International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 12 (2015) pp. 31871-31882

http://www.ripublication.com

Loop Transformation for High Level Synthesis of Iterative Algorithms

E.S.Preethi and B. Bala Tripura Sundari

Electronics and Communication Department
Amrita Vishwa Vidyapeetham, Coimbatore, India
espreethi.90@gmail.com
Electronics and Communication Department
Amrita Vishwa Vidyapeetham, Coimbatore, India
b_bala@cb.amrita.edu

Abstract

Digital signal processing applications (DSP) algorithms are iterative in nature and computationally intensive. Such computation-intensive procedures are represented by recursive equations and dependence graphs (DGs). Loops are the primary source of parallelism in iterative algorithm. For the purpose of throughput enhancement, loop transformation methods are commonly used in high-level synthesis. One of the most effective transformation techniques, named retiming, is a structural transformation that relocates the delays or registers in a circuit. This reduces the latency of the circuit without changing its functionality. Unfolding is another transformation technique used to improve the throughput of the system which when applied to DSP results in multiple iterations of the original program. Unfolding the program can also unearth buried concurrencies, leading to a drop in the iteration period and a proportionate increase in the throughput. The unfolding transformation is incorporated along with retimed DG to improve the parallel processing of the system. In this paper we have automated the modeling of DG and their transformation using high level language JAVA. The enhanced graphical user interface (GUI) and superior memory allocation system of JAVA make it ideal for such an application and for the realization of RTL generation of the unfolded and retimed DG of the benchmark circuits considered as the next phase of this work.

Keywords-dependence graph; retiming; unfolding; DSP algorithm; recursive algorithm; iterative algorithm; transformation technique; high level synthesis

I. INTRODUCTION

DSP is perhaps the most critical enabling technology behind the last few decade's communication and multi-media revolutions. It is used in numerous real time applications related with very large scale integration (VLSI) technology, such as wireless communications, transmission systems, multimedia, digital video, digital audio and radar systems[1].

DGs [1] are directed graphs that portray the dependency of the computations in a process. They can show multiple interdependent iterations in a simple manner. Thus they are ideal candidates to display DSP application flows or algorithms, which are intrinsically computation-intensive and iterative in nature. The nodes in the graph represent the combinational logic and the weights denote the propagation delay for the nodes or the number of register for the edges.

Loops are highly time thirsty parts of a DSP application, making them the primary target when ramping up the efficiency of a process which is where loop transformation techniques (LTT) [2] step in. One of the most prevalent LTT is retiming. It relocates the delays or registers within a path without altering the functionality. Repositioning the delays or registers helps in minimizing the latency of the circuit. DSP can also be transformed through the unfolding method. Here multiple iterations of the same program are related to increase throughput.

In this paper we are integrating the retiming and unfolding LTT. The former will reduce the latency thereby producing faster response and the latter will unroll the loops to improve the parallelism of the system. Together they will increase the throughput of the system. This will be implemented through JAVA programming.

This paper is structured as follows. The Section 2 gives a summary on literature survey. Section 3 highlights methodologies to transform the DG representation of DSP circuits to loop transformed circuits. Section 4 describes the implementation details. Section 5 depicts the results and the last section concludes the paper.

II. LITERATURE SURVEY

High level synthesis (HLS) is the technique of transforming behavioral description to a structural level specification. In the behavior description, the input and output behavior is defined in terms of data transfers and processes without any implementation details. Structural description maps these operations and data transfers into combinational functional block and registers on to hardware. The high level synthesis of DSP algorithms is necessary as it reduces time to market window. [3]

In HLS flow the conclusion of compilation gives rise to the scheduled dataflow graph (DFG) which in turn is representative of the application itself. DFGs transformed through resource allocation and hardware binding. Resource allocation consists of but is not limited to functional units (FU) and multiplexers. If sharing is one of the aims of design then hardware binding would involve bundling together processes in the scheduled data flow to FU's and then distributing the FU's through MUXs [4, 5].

For optimization purposes LTTs are also commonly used in HLS. Various optimization methods are available in literature [6]. DSP algorithms are repetitive in nature and periodical iterations must be repeated to execute the computations [7]. Here, iteration period [7] is the minimum time needed for computation and this is limited by critical path. The critical path maybe transformed through redistribution of the delays in a manner that conserves the functionality as well. Retiming algorithm is used to redistribute the delays without altering the functionality.

Retiming or index shift method is a traditional loop transformation techniques used to parallelize nested loops [9]. The concept behind the index shift method is to alter the order of precedence of particular statements so that the overall parallelism for an iteration is increased by the hyper plane method Retiming was originally used to reduce the loop duration of a synchronous circuit by uniformly allocating registers.

The retiming technique is an appreciated optimization technique in problems of digital filters which can be represented as data flow graphs and DG. Efficient filter systems are desirable to decline the overall computation time subsequently scientific applications can be recursive, non-recursive, and iterative. Retiming transformation along with other high level transforms like unfolding approach for filters in high level synthesis aids in increasing the throughput of the filter circuit. Unfolding the program opens up previously hidden concurrencies which minimizes the iteration period of the loop thereby maximizing the throughput of implementation.

III. METHODOLOGY

Graphical representations are efficient for exploring and scrutinizing the data flow properties of DSP circuits and for exploiting the inherent parallelism among the diverse subtask. More significantly graphical representation can be used to map the tasks of the DSP algorithm to hardware implementation. This graphical representation enables the step in high level synthesis for transforming algorithmic representation into structural implementation. It exhibits all parallelism and data driven properties of the system and provide an insight into space and time tradeoffs. The DSP applications can be represented in the form of dependency graphs (DG). The nodes in a DG symbolize computations and edges signify precedence constraints among nodes. DG is comprised of computations for every iteration present in an algorithm contains computations for all the iterations in an algorithm. The interdependencies amongst the inherited and data reuse features at the nodes in a dataflow graph can be depicted by a dependence graph.

A. Graph-Theoretic Framework and Attributes

This paper focuses solely on directed graphs. The dependence graph $G = (V, E, d, t)$ [8] is an edge weighted directed graph. The set of computational nodes is represented by V, set of edges is denoted by E, t t is a function from V V to a set of positive integers, representing the computation time of each node and d d is a function from E to a set of non-negative integers, representing the number of delays between any two nodes. Traditionally an edge signifies inter-iteration association when a delay is present and an intra-iteration relation otherwise.

The transformation technique ensures that the original functionality of the circuit is retained and we get a better performance. In this paper we are automating the dependency graph creation process for various DSP benchmark circuits and then incorporating the transformation techniques, retiming and unfolding, to improve the throughput and iteration bound of the circuits.

A high level language enables us to add attributes to the graph. Java is just such a language with specific advantages over low level languages like C or C++. Java cuts down on memory usage by removing to the garbage collector, objects that no longer have any references to them. Due to the above reasons we opted to use JAVA.

B. Transformation Technique for Loop Algorithm Retiming

Retiming is a loop modification method that is used for altering the positions of delay components along a path without disturbing the circuits input output features. The elementary idea is to relocate registers along the paths in the circuit so as to diminish combinational rippling. Retiming works with a behavioral characterization of the algorithm, such as a data flow graph and DG. Retiming [8] is a critical tool used to translate behavioral descriptions of algorithms into physical pathways. It is resorted to while designing software for programmable DSP's, during HLS of application-specific integrated circuits (ASIC's), and while designing the reconfigurable hardware such as field-programmable gate arrays (FPGA's).

A retiming r is a function that reallocates the delays in the original DG i.e. $G = \langle V, E, d, t \rangle$, resulting in a new DG $G_r = \langle V, E, d_r, t \rangle$ such that each iteration still maintains one execution of each node in G. Delay function varies accordingly to maintain dependencies, i.e. $r(v)$ corresponds to the delay units pushed into the edges v $\rightarrow$ w, and subtracted from the edges u $\rightarrow$ v, where $u, v, w \in G$.Thus, we have $d_r(e) = d(e) + r(u) - r(v)$ for each edge u $\rightarrow$ v and $d_r(l) = d(l)$ for every cycle $l \in G$.

Unfolding

Unfolding [9] is an transformation procedure which when applied to the DG generates a multiple DG relating more than one iteration of the original DG under consideration. It increases the computational parallelism in a DG by duplicating actors. These are generally used to process data streams with very high sample rates. Unfolding does not change the transient behavior of a DG, but modifies the throughput. The notion of unfolding is the parallel implementation of numerous instances of a specified DG. Unfolding raises the level of parallelism by introducing copies of an iteration at multiple points. Unfolding factor f is nothing but the quantity of replicas of the original loop in an unfolded loop body. In the unfolded graph G_f, f number of copies of the nodes of untransformed DG is introduced. The iteration period P of a loop is described as the average computation time of iteration. The iteration bound and loop bound of cyclic DG which is defined as:

C. Iteration Bound and Loop Bound

In a DG, series of linked edges are defined as directed path. For a directed path to be termed a loop it has to meet two conditions. The first is the presence of identical start and end nodes. The second is the start node being the exclusive multi executed node in the path. The aggregate of the delay counts of the edges in a cycle gives the delay count. Summing up the computation time of the nodes in the loop results in the total computation time of a cycle. The time prerequisite for carrying out one iteration of the algorithm gives the iteration period of a loop. Reciprocating the iteration period results in the iteration rate of the loop. The duration for loop execution depends upon the precedence relations defined by the edges of DG. The lower bound on the loop computation time, which is also known as the loop bound [7], [9] is computed by:

$$\text{Loop Bound} = \left\{ \frac{t_l}{w_l} \right\} \tag{1}$$

where

t_l is the total computation time of a loop and

w_l is the total number of delays of the loop

The loop with the maximum loop bound is the critical loop of the DG. The loop bound of the critical loop is the lower bound on the iteration period of the DSP program. It is independent of the amount of computing resources available. Iteration bound of the DG depends on total computation time t_l and total number of delays w_l according to the equation below

$$\text{Iteration Bound } T_\infty = \max\left\{ \frac{t_l}{w_l} \right\} \tag{2}$$

D. Throughput

Throughput [10] of a system is the highest rate at which it can receive and process the input data.

The throughput for an IIR filter is computed as

$$f_{sample} \leq \frac{1}{T_M + T_A} \tag{3}$$

Throughput for a FIR filter is computed as

$$f_{sample} \leq \frac{1}{T_M + 2T_A} \tag{4}$$

Where
T_M denotes the computation time for the multiplier
T_A denotes the computation time for the adder

In this paper we have considered the computation time T_M to be 5ns and T_A as 2ns.Generally the multiplication time of the circuit is larger than the addition time.

IV. IMPLEMENTATION

A. Modelling of Dependency Graph

We modelled DG for various DSP benchmark circuits through JAVA programming. The DGs are a convenient form to apply the LTT. Considering a simple second order IIR filter which has eight nodes and eleven edges, The fig. 1 depicts the DG representation of the DSP filter.

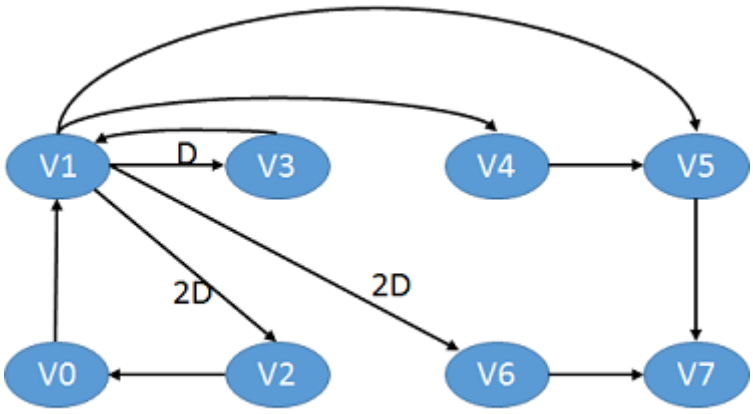

Fig. 1. Second order IIR filter

The nodes are represented as V0, V1, V2…and V7 respectively and the edges are those which interconnect these nodes. The delay information of the circuit are carried by the edges. In Fig.1 there are totally 3 delays. Node $V1 \rightarrow V3$ has a unit delay, $V1 \rightarrow V2$ and $V1 \rightarrow V6$ have 2 unit delays respectively.

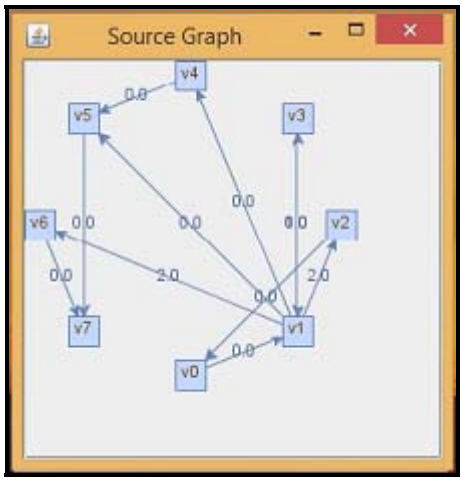

Fig 2. Source graph modelled in JAVA

The fig. 2 represents the DG of a second order IIR filter which was modelled through JAVA.

B. Application of Retiming Technique

The modelled DG is subjected with the retiming transformation algorithm which reduces the latency by reallocating the delays. The below Fig depicts the retimed version of the source graph.

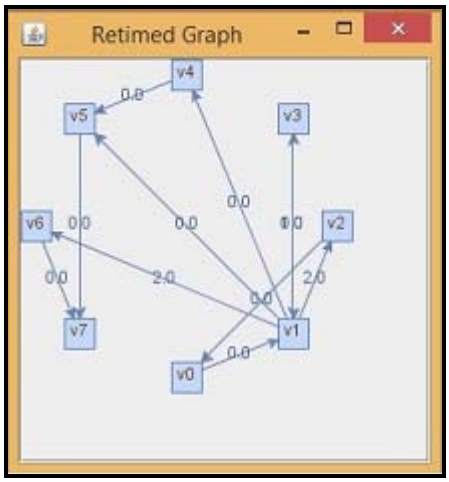

Fig 3. Retimed source graph in JAVA

C. Application of Unfolding Technique

The unfolding transformation algorithm is applied to the modelled DG. We considered the unfolding factor J= 2. There by the number of nodes in the circuit is

doubled and the delays are moved accordingly. This unfolding mechanism increases the throughput of the system as in a single iteration we are processing several nodes together.

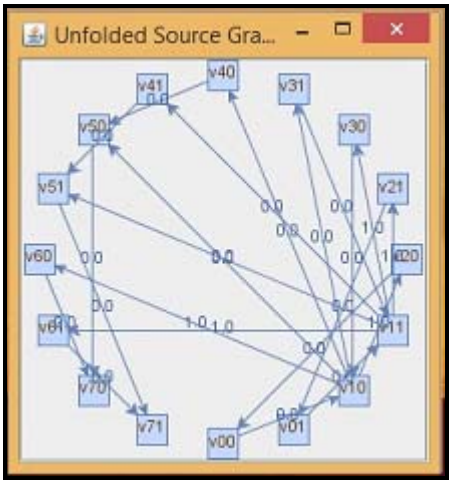

Fig 4. Unfolded source graph in JAVA

The fig. 5 represents the unfolded version of the second order IIR filter. The graphs have 16 nodes, which is twice that of the source graph. The edges which carry the delay information has also been modified according to the unfolding algorithm. There is an increase in the iteration bound of the circuit as in a single iteration several output is obtained which in turn increases the throughput of the circuit.

D. Simultaneous Application of Retiming and Unfolding Technique

The parallelism existent among different iterations is the crux upon which LTTs like retiming and unfolding are based to enhance the performance. Incorporating both the LTTs we get maximum throughput in the circuit. Below Fig represents the graph in which both the transformation i.e. unfolding and retiming has been applied to the second order IIR filter.

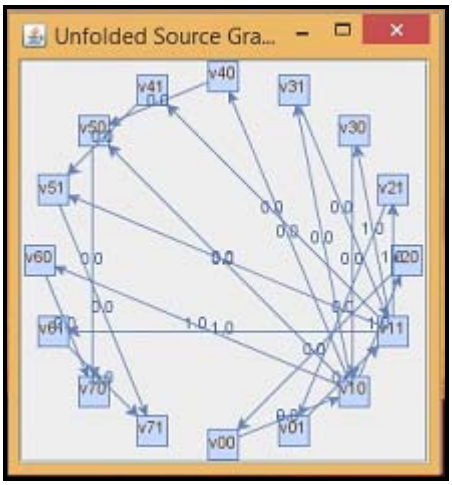

Fig 5. Unfolding and retimed graph in JAVA

V. RESULTS

The fig.6 represents the iteration bound for various DSP bench mark circuits after source graph has undergone unfolding transformation. Unfolding a DG with iteration bound $T\infty$ transform into a J unfolded DG with iteration bound $JT\infty$ as shown in Fig 6.

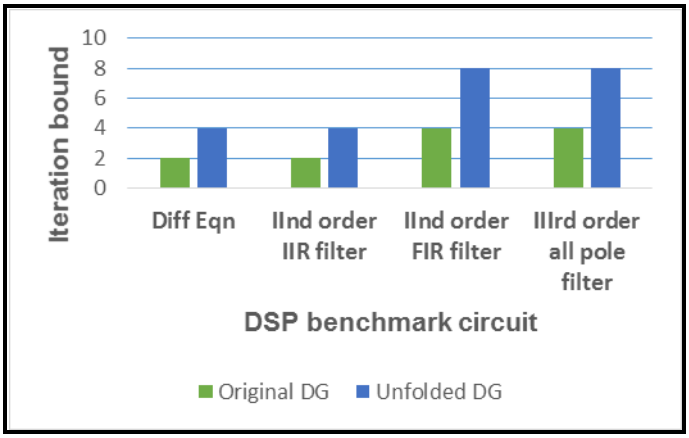

Fig. 6. Iteration bound computation

Throughput computation for various DSP benchmark circuits has been implemented with the formula

$$f_{sample} \leq \frac{1}{T_M + T_A}$$

We have considered the multiplication time as 5ns and addition time as 2ns. With these assumptions the throughput for different benchmark circuits are calculated. The Fig 7 represents the computed throughput for the unfolded DG. It implies there is an improvement in the throughput when unfolding LTT is applied to various DSP benchmark circuits.

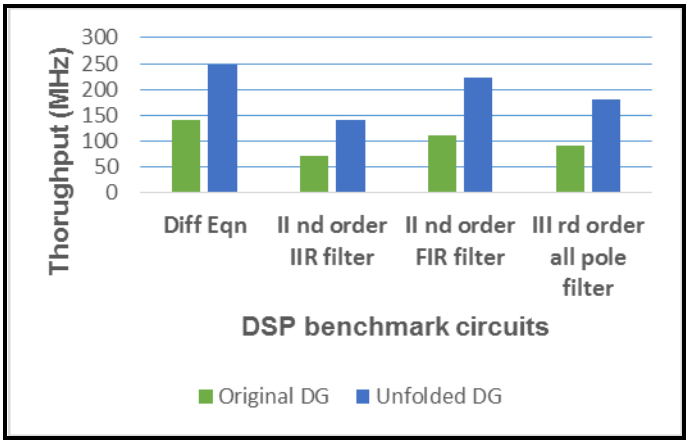

Fig 7. Throughput computation for unfolded DG

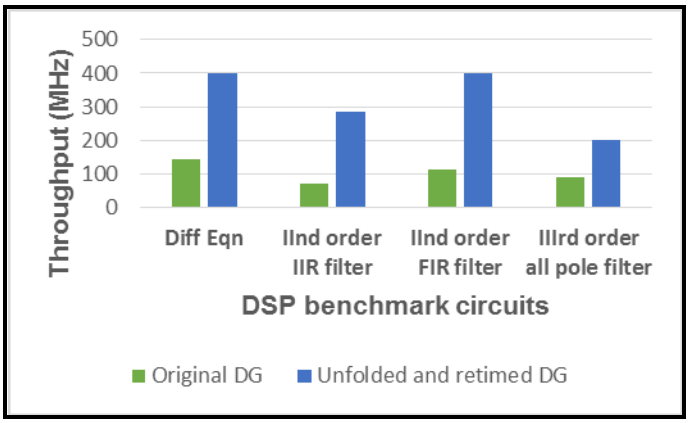

Fig. 8. Throughput computation for unfolded and retimed DG

The graph in fig.8 depicts the comparison of throughput for a source DG and the LTT applied DG. Simultaneous application of retiming and unfolding gives an increased throughput.

VI. CONCLUSION

The unfolding and retiming transformation techniques are universally employed procedures to attain instruction-level parallelism and achieve performance gain in DSP circuits. These algorithm used for different architecture is to reduce the latency

and hence, increase the sampling rate. The experimental results shows that the performance of the DSP circuits have been improved when both the LTT has been applied on the benchmark circuits. These have been implemented using JAVA which can be further utilized to generate RTL. The circuits are proposed to be considered with additional nodes of several types of additional nodes/blocks can be replaced as ports, MUXes and programmable elements and architectures therein after LTTS are to be analyzed to pick up the optimum. In addition depending upon the requirement of the circuit these can also be extended for several set of input libraries and generating possible output architectures for the DSP benchmark circuits.

REFERENCE

[1] K.K. Parhi, VLSI Digital Signal Processing System: Design and Implementation Wiley 2009.

[2] Ville Eerola n, JariNurmi, "High-level parameterizable area estimation modeling for ASIC designs", Integration, the VLSI journal, vol.47, pp.461–475, 2014.

[3] Deepa Yagain and A. Vijaya Krishna, "Design of Synthesizable, Retimed Digital Filters Using FPGA Based Path Solvers with MCM Approach: Comparison and CAD Tool", Hindawi Publishing Corporation VLSI Design Vol 2014.

[4] Sharad Sinha and Thambipillai Srikanthan, "Dataflow Graph Partitioning for Area-Efficient High-Level Synthesis with Systems Perspective", ACM Transactions on Design Automation of Electronic Systems, Vol. 20, No. 1, November 2014.

[5] Seokhyun Lee and Kiyoung Choï "Critical-Path-Aware High-Level Synthesis with Distributed Controller for Fast Timing Closure", ACM Transactions on Design Automation of Electronic Systems, Vol. 19, No. 2, March 2014.

[6] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry, "Algorithmica, vol. 6, no. 1–6, pp. 5–35, 1991.

[7] Ali Shatnawi, "Computing the Loop Bound in Iterative Data Flow Graphs Using Natural Token Flow" World Academy of Science, Engineering and Technology International Journal of Computer, Information, Systems and Control Engineering Vol.1, No.10, 2007.

[8] Liu, Z. Shao, M. Wang, M. Guo, and J. Xue, "Optimally Maximizing Iteration-Level Loop Parallelism", IEEE Trans. Parallel and Distributed Systems, vol. 23, no. 3, March 2012.

[9] Jos´e L. Ayala, David Atienza, Marisa L´opez-Vallejo, J. M. Mend´ıas, R. Hermida, C. A. L´opez-Barrio, "Optimal Loop-Unrolling Mechanisms and Architectural Extensions for an Energy-Efficient Design of Shared Register

Files in MPSoCs" Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'05), 2005.

[10] Alessio Bonfietti, Michele Lombardia, Michela Milano, Luca Benini "Maximum-throughput mapping of SDFGs on multi-core SoC platforms", Journal of Parallel Distributed. Computing. Vol.73, pp 1337–1350, 2013.