Optimization of Signal Processing Algorithms

Raza Ahmed[†] and Brian L. Evans[‡]

razaa@master.cna.tek.com and bevans@ece.utexas.edu

[†]*Measurement Business Division, Tektronix Inc., Beaverton, OR 97707*
[‡]*Dept. of Electrical and Computer Engineering, The University of Texas, Austin, TX 78712-1084*

Abstract

We optimize implementations of one-dimensional and multidimensional signal processing algorithms by rewriting subexpressions according to a set of algebraic identities. We encode the algebraic identities as conditional rules, and program hill climbing and simulated annealing search techniques to apply the rules. Both of these search techniques avoid an exponential explosion in memory usage because they only keep a single state in memory instead of building the entire tree of possible equivalent forms. We compare the effectiveness of these search techniques in optimizing implementations of one-dimensional multirate signal processing algorithms. Our prototype environment is written in Mathematica.

1 Introduction

We optimize signal processing algorithms by applying algebraic identities to computations in algorithms to reduce implementation cost. We handle one-dimensional and multidimensional signal processing algorithms that involve both fine grain and coarse grain computation. We represent algorithms in an algebraic notation in which operators (systems) are applied to functions (signals). Algorithms and implementation cost can be programmed directly into the environment or imported from another environment. Once optimized, algorithms can be exported to other environments.

By rearranging computation in algorithms, our automated tools can search for a form of the algorithm that is globally optimal or satisfies design constraints. For example, we recently designed a touchtone decoder [11] using the Ptolemy design environment [1][12]. Ptolemy generated assembly code that required more data memory that was available on the target digital signal processor board. By rewriting the decoder algorithm using simple identities with the goal of reducing data memory usage, Ptolemy generated code that fit on the board.

In order to automate the optimization of signal processing algorithms, we need

- comprehensive collections of algebraic identities for signal processing algorithms,
- search mechanisms to apply the rules in an intelligent manner, and
- accurate estimates of implementation cost.

We address each issue after discussing the background for this work. We end the paper with examples.

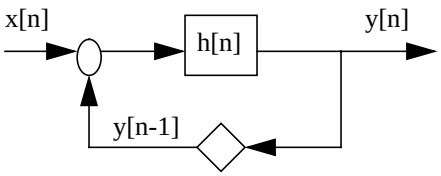
2 Background

Automated rearrangement systems based on applying algebraic identities to algorithms represented as algebraic expressions have produced efficient implementations of the following one-dimensional multirate signal processing algorithms:

- commutativity of an upsampler and downsampler in cascade [13],
- polyphase forms for rational rate changes [13], and
- multiband structures for optimal detectors based on the pruned Fast Fourier Transform [14].

In [13], the automated rearrangement system performed an exhaustive search for all alternative implementations. Because of the exhaustive search, this approach is severely limited because the searching requires an exponential amount of time and memory in the number of rearrangement rules and algorithm subexpressions. In [14], the automated rearrangement system applied heuristics to reduce the search. For example, the heuristics rearrange a set of parallel computations if and only if the parallelism (regularity) in the branches would be maintained. A similar system exists for morphological algorithms [19].

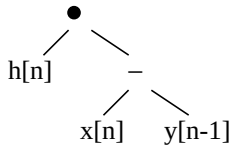
An algebraic notation is convenient for describing and analyzing algorithms. Algorithms in an algebraic notation become a sequence of equations. Equations are represented in an expression tree in which the nodes are operators (systems) and the leaves are functions (signals). Figure 1 shows three equivalent representations of a feedback system. Figure 1a gives a block diagram description. The diamond represents a delay of one sample and outputs a zero as the initial value (i.e., the value at y[-1] if the system begins at n=0). Figure 1b gives the algebraic equation for y[n]. Figure 1c gives the expression tree for y[n].



(a) Block diagram representation of feedback. The diamond represents a delay of one sample.

$$y[n] = h[n] \bullet (x[n] - y[n-1])$$
$$y[-1] = 0$$

(b) Algebraic representation of y[n]. The solid black dot represents the convolution operator.



(c) Expression tree for y[n].

Figure 1: Graphical and algebraic representations of feedback.

For Figure 1, it is possible to analyze the algebraic representation to derive the properties of the system such as stability, causality, linearity, and so forth, in terms of h[n]. We have not specified the data types for the system h[n] and the signals x[n] and y[n], nor how computation in the algorithm will proceed. The algorithm will work properly if it processes one sample at a time. If it were to process two samples at a time, then the flow graph in Figure 1a would *deadlock* because h[n] would require two input samples. Since the delay can only initially produce one sample in the beginning, y[-1], it cannot produce the second sample y[0] because y[0] has not been computed yet. This points out that algorithm rearrangement should be performed in the context of how the algorithm will be computed, and not imposed its own as in [13][14][23].

3 Algebraic Identities

The first step in an algebraic algorithm design environment is identifying and deriving the key identities used in optimizing algebraic expressions. The identities either simplify or rearrange expressions. Simplification removes redundant operations from an expression and therefore lowers implementation cost. Rearrangement produces a new equivalent expression that may sometimes give a lower or a higher or the same implementation cost.

One set of algebraic identities relies on the properties of the operator (system). Example properties, which include linearity and commutativity, are given in Table 1 [22]. This approach allows a compact representation of algebraic identities [13][14]. For example, the fact that the order of two linear time-invariant systems in cascade can be switched can be captured in one identity. In this way, we can add a new operator without needing to add any new identities, thereby avoiding an exponential explosion in the number of identities to capture all possible combinations for an identity. The identities based on system properties apply to a wide class of algorithms.

System Property	Meaning
Associative	can change grouping of inputs
Additive	distributes over addition
Commutative	can change order of inputs
Continuous	inputs are continuous
Delay	amount of delay before output is meaningful
Discrete	inputs are discrete
Homogeneous	scaled input gives a scaled output
Linear	additive and homogeneous
Linear Phase	true if the frequency response is a linear function of the frequency variable(s)
Memoryless	output does not depend on previous inputs and outputs
Separable	true if separable in all dimensions, false if completely non-separable, or a list of variables in which operator is separable
Shift Invariant	shifted input gives shifted output

TABLE 1. System properties used in algebraic identities.

We supplement the identities based on system properties. We use the identities for one-dimensional multirate digital signal processing operations report by Myers [13] and Covell [14]. We also add the collection of multidimensional multirate digital signal processing identities reported by Evans *et al.* [15].

We have encapsulated all of these identities in the computer algebra environment Mathematica [21] as if/then rules. We represent operators using the form

```
operator [ parameters ][ inputs ]
```

For example, a digital finite impulse response (FIR) filter with filter coefficients 1, 2 and 1 operating on signal x[n] becomes

```
DigitalFIRFilter [ {1, 2, 1}, n ] [ x[n] ]
```

Algebraic identities are expressed as conditional rules, e.g.

```
DigitalFIRFilter[{1,2,1}, n] [ Upsample[l, n][ x[n]] ] :>
   PolyphaseUpsample[l, DigitalFIRFilter[{1,2,1}, n], n][x[n]]
```

This rule rewrites an interpolator, represented as digital FIR filter that follows an upsample operation, in polyphase form. This particular rule has a similar form in multiple dimensions.

Evans *et al.* released a comprehensive set of identities in the form of rewrite rules for one-dimensional and multidimensional signal processing algorithms as part of the freely distributable Signal Processing Packages [22] for Mathematica [21]. The identities included over 60 rearrangement rules (13 are based on system properties) and 78 simplification rules. The packages also include hundreds of additional rules for computing one-dimensional and multidimensional Laplace, Fourier, *Z*, Discrete Fourier, and Discrete-Time Fourier Transforms of *algebraic* expressions. These transforms can aid in system analysis such as stability. The Signal Processing Packages are available on the World Wide Web from MathSource at `http://www.wolfram.com/`.

4 Search Techniques

Search techniques are susceptible to an exponential explosion in memory usage if they keep track of the equivalent forms they generate. In the worst case, both breath-first and depth-first searches have to build an exhaustive list of all of the possible equivalent forms to search for the optimal solution. We avoid having to keep track of all of the equivalent forms by using hill climbing and simulated annealing search techniques, which only keep a single state in memory [23].

Hill climbing and simulated annealing are *informed searches* in that they take advantage of structure in the expression to be optimized. Both approaches require an expression to optimize, a list of rearrangement rules, a successor function, a maximum number of iterations, and an evaluation function, as input arguments. The successor function takes an expression and a list of rearrangement rules and returns a list of equivalent expressions called successor states. The evaluation function measures the implementation cost of an expression and is used to rank equivalent forms.

Hill climbing takes the original expression as the current state. To find the next state, it generates the successor states by applying a set of rearrangement rules to each node in the expression tree. From the successor states, hill climbing chooses the successor state with the lowest implementation cost. The process continues until no successor state can be found with a lower implementation cost. In the solution space, hill climbing may terminate at a local minimum, become trapped in a flat valley, or oscillate in a crevice. When it fails to make progress, it can be restarted at a randomly chosen subexpression. If enough iterations are allowed, this approach will eventually find the optimal solution with respect to the rearrangement rules.

Simulated annealing is similar to hill climbing except that at every step, it chooses a random state from among the successor states. If the new state decreases the implementation cost, then it is always taken. Otherwise, the new state is taken with a probability that decays exponentially with the number of iterations performed so far. Our current experiments in applying hill climbing to rearranging polynomial expressions show that it outperforms the traditional search techniques of breadth-first and depth-first searching in execution time and memory usage to find the same optimal answer.

An example of using hill climbing to optimize the implementation of polynomial expressions is shown in Figure 2. The successor function returns a list of all possible pairs of a $x^n + b\ x^m$ terms from the current expression in factored form. The evaluation function is simply the number of operators (number of nodes in the expression tree). Hill climbing quickly finds the optimal implementation of the polynomial, which is Horner's form.

We have implemented depth-first, breadth-first, hill climbing, and simulated annealing search functions in a set of freely distributable Heuristic Search Packages for Mathematica. Accompanying the packages is an electronic notebook showing several examples of heuristic searches applied to optimize the computation of polynomials. The Heuristic Search Packages are available on the World Wide Web at `http://www.wolfram.com/` from MathSource.

5 Estimates of Implementation Cost

As we mentioned in Section 2, algorithm rearrangement must be performed with knowledge of how the algorithm will be computed in order to

- prevent introducing deadlock into the algorithm, and
- measure implementation cost.

The way an algorithm processes data is described by a *model of computation*. Dataflow models of computation, for example, express only the actual data dependencies that exist in an algorithm, so they track the flow of data. Dataflow models naturally describe data-intensive

```
In[3]:= poly = A x + B x^2 + C x^3 + D x^4 + E x^5 + F x^6
                    2       3      4       5      6
Out[3]= A x + B x + C x + D x + E x + F x

In[4]:= {timing, optpoly} = Timing[HillClimbing[poly]]
Out[4]= {1.41667 Second,
                  x (A + x (B + x (C + x (D + x (E + F x))))) }

In[5]:= optcost = initevalpoly[optpoly]
Out[5]= 35

In[6]:= initcost = initevalpoly[poly]
Out[6]= 110

In[7]:= FactorReductionInCost = N[initcost/optcost]
Out[7]= 3.14286
```

Figure 2: Hill climbing search technique in Mathematica for optimizing polynomial calculations. Multiplication cost was 5 units and addition cost was 1 unit. Implementation cost was reduced by a factor of 3.14.

operations such as multirate digital signal processing algorithms.

In dataflow, an algorithm can be scheduled in a variety of ways, but the schedule must honor the dependencies of data in the algorithm. Synchronous dataflow [2] models each operator (system) as a consumer and producer of a fixed number of data samples for each of its inputs and outputs. All of the data that is produced gets consumed. The model imposes that the flow of data cannot change in a way that is dependent on the value of data so that the model can be converted into a schedule (an ordering of computation) that is deterministic and fixed. The schedule can be repeated periodically without requiring unbounded memory. This type of data processing is well-suited for digital signal processing in which an algorithm is applied to every input sample or block of input samples over and over again.

Figure 4 shows a simple algorithm consisting of two components (subexpressions) labelled A and B. The algorithm is modeled as a block diagram (graph) using Synchronous Dataflow (SDF). In the SDF graph, component A must execute three times and component B must execute twice to balance the production and consumption of data samples. Several possible schedules exist, e.g. AAABB and 3(A) 2(B) and A 2(A B). In the first schedule, A is executed three times to produce a total of 6 samples which are stored in a buffer. For each invocation of A and B, the code implementing them is replicated. The second schedule uses a buffer of the same size, but requires much less program memory. The third schedule reduces the buffer size to four samples but trades this improvement for an increase in program memory. The implementation cost is a weighted sum of data memory, program memory, and execution time. The "best" schedule depends on the weights of the cost function, the schedule, and the implementation of A and B.

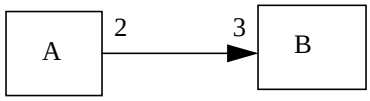


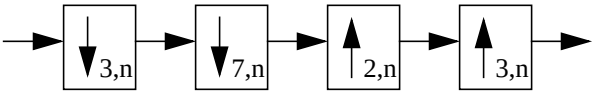
Figure 4: A simple algorithm modeled using Synchronous Dataflow. Component A produces 2 samples on its output port, and component B consumes 3 samples on its input port.

In order to obtain efficient rearrangements of algorithms, we need to get feedback from a software environment that can measure the implementation cost. The Ptolemy software environment [1][12] is a natural choice because it can simulate and generate code for a variety of dataflow models of computation. (Ptolemy supports many other models of computation for simulation such as discrete event and finite state machines, but Ptolemy cannot currently generate code for these non-dataflow models of computation.) Ptolemy 0.6 released in April of 1996 contains mature schedulers and code generators for dataflow graphs. Ptolemy 0.7 planned for April of 1997 will have more efficient code generators [7][8]. For Ptolemy 0.7, the authors have added the ability of the code generators to report the data memory usage, program memory usage, and execution time estimates for use with our Heuristic Search Packages and Signal Processing Packages. We are using feedback from Ptolemy as our cost estimates during algorithm rearrangement. When rearranging SDF algorithms, we use the SDF Composition Theorem to decide whether a rearrangement rule applies. The SDF Composition Theorem "establishes four clustering criteria that together provide a sufficient condition that a given clustering [rearrangement] operation involving two adjacent nodes [operators] does not introduce deadlock" [3].

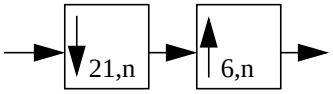
6 Examples

Armed with the rules governing multidimensional multirate signal processing and informed search techniques to apply the rules, we investigate the optimization of signal processing expressions. Since the use of Ptolemy and the Signal Processing Packages and the Heuristic Search Packages are still under development, we will give an example of automatically simplifying an expression using

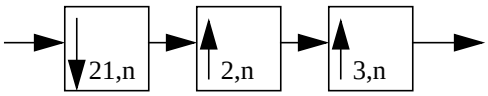
each of the four search techniques. The search techniques will attempt to optimize the following one-dimensional multirate digital signal processing operation:



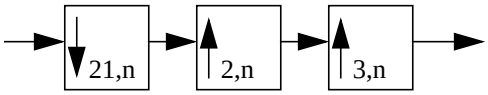
The downarrow represents downsampling, and the uparrow represents upsampling. We will use a simple evaluation function that measures the number of subexpressions. The result of simplifying the above algorithm by the various search techniques is given in Table 2.



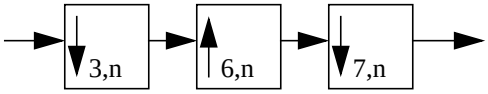
(a) Hill Climbing. Implementation Cost = 2



(b) Breadth-First Search. Implementation Cost = 3.



(c) Depth-First Search. Implementation Cost = 3.



(d) Simulated Annealing. Implementation Cost = 3.

TABLE 2. Evaluation of different search techniques applied to a one-dimensional multirate digital signal processing algorithm.

7 Conclusion

In order to automate the optimization of signal processing algorithms, we need

- comprehensive collection of algebraic identities for signal processing algorithms,
- search mechanisms to apply the rules in an intelligent manner, and
- accurate estimates of implementation cost.

We have implemented the first two as extensions to the Mathematica computer algebra environment. In order to obtain accurate estimates of implementation cost, we rely on the data memory usage, program memory usage, and execution time estimates returned by the Ptolemy software environment.

In our approach, we optimize one-dimensional and multidimensional multirate digital signal processing algorithms. Our approach allows the algorithms to have arbitrary granularity. The algorithms can contain coarse grain computations such as filters and resamplers, as well as fine grain computations such as additions and multiplications. Our approach can support multiple models of computation for the same algorithm, unlike approaches taken by [13][14][19][24][25]. A model of computation does not describe what task a subexpression computes but instead how it interfaces to the rest of the algorithm. In contrast, algebraic rearrangement takes into account the tasks that neighboring subexpressions perform when optimizing the entire algorithm. Combining optimizations performed by scheduling algorithms in loop with the complementary optimizations performed by algebraic rearrangement will produce highly optimize implementations.

Acknowledgments

This research is part of the Ptolemy project, which is supported by the Advanced Research Projects Agency and the U.S. Air Force (under the RASSP program, contract F33615-93-C-1317), Semiconductor Research Corporation (project 94-DC-008), National Science Foundation (MIP-9201605), Office of Naval Technology (via Naval Research Laboratories), the State of California MICRO program, and the following companies: Bell Northern Research, Cadence, Dolby, Hitachi, Mentor Graphics, Mitsubishi, NEC, Pacific Bell, Philips, Rockwell, Sony, and Synopsys.

References

[1] J. Buck, S. Ha, E.A. Lee, and D.G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *International Journal of Computer Simulation, special issue on Simulation Software Development*, vol. 4, 1994. http://ptolemy.eecs.berkeley.edu/papers/JEurSim

[2] E.A. Lee and D.G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, 1987.

[3] J.L. Pino, S.S. Bhattacharyya, and E.A. Lee, "A Hierarchical Multiprocessor Scheduling System for DSP Applications," *Proc. Asilomar Conf. on Signals, Systems, and Computers*, Pacific Grove, Oct. 1995. http://ptolemy.eecs.berkeley.edu/papers/hierStaticSched-asilomar-95

[4] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on

multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, 1992.

[5] V. Sarkar, *Partitioning and scheduling parallel programs for multiprocessors*, Cambridge, Mass.: MIT Press, 1989.

[6] G. C. Sih, *Multiprocessor scheduling to account for interprocessor communication*, Ph.D. Dissertation, UCB/ERL M91/29, Electronics Research Laboratory, University of California at Berkeley, 1991.

[7] S. S. Bhattacharyya, J. T. Buck, S. Ha, and E. A. Lee, "Generating compact code from dataflow specifications of multirate signal processing algorithms," *IEEE Transactions on Circuits and Systems — I: Fundamental Theory and Applications*, March, 1995.

[8] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Two complementary heuristics for translating graphical DSP programs into minimum memory implementations*, Memorandum UCB/ERL M95/3, Electronics Research Laboratory, University of California at Berkeley, January, 1995.

[9] S. Ritz, M. Pankert, and H. Meyr, "Optimum vectorization of scalable synchronous dataflow graphs," *Proceedings of the International Conference on Application-Specific Array Processors*, October, 1993.

[10] J.L. Pino and E.A. Lee, "Hierarchical static scheduling of dataflow graphs onto multiple processors," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Detroit, Michigan, IEEE, 1995. http://ptolemy.eecs.berkeley.edu/papers/hierStaticSched

[11] G. Arslan, B. L. Evans, F. A. Sakarya, and J. L. Pino, "Performance evaluation and real-time implementation of subspace, adaptive, and DFT algorithms for multi-tone detection," in *Proc. IEEE Int. Conf. on Telecommunications*, Istanbul, Turkey, Apr. 1996.

[12] J. L. Pino, S. Ha, E. A. Lee, and J. T. Buck, "Software synthesis for DSP using Ptolemy," *Journal on VLSI Signal Processing*, vol. 9, pp. 7-21, Jan. 1995. (Web: http://ptolemy.eecs.berkeley.edu/papers/interfaceSynthesis/).

[13] C. Myers, "Symbolic representation and manipulation of signals," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Dallas, TX, pp. 2400--2403, Apr. 1987.

[14] M. M. Covell and J. Richardson, "A new, efficient structure for the short-time Fourier transform with an application in code-division sonar imaging," *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 2041-2044, May 1991.

[15] B. L. Evans, R. H. Bamberger, and J. H. McClellan, "Rules for multidimensional multirate structures," *IEEE Transactions on Signal Processing*, vol. 42, no. 4, pp. 762-771, Apr. 1994.

[16] S. Wolfram, Mathematica: A System for Doing Mathematics by Computer. Redwood City, CA: Addison-Wesley, second ed., 1991.

[17] B. L. Evans, *A Knowledge-Based Environment for the Design and Analysis of Multidimensional Multirate Signal Processing Algorithms*. Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, June 1993. (This thesis "evans_thesis.ps.Z" and the Signal Processing Packages are available at ftp://ftp.eedsp.gatech.edu/Mathematica/).

[18] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[19] C. H. Richardson and R. W. Schafer, "A lower bound for structuring element decompositions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 365-369, April, 1991.

[20] B. L. Evans, R. H. Bamberger, and J. H. McClellan, "Rules for multidimensional multirate structures," *IEEE Transactions on Signal Processing*, vol. 42, no. 4, pp. 762-771, Apr. 1994.

[21] S. Wolfram, Mathematica: A System for Doing Mathematics by Computer. Redwood City, CA: Addison-Wesley, second ed., 1991.

[22] B. L. Evans, *A Knowledge-Based Environment for the Design and Analysis of Multidimensional Multirate Signal Processing Algorithms*. Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, June 1993. (This thesis "evans_thesis.ps.Z" and the Signal Processing Packages are available at ftp://ftp.eedsp.gatech.edu/Mathematica/).

[23] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[24] L. Guerra, M. Potkonjak, and J. Rabaey, "Divide-And-Conquer Techniques for Global Throughput Optimization", *Proc. IEEE VLSI Signal Processing Workshop*, San Francisco, CA, Oct. 1996, pp. 137-146,

[25] S. H. Huang and J. Rabaey, "An Integrated Framework for Optimizing Transformations", *Proc. IEEE VLSI Signal Processing Workshop*, San Francisco, CA, Oct. 1996, pp. 263-272.