A Pico Computing
Life Sciences White Paper

# Accelerating Bioinformatics Searching and Dot Plotting Using a Scalable FPGA Cluster

**Hardware accelerated platform enables fast analysis of DNA sequences**

**Greg Edvenson and Mark Hur**
**Pico Computing, Inc.**

**150 Nickerson, Suite 311**
**Seattle, WA 98109**
**(206) 283-2178**
**www.picocomputing.com**

November, 2009

# Accelerating Bioinformatics Searching and Dot Plotting Using a Scalable FPGA Cluster

## Introduction

In this white paper we present an FPGA-based accelerated solution for DNA sequencing and dot plotting. We describe how multiple FPGA devices can be deployed to create a scalable cluster dedicated to the task of analyzing large amounts of data, and how this clustered hardware application can be connected to a software application for visualization and analysis.

We discuss parallel FPGA optimizations, and we show how higher-level programming methods, using the C language, can speed the development of this and other types of highly parallel algorithms.

## DNA Sequencing and Dot Plots

DNA sequencing and analysis are key components of modern medical science. DNA sequencing is indispensable in basic research as well as in practical applications such as pharmaceutical development, disease prevention and criminal forensics.

DNA sequencing is just one step in the process of bioinformatics analysis. Managing and understanding the results of sequencing and comparing genetic data is critical to making bioinformatics a practical technology.

A dot plot is a graphical tool for visualization enabling the comparison of two biological sequences. A dot plot provides an easy way to understand a large amount of information about the relationship of two sequences, and serves as a framework for further analysis.

The most basic dot plot is a comparison of every acid in each DNA sequence to every acid in the other. These point-to-point comparisons are viewed as a 2-dimensional grid of dots, as shown in Figure 1. Each sequence is placed on an axis of the grid and a dot is drawn at each point in the grid at which the corresponding acids in each chain are equal. When you look at this image of dots—the dot plot—its lines, blocks, and other patterns clearly reveal the similarities between the two DNA sequences.

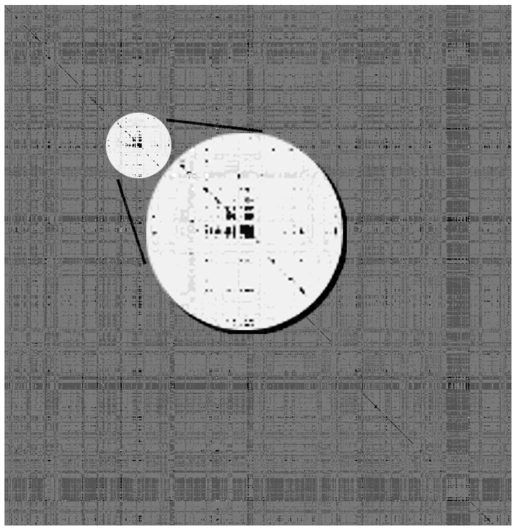

*Figure 1 – A DNA sequence dot plot*

## Software for Creating Dot Plots

Generating dot plots from paired base sequences is not a complicated computing problem, but it is a computationally expensive one. Dot plot generation software programs must apply iterative algorithms over millions of DNA sequences to generate a useable graph.

Because there are just four acids in a DNA sequence (Adenine, Thymine, Guanine, and Cytosine), there is a 25% chance that any specific pair of acids will match. Therefore, a simple pair-matching dot plot will be full of noise from all the randomly occurring matches. To hide the noise, the algorithm must use longer sequences of bases rather than just comparing single base pairs. Using runs of 25 bases is a common approach in today's algorithms.

When comparing runs, a scoring method must be used to determine how closely the two sequences match. One option would be to count only those runs in which all 25 bases match. That method would hide a lot of noise, as the probability of a random match is now $0.25^{25}$. However, this would also hide runs of 24 matching bases. So instead, scoring algorithms must assign a ranking to the runs such that longer matching runs have the most weight, but shorter matching runs aren't completely ignored. These scoring algorithms require substantial computing resources as they are iteratively applied to very long sequences having many millions of pairs.

## Accelerating DNA Scoring in FPGAs

The comparing and scoring of DNA sequences requires algorithms that are "embarrassingly parallel", meaning they are ideal candidates for FPGA acceleration. An FPGA accelerator for this purpose may be implemented as a coprocessor, for example a single large FPGA device closely coupled to a traditional processor, or it may be implemented as a dedicated appliance that includes dozens or even hundreds of FPGAs arranged in a cluster.

In our experiments, we have chosen to implement the DNA sequencing and dot plot application using both methods, using a single FPGA accelerator card attached to a laptop computer for initial development and performance measurement, and then scaling the algorithm up in size, using as many as 160 FPGAs using a Pico Computing EX-Series cluster system.

To make performance comparison easy, we used an open source tool called *Gepard*[i] as the basis for our FPGA-accelerated application. The *Gepard* program is written in Java and computes and displays dot plots as a 2D color graph. For our FPGA implementation we used the same scoring function as used in the *Gepard* program, and we modified the Gepard program to display the resulting software and hardware dot plot results side-by-side. We used a C-to-FPGA compiler available from Impulse Accelerated Technologies to create the FPGA implementation from a higher-level software description.

## Implementation Approach

Our first step was to re-implement the *Gepard* sequencing algorithm in C-language. We did this so we could verify the output of the FPGA implementation directly by comparing it to the output of the same code compiled as software. This would provide us with benchmark results for the software running on a CPU versus the same algorithm running in the FPGAs.

After writing and testing the software implementation, we prototyped the FPGA version of the algorithm by using the Impulse C-to-FPGA compiler to get a rough idea of how efficiently an FPGA would handle the algorithm and its inner code loops. We used a single Pico Computing E-17 card during the development, testing and optimization of the initial sequencing and scoring algorithm.

## A Scalable Approach to Development

The dot plotting application provides an excellent example of using a scalable FPGA solution for development and deployment. During initial algorithm development and testing, a single FPGA device, a Xilinx Virtex-5 FX70T encapsulated in a Pico Computing E-17 card, was used for prototyping and to generate initial performance results. Because the E-17 attaches directly to the development system with an ExpressCard interface (see Figure 2), we were able to make changes, download and test the algorithm without the need for an external FPGA development board or for FPGA downloading and programming cables. In fact, because the E-17 fits into a standard slot on the side of a laptop computer and draws under 2W of power, it was possible to download and test accelerated FPGA algorithms while running on laptop battery power alone.

*Figure 2 – Pico Computing E-17*

## Software Dot Plot Implementation

Before writing and optimizing C code for compiling to the FPGA, we wrote a software version of the algorithm in C for use as a benchmark. We took the *Gepard* program mentioned above as our model, and ported the part of it that calculates the dot plot. This calculation is performed using two functions.

The first function, ***window_score***, calculates the value of a single dot. A ***dot_plot*** function invokes ***window_score*** to generate the full dot plot for two sequences.

The ***window_score*** function uses a non-linear scoring in an easily unrolled loop and is hence highly parallelizable, making it ideal for running on FPGAs.

Our ***dot_plot*** function includes a parameter called ***zoom***, which has the effect of "zooming out" the dot plot when displayed. This allows us to fit large dot plots into a viewable area on the screen. This has the same effect as generating a non-zoomed dot plot and then zooming out with a photo editor. Zooming is important because even simple bacteria have DNA that is millions of bases long, and comparing two of these would produce trillions of individual dots. Zooming out lets us fit these dot plots on a screen that can handle just a few million pixels. We'll call these conglomerated zoomed-out dots "tiles." For example, if the zoom parameter is 10, then each resulting tile will be the combination of 10 X 10=100 dots.

## Creating the FPGA Firmware

The FPGA firmware for this example passed through many revisions and optimizations before reaching its final form. These optimizations were important to achieve a high level of performance, and should be considered for any software algorithm being moved into FPGA hardware.

Because FPGAs have relatively small memories, we optimized the FPGA algorithm and its associated hardware/software firmware to work on small pieces of the complete dot plot. We used software running on the host PC to split the job into smaller pieces for the FPGA to process, and then used software to reassemble the results coming back from the FPGA. Fortunately a dot plot is easy to split; we can take a small section from each DNA sequence and compute a rectangular section of the final dot plot. To complete the dot plot, we then proceed to plot every section of one sequence against every section of the other. For example, if we split each sequence into eight pieces, we'll need to compute 256 subplots, or "tiles" that we can then combine into the complete dot plot.

While making this optimization, we needed to decide what size pieces we would split the problem into. We could have used the zoom factor described above, but that would have limited the size of the pieces and limited our ability to zoom. To provide the needed flexibility, we decided to use multiples of the zoom factor. We coded an initial version of the application that would handle one tile, and compiled the code to create an FPGA bitmap.

## Optimizing Data Transfers: Aggregation

When we ran the first version of the algorithm on an FPGA, we found that moving the dot values from the FPGA to the CPU represented a significant performance bottleneck; the FPGA could calculate values much faster than the PCIe interface could transfer them.

Because the CPU displays a zoomed-out version of the dot plot to fit on the display, we realized we could eliminate the transfer bottleneck by doing the zooming on the FPGA. Rather than move individual dot values to the CPU and require the CPU to zoom out, we decided to compute the zoomed out values on the FPGA, in parallel with performing the scoring.

This FPGA computation was simplified by our choice of sub-tile sizes that were the same size as the zoom factor; all dots in a sub-tile could be merged into a single pixel of the zoomed out plot. This was trivial to implement on the FPGA, requiring only that we sum the dot values over an entire tile and then transmit the sum, rather than all the individual values. This saved both data transfer bandwidth and CPU time. For a zoom factor of 100, this optimization resulted in just one value being sent for each sub-tile, instead of 10,000 values, thereby eliminating the data transfer bottleneck.

## Optimizing for Core Level Parallelism

The dot plot algorithm itself is quite small, and begged to be parallelized. In fact, when we first built an FPGA bitmap we noticed that the entire algorithm required less than 5% of the resources in the Virtex-5 FX70 FPGA device.

Obviously we could increase performance by putting multiple copies of the algorithm on a single FPGA. When doing this, we can think of each instance of the algorithm as a distinct dedicated "core", echoing the idea of multiple CPU cores. Each of these “cores” implements a single hardware “process”, in this case to process individual tiles of DNA sequence data.

For this example we chose to put 16 of these cores/processes on the FPGA, with some additional communications logic. We created a process to read DNA sequence data for 16 tiles from the CPU and send one tile of data to each of the 16 processes. We also created a process to merge the output data into a single I/O stream, so the software doesn't have to read results from each of the 16 different copies separately.

Implementing this type of parallelism required very minor changes to the software, and gave us a nearly 16X performance improvement.

## FPGA-Level Parallelism

Once we had a working single-FPGA implementation with 16 cores, we could scale the algorithm larger by using multiple FPGAs. Pico Computing offers a number of platforms appropriate for this application, including the EX-300 PCI Express board that includes 16 Xilinx Spartan™ FPGA devices (Figure 3).

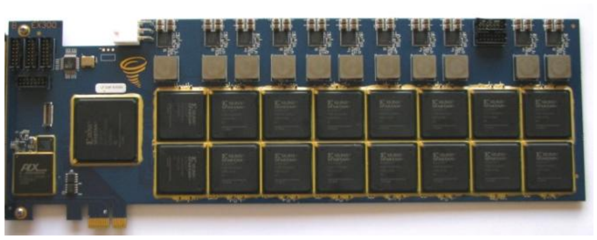
*Figure 2 – Pico Computing EX-300*

To scale the algorithm across multiple FPGAs, we created a software-side thread for each FPGA in the cluster. This allowed us to reuse our single-FPGA software with minimal changes. We created a shared-memory buffer for the final dot plot, and assigned each thread a piece of the input DNA sequences, and a piece of the output buffer. In this way we could run the FPGAs independently and avoid adding overhead for synchronization.

The nature of this algorithm makes it easy to scale across multiple FPGAs. Most high performance computing algorithms require more effort to scale up, so it's important to think of multiple-FPGA implementations when designing the initial algorithm.

Because the software interface for the Pico Computing E- and EX-Series cards and clusters are the same, we were able to recompile and run the algorithm on different Pico Computing platforms with no need to change or even recompile the software. In one cluster experiment, we used ten EX-300 cards, each having 16 FPGAs. Each FPGA had 16 cores, which meant we were running 10 X 16 X 16 = 2560 dot plot cores in one system.

The cluster experiment represented a huge performance improvement from the single FPGA core we had started with. Getting that speedup required packing extra copies of the same core on an each FPGA, and using additional management threads in software.

## Evaluating Performance

The results of running the dot plot application in software and on two clustered FPGA configurations are shown below:

| **Processing Platform** | **Total Time to Complete (4M bases)** |
|---|---|
| Core™2 Duo, 2.66GHz (Windows) | 17,853 minutes (est) |
| EX-300 board (16 Spartan™-3 FPGA) | 24.7 minutes |
| EX-300 cluster (160 Spartan™-3 FPGAs) | 2.5 minutes |

Note that the software run times are estimated, based on measurements of runtimes for smaller numbers of base pairs using the *Gepard* program.

## Summary

Bioinformatics is a category of applications requiring highly parallel, heavily pipelined algorithms. These algorithms are ideally suited to FPGA clusters. As we have shown with this example, using a scalable FPGA platform greatly speeds the development of bioinformatics applications.

## About Pico Computing

Pico Computing specializes in highly integrated high performance computing platforms based on Field Programmable Gate Array (FPGA) technologies. The company also provides consulting and engineering services for industries that include defense, industrial, financial and embedded computing.

## References

[i] Jan Krumsiek, Roland Arnold, and Thomas Rattei, *Gepard: A rapid and sensitive tool for creating dotplots on genome scale.* Oxford University Press, 2007.