MINIMIZING THE NUMBER OF OPERATIONS IN DSP COMPUTATIONS

Inki Hong and Miodrag Potkonjak

UCLA Computer Science Department, Los Angeles, CA 90095-1596, USA

ABSTRACT

Reduction of the number of operations optimizes the important design metrics such as area, cost, throughput, and power consumption for both custom ASIC and programmable processor implementations. We propose a novel technique to minimize the number of operations in DSP computations. The first step of the approach logically partitions a computation into strongly connected components. The second step optimizes each component separately. In the third step the components are merged to further optimize. Finally, the components are scheduled to minimize memory consumption. The effectiveness of our approach is demonstrated on real-life examples.

1. INTRODUCTION

Reducing the number of operations needed for a given computation decreases cost, area and power consumption, and increases the throughput of custom datapath ASIC implementations. In the case of programmable processor implementations, the throughput is mostly determined by the number of operations, and power consumption can be decreased through effective voltage scaling technique which is enabled by the extra throughput.

We illustrate the key ideas of our approach for minimizing the number of operations by considering the computation of Figure 1. Each node represents a subpart of the computation. We make the following assumptions only specifically for clarifying the presentation of this simplified example.

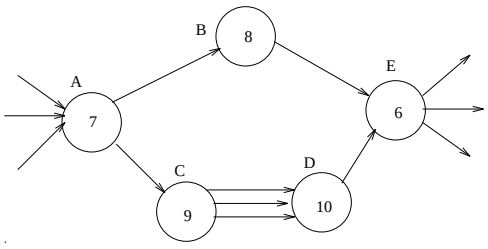

Figure 1. A motivational example

$S_1[n] = 2 * S_1[n-1] + S_2[n-1] + 4 * X[n-1]$
$S_2[n] = 3 * S_1[n-1] + 5 * S_2[n-1] + X[n-1]$
$Y[n] = 2 * S_1[n-1] + 3 * S_2[n-1]$
addition = 5, # multiplication = 6
operations = # addition + # multiplication = 11

Figure 2. A simple example for calculating the number of operations in a maximally fast procedure [7]

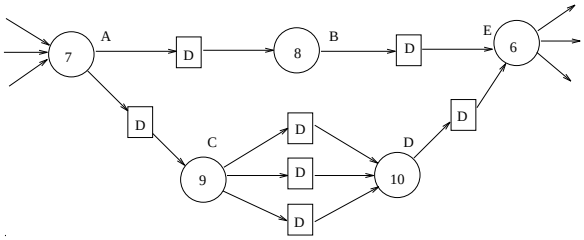

Figure 3. A motivational example after the isolation step

We stress here that the assumptions are not necessary for our approach. We assume that each subpart is linear and dense, which means that every output and state in a subpart are linear combinations of all inputs and states in the subpart with no 0, 1, or -1 coefficients. The number inside a node is the number of delays or states in the subpart. We assume that when there is an arc from a subpart X to a subpart Y, every output and state of Y depend on all inputs and states of X.

The number of operations per input sample is initially 2081 (We illustrate how the number of operations is calculated in a maximally fast procedure [7] using a simple linear computation with 2 states and 1 output which is described in Figure 2). Using the technique of [10] which unfolds the entire computation, the number can be reduced to 725 with an unfolding factor of 12. Our approach can optimize each subpart separately, which is enabled from isolating the subparts using pipeline delays. The Figure 3 shows the resulting computation after the isolation step. Separate op-

timization step results in 522.27 operations. We perform subparts merging to further optimize. If the subparts C and D are merged and optimized together, the number of operations is further reduced to 399.4. The approach has reduced the number of operations by a factor of 1.82(5.2) from the previous technique of [10] (from the initial number of operations).

The main technical innovation of the research presented in this paper is the first approach for the minimization of the number of operations in general computations. The approach does not treat just significantly wide set of computations than the other previously published techniques [10], but also outperforms or performs at least as well as other techniques on all examples.

The rest of the paper is organized in the following way. In Section 2. we briefly review the related work on the minimization of the number of operations. Section 3. presents the key idea of the new approach and describes optimization techniques for the approach. Section 4. illustrates the effectiveness of the technique using real-life examples. Finally, Section 5. draws conclusions.

2. RELATED WORK

In this section, we briefly review the related work on the minimization of the number of operations. Potkonjak and Rabaey [7] addressed the minimization of the number of multiplications and additions in linear computations in their maximally fast form so that the throughput is preserved. Potkonjak et al. [8] presented a set of techniques for minimization of the number of shifts and additions in linear computations. Sheliga and Sha [9] presented an approach for minimization of the number of multiplications and additions in linear computations. Srivastava and Potkonjak [10] developed an approach for the minimization of the number of operations in linear computations using unfolding and the application of the maximally fast procedure. Guerra et al. [2] developed a divide and conquer approach for minimizing critical paths.

3. OPTIMIZATION APPROACH

The core of the approach is presented in the pseudo-code of Figure 4. The rest of this section explains the global flow of the approach in more detail.

The first step of the approach is to identify the computation's strongly connected components(SCCs), using the standard depth-first search-based algorithm [11]. For any pair of operations A and B within a SCC, there exist both a path from A to B and one from B to A. The SCCs are isolated from each other using pipeline delays, which enables us to optimize each subpart separately. The inserted

```
Decompose a computation into strongly connected
components(SCCs);
Use pipelining to isolate the SCCs;
Minimize the number of delays using retiming;
For each of SCCs
  If (the SCC is linear)
    Apply unfolding;
  Else
    Isolate nonlinear operations;
    Decompose the linear subpart into SCCs;
    Apply unfolding to decomposed linear SCCs;
Merge linear SCCs to further optimize;
Schedule subparts to minimize memory usage;
```

Figure 4. The core of the approach

$i_{opt} = \lfloor \sqrt{\frac{(2R-1)R}{PQ}} - 1 \rfloor, or \lceil \sqrt{\frac{(2R-1)R}{PQ}} - 1 \rceil$, which gives smaller value of $i_{opt}(PQ - \frac{R(2R-1)}{i_{opt}+1})$.
$N(*, i) = \#$ multiplications for i times unfolded system $= R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ$
$N(+, i) = \#$ additions for i times unfolded system $= R^2 + (i+1)PR + (i+1)QR + \frac{(i+1)(i+2)}{2}PQ - R - (i+1)Q$

Figure 5. Closed-form formula of unfolding for dense linear computation with P input, Q output, and R states.

pipeline delays are treated as a subpart input or output. As a result, every output and state in a subpart depend only on the subpart's inputs and states. Note that this isolation is not affected by unfolding.

In the next step, the number of delays in the computation is minimized using retiming by the Leiserson-Saxe algorithm [6]. It is obvious that smaller number of delays will require smaller number of operations since both the next states and outputs depend on the previous states.

The SCCs are further classified as either linear or nonlinear. Linear computations can be represented using the following state-space equations: $S[n] = AS[n-1] + BX[n], Y[n] = CS[n-1] + DX[n]$, where X, Y, and S are the input, output, and state vectors respectively and A, B, C, and D are constant coefficient matrices. We have used an approach of [10] for optimization of linear SCCs, which uses unfolding and the maximally fast procedure [7]. We note that instead of maximally fast procedure the ratio analysis by [9] can be used. [10] has provided the closed-form formula for the optimal unfolding factor with the assumption of dense linear computations which are provided in Figure

5. For sparse linear computations, they have proposed a heuristic which continues to unfold further until there is no improvement.

When a SCC is classified as nonlinear, all nonlinear operations are isolated from the SCC so that the remaining linear subparts can be optimized. All arcs from nonlinear operations to the linear subparts are considered as inputs to the linear subparts, and all arcs from linear subparts to the nonlinear operations considered as outputs from the linear subparts. The linear subparts are logically partitioned into SCCs and each SCC is optimized by the same approach in the previous paragraph.

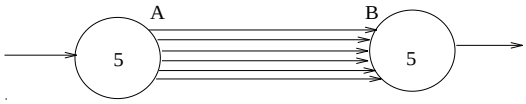

Figure 6. A motivational example for SCC merging

$$
\begin{aligned}
&S[n+i] = A^{i+1}S[n-1] + A^iBX[n] + \\
&A^{i-1}BX[n+1] + ... + BX[n+i] \\
&Y[n] = CS[n-1] + DX[n] \\
&Y[n+1] = CAS[n-1] + CBX[n] + DX[n+1] \\
&\qquad ... \\
&Y[n+i] = CA^iS[n-1] + CA^{i-1}BX[n] + \\
&CA^{i-2}BX[n+1] + ... + CBX[n+i-1] + DX[n+i]
\end{aligned}
$$

Figure 7. i times unfolded state-space equations

Sometimes it is beneficial to decompose a computation into larger subparts than SCCs. We consider an example given in Figure 6. We use the same assumptions made for the motivational example in Section 1.. Separately optimizing SCCs A and B costs 211 operations while optimizing the entire computation produces 63.67 operations. The reason why separate optimization does not perform well in this example is because there are too many intermediate outputs from SCC A to SCC B. The observation leads us to develop an approach of merging SCCs for further reduction of the number of operations.

Initially, we only consider merging of SCCs. When two SCCs are merged, however, the merged SCCs does not form a SCC. Thus, in general, we must consider merging of any adjacent arbitrary subparts. Suppose we consider merging of subparts i and j. The gain $GAIN(i,j)$ of merging subparts i and j can be computed as follows; $GAIN(i,j) = COST(i) + COST(j) - COST(i,j)$, where $COST(i)$ is the number of operations for subpart i and $COST(i,j)$ is the number of operations for the merged subpart of i and j. To compute the gain, $COST(i,j)$ must be computed, which requires to get constant coefficient matrices A, B, C, and D for only the merged subpart of i and j. It is easy to construct the matrices using the depth-first search [11]. The i times unfolded system can be represented by the state-space equations in Figure 7. From the equations, the total number of operations can be computed for i times unfolded subpart. Let $N(*,i)$ and $N(+,i)$ denote the number of multiplications and the number of additions for i times unfolded system respectively. The resulting number of operations $\frac{N(*,i)+N(+,i)}{i+1}$ because i times unfolded system uses a batch of $i+1$ input samples to generate a batch of $i+1$ output samples. We continue to unfold further until no improvement is achieved. If there are no coefficients of 1 or -1 in the matrices A, B, C, and D, then closed-form formula of the optimal unfolding factor i_{opt} and of the number of operations for i times unfolded system can be obtained [3]. The formula are provided in Figure 8.

$$
\begin{aligned}
&i_{opt} = \lfloor \sqrt{\frac{\sum_{j=1}^{n} S_j(2SS_j-1)}{\sum_{j=1}^{m} O_jIO_j}} - 1 \rfloor, \textit{ or} \\
&\lceil \sqrt{\frac{\sum_{j=1}^{n} S_j(2SS_j-1)}{\sum_{j=1}^{m} O_jIO_j}} - 1 \rceil, \text{ which gives smaller value of} \\
&i_{opt}(\sum_{j=1}^{m} O_jIO_j - \frac{\sum_{j=1}^{n} S_j(2SS_j-1)}{i_{opt}+1}). \\
&N(*,i) = \sum_{j=1}^{n}(S_jSS_j + (i+1)S_jIS_j) + \\
&\sum_{j=1}^{m}((i+1)O_jSO_j + \frac{(i+1)(i+2)}{2}O_jIO_j) \\
&N(+,i) = \sum_{j=1}^{n}(S_j(SS_j-1) + (i+1)S_jIS_j) + \\
&\sum_{j=1}^{m}((i+1)O_j(SO_j-1) + \frac{(i+1)(i+2)}{2}O_jIO_j) \\
&\text{, where } m = \#\text{ output groups, } n = \#\text{ state groups} \\
&S_j = \#\text{ states in state group j} \\
&O_j = \#\text{ outputs in output group j} \\
&IO_j = \#\text{ inputs that output group j depends on} \\
&IS_j = \#\text{ inputs that state group j depends on} \\
&SO_j = \#\text{ states that output group j depends on} \\
&SS_j = \#\text{ states that state group j depends on}
\end{aligned}
$$

Figure 8. Closed-form formula for unfolding; If two outputs depend on the same set of inputs and states, they are in the same group, and the same is true for states.

```
While (there is improvement)
  For all possible merging candidates,
    Compute the gain;
  Merge the pair with the highest gain;
```

Figure 9. A pseudo-code of a greedy heuristic

Now, we can evaluate possible merging candidates. We propose two heuristic algorithms for SCC merging. The first heuristic is based on greedy optimization approach.

Design	Init Ops	[10]	New Method	Imp.
DAC	2098	2098	1327.83	1.58
modem	213	213	148.83	1.43
GE controller	180	180	105.26	1.71
APCM receiver	2238	N/A	1444.19	1.55
Audio Filter	228	N/A	92.0	2.48
Video Filter	398	N/A	184.5	2.16

Table 1. Experimental results for real-life examples

The pseudo-code is provided in Figure 9. The algorithm is simple. Until there is no improvement, merge the pair of subparts which produces the highest gain. The other heuristic algorithm is based on a general combinatorial optimization technique known as simulated annealing [4].

Since the subparts of a computation are unfolded separately by different unfolding factors, we need to address the problem of scheduling of the subparts. They should be scheduled so that memory requirements for code and data of a schedule are minimized. We observe that the unfolded subparts can be represented by multi-rate synchronous dataflow graph [5] and the works of [1] can be directly used.

4. EXPERIMENTAL RESULTS

This section presents the experimental results of our technique for real-life examples, where Table 1 summarizes the results. Our set of benchmark designs include the following typical portable DSP, video, communication, and control applications: DAC - 4 stage NEC digital to analog converter (DAC) for audio signals; modem - 2 stage NEC modem; GE controller - 5-state GE linear controller; APCM receiver - Motorola's adaptive pulse code modulation receiver; Audio Filter - analog to digital converter (ADC) followed by 18 order parallel filter; and Video Filter - two ADCs followed by 12-order two dimensional (2D) IIR filter. DAC, modem, and GE controller are linear computations and the rest are nonlinear computations. The fifth column of Table 1 provides only the improvement factor of our method from the initial number of operations since [10] is either ineffective or inapplicable for all examples. Our method has reduced the number of operations by an average factor of 1.82 (average 42.9 %) for the examples, which clearly indicates the effectiveness of our new method.

5. CONCLUSION

We proposed a novel technique to minimize the number of operations in DSP computations. The effectiveness of our approach was demonstrated on real-life examples. Our method has reduced the number of operations by an average factor of 1.82 (average 42.9 %) for the examples that previous techniques are either ineffective or inapplicable.

REFERENCES

[1] S. S. Bhattacharyya et al,"A scheduling framework for minimizing memory requirements of multirate signal processing algorithms expressed as dataflow graphs,", VLSI Signal Processing VI, pp. 188-196, 1993.

[2] L. Guerra, M. Potkonjak, J. Rabaey, "Divide-and-Conquer Techniques for Global Throughput Optimization", VLSI Signal Processing Workshop, pp. 137-146, San Francisco, CA, October 1996.

[3] I. Hong, M. Potkonjak,"Minimizing the Number of Operations in DSP Computations", UCLA CSD Tech. Rep. (to appear).

[4] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, pp. 671-680, 1983.

[5] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow" , Proceedings of the IEEE, Vol. 75, No. 9, pp. 1235-1245, 1987.

[6] C. E. Leiserson, J. B. Saxe, "Retiming synchronous circuitry," Algorithmica, Vol. 6, No. 1, pp. 5-35, 1991.

[7] M. Potkonjak, J. Rabaey, "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations," IEEE International Conference on Computer-Aided Design, pp. 304-308, 1992.

[8] M. Potkonjak et al, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination", IEEE Trans. on CAD, Vol. 15, No. 2, pp. 151-165, 1996.

[9] M. Sheliga, E.H.-M. Sha, "Global node reduction of linear systems using ratio analysis", International Symposium on High-Level Synthesis, pp. 140-145, 1994.

[10] M. Srivastava, M. Potkonjak, "Power optimization in programmable processors and ASIC implementations of linear systems: transformation-based approach," Design Automation Conference, pp. 343-348, 1996.

[11] R. E. Tarjan, "Depth first search and linear graph algorithms," SIAM Journal on Computing, Vol. 1, No. 2, pp. 146-160, 1972.