Title
Large-scale neural circuit mapping data analysis accelerated with the graphical processing unit (GPU)

Authors
Shi, Yulin
Veidenbaum, Alexander V
Nicolau, Alex
et al.

Publication Date
2015

DOI
10.1016/j.jneumeth.2014.09.022

Peer reviewed

Published in final edited form as:
J Neurosci Methods. 2015 January 15; 0: 1–10. doi:10.1016/j.jneumeth.2014.09.022.

Large scale neural circuit mapping data analysis accelerated with the graphical processing unit (GPU)

Yulin Shi[1], **Alexander V. Veidenbaum**[2], **Alex Nicolau**[2], and **Xiangmin Xu**[1,3,4,*]

[1]Department of Anatomy and Neurobiology, School of Medicine, University of California, Irvine, CA 92697-1275

[2]Department of Computer Science, University of California, Irvine, CA 92697-3435

[3]Department of Biomedical Engineering, University of California, Irvine, CA 92697-2715

[4]Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92697- 2625

Abstract

Background—Modern neuroscience research demands computing power. Neural circuit mapping studies such as those using laser scanning photostimulation (LSPS) produce large amounts of data and require intensive computation for post-hoc processing and analysis.

New Method—Here we report on the design and implementation of a cost-effective desktop computer system for accelerated experimental data processing with recent GPU computing technology. A new version of Matlab software with GPU enabled functions is used to develop programs that run on Nvidia GPUs to harness their parallel computing power.

Results—We evaluated both the central processing unit (CPU) and GPU-enabled computational performance of our system in benchmark testing and practical applications. The experimental results show that the GPU-CPU co-processing of simulated data and actual LSPS experimental data clearly outperformed the multi-core CPU with up to a 22x speedup, depending on computational tasks. Further, we present a comparison of numerical accuracy between GPU and CPU computation to verify the precision of GPU computation. In addition, we show how GPUs can be effectively adapted to improve the performance of commercial image processing software such as Adobe Photoshop.

Comparison with Existing Method(s)—To our best knowledge, this is the first demonstration of GPU application in neural circuit mapping and electrophysiology-based data processing.

*Corresponding author: Dr. Xiangmin Xu, Department of Anatomy and Neurobiology, School of Medicine, University of California, Irvine, CA 92697-1275, Tel: 949 824 0040, Fax: 949 824 8549, xiangmin.xu@uci.edu.

Conclusions—Together, GPU enabled computation enhances our ability to process large-scale data sets derived from neural circuit mapping studies, allowing for increased processing speeds while retaining data precision.

Introduction

Computer-based resources can greatly facilitate neuroscience research. As neuroscientists demand more computing power for circuit mapping studies, researchers may resort to expensive solutions such as clusters and supercomputers for large-scale computational tasks. Due to recent advances in the graphics processing unit (GPU) computing technology, today's GPU does much more than rendering graphics as originally intended, and GPUs can provide an inexpensive and computationally powerful alternative to CPU based solutions (Nageswaran et al., 2009; Baladron et al., 2012).

Traditionally, a central processing unit (CPU) has been the computation core of computers. The CPU is specialized in optimizing serial operations while containing various sub-circuits for multiple types of tasks, such as coordinating concurrent software processes, predicting branches, handling high priority interrupts and managing cache-memory traffic. However, its serial processing nature limits its ability to perform intensive parallel computation. In comparison, the newest generations of GPUs have stream multiprocessors and act as powerful massively parallel coprocessors. Yet, the price paid for such power at a relatively modest cost is that GPUs are unconventional in their organization, and in particular are highly constrained in their communication bandwidth with the main CPU and in the type of operations that can efficiently be executed in parallel. It is often extremely difficult to map a given application for effective (fast) execution on GPUs. Regardless, because of their promise, scientists and engineers have returned to exploring the power of GPUs for various applications ranging from astrophysics and finance to biomedical research, enabled by the recent advances in hardware and integrated programming interfaces such as Nvidia's Compute Unified Device Architecture (CUDA) platform. In a custom-designed system, GPU-accelerated applications can split their computationally intense tasks into a number of threads which could be further processed by thousands of cores of GPUs. In addition, the recent addition of double-precision floating point units provides GPUs the capability to produce accurate results that can satisfy the strictest computational requirements.

In the field of neuroscience, GPU computing has been successfully used for computer-intensive tasks, including large-scale modeling and simulation of neural network, high-speed imaging and real-time reconstructions (Nageswaran et al., 2009; Wilson, 2011; Yang et al., 2011; Baladron et al., 2012; Tomer et al., 2012; Fang and Lee, 2013). However, to our best knowledge, no or few studies have tapped into the GPU computation for neural circuit mapping data analysis while large scale neural circuit mapping experiments in vitro (Shepherd et al., 2005; Xu et al., 2010; Franke et al., 2012; Kuhlman et al., 2013) and in vivo (Ohki et al., 2005; Nauhaus et al., 2008; Nauhaus et al., 2009) produce large amounts of data in short times that require tremendous computation power for post-hoc processing and analysis. The ability to take advantage of GPU-enabled computation in a desktop computer system presents a sufficient and cost-effective solution to individual neuroscience research laboratories that have no access to or cannot afford the access to high-performance

computing centers. To promote a wider application of GPUs in neuroscience research, in this paper, we first introduce how to assemble a GPU-enabled desktop computer system with detailed hardware and software information. We then compare CPU and GPU-enabled computational performance of our system in benchmark testing and practical applications. We specifically present an approach of using GPUs in Matlab to achieve accelerated processing of large amounts of neural circuit mapping data. Finally, we present an analysis of numerical accuracy between GPU and CPU computation to verify the precision of GPU computation.

Materials and Methods

In this section, we describe the configuration and assembly of a GPU-enabled desktop computer system. We also describe our data acquisition and relevant analysis, and detail on the algorithms used to optimize the data organization, exploit the memory hierarchy and integrate GPU computation.

System Components

Our system consists of an Intel i7-3930K CPU and an Nvidia Tesla K20c GPU (Figure 1). The CPU has 6 physical cores which share a 12MB L3 cache. It communicates with 32 GB system memory at a bandwidth of 49.64 GB/sec. Note that the CPU memory bandwidth varies at different memory clocks. The GPU has 2496 physical cores that are clustered in 13 Stream Multiprocessors (SMX), which share a 1.5MB L2 cache and communicate with a 5 GB graphic memory with 208 GB/sec bandwidth. The system memory and GPU are connected througha PCI-E 2.0 x16 bus with a transfer speed of 8 GB/s each way. Please see Table 2A for detailed comparison of specifications.

For such a custom built desktop computer system, we have selected components (see Table 1) to maximize the performance while keeping the cost within a reasonable range. The CPU is from Intel’s high performance product line. We have carefully over-clocked it and the CPU performance increased by 25–29% as tested using GPUBench (see below). The RAM used is also a high-clocked version from a reputable manufacturer to match the over-clocked CPU. The whole system is cooled by 7 fans with the CPU cooled by a closed-loop liquid cooler. Such intense cooling is necessary since overheating can trigger the protection mechanism in the CPU, resulting in reduced performance, less accurate computation and even chip degradation. The motherboard should support two graphic cards running concurrently at PCIE x16 2.0. A full size tower case is required to fit all the components including the heavy-duty cooling and ventilation system. The whole computer system needs to prove stable by running intensive computation tasks for 48 hours and the maximum recorded CPU core temperature needs to be below 75 °C, recommended by the Intel documentation. The final cost of the entire system (as per Table 1) was around $7500, and the Tesla K20c GPU, although donated by NVidia, had a market value of $3500 by the time the system was assembled.

System Assembly

The general computer assembly has been described in many existing guides (e.g., http://www.gskill.us/forum/showthread.php?t=10512). It is a good practice to assemble basic components on the bench and have them individually tested outside of the tower case. After the initial test, we can now fit every component into the case. Although the layout of the components is fixed with the specified tower case and motherboard, it is important to route connection cables between components in a tangle-free manner. It is equally important to determine the location of the CPU liquid cooler radiator and air flow directions. Ambient cool air needs to be drawn by the fan from the case bottom, and should be heated up as it travels through the hard drives, GPUs, the CPU radiator and RAMs. The cooling fans placed at the top and rear of the tower case draws the heated air out of the case. Dust filters at the front of the incoming fan helps to reduce dust accumulation on internal components. Please see Supplemental Figures 1 and 2 for the images of our final assembled system. After the system assembly is completed, we recommend to overclock the system to achieve performance gains. Finally, it is important to test the stability for a custom assembled system. Many tools are available for free on the internet, and the common choices include Intel Burn Test, Linx, prime95 and Memtest.

The necessary software includes the operating system software, driver software, Matlab (Mathworks, Natick, MA) and CUDA Software Development Kit (SDK) for support of GPU computing. We use Windows 8 x64 due to its large user volume and better support for Solid-State hard drives. After Window 8 is installed, it is recommended to run the embedded system assessment test to ensure that the operating system (OS) can fully evaluate the hardware performance. Without such a test, the OS may use a default performance profile, limiting the hardware capability. The drivers for the motherboard, graphic card and Tesla Card are available for download at the manufacturer's websites, and they should be specific to your component choices. We use a system monitoring software, Aida64 (http://www.aida64.com/) to keep track of the system status. It provides monitoring data of the CPU and GPU temperatures, voltages, cooling performance, and other parameters. This software has a safe-operating mechanism which alerts and puts the computer to a standby mode when any overheating is detected.

We use Matlab software (version r2013b) as a GPU computing platform as the software is commonly used in neuroscience research and the new version contains the latest Parallel Computing Toolbox with many GPU enabled functions. In order for Matlab to have access to the Tesla card, CUDA driver from Nvidia website (https://developer.nvidia.com/cuda-downloads) has been downloaded and installed. A simple benchmark using GPUBench (provided by the Parallel Computing Team of Matlab) is performed to ensure every hardware component is running without any unexpected throttling.

GPU computation for accelerated processing of experimental data

Different from most previous neuroscience applications in neuronal network modeling and simulation, this custom system is intended for using GPU-parallel computation in actual experimental data analysis that requires intensive computation. We have developed and applied photostimulation-based mapping techniques for local cortical circuit connectivity

analysis. Particularly, laser scanning photostimulation (LSPS) combined with whole cell recording in living brain slice preparations allows high resolution mapping of regional distributions of presynaptic input sources to single neurons (Figure 2). Because the simultaneous recording from a postsynaptic neuron with photostimulation of clusters of presynaptic neurons at many different locations, the LSPS methodology provides quantitative measures of spatial distribution of excitatory or inhibitory inputs.

The experimental procedure and the design of our laser scanning photostimulation system has been described previously (Shi et al., 2010; Xu et al., 2010; Kuhlman et al., 2013). A laser unit (model 3501, DPSS Lasers, Santa Clara, CA) was used to generate a 355 nm UV laser for glutamate uncaging. Once stable whole cell recordings were achieved with good access resistance (usually <20 MΩ), the microscope objective was switched from 60× to 4× for laser scanning photostimulation. At low magnification (4× objective lens, 0.16 NA; UplanApo, Olympus), the slice images were acquired by a high-resolution digital CCD camera (Retiga 2000, Q-imaging, Austin, TX) and used for guiding and registering photostimulation sites in cortical slices. For each recorded neuron, we usually map photostimulation-evoked inputs from a grid of 1 mm × 1mm with 16 × 16 stimulation sites (Figure 2). Various laser stimulation positions were achieved through galvanometer-driven X-Y scanning mirrors (Cambridge Technology, Cambridge, MA), as the mirrors and the back aperture of the objective were in conjugate planes, thereby translating mirror positions into different scanning locations at the objective lens focal plane. During mapping experiments, photostimulation was applied to the 16×16 patterned sites (centered at the recorded neuron) in a nonraster, nonrandom sequence, while whole-cell voltage-clamp recordings were made from the recorded postsynaptic neurons with EPSCs and IPSCs measured at the holding potential of −70 mV and 0 mV, respectively, across photostimulation sites. Data were acquired with a Multiclamp 700B amplifier (Molecular Devices, Sunnyvale, CA), data acquisition boards (models PCI MIO 16E-4 and 6713, National Instruments, Austin, TX), and custom-modified version of Ephus software (Ephus, available at https://www.ephus.org/). Data were digitized at 10 kHz, and stored on a computer. Electrophysiological signals were recorded for 1000 ms in length for each stimulation. Each map dataset of 256 traces is about 20.48 MB in size. Although the example data size is not large, it served a demonstration purpose.

As for the analysis of photostimulation map data, a new technique that combines the design of a bank of approximate matched filters with the detection and estimation theory was implemented for automated detection and extraction of photostimulation-evoked EPSCs or IPSCs (Shi et al., 2010). Specifically, the Matlab program, “synaptic_event_detection”, uses a bank of existing matched templates to detect synaptic event in the signal traces. During the detection stage with the CPU computation, the data trace under investigation is first high-pass filtered (>10 Hz) with a 5^{th} order, infinite impulse response Butterworth filter. The role of this filter is to minimize the effect of the direct uncaging response, whose duration is much longer than that of synaptically mediated indirect responses. To minimize the phase distortions, this filter is implemented as a zero-phase forward and reverse digital filter. The high-pass filtered signal is then convolved with all the filters from the bank, with potential EPSCs or IPSCs providing a better fit to the templates and thus exhibiting larger convolution amplitudes. The convolution traces (one for each filter) are time-shifted to

minimize the difference between the time of the convolution peak and a potential EPSC/IPS peak, and thus facilitate a more precise estimation of event occurrence times. Time-shifted convolution traces are then compared to detection thresholds. For each convolution trace, the samples that exceed the detection threshold form the so-called suprathreshold time segments. Within each eligible suprathreshold segment, a synaptic event was extracted and the center of mass of each convolution trace is found and declared as an occurrence time candidate of an EPSC/IPSC. For more technical information, please refer to Shi et al. (2010). Of the above processing steps, the convolution of matched filters with high-passed signal traces requires the most intensive computation, which we have targeted for GPU implementation.

GPU-oriented Programming Considerations

The CPU and GPU architectures are illustrated in Figure 1. The stream processors of a GPU can only access the data that are stored in its video memory, cache or shared memory. The video RAM (VRAM) is large enough (5 GB on the K20c GPU) to store the data and results relevant to the computation. To use a GPU for computation, the data needs to be explicitly transferred to VRAM from the system memory (RAM) through the PCIE 2.0 bus. The transfer rate of PCIE 2.0 is 8.0GB/s for each direction, which is much lower compared with the GPU-VRAM or CPU-RAM speed (Fig. 1). There is also an initiation latency of data transfer, with the average measured latency being 0.3 ms. To maximize the GPU performance, it is critical to reduce the number of data transfers between RAM and VRAM.

We modified the “synaptic_event_detection” program in light of the above consideration of effective GPU computation (Figure 3). All the high-passed signal traces from each map data set are concatenated as one large vector and transferred to VRAM altogether. The same operation is done for the matched filters, which are originally stored in a Matlab cell array. An auxiliary vector is created to store the lengths of matched filters, and help index individual matched filters in the filter vector. In GPU-enabled computation, the large vector of signal traces is convolved with each matched filter and the resultant trace is then transferred back to RAM for event detection and extraction. The code for such GPU-tailored modifications is shown in Appendix 2. A complete CPU code of the “synaptic_event_detection” program is described in our previous publication (Shi et al., 2010) and available at our website: http://xulab.anat.uci.edu/synapticeventdetection/index.htm).

Results

CPU versus GPU Performance Benchmarking in Matlab

To compare the potential speedup of using GPUs, we first examined the performance of CPU versus GPU-enabled computation using ‘GPUBench’ in Matlab. Basic scientific computations were performed through different GPUBench tests including MTimes, Backslash and fast Fourier transform (FFT). MTimes involves the multiplication of two matrices, A and B, whereas Backslash involves the matrix “division”, A/B. The Backslash operation essentially finds the solution of a system A*X=B by Gaussian elimination. FFT computes the discrete Fourier transform (DFT) of a vector X. A, B and X were generated

using Matlab's random number function, *rand()*. GPUBench uses the Matlab integrated timer command (*tic*, *toc*) to measure the processing time of GPU and CPU kernels. The performance of both kernels in GFlops is then calculated by dividing the number of floating point operations by the actual processing time. The processing time does not include the allocation/deallocation of RAM or VRAM, since the memory spaces are allocated at the beginning and are not reallocated during benchmarking. Also, the GPU benchmark does not include the time to transfer the data to VRAM and the computational results back to RAM.

The performance of both single and double precision floating point computation was evaluated in different tests (Figure 4A; Table 2B). Compared to CPU performance in terms of GFlops, GPU-enabled computation achieved a speed up of 705%, 315% and 2207%, respectively, in single precision computation of MTimes, backslash and FFT, while their respective speed up was 577%, 388% and 1326% in double precision computation. These dramatically increased speeds are due to powerful massively parallel coprocessors in the GPU.

Convolution Speed Comparison

For LSPS map data analysis, we use the custom-made Matlab program, "synaptic_event_detection", which uses a bank of existing matched templates to detect synaptic events in signal traces. Since convolution using the Matlab function, *conv()*, is a computational core of the program for synaptic event detection, we are interested in examining the performance of CPU and GPU-enabled computation in convolving data vectors of different lengths with an 1-dimensional (1-D) template of a fixed length. As the convolution computation is not included in the GPUBench software, we implemented a simple benchmark code (see Appendix 1) that performs 1-D convolution in Matlab. Vectors of different lengths were generated by using the Matlab *rand()* function. The lengths of data vectors are $2^X * 10000$ ($X \in [1, 2, \ldots 8]$). A second vector was generated by the Matlab *rand()* function to be used as a convolution filter with its length being 20000. The data vectors of different lengths were convoluted with this fixed filter. Again, the time of memory allocation/deallocation is not included in actual computation time; the GPU computation time includes the necessary data transfer between RAM and VRAM. As shown in Figure 4B and Table 2C, GPU constantly outperformed CPU in convolving data with a vector length of 40000 and higher. Relative to CPU processing, the speed up of GPU computation tends to increase with the larger vector length. When the data vector length is $2^8 * 10000$ (the total length of 256 traces contained in one typical LSPS map data set), the speed up is about 700%.

Acceleration of Electrophysiological Data Processing

To examine the performance gains in actual experimental data analysis, we compared the processing speed of the CPU version of the "Synaptic Event Detection" program versus its GPU-enabled version (see Appendix 2 for code comparison). As described in Methods, the GPU version of this function was modified to minimize the number of data transfer over PCIE x16 2.0 bus. All individual traces (256 traces from one map data set) were first concatenated to a large trace. The data of both the signal trace and matched filters are of double precision floating point types. During the automated event detection stage of the

"Synaptic Event Detection" program, the key step that requires intensive computation is to convolve signal traces with a bank of matched filters (Figure 5A). Typically, as illustrated in Figure 3, the CPU completed the convolution of 256 signal traces with each of the 29 matched filters in a total of 2.55 seconds. In comparison, the GPU completed the convolution in 0.51 seconds, which corresponds to a speed increase of 500% for this operation. Note that the data transfer overhead was 0.03 seconds each way and the cost for reformatting data for the GPU computation was 0.15 seconds. Since both event detection cores are on the CPU, the processing time is comparable at 1.90 seconds and 1.94 seconds in the CPU and GPU versions, respectively. The GPU version takes much less time (59% of the CPU version) for processing all 256 electrophysiological traces that are contained in one data map. This data processing acceleration would make a huge difference in computational times in consideration of our batch analysis of hundreds of different data maps.

Although both CPUs and GPUs are compatible with IEEE 754-2008 Floating-Point Standard, it is important to compare the accuracy of GPU computation results to those of CPU as the CPU and GPU use different architectures for double floating point computation. We used CPU results as a standard and examined how much the GPU results deviated from them. The convolution of the same signal trace using 29 different matched filters was acquired from both CPU and GPU versions of the detection program. The difference was calculated by comparing the convolution traces of using the GPU versus the CPU program. As shown in Figure 5B, the maximum difference between CPU and GPU convolved traces is $81.5\pm 3.72 * 10^{-13}$ pA. These tiny differences may result from different floating point rounding modes used by the CPU and GPU (Yablonski, 2011). In practice, these numerical differences have no impact on our data analysis. When we used the synaptic events that were detected and extracted in both programs for quantification and visualization, there is no difference across the data maps and they are identical (Figure 6). Thus, GPU enabled computation enhances our ability of processing large data sets derived from photostimulation circuit mapping experiments. It also allows us to increase data processing speeds while retaining data precision.

Image Processing Application

GPUs have been used for enhancing image visualization through custom-written programs (Eilemann et al., 2012). In this paper, we also show GPUs can be effectively adapted to improve the performance of image processing in Adobe Photoshop. This is major commercially available software that is widely used in academic research for routine image edits and analysis. The Photoshop software has adopted GPU computation since the release of the CS4 version. However, the latest version of Photoshop CC cannot utilize the Tesla K20c card. We used its sibling GTX680 card (a high-end consumer-grade GPU installed in our desktop system) to complete this test. GTX680 shares the same microarchitecture of Tesla K20c but it has a lower configuration with 1536 physical cores running at 1006MHz with 2GB VRAM and costs a fraction of the price of the Tesla K20c. We examined how much the GPU could speed up general Photoshop operations on two testing images of different sizes. The images are in .tif formats with 24 bit depth for the RGB mode; image 1 has 8176 pixels × 6132 pixels with a size of 150MB, and image 2 has 16352 × 12264 with a size of 573MB. To ensure consistent benchmark results, we used a fixed series of Photoshop

actions (http://cdn.pugetsystems.com/articles/PugetBench.zip) to apply blurry effects to the tested images. These actions included Field blur, Iris blur and Tilt-Shift effect. We found that GPU-enabled computation achieved 165% speedup (16 sec versus 26 sec) in completion of the three actions on the smaller image, and 196% speedup on the larger image (52 sec versus 102 sec), respectively.

Discussion

Although GPUs have been used in several neuroscience fields including neural network simulations and high speed imaging (Nageswaran et al., 2009; Baladron et al., 2012), our work addresses the GPU application in neural circuit mapping and electrophysiology-based data processing. To our best knowledge, this is the first demonstration in this subfield of neuroscience. In addition, we extend previous studies by presenting a comparison of numerical accuracy between GPU and CPU computation to verify the precision of GPU computation. This is an important aspect of GPU application, but has not been rigorously tested before.

We utilize GPU and CPU co-processing to speed up large-scale neural circuit mapping data analysis, taking LSPS mapping data analysis as an example. A GPU-enabled desktop computer system was designed and assembled toward this purpose. We described GPU-oriented program considerations, and presented the algorithms for integrating GPU processing with detailed flow-chart diagrams of the implementation for an accelerated version of the synaptic event detection program. Our results indicate that cost-effective GPU computation facilitates large-scale data with increased speeds and with unaltered accuracy. In addition, we showed that even a high-end consumer-grade GPU is capable of speeding up Adobe Photoshop's performance by 50–90%, depending on the operations and the image sizes.

Compared with CPUs, the newest GPUs have an order of magnitude higher computation power and memory bandwidth. However, GPUs are designed as special-purpose co-processors and their programming interfaces are harder to use than those on the CPUs. Although GPU manufacturers and other organizations have developed various development kits that allows users with programming knowledge of high-level languages to take control of many stream processors of a GPU, it still requires steep learning to understand the GPU programming paradigm, data structure and complex optimization rules. As Matlab is a commonly used mathematical programming suite and its new parallel computing toolbox contains many functions that support GPU-enabled computing, we chose to use Matlab as a computing platform. We modified the existing Matlab programs to utilize GPU computation with merely three basic steps: 1) transfer data to the GPU, 2) compute and 3) gather results. As the GPU on-board memory limits the capabilities of processing large volumes of data, preprocessing data with CPUs is necessary. However, the efficiency may be significantly hampered by the relative high-latency of the data exchange between CPUs and GPUs. Therefore we optimize the Matlab GPU program and follow the rule of "Communicate less, compute more". For our synaptic event detection program, the GPU takes over the intensive computation load from the CPU and returns the results in a quick and accurate manner to the system memory (with a 5x speed increase) for further CPU processing. In addition, as

Matlab-derived CUDA code is suboptimal and can be further improved, we plan to re-write the Matlab GPU kernel in CUDA code and then have it run in the Matlab environment. Such an approach could achieve a balance between execution efficiency and code re-writing effort.

Although we focus on data analysis of synaptic event data sets derived from LSPS experiments, the performance gain enabled by GPU computation should be likely extended to data analysis of multiple electrode array (MEA) recordings, fast voltage sensitive dye (VSD) imaging and large-scale single cell resolution calcium imaging experiments (Ohki et al., 2005; Nauhaus et al., 2008; Franke et al., 2012). MEA and bulk loading calcium imaging techniques offer the possibility to simultaneously record from large numbers of neurons (e.g., up to several thousand neurons) with relative ease, but at the expenses of increased efforts to detect and extract single neuronal activities from the recorded ensembles. Their data processing that requires intensive computation for spike sorting and calcium transient event detection can be similarly accelerated with GPUs using the algorithms outlined in this paper. Furthermore, while the speedup and accuracy demonstrated in the present study using the particular LSPS data analysis leads the GPU to be considerable for electrophysiology-based neural circuit mapping, GPU-enabled desktop computer systems will be potentially essential for real-time closed-loop experiments, with the potential for running data analysis in real time while collecting data from technically challenging, large scale recordings. We believe that GPU-enabled desktop computer systems will facilitate real-time closed-loop experiments, with the potential for running data analysis in real time while collecting data from technically challenging recordings. In addition, we record from rare types of neurons and we would like to maximize the number of different protocols carried out for each recording. One limitation is that we only estimate when we have data sufficient for achieving required statistical power for any given protocol. Ideally if we could confirm such statistical thresholds by concurrent real time analysis enabled by the GPU system, we could maximize the amount of data extracted from a given recording. As our high-speed VSD imaging experiments generate large amount of data (Clement et al., 2012; Olivas et al., 2012; Shi et al., 2014) and require extensive data processing such as noise filtering, averaging and linear drift removal, we will also be able to extend the GPU-enabled computation to facilitate image data acquisition and analysis.

In conclusion, we believe that when appropriately applied, affordable GPU-enabled computation is efficient and powerful. Thus we advocate for its wider applications in neuroscience research.

Supplementary Material

Refer to Web version on PubMed Central for supplementary material.

Acknowledgments

This work was funded by a US National Institutes of Health grant NS078434 to X.X. We thank Nvidia for providing a Tesla K20c to support this work. We also would like to thank Dr. Zoran Nenadic and Jeffrey Ingeman for commenting on this manuscript.

Glossary

GPU	Graphical processing Unit
CPU	Central Processing Unit
Ram	Random Access Memory
VRAM	Video Random Access Memory
LSPS	Laser Scanning Photostimulation

References

Baladron J, Fasoli D, Faugeras O. Three Applications of GPU Computing in Neuroscience. Computing in Science & Engineering. 2012; 14:40–47.

Clement JP, Aceti M, Creson TK, Ozkan ED, Shi Y, Reish NJ, Almonte AG, Miller BH, Wiltgen BJ, Miller CA, Xu X, Rumbaugh G. Pathogenic SYNGAP1 mutations impair cognitive development by disrupting maturation of dendritic spine synapses. Cell. 2012; 151:709–723. [PubMed: 23141534]

Eilemann, S.; Bilgili, A.; Abdellah, M.; Hernando, J.; Makhinya, M.; Pajarola, R.; Schürmann, F. Parallel Rendering on Hybrid Multi-GPU Clusters. Eurographics Symposium on Parallel Graphics and Visualization; 2012. p. 9

Fang Z, Lee JH. High-throughput optogenetic functional magnetic resonance imaging with parallel computations. J Neurosci Methods. 2013; 218:184–195. [PubMed: 23747482]

Franke F, Jackel D, Dragas J, Muller J, Radivojevic M, Bakkum D, Hierlemann A. High-density microelectrode array recordings and real-time spike sorting for closed-loop experiments: an emerging technology to study neural plasticity. Front Neural Circuits. 2012; 6:105. [PubMed: 23267316]

Kuhlman SJ, Olivas ND, Tring E, Ikrar T, Xu X, Trachtenberg JT. A disinhibitory microcircuit initiates critical period plasticity in visual cortex. Nature. 2013

Nageswaran JM, Dutt N, Krichmar JL, Nicolau A, Veidenbaum A. Efficient Simulation of Large-Scale Spiking Neural Networks Using CUDA Graphics Processors. Ieee Ijcnn. 2009:3201–3208.

Nauhaus I, Benucci A, Carandini M, Ringach DL. Neuronal selectivity and local map structure in visual cortex. Neuron. 2008; 57:673–679. [PubMed: 18341988]

Nauhaus I, Busse L, Carandini M, Ringach DL. Stimulus contrast modulates functional connectivity in visual cortex. Nat Neurosci. 2009; 12:70–76. [PubMed: 19029885]

Ohki K, Chung S, Ch'ng YH, Kara P, Reid RC. Functional imaging with cellular resolution reveals precise micro-architecture in visual cortex. Nature. 2005; 433:597–603. [PubMed: 15660108]

Olivas ND, Quintanar-Zilinskas V, Nenadic Z, Xu X. Laminar circuit organization and response modulation in mouse visual cortex. Front Neural Circuits. 2012; 6:70. [PubMed: 23060751]

Shepherd GM, Stepanyants A, Bureau I, Chklovskii D, Svoboda K. Geometric and functional organization of cortical circuits. Nat Neurosci. 2005; 8:782–790. [PubMed: 15880111]

Shi Y, Nenadic Z, Xu X. Novel use of matched filtering for synaptic event detection and extraction. PLoS One. 2010; 5:e15517. [PubMed: 21124805]

Shi Y, Ikrar T, Olivas ND, Xu X. Bidirectional global spontaneous network activity precedes the canonical unidirectional circuit organization in the developing hippocampus. J Comp Neurol. 2014; 522:2191–2208. [PubMed: 24357090]

Tomer R, Khairy K, Amat F, Keller PJ. Quantitative high-speed imaging of entire developing embryos with simultaneous multiview light-sheet microscopy. Nat Methods. 2012; 9:755–763. [PubMed: 22660741]

Wilson, JA. Using General-Purpose Graphic Processing Units for BCI Systems. 2011 Annual International Conference of the Ieee Engineering in Medicine and Biology Society (Embc); 2011. p. 4625-4628.

Xu X, Olivas ND, Levi R, Ikrar T, Nenadic Z. High precision and fast functional mapping of cortical circuitry through a combination of voltage sensitive dye imaging and laser scanning photostimulation. J Neurophysiol. 2010; 103:2301–2312. [PubMed: 20130040]

Yablonski, D. Numerical accuracy differences in CPU and GPGPU codes. Northeastern University ELECTRICAL AND COMPUTER ENGINEERING MASTER'S THESES; 2011.

Yang O, Cuccia D, Choi B. Real-time blood flow visualization using the graphics processing unit. Journal of Biomedical Optics. 2011:16.

Appendix 1: 1-D convolution benchmark code

```
%Initiate the vector for timing
result_cpu=zeros(8,1);
result_gpu=zeros(8,1);
%%Create the fixed length vector
b=rand(20000,1);
%% Start computation
for i =1:8
A=rand(10000 *2^i,1);
tic
tmp_CPU_result=conv(A,b);
result_cpu(i)=toc;
tic
tmp_GPU_result=conv(gpuArray(A), gpuArray(b));
GPU_result=gather(tmp_GPU_result);
result_gpu(i)=toc;
gpuDevice(1);
end
```

Appendix 2. Code comparison of the CPU and the GPU version of "synaptic_event_detection"

The code is simplified to emphasize the optimization of GPU performance. Note how the convolution of all traces are gathered together to minimize the data transfer overhead.

%% CPU CODE %%

```
for i=1 : size(Original_Trace,2)
%%High pass filter of original trace
sf = filtfilt(NUM,DEN,Original_Trace(:,i));
for j=1: cellcount-1
%% The filter is produced after de-mean and normalization
%% of the model.
m_dmean=Model{j}-mean(Model{j});
Filter = m_dmean/sum(abs(m_dmean));
%% Convolute signal trace with the model
```

```
temp=conv(sf,Filter);
%% Offset the convoluted trace so that it would better
%% align with the original trace.
fitting{j}=temp((1+round(length(Model{j})/2)...
+Phase_Diff(j)):(size(Original_Trace,1)...
+round(length(Model{j})/2)...
+Phase_Diff(j)));
end
%%%Misc code omitted
%% Identify events
vara=Align_Peaks(Original_Trace(:,i), fitting{j});
end
%Save results
```

%% GPU CODE%

```
%% Concatenate the filters into one single vector
for j=1:cellcount-1
m_dmean=Model{j}-mean(Model{j});
Filter = [Filter,m_dmean/sum(abs(m_dmean))];
F_Length(j+1)=F_Length(j)+length(m_dmean);
end
%% High pass filter of original trace
sf = (filtfilt(NUM,DEN,Original_Trace));
%% Convert the high passed signal trace to
%% a one-dimension vector
%% Transfer the data to GPU memory
sf=gpuArray(sf(:));
Filter=gpuArray(Filter);
F_Length=gpuArray(F_Length);
%% Assign memory in host for
%% storing convolution trace
result_gpu=cell(cellcount-1,1);
for j=1: cellcount-1
%% Computation on GPU,
%% then transfer data back to host memory.
tmp=Filter(gpuArray.colon(F_Length(j)+1,...
F_Length(j+1)));
result_gpu{j}=gather(conv(sf,tmp));
%% Offset the convoluted trace so that it would better
%% align with original trace.
result_gpu{j}=result_gpu{j}...
((1+round(length(tmp)/2)+...
Phase_Diff(j)):(size(sf,1)+...
```

```
round(length(tmp)/2)+Phase_Diff(j))...
);
end
for i=1 : size(Original_Trace,2)
%% Separate the convolution traces
%% for each trace
for j=1: Filter_Count-1
fitting{j}=result_gpu{j}...
((i-1)*10000+(1:10000));
end
%% Identify events
vara=Align_Peaks(Original_Trace(:,i),...
fitting);
end
```

Highlights

- We describe the use of GPU technology for large-scale neuroscience data processing.
- The GPU application focuses on neural circuit mapping-based data analysis.
- To our best knowledge, this is the first demonstration in this subfield of neuroscience.
- GPU computation allows for increased speeds while retaining data precision.

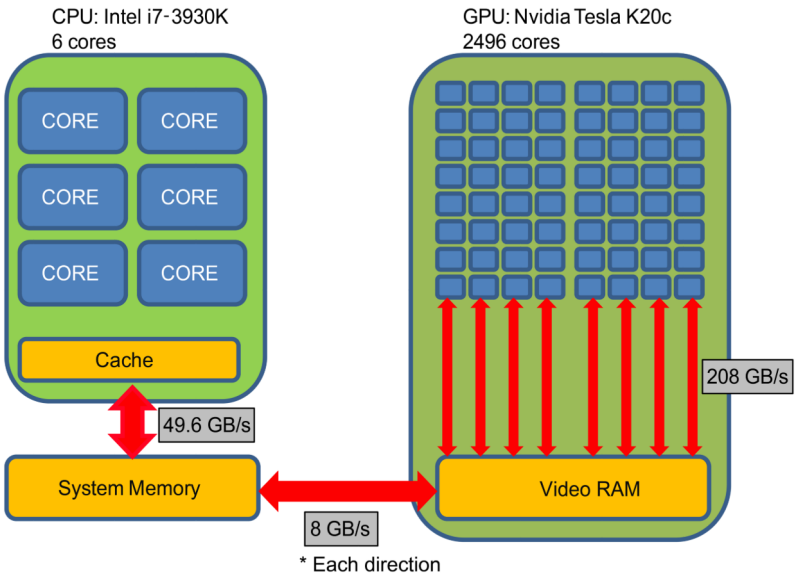

Figure 1. Architectural structures of the central processing unit (CPU) and the graphic processing unit (GPU) in our custom-made desktop computer system

We use the CPU of Intel i7-3930K (6 cores) and the GPU of Nvidia Tesla K20c (2496 cores). The double arrows indicate memory transfer for computation. The transfer speeds between the cache and the system memory (i.e., RAM), between GPU cores and the video RAM (VRAM), and between the VRAM and the system memory are 49.64 GB/sec, 208GB/sec and 8 GB/sec, respectively.

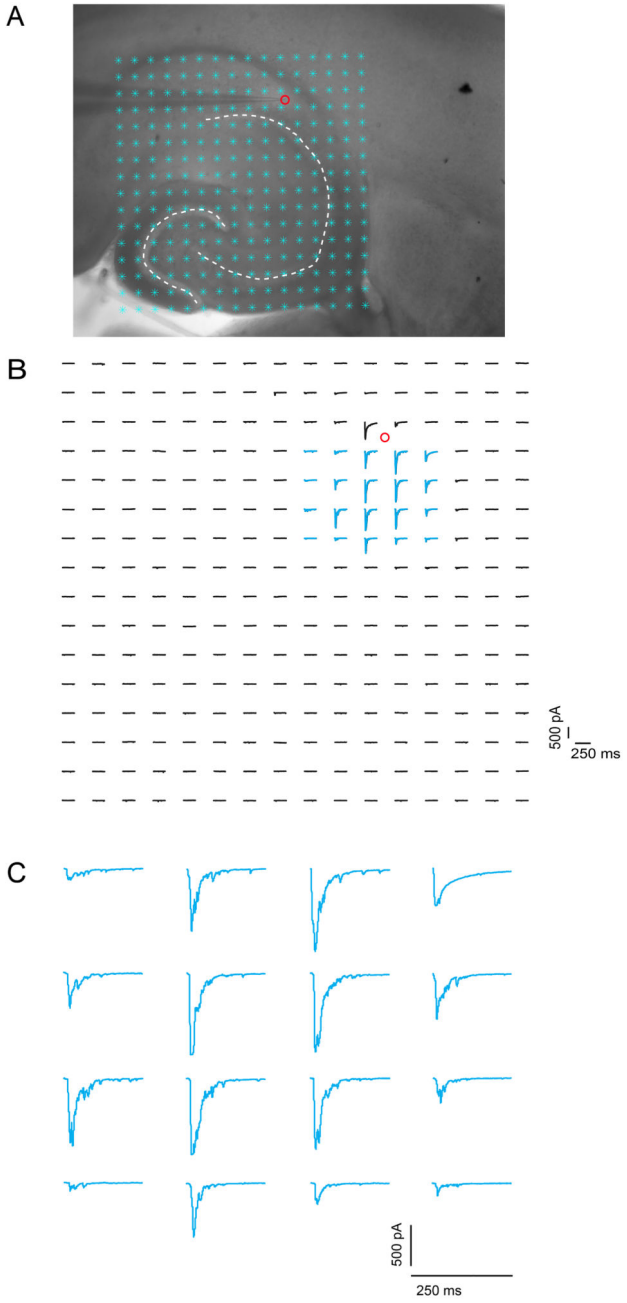

Figure 2. Illustration of laser scanning photostimulation (LSPS) combined with whole cell recordings to map local circuit input to a hippocampal CA1 interneuron

A shows a mouse hippocampal slice image with the superimposed photostimulation sites (16×16 cyan *, spaced at 100 μm × 100 μm). The glass electrode was recording from an interneuron in the oriens lacunosum-moleculare (OLM) layer of CA1. The red circle indicates the cell body location. B shows an array of photostimulation-evoked response traces from the photostimulation locations shown in A, with the cell held at −70 mV in voltage clamp mode to detect inward excitatory synaptic currents (EPSCs). Only the 250 ms of the recorded traces after the onset of laser photostimulation (1 ms, 20 mW) are shown. C

shows the enlarged and expanded traces from the highlighted traces in B. This type of experimentation generates large amounts of data, and requires intensive computation for post-hoc data analysis.

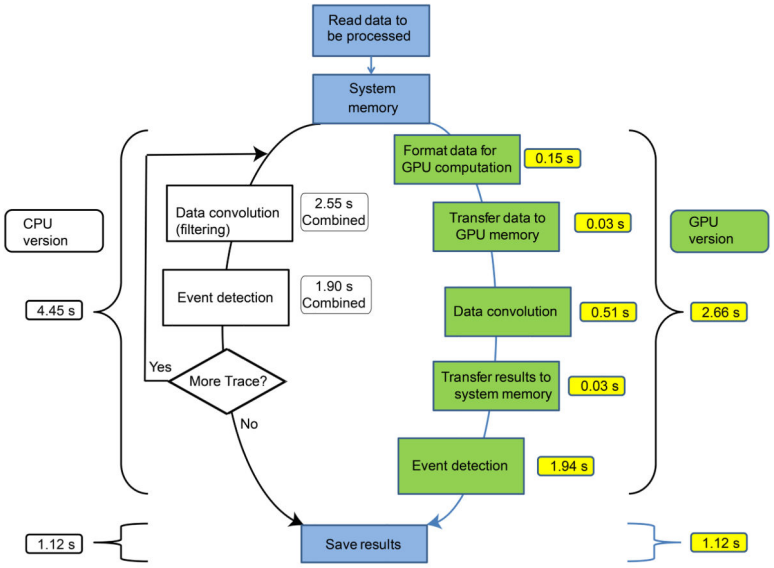

Figure 3. Example data processing flowchart for CPU and GPU computation

The data used here are from a typical LSPS map data set (shown in Figure 2B) with 256 signal traces (each containing 10000 sample points). The two versions share the steps of 'Read Raw electrophysiology trace', 'System Memory' and 'Save results'. They differ in major computation processes, with the left branch for CPU processing and the right branch for GPU processing. The computation that takes place in the GPU is coded by the green color. The processing time of each step is noted in yellow. For CPU computation in a FOR loop, the processing time for each iteration is summed up to get the combined time. The total processing times of CPU and GPU branches are marked outside of the brackets.

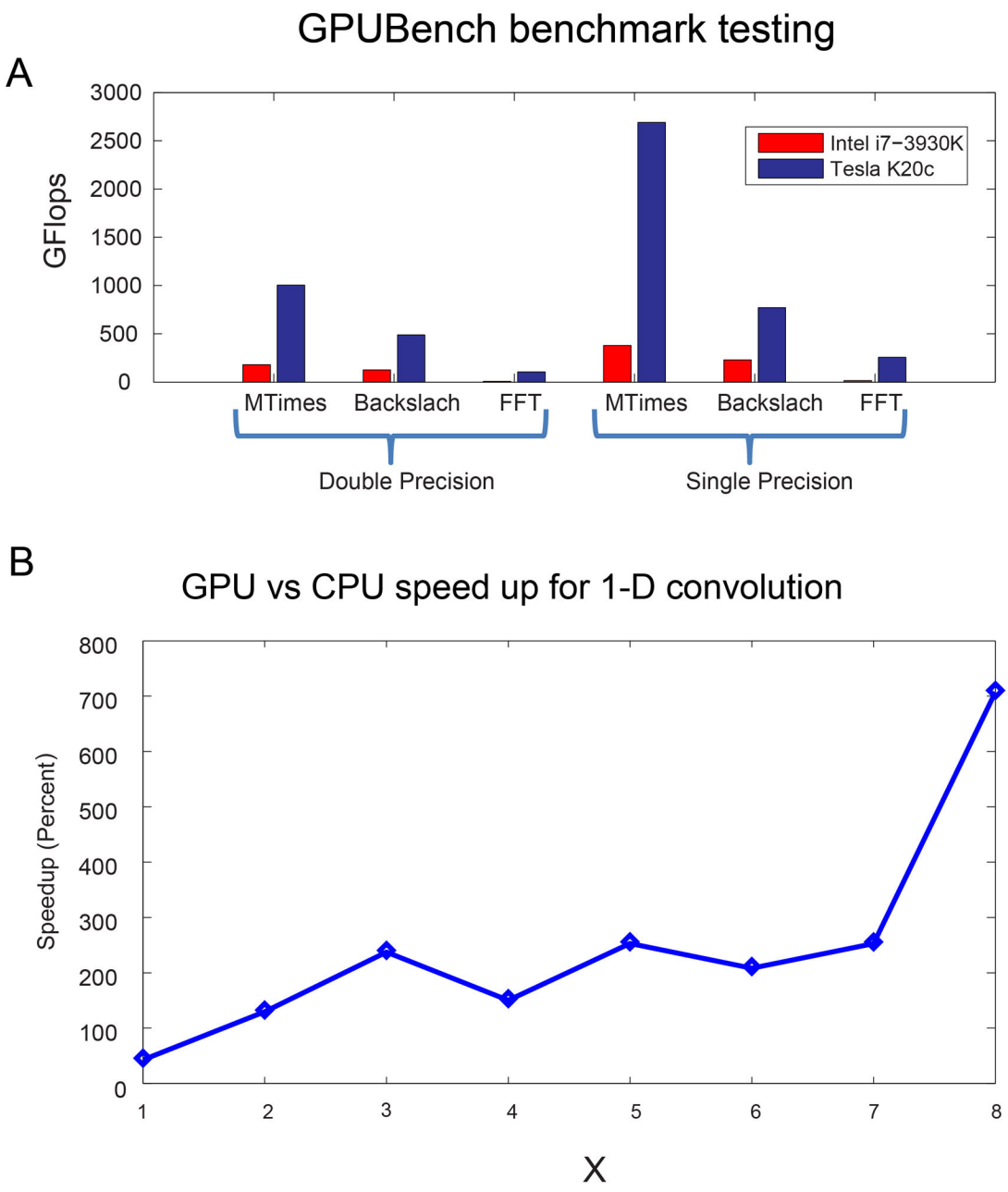

Figure 4. Basic comparisons of CPU versus GPU performance

A. Benchmark testing comparison of the CPU and GPU processing capabilities, assessed by their performance (GFlops) in three widely used computation, MTimes, Backslach and FFT, in the software platform of Matlab. B. GPU versus CPU speed up of 1-D convolution. Vectors of different lengths were generated by using the Matlab function, *rand()*. The vector length = $2^X * 10000$ ($X \in [1, 2, \ldots 8]$). The x-axis represents the value of X. The filter vector with its length of 20000 points was generated by the *rand()* function as well. The y-axis is the speed up of GPU convolution versus CPU convolution on the same vector and filter pair. The time of GPU convolution included the time of transferring data to and back from the GPU memory.

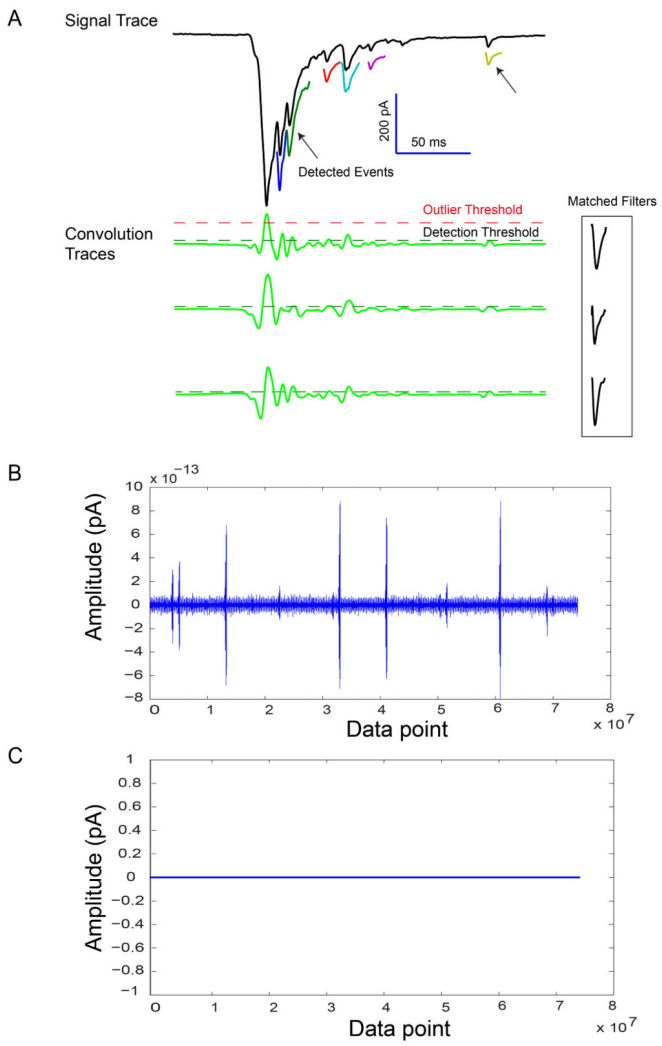

Figure 5. Data convolution accuracy via GPU computing

A. Example data trace convolution through GPU enabled computation. The top portion shows the actual signal trace with 6 EPSC events (color-coded) identified and extracted shown below. Note the first, large peak is determined to be a direct uncaging response. The bottom portion shows three convolved traces of the signal trace to the 3 example matched filters shown by the right side, with potential EPSCs having better fitting of the templates and exhibiting larger convolution amplitudes. For the subsequent event detection, the convolution traces are compared to a detection threshold and an outlier threshold (see the Method). **B–C.** The difference of CPU-based versus GPU-enabled convolution results of the same data trace. The x-axis represents data points while the y-axis shows the difference in convolution amplitude in the unit of pA. In B the y-axis scale is set to [−8, 10] $*10^{-13}$. C shows the same trace as shown in B while the y-axis scale is set to [−1, 1].

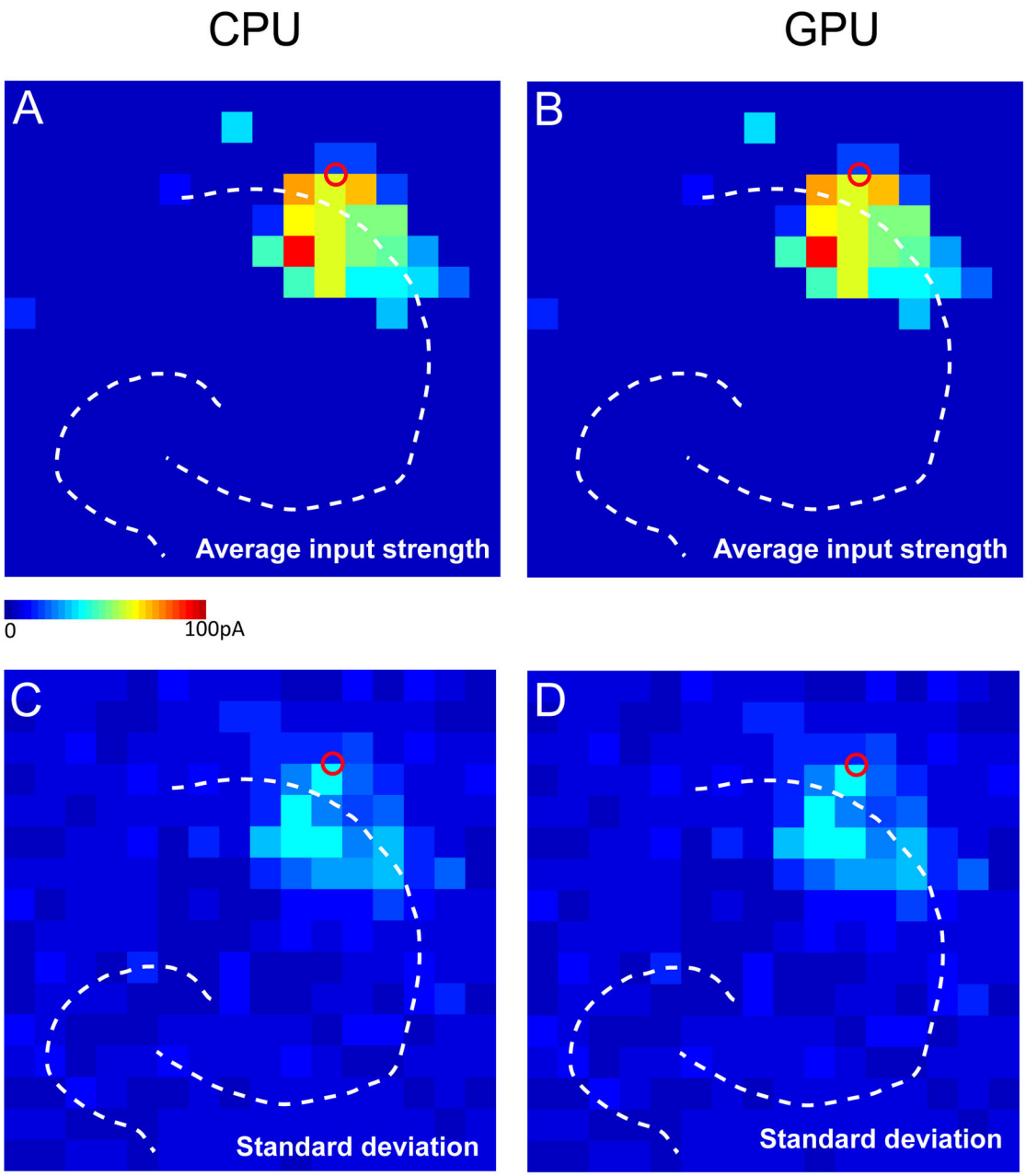

Figure 6. Visualization of the final detected events by using CPU and GPU-enable computation via the "Synaptic Event Detection" program

A–B, Color-coded maps of average input strength to the recorded cell, based on the detected EPSC events using CPU(A) and GPU(B) computation, respectively. Warm colors indicate stronger excitatory synaptic input from the photostimulation sites. The maps shown were constructed based on detection results of 8 raw data maps as shown in Figure 2B. **C–D**, The standard deviation maps of input strength across the 8 raw maps, using CPU(C) and GPU(D) computation, respectively. These map results are consistent and look identical.

Table 1

Structural components and general computation capabilities of the CPU and GPU used in our desk computer system.

Components	Model	Price, each ($)	Note
Hard Drive	Samsung 840 Pro	218	Two units In RAID 0 mode
Motherboard	ASUS Rampage IV extreme	439	
CPU	Intel i7-3930k	499	Running at 4.5 GHz
CPU Cooler	Corsair H100i closed loop liquid cooler	105.99	
System Memory	G.SKILL 32 GB Quad Channel DDR3	300	Running at 2.133 GHz
Graphic Card	Asus GTX 680 A455-0686	520	
CUDA card	Nvidia Tesla K20c	3,500	
Case	Cooler Master HAF X RC-942-KKN1	200	
Power	Corsair AX1200i	320	
Keyboard/Mouse	Logitech MK520 Combo	41.99	
Monitor	Asus PB278Q 27″	649	
Speaker	Creative T10	40	
Operating System	Windows 8 Pro x64	149	

All the data shown in the table are provided by the published documents from the manufactures (http://www.techpowerup.com/cpudb/858/core-i7-3930k.html; http://www.techpowerup.com/gpudb/564/tesla-k20c.html). The price info is as of June 30th, 2013.

Table 2

System and performance comparison of CPU and GPU-enabled computing

A. The manufacturer specifications								
	CORES			Memory		Processing Power(peak)(GFlops)		Power Consumption (Peak)(W)
	# of Cores	Frequency	# of Transistors	Bandwidth	Size	Single Precision	Double Precision	
CPU (Intel I7-3930k)	6	3.2GHz	2.27 Billion	Up to 51.2GB/s	Up to 64G B			130
GPU (Nvidia Tesla K20c)	2496	706MHz	7.1 Billion	208 GB/s	5GB	3520	1170	225

B. CPU and GPU benchmark testing results with GPUBench						
	Results for data-type ‘double’ (In GFLOPS)			Results for data-type ‘single’ (In GFLOPS)		
	MTimes	Backslash	FFT	MTimes	Backslash	FFT
CPU	174.38	126.78	7.92	375.05	245.17	11.53
Tesla K20c	1004.98	486.86	105.08	2657.88	772.11	254.95

C. 1D-convolution benchmark results								
x	1	2	3	4	5	6	7	8
CPU	0.006	0.024	0.0624	0.0644	0.187	0.2834	0.6574	3.5734
Tesla K20c	0.014	0.018	0.026	0.042	0.073	0.134	0.257	0.503

Note that the speedup is also shown in Figure 4B. The benchmark data length = 2^X * 10000. The results are the running times of the convolution in seconds.