An FPGA Cluster for Real-Time Parallel Human Genome Assembly

***Abstract*— This paper presents a pixel based comparison engine for the purpose of speeding up the DNA seqeuncing process. Using the innate parallelism of FPGA hardware with a highly parallel algorithm this paper shows that an important part of the sequencing processing can be laid off to a hardware acceleration cluster. A demonstration of this was provided on a small cluster of Altera DE0 devices, with comparison speeds reaching up to 23 Million base comparisons per second in simulation, while the practical application reached 1.6 Million base comparisons per second for read lengths of 50 base pairs.**

I. Introduction

DNA Sequencing has been an area of a large amount of research in recent years with a number of Next Generation Technologies significantly increasing the speed at which sequencing can be carried out. However, these techniques have a common bottle-neck due to the massively parallel approach of the detection technologies. In order to allow for long seqeunces to be detected a number of tool-chains have been designed with complex digital processing back-ends.

The most computationally intensive part of this task is often found to be the all-against-all of sub-seqeunces comparison needed for some algorithms. This task has an $O(n^2)$ complexity, making it scale poorly.

Currently the primary algorithms used for the recombination process are CPU and GPU based, using large amounts of compute power often over clusters of machines to solve this issue. In this paper a new application of an existing comparison engine will be presented, using the parrallel nature of FPGA devices to speed-up the all-against-all comparison.

Section 2 will introduce the background of DNA sequencing in terms of the common methods used and the requirements these configurations put upon the comparison engines. Section 3 will then introduce a comparison engine algorithm that can be implemented in a massively parallel manner suitable for FPGA designs and highlights the key features which make this a pracitcal solution for high speed processing. This will then be expanded upon in section 4 introducing a new system design for implementing this algorithm on an FPGA cluster, modifying the core algorithm to meet the practical constraints of the FPGA hardware. Section 5 will then discuss a practical implementation of this system on a set oF Altera FPGAs used as a proof-of-concept. Section 6 will explore the performance characteristics of this design with particular focus put on the scalability of the design as well as core computational speed.

II. Background

In order to understand why a complex digital back-end process is necessary, the common approaches to DNA detection must be understood. The state of the art in DNA detection hardware lies in Next Generation Sequencers which use shotgun sequencing methods to increase the speed of sequencing. An example of this is pyrosequencing. This method uses sequencing by synthesis, the DNA sequence is first split along its base pair bonds to create a long string of single bases, rather than base pairs. The sequence is then introduced to a primer string base by base, this process causes the primer to react with the currently available base releasing hydrogen ions and pyrophosphate. These by-products can be detected and using the known primer sequence it can be deduced which base was present at a certain point in the DNA sequence. The main advantage of pyrosequencing is that process can be miniaturised, allowing up to 1 million detectors on a single chip. However, as this process relies on the chemical reaction of the bases with a primer chain each base can take up to 4 seconds to detect [**?**]. In order to sequence an extremely long sequence of DNA another method must hterefore be used. This is done through exploiting the miniaturisation of the process, by duplicating hte sample DNA many times and hten cutting it into extremely small sub-reads the detection process can be shortened to only the length of the sub-read. This then requires a digital processing back-end to recombine the short-reads into the original seqeunce.

The advantage of this process is that it can miniaturised to fit up to 1 million detectors on a single chip. However, the parallel reads are complete but of an unknown order. The short reads must be reconstructed into a single long DNA seqeunce using the overlaps found in the short reads due to the duplication that happened earlier. Several algorithms can be used to re-order this data and in this paper specifically De-Bruijn and Overlap-Layout-Concensus methods will be considered. Both of these algoirthms first require an all-against-all comparison of the input reads before that data can be used to order them. This is a highly complex process with scaling of O(n^2). It is this area of the digital processing system that a significant speed advantage can be found. This is particularly bad when it is considered that for correct re-ordering up to 40 duplicates of the original seqeunce must have been made, making the

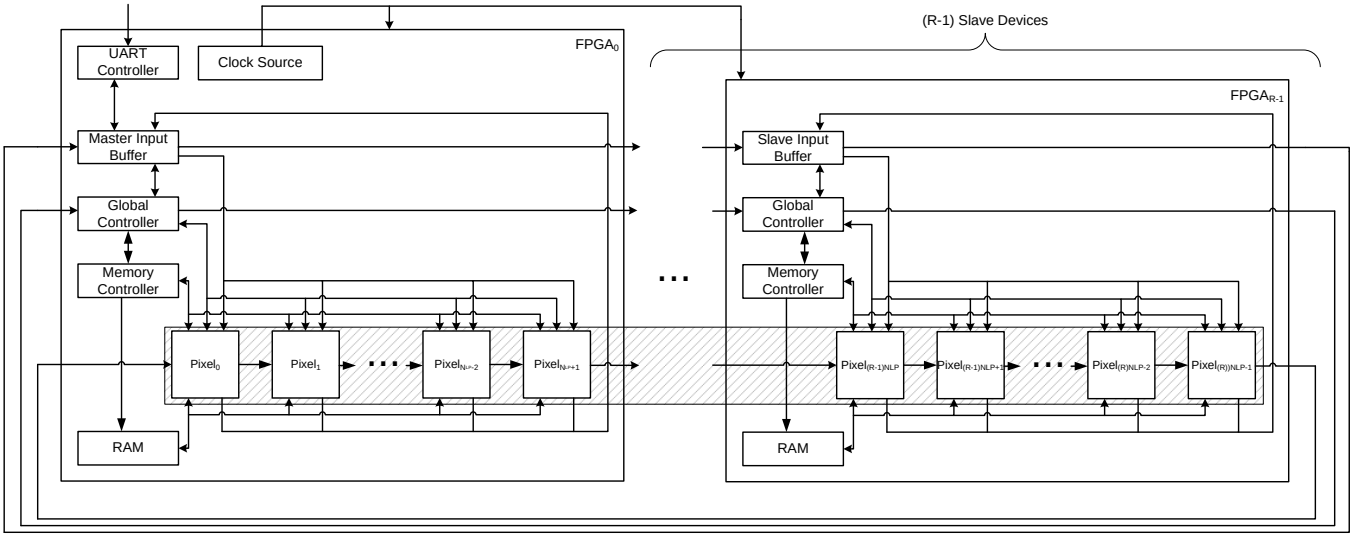

Fig. 1. Top Level Diagram, 1 Host, R-1 Slave Devices

comparison section up to 1600 times longer. A method for doing this was discussed by Y Hu et al. in 2011 [**?**], where a parallel approach was applied to the comparison engine stage to increase the speed of processing.

III. The Comparison Engine Algorithm

A comparison engine algorithm was adopted that used a highly parallel pixel based algorithm. This design was previous adopted in a paper in 2011 by Y Hu et al. The primary design of this algorithm is a ring network of pixels. Each pixel caches a local short-read in memory and are arranged in a ring network. The ring network is used for the pixels to forward their local data around all the other pixels, in this way each pixel receives the local read from every other pixel. Once the data has been fully circulated around the ring each pixel has formed a full comparison library which holds data about how all the strings overlap with its local string. The internal strucutre of the pixel is structured to work on sub-string comparison, this means the comparison engine begins to execute and forward data around the network whilst only a sub-part of the local reads have been received. This allows the whole engine to mask the comparison time with the base detection time when put into a detection tool-chain. This is one by starting to perform the early sub-string comparisons whilst the detection is still taking place. IN cases for read lengths of 50 and the detection times of 4 seconds as described earlier this essentially provides 3 minutes and 20 seconds of free processing time. It is important to note that while this is important for the speed of the algorithm in an important setting this could not be exploited when the algorithm is used for pre-detected data as used later on in the test set-up.

IV. System Overview

The implementation of the comparison engine was optimised for simplicity, scalability, and compatability with FPGA hardware. For this reason this implementation uses a UART controller to stream input and output data to and from a data source. For test bench purposes a simple RS-232 cable was used with a python GUI on a host computer triggering computation, however more practical implementations could stream data directly from a detection chip. This UART controller then used a simple interface with the Input/Output buffer on the master device to stream input data, this module is therefore replaceable for other implementations on faster IO platforms such as PCI-Express.

This input controller then directly forwards all input data to all devices, allowing each one to both select their own data and synchronise the feed of data into the processors with all other devices. In this way lock-step processing was guaranteed with fixed latency and zero communication between the input buffers on each device. In the case of this implementation the primary use of the IO buffer was to perform the conversion from the local parallel data communication widths to the byte-wise communication of the UART.

The core of the algorithm remained unchanged with the ring network of processing pixels being unmodified except for a simple interface to consecutively read out result data. The pixels are shared symmetrically across all FPGA devices, with each device image taking close to the same number of logic elements. The core of the algorithm is not deterministic in speed though, with a shared memory on each device causing an arbitrary delay in the global controller's progression. As a result control logic is implemented in another ring network, allowing all devices to proceed only when every device has reported completion of the memory accesses.

A key design decision in this implementation of the comparison engine was lock-step operation, using a shared clock across multiple devices with comparison computation happening synchronously across all pixels which is reflected in the top level design. Another feature of this design was to

have true scalability across multiple FPGAs, this was done through the use of a single master, multiple slave design whereby the master would do all host interfacing and control the organisation of data. As seen in figure 1 each device has a single input/output buffer to handle interfacing, kept separate from the computation. This buffer feeds data across all the devices and then ensures it is made available at the same time to all pixels. The computation of the pixels is then handled by the global controller which uses a 2-bit bus to synchronise the cache-compare-forward work cycle.

Result data is read out through the same interface as the input data, with results read out from each pixel where they were locally stored. The comparison library consists of a modifiable number of entries based on the density of read data, with the following identifying data. It is important to note here, that the library is the most significant factor in logic utilisation, a denser coverage of a string of DNA will significantly increase the logic size within each pixel.

The result data from the comparison engine is stored in a distributed way within each pixel. This data consists of the quality of a match, the position of the match and the string with which the match was found. Within the original algorithm this was stored as a number of integers within a record. At the end of the full comparison cycle this is then converted to 8-bit chunks and read over the UART. Interpretation of the data after the comparison can be done purely using the fixed width formatting determined by the comparison engine parameters.

A. Tool-Chain Interface

The comparison engine fits into the DNA sequencing toolchain by using the UART interface with a set protocol. Data is read in to the device with a simple binary encoding of the bases Adenine - 00, Thymine - 01, Cytosine - 10, Guanine - 11. This is ordered by first detection order, and then by read identity with 4 base reads packed into a single transmission on the byte-based UART. At the end of the comparison data is read back out through the UART, reading the fixed library in order of pixel identity.

V. Hardware Platform

The hardware platform used for this algorithm was a set of 3 Altera DE0 boards. These boards were found suitable due to both their low cost, and large number of General Purpose input output ports. The hardware setup can be seen in figure [**?**]. The 2x40-pin GPIO connectors allowed for the ring network to be implemented, and an on-board level shifter was used for full RS-232 standard communication. These devices have a local 50-MHz oscillator which was used with a PLL on the master device to send a common clock across all 3 devices.

An important limitation on the hardware was found to be the number of inter-communication pins necessary. Each pixel is capable of doing 4 comparisons in parallel and as such needs suffix data of 8 bits wide, this combined with the 10-bit input data bus and control signals meant that each device needed 44-pins. This would be a limitation for some boards but significantly simplified logic as no protocols were necessary for the transmission of data.

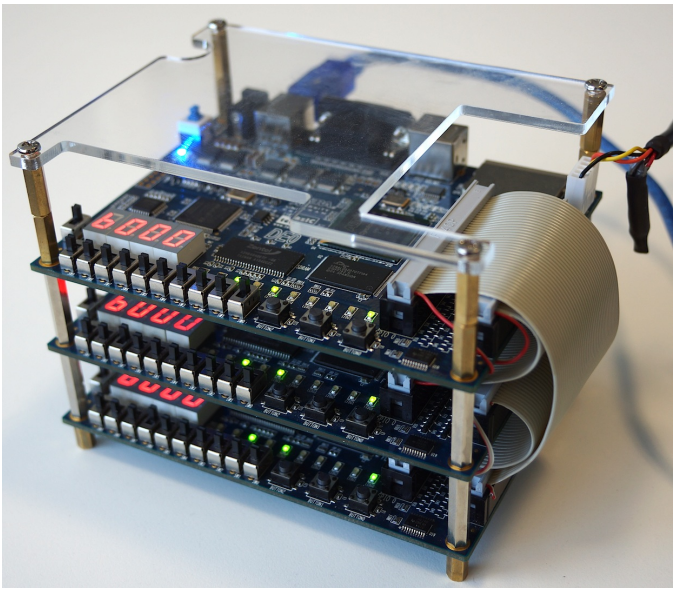

Fig. 2. Top Level Diagram, 1 Master, 2 Slave Devices

Pixel Count	Cluster Size	Logic Elements (Master) (Slave)	Memory Bits (Master) (Slave)
9	3	(5,808) (5,175)	(5,620) (4,532)
15	3	(7,946) (8,230)	(6,612) (4,732)
27	3	(14,097) (14,021)	(8,908) (5,140)
45	3	(21,701) (26,448)	(12,124) (5,740)
99	3	(44,825) (48,578)	(22,732) (7,548)
40	1	(58,609) (N/A)	(13,896) (N/A)
40	2	(28,284) (26,957)	(11,904) (6,240)
40	4	(15,084) (14,116)	(10,888) (5,240)
40	5	(12,623) (10,931)	(10,696) (5,040)
40	8	(8,799) (7,735)	(10,420) (4,740)
40	10	(7,442) (5,958)	(10,336) (4,640)

TABLE I
Resource Utilisation

VI. Results

The implementation of this system has been tested both in the resource overhead associated with the input/output data transfers and the scalability of the design across many devices.

A. Resource Utilisation

In terms of resource utilisation both memory utilisation and logic utilisation was measured when synthesising for different numbers of detection pixels and cluster size. It was important in these measurements that scalability was taken into account. Figure I shows that the resource utilisation of this design scales linearly with pixel count with a small constant resource utilisation. This constant offset of around 5,000 logic elements per device is the full logic element cost of all the data transfer logic. The memory utilisation also scales linearly with pixel count and number of devices, however it can be seen the

memory utilisation is of the same order of magnitude as the logic elements used. For common FPGA devices produced today memory resources are significantly more abundant than logic elements and as such the memory never becomes a limiting factor even when caching all read data on chip.

These results were produced by running synthesis on code in Quartus 13.0 build 156 for Altera Cyclone III devices. These results show that the host interface logic is responsible for a constant 4,000 logic elements, above this the logic count scales linearly with pixel count on both the master and slave devices. The scaling in terms of cluster size shows that the interface logic is present on each device, so a small incentive in favour of a small number of powerful devices is found. This effect is compounded by the clock skew that is found by using larger numbers of devices as discussed below. It must be noted here that library size scales badly, this is because the library must only store the best results found by the comparison engine, this results in all the library entries being compared each cycle which is highly complex. It is important to note however, that library size should be set by the duplication of DNA performed during detection and is therefore unlikely to scale to larger sizes.

The memory utilisation also scales linearly with problem size, however the absolute values of memory utilisation are orders of magnitude smaller than the memory available on a normal device so do not become a limiting factor.

B. Design Scalability

As covered in the resource utilisation section the logic utilisation scales linearly with pixel count, and spreads logic evenly across all devices in the cluster. An important limiting factor is the library size, the library size is directly effected by the coverage of the DNA seqeunce. Due to design choices made in the design of each pixel the library maps primarily to logic cells instead of memory bits. As a result memory utilisation is relatively low, often utilising less than 3% of device memory resources and comparison speed is higher.

There are two further limitations to the design scalability of this architecture; the first being the practical clock synchronisation limits, and the second being the processing delay introduced to the algorithm by inter-FPGA communications.

Due to the synchronisation of data between devices each extra device requires a 2 clock cycle delay to be added to each comparison cycle. This means that whilst more pixels can be added, the existing pixels run marginally slower, this produces an effect which diminishes the returns on added extra hardware as shown in figure [**?**].

C. Application to Large Problem Sizes

The design of the pixel-based algorithm is such that an all-against-all comparison can only be executed in a symmetrical way, where every pixel is compared only against strings that have their own local pixels. This can be considered in matrix form, where hte algorithm is only capable of peforming a square matrix of comparisons with each entry in the matrix representing the row's string compared to the columns's string. A limitation of this algorithm therefore, is that larger comparison matrices cannot be decomposed into non-overlapping square submatrices that can be computed with this algorithm, only the leading diagonal of the larger matrix can be computed without duplicated effort. For this reaosn the algorithm is limited to comparisons with only the number of input strings equal to the number of computational pixels. A future development of this algorithm could however, break the ring network loop and feed in extra data, allowing for an arbitrary sized rectangular matrix of comparisons to be performed. This would allow non-dunplication of work with only minor modifications to the core algorithm.

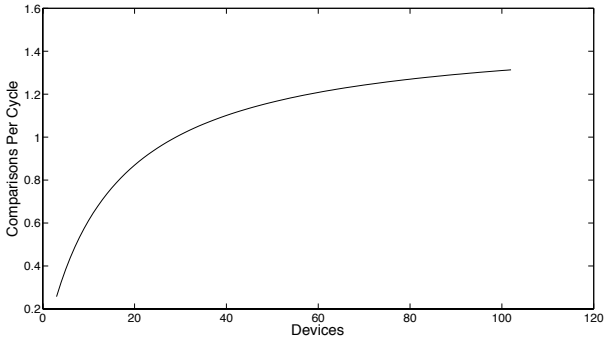

Fig. 3. Scaling of Algorithm Speed with Cluster Size

VII. Conclusion

In conclusion, a small FPGA cluster has shown as a proof of concept that considerable processing work can be passed off in the DNA sequencing tool-chain. The logic utilisation scaled linearly with problem size and split computation evenly across multiple devices. At the small scale that this work was completed on, no speed-up was achieved over software based approaches, however the code has been shown to scale well and the key speed-ups would be seen as problem size increases. This solution also shows significant promise in direct integration into a detection tool-chain using the sequencing process to mask the processing time of the comparison engine. Due to the way the design scaled it was shown that small clusters of high power devices would be the ideal way of producing the most processing power.