Highly Scalable Parallel Processing of Extracellular Recordings of Multielectrode Arrays

Tiago V. Gehring[1], Eleni Vasilaki[1,2,3], Michele Giugliano[1,2,4]

***Abstract*— Technological advances of Multielectrode Arrays (MEAs) used for multi- site, parallel electrophysiological recordings, lead to an ever increasing amount of raw data being generated. Arrays with hundreds up to a few thousands of electrodes are slowly seeing widespread use and the expectation is that more sophisticated arrays will become available in the near future.**

In order to process the large data volumes resulting from MEA recordings there is a pressing need for new software tools able to process many data channels in parallel. Here we present a new tool for processing MEA data recordings that makes use of new programming paradigms and recent technology developments to unleash the power of modern highly parallel hardware, such as multi-core CPUs or GPUs.

Our tool builds on and complements existing MEA data analysis packages. It can be used to speed up some performance critical pre-processing steps such as data filtering and spike detection, helping to make the analysis of larger data sets tractable.

I. INTRODUCTION

Technological advances in the construction of MEAs lead to collections of an increasing amount of data which has to be analysed. For example, a simple calculation shows that recordings employing new large-scale MEAs with 4096 electrodes [1] with a sampling frequency of $7.7\,\mathrm{kHz}$ generate about $63\,\mathrm{MiB/sec}$, ($2 \times 4096 \times 7700$ - assuming a $16\,\mathrm{bit}$ analog-digital converter), or almost $230\,\mathrm{GiB}$ of raw (uncompressed) data per recorded hour. It is expected that future technology advances will lead to even larger amount of data being generated which has then to be processed.

Typical processing pipelines of MEA data recordings usually involve a filtering step followed by a spike detection and artefact removal algorithm [2]. These pre-processing steps are then typically followed by a spike sorting algorithm and/or other post-processing steps, such as burst-detection in the case of cultured cells [3]. In order to simplify and automatise these processing steps different software packages were proposed over the years [4], [5], [2]. These pioneering tools, however, were designed primarily with ease of use and not performance in mind, or were designed during a time were parallel hardware was not so prevalent as it is today. They therefore cannot make efficient use or do not scale well on modern hardware architectures (e.g. multi-core CPUs with vector instruction sets or GPUs).

Here we present a new MEA data processing tool, PSpike, that offers high scalability and performance on single machine parallel hardware architectures. The new tool is based on OpenCL (Open Computing Language), an industry standard for programming multi-core CPUs, GPUs, and dedicated accelerators.

The focus of the new tool is to optimise the performance-critical, pre-processing steps of the data processing flow which dealing with large amounts of data. The objective of the new tool is not to re-create completely new analysis frameworks but rather to complement and speed-up existing ones. For this reason the processing steps performed here match closely the ones from the QSpike tools [2], a previously released and freely available software package. The results from the pre-processing stages of both tools are interchangeable, so that the available report generating facilities provided by the QSpike tools can be used without major modifications with the output data from the new tool.

II. METHODS

A. Code

The software tool presented here, PSpike, was developed in C++ and is platform agnostic (the code was tested on Linux, Windows, and OSX systems). Both a command line tool for manual processing of offline recordings (Mutichannel Systems' MCD files are currently supported) and a daemon that continuously checks for new files and processes them are provided.

1) OpenCL: Modern CPUs consist of multiple cores and rely increasingly on SIMD (single instruction multiple data) extensions, such as SSE (Streaming SIMD Extensions) and more recently AVX (Advanced Vector Extensions), to increase throughput. Other dedicated highly parallel hardware, such as GPUs or Intel's MIC (Many Integrate Core), available in the form of the Xeon Phi line of dedicated accelerators, are becoming increasingly widespread.

OpenCL is a widely used and supported framework for writing programs that target modern parallel hardware architectures (see [7] for a short introduction, [8] for a detailed treatment). The main advantages of using OpenCL, besides being an industry standard, is that it vastly simplifies and abstracts the programming of parallel code that has to run in multiple hardware configurations. This is achieved by means of OpenCL kernels, or code which is compiled at runtime. The kernels are just special functions written in a subset of the C programming language. Because they are compiled at

[1]Department of Computer Science, University of Sheffield, Sheffield, UK t.gehring@sheffield.ac.uk

[2]Theoretical Neurobiology and Neuroengineering Lab, Department of Biomedical Sciences, University of Antwerp, Wilrijk, Belgium

[3]INSIGNEO Institute for in Silico Medicine, University of Sheffield, Sheffield, UK

[4]Laboratory of Neural Microcircuitry, Brain Mind Institute, Swiss Federal Institute of Technology of Lausanne, cole Polytechnique Fdrale de Lausanne, Switzerland

runtime, the code can be highly optimised for the specific hardware that will run it (e.g. a 4-core CPU or a GPU with thousands of simplified cores). Fig. 1 shows the general architecture of an OpenCL application.

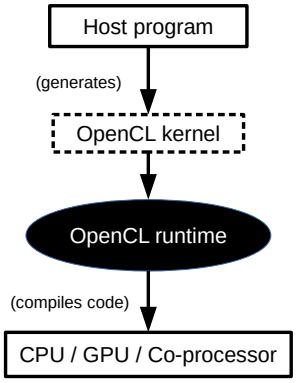

Fig. 1. Overview of the OpenCL programming and runtime architecture. The host application generates the OpenCL kernels, or special functions, which are then compiled at runtime targeting the specific hardware on which the code is to be run (e.g. a multi-core CPU or a GPU). Because of the runtime compilation highly optimised code can be generated.

The performance critical functions of our tool, such as the data filtering and spike detection, were written in the form of OpenCL kernels in order to extract maximum performance from the available hardware.

2) Block processing, MCD files: Our tool has currently built in support for processing Multichannel Systems' MCD files. However, other file formats or data sources such as online recorded data, can be easily supported by adding a new data source type to the code (any source which provided a stream of continuous data can be used).

The core algorithms the code works on data blocks, which can be read an processed in parallel. This can speed up the processing considerably as, for example, large MCD files can be broken down into smaller blocks and the reading and processing of data can progress concurrently. It is to be noted, however, that using different block sizes can have an impact on the results as the results of the backward filtering step (see below) and the estimate of the spike detection threshold will vary slightly. Block processing is therefore optional (as long as there is enough system memory available MCD files can be processed as one block); the block size can also be specified manually.

B. Processing workflow

Fig. 2 shows the different data processing steps performed by the tool. Except by the generalised concept of input and output streams and block processing (described above), these steps are exactly the same as the ones performed by the original QSpike tools implementation in Matlab [2]. When a single data block is used the results of both tools is exactly the same.

1) Filtering: The first step of the processing performs an (optional but active by default) band pass filter of the data using an IIR (Infinite Impulse Response) filter. By default a bi-directional (non-casual) 2nd order elliptic filter with a

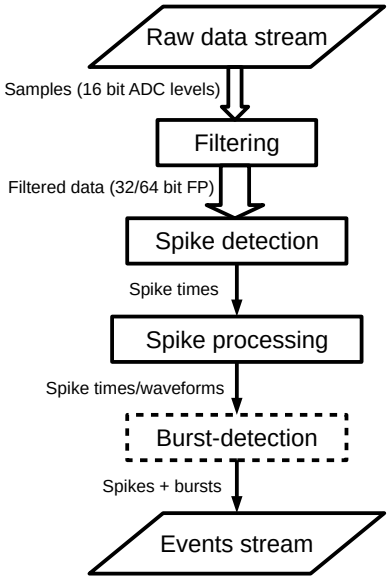

Fig. 2. Overview of the data processing steps. Wider arrows symbolise steps where larger amounts of data have to be processed.

$400-3000\,\mathrm{Hz}$ pass band is used. This can however easily be changed by providing custom filter coefficients or a custom filter coefficient file with multiple filter coefficients (one set for each sampling frequency). A non-casual (or zero-phase) filter (which requires a forward and a backward pass over the data) is chosen by default in order to minimise spike shape distortion and time lags [9]. This can, however, be easily changed by a casual (single pass) filter with a single line code change.

It is to be noted that the filtering step also expands the input data from discrete ADC (Analog Digital Converter) values (usually $16\,\mathrm{bit}$ integers) to 32 or $64\,\mathrm{bit}$ floating point values (depending on the compile time flags) in the first pass. The data conversion and filtering operations are combined to avoid an extra time consuming data copy operation.

2) Spike detection: The spike detection algorithm is based on a fixed threshold with is estimated from the background noise of the data. The noise is computed from the median of the signal, as this was shown to be a better estimator than the standard deviation [6]. For the median computation a recursive binning algorithm is used [10].

3) Burst detection: The burst detection uses the method described in Van Pelt et al. [11], which first bins the spikes for discrete time windows and then calculates the number of active electrodes (ones in which a spike was detected) at each time step. A burst is detected if the number of active electrodes cross a fixed threshold.

4) Post processing: Our new code does not provide any new post-processing code, such as report generation, but rather makes use of the already available QSpike tools software stack. Only minor modifications had to be made for the QSpike post-processing code to be compatible with the output generated by our new PSpike tool. As an example of post-processing output, Fig. 3 shows a raster plot produced by QSpike tools.

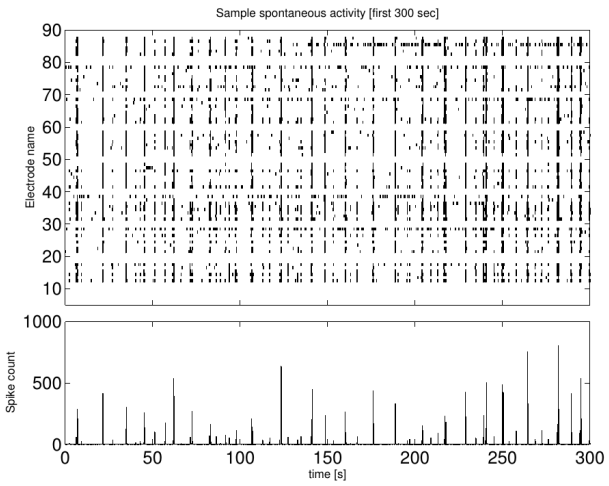

Fig. 3. Example raster plot generated by the QSpike tools framework

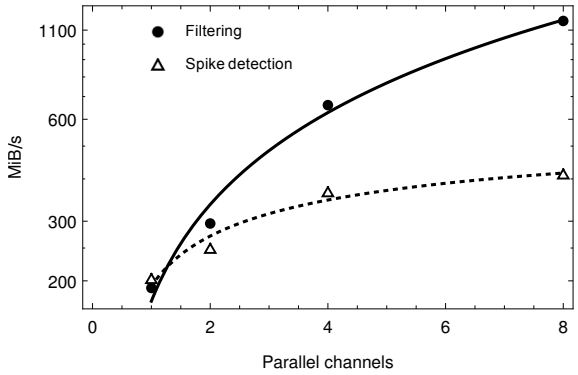

Fig. 4. Performance scaling of filtering and spike-detection operations. Hardware: Intel Core i7-4600U 2.1 GHz, 2 cores, 4 threads. OpenCL implementation: Intel OpenCL runtime. Filtering shows close to linear scaling with the number of parallel channels; performance starts to flatten only when saturating all hardware lanes (2 physical cores with 256 bit SIMD instructions). In the case of spike detection the code cannot make full use of vector instructions due to branching in the code; the performance peak is therefore reached at 4 parallel channels, or the number of hardware threads.

III. RESULTS

A. Performance scaling

Fig. 4 shows how the main pre-processing operations, data filtering and spike detection, scale with different number of channels being processed in parallel. As the graph shows filtering scales almost linearly when processing up to 8 channels in parallel. This can be explained by the fact that vectorization of the filter is simple since it consists only of arithmetic operations. These operations can then be replaced by their SIMD counterpart (SSE or AVX instructions) and therefore process multiple channels at a time on a single hardware thread (4 for the CPU used in the tests which supported 256 bit AVX instructions). Peak performance of the bi-directional filter used in the tests is close to $1.2\,\mathrm{GiB/s}$ even on the relatively modest hardware used here, showing that it does not represent a performance bottleneck.

Scaling of the spike detector, however, does not scale as well as the data filter (Fig. 4). This is due to the fact that vectorization of the spike detector is much harder because of the code branching involved. The branching arises from the conditional check for a threshold crossing, which signals that a spike was found. The only part of the spike detection code that was vectorized is the standard deviation (SD) computation; the SD is used for the median computation which in turn is used to estimate the background noise and the spike threshold for each channel. The performance gains here, however, are overshadowed by the more computationally intense parts of the spike detection algorithm that are not vectorized, so that the code scales poorly and performance peaks when the number of hardware threads is reached (4 in this case).

B. Processing of offline data

Fig. 5 compares the overall time to process a MCD file (Multichannel Systems) when using different numbers of parallel channels in the processing. The results show that performance scales up to 8 parallel channels, due to the hardware limitations of the test setup (2 physical cores, 4 thread Intel Core i7 CPU). The figure compares also results using different OpenCL implementations: the Intel OpenCL runtime, which comes as binary, closed-source libraries and pocl (portable-computing language) [12], an open-source OpenCL implementation that aims to be easily portable to different architectures. As the results show, the pocl implementation offers a major improvement over Intel's libraries. This because the former was compiled from the sources and is highly optimised to the underlying hardware during its installation.

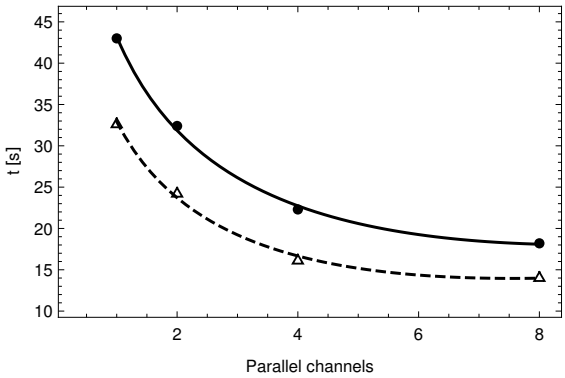

Fig. 5. Time to process a 1.8 GiB MCD file with 60 data channels sampled at 25 kHz (10 s recording) using the Intel OpenCL runtime (solid line) and the pocl OpenCL implementation (dashed line). Hardware: Intel Core i7-4600U, 2 cores, 4 threads (see Appendix for details of the system used). Performance is again seen to scale well until the maximum number of hardware lanes is reached. The pocl OpenCL implementation, which is compiled from the sources and highly optimised to the hardware at hand, offers a major performance advantage over the Intel libraries (binary only).

Fig. 6 shows performance results of the new and original code of QSpike tools. The performance of the new code is measured for different block sizes. As can be seen performance improves with decreasing block size. This is due to

the fact that smaller blocks lead to increased levels of reading and processing parallelism. The rightmost point, with a block size as large as the file, means that reading and processing operations are sequential, as is the case in the original Matlab implementation. As described in the Methods Section, the block size does have an impact on the results as the spike threshold with varying block sizes will vary slightly.

The results show that the new implementation of the pre-processing stages can outperform the original one even when the former is running in a much more modest machine (a 2013 Ultrabook class Laptop compared to a 12 core workstation with 72 GB of RAM).

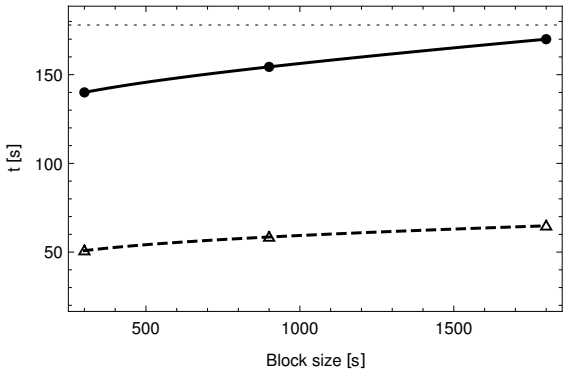

Fig. 6. Time to process a 5.4 GiB MCD file with 60 data channels for different block sizes. Sampling frequency: 25 kHz (30 s recording). Hardware and software configurations are given in the Appendix. Continuous and dashed lines show results using Intel's OpenCL runtime and pocl OpenCL implementation respectively. Dotted line shows the performance of the original QSpike tools pre-processing stage running on a dual-socket, 12 core, 24 thread Intel Xeon system with 72 GB of RAM. As the results show the highly optimised pocl OpenCL implementation offers again a large (over 3 times in this case) performance advantage over Intel's libraries. In both cases the performance of processing improves when using smaller block sizes because this allows for data extraction and processing to run in parallel (the rightmost point is the point where both operations are sequential - first all data is read and then processed, as is done in the original implementation in Matlab). In all the cases the new pre-processing code is able to outperform the original even when running on the much more modest hardware setup.

IV. CONCLUSIONS

Our results show that by employing modern programming technologies, such as OpenCL, highly efficient software that can target modern parallel hardware architectures can be developed. This is especially well suited for processing the large data sets generated by multi-electrode array recordings. Parallelization does not pose major difficulties because data generated from multi-electrode recordings is inherently parallel because the individual data channels are independent from one another.

As technology advances the tendency is for multi-site recordings with an increasingly large number of recording channels to become commonplace. At the same time modern hardware architectures are also becoming increasingly parallel. This means that software tools that can extract the maximum performance from state-of-the-art hardware will become increasingly important as well.

In the present work, we focus on improving the performance of the pre-processing stages of MEA data analysis frameworks, such as QSpike. Our results demonstrate substantial performance improvements of data analysis compared to the original pre-processing stages of the QSpike tools. This leaves space for future improvements, such as parallelization of the spike sorting algorithm. We expect that moving to an all parallel implementation of the QSpike framework will result in further performance improvements, reducing the requirements of dedicated high specs computers.

APPENDIX

A. System setup

All benchmarks were conducted on an Intel i7-4600U (Haswell) @ 2.1 GHz laptop with 8 GiB RAM and a Samsung 830 SSD. The operating system was Gentoo Linux 64 bit, Kernel version 3.17.4. For the OpenCL runtime Intel OpenCL runtime version 4.4.0.117 and pocl library version 0.11 were used.

ACKNOWLEDGEMENT

This work was supported by the EC-FP7-PEOPLE sponsored NAMASEN Marie-Curie Initial Training Network (grant n. 264872).

REFERENCES

[1] Ferrea, E., Maccione, a, Medrihan, L., Nieus, T., Ghezzi, D., Baldelli, P., Berdondini, L. (2012). Large-scale, high-resolution electrophysiological imaging of field potentials in brain slices with microelectronic multielectrode arrays. Frontiers in Neural Circuits, 6(November), 80. doi:10.3389/fncir.2012.00080

[2] Mahmud, M., Pulizzi, R., Vasilaki, E., Giugliano, M. (2014). QSpike tools: a generic framework for parallel batch preprocessing of extracellular neuronal signals recorded by substrate microelectrode arrays. Frontiers in Neuroinformatics, 8(March), 1-14.

[3] Van Pelt, J., Wolters, P. S., Corner, M. a, Rutten, W. L. C., Ramakers, G. J. a. (2004). Long-term characterization of firing dynamics of spontaneous bursts in cultured neural networks. IEEE Transactions on Bio-Medical Engineering, 51(11), 20516

[4] Egert, U., Knott, T., Schwarz, C., Nawrot, M., Brandt, a., Rotter, S., Diesmann, M. (2002). MEA-Tools: An open source toolbox for the analysis of multi-electrode data with MATLAB. Journal of Neuroscience Methods, 117(1), 3342.

[5] Wagenaar, D., Demarse, T. B., Potter, S. M. (2005). MeaBench: A toolset for multi-electrode data acquisition and on-line analysis. 2nd International IEEE EMBS Conference on Neural Engineering, 2005, 518521.

[6] Quiroga, R. Q., Nadasdy, Z., Ben-Shaul, Y. (2004). Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. Neural Computation, 16(8), 166187.

[7] Tompson, Jonathan, and Kristofer Schlachter. An Introduction to the OpenCL Programming Model. Digital version (2012).

[8] Munshi, Aaftab, et al. OpenCL programming guide. Pearson Education, 2011.

[9] Quian Quiroga, R. (2009). What is the real shape of extracellular spikes? Journal of Neuroscience Methods, 177(1), 1948.

[10] Tibshirani, Ryan J. (2008) Fast computation of the median by successive binning. Unpublished manuscript, http://stat.stanford.edu/ryantibs/median.

[11] Van Pelt, J., Vajda, I., Wolters, P. S., Corner, M. a, Ramakers, G. J. a. (2005). Dynamics and plasticity in developing neuronal networks in vitro. Progress in Brain Research, 147, 17388.

[12] Jskelinen, P, de La Lama, C. S., Schnetter, E, Raiskila, K., Takala, J., Berg, H. (2014). pocl: A Performance-Portable OpenCL Implementation. International Journal of Parallel Programming, 1-34