

Maximally Fast and Arbitrarily Fast Implementation of Linear Computations

Miodrag Potkonjak

C&C Research Laboratories, NEC USA, Princeton, NJ

Jan Rabaey

Department of EECS, University of California, Berkeley, CA

ABSTRACT: Linear systems are the most often used type of systems in many engineering and scientific areas. By establishing a relationship between the basic properties of linear computations and several optimizing transformations, it is possible to optimally speed-up linear computations with respect to those transformations while keeping the latency fixed. Furthermore, arbitrarily fast, asymptotically optimal implementations can be obtained by adding retiming and loop unrolling to the transformations set and trading latency for throughput. The proposed techniques have yielded results superior to the best published previously on all benchmark examples. Finally, the presented approach is also applicable to general (non-linear) computations.

1.0 Motivation and Prior Art

The major goal of this paper is to demonstrate how for the large class of linear computations the maximally fast implementation with respect to five important and powerful transformations (associativity, distributivity, commutativity, common subexpression and constant propagation) can be efficiently derived and to show how an arbitrarily fast, asymptotically optimal (with respect to the hardware cost) implementation of a general linear computation can be produced combining those five transformations with retiming and loop unfolding.

Transformations alter the organization of a computation in a such a way that the user specified input/output relationship is maintained. They are often used as an effective approach for the improvement of the implementation of computations. Their use in compilers [1,5], theoretical computer science [2] and high level synthesis [3, 7, 9, 11] is surveyed in [10].

2.0 Main Ideas and Introductory Examples

We will introduce main ideas using simple, yet non-trivial examples. Figure 1 shows the data-control flow graph (DCFG) of a 3-port serial adaptor which is regularly used in many ladder digital filter structures [4]. Since it is often a part of the feedback path of the complete filter and therefore cannot be pipelined, it usually dictates the overall throughput. Its critical path equals 7 clock cycles, assuming that each operation takes one cycle.

Applying algebraic transformations will only have a limited success. Certain transformations, which could lead to faster implementations are initially inhibited: for instance, associativity can only be applied when the intermediate variable does not have any extra fanout. Our goal is to remove all those inhibiting factors first and hence enable other transformations. This can be achieved in the following way:

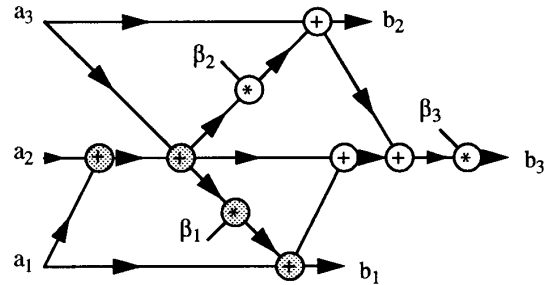


FIGURE 1. DCFG of a 3-port serial adaptor

$$\begin{aligned}
 b_1 &= \beta_1 * (a_1 + a_2 + a_3) + a_1 \\
 b_2 &= \beta_2 * (a_1 + a_2 + a_3) + a_1 \\
 b_3 &= \beta_3 * (\beta_1 * (a_1 + a_2 + a_3) + \beta_2 * (a_1 + a_2 + a_3) + \\
 &\quad \beta_1 * (a_1 + a_2 + a_3) + a_1 + \beta_2 * (a_1 + a_2 + a_3) + a_1)
 \end{aligned}$$

FIGURE 2. Functional dependences between output and input nodes

$$\begin{aligned}
 b_1 &= (\beta_1 + 1) * a_1 + \beta_1 * a_2 + \beta_1 * a_3 \\
 b_2 &= \beta_2 * a_1 + (\beta_2 + 1) * a_2 + \beta_2 * a_3 \\
 b_3 &= \beta_3 * (2 * \beta_1 + 2 * \beta_2 + 1) * a_1 + \beta_3 * \\
 &\quad (2 * \beta_1 + 2 * \beta_2 + 1) * a_2 + 2 * \beta_3 * (\beta_1 + \beta_2) * a_3
 \end{aligned}$$

FIGURE 3. Functional dependences after the application of several transformations

The nodes which are in transitive fan-in of the output b_1 are shaded in Figure 1. Obviously, in order to correctly compute this output it is sufficient to take into account only this part of the computation. Figure 2 shows the functional dependencies between all output nodes b_i and input nodes a_i . To get from the structure of Figure 1 to the result of Figure 3, we have effectively replicated all subexpressions, common between the different output variables and hence made the computational trees independent. The resulting DCFG has only additions and multiplications with constants as computational nodes. So, each output is a linear combination (weighted sum) of the inputs. This observation directly leads to a significant reduction in the length of critical path

Using simple algebraic manipulations, followed by the use

of constant propagation to precompute all input weights, the dependencies of Figure 3 are obtained. The DCFG of the restructured computation is presented in Figure 4. The critical path is reduced to 3 clock cycles. Although the speed-up is impressive, it can be achieved in a simple and effective way.

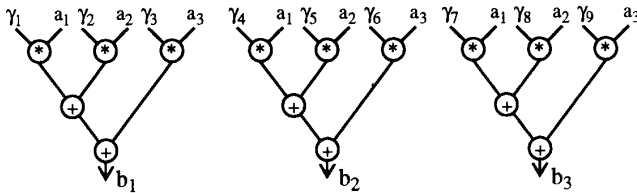


FIGURE 4. DCFG for the restructured adapter. Values for γ_i can be easily obtained from Figure 3. Note that some of them are 1, for those multiplications are not needed

If the application allows for the introduction of additional latency, the throughput can be increased even more by extending the transformational set. This is even true when the designs have feedback loops which prevent the use of pipelining.

Before Unfolding:

$$\begin{aligned} x_1 &= In_1 + a * x_0 + b * y_0 \\ y_1 &= x_0 \\ Out_1 &= x_1 + c * x_0 + d * y_0 \end{aligned} \quad (a)$$

After Unfolding:

$$\begin{aligned} x_1 &= In_1 + a * x_0 + b * y_0 & x_2 &= In_2 + a * x_1 + b * y_1 \\ y_1 &= x_0 & y_2 &= x_1 \\ Out_2 &= x_2 + c * x_1 + d * y_1 \end{aligned} \quad (b)$$

After Transformations:

$$\begin{aligned} x_2 &= In_2 + a * In_1 + (a * a + b) x_0 + a * b * y_0 \\ y_2 &= In_1 + a * x_0 + b * y_0 \\ Out_2 &= In_2 + (a + c) In_1 + (a * a + b + c * a + d) x_0 + \\ &\quad + (a * b + c * b) y_0 \end{aligned} \quad (c)$$

FIGURE 5. Functional dependences for 2nd order IIR filter before and after unfolding and transformations

To convey the ideas behind the procedure, we again will use a small, but real life example, as shown in Figure 6. This figure shows the second order IIR filter. The filter has two feedback loops, which prevent the direct use of pipelining or (time) loop unfolding. Figure 5a shows the analytical expressions of the computations to be performed in a single iteration. A single unfolding of time loop (corresponding to the substitution of one equation into the next) results into analytical expression of Figure 5b. The DCFG of the unfolded filter is shown in Figure 7. The advantage of the new structure

is that two iterations can be computed simultaneously. Unfortunately the critical path of the unfolded structure is doubled, therefore the throughput remains unchanged.

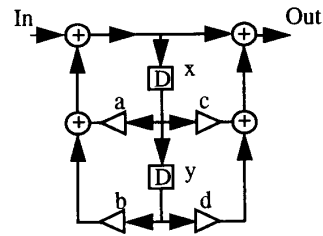


FIGURE 6. Second order IIR filter

However, by transforming the graph using common sub-expression replication, algebraic transformations and constant propagation, the equations can be reorganized as shown in Figure 5c. Notice that the graph of Figure 5c only captures the odd cycles. An identical structure is needed to compute the even samples in parallel. That effectively doubles the amount of hardware needed. On the other hand, the minimal execution time for this graph is only 3 clock cycles, which is identical to the critical path of the original graph of Figure 5a. As a result, the throughput has been actually doubled as two iterations can be performed in the time initially needed for one.

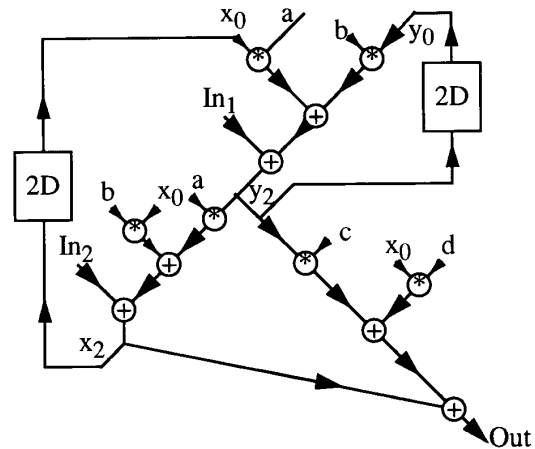


FIGURE 7. DCFG of the unfolded IIR filter

Of course, we can continue with the loop unrolling and the subsequent reduction of the critical path. The crucial observation is that regardless of the number of times the initial iteration is unfolded, each output will depend on only two initial states and as many inputs as the number of unfoldings. All primary inputs are out of loops, and can be pipelined. So, after organizing all those computations so that all multiplica-

tions are done in the first cycle, followed by additions organized in a tree format, the critical path is $\lceil \log NS + 1 \rceil$, where NS is the number of initial states. If the basic iteration is unfolded N times, the number of simultaneously processed samples increases linearly, while the critical path is not altered. Therefore the throughput increases at the rate N . So an arbitrary fast implementation can be achieved by using the appropriate number of unrollings.

3.0 Linear Systems

Linear systems have “the widest signal processing application range” [8]. The basic facts about linear systems can be summarized by the following definition.

A linear system is a system L such that:

- (1) If the response to a signal x is a signal y , then the response to ax is ay , where a is an arbitrary number (homogeneity or scaling property);
- (2) If the responses to x_1 and x_2 are y_1 and y_2 , respectively, then the response to $x_1 + x_2$ is $y_1 + y_2$ (additivity property).

Linear computation uses as computational elements only addition, subtraction and multiplication with constants.

4.0 Algorithms

Due to dramatic improvements in silicon technology, design for minimal area has become less stringent recently and trading off area for performance has become a viable alternative. Although the primary goal of the new approach is to maximize throughput, several other factors cannot be ignored. Among them is the area of design, which is treated as the secondary goal. In this section we will present algorithms for: (1) Maximally fast implementation of linear computations under latency constraints and (2) Arbitrarily fast implementation of linear computations. We will also discuss how the number of operations in the final design can be minimized.

4.1 Fast Implementation of Linear Computations

The critical path minimization algorithm can be formalized with the following pseudocode.

Maximally fast implementation of Linear Computations:

- (1) Express each output node as linear combination of the input nodes;
- (2) Compute the values of all output nodes using the fastest tree structure;
- (3) Minimize the number of operation in trees by using algorithm from subsection 4.3.

The first step of the algorithm uses common subexpression replication as the enabling transformation for the critical path minimization, which is conducted during the second step. In order to figure out how all output nodes depend on one particular input node, the value 1 is assigned to this input

node and the value 0 to all other inputs nodes. The second step is the well known transformation from chain of additions to binary tree structure using associativity. The first step acts as an enabler for this procedure.

The last step does not influence the length of the critical path. It is used for the optimization of the secondary goal, area. This step is explained in the subsection 4.3.

It can be easily derived that the run-time of the algorithm is quadratic in the number of nodes. Theorem 1 establishes the important property that the transformed design, as obtained from the proposed algorithm, is the fastest possible. The proof of all theorems presented in this paper can be found in [10].

Theorem 1: The implementation provided by the proposed algorithm is maximally fast.

4.2 Arbitrarily Fast Implementation of Linear Computations

When it is allowed to introduce additional latency into the design, we can additionally reduce the critical path by trading latency for throughput. An efficient algorithm for this problem can be described using the following pseudocode:

Arbitrarily Fast Implementation of Linear Computations:

- (1) Minimize the number of delay s using retiming.
- (2) Unfold the basic body of the iteration N times (as dictated by latency other user-defined constraints).
- (3) Using the algorithm from subsection 4.1 minimize the critical path in the unfolded structure.

The first step can be solved using the Leiserson-Saxe retiming algorithm [6]. After the unfolding by the factor N of the basic iteration, the outputs in the transformed design will linearly depend on the inputs in N iterations and the states (delays). The second step, unfolding of the initial iteration over time loop, is straightforward and no optimization is involved. The important point is that when we want to compute all sample outputs (as it is required in almost all ASIC designs), it is necessary to replicate hardware as many times, N , as the initial iteration is unfolded. Each of those replications computes one of the N samples in parallel. The final step is the direct application of the algorithm described in the previous subsection. The run time of the algorithm for arbitrarily fast implementation of linear computation is also quadratic.

The major properties of the algorithm are summarized in the following theorems:

Theorem 2: The ratio of the initial and the final AT^2 product is constant for an arbitrary speed improvement.

Theorem 3: The proposed algorithm produces the design which has asymptotically the minimum critical path among all designs with the same latency.

4.3 Minimization of the Number of Operations

While targeting speed as a primary goal, keeping the area of the design as small as possible is definitely important as well. The additional use of two transformations, common subexpression elimination and distributivity, can reduce the number of multiplications and additions/subtractions significantly. As a result, the final implementation does often not have only a high throughput, but also small area.

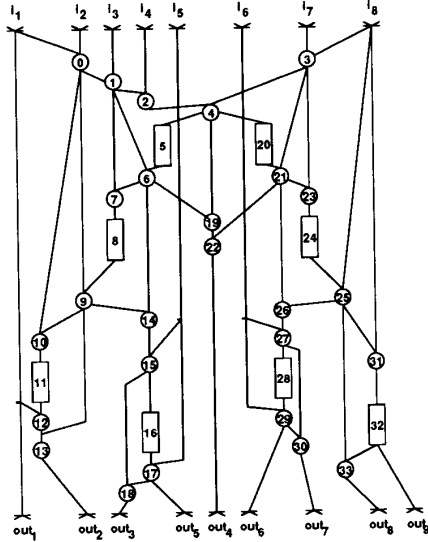


FIGURE 8. Fifth Order Wave Elliptical Filter

We will use 5th order wave digital elliptical (5WD) filter as an example. Figure 8 shows the computational graph of this popular benchmark. Figure 9 shows for each output O_i the inputs I_j on which it depends. By counting operations we can conclude that 53 multiplications and 44 additions are needed. Figure 9 show also the functional dependencies between outputs and inputs (assuming that weighting factors for multiplications 5, 20, 8, 24, 11, 16, 28 and 32 (see Figure 8) are set to the values of 2, 3, 4, 5, 6, 7, 8, 9¹ respectively) after the application of distributivity for all inputs, which are multiplied with the same weighing coefficient in the final implementation. Also note, that several intermediate sums (e.g. $i_1 + i_2 + i_3 + i_4$ in the outputs o_6, o_7, o_8) are the same and have to be computed only once (common subexpression elimination).

Therefore, a simple algorithm which first minimizes the number of multiplications using distributivity for each output and then uses common subexpression elimination on the intermediate result for the different outputs, can significantly

1. Those values have no physical meaning.

reduce the number of operations in the transformed design.

$$\begin{aligned}
 o_1 &= i_1; \\
 o_2 &= 126 * i_1 + 125 * i_2 + 112 * i_3 + 56 * (i_4 + i_7 + i_8) \\
 o_3 &= 160 * (i_1 + i_2) + 152 * i_3 + 9 * i_5 + 80 * (i_4 + i_7 + i_8) \\
 o_4 &= 7 * (i_1 + i_2 + i_3 + i_7 + i_8) + 6 * i_4 \\
 o_5 &= 140 * (i_1 + i_2) + 133 * i_3 + 8 * i_5 + 70 * (i_4 + i_7 + i_8) \\
 o_6 &= 144 * (i_1 + i_2 + i_3 + i_4) + 9 * i_6 + 232 * i_7 + 240 * i_8 \\
 o_7 &= 162 * (i_1 + i_2 + i_3 + i_4) + 10 * i_6 + 261 * i_7 + 270 * i_8 \\
 o_8 &= 150 * (i_1 + i_2 + i_3 + i_4) + 250 * i_7 + 269 * i_8 \\
 o_9 &= 135 * (i_1 + i_2 + i_3 + i_4) + 225 * i_7 + 243 * i_8
 \end{aligned}$$

FIGURE 9. Functional dependences after the application of distributivity

For this example, only 27 multiplications and 22 additions are finally needed. The critical path is 5 cycles (assuming that a multiplication takes 2 cycles). The original implementation uses 8 multiplications and 26 additions and has a critical path of 17 cycles. The number of additions is smaller then in the initial design. The number of multiplications increased but at a slower rate then the reduction in the critical path. Therefore, although our primary goal was the speed, the AT product is also improved for this example.

5.0 Experimental Results

The algorithms presented here are of polynomial complexity (actually most often quadratic) and guarantee the optimum solution. So the major question is not the validation of algo-

| Example | ICP | CPAF | IA | IM | FA | FM |
|---------------------------------|-----|------|----|----|----|----|
| 7th order IIR | 8 | 3 | 13 | 13 | 13 | 13 |
| 5th WDF | 14 | 4 | 26 | 8 | 21 | 20 |
| speech filter | 11 | 3 | 21 | 21 | 21 | 19 |
| 2nd order Volterra ² | 12 | 3 | 17 | 10 | 17 | 10 |

TABLE 1. Fast Implementation of Linear Programs

ICP - initial critical path; CPAF - critical path in the final design, IA and IM - the number of additions and multiplications in the initial implementation, FA and FM - the number of additions and multiplications in the final implementation

gorithms, but how much improvement can be achieved using the concept

The critical path before and after the application of the algo-

2. An example of non-linear computation

rithm for the maximally fast implementation of linear computations on a number of benchmarks is shown in Table 1 (It is assumed here that both addition and multiplication take one clock-cycle).

| Unfolding Factor | Effective CP | Effective CP |
|------------------|--------------|--------------|
| 1 | 4 | 5 |
| 2 | 2 | 2.5 |
| 5 | 0.8 | 1 |
| 10 | 0.4 | 0.5 |
| 20 | 0.2 | 0.25 |
| 50 | 0.08 | 0.1 |
| 100 | 0.04 | 0.05 |

TABLE 2. Arbitrarily fast implementation for the 5th order wave digital elliptical filter

The effects of the use of the algorithm for arbitrarily fast implementation are shown in Table 2 for the 5th order wave digital elliptical filter. The first column shows how many times the initial iteration is unfolded and the second the effective critical for one iteration after unfolding, assuming that all operations take one cycle. If we assume that a multiplication takes two cycles and an addition one, then the effective critical path for the unfolded design is shown in the last column.

6.0 Nonlinear Computations

A widely used technique in many engineering and scientific fields is to approximate nonlinear system by piecewise linear systems. The effectiveness of this approach is correlated to the level of non-linearity in the system.

A similar approach can be used for the techniques presented in this paper. A measure of non-linearity of computation can be introduced using a recursive approach. The non-linearity level of the input nodes is 0. Any time when, while traversing the DCFG in a reverse topological order, a node is visited which is not an addition (subtraction) or a multiplication with a constant, the non-linearity level of the node is calculated as a sum of the levels of its inputs increased by one. Otherwise the non-linearity level for the node is the maximum value for the non-linearity of its inputs.

The goal of the preprocessing step is now modified such that the maximum non-linearity level of all outputs is minimized. After that, the original algorithms are applied on the linear subparts.

Although this approach is conceptually simple, it is surpris-

ingly effective, as shown in Table 1 on the 2nd order Volterra filter. The similarly modified algorithm for arbitrarily fast implementation can be used for the further reduction of the critical path, by trading off latency.

7.0 Conclusion

Using the relationship between the superposition property for linear computations and the set of optimizing transformations, algorithms for the transformation of arbitrary linear computations to a maximally fast implementation for a fixed latency, and arbitrarily fast implementations have been developed. The proposed techniques produced for all used benchmark examples results superior to the best published previously. For example, the critical path of the 5th order wave digital elliptical filter was reduced from the initial 17 cycles to only 5 cycles without introducing any additional latency (this compared to the previous best of 12 cycles). We presented how the new approach can be modified so that it is applicable to general computation. The detail description of the new approach, including the properties of transformed designs, can be found in [10].

8.0 References

- [1] A.V. Aho, J.D. Ullman: "Principles of Compiler Design", Addison-Wesley, Reading, MA, 1977.
- [2] A. Borodin, I. Munro: "The Computational Complexity of Algebraic and Numeric Problems", American Elsevier Pub, New York, NY, 1975.
- [3] R. Camposano, R. Walker: "A Survey of high-level synthesis systems", Kluwer, Boston, 1991.
- [4] A. Fettweis: "Digital Filter Structures Related to Classical Filter Network", Archiv Electronic Ubertragungstechnik, Vol. 25, pp. 79-89, 1971.
- [5] C.N. Fischer, R.J. Le Blank: "Crafting a Compiler", The Benjamin/Cummings Publishing Co., Menlo Park, CA, 1985.
- [6] C.E. Leiserson, J.B. Saxe: "Retiming Synchronous Circuitry", Algorithmica, Vol. 6, pp. 5-35, 1991
- [7] D.A. Lobo, B.M. Pangrle: "Redundant Operation Creation: A Scheduling Optimization Technique", 28th ACM/IEEE Design Automation Conference, pp. 775-778, 1991
- [8] A.V. Oppenheim, R.W. Shafer: "Discrete-time Signal Processing", Prentice Hall, Englewood Cliffs, NJ, 1989.
- [9] M. Potkonjak, J. Rabaey: "Optimizing Resource Utilization Using Transformations", *IEEE ICCAD91*, pp. 88-91, 1991.
- [10] M. Potkonjak, J. Rabaey: "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations", NEC Technical Report #92-551004, 1992.
- [11] H. Trickey: "Flamel: A high-Level Hardware Compiler", IEEE Transaction on CAD, Vol. 6, No. 2, pp. 259-269, 1987.
- [12] J.D. Ullman: "Computational Aspects of VLSI", Computer Science Press, Rockville, MD, 1984.