ABSTRACT
Hardware Implementation of ECG clustering algorithm based on Hermite Basis functions is presented. Performance of this algorithm has previously been analysed on multi-core CPUs and GPUs. Here using AHIR compiler for VHDL generation and RIFFA framework for communication {References}, we explore the same on an FPGA. For each QRS complex, the basis with least mean square error is computed, Hermite coefficients of which could further be used for grouping the beats. Primary focus is on latency of calculation for single beat which is a critical aspect for Real Time applications. The FPGA implementation exhibits >11x improvement over the former ones.

INTRODUCTION

Manual Analysis of Medical Data is tedious and hence, there has been lot of research on Algorithms for it. Software has been the primary platform for writing complex Algorithms for Biomedical Applications like ECGs and Scan Analyzers. Heavy parallelism offered by GPUs has further enhanced the practice in recent years. Inspite of its potential to offer better Performance than corresponding S/W, a dedicated Hardware approach for such tasks has not come in mainstream, primarily because of the complexity of Hardware Development. Even though High Level Synthesis tools try to overcome this hurdle, it is yet difficult to synthesize a big design with high performance and manageable H/W cost.

In this paper, we explore the problem of ECG clustering using Hermite functions {Reference}. For HDL generation, we use AHIR, an HLS compiler developed at IIT Bombay {Reference}. Equipped with a library of heavily pipelined operators and with Loop Pipelining enabled, it can offer HDL implementations competitive to modern day processors. This feature allows the hardware to have multiple iterations of a loop running simulatneosly adjusted to dependencies in between them. For uncorrelated computations, this enables each operator to work at maximum efficiency as the next loop iteration doesn't have to wait for its predecessor to finish. H/W can further be customised for Power, Area and Speed trade offs.

We'll discuss in brief the algorithm involved and some aspects of implementation. Computation Time results have been compared with the same implementation on CPU and GPU (details from Gabriel's paper){Reference}.

ALGORITHM

Datapoints from a high-pass filtered and windowed ECG signal are used to derive coefficients of Hermite Basis Function. QRS complexes thus obtained, are written as linear combination of Hermite Basis Functions{Reference}. Coefficients of linear equation are derived by taking Dot Product of all the basis functions with Data vector. We have used a set of 6 orthonormal Hermite functions for each value of width [sigma].

Since, the width of QRS complexes varies with rate of heartbeat, singular [sigma] will not give a good fit to different complexes. To avoid error resulting from this, multiple [sigma]-values are chosen and the best amongst those is selected. Apart from a linear increase in dot products, we also have to obtain the [sigma] that gives best fit to the data. Mean Square Errors for all fit coefficients are calculated to predict the best fit which increases the number of computation by a factor of 2.

RESULTS

Both Dot Product and Mean Square Error computation have similar nested loop structure. There are 2 for loops - one iterating on all sigmas and other on all data samples received. There are 144 samples from a QRS complex and we have chosen 10 [sigma]-values uniformly in range (x,y) with 6 orthornormal hermite basis functions each. All samples undergo MAC operation twice, thus requiring 144*10*6*2 = 17280 multiplications and same number of additions. Apart from this, error calculation also requires one subtraction and squaring per sample per sigma increasing the count to 18720 MAC operations. Including data transfer, some peripheral assignments and operations, we estimate ~20K flops for processing 1 QRS complex, all sequential.

Operating frequency of the logic is 100MHz on a Virtex-6 FPGA. Assuming that all operators work to their maximum efficiency (1 output each cycle), a minimum of 0.2ms processing time will be required per beat. However, on first run, the time observed was 1.81ms as shown in {table1.1}. The structure of program was this way - outer loop iterates over all sigmas and inner loop over the samples. Since in dot product and error calculation, the inner loop is reading from and writing to same memory location each time it iterates. this chokes the pipeline and execution stalls unless value calculated in previous iteration is read. The maximum number of iterations running simultaneously can be 2 in this case.

To improve upon current performance, we tried Loop Unrolling. Time consumption reduced in linear proportion to number of unrolls and device Hardware Utilization went up. This is because of multiple locations in same loop accessing shared FP multipliers and adders. MUXing from and DEMUXing to multiple registers shoots up LUT utilization for logic {table 1.1}. However, with more an more unrolling, improvement becomes sublinear and will saturate near lower bound.

Another optimization tried was to reverse the order of loop traversal i.e. to change the hierarchy of loops. This removed the Read dependence on previous iteration in inner loop but introduced another constraint. Loop Pipelining in AHIR applies only to inner loop, number of iterations of which got reduced from 144 to 10, while the outer loop iterations went up from 10 to 144. Computation time was slightly better than previous case but still not close to the minima. Hence, we unrolled the outer loop in this case and achieved significant improvement in performance {table 1.2}. There is no change in amount of resources used. So, Device Utilization Trends are similar to previous case {table 1.2}.

In all, we achieved a minimum latency of 0.43ms for single beat processing which is 11x better than incurred on GPU {Reference}. Large number of cores may enable GPUs to process a whole block of 1000 beats simultaneously, reducing the per beat analysis time but, latency is significantly greater than corresponding H/W implementation, which remains the bottleneck in Real Time Applicatoins.

CONCLUSIONS & FUTURE WORK

Time measurements are currently taken on software side, from the time when data is sent till first output is received. Having a counter embedded in H/W can give deeper insights about critical portions of code.

Parallelism has yet not been exploited. In principle, the compute MSE block can work in parallel with Inner Products module. We can further have multiple threads operating over different [sigma]-values. With the current Device Utilization trends, it doesn't seem feasible on Virtex-6 but could be done on a bigger FPGA.

[Power measurements]

This work opens up a new perspective of looking at Automation in Biomedical Engineering. Shifting from S/W to H/W for other Real Time applications like {check these examples}CAT/Ultrasound scans can significantly enhance the performance. With a dedicated ASIC for the purpose, computation can be faster and more efficient.

ECG code	Slice LUT utilization	Slice Register Utilization	Computation Time	FPGA core Power Consumption
No unrolling	51478(34%)	61028(20%)	1.847 ms	2.723W
Twice Unrolled	58280(38%)	73777(24%)	1.033ms	2.793W
Four Unrolled	73842(48%)	100992(33%)	0.561ms	3.034W

ECG code	Slice LUT utilization	Slice Register Utilization	Computation Time	Power Consumption
No unrolling	56839(37%)	65995(21%)	1.395 ms	2.754W
Two Unrolls	65895(43%)	80709(26%)	0.801ms	2.876W
Four Unrolls	84331(55%)	110165(36%)	0.446ms	3.093