Scheduling of RLS Algorithm for HW architectures with Pipelined Arithmetic Units

Submitted By:

Onkar Mahadev Randive

Abstract:

In most modern Superscalars, Execution Units are slowly becoming the bottleneck in the pipelining stages because of the fact that Floating Point arithmetic requires a large amount of time for its execution. The paper on which the project is based upon suggests a new library-HSLA (High–Speed Logarithmic Arithmetic) which will make the floating point operations faster. As all the operations are considered logarithmic (base 2), multiplication, division become easier as they are now just shifting right or left operations.

This projects implements a scheduling technique for the library of arithmetic logarithmic modules for FPGA illustrated on a RLS filter for active noise cancellation. The problem under assumption is to find an optimal periodic schedule satisfying the timing constraints. This implementation deals with automatic parallelisation of algorithms (for example the RLS filter) typically found in control and signal processing applications. The hardware architecture under consideration is based on a library of arithmetic logarithmic modules implemented in FPGA. This library contains a pipelined addition/subtraction unit, which is unique on contrary to many multiplication/division/square root units. Therefore, our scheduling problem is different from all the others. The project is based upon a much larger works and only implements a smaller part of that work. It is **based on the IEEE paper 'Scheduling of Iterative Algorithms on FPGA with Pipelined Arithmetic Unit'** written by Premysl Sucha, Zdenek Pohl and Zdenek Hanzalek.

Implementation:

The basic idea of the project is to implement the RLS algorithm in MATLAB and then find dependencies in the algorithm. The dependencies are then drawn in the form of a directed graph which shows the flow of all the dependencies. Then we try to create tasks in the TORSCHE toolbox corresponding to the nodes in the graph. Further, we create a precedence matrix, which gives the precedence between various nodes of the graph. Then we create a task-set containing all the created tasks and pass the precedence matrix to it so that it knows which tasks preceded which tasks. Next, we create a problem task specifying the constraints and execute the problem by passing the task-set as argument. Finally we have a schedule where we can observe the automatic parallelisation of the algorithm and see that the tasks with no dependencies are executed and do not wait for other dependent tasks.

Introduction:

This project deals with automatic parallelisation of algorithms (for example the RLS filter) typically found in control and signal processing applications. Dynamic properties of these applications are characterized by their time constraints. The main objective is to implement an iterative algorithm to find data dependencies and implement them onto the HSLA library to increase the rate of execution of floating point arithmetic statements. We will first discuss the concepts used in this project,

TORSCHE Toolbox:

TORSCHE stands for time optimization of resources, scheduling. It is a freely (GNU GPL) available toolbox, mainly dedicated for the utilization and development of the scheduling algorithms. TORSCHE has been developed at the Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering. The TORSCHE toolbox provides the basic data structures and methods for the development mainly in the area of scheduling.

Solving procedure of your scheduling problem can be divided into three basic steps:

1. Define a set of tasks.

2. Define the scheduling problem.

3. Run the scheduling problem for the defines set of tasks.

The above steps are followed and can be seen in the code provided with the report.

HSLA : High Speed Logarithmic Arithmetic Unit

- HSLA is an alternative approach to floating-point arithmetic.
- A real number is represented as the fixed point value of logarithm to base 2 of its absolute value.
- As all the operations are considered logarithmic (base 2), multiplication, division become easier as they are now just shifting right or left operations.
- Multiplication, division and square root are implemented as fixed-point addition, subtraction and right shift. Therefore, they are executed very fast on a few gates.
- Addition and subtraction require more hardware elements on the target chip, hence only one pipelined addition/subtraction unit is usually available for a given application.
- Example for processing time using HSLA:

operation of HSLA	proc. time p
$+, -$	9
$*, /, ^2, \sqrt{}$	2

RLS Algorithm:

- The Recursive least squares (RLS) is an adaptive filter which recursively finds the coefficients that minimize a weighted linear least squares cost function relating to the input signals.
- This is in contrast to other algorithms such as the least mean squares (LMS) that aim to reduce the mean square error. In the derivation of the RLS, the input signals are considered deterministic, while for the LMS and similar algorithm they are considered stochastic.
- Compared to most of its competitors, the RLS exhibits extremely fast convergence.
- However, this benefit comes at the cost of high computational complexity.

The RLS algorithm for a p-th order RLS filter can be summarized as

Parameters: $p =$ filter order

$\lambda =$ forgetting factor

$\delta =$ value to initialize $\mathbf{P}(0)$

Initialization: $\mathbf{w}(n) = 0$,

$x(k) = 0, k = -p, \ldots, -1$,

$d(k) = 0, k = -p, \ldots, -1$

$\mathbf{P}(0) = \delta^{-1} I$ where I is the identity matrix of rank $p+1$

Computation: For $n = 1, 2, \ldots$

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-p) \end{bmatrix}$$

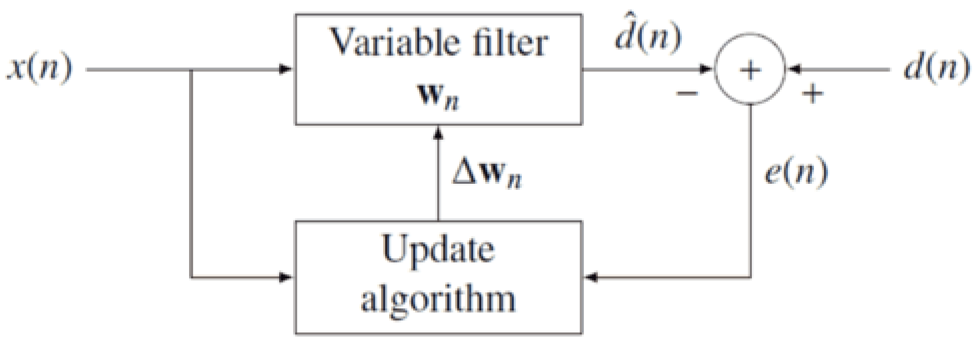
$$\alpha(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)$$

$$\mathbf{g}(n) = \mathbf{P}(n-1)\mathbf{x}(n)\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\}^{-1}$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha(n)\mathbf{g}(n).$$

- The idea behind RLS filters is to minimize a cost function by appropriately selecting the filter coefficients, updating the filter as new data arrives.
- Cost function in this case = completion time.
- The problem under assumption is to find an optimal periodic cyclic schedule satisfying the timing constraints.

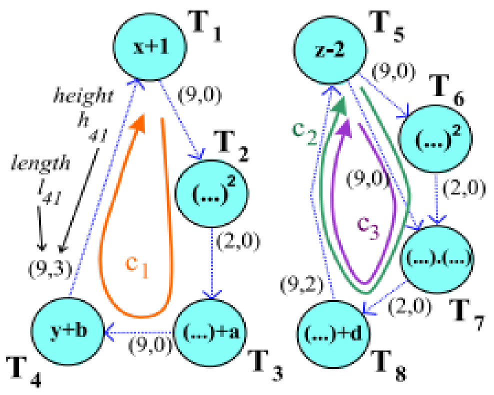


Transition from Code to Graph:

Operations in a computation loop can be considered as a set of n generic tasks T ={T1,T2,...,Tn} to be performed N times where N is usually very large. One execution of T labeled with integer index k ≥ 1 is called an iteration. Let us denote by hi,ki the kth occurrence of the generic task Ti, which corresponds to the execution of statement i in iteration k. The following figure shows an example of a simple computation loop with corresponding processing times.

```
for k=1 to N do
    $y(k) = (x(k-3)+1)^2 + a$
    $x(k) = y(k) + b$
    $z(k) = (z(k-2)-2)^3 + d$
end
```

operation of HSLA	proc. time p
$+, -$	9
$*, /, ^2, \sqrt{}$	2



The above example shows the directed graph showing the dependencies in the small code snippet shown along side it. It can also observed that the processing time of multiplication and division operations is way much higher than that of addition and subtraction. The above graph is just the graph for the small code and we have to find the graph for the whole RLS algorithm. The above graph can be further reduced as:

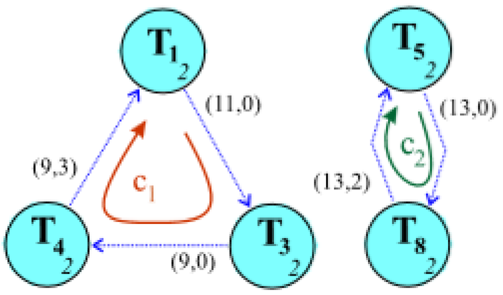


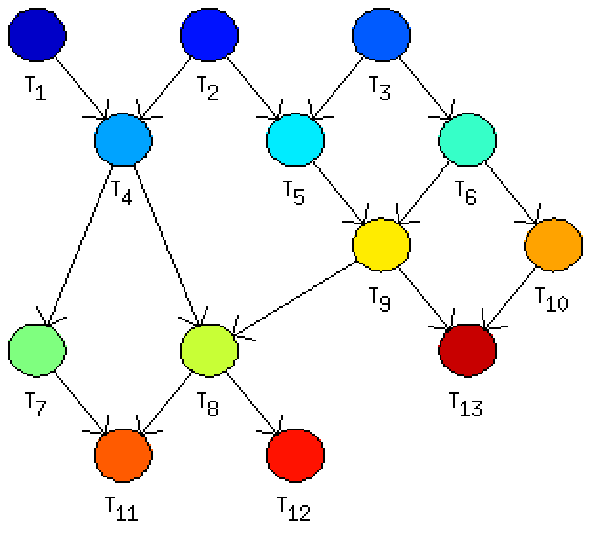
Figure : Reduced graph G'.

Thus for our RLS algorithm code , the directed graph will be as

```
clc
close all
clear all

N=input('length of input sequence N = ');
t=[0:N-1];
ita=10^4;
I=ones(1,N);
R=ita*I;
w0=0.001;  phi=0.1;
d=sin(2*pi*[1:N]*w0+phi);
x=d+randn(1,N)*0.5;
w=zeros(1,N);

for i=1:N
   y(i) = w(i)' * x(i);
   e(i) = d(i) - y(i);
   z(i) = R(i) * x(i);
   q = x(i)' * z(i);
   v = 1/(1+q);
   zz(i) = v * z(i);
   w(i+1) = w(i) + e(i)*zz(i);
   R(i+1) = R(i) - zz(i)*z(i);
end
for i=1:N
yd(i) = sum(w(i)' * x(i));
end
```

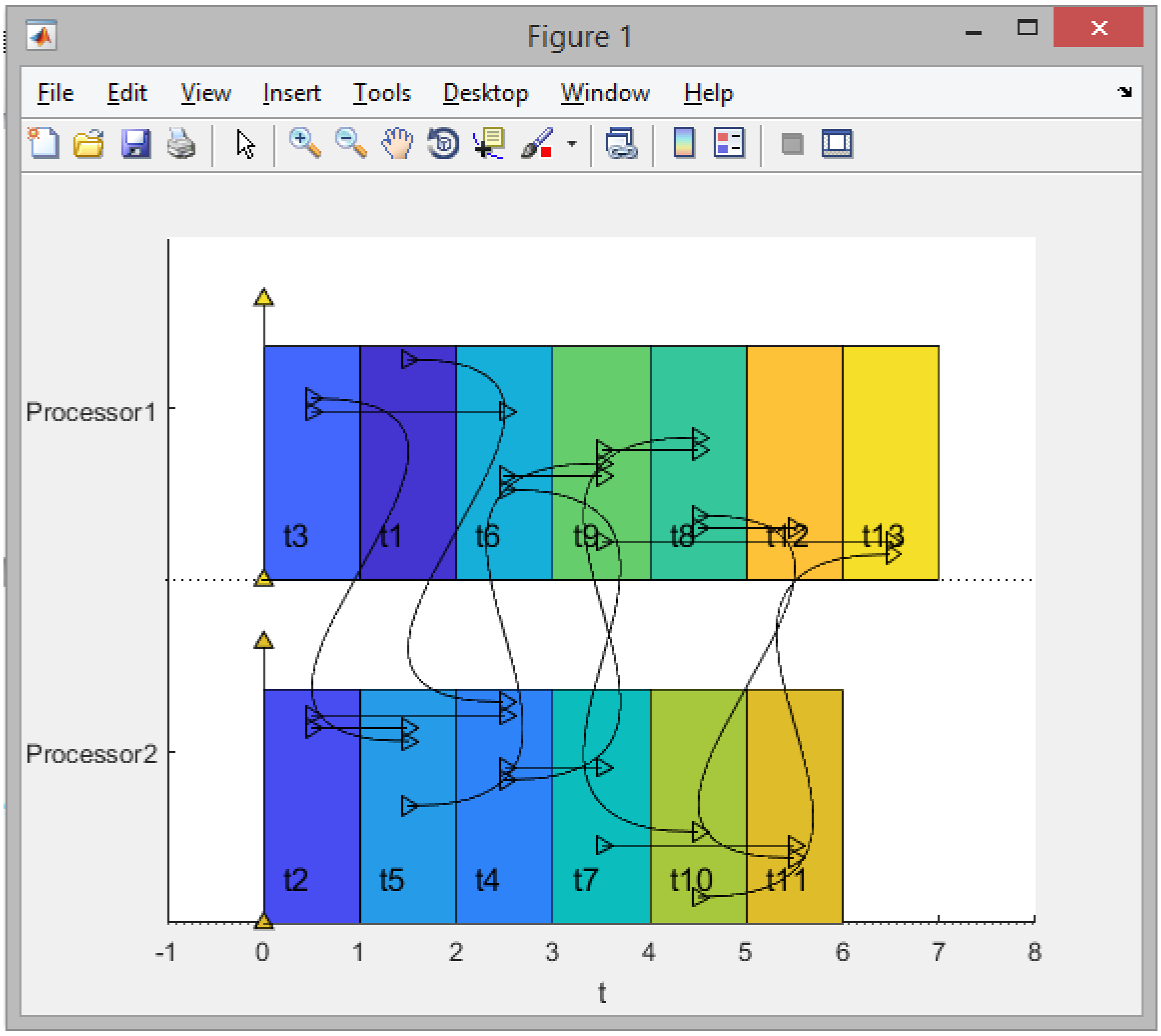


```
0  0  0  1  0  0  0  0  0  0  0  0  0
0  0  0  1  1  0  0  0  0  0  0  0  0
0  0  0  0  1  1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  1  0  0  0  0  0
0  0  0  0  0  0  0  0  1  0  0  0  0
0  0  0  0  0  0  0  0  1  1  0  0  0
0  0  0  0  0  0  0  0  0  0  1  0  0
0  0  0  0  0  0  0  0  0  0  1  1  0
0  0  0  0  0  0  0  1  0  0  0  0  1
0  0  0  0  0  0  0  0  0  0  0  0  1
0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0
```

Note: For simplicity and to avoid obfuscation, we avoid the length of the edges and consider that all the tasks can be scheduled on either of the two processors.

The precedence matrix used in the code to show precedence between nodes which are later converted to tasks is shown above at the right side.

Scheduling Diagram:



Observations:

The scheduling diagram shows the scheduling for the above shown directed graph showing dependencies for the RLS algorithm. The dependencies between processes can be seen through the arrows. It can be also observed that some tasks with no dependencies are executed before tasks that are dependent on other tasks.

Conclusion :

This project presents ILP–based cyclic scheduling method used to optimize computation speed iterative algorithms running on HSLA. The approach is based on a transformation to monoprocessor cyclic scheduling with precedence delays. Further, an optimal periodic solution is searched iteratively.

References:

[1] Premysl Sucha, Zdenek Pohl and Zdenek Hanzalek.-Scheduling of Iterative Algorithms on FPGA with Pipelined Arithmetic Unit - Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings. RTAS 2004. 10th IEEE.

[2] A. Heˇrm´anek, Z. Pohl and J. Kadlec. Fpga implementation of the adaptive lattice filter. In. Proc. FPL2003, Springer, Berlin, 2003.

[3] R. Govindarajan, E. R. Altman, and G. R. Gao. A framework for resource-constained rate-optimal software pipelining. IEEE Transactions on Parallel and distributed systems, vol. 7, no. 11, 1996.

[4] https://en.wikipedia.org/wiki/Recursive_least_squares_filter

[5] TORSCHE toolbox –User manual - http://rtime.felk.cvut.cz/scheduling-toolbox/