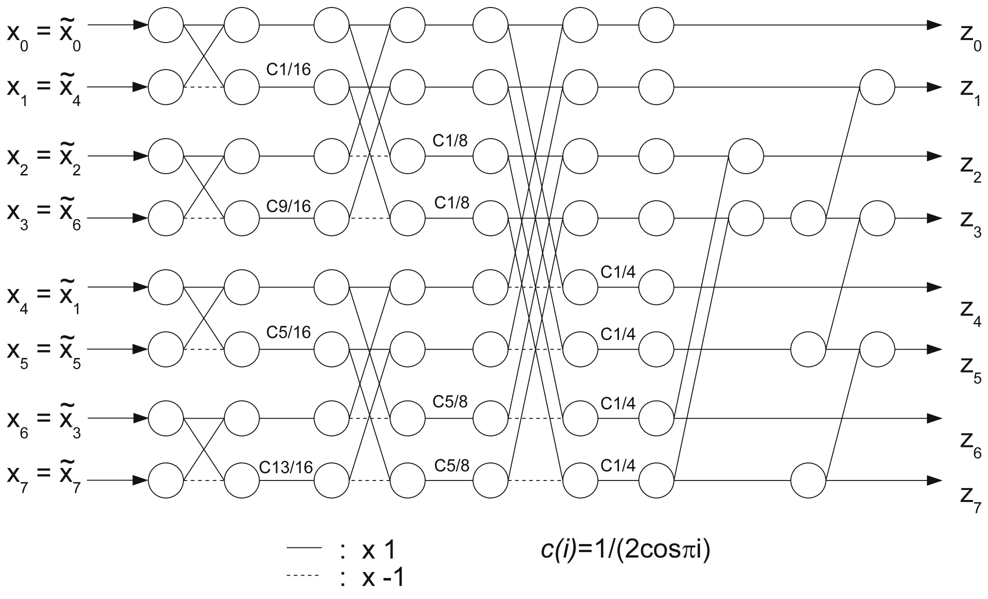


Figure 5 8-point HT-DCT [2].

intermixes the adding structures and with the butterfly-network (BN) structures, making it more feasible to utilize a merged BM/Add functional primitive. Figure 8 shows a DFG for an 8-point WP-DCT using a unified functional primitive. Note that if all the successive BM and add stages were like in the second BN column we could implement a size-2 common-data integrated functional primitive, i.e. a functional primitive that would perform BM followed by addition on the same two points. However, this doesn't happen throughout the rest of the structure, as BMs are followed by permutations. This eliminates the practicality of integrating BM and adds as common-data unified functional primitive. Thus, we have a $2n - 1$ computational column structure, as shown in Fig. 8. Once again, the lack of regularity and the explicit separation of functionalities limit SS-FCT chances for being factorized in a CT-like manner.

3.4 NPS-DCT Algorithm

NPS-DCT algorithm has several features that facilitate its pipelined hardware implementation [1]. Figure 9 shows a DFG of an 8-point NPS-DCT. First, it is almost perfectly regular across each of its computational columns, an essential characteristic to vertically fold its columns into a pipeline. Second, irregularities have been distributed across the computation in such manner that they operate on the same data sets as the previous BM structure. Third, data permutations between successive computational columns are kept to varying sizes of perfect-shuffle permutations, for which efficient pipeline structures have been proposed [14]. The only feature missing for NPS-DCT to be fully amenable to our partitioning methodology is the capability of generating multiple formulations for a given DCT size. From the NPS-DCT DFG in Fig. 9, we can

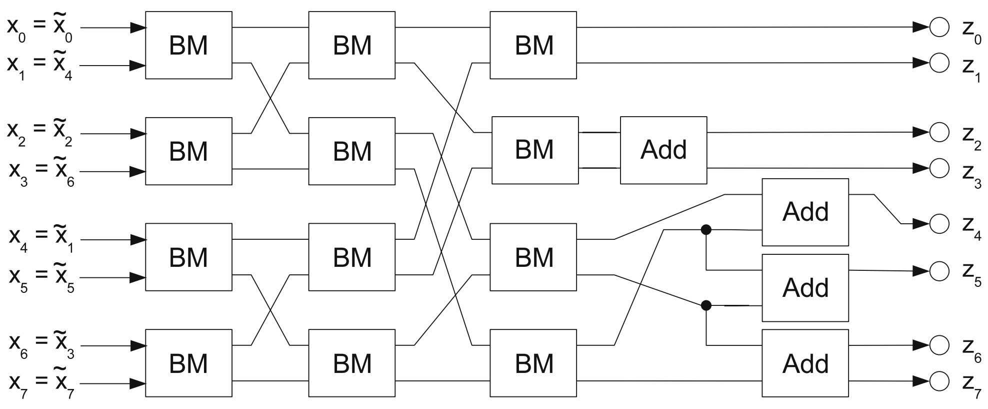


Figure 6 8-point HT-DCT using a single functional primitive that performs both the BM and add (post-processing) functionalities.

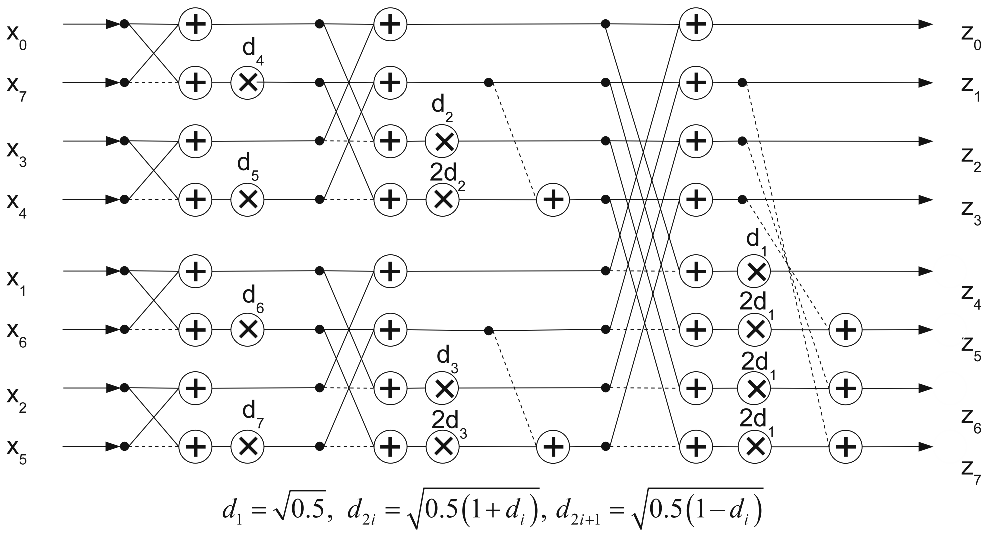

x0
x7
x3
x4
x1
x6
x2
x5
d4
d5
d6
d7
d2
2d2
d3
2d3
d1
2d1
2d1
2d1
z0
z1
z2
z3
z4
z5
z6
z7
$d_1 = \sqrt{0.5},\ d_{2i} = \sqrt{0.5(1+d_i)},\ d_{2i+1} = \sqrt{0.5(1-d_i)}$

Figure 7 8-point SS-FCT [10].

begin to identify a functional primitive that is common throughout the complete PS-DCT structure (identified in Fig. 9 by dashed boxes). This serves as basis for the development of an arbitrary clustering/factorization technique, described in the next section.

4 CT-Like Decomposition for NPS-DCT

We have derived a functional primitive-equivalent CT-like factorization scheme based on NPS-DCT algorithm. Section 4.1 summarizes the basic stride permutation rules used throughout the derivation, while Section 4.2 details the derivation itself.

4.1 Stride-Permutation Theorems

A fast version of a DST typically consists of a series of computational columns with data point stride permutations in between. Stride permutations can be factorized/combined to form smaller/larger permutations that can expose opportunities for a more effective implementation of a DST to a given computational architecture. For instance, the Pease FFT formulation, which can be obtained from the original CT formulation by using permutation decomposition rules, is well suited for hardware implementation as its DFG easily folds both vertically and horizontally [8]. The following theorems were used as part of the derivation to combine and reformulate stride permutations. Proofs for these theorems and corollaries can be found in the works by Nikara and Takala [1, 14].

Theorem 1 *Factorizations of stride permutations*

Let $L_{m,n}$ be a size-m stride-n permutation matrix and I_m a size-m identity matrix. Then,

$$L_{a,bc} = L_{a,b} L_{a,c} \tag{12}$$

$$L_{abc,c} = \left(L_{ac,c} \otimes I_b\right)\left(I_a \otimes L_{bc,c}\right) \tag{13}$$

$$L^T_{abc,c} = L_{abc,ab} = \left(I_a \otimes L_{bc,b}\right)\left(L_{ac,a} \otimes I_b\right) \tag{14}$$

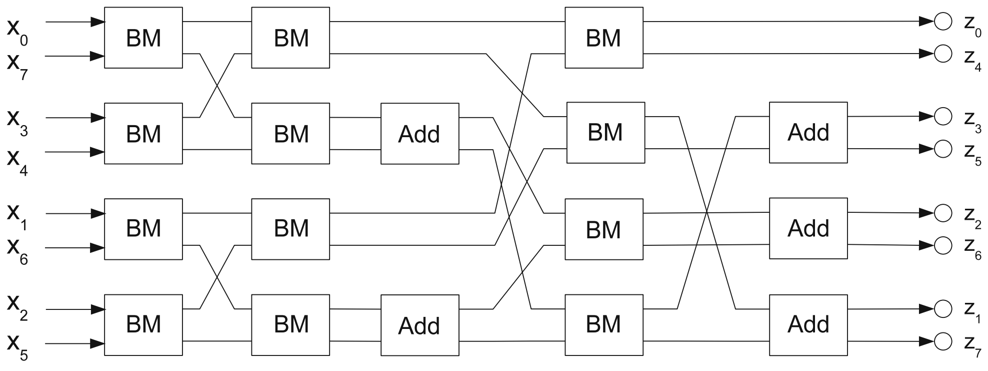
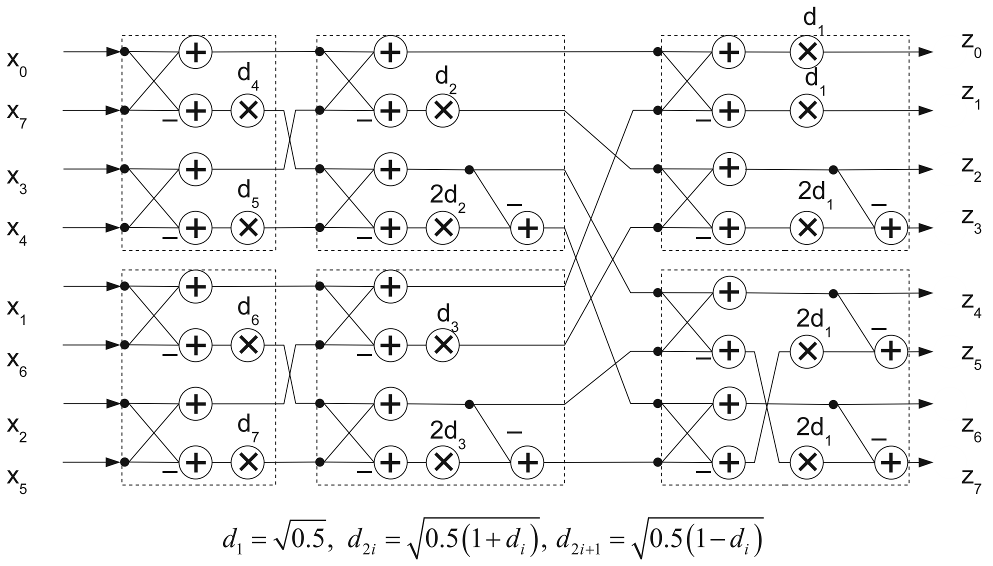


Figure 8 8-point SS-FCT using functional primitive blocks.



x0
x7
x3
x4
x1
x6
x2
x5
d4
d5
d6
d7
d2
2d2
d3
2d3
d1
d1
2d1
2d1
2d1
z0
z1
z2
z3
z4
z5
z6
z7
$d_1 = \sqrt{0.5},\ d_{2i} = \sqrt{0.5(1+d_i)},\ d_{2i+1} = \sqrt{0.5(1-d_i)}$

Figure 9 DFG for an 8-point NPS-DCT.

Theorem 2 *Commutative property for tensor products. Let A be $a \times a$ and B be $b \times b$ matrices. Then,*

$$(A \otimes B) = L_{ab,a} (B \otimes A) L_{ab,b} \tag{15}$$

Corollary 1 *Factorization of a stride-2 permutation.*

$$L_{2^{n+1},2} = \prod_{q=n-1}^{0} \left(I_{2^{n-q-1}} \otimes L_{4,2} \otimes I_{2^q}\right) \tag{16}$$

Corollary 2 *Factorization of $L_{2^{n+K},2^{n+1}}$*

$$L_{2^{n+K},2^{n+1}} = \prod_{q=0}^{n} \left(I_{2^{n-q}} \otimes L_{2^K,2} \otimes I_{2^q}\right) \tag{17}$$

4.2 CT-Like Formulation for the DCT

NPS-DCT is formulated as a product of sparse matrices using Kronecker Algebra operators. Let $C_{2^n}^{II}$ be a 2^n-point DCT type-II formulation,

$$C_{2^n}^{II} = \sqrt{\frac{2}{2^n}} U_{2^n}^{(n-1)} \prod_{s=n-1}^{1} \left[A_{2^n}^{(s)} \left(I_{2^{n-s-1}} \otimes L_{2^{s+1},2^2}\right)\right] A_{2^n}^{(0)} P_{2^k}^{H}, \tag{18}$$

where

$$A_N^{(s)} = M_N^{(s)} D_N^{(s)} H_N^{(s)} F_N, \tag{19}$$

$$M_{2^k}^{(s)} = \bigoplus_{i=0}^{2^{k-1}-1} \begin{pmatrix} 1 & 0 \\ -\mu_s(i) & 1 \end{pmatrix}, \tag{20}$$

$$D_{2^k}^{(s)} = \operatorname{diag}\left(g_k(i,s)\right),\ i = 0, 1, ..., 2^k - 1, \tag{21}$$

$$g_k(i,s) = \left(2^{\mu_s(\lfloor i/2 \rfloor)} d\left(2^{k-s-1} + \lfloor i/2^{s+1} \rfloor\right)\right)^{f_k(i,s)}, \tag{22}$$

$$f_k(i,s) = (i \bmod 2) + (1 - \tau_0(i))(1 - \tau_{k-1}(s)), \tag{23}$$

$$\tau_i(s) = \begin{cases} 0, & s = i \\ 1, & s \neq i \end{cases}, \tag{24}$$

$$H_{2^k}^{(s)} = \bigoplus_{i=0}^{2^{k-2}-1} (Q_4 R_4 Q_4)^{\mu_{s-1}(i)}, \tag{25}$$

$$F_{2^k} = I_{2^{k-1}} \otimes F_2, \tag{26}$$

and $P_{2^k}^{H}$ is a Hadamard permutation matrix of order N. To allow us to concentrate on the main computational components, we rewrite Eq. 18 as follows:

$$C_{2^n}^{II} = \sqrt{\frac{2}{2^n}} U_{2^n}^{(n-1)} \Gamma_{2^n} P_{2^k}^{H} \tag{27}$$

where

$$\Gamma_{2^n} = \prod_{s=n-1}^{1} \left[A_{2^n}^{(s)} \left(I_{2^{n-s-1}} \otimes L_{2^{s+1},2^s}\right)\right] A_{2^n}^{(0)}, \tag{28}$$

As evidenced by Eq. 28, actual arithmetic operations are performed by the $A_n^{(s)}$ terms. Upon closer examination, it was noticed that the sparse matrices involved in the computation of these $A_n^{(s)}$ terms are all composed of kernels operating on 2 or 4 points. Thus, a complete formulation can be represented in terms of 4-input functional primitives, as illustrated by the dashed lines in Fig. 9. Furthermore, these 4-input functional primitives can be systematically combined into clusters with $4p$ inputs. If we define the operation performed on each 4-input group to be a non-divisible functional primitive Φ_4, we can rewrite a Nikara's formulation as follows:

$$\Gamma_{2^n} = \prod_{s=n-2}^{1} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] \times (I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-2}} \otimes L_{4,2} \right) (I_{2^{n-2}} \otimes \Phi_4) \quad (29)$$

In other words, PSDA consists of n processing columns, each consisting of 2^{n-2} Φ_4 components interconnected using the perfect-shuffle sequence. Each of these Φ_4 components encapsulates the variabilities introduced by the parameterized A_n terms. Thus, if we envision each of Φ_4 kernels as able to contain the necessary hardware to implement the minor computational differences introduced by the parameterized A_n terms, Eqs. 28 and 29 are structurally equivalent.

Let us define Φ_{2^n} as:

$$\Phi_{2^n} = \prod_{s=n-2}^{1} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] (I_{2^{n-2}} \otimes \Phi_4) \quad (30)$$

Equation 30 can be split into three products of length $m-1$, 1, and $k-1$, where $n = m+k+1$:

$$\Phi_{2^n} = \prod_{s=n-2}^{n-m} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] \times (I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-k-2}} \otimes L_{2^{k+2},2^{k+1}} \right) \times \prod_{s=k-1}^{1} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] \times (I_{2^{n-2}} \otimes \Phi_4) \quad (31)$$

For discussion purposes, Eq. 31 is rewritten as follows:

$$\Phi_{2^n} = A \, (I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-k-2}} \otimes L_{2^{k+2},2^{k+1}} \right) B \, , \quad (32)$$

where

$$A = \prod_{s=n-2}^{n-m} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] (I_{2^{n-2}} \otimes \Phi_4) \quad (33)$$

$$B = \prod_{s=k-1}^{1} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] (I_{2^{n-2}} \otimes \Phi_4) \, . \quad (34)$$

Since, $n-2 = m+k-1$, we factor out I_{2^m}

$$B = I_{2^m} \otimes \prod_{s=k-1}^{1} \left[(I_{2^{k-1}} \otimes \Phi_4) \left(I_{2^{k-s-1}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] \times (I_{2^{k-1}} \otimes \Phi_4) = I_{2^m} \otimes \Phi_{2^{k+1}} \quad (35)$$

Expanding the A term exposes the permutations between the computational columns (permutation terms are shown underlined):

$$A = \prod_{s=n-2}^{n-m} \left[(I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{n-s-2}} \otimes L_{2^{s+2},2^{s+1}} \right) \right] = (I_{2^{n-2}} \otimes \Phi_4) \left(L_{2^n,2^{n-1}} \right) (I_{2^{n-2}} \otimes \Phi_4) \times \underline{\left(I_{2^{n-2}} \otimes L_{2^{n-1},2^{n-2}} \right)} \ldots (I_{2^{n-2}} \otimes \Phi_4) \times \underline{\left(I_{2^{m-2}} \otimes L_{2^{n-m+2},2^{n-m+1}} \right)} \quad (36)$$

The underlined terms in Eq. 36 are expanded using the permutation property shown in Eq. 14, yielding the underlined terms in Eq. 37.

$$A = (I_{2^{n-2}} \otimes \Phi_4) \underline{\left(I_2 \otimes L_{2^{n-1},2^{n-2}} \right) \left(L_{4,2} \otimes I_{2^{n-2}} \right)} \times (I_{2^{n-2}} \otimes \Phi_4) \underline{\left(I_{2^{n-1}} \otimes L_{2^{n-2},2^{n-3}} \right) \left(I_2 \otimes L_{4,2} \otimes I_{2^{n-3}} \right)} \times \ldots (I_{2^{n-2}} \otimes \Phi_4) \underline{\left(I_{2^{m-1}} \otimes L_{2^{n-m+2},2^{n-m+1}} \right)} \times \underline{\left(I_{2^{M-1}} \otimes L_{2^{n-2},2^{n-3}} \otimes I_{2^{n-2-(m-2)}} \right)} \quad (37)$$

The $(I_X \otimes L_{4,2} \otimes I_Y))$ expressions are moved to the end of the formulation and rewritten as a multiplication series:

$$A = (I_{2^{n-2}} \otimes \Phi_4) \left(I_2 \otimes L_{2^{n-1},2^{n-2}} \right) (I_{2^{n-2}} \otimes \Phi_4) \times \left(I_{2^2} \otimes L_{2^{n-2},2^{n-3}} \right) \ldots (I_{2^{n-2}} \otimes \Phi_4) \times \left(I_{2^{m-1}} \otimes L_{2^{n-m+2},2^{n-m+1}} \right) \times \prod_{q=m-2}^{0} \left(I_{2^{m-2-q}} \otimes L_{4,2} \otimes I_{2^q} \right) \otimes I_{2^{n-m}} \quad (38)$$

Then, using Corollary 1:

$$A = (I_{2^{n-2}} \otimes \Phi_4) \left(I_2 \otimes L_{2^{n-1},2^{n-2}} \right) (I_{2^{n-2}} \otimes \Phi_4) \times \left(I_{2^2} \otimes L_{2^{n-2},2^{n-3}} \right) \ldots (I_{2^{n-2}} \otimes \Phi_4) \times \left(I_{2^{m-1}} \otimes L_{2^{n-m+2},2^{n-m+1}} \right) L_{2^m,2} \otimes I_{2^{n-m}} \quad (39)$$

The same procedure is repeated, yielding the underlined term:

$$A = (I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^2} \otimes L_{2^{n-2},2^{n-3}} \right) (I_{2^{n-2}} \otimes \Phi_4) \times \left(I_{2^3} \otimes L_{2^{n-3},2^{n-4}} \right) \ldots \times (I_{2^{n-2}} \otimes \Phi_4) \left(I_{2^{m-2}} \otimes L_{2^{n-m+1},2^{n-m}} \right) \times \left(\underline{I_2 \otimes L_{2^m,2} \otimes I_{2^{n-m-1}}} \right) \left(L_{2^m,2} \otimes I_{2^{n-m}} \right) \quad (40)$$

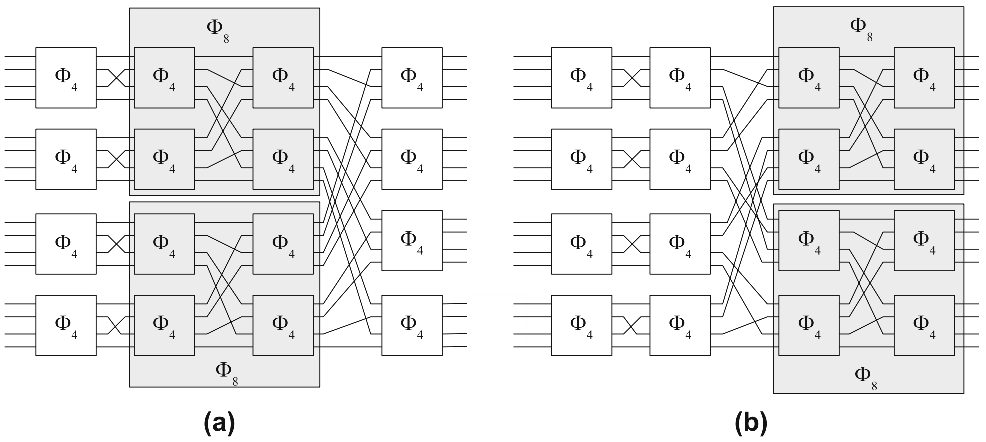


Figure 10 DFG for Eqs. 46 and 47.

After k iterations of this procedure, exhibited in Eqs. 36–40, another multiplication series is obtained:

$$\begin{aligned} A = {} & (I_{2^{n-2}} \otimes \Phi_4)\left(I_{2^k} \otimes L_{2^{n-k},2^{n-k-1}}\right) \\ & \times (I_{2^{n-2}} \otimes \Phi_4)\left(I_{2^{k+1}} \otimes L_{2^{n-(k+1)},2^{n-(k+2)}}\right) \ldots \\ & \times (I_{2^{n-2}} \otimes \Phi_4)\left(I_{2^{k+m-2}} \otimes L_{2^{n-m+1},2^{n-m}}\right) \\ & \times \prod_{q=0}^{k-1}\left(I_{2^{k-1-q}} \otimes L_{2^m,2} \otimes I_{2^q}\right) \otimes I_{2^2} \end{aligned} \tag{41}$$

Corollary 2 is used, yielding:

$$\begin{aligned} A = {} & I_{2^k} \otimes \prod_{s=m-1}^{0}\left[(I_{2^{m-1}} \otimes \Phi_4)\left(I_{2^{m-1-s}} \otimes L_{2^{s+2},2^{s+1}}\right)\right] \\ & \times \left(L_{2^{m+k-1},2^k} \otimes I_{2^2}\right) \end{aligned} \tag{42}$$

Since $m + k - 1 = n - 2$, we finally obtain:

$$\begin{aligned} A = {} & I_{2^k} \otimes \prod_{s=m-1}^{0}\left[(I_{2^{m-1}} \otimes \Phi_4)\left(I_{2^{m-1-s}} \otimes L_{2^{s+2},2^{s+1}}\right)\right] \\ & \times \left(L_{2^{n-2},2^k} \otimes I_{2^2}\right) \end{aligned} \tag{43}$$

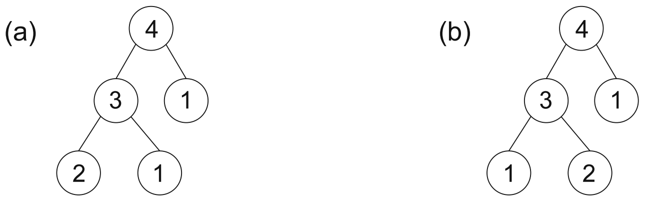


Figure 11 Split trees for Eqs. 46 and 47.

Thus,

$$\begin{aligned} \Phi_{2^n} = {} & A\,(I_{2^{n-2}} \otimes \Phi_4)\left(I_{2^{n-k-2}} \otimes L_{2^{k+2},2^{k+1}}\right) B \\ = {} & \underbrace{I_{2^k} \otimes \prod_{s=m-1}^{0}\left[(I_{2^{m-1}} \otimes \Phi_4)\left(I_{2^{m-1-s}} \otimes L_{2^{s+2},2^{s+1}}\right)\right](I_{2^{n-2}} \otimes \Phi_4)}_{I_{2^k} \otimes \Phi_{2^{m+1}}} \\ & \left(L_{2^{n-2},2^k} \otimes I_{2^2}\right)\left(I_{2^{n-k-2}} \otimes L_{2^{k+2},2^{k+1}}\right)(I_{2^m} \otimes \Phi_{2^{k+1}}) \end{aligned} \tag{44}$$

Finally,

$$\begin{aligned} \Phi_{2^n} = {} & (I_{2^k} \otimes \Phi_{2^{m+1}})\left(L_{2^{n-2},2^k} \otimes I_{2^2}\right) \\ & \times \left(I_{2^{n-k-2}} \otimes L_{2^{k+2},2^{k+1}}\right)(I_{2^m} \otimes \Phi_{2^{k+1}}) \end{aligned} \tag{45}$$

Algorithm 2 Formulation exploration based on top-down breakdown using CT-like factorization.

Input: Discrete signal transform Kronecker product expression D

Output: Optimized expression D'

```
1.  $Cost \leftarrow \infty$
2.  $D' =$ **InitialBreakdown**$(D)$;
3.  $Cost' =$ **Partition**$(D')$
4.  **While** ( $Cost' < Cost$ )
    4.1.  $Cost \leftarrow Cost'$
    4.2.  $Cost' \leftarrow \infty$
    4.3.  $H =$ **NextChildToSplit**(D')
    4.4.  **For every** split $(a, b)$ of $H$
          4.4.1.  $D\_split =$ **Split**$(D,H,(a,b))$
          4.4.2.  $Split\_Cost =$ **Partition**$(D\_split)$
          4.4.3.  **If** $(Split\_Cost < Cost')$ **Then** $Best\_D \leftarrow$
                  $D\_Split$
    4.5.  **End For**
    4.6.  $D' \leftarrow Best\_D)$;
    4.7.  $Cost' =$ **Partition**$(D')$
5.  **End While**
```

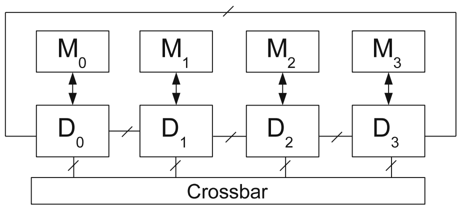


Figure 12 Target topology for experiments.

For example, the following two decompositions of DCT_{2^4} can be obtained using Eqs. 46 and 47. Figure 10a and b illustrate the two decompositions.

$$\Gamma_{2^4} = \left[(I_2 \otimes \Phi_8)\left(L_{4,2} \otimes I_4\right)\left(I_2 \otimes L_{8,4}\right)(I_4 \otimes \Phi_4)\right] \times \left(I_4 \otimes L_{4,2}\right)(I_4 \otimes \Phi_4) \qquad (46)$$

$$\Gamma_{2^4} = \left[(I_4 \otimes \Phi_4)\, L_{16,8}\,(I_2 \otimes \Phi_8)\right]\left(I_4 \otimes L_{4,2}\right)(I_4 \otimes \Phi_4) \qquad (47)$$

5 Formulation Exploration Heuristic

The formulation space obtained using Eq. 45 grows exponentially with DCT size. In order to develop a heuristic that searches the state space in a non-exhaustive yet effective manner, we began by observing the effect of formulation decomposition (with Eq. 45) on partition solution quality. Exhaustive formulations for DCTs of sizes 32 to 256 were generated, converted to DFGs, and partitioned using the KL-H graph partitioning heuristic. For analysis purposes, DCT formulations were represented as split trees since this allows for better understanding of the factorization sequence. *Split trees* are a common graphical representation of DST decomposition strategies [15]. For instance, Fig. 11a and b show the two split trees for Eqs. 46 and 47. Each node in the split tree is labeled with the log_2 of the size of the computational block (Φ) that it represents. The children of a node indicate how the node's transform is recursively computed.

As exemplified by Eqs. 46 and 47 and Fig. 10, CT-like formulation DFGs consist of computational columns of smaller computational kernels (i.e. the $(\Phi_N \otimes I_M)$ expressions) and inter-column connections (i.e. the permutation expressions). Notice that each split tree *leave* corresponds to a computational column in the DFG. Thus, once the DFG has been partitioned, information from the partition process can be related back to the split tree and used to guide further factorization of the tree. This concept is used in the formulation-exploration algorithm to decide the breakdown strategy for partitioning improvement.

Results of the exhaustive DCT formulations were analyzed in search of common decisions throughout the decomposition process which generated superior solution quality. These heuristic decisions were incorporated into a greedy formulation-exploration algorithm, shown in Algorithm 2. The algorithm starts by factoring the transform to a split tree with a distribution of children's sizes that has been observed to lead to partition-friendly formulations in smaller cases. This formulation is partitioned and its communication costs are measured. Partitioning cost information is used to determine which leaf of the current formulation to split. As was detected in the analysis of smaller transforms, an effective heuristic to guide the breakdown is to split the tree on a leaf that interfaces with the highest stage cost. The chosen leaf is split exhaustively into its children and the cost of each resulting formulation is measured. If the best cost of the split formulations is better than the current, this best-cost split formulation is kept and a new level is explored. This continues until no further improvement is obtained or when the split tree cannot be further decomposed.

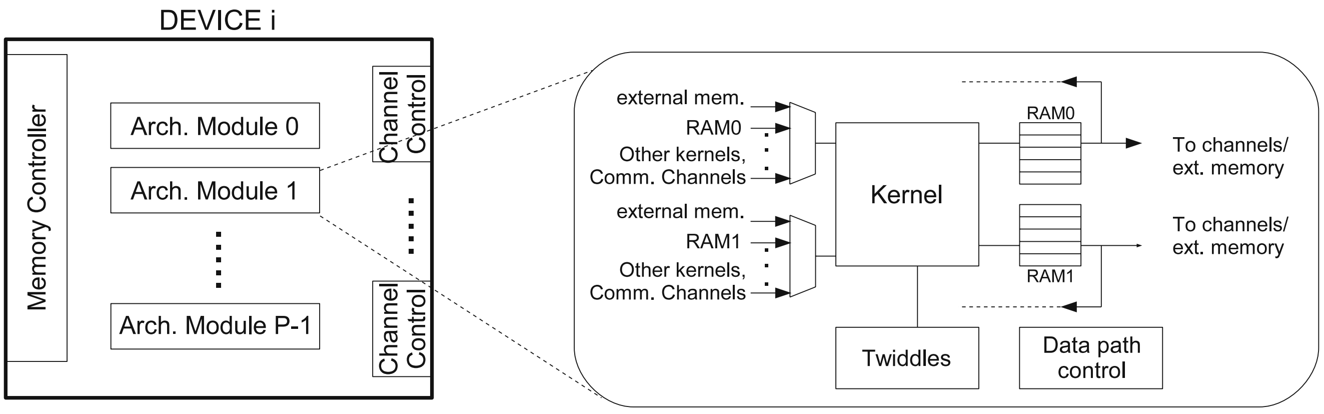


Figure 13 Device-level architectural model.

Table 1 Latency in c-steps for various sizes of DCT formulations.

	PSDA	Hsiao	Wang	Pueschel	Best of rest	PSDA-CT	Latency decrease
64	50	48	46	40	40	35	12.50%
128	54	80	72	65	54	57	−5.56%
256	86	161	112	118	86	75	12.79%
512	160	385	219	199	160	131	18.13%
1024	318	669	387	404	318	267	16.04%

A 2^n-point DCT formulation (that has not been fully expanded) can be decomposed into $O(n)$ different formulations at each application of Eq. 45. Since Algorithm 2 chooses one formulation at each step and the number of decomposition steps is $O(n)$, time complexity for the formulation exploration mechanism is $O(n^2)$.

6 Experiments

In order to assess the suitability of the various DCT formulations for distributed implementation, we partitioned them using the DMAGIC methodology. For DCT formulations where only one algorithm is available per size, such as HT-DCT and SS-FCT, the methodology only conducts a graph partitioning without formulation exploration. When using the developed formulation, the CT-like property is used to explore alternative formulations, as directed by Algorithm 2. In all cases, the DCT formulations are partitioned neglecting the post-scaling factor b_m (Eq. 4) . A scheduling algorithm using the As-Soon-As-Possible heuristic is run after partitioning to map the various DCT nodes to the available hardware kernels in each device.

Figure 12 shows the target topology for our experiments. It consists of Virtex-2 Pro XC2CP4 FPGAs connected in ring topology with a crossbar which mainly serves to communicate non-adjacent devices. Latency for communication through the direct channels and the crossbar is 1 and 2 cycles, respectively. Width for all communication channels is 16 bits. Latency for operations (addition, subtraction and multiplication) is 1 cycle. DMAGIC can target architectures that consist of an arbitrary number of devices and topologies. The target architecture was chosen since it is representative of common multi-FPGA boards produced by vendors such as Annapolis and Gidel, as well as high-end academic reconfigurable systems such as the Berkeley Emulation Engine 2 (BEE2) [16].

Figure 13 shows DMAGIC's target architectural model for each dedicated DHA device. The computational load of a partition is implemented by a set of interconnected modules that function as a customizable vertical folding structure. Each module consists of a functional primitive common throughout a particular DST's structure, as well as the necessary storage, control and data path options to implement the various stages of the transform. For instance, Fig. 13 shows a module with a 2-input kernel, data memories to store intermediate results, a twiddle table that contains the various coefficients for multiplication, and data path/control elements to establish data movement throughout execution.

Tables 1 and 2 summarize our results. In the majority of cases, the use of the PSDA-CT algorithm within the formulation-exploration methodology obtained better latency values than the rest of formulations, with up to 18% improvement over the best of the other cases. DMAGIC did not achieve the best latency overall for the 128-point DCT, however, its result (57 c-steps) was only surpassed by the fully fine-grained PSDA

Table 2 Execution time in seconds for various sizes of DCT formulations.

	PSDA	Hsiao	Wang	Pueschel	Best of rest	PSDA-CT	Time decrease
64	0.2	1	1	1	0.2	0.2	0.00%
128	0.2	11	5.2	17	0.2	0.2	0.00%
256	6	123	99	142	6	1	83.33%
512	14	1234	934	1321	14	3	78.57%
1024	189	18701	21509	16015	189	45	76.19%

formulation. Execution time was also significantly reduced when compared to the rest of formulations with reductions up to 83%. These results evidence the advantage of exploring different formulations of a given transform as part of the partitioning process. They also demonstrate that formulation-exploration can be performed in a non-exhaustive manner and yield acceptable results. Run time reduction can be attributed to the fact that our formulation exploration strategy starts by considering coarser-granularity formulations. This requires a smaller number of nodes and consequently a reduced graph partitioning vs. the rest of formulations, whose format requires them to be in their finest-granularity. Among the previously proposed formulations, PSDA consistently obtained the best results, both in latency and execution time, highlighting the importance of regularity on distributed hardware implementation of DCT.

7 Conclusions

We presented DMAGIC, a new algorithmically-aware, high-level partitioning methodology for DCTs targeted to DHAs. A study of previously proposed regular DCT formulations exposed their limitations for the type of distributed implementations that DMAGIC envisions. In order to take advantage of the formulation-level exploration capabilities of the methodology, a systematic factorization algorithm for the DCT was developed based on NPS-DCT algorithm. The derived PSDA-CT algorithm allows the decomposition of the PSDA into arbitrarily-sized functional primitives, while maintaining regularity. This significantly broadens the DCT formulation space, and allowed DMAGIC to find improved partitioning solutions as compared to previously proposed regular one-dimensional DCT formulations. The currently proposed strategy can be extended to multidimensional transforms by expressing them as tensor products of unidimensional transforms and lexicographically-ordering the n-dimensional input data into a vector. Future work includes the study of additional regular 2-D DCTs formulations to identify further partitioning opportunities at this dimensionality.

Acknowledgements This work has been performed at the University of Puerto Rico at Mayagüez with support from NSF grants $CNS - 0424546$ and $HRD - 9817642$.

References

1. Nikara, J. (2004). Application-specific parallel structures for discrete cosine transform and variable length coding. PhD thesis, Tampere University of Technology.
2. Hsiao, S.-F., & Tseng, J.-M. (2001). Parallel, pipelined and folded architectures for computation of 1-D and 2-D DCT in image and video codec. *Journal of VLSI Signal Processing, 28*(3), 205–220.
3. Srinivasan, V., Govindarajan, S., & Vemuri, R. (2001). Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 9*(1), 140–159.
4. Bringmann, O., Menn, C., & Rosenstiel, W. (2000). Target architecture oriented high-level synthesis for multi-FPGA based emulation. In *Proceedings of the European design and test conference 2000* (pp. 326–332).
5. Duncan, A. A., Hendry, D. C., & Gray, P. (2001). The COBRA-ABS high-level synthesis system for multi-FPGA custom computing machines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 9*(1), 218–223.
6. Arce-Nazario, R. A., Jimenez, M., & Rodriguez, D. (2006). Functionally-aware partitioning of discrete signal transforms for distributed hardware architectures. In *Proceedings of the 49th midwest symposium on circuits and systems* (pp. 1438–1441).
7. Arce-Nazario, R. A., Jimenez, M., & Rodriguez, D. (2007). Algorithmic-level exploration of discrete signal transforms for partitioning to distributed hardware architectures. *IET Computers and Digital Techniques, 1*(5), 557–564.
8. Nordin, G., Milder, P. A., Hoe, J. C., & Püschel, M., (2005). Automatic generation of customized discrete Fourier transform IPs. In *Proceedings of the 2005 design automation conference* (June).
9. Bornstein, C. F., Litman, A., Maggs, B. M., Sitaraman, R. K., & Yatzkar, T. (1998). On the bisection width and expansion of butterfly networks. In *Proceedings of the 12th international parallel processing symposium* (pp. 144–150) (March).
10. Wang, Z. (1991). Pruning the fast discrete cosine transform. *IEEE Transactions on Communications, 39*(5), 640–643 (May).
11. Püschel, M., Moura, J. M. F., Johnson, J., Padua, D., Veloso, M., Singer, B. W., et al. (2005). SPIRAL: Code generation for DSP transforms. In *Proceedings of the IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, vol. 93(2).
12. Puschel, M. (2003). Cooley-Tukey FFT like algorithms for the DCT. In *Proceedings of the IEEE international conference on acoustics, speech, and signal processing*, vol. 2, pp. 501–504 (April).
13. Takala, J., Akopian, D., Astola, J., & Saarinen, J. (2000). Constant geometry algorithm for discrete cosine transform. *IEEE Transactions on Signal Processing, 48*(6), 1840–1843.
14. Takala, J. H., Jarvinen, T. S., Salmela, P. V., & Akopian, D. A. (2001). Multi-port interconnection networks for radix-r algorithms. In *Proceedings IEEE international conference on acoustics, speech, and signal processing (ICASSP '01)*.
15. Singer, B., & Veloso, M. (2003). Learning to construct fast signal processing implementations. *Journal of Machine Learning Research, 3*, 887–919.
16. Brodersen, B., Chang, C., Wawrzynek, J., Werthimer, D., & Wright, M. (2004). BEE2: A multi-purpose computing platform for radio telescope digital signal processing applications. In *International square kilometre array meeting*.

Rafael A. Arce-Nazario received the B.Sc. degree in Computer Engineering from the University of Puerto Rico - Mayagüez (UPRM), in 1992, the M.Sc. degree in electrical engineering from the University Wisconsin - Madison, in 1993, and the Ph.D. degree in Computing and Information Science and Engineering from UPRM, in 2008. He is assistant professor in the Department of Physics and Electronics, University of Puerto Rico - Humacao. His research interests include electronic design automation, design and implementation of algorithms, and reconfigurable computing.

Domingo Rodríguez received the B.Sc. in Electrical Engineering (EE) from the City University of New York (CUNY) in 1979, the M.Sc. in EE from Union College, Schenectady, New York in 1981 and the Ph.D. in EE from CUNY in 1988. His research interests include hardware techniques for Digital Signal Processing and computational problems associated with large scale signal processing applications. He is currently the leader of the WALSAIP project (Wide Area Large Scale Automated Information Processing), and founder/coordinator of the Automated Information Processing Laboratory (AIP) at the UPRM.

Manuel Jiménez received the B.Sc. degree in electromechanical engineering from Universidad Autónoma de Santo Domingo, Dominican Republic, in 1986, the M.Sc. degree in Electrical Engineering from University of Puerto Rico at Mayagüez, PR (UPRM) in 1991, and the Ph.D. degree in electrical engineering from Michigan State University, in East Lansing, MI in 1999. He is currently a faculty member in the ECE Department of the UPRM. His current research interests include physical design automation of electronic systems, behavioral modeling of mixed-signal systems, embedded systems design, rapid systems prototyping, and engineering education. He is a member of the IEEE-Circuits and Systems Society and the American Society for Engineering Education.