Developing a Data Driven System for Computational Neuroscience

Ross Snider and Yongming Zhu

Montana State University, Bozeman MT 59717, USA

Abstract. A data driven system implies the need to integrate data acquisition and signal processing into the same system that will interact with this information. This can be done with general purpose processors (PCs), digital signal processors (DSPs), or more recently with field programmable gate arrays (FPGAs). In a computational neuroscience system that will interact with neural data recorded in real-time, classifying action potentials, commonly referred to as spike sorting, is an important step in this process. A comparison was made between using a PC, DSPs, and FPGAs to train a spike sorting system using Gaussian Mixture Models. The results show that FPGAs can significantly outperformed PCs or DSPs by embedding algorithms directly in hardware.

1 Introduction

A data driven system is being developed for computational neuroscience that will be able to process an arbitrary number of real-time data streams and provide a platform for neural modeling that can interact with these real-time data streams. The platform will be used to aid the discovery process where neural encoding schemes through which sensory information is represented and transmitted within a nervous system will be uncovered. The goal of the system is to enable real-time decoding of neural information streams and allow neuronal models to interact with living simple nervous systems. This will enable the integration of experimental and theoretical neuroscience. Allowing experimental perturbation of neural signals while in transit between peripheral and central processing stages will provide an unprecedented degree of interactive control in the analysis of neural function, and could lead to major insights into the biological basis of neural computation.

Integrating data acquisition, signal processing, and neural modeling requires an examination of the system architecture that is most suitable for this endeavor. This examination was done in the context of spike sorting, which is a necessary step in the acquisition and processing of neural signals. The spike sorting algorithm used for this study was based on Gaussian mixture models that have been found useful in signal processing applications, such as image processing, speech signal processing, and pattern recognition [1,2]. The hardware platforms compared were the desktop PC, a parallel digital signal processing (DSP) implementation, and the use of field programmable gate arrays (FPGAs).

2 Gaussian Mixture Model

Spike sorting is the classification of neural action potential waveforms. The probability that an action potential x belongs to neuron o_i is given by Bayes' rule:

$$p(o_i|x) = \frac{p(x|o_i)p(o_i)}{\sum_{k=1}^{M} p(x|o_k)p(o_k)} \tag{1}$$

where the Gaussian component density $p(x|o_i)$ is given by

$$p(x|o_i) = \frac{1}{(2\pi)^{R/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)} \tag{2}$$

and μ_i is the mean vector waveform and Σ_i is the covariance matrix of the Gaussian model o_i.

2.1 Expectation Maximization Algorithm

The parameters of the Gaussian mixture model were found via the Expectation-Maximization (EM) algorithm. The EM algorithm is a general method of finding the maximum-likelihood estimate of the parameters of an underlying distribution from a given data set. Details of this iterative parameter estimation technique can be found in [3]. A log version of the EM algorithm was used to deal with underflow problems.

3 PC Implementation

For comparison purposes we implemented the training of the EM algorithm in Matlab on various PCs. We used version 6.5 Release 13 of Matlab since it significantly increases the computational speed of Matlab as compared to prior versions. The spike sorting process can be divided into two phases: the model training phase which is computationally expensive and the spike classification phase which is much faster. The test results for estimating the parameters of 5 Gaussian components from 720 160-dimensional waveform vectors are shown in table 1.

Table 1. Training time of EM algorithm using Matlab on PCs. Times are in seconds

	Pentium III 1.2 GHz[1]	AMD 1.37 GHz[1]	AMD Dual 1.2 GHz[2]	Pentium IV 3.3 GHz[3]	Average	Best
Training Time	8.33	3.42	1.45	1.31	3.22	1.31

1 The Pentium III 1.2GHz and AMD 1.37GHz systems both had 512 MB DDRAM

2 The Dual AMD 1.2GHz system had 4 GB DDRAMM

3 The Pentium IV 3.3GHz system has 1GB DDRAM.

The best performance for training was 1.31 seconds on the 3.3 GHz Pentium IV system. Classification performance was 0.05 seconds per waveform vector which meets real time performance since typical neural spike duration is around 2 ms. However, the training process ran much slower than the classification process and needs to be accelerated to minimize delays in an experimental setup.

4 Parallel DSP Implementation

To speed up the training, the first approach we tried was to implement the EM algorithm on a parallel DSP system consisting of 4 floating-point DSPs. Digital signal processors have become more powerful with the increase in speed and size of on-chip memory. Furthermore, some modern DSPs are optimized for multiprocessing. The Analog Devices ADSP-21160M DSP has two types of integrated multiprocessing support, which are the link ports and a cluster bus. We used Bittware's Hammerhead PCI board with 4 ADSP-21160M DSPs to speed up the training algorithm. We used the profile command in Matlab to analyze the EM algorithm and found that 70% of the execution time was spent in two areas. These were calculating $p(x|o_i)$ and updating the means and covariance matrices. We wrote a parallel version of the EM to take advantage of the multiprocessing capability of the board. The results are shown in table 2.

Table 2. Training time of EM algorithm using Parallel DSP board. Times are in seconds

Data Transfer Method	Single DSP	Four DSPs
No Semaphores	N/A	stopped at 7000
Semaphores	14.5	13.5
DMA transfer	15.9	3.4

Not using semaphores in data transfers resulted in bus contention that significantly lowered performance. Using semaphores to transfer data eliminated the bus contention but there was hardly any performance gain when going from one DSP to 4 DSPs. The code was then written to take advantage of the on board DMA controllers and this resulted in a nearly linear speedup. The 4 DSP solution appears to be faster than 4 times the single DSP solution. This is because all the data could be stored in internal memory with 4 DSPs and not with the single DSP solution.

The parallel DSP method ended up being slower than the best PC implementation. The reason for this was that the clock speed of DSP was much less than the high end PC. The clock speed of the ADSP 21160 ran at 80MHz in order to get single cycle multiplies, while the clock speed of the Pentium IV ran at 3.3 GHz, which is heavily pipelined. Thus, the high-end PC was about 41 times faster than the DSP than in terms of clock speed. Even so, the DSP performance was only 11 times slower, highlighting the architectural advantage that the DSPs have in terms of multiply and accumulate operations.

5 FPGA Implementation

Field Programmable Gate Arrays (FPGAs) are reconfigurable devices where algorithms can be embedded directly in hardware. The advantage with using FPGAs is there can be hundreds of embedded multipliers running in parallel tied to custom logic. We targeted Xilinx's Virtex II FPGA that had the following resources (table 3).

Table 3. Virtex II XC2V3000 Resources

System Gates	CLB Slices	Multipliers	Block RAMs
3M	14336	96	96

One drawback of using FPGAs to implementation complex algorithms like the EM algorithm is the long time it can take to design and optimize the algorithms for internal FPGA resources. We used a recently developed tool, AccelFPGA [4], to shorten the design time. AccelFPGA can compile Matlab code directly into VHDL code. By using AccelFPGA, we could focus on optimizing the algorithm in Matlab without dealing with low level hardware details inside the FPGA. The design cycle of the FPGA implementation using this method can be reduced significantly. The FPGA implementation through AccelFPGA is not as optimal as directly coding the algorithms in VHDL by hand. However, for a prototype design, AccelFPGA can provide a time-efficient FPGA solution with reasonably good performance. The performance of the FPGA implementation is shown in table 4.

Table 4. Training time of EM algorithm using FPGA

FPGA Execution Time (sec)	Speedup relative to best PC	Speedup relative to best DSP implementation
0.08	16.4	42.5

Using AccelFPGA v1.6 and Xilinx ISE v5.2.02i

The FPGA implementation was 16.4 times faster than the fastest PC and 42.5 times faster than the best parallel DSP implementation. It should be noted that FPGAs are best for fixed-point processing and that the embedded multipliers are 18x18 bits. Thus the optimal use of FPGAs require the conversion from floating-point to fixed-point. This is ideally done in Matlab using the fixed-point toolbox where simulations can be made examining the effects of reduced precision. Given that the algorithm can be mapped effectively to fixed-point, the next step is to add compiler directives for AccelFPGA. These directives can specify the usage of FPGA's embedded hardware resources such as BlockRAM and embedded multiplier blocks. The third step is the creation of the RTL model in VHDL and the associated testbenches which are automatically created by the AccelFPGA compiler. The fourth step is to synthesize the VHDL models using a logic synthesis tool. The gate-level netlist is then simulated to ensure functional correctness against the system specification. Finally the gate-level netlist is placed and routed by the place-and-route appropriate for the FPGAs used in the implementation. The design used 42 of the 96 embedded multipliers (42%), 58 of the 96 BlockRAMs (60%), and 6378 of the 14336 logic slices (44%).

One advantage of AccelFPGA is that bit-true simulations can be done directly in the Matlab environment. Testbenches are automatically generated along with the simulations. You can use these testbenches later in the hardware simulation and compare the results with the Matlab simulation. As a result, it is very easy to know whether your hardware implementation is correct or not.

Since Matlab is a high level language, the compiled implementation will typically no be as good as direct VHDL coding. However, by using proper directives, you can specify the parallelism of your algorithm and maximize the usage of the on-chip hardware resources. Hence, you can get a reasonably good performance out of your implementation.

The structure of the FPGAs is optimized for high-speed fixed-point addition, subtraction and multiplication. However, a number of other math operations such as division or exponentiation have to be implemented. This was done via lookup tables using BlockRAMs.

6 Conclusion

It appears that the use of FPGAs are ideal for use in data driven systems. Not only are they suitable for digital signal processing applications where incoming data streams need to be processed in real-time, they can also be used to implement sophisticated algorithms such as the EM algorithm.

References

1. Yang, M.H., Ahuja, N.: Gaussian Mixture Model for Human Skin Color and Its Application in Image and Video Databases. Proc. of the SPIE, (1999) 3635
2. Reynolds, D.A.: Speaker identification and verification using Gaussian mixture speaker models. Speech Communication, **17** (1995)
3. Redner, R.A., Walker, H.F.: Mixture Densities, Maximum Likelihood and EM Algorithm. SIAM Review, **26** (1984) 195-239.
4. AccelFPGA User's Manual. V1.7. www.accelchip.com, (2003)