UNIVERSITÀ DELLA SVIZZERA ITALIANA

Meeting - 28.03.2014

Simon Maurer

April 10, 2014

1 CHOICE OF DEVICE

A reasonable board: Nexys 4 Artix-7 FPGA Board [1] Having a look at the requirements for the exponential function on a FPGA:

- *double precision: 2024 slices + 3 DSP slices* on a Artix-7 100 T FPGA from Xilinx, there is enough space for 7 exp functions in parallel. One exp function can be fully pipelined [2].
- *single precision: 522 slices + ? DSP slices* on a Artix-7 100 T FPGA from Xilinx, there is enough space for 30 exp functions in parallel. One exp function can be fully pipelined [1].

The algorithm exposes N^2 parallel exponential functions. Assuming $N = 100$, for maximal performance a total amount of 10000 parallel exponential functions are needed (with pipelining this number can be reduced at a performance cost). No available FPGA can handle this amount of exp functions in parallel.

2 EXTENDED FORWARD ALGORITHM

Initialization of the forward algorithm. The variables dk and ok need to be read from memory at each step k (not shown in the code below).

```
% computation of the extended forward algorithm
% @param N: number of states
% @param L: number of observation symbols
```

[1] http://www.xilinx.com/products/boards-and-kits/1-3YZNP5.htm

```
4  % @param PI: initial state probability vector. size N
5  % @param B: matrix of emission probabilities. size N, L
6  % @param C: precalculated factors to compute the extended transition
7  %           probabilities
8  % @return alpha: forward variables. size N, L
9  % @return scale_coeff: coefficients. size L
10 function [alpha scale_coeff] = forward_s(N, L, PI, B, C)
11     k = 1;
12     % initialize forward variables
13     % read o0: index of first observation symbol
14     for i=1:N,
15         alpha(i, 1) = PI(i)*B(i, o0);
16     end
17     % scaling
18     alpha_sum = 0;
19     for i=1:N,
20         alpha_sum += alpha(i, 1);
21     end
22     scale_coeff(1) = 1 / alpha_sum;
23     for i=1:N,
24         alpha(i, 1) *= scale_coeff(1);
25     end
26     % forward algorithm
27     while (k < L),
28         % read ok: index of k-th observation symbol
29         % read dk: delay of k-th observation symbol
30         % compute one step of forward algorithm
31         [alpha(:, k+1) scale_coeff(k+1)] = ...
32             forward_step_s(N, dk, alpha(:, k), B(:, ok), C);
33         k++;
34     end
35 end
```

By focusing on one step of the forward algorithm, it is important to note, that the extended forward algorithm differs only in the computation of the transition probabilities (function call in line 12) in comparison to the normal forward algorithm.

```
1  % computation of one step of the extended forward algorithm
2  % @param N: number of states
3  % @param dk: delay of k-th observation symbol
4  % @param alpha: forward variables of step k-1. size N
5  % @param B: emission probabilities of step k. size N
6  % @param C: precalculated factors to compute the extended transition
7  %           probabilities
8  % @param alpha_new: forward variables of step k. size N
9  % @return scale_coeff: coefficient of step k
10 function [alpha_new scale_coeff] = forward_step_s.m(N, dk, alpha, B, C)
11     % compute extended forward factors
12     v = compute_v(N, dk, C);
13     % compute forward algorithm
14     for j=1:N,
15         alpha_new(j) = 0;
16         for i=1:N,
17             alpha_new(j) += alpha(i) * v(i, j);
```

```
        end
        alpha_new(j) *= B(j);
    end
    % scaling
    alpha_sum = 0;
    for i=1:N,
        alpha_sum += alpha_new(i);
    end
    scale_coeff = 1 / alpha_sum;
    for i=1:N,
        alpha_new(i) *= scale_coeff;
    end
end
```

Following the most expensive part of the algorithm, due to the exponential function: the computation of the extended transition probabilities. The extended transition probabilities depend on the chosen kernels. Here only one kernel has been chosen (Gaussian cumulative distribution function).

```
% computation of the extended transition probabilities
% @param N:     number of states
% @param dk:    delay of k-th observation symbol
% @param C:     precalculated elements
% @return v:    extended transition probabilities
function [v] = compute_v(N, dk, C)
    % compute all elements of v
    for i=1:N,
        for j=1:N,
            v(i, j) = C.c1(i, j) - C.c2(i, j)*exp(C.c3(i, j)*dk);
        end
    end
    % correct diagonal elemnts of v
    for i=1:N,
        for j=1:N,
            v_sum(i) += v(i, j);
        end
    end
    for i=1:N,
        v_sum(i) -= v(i, i);
        v(i, i) = 1 - v_sum(i);
    end
end
```

During the meeting, the following decision has been taken: First focus only on the implementation of the forward algorithm (by omitting the computation of the extended transition probabilities) in FPGA.

3 Computation Intense Functions

A reference for computational intense functions has been named: (James M.) Krause.

4 PRECISION

Precision is a very important parameter in order to create an efficient accelerator. For the moment it is very difficult to define a necessary precision to get reasonable results. After the implementation of the forward algorithm on a FPGA it may be easier to discuss this problematic.

REFERENCES

[1] J. DETREY AND F. DE DINECHIN, *A parameterized floating-point exponential function for FPGAs*, in IEEE International Conference on Field-Programmable Technology, ICFPT, Dec 2005, pp. 27–34.

[2] R. POTTATHUPARAMBIL AND R. SASS, *Implementation of a cordic-based double-precision exponential core on an FPGA*, Proceedings of RSSI, (2008).