Approximate Signal Processing

S. Hamid Nawab,* Alan V. Oppenheim,† Anantha P. Chandrakasan,†

Joseph M. Winograd,* and Jeffrey T. Ludwig†

∗ ECE Dept., Boston University, 44 Cummington St., Boston, MA 02215

† EECS Dept., Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA 02139

— To appear in *J. VLSI Signal Processing*, vol. 15, no. 1–2, 1997. —

Abstract

It is increasingly important to structure signal processing algorithms and systems to allow for trading off between the accuracy of results and the utilization of resources in their implementation. In any particular context, there are typically a variety of heuristic approaches to managing these tradeoffs. One of the objectives of this paper is to suggest that there is the potential for developing a more formal approach, including utilizing current research in Computer Science on Approximate Processing and one of its central concepts, Incremental Refinement. Toward this end, we first summarize a number of ideas and approaches to approximate processing as currently being formulated in the computer science community. We then present four examples of signal processing algorithms/systems that are structured with these goals in mind. These examples may be viewed as partial inroads toward the ultimate objective of developing, within the context of signal processing design and implementation, a more general and rigorous framework for utilizing and expanding upon approximate processing concepts and methodologies.

1 Introduction

In many contexts it is desirable that algorithms and systems be structured so as to allow for the possibility of trading off between the accuracy or optimality of the results they produce and their utilization of resources such as time, power, bandwidth, memory, and system cost. For example, in communications systems, lossy source coding which results in approximate rather than exact signal transmission provides the opportunity to reduce the required transmission bandwidth. In communication networks with real-time constraints on the transport of multimedia data such as speech or video, scalable compression algorithms enable signal quality to be sacrificed in order to utilize reduced bandwidth or reduce transmission delay. In computer science, heuristic and approximate search strategies have been developed for applications where exact and exhaustive search strategies are intractable. The balance between accuracy and resource requirements in a system that performs digital signal processing (DSP) can be influenced by the selection of word lengths, filter order, and sampling rate.

In these various problem domains, there exist both formal and informal approaches to managing the tradeoff between accuracy and resources. Over the last decade there has been a growing interest in the development of more formal and structured approaches to obtaining a satisfactory balance between these opposing design factors. This work has been driven primarily by a desire to realize systems that perform demanding tasks within dynamically evolving environments. Various authors have used the terms approximate processing [1], imprecise computation [2], and flexible computation [3] to describe this basic approach to system design. The introduction of formal approaches to approximate processing offers the possibility of continuously optimizing system performance within the constraints imposed by the currently available resources and, in this way, achieving graceful degradation of performance in adverse circumstances as an alternative to system failure.

The establishment of basic design principles is a fundamental aspect of developing such formal approaches and certain concepts have been identified as being generally useful. In communications, for example, a well established approach to time-varying or unpredictable channel bandwidth is the notion of *embedded coding* [4] whereby the coding strategy involves a hierarchy of data. Including

additional levels of the hierarchy implies incrementally refining the quality of the decoded signal. A closely related and well-established example is the use of *progressive transmission* [5] for data or images in which signal transmission is structured so that successively better approximations to the signal are obtained as transmission proceeds in time. This is useful in a variety of situations, such as transmission over a channel which is only available for a limited and unpredictable time duration or when it is appropriate to halt transmission once a sufficiently accurate signal reconstruction is obtained.

A central concept in the approximate processing literature is that of computation structures with the *incremental refinement* property. Such structures, which have also been referred to as successive approximation, iterative refinement, or anytime [6] algorithms, are defined to consist of a succession of stages, each of which improves upon the answer produced by the previous one. In general, the improvement in each answer is measured with respect to the degree to which it approximates an ideal answer for the given application. A straightforward example of an incremental refinement algorithm is the long division procedure, which produces an additional significant digit of the quotient at each iteration. Another common example is Newton's root-finding method, in which the number of iterations (stages) that are performed is based on the desired amount of accuracy. In the context of signal processing, a recent example, to be discussed in greater detail in Section 3, is an FFT-based maximum-likelihood (ML) signal detector. Specifically, we show that by using the ML detection strategy after each stage of the FFT, a series of suboptimal detectors is obtained with performance improving incrementally toward that obtained with the full FFT. More generally, since multistage and iterative algorithms are common in signal processing, it is not surprising that DSP offers a fertile ground for exploring formal usage of approximate processing concepts in general and incremental refinement structures in particular.

Incremental refinement structures for basic categories of signal processing computations (such as transforms and filters) may be used as building blocks to aid the design of application-specific systems. Some applications may simply call for using a *fixed* number of stages of a given incremental refinement structure. In this case, the availability of an incremental refinement structure offers design-time flexibility for selecting the most suitable number of stages for the application system. In making this

selection, the designer would take into account both resource availability (such as processor capacity) and the expected effect on overall system performance. In the context of the model-year concept for the rapid prototyping of systems [7], this also offers the advantage that as the underlying hardware technology improves, one may obtain improved system performance in future design cycles by simply utilizing a greater number of stages of the same incremental refinement structure. With this approach, it would then not be necessary to modify the original processor architecture. The incremental refinement structures discussed in sections 3 to 5 are all amenable to this type of design philosophy.

In other applications, it is desirable for the overall system to select, during run time, the number of stages to use from a given incremental refinement structure. This type of situation would arise if the system is to operate in a dynamically changing environment (either in terms of its input data or in terms of resource availability). In such cases, it is desirable to design an adjunct control mechanism for the run-time adaptation of the number of stages to be used. An example of an adjunct control mechanism for incremental refinement structures in the context of a low-power digital filtering application is discussed in Section 6. As illustrated in that section, it is obviously important to keep the costs associated with the control mechanism low relative to the savings achieved by run-time adaptation. In the context of rapid prototyping, as the underlying hardware technology improves, improved performance may be obtained without requiring basic changes in the processor architecture.

Over the past several years, we have been exploring incremental refinement structures for basic categories of signal processing tasks and examining their implications for the design of application-specific systems. In this paper, we describe some of our major results and place them within the context of various concepts and formalisms emanating from the Approximate Processing subfield of Computer Science. Specifically, in Section 2 we provide a brief overview of that subfield and the relevance of its concepts to approximate signal processing. We then present in sections 3-6 various case-studies from our research on incremental refinement structures for DSP. In Section 3 we illustrate how the FFT may be used as an incremental refinement structure for signal detection. In Section 4 we derive new incremental refinement structures based on the DFT for real-time spectral analysis. In Section

5, we illustrate how an existing DSP computational structure for a particular application (DCT-based Image Encoding/Decoding) may be modified to obtain an incremental refinement structure. Section 6 describes how incremental refinement structures for certain IIR and FIR digital filters may be used in conjunction with a suitable control mechanism to design adaptive resource-conserving systems for low-power frequency-selective filtering.

2 Approximate Processing

In several areas of Computer Science there has been considerable interest in the development of formal approaches to the design of systems employing approximate processing techniques. The earliest studies dealing explicitly with this topic were performed independently in the areas of Artificial Intelligence and Real-Time Systems. In his study of automated reasoning systems for emergency medical diagnosis, E. J. Horvitz proposed [3] in 1987 the explicit use of resource constraints to inform heuristic reasoning strategies within a decision-theoretic framework. He suggested that reasoning strategies with incremental refinement capabilities could be used to maximize the utility of a system's results by explicitly weighing the risk of acting on an uncertain medical diagnosis against the cost of performing additional computation. He proposed that by quantifying the manner in which delayed results decreased a systems effectiveness and the amount by which the accuracy of the diagnosis improved with the amount of time spent in computation, one could derive the optimal amount of computation to invest in obtaining a diagnosis. That same year, in a series of publications, J. W.-S. Liu *et al.* outlined an approach to designing general purpose real-time computer systems that can skip non-critical portions of scheduled jobs in order to avoid missed deadlines during system overloads [2, 8, 9]. This work included new results in scheduling as well as a design tool for constructing and testing Approximate Processing systems. It also relied on the use of algorithms with the incremental refinement property.

Since those initial studies, a large number of publications on formal methods for employing approximate processing have appeared in the Computer Science literature. The ideas presented in the initial papers have been further developed, alternative approaches have been proposed, and some important

new results have been obtained. Several of these are presented in Sections 2.1-2.2. Other significant approaches are discussed in a recent review paper [10].

Although significant activity on this topic has been reported in the Computer Science literature, there has been very little direct migration of these results into other areas or into specific applications. This may be due in part to the lack of incremental refinement algorithms for many computational tasks and the lack of dependable analyses relating resource allocation and output quality for those that exist. In particular, the application of approximate processing methodology to the area of DSP has not been previously considered. Section 2.3 discusses several domains in which approximate processing methodologies have been successfully applied and addresses issues relevant to their application in the context of DSP.

2.1 Algorithm Characterization Using Performance Profiles

An important component of any formal approach to the design of systems employing approximate processing is the quantification of the tradeoffs between output quality and resource usage. Such analyses supply an abstract means for the representation and comparison of algorithms and provide a language in which various design problems can be formulated and solved.

The most comprehensive framework that has been proposed for characterizing quality/cost tradeoffs is based on the use of *performance profiles* [11] [12] [13]. Performance profiles are functions that map the possible resource allocations for an algorithm onto a numerical measure of output quality. This framework includes several different performance profile structures with varying degrees of complexity.

For a given algorithm $\mathcal{A}$, the simplest performance profile is a function $\mathcal{P}_{\mathcal{A}} : \mathbb{R}^+ \rightarrow \mathbb{R}$ that maps a resource allocation onto a quality measure. The resource allocation value may represent any useful resource measure such as time, memory, arithmetic operations, power, or number of processors. Quality is an objective measure of some property of the algorithm's output. Fig. 1 illustrates the performance profiles associated with two different computational structures. In Fig. 1(a), we see the profile associated with a "standard" algorithm—one that requires some fixed amount of resources to compute its

Figure 1: Typical performance profiles for (a) a "standard" algorithm and (b) an incremental refinement algorithm. (Adapted from [12].)

results and for which smaller allocations do not produce any meaningful results. Fig. 1(b) illustrates a performance profile associated with an incremental refinement algorithm. The output quality obtained from this algorithm can be seen to increase incrementally as additional resources are provided. It should be noted that *every* algorithm that uses resources and produces output has a performance profile—the concept is not restricted to approximation algorithms. In some instances, however, determining a meaningful performance profile for a given algorithm may be difficult.

There are several important aspects of algorithm performance that this simple performance profile does not capture. One such aspect is that for some algorithms, the quality of the output obtained may vary although its resource usage is fixed. In such instances, a *performance distribution profile* (PDP) can be used. A PDP for an algorithm $\mathcal{A}$ is a function $\mathcal{D}_{\mathcal{A}}: \mathbb{R}^+ \rightarrow \text{Prob}\{\mathbb{R}\}$ that maps a resource allocation onto a probability distribution over the quality of the results. One technique for avoiding the additional complexity of PDPs is to use an *expected performance profile* (EPP). An EPP is defined as a function $\mathcal{E}_{\mathcal{A}}: \mathbb{R}^+ \rightarrow \mathbb{R}$ mapping a resource allocation onto the expected value of the associated output quality probability distribution. EPPs have been utilized in several studies [3] [14] of approximate processing.

For some algorithms, the distribution of output quality obtained for a given resource allocation may depend on characteristics of the input data to the system. In such cases, the distribution of output quality can be conditioned on the quality of the inputs. This dependency is represented through the use of a *conditional performance profile* (CPP). Assuming input quality to be represented by a one-

dimensional measure, a CPP for an algorithm $\mathcal{A}$ is defined as a function $\mathcal{C}_{\mathcal{A}} : \mathbb{R} \times \mathbb{R}^{+} \rightarrow \text{Prob}\{\mathbb{R}\}$ that maps a measure of input quality and a resource allocation to a probability distribution over the quality of the results. Each of these types of performance profiles has been found to be useful in the context of approximate signal processing, as we illustrate in Sections 3–6.

Within this framework, a variety of useful relations regarding algorithms have been formulated [13]. These include formal definitions of the monotonicity of the quality of output obtained from an algorithm as its input quality or resource allocation is increased and the superiority of one algorithm over another (in both deterministic and stochastic senses).

2.2 Resource Allocation for Approximate Processing

A primary benefit of designing systems using incremental refinement algorithms is that as the amount of time available for computation fluctuates, the system can easily adjust the amount of computation performed to ensure timely completion of all tasks. Since the earliest studies on the subject, a central issue in the development of formal methods for approximate processing has been the optimal allocation of resources for systems comprised of multiple incremental refinement algorithms. Many variations of this problem have been studied, each with its own set of assumptions and goals.

A number of the problem formulations have been based on the large body of existing results for problems in real-time scheduling [15] [16]. In the general real-time preemptive scheduling problem, a set of tasks that are independent (or related through precedence constraints) must be scheduled on one or more processors. Each task has an associated time at which it becomes ready for execution (*ready time*), a time by which it must be completed (*deadline*), a measure of relative importance (*weight*), an amount of processor time that it requires to run to completion (*processing time*), and a time at which the scheduling algorithm is made aware of the existence of the task and its requirements (*arrival time*). Tasks can be either periodic or aperiodic. When the arrival time is zero for all tasks in a system, the scheduling problem associated with that system is *off-line*. Systems containing tasks with positive arrival times require *on-line* scheduling.

To develop scheduling algorithms for systems employing approximate processing, the established model for real-time scheduling has been extended [17] to allow the processing time assigned to a task to vary between some *mandatory time* and its *total processing time*. Associated with each task is an *error function* that maps the time allocation for the task and the error associated with the tasks that immediately precede it to a corresponding measure of the error in its output.[1] Tasks are said to be *error independent* if there is no relation between their errors. The *total weighted error* in a schedule is defined to be the sum of the errors of each task multiplied by their respective weights.

When all tasks in a system are off-line, error independent, aperiodic, have equal weights, and possess linear error functions,[2] an optimal scheduling algorithm has been found [18] that minimizes the total weighted error in the schedule for single processor systems, given that a feasible schedule[3] for the set of tasks exists. This algorithm employs an *earliest-deadline-first* policy [19] and has time complexity $O(n \log n)$, where n is the number of tasks being scheduled. An optimal scheduling algorithm has also been found [20] for the on-line counterpart of this problem, given that the mandatory portion of each task can be feasibly scheduled at its arrival time. This has been shown to have complexity of at most $O(n \log^2 n)$. When tasks have different weights, there exists an off-line algorithm [18] that minimizes the total weighted error using a *largest-weight-first* policy. This algorithm has complexity $O(n^2 \log n)$. Optimal algorithms for off-line scheduling on multiprocessor systems have also been developed [21] for the equal weight and differing weight cases. These algorithms are of complexity $O(n^2 \log^2 n)$ and $O(n^2 \log^3 n)$ respectively. Optimal scheduling has also been shown possible for tasks with some non-linear error functions. When the error functions are convex, a schedule that minimizes the *maximum normalized error* over all tasks can be found [22] in $O(n^2)$ time. The normalized error is defined as the difference between the time allocated to a task and its total processing time divided by its total

[1] In comparison with the framework for performance characterization described in section 2.1, this model incorporates the conditioning of output quality on that of the inputs but fails to incorporate the variability in output quality that may occur across different instances of inputs of the same quality. Error is complementary to quality and can be considered to be the difference between a given quality and the quality of an optimal result.

[2] This is equivalent to the error associated with each task being equal to the difference between the total processing time and the amount of time allocated to the task.

[3] A schedule is feasible if all tasks are scheduled to a processor for at least their mandatory processing time during the time interval between their ready time and their deadline time.

processing time.

No optimal scheduling algorithms have been found using the real-time scheduling framework for systems with periodic tasks, error-dependent tasks, or tasks with arbitrary error functions. In fact, the scheduling problem with periodic and error-dependent tasks has been shown [23] to be NP-hard for even the simplest case, where all tasks have identical periods and error functions. This has led to the study of suboptimal heuristic algorithms for such scheduling problems [24].

Another framework for the resource allocation problem has been proposed for systems whose structure can be expressed in the form of a *functional expression* [11] [12] [13]. This term is used in the sense of a pure functional programming language [25], where each function performs a fixed mapping from an input to an output without memory or side-effects. In this framework, the behavior of functions under varying resource allocations are characterized by CPPs as described in section 2.1. An example of a functional expression is $E(x) = F(G(x), H(x))$, where E is a functional expression composed of the elementary functions F, G, and H, and x is its input.

A functional expression is also a function and it too has a performance profile. However, when the functions that comprise the expression are themselves incremental refinement algorithms, the performance profile of the expression as a whole is dependent on the way in which the total computation time is divided among its constituent functions. The task of determining an appropriate division of the total computation time among the elementary functions has been termed the *compilation* of functional expressions. The problem of optimal compilation can be formulated as a search for the allocation of computing time among elementary functions that results in a CPP for the functional expression as a whole that has no superiors.

The problem of optimal compilation for functional expressions has been shown to be NP-complete but pseudo-polynomial [12]. That is, it is NP-complete in general but an optimal solution can potentially be found in polynomial time if a fixed bound is placed on the number of inputs that any function may have and if all elementary functions' CPPs are monotonically increasing over a bounded range of resource allocations. An algorithm has been found that produces the optimal CPP when the functional

expression contains no repeated subexpressions and the component functions' performance profiles are piecewise linear [12]. The complexity of this algorithm is linear in the number of elementary functions in the functional expression. For functional expressions with repeated subexpressions, no algorithm has been found that can perform optimal compilation in polynomial time. Several polynomial time heuristic solutions have been proposed [12].

2.3 Applications of Approximate Processing Methodology

From the results described in the previous sections it is clear that progress has been made in the development of a formal methodology for approximate processing. The methodology remains largely unproven in practice, however. This can be attributed in part to the fact that some of the important advances are quite recent. It is also due to the unique requirements that the methodology places on its area of application. Nevertheless, several experimental systems employing formal approaches to approximate processing have been reported and some work has been performed on the development of new algorithms to meet the needs of the methodology. A discussion of some of these systems and algorithms serves as the starting point for our consideration of applying approximate processing methodology to DSP.

The earliest reported experimental system employing approximate processing methodology was the Protos system for ideal control of probabilistic inference under resource constraints [26] [27]. The system used measures of the *expected value of computation* to guide the allocation of computational resources to incremental refinement algorithms for inferencing in a Bayesian probabilistic network [28]. Its advantages over inflexible reasoning systems in time-critical situations were demonstrated for three problems in medical diagnosis [29].

The framework under Protos for determining the expected value of computation was extended into a comprehensive philosophy of *metareasoning* [30] (reasoning about reasoning computation), which was used to develop new algorithms [31] for intractable search problems. By using explicit consideration of the value of computation, metareasoning-based approaches have significantly outperformed traditional

algorithms for game playing (othello) and problem-solving (robot path planning) [31].

The problem of robot path planning has also been addressed in a system employing compilation of functional expressions as described in the previous section [32]. This system incorporated a simulated incremental refinement algorithm for visual sensor interpretation and an actual incremental refinement algorithm for path planning in order to obtain the optimal CPP for the system as a whole. No actual incremental refinement algorithm for image interpretation was used in this work. The simulated incremental refinement algorithm artificially produced a map of the domain in which the probability of an error at each location was related to the amount of time allocated to that module according to a CPP chosen by the system designers.

In other work, a series of studies (e.g. [1, 33, 34]) was conducted on the use of approximate processing in a remote vehicle monitoring application. Rather than using incremental refinement structures, these systems were constructed with multiple methods for performing each system task, where each method produced a different tradeoff between resource usage and output quality. A heuristic scheduling algorithm was used to select from among the various methods at run-time. This approach, termed *design-to-time* scheduling, offers the advantage of having no reliance on the availability of incremental refinement structures for the tasks at hand. As disadvantages, however, there exist few results on optimal scheduling policies for design-to-time systems and the approach requires inherently higher system complexity through the use of functional redundancy (in the form of multiple methods).

Many of the advances in the development of formal methods for approximate processing have been based on two basic assumptions. First, that there exist incremental refinement algorithms for the desired system's tasks and, second, that the performance of these algorithms is adequately quantifiable. These requirements are not met by the currently available algorithms in most application areas and have therefore fostered efforts to develop and analyze incremental refinement algorithms for a variety of different applications.

A consideration of the field of DSP in this regard turns up a wide variety of important algorithms which have a natural incremental refinement structure. For example, Levinson's recursion algorithm

[35] for linear prediction, which is widely used in speech processing and other applications, produces all-pole signal models in successive iterations with successively increasing model order. The wavelet signal decomposition [36] can be implemented using a tree-structured filter bank in which each branch of the tree produces successively more detailed analyses of the time-frequency composition of the signal. Some of the less obvious methods for obtaining incremental refinement behavior in DSP applications are explored in Sections 3-6.

There also exists a rich set of tools for evaluating the performance of approximate DSP algorithms. The effects of variability in such system parameters as sampling rate, quantization, time and frequency resolution, filter and model order, and noise corruption have been studied in depth and are well understood. Resource usage requirements such as arithmetic complexity, memory requirements, and parallelization have also been analyzed in great detail.

The firm mathematical foundations on which DSP is built differentiate it significantly from the general realm of computational problems. As evidenced by some of our recent research, it is this property that most clearly suggests the potential for application of formal approaches to approximate digital signal processing. In the next four sections we discuss specific examples of incremental refinement signal processing algorithms. In each case we indicate how they may be viewed within the general conceptual framework of Section 2.

3 Signal Detection Using the FFT

Incremental refinement is a context-dependent property. That is, the intermediate results obtained from an algorithm may improve incrementally according to some measures of quality while not improving incrementally according to others. It is the context of a particular application that determines what the relevant quality metrics are, and consequently whether an algorithm has the incremental refinement property. An important task in applying approximate processing methodology to DSP is establishing the contexts in which existing algorithms have incremental refinement behavior.

In the context of detecting sinusoids in noise, the FFT possesses the incremental refinement property.

By considering the performance of the maximum-likelihood detection strategy applied after successive FFT stages, we have shown [37] that the performance of the resulting suboptimal detector improves incrementally, converging ultimately to that of the exact ML detector. This leads to important consequences such as the fact that for a wide range of SNR values at the input of the FFT, high probabilities of detection are obtained without the necessity of going to the last stage of the FFT. In this section, we review these results.

We begin by describing the traditional FFT-based approach to ML detection and the analysis of its performance. Next, we consider the data obtained at intermediate stages of the FFT in the context of ML detection and point out that at each successive stage of computation the effective SNR is doubled while the number of channels which could contain signal energy is halved. Compact expressions for the probability of detection, probability of false alarm, and the receiver operating characteristic (ROC) are discussed. We conclude our consideration of the FFT with a brief discussion of how these performance results can be applied to obtain a CPP.

3.1 FFT-Based Maximum-Likelihood Detection

The detection of a complex sinusoid of unknown frequency and phase in additive white Gaussian noise (WGN) can be formulated as a decision D between the two alternative hypotheses:

$$\begin{aligned} H_w &: x(n) = w(n), \\ H_s &: x(n) = s(n) + w(n), \end{aligned} \tag{1}$$

where $x(n)$ is the received data sequence, observed for $n = 0, 1, \ldots, M-1$, $w(n)$ is the noise process, and $s(n)$ is the sinusoid to be detected. The hypothesis H_w represents the case when only noise is present, and H_s the hypothesis that the signal is present.

We consider the detection of complex sinusoids of the form

$$s(n) = \sqrt{E} e^{j\frac{2\pi}{M} ln + j\phi}, \quad n = 0, 1, \ldots, M-1, \tag{2}$$

where E is the signal power (which is known), l is an unknown integer frequency index in the range $0 \leq l \leq M-1$, and ϕ is the unknown phase with possible values $0 \leq \phi < 2\pi$. The complex-valued noise process $w(n)$ with power spectral density $N_0/2$ can be defined as

$$w(n) = q(n) + jr(n), \quad n = 0, 1, \ldots, M-1, \tag{3}$$

where $q(n)$ and $r(n)$ are both real-valued WGN processes with variance $N_0/4$.

The maximum-likelihood detector for $s(n)$ consists of a bank of correlators followed by a comparator of their outputs and a threshold detector [38]. Each correlator can be thought of as producing at its output the magnitude-squared of the output obtained at time $n = M-1$ from a filter matched to one of the possible sinusoidal frequencies. Denoting by $C(k)$ the output from the correlator associated with frequency index k, we obtain

$$C(k) = \left| \sum_{n=0}^{M-1} x(n) e^{-j\frac{2\pi}{M} kn} \right|^2, \quad k = 0, 1, \ldots, M-1. \tag{4}$$

This output is equivalent to the magnitude-squared of the M-point DFT of $x(n)$ and is typically implemented using the FFT algorithm at a significantly reduced computational cost in comparison with a filter-based implementation.

The ML detection strategy dictates that the output of these correlators be compared by selecting the maximal value over all $C(k)$. If this value is greater than a threshold η, then the sinusoid is declared to be present (i.e. $D = H_s$), otherwise it is declared absent (and $D = H_w$). Using the Neyman-Pearson detection criterion [38], the threshold value is selected so that a fixed false alarm probability ($P_{FA} = \text{Prob}\{D = H_s | H_w\}$) is obtained.

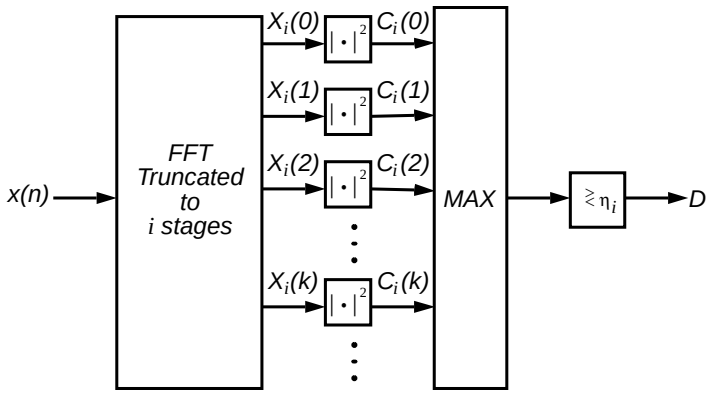


Figure 2: Incremental refinement detector of sinusoids in noise.

The probability of detection and ROC for this detector are determined by forming the distribution of the maximum energy value found across all elements of $C(k)$ under each input hypothesis [39]. Under hypothesis H_w, the FFT output consists of M complex-valued random variables each with real and imaginary parts that are independent and Gaussian-distributed with zero mean and variance $M \cdot N_0/4$. This results in values of $C(k)$ that are independent and χ^2-distributed with two degrees of freedom. Under hypothesis H_s, the FFT output for $k = l$ is equal to $M\sqrt{E}e^{j\phi}$ perturbed by a complex-valued noise component with independent Gaussian-distributed real and imaginary parts of zero mean and variance $M \cdot N_0/4$. Hence, $C(l)$ is noncentral χ^2-distributed with two degrees of freedom and noncentrality parameter M^2E. The values of $C(k)$ for $k \neq l$ have the same distribution as for the noise-only case.

3.2 Signal Detection from FFT Stages

For applications in which reduction in computation is desired, one may consider the result of terminating the FFT algorithm (radix-2 DIT or DIF) after an intermediate stage of computation and using its incomplete results as the basis for detection. The structure of a detector employing this approach is illustrated in Fig. 2, where we denote by $X_i(k)$ the output of the ith FFT stage and $C_i(k)$ is the magnitude-squared of $X_i(k)$.

By forming the distribution of the maximum energy value found across all elements of $C_i(k)$ under

each input hypothesis [39], we can determine the threshold values required to obtain a given probability of false alarm, the resulting probability of detection, and the receiver operating characteristic achieved by applying the ML detection strategy after any FFT stage.

Application of the ML detector according to the Neyman-Pearson criterion requires that we obtain the threshold value which gives the desired probability of false alarm (P_{FA}). Since the noise distribution depends on i, so must the threshold, which we denote by η_i. The probability of producing a false alarm, given that a threshold of η_i is applied, is [37]

$$P_{FA} = 1 - \left(1 - \exp\left(-\frac{\eta_i}{2^{i-1}N_0}\right)\right)^M. \tag{5}$$

It follows that a given value of P_{FA} is obtained when

$$\eta_i = -2^{i-1}N_0 \ln\left[1 - (1 - P_{FA})^{1/M}\right]. \tag{6}$$

The probability of detection $P_D(i)$ obtained when FFT processing is terminated after i stages can be derived from the distribution of $C_i(k)$ under hypothesis H_s [37]:

$$P_D(i) = 1 - \left(1 - \exp\left(-\frac{\eta_i}{2^{i-1}N_0}\right)\right)^{M-(M/2^i)} \left(1 - Q\left(\sqrt{2^{i+1}\mathrm{SNR}_{\mathrm{in}}}, \sqrt{\frac{\eta_i}{2^{i-2}N_0}}\right)\right)^{M/2^i}. \tag{7}$$

where $\mathrm{SNR}_{\mathrm{in}} = 2E/N_0$ and $Q(\cdot,\cdot)$ is Marcum's Q-function [38]. The receiver operating characteristic is found by substituting Eq. (6) into Eq. (7):

$$P_D(i) = 1 - (1 - P_{FA})^{1-2^{-i}} \left(1 - Q\left(\sqrt{2^{i+1}\mathrm{SNR}_{\mathrm{in}}}, \sqrt{-2\ln\left[1 - (1 - P_{FA})^{1/M}\right]}\right)\right)^{M/2^i}. \tag{8}$$

This performance analysis enables us to verify that the detector performance improves monotonically across stages. By considering the first derivative of the ROC, taken with respect to i, and making term-wise comparisons on the infinite series expansion of Marcum's Q-function, we obtain the result that for

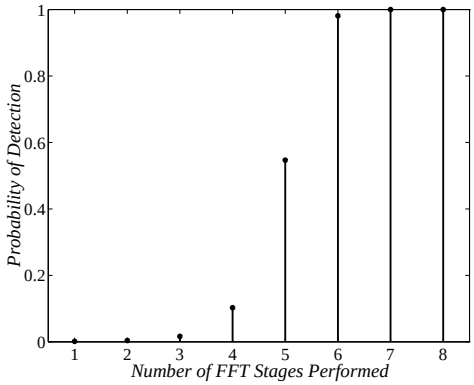


Figure 3: Detection probabilities at successive FFT stages when $\mathrm{SNR}_{\mathrm{in}} = -6$ dB, $P_{FA} = 10^{-4}$, and $M = 256$.

any input SNR and false alarm probability, the probability of detection increases monotonically with i. By the last stage, the performance obviously converges to that of the exact ML detector.

For any fixed input SNR and P_{FA}, the improvement in $P_D(i)$ between successive FFT stages is non-uniform across any given FFT algorithm. This is exemplified by Fig. 3 which shows a typical characteristic for $P_D(i)$ when $P_{FA} << 1$. In general, the change in $P_D(i)$ between successive stages depends upon two counteracting effects: the doubling of SNR at the output of the FFT, which increases the probability of detection, and the halving of the number of channels containing signal energy, which decreases it. Since $P_D(i)$ increases monotonically, it follows that the shape of the Q function, which increases monotonically with its first parameter, is the primary influence on the change in detection probability at each stage. In comparison, the reduction in the number of channels containing signal energy is of secondary importance.

Using the performance analysis discussed above, we can determine the number of FFT stages that must be completed in order to obtain a desired detection performance. Such information provides a sound basis for establishing a CPP for the FFT applied to the problem of detection, with input SNR as the conditioning input quality, the number of stages (or an equivalent measure of time or arithmetic complexity) as the resource measure, and the probability of detection as the output quality metric. Fig. 3 then corresponds to a "slice" of the CPP conditioned on an input SNR of -6 dB.

4 Spectral Analysis Using the DFT

Spectral analysis is an important component of many DSP systems and the most widely-used technique for its implementation is the DFT. The development of incremental refinement approaches to spectral analysis using the DFT can therefore be expected to have a significant impact on the applicability of approximate processing techniques for those systems.

Many different approximate DFT algorithms have been proposed. The most well-known are the "pruning"-type algorithms which obtain computational efficiency by excluding some subset of input and/or output points. Algorithms of this type include the FFT pruning algorithms [40] [41] [42], Goertzel's algorithm [43], and others [44] [45]. Advantages of pruning algorithms include the possibility of using the efficient FFT structure and the ease with which the error introduced through the approximation may be quantified. In contrast to pruning approaches, one may consider sacrificing the precision with which the DFT is computed. For example, such DFT approximations have been obtained using the summation by parts approach [46], the Poorman's approach [47], and the quantization and backward differencing (QBD) approach [48]. Of these various approximate DFT algorithms, only the QBD approach has been found to offer incremental refinement behavior for spectral analysis.

We have developed a new class of approximate DFT algorithms [49] which use both QBD approximation and pruning and have the incremental refinement property. We refer to these algorithms as DFT incremental refinement (DFT-IR) algorithms. Each of these algorithms consists of multiple stages, where each stage improves upon the DFT approximation produced by the previous stage. The quality of the DFT approximation after each stage can be characterized in terms of commonly used input-independent metrics for spectral quality: SNR, frequency resolution, and frequency coverage. The arithmetic complexity of each stage, however, depends upon the nature of the input signal. Thus, the characterization of the performance of these algorithms using a performance profile, as described in section 2.1, requires that we establish the probability with which a given level of output quality is obtained, given the characteristics of the input signal. Our approach to this task is to assume that the input signals in the application may be characterized by a Gaussian-distributed stationary process

with a known autocorrelation. From this, we have derived the probability of completing any particular algorithm stage, and thus producing a corresponding level of spectral degradation.

4.1 DFT-IR Algorithms

Every DFT-IR algorithm can be viewed as a cascade of stages, each of which takes a DFT approximation $\hat{X}_{i-1}(k)$ and produces an improved approximation $\hat{X}_i(k)$. Examples of this structure are illustrated using a block-diagram format in Fig. 4(a)-(c). The refinement process is "jump-started" with the computation of an initial approximation $\hat{X}_0(k)$, defined later in this section, whose computation is carried out using the block of type J in Fig. 4. Each subsequent stage performs one of three different updates, and each of these updates improves the previous approximation in a different way. The blocks of type S in Fig. 4 perform *SNR updates*. That is, each improves the SNR of the previous approximation by performing additional computation. Similarly, the blocks of types R and C represent *resolution updates* and *coverage updates* respectively. The specific arithmetic operations that are performed in any particular update depend upon the sequence of blocks that precede it. We indicate this dependence in Fig. 4 by denoting successive instances of a particular update block by S', S'', etc. The operations performed in each update, however, are independent of the *order* of the preceding blocks—they depend only on the number of each type of block that precede that update.

Every unique sequence of updates corresponds to a different DFT-IR algorithm. For the M-point transform of a Q-bit input signal, the total number of different DFT-IR algorithms, which we denote by N_A, can be shown to be

$$N_A = \frac{(Q + (3M/2) - 3)!}{(Q-1)!((M/2)-1)!(M-1)!}. \tag{9}$$

We represent each of these algorithms using a set of control parameters, s_i, r_i, and c_i. Table 1 lists the control parameters associated with the sequence of stages shown in Fig. 4(a). For each i, the control parameter values essentially represent the number of updates of the corresponding type that

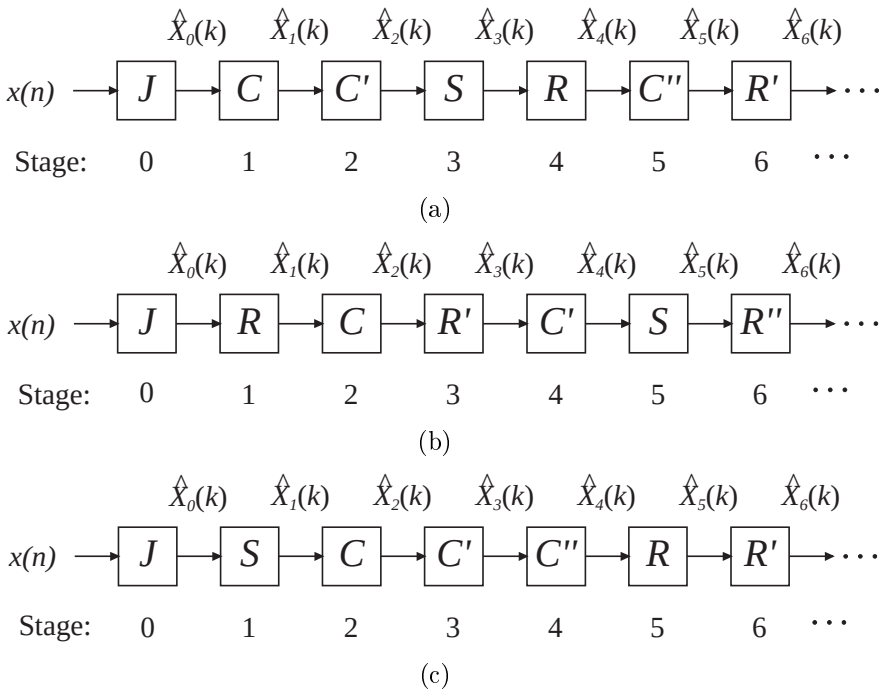


Figure 4: Block diagram depiction of three different DFT-IR algorithms. Each algorithm stage computes either the initial approximation (J), a SNR update (S), a frequency resolution update (R), or a frequency coverage update (C). The approximation $\hat{X}_i(k)$ is the output of the ith stage of computation. The operations that are performed in each block depend upon both the sequence of blocks that precede it as well as the values of the input data.

Control Parameter	*Stage (i)* 0	1	2	3	4	5	6	$\cdots$
s_i	1	1	1	2	2	2	2	$\cdots$
r_i	1	1	1	1	2	2	3	$\cdots$
c_i	1	2	3	3	3	4	4	$\cdots$

Table 1: Control parameters associated with the DFT-IR algorithm shown in Fig. 4(a).

are present up to and including the ith stage. For example, $s_i - 1$ is equal to the number of SNR updates performed through the ith stage. The offset by one in each of the control parameters accounts for the initial approximation produced by the "jump-start" stage.

The operation performed by each stage of a DFT-IR algorithm can be stated in terms of the values of the input data and the control parameters associated with that stage. We begin by defining the three updates mathematically. Their implementation is discussed in section 4.2. The *SNR update* improves an approximation by the incorporation of an additional bit level of the input signal. It is defined by

$$\hat{X}_i(k) = \begin{cases} \hat{X}_{i-1}(k) + \sum_{n=0}^{r_{i-1}} g_{s_i}(n) G_{s_i,n}(k), & k = 1, 2, \ldots, c_i, \\ \hat{X}_{i-1}(k), & \text{otherwise}, \end{cases} \tag{10}$$

where $G_{q,n}(k)$ is defined as

$$G_{q,n}(k) = \begin{cases} -e^{-j\frac{2\pi}{M}kn}/(1 - e^{-j\frac{2\pi}{M}k}), & q = 1 \\ 2^{1-q} e^{-j\frac{2\pi}{M}kn}/(1 - e^{-j\frac{2\pi}{M}k}), & q = 2, 3, \ldots, Q, \end{cases} \tag{11}$$

and $g_q(n)$ is the first circular backward difference of the bit vector $x_q(n)$, or

$$g_q(n) = \begin{cases} x_q(0) - x_q(M-1), & n = 0 \\ x_q(n) - x_q(n-1), & n = 1, 2, \ldots, M-1, \end{cases} \tag{12}$$

where $x_q(n)$ denotes the qth bit of the two's complement binary fraction representation of $x(n)$. In this

signal representation, the value of each element of the Q-bit signal $x(n)$ is related to the corresponding values of its component bit vectors by:

$$x(n) = -x_1(n) + \sum_{q=2}^{Q} 2^{1-q} x_q(n). \tag{13}$$

The *resolution update* improves the frequency resolution of the approximation by including an additional time sample of the input signal. It is defined as

$$\hat{X}_i(k) = \begin{cases} \hat{X}_{i-1}(k) + \sum_{q=1}^{s_i} g_q(r_i - 1) G_{q,r_i-1}(k) & k = 1, 2, \ldots, c_i, \\ \hat{X}_{i-1}(k), & \text{otherwise.} \end{cases} \tag{14}$$

The *coverage update* improves the approximation by adding to it an additional frequency sample. It is defined as

$$\hat{X}_i(k) = \begin{cases} \hat{X}_{i-1}(c_i) + \sum_{q=1}^{s_i} \sum_{n=0}^{r_i-1} g_q(n) G_{q,n}(c_i), & k = c_i, \\ \hat{X}_{i-1}(k), & \text{otherwise.} \end{cases} \tag{15}$$

The "jump-start" stage computes an initial approximation, $\hat{X}_0(k)$. It is defined as

$$\hat{X}_0(k) = \begin{cases} g_1(0) G_{1,0}(1), & k = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

The amount of spectral degradation present in the ith successive DFT approximation can be characterized in terms of the values of the control parameters. As an alternative to the recursive update form, $\hat{X}_i(k)$ can be expressed as

$$\hat{X}_i(k) = \sum_{q=1}^{s_i} \sum_{n=0}^{r_i-1} g_q(n) G_{q,n}(k), \quad k = 1, 2, \ldots, c_i. \tag{17}$$

From this equation, it can be seen that c_i dictates the spectral bandwidth, $2\pi c_i/M$ radians, over which $\hat{X}_i(k)$ is evaluated. The approximate transformation truncates $g_q(n)$ to a length of r_i samples, effectively reducing the frequency resolution of the transformation so that not more than r_i distinct frequencies can be resolved. The SNR of the approximate transform $\hat{X}_i(k)$ is reduced by signal quantization to approximately $6s_i$ dB.

4.2 Arithmetic Complexity

The DFT-IR algorithms implement the update equations (10), (14), and (15) without multiplications using a technique based on the summation of pre-computed vectors [50]. The use of pre-computed partial results to perform linear combinations is generally referred to as *distributed arithmetic* [51] [52]. The application of various distributed arithmetic techniques to DFT processing has been considered by others [53] [54], though not in the context of approximate DFT algorithms.

In the vector summation approach [50] used by the DFT-IR algorithms, the complex values of $G_{q,n}(k)$ are stored in memory and are added or subtracted from $\hat{X}_{i-1}(k)$ according to the value of $g_q(n)$ as dictated by the update equations. All summations corresponding to $g_q(n) = 0$ are skipped, resulting in a significant reduction in computation. The total number of real additions, κ_i, required for evaluating all the stages up to and including the ith stage is

$$\kappa_i = s_i r_i + 2\gamma(s_i, r_i)c_i, \tag{18}$$

where

$$\gamma(s_i, r_i) = \sum_{q=1}^{s_i} \sum_{n=0}^{r_i-1} |g_q(n)|. \tag{19}$$

The $s_i r_i$ term in (18) accounts for the backward differencing operations required to produce $g_q(n)$ from $x_q(n)$ over the region included through the ith stage of processing (recall that $g_q(n)$ is the backward differenced vector defined in Eq. (12)). The second term in (18) reflects the number of additions

required to evaluate only those terms of the update equations for which $g_q(n) \neq 0$.

The quantity $\gamma(s_i, r_i)$, defined in Eq. (19), is the total number of non-zero elements in the portion of the backward differenced signal vectors $g_q(n)$ included through the ith stage. We consequently refer to it as the *non-zero count* for stage i. For notational convenience, we abbreviate the quantity $\gamma(s_i, r_i)$ by γ_i, though it should be understood that there is a dependence on the control parameters associated with the ith stage. The non-zero count is related to the input signal $x(n)$ through Eqs. (12), (13), and (19). It represents the total signal-dependent contribution to the arithmetic cost of completing the ith stage of processing. For any two input signals the non-zero count may take on different values, resulting in a different arithmetic cost for performing the same sequence of stages on those signals.

A complete characterization of the arithmetic complexity of the DFT-IR algorithms requires that the signal-dependence of the non-zero count be determined more precisely. Our approach to this problem is based on a probabilistic analysis. When the input signal $x(n)$ is modeled as a stochastic process, the non-zero count, γ_i, is a random variable. The total arithmetic cost of completing all the stages up to and including the ith stage, κ_i, is related to γ_i through Eq. (18). Thus, if we wish to determine the probability distribution of the arithmetic cost for a class of signals, we must first obtain the probability distribution of the non-zero count. Such an analysis is reported in [55].

4.3 Spectral Degradation and Arithmetic Bounds

In the context of the formal approach to algorithm characterization described in section 2.1, we are especially interested in determining the effect of terminating any particular DFT-IR algorithm when its arithmetic cost reaches a specified bound B. As discussed earlier, we characterize this effect in terms of the probabilities of completion associated with each of the individual algorithm stages.

We define the *probability of completion*, P_i, of a DFT-IR algorithm to be the probability with which all stages up to and including stage i of that algorithm are completed using not more than B arithmetic

operations. That probability can be expressed as

$$P_i = \text{Prob}\left\{\kappa_i \leq B\right\}, \tag{20}$$

where κ_i is the arithmetic complexity measure given in Eq. (18). In turn, this leads to the conclusion:

$$P_i = \text{Prob}\left\{\gamma_i \leq \frac{B - s_i r_i}{2c_i}\right\}. \tag{21}$$

In order to determine the probability P_i, we once again need to determine the probability distribution of the non-zero count γ_i. That is, we need to characterize the same random variable which arose in the context of the arithmetic complexity measure κ_i.

The associated analysis is reported in [55], where distributions for γ_i are derived based on stationary Gaussian signal models. The results of that analysis are illustrated in Table 2, which indicate that reasonable *a priori* estimates of the probability of completion are obtained. The first eight columns of the table list the control parameters and associated output quality for selected stages from two different 256-point DFT-IR algorithms. The ninth column gives the predicted probability with which each of the stages will complete within a bound of 1000 arithmetic operations, based on the assumption of a power spectrum equal to the long term average spectrum of male speech. The tenth column gives the relative frequency with which each of the stages was observed to complete in 50,000 Monte Carlo trials, when applied to a signal having the assumed statistics.

4.4 Discussion

We have presented the DFT-IR class of approximate DFT algorithms and a probabilistic analysis of their arithmetic complexity. Through judicious use of suitable assumptions we have derived expressions for obtaining the probabilities of completion for the stages of any DFT-IR algorithm in the presence of a fixed upper bound on arithmetic complexity. These results may be used for establishing a PDP for any DFT-IR algorithm. Our results also lead to further interesting questions regarding algorithm

Algorithm	*Stage*	*Control Parameters*			*Output Quality*			*Probability of Completion (Theoretical)*	*Probability of Completion (Measured)*
	i	s_i	r_i	c_i	*SNR*	*Resolution*	*Coverage*		
1	70	1	8	64	6	8	$\pi/2$	0.999	0.999
	78	1	16	64	6	16	$\pi/2$	0.967	0.956
	94	1	32	64	6	32	$\pi/2$	0.635	0.628
	126	1	64	64	6	64	$\pi/2$	0.154	0.137
2	70	1	8	64	6	8	$\pi/2$	0.999	0.999
	71	2	8	64	12	8	$\pi/2$	0.894	0.841
	72	3	8	64	18	8	$\pi/2$	0.353	0.421
	73	4	8	64	24	8	$\pi/2$	0.042	0.095

Table 2: Description and analysis of four stages taken from each of two different 256-point DFT-IR algorithms. SNR is given in dB, frequency resolution is the maximum number of resolvable frequency components, and frequency coverage is in radians. Probabilities resulting from the theoretical analysis and Monte Carlo analysis are given for the completion of each stage within 1000 arithmetic operations. The input signal is presumed to have the long-term average spectrum of male speech sampled at 32 kHz.

selection from among the many algorithms in the DFT-IR class. For example, one may consider the development of procedures for efficiently obtaining the DFT-IR algorithm whose stage sequence may be considered as optimal with respect to application specific requirements on the evolution of output quality according to the three quality dimensions over time. Some preliminary investigations of such procedures [56] have shown this to be a promising avenue of exploration.

5 Image Decoding Using the 2D-IDCT

An important context in which approximate processing and incremental refinement are relevant is when an image or video sequence is broadcast across a variable bandwidth network or to receivers whose characteristics are not known to the sender and which possess a wide range of capabilities. Reduction of quality may be required at the receivers due to local performance limitations or in order to adjust to variable data rates. In either of these cases, the use of an incremental refinement structure for the decoder implementation enables performance to be easily adapted.

In the context of such applications, the two-dimensional inverse discrete cosine transform (2D-IDCT) represents another candidate for the use of an incremental refinement structure. The energy compaction

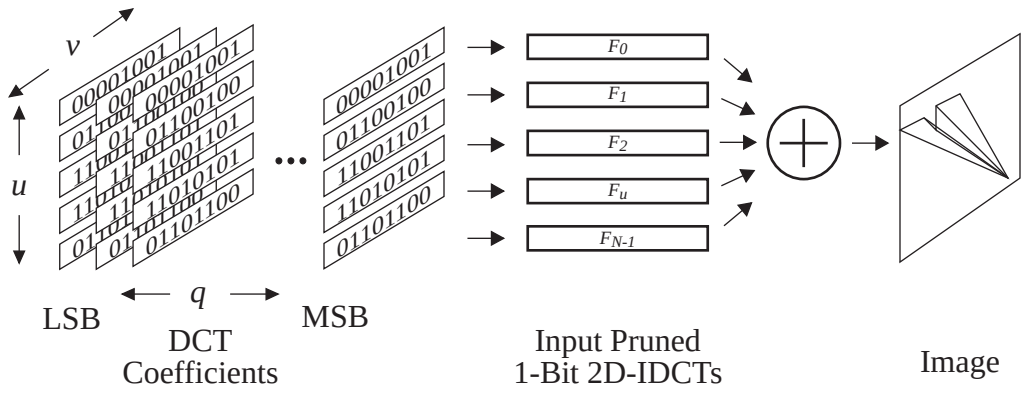


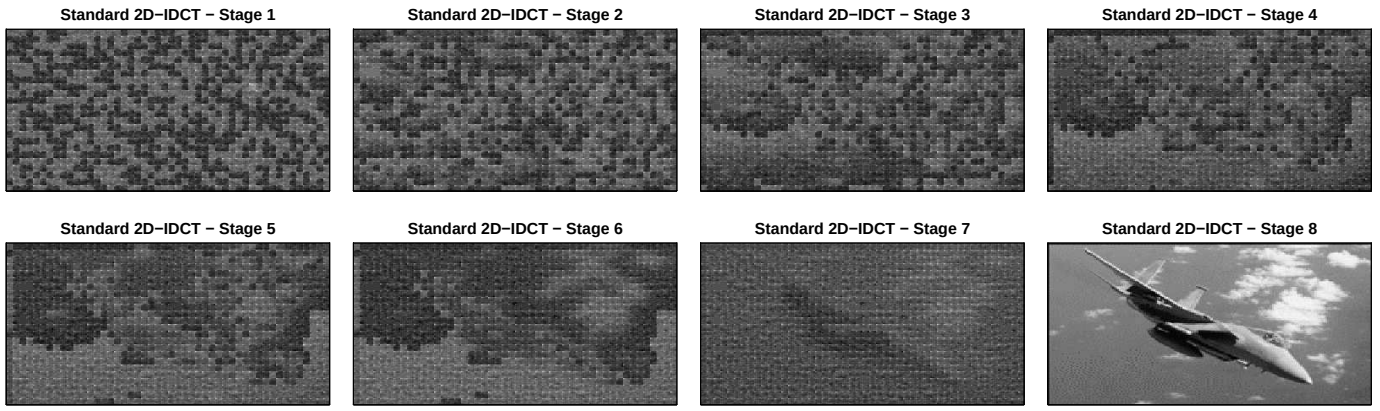
Figure 5: Schematic diagram of the proposed architecture for incremental refinement of 2D-IDCT approximations.

properties of the DCT make it a popular tool for image and video coding. Accordingly, IDCT computations comprise a significant proportion of the computational effort required in the decompression of the most widely used image and video coding standards.

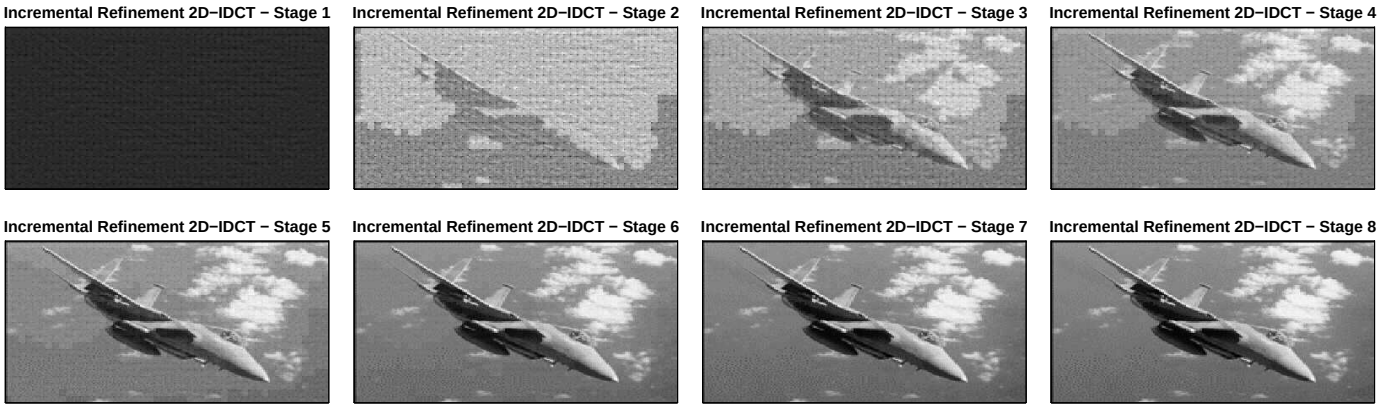
Our incremental refinement structure for the 2D-IDCT has the distributed arithmetic (DA) architecture shown schematically in Fig. 5. Other structures for computing the 2D-IDCT have previously been developed [57] [58] [59] using DA but they do not have the incremental refinement property.

A primary difference between our incremental refinement structure for the 2D-DCT and these other structures lies in the bit-serial ordering in which the distributed arithmetic operation is performed. Our architecture begins processing at the most significant bit of the input words, advancing progressively towards the least significant. With this approach, the intermediate results obtained at the output of the DA sub-system represent an approximation of the exact result based on the quantization of the input data to a fewer number of representation levels.

Another important innovation in our work lies in the basic manner in which distributed arithmetic has been applied to the 2D-IDCT. Previously reported implementations [57] [58] [59] are based upon the decomposition of the 2D-IDCT into the 1D-IDCT of the rows of the input data followed by the 1D-IDCT of each of the columns. Obtaining satisfactory incremental refinement behavior from this architecture is hindered by the fact that even when MSB-to-LSB bit ordering is used, the intermediate results produced by the first stage of row 1D-IDCT processing do not represent approximations to the



(a)



(b)

Figure 6: (a) The successive results obtained using a standard distributed arithmetic approach to performing the 2D-IDCT on 8x8 pixel blocks of a 384x192 pixel 8-bit image. (b) The results obtained during 8 successive stages of 2D-IDCT refinement using the successive refinement architecture described in the text.

desired output. This observation is illustrated in Fig. 6(a), which shows the successive results of the intermediate calculations for a standard distributed arithmetic approach to performing the 2D-IDCT on an image [57]. In Fig. 6(b) we show the results of the intermediate calculation using the distributed arithmetic architecture described in this section.

As in all applications of distributed arithmetic, the selection of an appropriate DA structure is strongly influenced by tradeoffs between performance and memory usage. For instance, a direct DA implementation of the 8x8 2D-IDCT would require 2^{64} words of ROM. In contrast, the architecture described here, with no memory saving optimizations applied, requires 2^{17} words of ROM. Due to the periodic structure of the IDCT basis functions, there exists considerable potential for reducing this memory requirement further. Such techniques have been successfully applied in the separable 2D-IDCT implementation [57], for which the memory requirements for the 16x16 transform were reduced from 2^{21} words to 2^{10} words.

To examine our 2D-IDCT incremental refinement structure, consider the NxN 2D-IDCT of $X(u,v)$:

$$x(i,j) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)X(u,v) \cos\left[\frac{(2i+1)u\pi}{2N}\right] \cos\left[\frac{(2j+1)v\pi}{2N}\right], \tag{22}$$

where $C(0) = 1/\sqrt{2}$ and $C(u) = C(v) = 1$ for $u, v \neq 0$. Throughout our derivation, $u,\ v,\ i,\ j \in \{0..N-1\}$. When $X(u,v)$ is encoded in two's complement binary the 2D-IDCT can be written as:

$$x(i,j) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v) \sum_{q=1}^{Q} X_q(u,v)\beta_q \cos\left[\frac{(2i+1)u\pi}{2N}\right] \cos\left[\frac{(2j+1)v\pi}{2N}\right], \tag{23}$$

with $X_q(u,v)$ denoting the qth bit of the binary representation of $X(u,v)$ and

$$\beta_q = \begin{cases} -1, & q = 1, \\ 2^{1-q}, & q = 2, 3, \ldots, Q. \end{cases} \tag{24}$$

To express the 2D-IDCT in a form suitable for applying distributed arithmetic, we rewrite Eq. (23) as

$$x(i,j) = \sum_{q=1}^{Q} \sum_{u=0}^{N-1} F_u(X_q(u,v), i, j)\beta_q, \tag{25}$$

with

$$F_u(X_q(u,v), i, j) = C(u)\frac{2}{N} \sum_{v=0}^{N-1} C(v)X_q(u,v) \cos\left[\frac{(2i+1)u\pi}{2N}\right] \cos\left[\frac{(2j+1)v\pi}{2N}\right]. \tag{26}$$

The arguments to each function F_u are a row vector of N bits (indexed by v) taken from the q-th position of the u-th row of $X_q(u,v)$, and a coordinate of $x(i,j)$. It's output is the 2D-IDCT of the given row vector of bits evaluated at position (i,j). By pre-computing and storing in memory the values of F_u, and implementing separately the F_u functions as indicated in Fig. 5, the entire summation over u in Eq. (25) can be evaluated in parallel for a single value of q. Thus, at each stage of computation (i.e. for each value of q) the structure updates its previous result with the 2D-IDCT corresponding to an entire additional bit plane of the input coefficients. The scaling associated with β_q in Eq. (25) is implemented via bit shifting in the output accumulators.

The incremental refinement structure outlined above for the 2D-IDCT may be used in a practical DCT-based image encoding/decoding system. The inclusion of each additional bit plane of the coefficients in the output of the 2D-IDCT corresponds to an increase in the SNR of the output by approximately 6 dB, making the derivation of a CPP for this incremental refinement structure a straightforward process. Consequently, an appropriate control strategy can be used by each receiver for terminating the decoding process at any intermediate stage in accordance with the availability of system resources and/or the desired quality of the decoded image.

6 Low-Power Frequency-Selective Filtering

Another area of interest in approximate signal processing is the formulation of efficient control mechanisms for the run-time adaptation of the number of stages to use in a given incremental refinement structure. The goal of such adaptation could, for example, be to conserve a limited resource (such as battery power) or to respond to dynamic changes in resource availability (such as processor cycles in a shared environment). In this section we discuss an example from our recent results on low-power frequency-selective digital filtering [60, 61, 62, 63] as an illustration of run-time adaptation for resource conservation.

The increasing demand for battery operated portable electronic devices has elevated power dissipation to be a critical design parameter. Since digital signal processing is pervasive in such applications, it is useful to consider how algorithmic approaches may be exploited in constructing low-power solutions. The average power consumption, P, of a digital system may be approximately represented through the expression:

$$P = \sum_i N_i C_i V_{dd}^2 f_s, \tag{27}$$

where C_i is the average capacitance switched per operation of type i (corresponding to addition, multiplication, storage, or bus accesses), N_i is the number of operations of type i performed per sample, V_{dd} is the operating supply voltage, and f_s is the sample frequency. There are many applications in signal processing such as real-time digital filtering where there is no advantage in exceeding a bounded computation rate (i.e., the sample rate is fixed). This attribute can then be exploited to reduce power dissipation.

Power reduction in signal processing systems in general involves optimization at all levels of the design abstraction including consideration of process technology, logic and circuit design, architecture design, and algorithm selection [64]. Typically, optimization to lower the power dissipation is done statically at design time. For example, parallel and pipelined architectures can be used to aggressively

scale power supply voltages without loss in functional throughput. An order of magnitude power reduction is possible over conventional approaches using this technique. Another trade-off involves choosing sign-magnitude representation to lower transition activity relative to two's complement representation.

Significant power gains can be achieved if optimization is done dynamically at run time, by considering and adapting to time varying signal statistics. Low-level data-dependent clock gating is one example of dynamic optimization currently in use by many low-power microprocessors. Another example involves the use of an adaptive power supply voltage system which exploits dynamically varying processing requirements [65]. The basic idea involves using a lower power supply voltage when the computational workload reduces rather than working at a fixed voltage and idling. The technique we discuss below enables the dynamic adjustment of the computational workload of frequency-selective digital filters.

Our approach involves exploiting signal statistics to reduce the effective capacitance switched in digital filters. Rather than using a fixed filter order (as is the case in conventional filter design), the filter order is allowed to vary with the aim of keeping it as small as possible while ensuring that the ratio of the passband power to the stopband power for the filter output, is kept above a specified threshold. Power consumption is reduced since the number of operations (N_i of equation (27)) is dynamically minimized rather than working at a fixed filter order optimized for the worst case signal statistics. To illustrate the ideas involved, we first introduce an incremental refinement structure for an IIR Butterworth filter. We then outline an adaptation framework for dynamically varying the number of stages used in such incremental refinement structures. It should be noted that our overall approach is not restricted to Butterworth or even IIR filters. For example, an incremental refinement structure for FIR filtering is described in [61].

6.1 Butterworth Incremental Refinement Structure

As an example of an incremental refinement structure for a lowpass IIR filter, let us consider the case of a Butterworth filter of order $2M_0$. A cascade structure for this filter consists of a serial connection

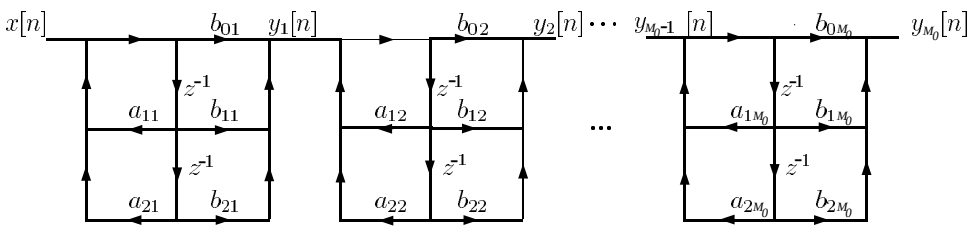


Figure 7: Incremental refinement structure for an IIR digital filter.

of M_0 second-order Direct-Form II sections, as shown in Figure 7. Each section corresponds to a pair of conjugate poles of the Butterworth filter and two zeros (both located at $z = -1$). Denoting the frequency response of the order-$2M_0$ Butterworth filter by $H_{M_0}(\omega)$, we may write

$$H_{M_0}(\omega) = G_1(\omega)G_2(\omega)G_3(\omega)...G_{M_0}(\omega) \tag{28}$$

where $G_i(\omega)$ denotes the frequency response of the ith second order section in the cascade structure of Figure 7. It can be furthermore assured that $G_i(0) = 1$. If only the first N sections ($N \leq M_0$) of the cascade structure in Figure 7 are utilized, the resulting order-$2N$ *truncated* Butterworth filter has the frequency response $H_N(\omega)$, given by:

$$H_N(\omega) = \prod_{k=1}^{N} G_k(\omega) \tag{29}$$

The Butterworth pole pairs are assigned to each of the second-order sections so that as the number of second-order sections is increased, the average attenuation in the stopband of the filter also increases, while keeping the passband gain close to unity. An empirical strategy for making such a pole-pair assignment is as follows: the pole pair for the M_0th section is selected first as the one which results in $|H_{M_0-1}(\omega)|$ having the smallest maximum deviation (from unity) in the passband. From the remaining pole pairs, the pair for the $(M_0-1)st$ section is selected as the one which results in $|H_{M_0-2}(\omega)|$ having the smallest maximum deviation (from unity) in the passband. The process is continued backwards in an analogous manner until each section has been assigned its corresponding pole pair. To illustrate,

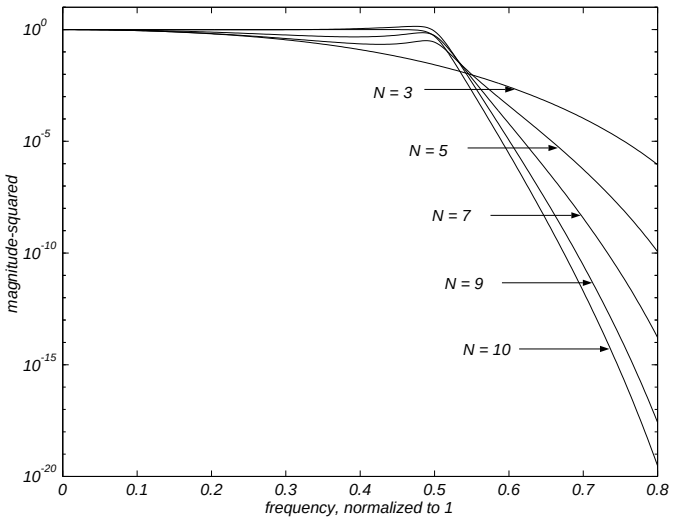


Figure 8: Magnitude-squared frequency responses for truncations of a 20th-order Butterworth filter with 3, 5, 7, 9, and 10 second-order sections.

consider the application of this strategy to a 20th order Butterworth filter with half-power frequency of $\pi/2$. The functions $|H_N(\omega)|^2$ obtained in this case are shown in Figure 8. It should be observed that as the number of sections (N) is increased, the attenuation in most of the stopband also increases. On the other hand, the filter gain remains close to unity in most of the passband.

6.2 Adaptation Objective

Suppose that a stationary input $x[n]$ with power spectrum $S_x(\omega)$ is filtered using the N-section truncated Butterworth filter, where $1 \leq N \leq M_0$, to obtain an output $y[n]$. We define the signal-to-noise ratio, SNR, as the ratio of the power in the passband to the power in the stopband. Specifically, the input SNR may be defined as

$$\text{ISNR} \triangleq \frac{P_x^{PB}}{P_x^{SB}}, \tag{30}$$

where

$$P_x^{PB} = \frac{1}{2\pi} \int_{PB} S_x(\omega) d\omega \tag{31}$$

and

$$P_x^{SB} = \frac{1}{2\pi} \int_{SB} S_x(\omega) d\omega. \tag{32}$$

Correspondingly, the output SNR is defined as

$$\text{OSNR}[N] \triangleq \frac{P_y^{PB}[N]}{P_y^{SB}[N]}, \tag{33}$$

where

$$P_y^{PB}[N] = \frac{1}{2\pi} \int_{PB} S_x(\omega) |H_N(\omega)|^2 d\omega \tag{34}$$

and

$$P_y^{SB}[N] = \frac{1}{2\pi} \int_{SB} S_x(\omega) |H_N(\omega)|^2 d\omega. \tag{35}$$

Ideally, one would like to select N to be the smallest value for which

$$\text{OSNR}[N] \geq \text{OSNR}_{tol}, \tag{36}$$

where OSNR_{tol} is the minimum tolerable output SNR for the application. Furthermore, if the input is non-stationary, $\text{OSNR}[N]$ will be time varying and consequently the filter order would have to adapt over time to reduce power consumption. Of course, this requires an adaptation framework whose overhead is low relative to the expected savings in power consumption.

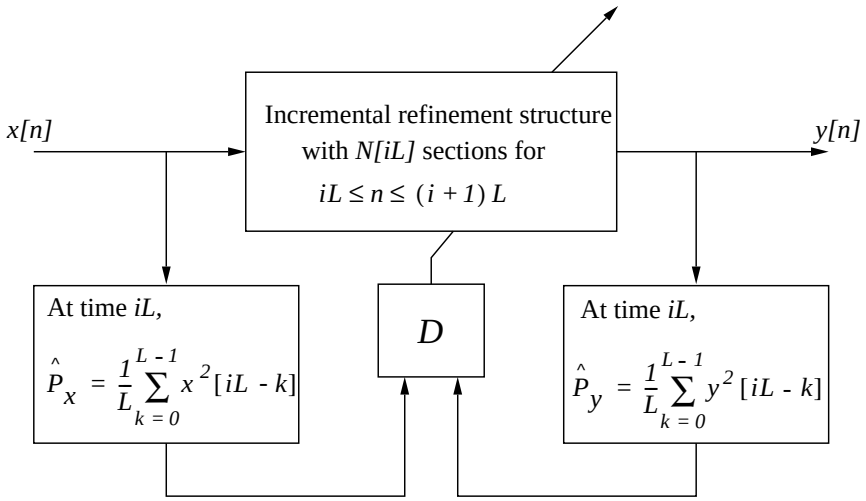


Figure 9: Adaptation strategy for updating the filter order after each new set of L output samples is computed.

6.3 Adaptation Framework

One of the low-cost adaptation strategies that we have developed [62] is illustrated in Figure 9. The number of sections utilized in the filter's incremental refinement structure is updated for every new set of L output samples. The low-cost update procedure involves the calculation of input and output signal-power estimates followed by the application of the decision module D shown in Figure 9. This module uses the signal-power estimates to form an estimate of the temporally local ISNR. This ISNR estimate is then used as the basis for selecting the filter order to be applied in computing the next set of L output samples. The precise formulation of the decision rule is based upon the following set of assumptions:

- $S_x(\omega)$ in the passband is *arbitrary*.
- $S_x(\omega)$ in the stopband is white but with unknown power.
- $S_x(\omega)$ in the transition band is negligible.
- $|H_N(\omega)|^2$ in the passband is equal to 1.

Let us consider a situation where an N_0-section ($N_0 \leq M_0$) truncated Butterworth filter is applied to a stationary input $x[n]$ to obtain the output $y[n]$. It can be shown [62] that under the stated assumptions,

$$\text{OSNR}[N_0] = \frac{P_y[N_0] - (P_x - P_y[N_0])\, P_h^{SB}[N_0]}{(P_x - P_y[N_0])\, P_h^{SB}[N_0]}. \tag{37}$$

where

$$P_x = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_x(\omega) d\omega, \tag{38}$$

$$P_y[N_0] = \frac{1}{2\pi} \int_{-\pi}^{\pi} S_x(\omega) |H_{N_0}(\omega)|^2 d\omega, \tag{39}$$

and

$$P_h^{SB}[N_0] = \left(\frac{1}{2\pi} \int_{SB} \left[1 - |H_{N_0}(\omega)|^2 \right] d\omega \right)^{-1} \left(\frac{1}{2\pi} \int_{SB} |H_{N_0}(\omega)|^2 d\omega \right). \tag{40}$$

Furthermore,

$$\text{ISNR} = \frac{P_y[N_0] - (P_x - P_y[N_0])\, P_h^{SB}[N_0]}{(P_x - P_y[N_0])}. \tag{41}$$

If now a filter with N sections ($N \leq M_0$) were to be used (instead of the N_0-section filter used previously) to process the same signal $x[n]$, we would obtain

$$\text{OSNR}[N] = \frac{\text{ISNR}}{P_h^{SB}[N]}. \tag{42}$$

To minimize power consumption, we would want to choose the smallest permissible value for N such that

$$\text{ISNR}\left(\frac{1}{P_h^{SB}[N]}\right) \geq \text{OSNR}_{tol}. \tag{43}$$

or, equivalently,

$$P_y[N_0] - (P_x - P_y[N_0])\, P_h^{SB}[N_0] \geq \text{OSNR}_{tol}\left(P_h^{SB}[N]\right)(P_x - P_y[N_0]) \tag{44}$$

To obtain a practical (low cost) decision rule on the basis of the above theory, suppose that we have applied a filter of order N_0 to obtain the output signal prior to and including time n. We may then obtain the following estimates:

$$\hat{P}_x = \frac{1}{L}\sum_{k=0}^{L-1} x^2[n-k] \tag{45}$$

$$\hat{P}_y[N_0] = \frac{1}{L}\sum_{k=0}^{L-1} y^2[n-k]. \tag{46}$$

Using these estimates in the inequality (44) and recognizing that the values of the product $\text{OSNR}_{tol}\left(P_h^{SB}[N]\right)$ can be prestored, the decision rule may be designed to search for the smallest value of N satisfying the inequality. This search requires $O(M_0)$ operations—at most M_0 table look-up operations, two subtractions, at most M_0+1 multiplications and at most M_0 comparisons—where M_0 is the number of sections in the original Butterworth filter. The selected value of N becomes the number of sections used in the incremental refinement structure to produce the next L output samples. The process continues in this way, updating the filter order every L samples.[4] Thus, for every new set of L output samples, the adaptation overhead involves $O(M_0)$ operations for the decision module D, and on

[4]Other variations on this update procedure have also been formulated [61, 60] but they are not considered here.

the order of $2L$ multiplications and $2L$ additions for calculating the estimates $\hat{P}_x$ and $\hat{P}_y$. Assuming that L is much greater than M_0, it is clear that the overhead is approximately two multiplications and two additions per output sample. For comparison, it should be noted that each second-order section of the filter requires five multiplications and four additions per output sample. Thus, even if the adaptive technique reduces the number of sections by only 1 over a particular interval, there is a net reduction in power consumption over that interval.

6.4 Performance Examples

From (42) we see that the quantity $1/P_h^{SB}[N]$ represents the factor by which the first set of N sections of the Butterworth incremental refinement structure improves upon the input SNR. We can thus provide a performance profile for this structure. Specifically, Fig. 10 shows $1/P_h^{SB}[N]$ as a function of N for the case of truncations of an original 10-section Butterworth filter with half-power frequency of $\pi/2$. The stopband in this case was defined to be between $5\pi/8$ and π. Clearly, the higher the filter order, the greater is the resulting improvement in output SNR. This incremental refinement structure along with the adaptation strategy described in the previous section (with $L = 100$ and $\text{OSNR}_{tol} = 1000$) was applied to two speech signals which had been frequency-division multiplexed. One signal was in the passband region of the lowpass filter and the other was in the stopband region. The sampling rate for the FDM speech signal was 16000 Hz. Figure 11 shows the speech signal in the passband, the speech signal in the stopband, and the evolution of the number of filter sections used by the adaptive filtering technique. Examination of the figure shows that as would be expected, the number of filter sections (N) used is large when the input SNR is small and is small when the input SNR is high.

7 Summary and Conclusions

As sophisticated signal processing systems proliferate in consumer electronics, distributed and networked environments, unattended sensors, etc. there is increasing emphasis on efficient use of associated resources. For these and other reasons, the concepts of approximate processing and incremental

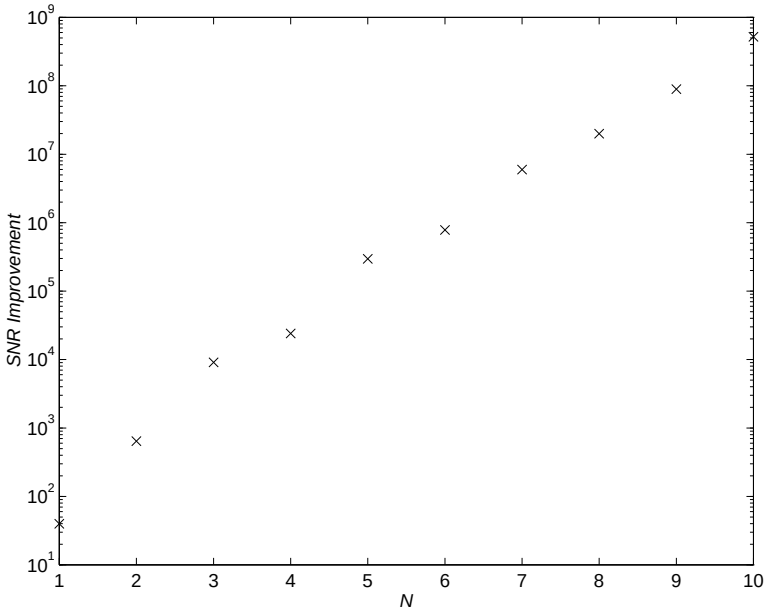


Figure 10: Performance profile for truncations of a 20th-order Butterworth filter with half-power frequency $\pi/2$.

refinement will undoubtedly play an increasingly important role in signal processing systems. In practice it has always been the case that many signal processing systems have utilized approximations to exact algorithms based on empirical or heuristic strategies. Also, many exact algorithms have been formulated mathematically in an iterative or recursive form which in many cases naturally result in an implementation with incremental refinement. However, many iterative algorithms are structured in this form for efficient realization of the exact algorithm rather than with the goal of utilizing them in the context of approximate processing. Thus in many cases, the iterative or recursive structure does not produce useful intermediate or approximate results until convergence is almost reached.

One of our objectives in this paper has been to suggest that there is the potential for developing a more formal approach to approximate processing and incremental refinement in the context of signal processing algorithms. Toward this end, we believe that there is significant potential in the formalism of approximate processing currently being developed in the computer science community. At present that formalism has only been applied in a few instances. However, among other things, it suggests an approach and an interest in developing structured ways of thinking about, evaluating, and generating

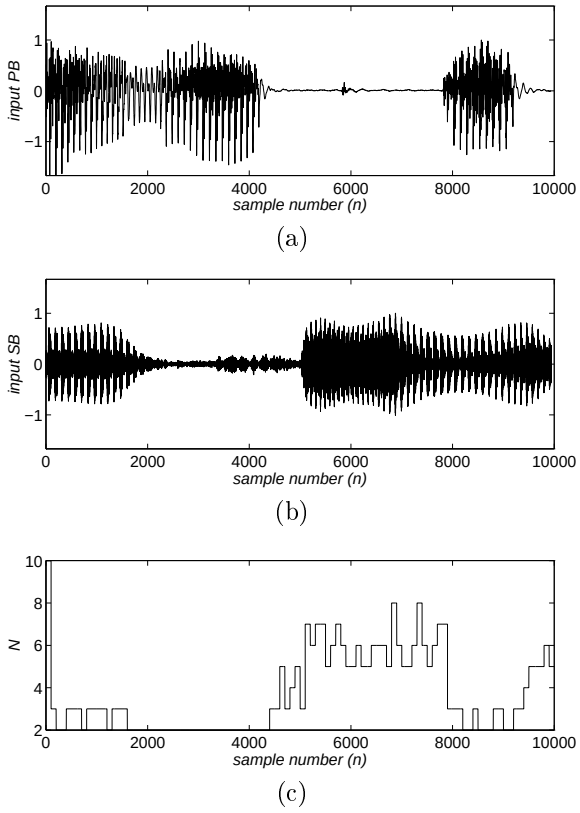


Figure 11: Demultiplexing of FDM speech using low-power frequency selective filtering. (a) Passband speech, (b) stopband speech, and (c) number of filter sections as a function of sample number.

algorithms of this type.

In the first part of this paper we have summarized some of the ideas and approaches toward approximate processing as currently being formulated in the computer science literature. We then presented four examples of signal processing algorithms that are structured with these goals in mind. As is evident in the discussion, in a general sense these four examples fit within the broad conceptual framework of approximate processing as discussed in Section 2. However, there remains a large gap between the formal structure and its application in detail to these four examples and more broadly to the field of signal processing in general. Closing this gap and expanding the formalism is one of the exciting challenges of this emerging topic.

The four examples that were presented in sections 3-6 were chosen as case studies to illustrate some specific points. The algorithm in Section 3 was chosen to illustrate how an existing recursive structure can be adapted as an incremental refinement structure. The particular example exploits the recursive structure of the FFT algorithm to obtain an algorithm for signal detection with incremental refinement. Section 4 presents an example of the development of a new incremental refinement structure for an existing signal processing transform (the DFT) in the context of a particular class of applications, specifically real-time spectral analysis. In Section 5, through the example of DCT-based image encoding/decoding, we illustrate how an existing computational structure, which does not have the characteristic of incremental refinement, can be modified so that it does. In Section 6, through the discussion of low-power algorithms for approximate processing in the context of digital filters, we illustrate the development of approximate processing algorithms for conservation of resources.

8 Acknowledgments

This paper was prepared in part through collaborative participation in the Advanced Sensors Consortium sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies either expressed or implied, of the Army Research

Laboratory or the US Government.

This work was sponsored in part by the Department of the Navy, Office of the Chief of Naval Research, contract number N00014-93-1-0686 as part of the Advanced Research Projects Agency's RASSP program.

References

[1] V. R. Lesser, J. Pavlin, and E. Durfee, "Approximate processing in real-time problem solving," *AI Magazine*, vol. 9, pp. 49–61, Spring 1988.

[2] K. J. Lin, S. Natarajan, and J. W. S. Liu, "Imprecise results: Utilizing partial computations in real-time systems," in *Proc. Eighth Real-Time Sys. Symp.*, (San Jose, CA), pp. 210–217, Dec. 1987.

[3] E. J. Horvitz, "Reasoning about beliefs and actions under computational resource constraints," in *Third Workhsop on Uncertainty in Artificial Intelligence*, (Seattle, WA), pp. 429–439, July 1987.

[4] N. S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video.* Englewood Cliffs, NJ: Prentice Hall, 1984.

[5] K. Knowlton, "Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes," *Proc. IEEE*, vol. 68, pp. 885–896, July 1980.

[6] T. Dean and M. Boddy, "An analysis of time-dependent planning," in *Proc. Seventh Nat'l. Conf. on Artificial Intelligence*, (St. Paul, MN), pp. 49–54, Aug. 1988.

[7] G. Caracciolo and J. Pridmore, "Architectures for rapid prototyping of embedded signal processors," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, (Detroit, MI), pp. 2703–2706, 1995.

[8] J. W. S. Liu, S. Natarajan, and K. J. Lin, "Scheduling real-time periodic jobs using imprecise results," in *Proc. Eighth Real-Time Sys. Symp.*, (San Jose, CA), pp. 252–260, Dec. 1987.

[9] K. J. Lin, S. Natarajan, and J. W. S. Liu, "Concord: A system of imprecise computations," in *Proc. 1987 IEEE Compsac*, (San Jose, CA), pp. 75–81, Dec. 1987.

[10] A. Garvey and V. Lesser, "A survey of research in deliberative real-time artificial intelligence," *J. Real-Time Sys.*, vol. 6, pp. 317–347, May 1994.

[11] S. J. Russell and S. Zilberstein, "Composing real-time systems," in *Proc. 12th Int. Joint Conf. Artif. Intel.*, (Sydney, Australia), pp. 212–217, 1992.

[12] S. Zilberstein and S. Russell, "Optimal composition of real-time systems," *Artif. Intel.*, vol. 82, pp. 181–213, Dec. 1995.

[13] S. Zilberstein, *Operational Rationality through Compilation of Anytime Algorithms*. Ph.D. thesis, U. C. Berkeley, 1993.

[14] M. Boddy and T. L. Dean, "Solving time-dependent planning problems," in *Proc. Eleventh Int'l. Joint Conf. on Artificial Intelligence*, (Detroit, MI), pp. 979–984, 1989.

[15] J. A. Stankovic and K. Ramamritham, *Hard Real-Time Systems*. Washington, DC: IEEE Computer Society Press, 1988.

[16] J. A. Stankovic and K. Ramamritham, *Advances in Real-Time Systems*. Washington, DC: IEEE Computer Society Press, 1993.

[17] J. W. S. Liu, W. K. Shih, K. J. Lin, R. Bettati, and J. Y. Chung, "Imprecise computations," *Proc. IEEE*, vol. 82, pp. 83–93, Jan. 1994.

[18] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, and J. Y. Chung, "Algorithms for scheduling imprecise computations," *Computer*, vol. 24, pp. 58–68, May 1991.

[19] E. L. Lawler and J. M. Moore, "A functional equation and its application to resource allocation and scheduling problems," *Management Science*, vol. 16, pp. 77–84, 1969.

[20] W. K. Shih and J. W. S. Liu, "On-line scheduling of imprecise computations to minimize total error," in *Proc. 13th Real-Time Sys. Symp.*, (Pheonix, AZ), pp. 280–289, Dec. 1992.

[21] J. Y. Chung, W. K. Shih, J. W. S. Liu, and D. W. Gillies, "Scheduling imprecise computations to minimize total error," *Microprocessing and Microprogramming*, vol. 27, pp. 767–774, 1989.

[22] K. I. J. Ho, J. Y. T. Leung, and W. D. Wei, "Minimizing maximum weighted error of imprecise computation tasks," technical report, Dept. of Computer Science and Engineering, University of Nebraska, 1992.

[23] J. Y. Chung and J. W. S. Liu, "Scheduling periodic jobs that allow imprecise results," *IEEE Trans. Computers*, vol. 39, pp. 1156–1173, Sept. 1990.

[24] I. K. Cheong, *Scheduling Imprecise Hard Real-Time Jobs with Cumulative Error.* Ph.D. thesis, University of Illinois at Urbana-Champaign, 1992.

[25] P. Henderson, *Functional Programming: Application and Implementation.* Englewood Cliffs, NJ: Prentice Hall, 1980.

[26] E. J. Horvitz, G. F. Cooper, and D. E. Heckerman, "Reflection and action under scarce resources: Theoretical principles and empirical study," in *Proc. 11th Int. Joint Conf. Artif. Intel.*, (Detroit, MI), pp. 1121–1127, 1989.

[27] E. J. Horvitz and G. Rutledge, "Time-dependent utility and action under uncertainty," in *Proc. 7th Conf. on Uncert. in Artif. Intel.*, (Los Angeles), pp. 151–158, July 1991.

[28] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artif. Intel.*, vol. 29, pp. 241–288, 1986.

[29] E. J. Horvitz, *Computation and Action Under Bounded Resources.* Ph.D. thesis, Stanford Univ., 1990.

[30] S. Russell and E. Wefald, "Principles of metareasoning," *Artif. Intel.*, May 1991.

[31] S. J. Russell and E. Wefald, *Do the Right Thing: Studies in Limited Rationality.* Cambridge, MA: MIT Press, 1991.

[32] S. Zilberstein and S. J. Russell, "Anytime sensing, planning and action: A practical model for robot control," in *Proc. 13th Int. Joint Conf. Artif. Intel.*, (Chambery, France), pp. 1402–1407, 1993.

[33] A. J. Garvey and V. R. Lesser, "Design-to-time real-time scheduling," *IEEE Trans. Sys., Man, and Cybernetics*, vol. 23, pp. 1491–1502, Nov. 1993.

[34] K. Decker, V. Lesser, and R. C. Whitehair, "Extending a blackboard architecture for approximate processing," *J. Real-Time Sys.*, vol. 2, no. 1, pp. 47–79, 1990.

[35] N. Levinson, "The Wiener RMS error criterion in filter design and prediction," *J. Math. Phys.*, vol. 25, pp. 261–278, Jan. 1947.

[36] M. Vetterli and C. Herley, "Wavelets and filter banks: Theory and design," *IEEE Trans. Sig. Proc.*, vol. 40, pp. 2207–2232, Sept. 1992.

[37] J. M. Winograd, S. H. Nawab, and A. V. Oppenheim, "FFT-based incremental refinement of sub-optimal detection," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, (Atlanta, GA), pp. 2479–2482, May 1996.

[38] H. L. Van Trees, *Detection, Estimation, and Modulation Theory—Part I.* New York: John Wiley & Sons, 1968.

[39] R. J. Kenefic, "Generalized likelihood ratio detector performance for a tone with unknown parameters in Gaussian white noise," *IEEE Trans. Sig. Proc.*, vol. 39, pp. 978–980, Apr. 1991.

[40] J. D. Markel, "FFT pruning," *IEEE Trans. Audio and Electroacoustics*, vol. AU-19, pp. 305–310, Dec. 1971.

[41] D. P. Skinner, "Pruning the decimation in-time FFT algorithm," *IEEE Trans. Acoust., Speech, and Sig. Proc.*, pp. 193–194, Apr. 1976.

[42] T. V. Sreenivas and P. V. S. Rao, "FFT algorithm for both input and output pruning," *IEEE Trans. Acoust., Speech, and Sig. Proc.*, vol. ASSP-27, pp. 291–292, June 1979.

[43] G. Goertzel, "An algorithm for the evaluation of finite trigonometric series," *Amer. Math. Monthly*, vol. 65, pp. 34–35, Jan. 1958.

[44] H. V. Sorensen and C. S. Burrus, "Efficient computation of the DFT with only a subset of input or output points," *IEEE Trans. Sig. Proc.*, vol. 41, pp. 1184–1200, Mar. 1993.

[45] O. V. Shentov, S. K. Mitra, U. Heute, and A. N. Hossen, "Subband DFT — Part I: Definition, interpretation, and extensions," *Signal Processing*, vol. 41, pp. 261–277, Feb. 1995.

[46] G. F. Boudreaux-Bartels and T. W. Parks, "Discrete Fourier transform using summation by parts," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 3, (Dallas, TX), pp. 1827–1830, 1987.

[47] M. P. Lamoureux, "The Poorman's transform: Approximating the Fourier transform without multiplication," *IEEE Trans. Sig. Proc.*, vol. 41, pp. 1413–1415, Mar. 1993.

[48] S. H. Nawab and E. Dorken, "Efficient STFT approximation using a quantization and differencing method," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, (Minneapolis, MN), pp. 587–590, Apr. 1993.

[49] J. M. Winograd and S. H. Nawab, "Incremental refinement of DFT and STFT approximations," *IEEE Sig. Proc. Letters*, vol. 2, pp. 25–28, Feb. 1995.

[50] S. H. Nawab and E. Dorken, "A framework for quality versus efficiency tradeoffs in STFT analysis," *IEEE Trans. Sig. Proc.*, vol. 43, pp. 998–1001, Apr. 1995.

[51] A. Peled and B. Liu, "A new hardware realization of digital filters," *IEEE Trans. Acoust., Speech, and Sig. Proc.*, vol. ASSP-22, pp. 456–462, Dec. 1974.

[52] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, vol. 6, pp. 4–19, July 1989.

[53] S. Chu and C. S. Burrus, "A prime factor FFT algorithm using distributed arithmetic," *IEEE Trans. Sig. Proc.*, vol. ASSP-30, pp. 217–226, Apr. 1982.

[54] F. J. Taylor, "An RNS discrete Fourier transform implementation," *IEEE Trans. Acoust., Speech, and Sig. Proc.*, vol. 38, pp. 1386–1394, Aug. 1990.

[55] J. M. Winograd and S. H. Nawab, "Probabilistic complexity analysis for a class of approximate DFT algorithms," accepted for publication in *J. VLSI Sig. Proc.*

[56] S. H. Nawab and J. M. Winograd, "Approximate signal processing using incremental refinement and deadline-based algorithms," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, (Detroit, MI), pp. 2857–2860, May 1995.

[57] M.-T. Sun, T.-C. Chen, and A. M. Gottlieb, "VLSI implementation of a 16x16 discrete cosine transform," *IEEE Trans. Circ. and Sys.*, vol. 36, pp. 610–617, Apr. 1989.

[58] S. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, Y. Yamashita, H. Terane, and M. Yoshimoto, "A 100-MHz 2-D discrete cosine transform core processor," *IEEE J. Solid State Circ.*, vol. 27, pp. 492–498, Apr. 1992.

[59] H. Fujiwara, M. L. Liou, M.-T. Sun, K.-M. Yang, M. Maruyama, K. Shomura, and K. Ohyama, "An all-ASIC implementation of a low bit-rate video codec," *IEEE Trans. Circ. and Sys. for Video Tech.*, vol. 2, pp. 123–133, June 1992.

[60] J. T. Ludwig, S. H. Nawab, and A. Chandrakasan, "Low power filtering using approximate processing for DSP applications," in *Proc. Custom Integrated Circuits Conf.*, (Santa Clara, CA), pp. 185–188, May 1995.

[61] J. T. Ludwig, S. H. Nawab, and A. Chandrakasan, "Low-power digital filtering using approximate processing," *IEEE J. Solid State Circ.*, vol. 31, pp. 395–400, Mar. 1996.

[62] J. T. Ludwig, S. H. Nawab, and A. Chandrakasan, "Convergence results on adaptive approximate filtering," *Advanced Signal Processing Algorithms,* (F. T. Luk, ed.), Proc. SPIE 2846, Aug. 1996.

[63] J. T. Ludwig, S. H. Nawab, and A. Chandrakasan, "Approximate filtering using incremental refinement structures," in preparation for submission to *IEEE Trans. Sig. Proc.*, Summer 1996.

[64] A. Chandrakasan and R. Broderson, *Low Power Digital CMOS Design.* Norwell, MA: Kluwer Academic Publishers, 1995.

[65] V. Gutnik and A. Chandrakasan, "An efficient controller for variable supply-voltage low power processing," in *VLSI Circuits Symposium*, June 1995.