Abstract – A data-driven prototype software is presented for EEG processing and visualization. The system relies on the GPU architecture for providing simultaneous processing and visualization of the EEG data. Two example brain imaging algorithms, the surface Laplacian and the spherical forward solution are used for illustrating the effective use of the massively parallel GPU hardware in speeding up computations. The paper describes the architecture of our system, the key design decisions, and the performance optimization of the parallel implementation. Using the CUDA-OpenGL interoperability, the computing subsystem can directly modify potential data in the OpenGL vertex memory, avoiding unnecessary GPU-Host data transfers. The system and our parallel implementations demonstrate that real-time processing and visualization is possible for a range of algorithms during EEG processing. We are confident that these results can pave the way for supercomputing-class implementations and open up new opportunities in the clinical practice and neuroscience research.

I. INTRODUCTION

High-resolution EEG imaging is an increasingly important tool in neuroscience [1]. Unlike CT or MRI that only provide structural information, EEG can measure functional brain activity. Its time resolution typically is in the millisecond range which is vastly superior to other functional imaging methods, such as PET or fMRI. Consequently, EEG is indispensable in areas where high temporal resolution brain activity mapping is required, e.g. in epilepsy diagnosis in the clinical setting [2], [3] or in cognitive experiments based on evoked or event-related potentials [4].

The neuroscience community relies on commercial as well as open source software products for EEG processing. The commercial systems are more common in the clinical area, whereas open source packages, such as the EEGLab [5], Fieldtrip [6] or CSD [7] Matlab toolboxes, dominate in the research labs. While the open source approach reduces development time, facilitates sharing of methods and data as well as serves as a verification infrastructure, by being a de-facto standard, halts innovation in the area of EEG processing software. We argue that innovation is required as the processing speed achieved during routine evaluations is intolerably low.

Since the state-of-the-art EEG systems can employ up to 256 electrodes and operate at as high as 16 kHz sampling frequency, even a few second measurement in such a system can generate hundreds of MBBytes of data. The evaluation of these experiments results in long execution times (varying from minutes to hours depending on methods, data size and control parameters). This can be somewhat tolerated in single-shot experiments but it clearly presents a serious obstacle in large multi-patient studies or if EEG is ever to be considered as a diagnostic tool in the daily clinical routine.

This paper presents a novel parallel computing approach to EEG data processing and describes an experimental software environment developed at the University of Pannonia. We have developed a GPU-based massively parallel system that demonstrates that near real-time processing speed is achievable in key brain imaging algorithms, such as the spherical surface Laplacian [8] or the forward solution for source localization [9]. Although the program is created for EEG brain imaging, it can be used after little modifications for ECG-based multi-channel body-surface mapping applications [10], [11], [12].

At the center of our implementation is the GPU architecture that acts as a computing and visualization engine. The most advanced cards today can achieve up to 5 teraflops (10^{12} floating point operations per second) computational performance on a single chip using approximately 100 W power consumption. This performance-per-watt value cannot be matched in any other CPU-based parallel system. We use OpenGL [13] for visualization and the NVIDIA CUDA [14] technology for the parallel implementation on the GPU.

The structure of our paper is the following. Section II describes two illustrative imaging method, the surface Laplacian mapping as well as the forward algorithm indispensable in solving the inverse problem. Both approaches aim at detecting the spatial and temporal patterns for cortical brain activity. Section III describes the rationale behind using graphical processors for increasing processing speed as well as providing data visualization capabilities in our program. We highlight peculiarities of these systems from the algorithm design and optimization point of view. Section IV explains the core architecture of our software and the key design concepts, while Section V discusses our results to date.
II. EEG IMAGING METHODS

The electric potential measured on the scalp is the combined result of different activity sources in the brain. The potential is rarely usable as recorded due to the low signal-to-noise ratio and the presence of unwanted artifacts, such as the effect of eye movements, hence require filtering and other pre-processing steps. Nonetheless, the filtered potential data is seldom used on its own for activity mapping. A fundamental problem in the interpretation of scalp potential maps and in the solution of the forward/inverse problem is the so-called smearing effect of the skull. As the electric field propagates within the head, regions with the different conductivity spread the effect of the original sources, making their size grow on the scalp.

A. The Surface Laplacian

Laplace imaging can be used to ‘sharpen’ the scalp potential images, i.e. to reduce the smearing effect. The surface Laplacian [15], [16], [17] – the second derivative of the surface potential – computes the current source density (CSD) of the brain that is strictly related to the activity level of cortical areas. Several methods are known that apply the Laplace operator to focus the potential image before solving the inverse problem [18], [19], [20], [17].

The advantage of the Laplacian is that it is independent of the reference electrode choice, relatively insensitive to eye-movement artifacts and generates a relative topographic image that is well suited to activity pattern recognition. The Laplacian of the potential filed in the three-dimensional Cartesian coordinate system is expressed as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$$

Where $f$ is the potential function represented by the discrete potential values measured at the electrodes. One of the most widely used algorithm for computing the surface Laplacian on a sphere was proposed by Perrin [8], [21], which uses spherical splines

For a point $E$ on the sphere, the Laplacian (a.k.a. the current source density), $C$, is given as

$$C(E) = \sum_{i=1}^{n} c_i h(cos(E, E_i)),$$

where $E_i$ is the $i$th electrode on the surface of the sphere, and the $c_i$ are the solution of

$$GC + TC_0 = Z$$

$$T'C = 0$$

with

$$T' = (1,1, ..., 1), C' = (c_1, c_2, ..., c_n),$$

$$Z' = (z_1, z_2, ..., z_n), G = (g_{ij}) = (g(cos(E_i, E_j)))$$

where $z_i$ is the potential measured at electrode $E_i$. The function $h(x)$ is defined as the sum of the following series

$$h(x) = \frac{1}{4\pi} \sum_{n=1}^{\infty} \frac{(2n + 1)}{n^{m-1}(n + 1)^{m-1}} P_n(x)$$

where $P_n(x)$ is the $n$th degree Legendre polynomial and $m$ is the stiffness parameter of the spline.

A typical application of the surface Laplacian is in event-related experiments, in which normally multi-trial average of a 5-10 second time window of brain activity is analysed. In the widely used Matlab-based CSD Toolbox [7] program, the calculation is carried out in two steps; first, an $H$ matrix is created by solving $h(x)$, than the Laplacian is calculated for the entire data window. The Matlab execution times are given in Tables I and II measured on an Intel Core i7-3820 2.70 GHz processor. Table I shows the calculation time of the $H$ matrix for a different number of electrodes and summation limits for $h(x)$. Table II shows the overall execution time including the $H$ matrix and the window calculations for $N = 15$. Even at this summation limit, it is evident that one window can take 10s of seconds to calculate. For $N = 50$, the execution time moves to the order of minutes.

### TABLE I. Calculation time [sec] of the H matrix (CSD)

<table>
<thead>
<tr>
<th>Number of electrodes</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>5.74</td>
<td>22.84</td>
<td>92.50</td>
</tr>
<tr>
<td>50</td>
<td>34.00</td>
<td>137.51</td>
<td>552.12</td>
</tr>
</tbody>
</table>

### TABLE II. Total Surface Laplacian calculation time [sec], N=15

<table>
<thead>
<tr>
<th>Number of samples</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.74</td>
<td>22.84</td>
<td>92.51</td>
</tr>
<tr>
<td>512</td>
<td>5.81</td>
<td>23.03</td>
<td>92.89</td>
</tr>
<tr>
<td>1024</td>
<td>5.89</td>
<td>23.15</td>
<td>93.28</td>
</tr>
<tr>
<td>2048</td>
<td>6.04</td>
<td>23.47</td>
<td>94.07</td>
</tr>
<tr>
<td>5120</td>
<td>6.50</td>
<td>24.40</td>
<td>96.33</td>
</tr>
<tr>
<td>7160</td>
<td>6.80</td>
<td>25.03</td>
<td>97.94</td>
</tr>
</tbody>
</table>

B. The forward solution

While the surface Laplacian can help in focusing the potential image obtained on the scalp, it does not contain information about the cortical placement of the activity sources. Source localization methods can help in extracting this information, whose aim is to identify the location and magnitude of activity sources (dipoles) based on the measured potentials. The difficulty lies in the problem that any number of dipole combinations can generate identical scalp potential maps. This is the bioelectrical inverse problem which has no unique solution except when a priori constraints can be applied.

The basis of the inverse solution is that one or more hypothetical dipoles are placed in the brain within the head model and the generated potentials are computed – this is called the forward problem – then compared to the measured values. During the process, the position, orientation and magnitude of the dipoles are varied until an acceptable solution is found. Since there is no theoretically unique solution (except in the case of a single equivalent dipole), several solution methods have been proposed. Reviews and comparisons of these and other
frequently used inverse solution methods can be found in [22], [23], [24], [25].

The inverse solution requires the computation of hypothetical dipole generated potentials, known as the bioelectrical forward problem. This is based on the electrical field computation in the head model. The simplest head model is the spherical models that can contain up to five layers. Spherical models have closed form solutions, therefore they are computationally less demanding. Various solutions are developed for the multi-layer spherical head model forward calculation [26], [9]. The main advantage of the spherical head models is computational efficiency. Due to the closed form solution, coefficients for each layer given as layer spherical head model forward calculation [26], [9].

The widely used Fieldtrip Matlab toolbox [6] contains the implementation of the 4-layer spherical forward solution algorithm proposed by Cuffin and Cohen [9]. The computation of the generated potential is based on the following equation that calculates the effect of the x component of a source dipole vector. Similar equations are used for the y and z components.

\[
V = \frac{P_n \cos \theta}{4\pi \sigma R^2} \sum_{n=1}^{\infty} \frac{(2n+1)^4 f^{n-1}(c * d)^{2n+1} P_n^1(\cos \theta)}{n!}
\]

where

\[P_n^1(\cos \theta) = \frac{\partial^2}{\partial \cos \theta^2} P_n(\cos \theta)\]

\[\Gamma = \frac{d^{2n+1}}{n!} \frac{(k_1 - 1)(k_2 - 1)(n + 1)}{n!} \]

\[f = \frac{b^2}{b^2 + c^2 + d^2} \]

\[d = \frac{c}{b - c} \]

\[R = \frac{d}{b} \]

The traditional approach to speeding up the illustrated algorithms is to implement the algorithms in C/C++ and ideally write it in a parallel manner so it could scale over a range of CPU cores. Unfortunately, the achievable parallelism is limited in traditional computers as the number of cores is limited to 8 on Core i7 systems and 16 on state-of-the-art Xeon processors. We ignore the distributed-memory systems using multiple CPUs as they are not typical in the medical field.

### III. EMBRACING GPU TECHNOLOGY

A. GPU overview

Graphical processors, due their origin in 3D graphics, are designed for massively parallel execution of pixel graphics instructions. Today’s GPUs are generally programmable and can be used the general computational tasks. State-of-the-art chips can contain over 2000 cores and their computational performance can exceed 4 teraflops. The key to this performance is the very efficient and coordinated execution of thread groups. GPU programs are ideally data-parallel, executing the same instruction sequence for a large amount of data. Threads can be arranged into 1D, 2D or 3D grids and efficiently mapped onto physical cores.

GPUs run the program from their own global memory that resides on the GPU card. They also use faster memories, such as the shared memory that can be used by a group of threads or registers for the exclusive use of a single thread. Global memory access is very expensive, thousands of instructions can be executed with a single read/write cycle. Even more expensive is the data transfer between the host and the GPU, shown in Fig. 1. While the GPU is treated by many as a co-processor, the efficient use requires that the GPU can execute at its own speed without the tight control of the CPU program.

B. Parallel execution strategy

Our primary aim was to explore various GPU parallelization strategies and to compare different implementations in order to a) speed up computation as
much as possible as well as to b) investigate the effect of design decisions on performance.

Both algorithms fall into the data-parallel class of problems. The first design decision is how to partition the problem in a scalable way, the second is how to use the memory the most efficient way and the third is how to minimize the number of cycles not spent with computation.

1) The GPU surface Laplacian

In contrast to the original surface Laplacian algorithm, in which the Laplacian is only computed for the electrode coordinates, in our system we are interested in the values at every position of the head. Our implementation computes interpolated values for a larger set or points. The exact number of points is determined by the resolution of the 3D sphere mesh used to build the head model.

The problem is partitioned by the surface points. The Laplacian at each point is computed independently from the other points. Since the spline interpolation is a global operation, all the electrodes are shared by the parallel threads (kernels).

2) The GPU forward solver

Our parallel forward solver implementation is based on the work reported in [27]. The underlying structure of the algorithm is a dipoles × channels thread grid, in which each thread calculates the effect of one dipole on one scalp electrode. We have added several performance optimizations to this algorithm. Using an optimized grid allocation strategy, restructuring code to optimize memory access and exploit on-chip instruction level parallelism better, we were able to almost double the performance of the original algorithm.

Another approach for improving the overall execution time was to use a more efficient base algorithm. The algorithm proposed by Sun [28] uses an interpolated function to avoid the lengthy computation of the Legendre sums in the original Cuffin algorithm. This version brings another ten-fold increase in performance.

IV. SOFTWARE ARCHITECTURE

Traditional EEG processing systems follow the traditional batch processing style of computation; data is input from a file, a set of algorithms are executed on the data and the results are saved in output files. They are also CPU-centric, i.e. the CPU controls the entire execution flow as well as performs the heavy mathematical computations. This type of software architecture is the result of the type of traditional computer architecture used in the past few decades that are often commonly referred to as “Denial Architecture” [29]. They pretend that computers operate sequentially and that the memory architecture is flat. GPU architecture explicitly exposes the underlying parallel hardware components and encourage the user to take advantage of locality in the hierarchical memory organization offering new opportunities for increasing performance and optimizing execution.

Our software architecture follows the explicit parallelism design principle and is based on the stream computing paradigm. The basic underlying model is of the data pipeline. Data is pumped from the measurement instrument or fetched from the data file, as required, and passed on through FIFO channels to processing nodes. The nodes output data to further processing or visualization nodes. The execution of the algorithms are triggered by the arrival of data and is carried out with maximum level of hardware parallelism. Other key design decisions were to remove any unnecessary data movement (keep as much data as possible on the GPU) and overlap CPU-GPU data transfer with computation to hide data latency.

A. Visualization

The visualization subsystem is implemented in OpenGL. We have avoided the use of any open-source visualization toolkit to retain maximum design freedom. Since the primary visualization aim is the display of time-varying potential (or potential-derivative) data on a static 2D or 3D geometry, head and electrode cap geometry data is cached into the GPU memory on initialization. Only potential data is updated continuously. A thin Java object wrapper layer is created to structure the view into higher-level objects (head model, electrodes, etc.) and to control the underlying OpenGL implementation.

B. Computation

Potential data is processed and transformed by the GPU using CUDA kernels. The largest potential performance penalty in a GPU-based computational and visualization system is if the GPU-computed results are copied back to the host in order to be displayed by the visualization system. The CUDA-OpenGL interoperability architecture allows us to avoid this by directly modifying OpenGL data structures from CUDA kernels.
Our algorithms are designed for scalability. As new GPU processors appear with more compute cores, the implementation automatically scales. If needed, the system can run with several GPU cards installed, in which case computation and visualization can be partitioned to separated GPU devices.

The CUDA threads are called via the JCuda wrapper library. To better manage the GPU code sections, the kernels are embedded in Java processor objects that manifest high-level EEG operations, e.g. MovingAverageFilter, SphericalLaplace, etc. These objects provide a clean programming interface to the rest of the system.

V. RESULTS

Our prototype EEG processing and visualization system is operational. The current system consists of an Intel Core-i5 hosted NVIDIA Quadro K4200 card (1344 cores, 2.1 teraflops) and four 46” Samsung edge-lit display panels arranged as a 2×2 grid. The surface area and the 4K resolution of the display wall is sufficient to simultaneously display the EEG measurement plots, the 2D and 3D potential and Laplacian maps as well as RMI images and (if available) video recordings of epileptic patients.

The EEG plot (Fig. 3) allows for amplitude and time axis zoom, selection of measurement sample with marker. The marker can be moved or the measurement can be replayed at selectable speed. Fig. 4 illustrates the result of the surface Laplacian computation and its visualization on a 3D realistic head model. The focusing effect the of the Laplacian is clearly visible.

A. Achieved computational performance

Our system has achieved considerable increase in computational performance. As shown in Table IV, using 128 electrodes and up to 2K (N=50) or 8K (N=15) surface points, the system can compute and visualize the Laplacian map real-time up to 2 kHz sampling frequency. The execution times shown in Tables IV and V are measured on an NVIDIA GTX 980 GPU card (2048 cores).

The performance results for the forward problem are similarly impressive. Using several code optimization steps we were able to improve the performance of the GPU-version of the Cuffin-Cohen algorithm [27]. Using Sun’s algorithm could further increase the performance and reach msec range execution time for 128 channels and 5000 dipoles. The results are displayed in Table V.

### Table IV. GPU-based Surface Laplacian Performance (128 Electrodes)

<table>
<thead>
<tr>
<th>Number of surface points</th>
<th>Execution time [msec]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N=15</td>
</tr>
<tr>
<td>1024</td>
<td>0.144</td>
</tr>
<tr>
<td>2048</td>
<td>0.152</td>
</tr>
<tr>
<td>4096</td>
<td>0.293</td>
</tr>
<tr>
<td>8192</td>
<td>0.444</td>
</tr>
<tr>
<td>16384</td>
<td>0.752</td>
</tr>
<tr>
<td>32768</td>
<td>1.386</td>
</tr>
</tbody>
</table>

### Table V. GPU-based Spherical Forward Solution Performance

<table>
<thead>
<tr>
<th>Number of dipoles</th>
<th>Cuffin &amp; Cohen</th>
<th>Sun-Stok</th>
</tr>
</thead>
<tbody>
<tr>
<td>E=64</td>
<td>E=128</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.341</td>
<td>0.720</td>
</tr>
<tr>
<td>10</td>
<td>0.402</td>
<td>0.891</td>
</tr>
<tr>
<td>100</td>
<td>0.626</td>
<td>1.337</td>
</tr>
<tr>
<td>1000</td>
<td>1.721</td>
<td>3.374</td>
</tr>
<tr>
<td>10000</td>
<td>8.852</td>
<td>17.002</td>
</tr>
</tbody>
</table>

Figure 3. Screenshot of the prototype showing the multi-channel EEG plot and the 2D Laplacian map computed from the potentials.

Figure 4. 3D visualization of the GPU-based surface Laplacian map generated from the measured scalp potentials. The realistic head model is segmented from real patient MRI data using the Freesurfer program package. The skull and the cortex layers are not shown.
VI. CONCLUSION

This paper presented a novel, GPU-based stream-oriented EEG processing software system. We demonstrated that the GPU architecture is well suited to simultaneous EEG visualization and processing. The computational performance offered by current GPUs make it possible to carry out simultaneous computation and visualization in real-time, or online if live measurement is performed. We were also interested in exploring new architectural and performance optimization methods that could lead towards the development of EEG software systems higher flexibility and with several orders of magnitude larger performance than that available today.

We have demonstrated with two important imaging algorithms how a GPU-based stream-oriented system can be created, and showed that computational performance up to 3 orders of magnitude higher than Matlab can be achieved while providing simultaneous interactive 2D/3D visualization. Our forward solver implementation is faster than any other GPU-based methods known to us in the literature. Our fast implementation of the surface Laplacian can serve as a basis for high-resolution cognitive studies and BCI applications. The system is integrated into a 4K visualization in real-time, or online if live measurement is performed while providing simultaneous interactive 2D/3D visualization in real-time, or online if live measurement is performed. We were also interested in exploring new architectural and performance optimization methods that could lead towards the development of EEG software systems higher flexibility and with several orders of magnitude larger performance than that available today.

ACKNOWLEDGMENT

The authors wish to thank their students A.G. Nagy (MSc) and D. Kenyeres (BSc) for their invaluable help in the development of the prototype system described in this paper.

REFERENCES