33rd Annual International Conference of the IEEE EMBS
Boston, Massachusetts USA, August 30 - September 3, 2011

Using General-Purpose Graphic Processing Units for BCI Systems

J. Adam Wilson

Abstract— **BioMEMS electrode array fabrication techniques are used to develop high-density arrays with hundreds of channels. However, it was previously impossible to process more than a fraction of these channels real-time for online BCI experiments due to computational resource restraints. It is now possible to use graphics processing units (GPUs), which can have several hundred processing cores each, to processes large amounts of data quickly. This paper summarizes advances in using GPUs for BCI processing for EEG, ECoG, and micro-electrode systems, with speedups of more than 30 times that of current state-of-the-art CPU-based BCI implementations.**

I. INTRODUCTION

Advances in real-time Brain-Computer Interface (BCI) research have been largely co-dependent with increases in computer processor speeds; as the processor (CPU) speed increases, so does the complexity of the extraction and translation algorithms that drive the BCI system. For most of the history of the personal computer, processor clock rates have obeyed Moore's Law, doubling every 18-24 months with increasing transistor density. However, within the last 10 years, processor speeds have plateaued; software developers can no longer count on faster CPUs to develop complex signal processing routines, or to process more data. The current trend is a move towards parallel processing, in which CPUs with multiple processing cores allow numerous calculations to be done simultaneously.

Currently, electrode fabrication methods utilizing biological microelectromechanical systems (BioMEMS) techniques, similar to those used for CPU and integrated circuit design, can produce electrode systems with channel counts that can exceed the processing capabilities currently available in most computers. Implantable systems for recording individual neurons, such as the Michigan array, Utah probe, high-density encephalogram (EEG), and a number of systems for recording cortical field potentials (micro-electrocorticogram, or μECoG [1], [2]) are capable of recording from up to 256 or more channels. This presents a challenge for BCI system designers, because the primary methods for working with such large amounts of data have remained largely unchanged in the last decade: the researcher records a baseline of brain signals, and manually selects a handful of channels with task-related brain activity, often discarding other data in the process. This methodology typically does not account for any complex interactions between multiple channels, because the required algorithms for a real-time BCI would be too computationally intensive for modern CPUs, and many BCI research labs do not have the expertise to develop highly customized hardware architectures using digital signal processing (DSP) chips or field programmable gate arrays (FPGAs)[3]. Furthermore, these hardware platforms do not easily allow rapid prototyping of new algorithms or analysis routines.

However, it has recently become possible to take advantage of the general-purpose processing capabilities of the graphics processing unit (GPU) present on many PCs. GPUs are designed for massively parallel computation, and are equipped with dozens to hundreds of processing cores and several gigabytes (GB) of RAM. Several programming interfaces, including NVIDIA's CUDA[4] and the open-source OpenCL[5], are now available for scientific computing applications that allow data to be processed using the GPU, with performance increases up to 100 times that of a standard CPU. While optimized for matrix operations, these GPU interfaces allow any generic C/C++ code to be written and run on the GPU, with much of the low-level architectural detail abstracted from the programmer. This highly-modular, highly-extensible, and highly-portable approach to massively parallel programming not only allows far greater numbers of neural channels to be processed real-time, it will allow the next generation of real-time algorithms to be developed that takes advantage of the inherently parallel nature of neural processing.

This paper introduces several important GPU concepts using the NVIDIA CUDA framework, provides information about GPU-based algorithms that have been developed for spike processing, and demonstrates the tremendous increases in speed possible. Additionally, previous results from [6] for EEG/ECoG processing will be summarized. It concludes with the implications of GPU computing for the BCI field, and several related applications.

A. Introduction to GPU Computing

Programs written using the CUDA interface involves running code on two platforms concurrently: the *host* system with one or more CPUs, and one or more *devices*, which are CUDA-enabled NVIDIA GPUs. These GPUs are designed to run thousands of lightweight threads, which perform simple computations in parallel. This differs significantly from CPU-based programming, in which one or very few threads perform computationally intensive tasks sequentially, relying on speed to finish the tasks rather than parallelism.

Any program using the CUDA interface will follow the same basic paradigm, in which the host reads and configures the data to be analyzed, initializes the GPU device, copies the data to the device for analysis, before finally running the

This work was supported by the NVIDIA Professor Partnership Grant

J. Wilson is with the Department of Neurosurgery, University of Cincinnati, Cincinnati, OH 45219, USA adam.wilson@uc.edu

device *kernel*, which is the set of instructions that process the data on the device. When the computations are complete, the results are typically transferred back to the host, or left on the GPU for further processing. The data transfer overhead is a very important consideration when writing a CUDA program, because while the kernel itself may provide a significant speed-up, this may be negated by long data transfer times to and from the GPU. This is particularly important for BCIs, which usually require processing times on the scale of <100 ms.

II. METHODS

A. Materials

The tests in this work were performed on three computers: a custom-built desktop workstation, a 2009 Macbook Pro (MBP) laptop, and an ASUS Netbook (Table I). These systems represent a wide range of possible BCI hardware implementations. The operating system (OS) for each computer was Windows 7 Enterprise.

TABLE I
SYSTEMS TESTED

	Workstation	MBP	Netbook
CPU	Intel Core i7 960 4 cores , 3.2 GHz	Intel Core2 Duo 2 cores, 2.8 GHz	Intel Atom 2 cores, 1.6 GHz
RAM	12 GB	8 GB	4 GB
GPU	GeForce FX 470 448 cores 1.2 GHz	GeForce 9600M GT 32 cores 120 MHz	NVIDIA ION 16 cores 450 MHz
GPU RAM	4 GB	512 MB	512 MB

The tests described below were performed on each system, and the processing time and processing time jitter were compared for the CUDA implementation, single-threaded (ST) CPU implementation, and multi-threaded (MT) CPU implementation.

B. BCI Signal Processing: ECoG and ECoG Signals

Detailed methodology in this section has been previously reported in [6], and is briefly discussed here. ECoG signals are typically high-bandwidth (>500 Hz) compared to EEG, and high-channel count (e.g., grids for epilepsy monitoring typically have 64 contacts). Therefore, ECoG-based BCIs, as well as high-density EEG and EEG analyses using complex algorithms, benefit greatly from GPU computing solutions. This section focuses on techniques for virtual movement tasks, such as cursor movement or controlling a robotic prosthesis.

Sensorimotor BCIs take advantage of the user's ability to voluntarily modulate brain activity while performing imagined movements. Preceding and during imagined movements, there is a suppression of 8-12 Hz (μ) and 18-28 Hz (β) activity, while high-frequency activity (e.g., >70 Hz, or high-γ) increases, which the BCI detects and translates into a device command, such as moving a cursor or robotic arm.

A typical processing chain for a movement task is: 1) spatially filter the signals using a common average reference (CAR) or Laplacian filter; 2) calculate the power spectra in specific frequency bins; 3) classify the bins and translate the power amplitudes into movement in 1, 2, or 3 dimensions; 4) normalize the control signals, and adaptively update signal weights based on previous results in training. Of these, the spatial filter and power spectral calculations are the most computationally intensive, and were therefore chosen for GPU implementation. The algorithms were obtained from the BCI2000 system [7]; these are natively written to utilize only a single thread for computation, and so the code was modified to run using both single and multiple threads.

C. BCI Signal Processing: Microelectrode Signals

BCIs that record signals using microelectrodes detect and classify action potentials (spikes), on each channel, and the firing rates of each neuron are used to determine the desired movement direction. This approach is based on the population vector theory from Georgopolos et. al. [8], which states that the firing rate of a neuron in motor cortex is proportional to the intended direction of movement, and follows a cosine tuning curve with a peak at its preferred direction. If sufficient neurons are used, it is possible to estimate the movement direction by using the vector summation of the firing rates and known preferred directions of all neurons, as in [9].

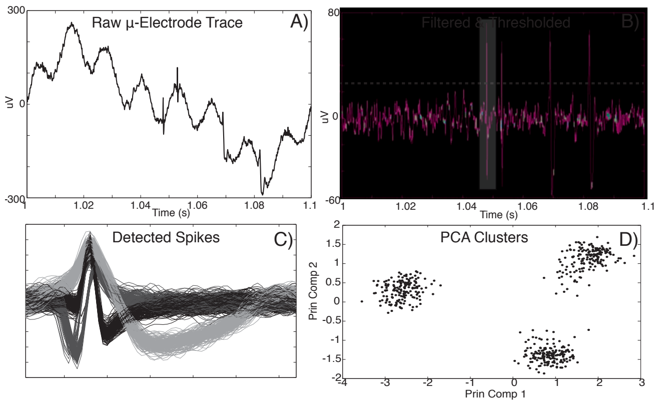

Fig. 1. Spikes are analyzed by A) high-pass filtering raw signals, B) using a threshold-crossing to detect spikes, C) aligning the spikes on the peaks, and D) classifying the spike waveform using principal component analysis and k-means clustering.

Therefore, an example processing chain for spike detection is: 1) high-pass filter the data at > 250 Hz to remove low-frequency field potentials; 2) detect signals above or below some threshold; 3) classify these spikes as neuronal or noise; and 4) calculate the population vector.

1) High-Pass Filter: Signals were HP filtered using an order 30 finite impulse response (FIR) filter with a corner frequency of 250 Hz. The filter and signals were converted to the frequency domain using the CUDA fast Fourier transform (FFT) library (CUFFT), convolved using multiplication, and converted back to the time domain for further processing.

2) Spike Detection: Each CUDA block processed a channel using 256 threads. Peaks above or below threshold were detected by searching for values that were greater than the closest neighboring values and past the threshold.

3) Spike Classification: The first three principal components of the spikes for each channel were previously determined offline. The PCA scores were calculated on the GPU by computing the dot product of each spike waveform with every component, a straightforward GPU procedure using reduction techniques. Spikes with similar waveforms tend to cluster together (see Figure 1, panel 4). A K-means clustering algorithm from [10] was implemented in CUDA to classify new incoming spikes.

D. Spike Tests

Generated data with a sampling rate of 25 kHz was used, and processed in blocks of 50 ms (1250 samples) Channel counts ranging from 1 channel to 1024 channels, incremented in powers of two, were used.

III. RESULTS

A. EEG/ECoG Processing

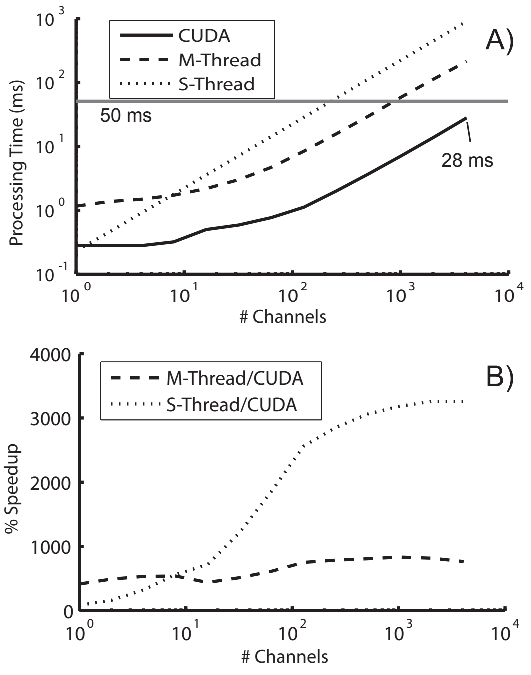

Fig. 2. A) The EEG/ECoG processing times for CUDA, multi-threaded (M-Thread), and single-threaded (S-Thread) execution. The gray bar at 50 ms shows the maximum allowable processing time for an online BCI. B) The speedup % for CUDA vs. multi-threaded and CUDA vs. single-threaded.

Results from [6] are updated here, using a newer video card. In Figure 2A the GPU processed > 4000 channels of 2.4 kHz data in 50 ms blocks (including spatial filtering and auto-regressive power estimation) in less than 28 ms. The ST version took 50 ms to process 128 channels, and the MT version processed 512 channels in 50 ms. Figure 2B shows that the GPU provided a speedup of up to 3000% compared to ST CPU processing, and nearly 800% compared to the MT version running on the 8-core CPU. The relative performance gains increased as the amount of data increased.

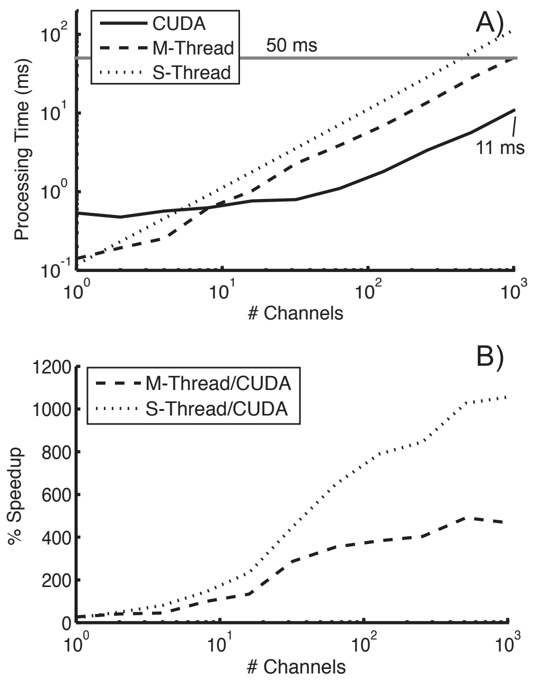

Fig. 3. A) The spike processing times for CUDA, multi-threaded (M-Thread), and single-threaded (S-Thread) execution. The gray bar at 50 ms shows the maximum allowable processing time for an online BCI. B) The speedup % for CUDA vs. multi-threaded and CUDA vs. single-threaded.

B. Spike Processing

Figure 3 show the processing times and % speedup for the CUDA, MT, and ST algorithms, including the time required to apply an order 31 high-pass filter using the FFTW and CUFFT libraries, threshold-based spike detection, spike classification, and population vector calculation. Once more than 8 channels are processed, the GPU consistently outperforms the CPU, and was more than 1000% faster than the ST version for > 512 channels. The GPU is able to process 1024 channels is less than 11 ms, compared to 51 ms for the 8-core CPU.

Another important consideration for a real-time BCI is the timing consistency and jitter, or the standard deviation (σ) of the processing speeds [11]. For both single and MT algorithms, the jitter increased significantly with the number of channels processed. The σ for 8, 128, and 1024 channels respectively was 0.21 ms, 3.32 ms, and 26.70 ms for the ST version, 0.08 ms, 1.31 ms, and 9.50 ms for the MT version, and 0.09 ms, 0.18 ms, and 1.16 ms for the GPU. Therefore, the GPU provides much better timing consistency compared to both CPU versions, a critical factor for designing BCIs.

C. System Comparison

Figure 4 shows the overall spike processing times for the Core-i7 8-core CPU, the GTX 480 GPU, the Macbook Pro laptop with a 9800M GT GPU, and the ION GPU. The GTX and 9800M GPUs outperform the CPU when more than 10 channels are processed, while the ION GPU outperforms the CPU when more than 64 channels are processed.

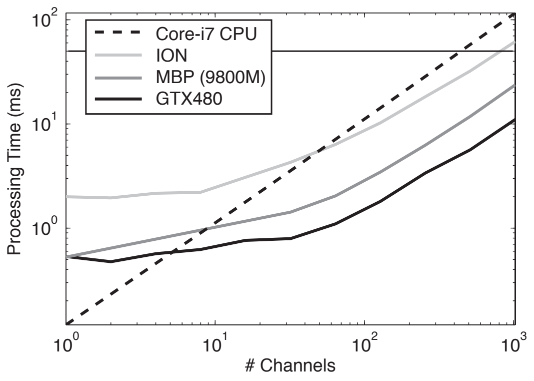

Fig. 4. A comparison of the processing times of the three GPU-based systems and the 8-core CPU system.

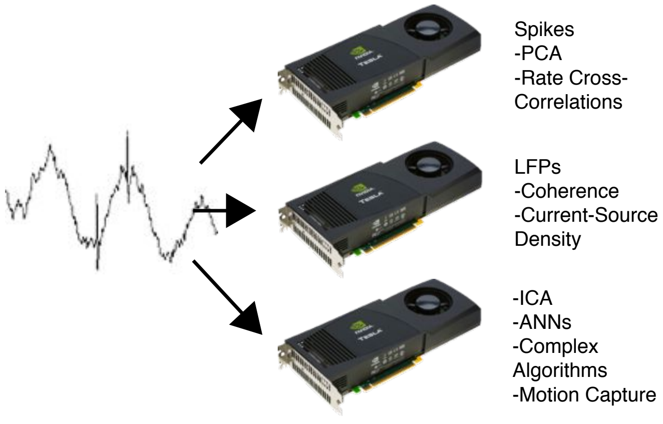

Fig. 5. Neural signals can be split into parallel processing pathways executed on multiple GPUs.

IV. CONCLUSIONS

A. Future Work

GPU computing opens the possibility of a new realm of real-time BCI and neurophysiology experimentation. While it is unlikely that many labs will be using hundreds or thousands of electrodes in the near future due to engineering limitations, GPU computing allows researchers to utilize additional processing steps after the initial calculations, e.g., calculating the cross-channel coherence or current-source density in local field potentials (LFPs) or EEG/ECoG, and additional waveform analysis and spike rate calculations in spikes, to name a few possibilities (Figure 5). Furthermore, GPU computing provides multiple parallel processing pathways in ways that were not computationally feasible previously. This is because it is possible to 'stack' GPUs in a single computer, using up to four for processing and display.

Therefore, a real-time experiment recording 100s or 1000s of wide-band neural channels containing both LFPs and spikes could be split to multiple GPUs, e.g., using one for spike processing, a second for field potential processing, and a third for more complex algorithms that were previously impossible to employ real-time, such as large artificial neural networks (ANNs), independent component analysis (ICA), and many others. These analyses would not need to be performed on a distributed processing platform, e.g. using a network connection, which requires synchronizing several computers and can introduce network delays and jitter. Furthermore, other monitoring modalities could be incorporated as well, such as motion capture and analysis. Finally, GPU technology continues to be further miniaturized, and is being integrated into tablets and cell phones, which might be used for portable BCI computing, e.g., for wheelchair control, in the near future.

B. Conclusions

GPU computing provides an inexpensive, scalable, and programmatically portable method for developing BCI and electrophysiology processing systems that outperform current state-of-the-art CPU systems. Frameworks such as OpenCL and CUDA provide programmers with a relatively simple interface to leverage the parallel computational power of GPUs, provide superior real-time timing characteristics, and can meet the processing requirements for 1000's of channels for the foreseeable future without the need to develop custom FPGA hardware or DSP systems.

V. ACKNOWLEDGMENTS

The author gratefully acknowledge the contribution of the NVIDIA Corp. Professor Partnership Grant.

REFERENCES

[1] J. Kim, J. A. Wilson, and J. C. Williams, "A cortical recording platform utilizing microECoG electrode arrays." *Conf Proc IEEE Eng Med Biol Soc*, vol. 2007, pp. 5353–5357, 2007.

[2] B. Rubehn, C. Bosman, R. Oostenveld, P. Fries, and T. Stieglitz, "A mems-based flexible multichannel ecog-electrode array," *J Neural Eng*, vol. 6, no. 3, p. 036003, Jun 2009.

[3] A. Malatesta, L. Quitadamo, M. Abbafati, L. Bianchi, M. Marciani, and G. Cardarilli, "Moving towards a hardware implementation of the independent component analysis for brain computer interfaces," in *Biomedical Circuits and Systems Conference, 2007. BIOCAS 2007. IEEE*. IEEE, pp. 227–230.

[4] NVIDIA, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide, v 4.0.*, 4th ed., NVIDIA Inc., 2011.

[5] Khronos Group Std. Rev 44. (2011, Jun) The opencl specification, version 1.1. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[6] J. A. Wilson and J. C. Williams, "Massively parallel signal processing using the graphics processing unit for real-time brain-computer interface feature extraction," *Frontiers in Neuroinformatics*, 2009 (Submitted).

[7] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: a general-purpose brain-computer interface (BCI) system," *IEEE transactions on bio-medical engineering*, vol. 51, no. 6, pp. 1034–43, Jun 2004.

[8] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey, "On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex." *J Neurosci*, vol. 2, no. 11, pp. 1527–1537, Nov 1982.

[9] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829–32, Jun 2002.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C: the art of scientific computing*. Cambridge Univ. Press Cambridge, Dec 1999.

[11] J. A. Wilson, J. Mellinger, G. Schalk, and J. Williams, "A procedure for measuring latencies in brain-computer interfaces," *IEEE Trans Biomed Eng*, vol. 57, no. 7, pp. 1785–97, Jul 2010.