Functionally-aware Partitioning of Discrete Signal Transforms for Distributed Hardware Architectures[*]

Rafael A. Arce-Nazario, Manuel Jimenez and Domingo Rodríguez
Electrical and Computer Engineering Department,
University of Puerto Rico, Mayagüez Campus, Mayagüez, Puerto Rico 00681-5000
Email: {rafael.arce, mjimenez, domingo}@ece.uprm.edu

Abstract— **A high-level partitioning methodology is introduced, which uses formulation-level discrete signal transform properties to provide improved results for their partitioning to distributed hardware architectures. We discuss how discrete signal transform characteristics were taken into account to focus design exploration during partitioning. Additionally, a description is given of the experiments conducted to determine the effect of formulation-level properties on solution quality. Perceived patterns in experimental results were used to generate 'partition-friendly' formulations for distributed hardware architectures.**

I. Introduction

The achievement of effective algorithm implementations to distributed hardware architectures (e.g multi-FPGA boards) is highly dependent on the process of partitioning. Most of the proposed high-level partitioning strategies apply generic local optimization techniques that miss out on alternate considerations which become apparent with knowledge of the algorithm's functionality [1]. Discrete signal transforms (DSTs) possess algorithmic level properties that have been used to manually obtain effective formulations for diverse computational architectures. Recently, methodologies such as FFTW and SPIRAL have been developed for the automated optimization of DST implementations to general purpose processor platforms [2][3]. However, these methods have yet to be successfully adapted for automated partitioning methodologies on dedicated distributed hardware architectures (DHAs). The current trend toward reconfigurable computers and multi-core systems-on-chip underlines the importance of developing effective partitioning techniques [4][5].

In our research we study the integration of properties such as symmetry, index mappings, and decomposition rules into automated partitioning strategies for DSTs to DHAs. We hypothesize that awareness of such characteristics during partitioning will result in a more focused exploration of the design space and improved solution quality. To this end, we designed a methodology that incorporates formulation-level transformations and other DST-derived strategies throughout the high-level partition process [6]. In this paper, we describe our methodology, emphasizing how DST-specific considerations influence the implementation of our partition optimization heuristic. We also describe several experiments designed to understand the effect of DST formulation-level characteristics on partition solution quality. The breakdown sequence of small-sized FFTs using the Cooley-Tukey factorization rule was found to have an orderly effect on partition results. Results from these experiments allowed us to define breakdown strategies for larger FFTs, achieving superior solution quality and reduced exploration time. Besides allowing us to define heuristic rules for our partitioning methodology, results from these experiments could be used by designers as guidelines for choosing DST formulations targeted to established topologies or to design cost-effective communication topologies for specific DSTs.

The rest of this paper has been organized as follows. Section II discusses related work in the area of DST partitioning to DHAs. In Sections III andIV we discuss the two main inputs to our problem: the discrete signal transforms and the target architecture. Next, Section V presents the proposed partitioning methodology and Sections VI and VII describe essential considerations taken, specifically, for partitioning DSTs and our chosen optimization heuristic. Section VIII reports on the experiments and their results. Finally, Section IX draws several conclusions and plans for future work.

II. Related Work

DSTs have received a variety of partitioning treatments when intended for DHA implementation. Generic high-level partitioning (HLP) methods are commonly benchmarked with DST algorithms such as the FFT and Discrete Cosine Transforms [1][7]. Throughout the HLP process, these methods treat DSTs as any other benchmark. They apply generic DFG-based local optimization techniques that don't consider functional-level opportunities. Among the few multi-device DST-oriented approaches, Kumhom used exhaustive exploration methods to search for optimal partitions of an FFT to a distributed memory multi-FPGA system [8].

Albeit these proposed automated methods, the favored approach to obtain high-performance DST implementations remains essentially manual and most benefits are obtained through careful manipulation and matching between the underlying computational architecture and the algorithm [9]. The implementation of fast DST algorithms requires a regular but congested communication scheme among the various computational elements, which can dominate performance even when the algorithm is manually mapped. A representative example of this situation was documented by Jones, et al.

[*] This research was partially supported by NASA Space Grant NGT5-40091* and NSF grant EPS-0223152.

[10]. The variety and manual nature of these treatments underlines the need for efficient automated methods to improve DST partitioning onto DHAs. We propose the use of DST algorithmic properties to improve their partitioning to DHAs.

Algorithmic-level rules have been successfully integrated into methodologies for automated DST code generation. FFTW and SPIRAL, two such methodologies, are essentially solution-space exploration engines which utilize DST factorization rules to generate and evaluate DST formulations for general-purpose processor architectures [2][3]. In FFTW, Cooley-Tukey's and Rader's algorithms are used to compose efficient FFT formulations by combining smaller, highly optimized blocks of code implementing smaller transforms [2]. SPIRAL incorporates a computer algebra system that uses breakdown and manipulation rules to explore alternative DST formulations as part of its design-exploration strategy [3]. An important objective of our work is to determine whether these rules can be successfully applied to the automated partitioning and mapping of DSTs to DHAs.

III. DSTs and Algebraic Framework

A linear separable d-dimensional transform of x is defined as:

$$X[k_1,..,k_d] = \sum_{a_d=0}^{N_d-1} .. \sum_{a_1=0}^{N_1-1} x[a_1,..,a_d]\alpha_1(a_1,k_1)..\alpha_d(a_d,k_d) \quad (1)$$

where the α_i's are the transform's functions. For example, for the d-dimensional DFT $\alpha_i(a_i,k_i) = e^{-j2\pi a_i k_i/N_i}$. A d-dimensional transform can be expressed as a tensor (Kronecker) product of unidimensional transforms:

$$\hat{X} = (A_{N_1} \otimes \ldots \otimes A_{N_d})\hat{x} \quad (2)$$

where A_{N_i} is the N_i point discrete signal transform matrix, $\otimes$ denotes Kronecker product and $\hat{x}$, and $\hat{X}$ are vectors of size $N = N_1 \cdot\cdot N_d$ obtained by ordering x and X lexicographically [11].

DSTs can be reformulated into algorithms with reduced computational complexity by taking advantage of symmetries in the transform's functions (e.g. the roots of unity in the case of the discrete Fourier transform). These fast algorithms can be expressed as a multiplication of the input signal vector by a succession of sparse matrices, which can be compactly expressed in Kronecker products algebra.

Expressions in Kronecker products algebra (KPA) are governed by well known rules, and have been used successfully, in a manual manner, to assist in the implementation of fast algorithms for the computation of DSTs [12]. The main idea is as follows. A given mathematical canonical formulation of an algorithm is expressed as a composition of functional primitives, i.e. factors which have been identified as efficient procedures on the targeted DHA, establishing in this a one to one correspondence. Variants of this canonical formulation are then sought using properties of KPA and trying to satisfy design criteria such as pipelining, parallelism, and data flow control. Each new variant, in turn, produces a different DHA implementation. The efficiency of the algorithm is evaluated using a cost objective function imposed on the design framework.

Our current efforts have been mostly directed toward Fast Fourier Transform implementations. Figure 1 shows a Cooley-Tukey formulation for the FFT size n=8 and its corresponding dataflow graph. DFGs derived from common FFT size n formulations (e.g. Cooley-Tukey, Pease) consist of $log_2(n)$ stages of $n/2$ butterfly-twiddle kernels. The high connectivity in DSTs, coupled with the limited inter-device communication resources available in DHAs emphasizes the need for effective mapping schemes.

$$F_8 = (F_2 \otimes I_4) T_1 (I_2 \otimes (F_2 \otimes I_2)) T_0 (I_4 \otimes F_2) R_8$$

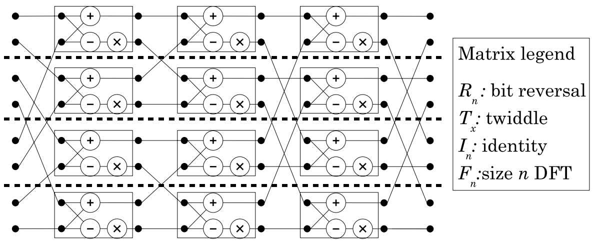


Fig. 1. Cooley-Tukey formulation of FFT size n=8 and its corresponding dataflow graph. Each kernel (denoted by a box) corresponds to a butterfly and twiddle multiplication.

IV. Target Architecture

Figure 2 illustrates our target architecture model, referred to as a Distributed Hardware Architecture (DHA). It consists of k dedicated hardware devices with local memory, connected in a ring or linear array topology with a crossbar serving as a global communication channel. This architecture is modeled after common multi-FPGA boards produced by vendors such as Annapolis (Wildforce) and Gidel (PROC20KE).

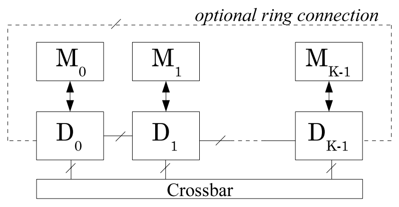


Fig. 2. Target topology.

V. Methodology

Figure 3 shows a conceptual map of our partitioning methodology. The inputs are (1) a DST specified as a KPA formulation and parameterized at least by the resolution of its points, and (2) a high-level specification of the target architecture, which includes the number and logic capacity of the devices and their connection topology. Based on the inputs, a series of heuristics reformulate the transform to expose characteristics that will be exploited by the partition/placement process. The algorithmic formulation is converted to a dataflow graph (DFG) whose nodes denote functional primitives. DFG nodes are as fine-grained as deemed necessary by the initial topology/transform heuristic analysis, in an effort to reduce design exploration time. Then, a partitioning/placement (P/P) algorithm is run on the DFG, assisted by high-level area and communication estimators. P/P is handled by a communication-channel-aware, n-way graph partitioning algorithm. Both Kernighan-Lin (KL) and simulated annealing based adaptations have been studied as iterative improvement strategies for P/P. The P/P solution and its cost are used by the control heuristics to guide the reformulation of the DST onto finer grained expressions. Formulation exploration ceases when a given number of successive reformulations have produced no considerable gain in partition quality.

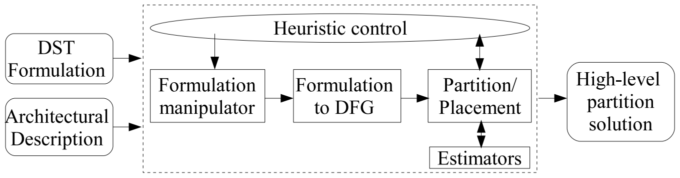


Fig. 3. Conceptual map of methodology.

To be able to define the heuristics controlling the optimization process, we need to have an idea of the effect of formulation properties on partitioning quality. We have conducted several experiments to gain insight into these effects and to see whether or not we can envision a strategy to heuristically guide the optimization. In the next sections we discuss our general partitioning rationale, the graph partition optimization heuristic, as well as the experiments performed to assess the effect of formulations in the partitioning results.

VI. DST Partitioning Rationale

Several DST considerations have been incorporated into our partitioning/placement algorithm to focus its optimization efforts and arrive at improved solutions. Fast DST algorithms, because of the regularity of their dataflow and inter-stage data dependence, have traditionally been partitioned vertically or horizontally [13][14]. Vertical partitioning maps computation as in an architectural-level pipeline, 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. We argue that, for the type of architecture we are targeting, horizontal partitioning will obtain lower latencies than the vertical scheme. To begin with, in vertical partitioning, all data enter through the first device and leave through the last. Thus, the potential of using each device as an input (reading new data from its own local memory) is lost. Furthermore, in a pipeline scheme, *every* data point must cross through each of the channels, while in a horizontal scheme even a naïve linear partitioning can reduce data communications requirements in half. When communication channels are the system's bottleneck, as they are in DHA systems, the excess communication in pipeline implementations hinders performance. Based on this reasoning, our partition/placement algorithm explores the solution space by exclusively considering horizontal partitioning schemes.

An additional distinctive feature of our partitioning algorithm is that the initial partition, instead of being random, is a balanced linear partition that respects the formulation's structure. The concept of balanced linear horizontal partitions is illustrated in Figure 1 by the dashed horizontal lines. Given a DFG, the initial partition scheme is obtained by dividing horizontally the structure into k equally weighted partitions. This initial solution strategy is used in an effort to conserve algorithm regularity on the final partitioning solution. Furthermore, linear partitions of common DST formulations already represent good solutions in themselves, potentially reducing exploration time.

Finally, most of the operations in fast DST formulations can only be scheduled to a specific c-step. Our algorithm takes advantage of this fact by only considering solutions that exchange nodes from the same computational stage. This schedule-aware consideration achieves a more focused solution space exploration.

VII. Kernighan-Lin Adaptation

The ultimate goal of the partitioning/placement process is to assign each of the DFG operations to an architectural device, while maximizing the estimated latency of the implementation. Our partitioning algorithm is an adaptation of the Kernighan-Lin heuristic extended for k-way partitioning to heterogeneous channel topologies. An overview of our implementation, KL-H, is presented in Algorithm 1. The objective function is the maximum communication cost of all available channels, where communication cost of a channel is defined as the product of the channel weight times the number of communications through that channel. The weight for a channel c is a relative measure of the impact on system latency of communicating a data point through c.

VIII. Experiments and Results

In order to use DST functional properties to improve their partitioning process, we need to first understand their effect on partition solution quality and exploration efficacy. To this end, several experiments were carried out to assess the effect of two properties that can be controlled algorithmically on DST formulations: inter-stage permutations and kernel granularity. The resulting observations are used to devise the heuristics employed throughout our proposed methodology. Diverse FFT formulations of various sizes and characteristics where partitioned using KL-H to target architectures consisting of four and eight devices. For all the experiments we assumed adjacent and crossbar channels weights of 1 and 2, respectively. Solution costs are measured according to the objective function in Algorithm 1.

Algorithm 1 KL-H: Adapted KL algorithm for n-way heterogeneous channel architectures.

```
Input: Data Flow Graph $G(V, E)$, Info about target architecture: devices $D = \{P_0 \dots P_{M-1}\}$, communication channels $C = \{C_0, \dots, C_{N-1}\}$ and their weights $W = \{W_0 \dots W_{N-1}\}$
Output: Partition assignment for each $v \in V$.
1.  Initial balanced partition $|P_0| \cong |P_1| \cong \dots |P_{M-1}|$
2.  while not all nodes are locked
    2.1.  Determine $a_i, b_i \in V$ s.t. $p(a_i) \leftrightarrow p(b_i)$ minimizes $cost()$
    2.2.  Perform swap $p(a_i) \leftrightarrow p(b_i)$. $C_i \leftarrow cost()$
    2.3.  Lock $a_i, b_i$
    2.4.  $[a_i, b_i, C_i] \rightarrow queue$
    2.5.  $i \leftarrow i + 1$
3.  end while
4.  Choose $k$ s.t. $C'_k = \min_{0<j<i}(C_j)$
5.  Reverse all swaps $a_j, b_j$ where $j > k$
6.  if $C'_k < \mathrm{C_k}$ then
    6.1.  $C'_k \leftarrow \mathrm{C_k}$, unlock nodes, goto step 2
7.  else stop

$cost() = \max_{i \in [0, N-1]} [R_i \times W_i]$,
where $R_i$ = number of required communications through $C_i$
```

A. Permutations

Solution quality of deterministic partitioning methods, such as Kernighan-Lin is highly dependent on the initial solution. Some popular graph partitioning algorithms actually run several times with different randomly created initial solutions and then chose the best result [15]. In an effort to salvage regularity through the final solution, our methodology generates initial linear horizontal partitions. Thus, formulations with different inter-stage permutations represent distinct initial solutions. To observe the effect of permutations, and possibly detect heuristic strategies to be applied in partitioning, we partitioned a range of sizes of five common FFT formulations: Cooley-Tukey (CT), Gentleman-Sande (GS), Pease (P), Stockham (S) and Transposed Stockham (TS).

Figure 4 illustrates a representative result from this experiment. The graph shows the percent difference in solution cost for various formulations on a architecture with 4 devices connected in linear array topology. Average solution costs for a randomly determined initial partition are also included. Two main observations can be drawn from these results. First, in general, randomly generated initial solutions yield inferior results than when starting with linear partitioning initial solutions. Secondly, none of the formulations exhibit a consistent advantage over others for all sizes and/or topologies.

B. Granularity

Clustering is commonly used during graph partitioning to help prune the solution space while improving solution quality and reducing time of convergence. When dealing purely with graphs, clustering techniques determine the formation of clusters based on graph qualities such as connectedness [15]. In generic HLP methods, information extracted from the high-level language algorithm specification or manually added information has been used to cluster DFG operations to form coarse-node graphs. At the formulation level, the granularity of a DST can be manipulated by decomposing larger sized DST kernels into combinations of smaller ones. We used the Cooley-Tukey factorization formula to study the effect of granularity in the partitioning of FFTs. This formula states that, if $n = pm$, then:

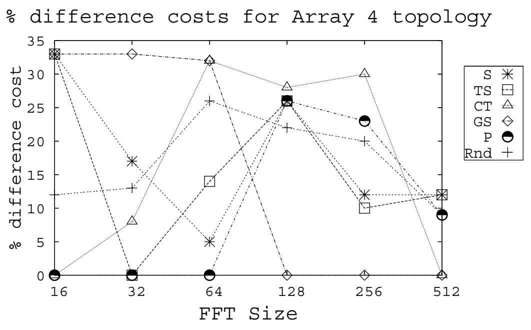


Fig. 4. Representative results from the permutation experiment.

$$F_n = (F_p \otimes I_m)\, T_{n,m}\, (I_p \otimes F_m)\, P_{n,p} \qquad (3)$$

where $n = mp$, F_n represents a size n DFT, I_n is an identity matrix, $T_{n,m}$ is a diagonal matrix of weights, and $P_{n,p}$ is a stride permutation matrix.

Using Equation 3 we generated formulations for every combination of stage granularities for a range of sizes of FFTs. For instance, for an FFT size $n = 8$, three formulations were generated: 2·2·2, 2·4, and 4·2, where each number corresponds to the size of kernels in each stage (e.g. the 2·2·2 formulation corresponds to that in Figure 1). The formulations were converted to their corresponding dataflow graphs and partitioned using KL-H. Table I summarizes our results by showing the formulations that achieved minimum cost for each of the FFT sizes 16 through 512. In this table, *Array P* and *Ring P* denote architectures with P devices connected in a linear array and ring topologies with an additional crossbar, correspondingly. Asterisks identify cases where multiple formulations achieved the minimum cost. For these cases, we show the minimum cost formulation with finest granularity.

TABLE I

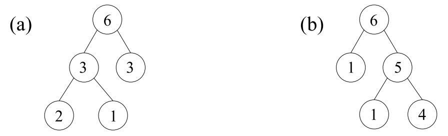
RESULTS FOR THE GRANULARITY EXPERIMENT.

	Array 4 Topology		Array 8 Topology	
Size	Cost	Min. Cost Form.	Cost	Min. Cost Form.
32	11	2·2·2·4*	32	2·2·2·2·2*
64	22	2·2·2·2·4*	48	2·2·2·2·4
128	43	2·2·2·2·2·2·2*	92	2,2,2,2,2,4
256	86	4·4·2·2·4	132	4·2·2·2·2·4
512	171	2·2·2·2·2·2·2·2·2*	276	2·2·2·2·2·2·4·2

As evidenced by the results, for the general case we cannot easily establish a correspondence between granularity scheme, architectural topology, and quality of solution. A practical fact that we can observe is that the finest grained formulations do not necessarily obtain the best results, so in many cases it would be wise to avoid these formulations as they also represent an increased exploration time.

C. Breakdown strategy

As evidenced in the previous experiments, the independent consideration of permutation and granularity did not reveal any discernible relationship with solution quality. For this reason, an additional experiment was conducted in which we explored the effect of *breakdown strategy* on the partitioning results. A breakdown strategy describes the order and divisors with which the decomposition rule such as Equation 3 is applied to arrive at a formulation. It ultimately has an effect on both granularity and permutations. *Split trees* are a common graphical representation of decomposition strategies [16]. Figure 5 shows two split trees for an FFT size $n = 2^6$ and their corresponding formulations.



$$\text{(a) } F_{64} = \left(\left(\left(F_4 \otimes I_2\right)T_{8,4}\left(I_4 \otimes F_2\right)P_{8,4}\right)\otimes I_8\right)T_{64,8}\left(I_8 \otimes F_8\right)P_{64,8}$$

$$\text{(b) } F_{64} = \left(F_2 \otimes I_{32}\right)T_{64,2} \left(I_2 \otimes \left(\left(F_2 \otimes I_{16}\right)T_{16,2}\left(I_2 \otimes F_{16}\right)P_{16,2}\right)\right)P_{64,2}$$

Fig. 5. Two split trees for FFT size $n = 2^6$ and their formulations. Each node v in the tree represents the computation of a 2^v-point DFT. The values of node v's children indicate how v's DFT is recursively computed.

All possible split trees were generated using Equation 3 for a range of FFT sizes from n =16 to 256 and partitioned for architectures with linear array and ring topology. A manual evaluation of the results revealed some common breakdown sequences to attain superior partitioning solutions. For instance, formulations from split trees where a first level equally distributes size among its children tend to obtain better partitioned solutions when targeting topologies consisting of 4 devices. In Figure 5, the formulation from tree (a) has better results than that of tree (b). The breakdown heuristics for generating 'partitioning-friendly' formulations vary across topologies, yet they remain similar across FFT sizes.

Breakdown strategy patterns observed in smaller FFTs were used to generate 'partition-friendly' split trees for larger FFT sizes, with excellent results. Table II reports results for the generated trees (GT) of several larger sized FFTs after partitioning with KL-H. As evidence to their solution quality, results are compared to the best solution obtained by far more time consuming simulated annealing (SA) runs.

TABLE II

SOLUTION COSTS FOR FFTS GENERATED USING BREAKDOWN STRATEGIES VS. BEST OBTAINED BY SIMULATED ANNEALING.

	Topology							
	4 Array		4 Ring		8 Array		8 Ring	
Size	GT	SA	GT	SA	GT	SA	GT	SA
512	171	171	103	109	220	288	116	115
1k	342	342	205	226	398	544	242	231
2k	683	683	410	475	758	1420	446	465
4k	1366	1430	820	923	1668	3074	870	983

IX. CONCLUSIONS

A new methodology for the partitioning of DSTs to distributed hardware architectures has been proposed which envisions the use of DST-specific considerations as part of its optimization loop. Experiments have been conducted to determine the effect of DST formulation on partitioning results for various target topologies. Of the tested properties, breakdown strategies were found to have an orderly effect on solution quality that can be exploited to generate 'partition-friendly' formulations. Our ongoing work includes experimentation with other common transforms, such as the discrete cosine and Hartley transforms, which will allow further development of our heuristics. Methodology components will be implemented and integrated into a complete prototype partitioning solution.

REFERENCES

[1] V. Srinivasan, S. Govindarajan, and R. Vemuri, "Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 9, no. 1, pp. 140–159, 2001.

[2] M. Frigo and S. G. Johnson, " The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[3] M. Püschel, et al., "SPIRAL: Code generation for DSP transforms," *Proceedings of the IEEE*, vol. 93, no. 2, 2005.

[4] K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Comput. Surv.*, vol. 34, no. 2, pp. 171–210, 2002.

[5] P. Magarshack and P. G. Paulin, "System-on-chip beyond the nanometer wall," in *DAC '03: Proceedings of the 40th conference on Design automation*. New York, NY, USA: ACM Press, 2003, pp. 419–424.

[6] R. Arce-Nazario, M. Jimenez, and D. Rodriguez, "An assessment of high-level partitioning techniques for implementing discrete signal transforms on distributed hardware architectures," in *Circuits and Systems, 2005. 48th Midwest Symposium on*, 2005, pp. 1438–1441.

[7] O. Bringmann, C. Menn, and W. Rosenstiel, "Target architecture oriented high-level synthesis for multi-FPGA based emulation," in *Proceedings of the European Design and Test Conference 2000*, 2000, pp. 326–332.

[8] P. Kumhom, "Design, optimization, and implementation of a universal FFT processor," Ph.D. dissertation, Drexel University, 2001.

[9] A. Chow, "Unleashing the power: A programming example of large FFTs on Cell," in *European Power.org Community Conference*, 2005.

[10] A. Jones, A. Nayak, and P. Banerjee, "Parallel Implementation of Matrix and Signal Processing Libraries on FPGAs," in *Proc. Intl. Conf. on Parallel and Distrib. Computing and Systems*, 2001.

[11] C. VanLoan, *Computational frameworks for the fast Fourier transform.* SIAM, 1992.

[12] G. Nordin, P. A. Milder, J. C. Hoe, and M. Püschel, "Automatic Generation of Customized Discrete Fourier Transform IPs," in *Proceedings of the 2005 Design Automation Conference*, June 2005.

[13] S.-F. Hsiao and J.-M. Tseng, " Parallel, Pipelined and Folded Architectures for Computation of 1-D and 2-D DCT in Image and Video Codec ," *The Journal of VLSI Signal Processing*, vol. 28, no. 3, pp. 205–220, 2001.

[14] G. Nordin, et al., "Automatic Generation of Customized Discrete Fourier Transform IPs," in *Proc. DAC*, 2005.

[15] G. Karypis, *Multilevel Hypergraph Partitioning*. Kluwer Academic Publishers, 2002, ch. 1.

[16] B. Singer and M. Veloso, "Learning to Construct Fast Signal Processing Implementations," vol. 3, 2002, pp. 887–919.