7th Annual International IEEE EMBS Conference on Neural Engineering
Montpellier, France, 22 - 24 April, 2015

Mulitchannel Real Time Spike Sorting for Decoding Ripple Sequences

Ankit Sethi[1] and Caleb T. Kemere[1,2]

Abstract— **In the CA1 region of the rat hippocampus, fast field oscillations termed sharp wave ripples have been identified as playing a crucial role in memory formation and learning. During ripple activity, particular sequences of neurons fire in a phenomena called replay. So termed because the spiking encodes patterns of past experiences, the exact role of the content of replay is an active subject of investigation in order to determines its relationship with learning and memory guided decision making. A need arises for systems that can decode replay activity during ripples in real time. This necessitates fast algorithms for both spike sorting and ripple detection with the lowest possible latency. A low latency implementation makes possible feedback experiments where decoded ripple sequences can, with minimal delay, trigger stimulating pulses that can disrupt particular kinds of decoded information before they can contribute to behavior. In this study, we optimize and implement a recently proposed online spike sorting algorithm for an increasingly popular electrophysiological software suite and measure improvements that greatly enhance its multi-tetrode decoding capabilities. Synchronizing with online ripple detection, this novel framework will allows experimenters to study the effects of disrupting replay activity with a degree of granularity hitherto unavailable.**

I. INTRODUCTION

Hippocampal replay is a sequential reactivation of neuronal sequences that resemble activity during an animals awake state but at an increased time scale[1]. Replay is believed to have a role in memory consolidation of spatial tasks[2]. Sharp wave ripples are fast oscillations (>150 Hz) that are associated with replay in the hippocampus[3]. To establish a causal link memory consolidation and replay it essential to detect ripples and decode concurrent replay information for feedback experiments that seek to disrupt specific patterns of replay. This requires a combination of online spike sorting and ripple detection. The authors of [4] have proposed an algorithm for real time spike sorting that they called `OPASS` (Online gamma Process Autoregressive Spike Sorting). This algorithm possesses several attractive features a) it is non-parametric. b) it offers real time performance and c) models time-varying statistics. The implementation devised by them is unoptimized but works in real time for a single channel and four-channel scenario, though in the latter case it takes 75% of data length in execution time. However, in the current state of electrophysiology, neural recordings often usually involve at least 16 electrodes with 4 channels each, giving us a minimum of 64 total channels that needs to be decoded. Recording experiments with up to 512 channels have also been performed [5] and seem to be the future norm for such research. Specifically, given the difficult nature of detecting replay in awake state [6], it is important to have the ability to perform spike sorting of several tetrodes simultaneously. Additionally, the algorithm in [4] is operated at a sampling rate of 10kHz while it is common in studies to record at data rates of up to 30-40 kHz [7]. This further increase in data rate by a factor of 3-4 represents an additional challenge for online systems. Our goal was to implement a low-latency online algorithm for spike sorting in C++ within the framework of open-ephys and combine it with a previously presented algorithm for low latency ripple detection [8] to perform decoding of replay firing patterns during ripple sequences. The eventual goal is to use this to perform closed loop experiments involving probing stimuli that can react based on real time decoding of information in ripple sequences. The first section introduces the existing form of `OPASS` and our proposed modifications along with a revised algorithm that significantly lowers the slope of algorithm time required per number of samples processed. Next, we present an outline of our software framework and some discussion about the tools used to create a version ready for use in experiments. In the next section, we explain how ripples and spikes data is concurrently extracted and combined. Finally, we present the results of timing tests to illustrate the improvements in execution time that bend the slope downwards enough to make the online decoding of several tetrodes feasible, and conclude with a discussion about directions for further improvement and investigation.

[1]Department of Electrical and Computer Engineering, Rice University, Houston `ankit.sethi@rice.edu`
[2]Department of Neuroscience, Baylor College of Medicine, Houston `caleb.kemere@rice.edu`

II. THE OPASS ALGORITHM

A. Structure

The algorithm begins by assigning a series of prior probabilities for spike shapes and encountering a new spike. Data is processed in batches of $L = 500$ (at 10kHz) samples. A rolling time window moves across the batch and calculates the likelihoods of the data segment being a) noise, b) a previously identified neuron or c) a new neuron. Posterior distributions of all putative neurons and their past spike times are maintained and updated after every new detection. C is the number of neurons found up to the current iteration, l_θ are the prior likelihoods of seeing a new neuron in a Chinese Restaurant Process (CRP), A is the dictionary of K columns, Σ is the noise correlation matrix (invertible due to Toeplitz AR(1) structure and $P >> 1$), and $\mathbf{Q}(c)$ is the posterior shape distribution of neuron c. The size of a window is P samples. A full description may be found in [4].

Algorithm:

978-1-4673-6389-1/15/$31.00 ©2015 European Union

$C = 1$;
$\mathbf{x_w} : L \times (L - P)$ matrix of sliding batch segments;
while *DAQ active* **do**
  Update prior $\mathbf{l}_\theta(1 : C)$ using the CRP
  Update noise likelihood:
  $\mathbf{l_{noise}} = -\frac{P}{2}\log(2\pi) + \frac{1}{2}\log(|\Sigma^{-1}|) - \frac{1}{2}\mathbf{x_w}^T\Sigma^{-1}\mathbf{x_w}$
  for $c = 1 : C - 1$ **do**
    $\mathbf{Q_t}(c) = \lambda_{clus}^{-1}(c) + R^{-1}(c)$
    $\mathbf{Q}(c) = \mathbf{\Sigma} + \mathbf{A}\mathbf{Q_t}(c)\mathbf{A}^T$
    $\bar{\mathbf{x}}_\mathbf{w} = \mathbf{x_w} - \mathbf{A}\mu(:, c)$
    Update likelihood of neuron c;
    $\mathbf{l_{on}}(c) =$
    $-\frac{P}{2} - \frac{1}{2}\log(|\mathbf{Q}^{-1}(c)|) - \frac{1}{2}\bar{\mathbf{x}}_\mathbf{w}\mathbf{Q}^{-1}(c)\bar{\mathbf{x}}_\mathbf{w}$
  end
  $\mathbf{Q_t} = \lambda_{\mathbf{clus}}^{-1}(C) + \mathbf{R}^{-1}(C)$
  $\mathbf{Q} = \mathbf{\Sigma} + \mathbf{A}\mathbf{Q_t}\mathbf{A}^T$
  Update likelihood of new neuron C;
  $\mathbf{l_{on}}(C) =$
  $-\frac{P}{2}\log(2\pi) - \frac{1}{2}\log(|\mathbf{Q}^{-1}(C)|) - \frac{1}{2}\mathbf{x_w}\mathbf{Q}^{-1}(C)\mathbf{x_w}$
  $\mathbf{l_{on}} = \mathbf{l_{on}} + \mathbf{l}_\theta$
  $\mathbf{H} = \mathbf{l_{on}} - \max(\mathbf{l_{on}})$
  $\mathbf{l_{thr}} = \mathbf{l_{noise}} - \max(\mathbf{l_{on}}) - \log\sum(\exp(\mathbf{H}))$
  idx = `find` $(\mathbf{l}_{thr} < T)$
  if *size (idx) > 0* **then**
    Find spike peak and offset idx accordingly;
    Find which neuron ID most likely candidate:
    $C_{new} = \max(\mathbf{l_{on}}(idx))$
    if $C_{new} > C - 1$ **then**
      $C = C_{new}$
    end
    update $\mathbf{R}, \lambda_{\mathbf{clus}}$ and others.
    calculate K-dim dictionary projection: $\hat{\mathbf{y}}$
  end
end

Fig. 1. OPASS algorithm (condensed)

B. Changes for real-time implementation

The following algorithmic changes were made:

1) OPASS processes data in a "small" batch of L ($= 500$) samples. A window of length R (≈ 50) slides over the batch length in increments of one sample. Likelihoods for the entire batch are calculated after which it searches for a likelihood value crossing a threshold. With the objective of keeping latency to a minimum, we removed the batch requirement, directly processing the incoming data in firehose fashion. Circular buffers of length R for each channel were used as windows, reducing the latency from L to R samples.
2) When OPASS detects a neuron, it searches among the near-future likelihoods over a time period of one spike length to identify the peak of the spike. This is needed for proper alignment of the detected spike before updating distribution parameters. Since likelihoods for the entire batch are already available at this point, this step is trivial. Our version processes incoming samples as they comes (save for the circular buffer latency), so it goes into a search mode after a threshold crossing where it briefly stops looking for further neurons, but keeps calculating likelihoods for successive windows over the range of a spike length. It then identifies the window with waveform most likely to be a spike, aligns, and proceeds with parameter updating.
3) All invariants terms in the likelihood equations were pre-calculated. The noise autocorrelation matrix Σ was estimated from the data before sorting and its inverse and determinant were also pre-calculated and stored.
4) Workarounds were devised for regularly updated matrix inverse operations. In timing tests, it was found that the most computationally expensive steps are in updating $\mathbf{Q}$ and then calculating $\mathbf{Q}^{-1}$ and $|\mathbf{Q}^{-1}|$. These costs were mitigated in two ways:
   a) $\mathbf{Q}$ changes after each neuron detection. Instead of calculating $\mathbf{Q}$ for every window, it was updated only after successful detections and reused otherwise.
   b) The update equation for $\mathbf{Q}$, $\mathbf{Q} = \mathbf{\Sigma} + \mathbf{A}\mathbf{Q_t}\mathbf{A}^T$, involves a low rank matrix ($\mathbf{Q_t}$) changing with each update. The matrix inversion lemma and its corollary for determinants can be profitably used to reduce calculations since $\mathbf{Q_t}$ is of dimension $K \times K$. $K \ll P$ since K is in the range of $2 - 5$ while P is between $30 - 50$, depending on the window size.

C. Revised Algorithm

Our revised algorithm proceeds as outlined below. $x(n)$ represents the incoming data for one channel with n denoting the most recent time index.

The revised algorithm is presented in Figure 2. For multiple channels, this procedure is easily generalized. The full code is available online at: SpikeSorter Module for OpenEphys.

D. Pre- and Post-Processing

The OPASS algorithm utilizes a dictionary based method to identify spikes. The authors of [4] use spike detection using power thresholding to store an array of spike waveforms. An SVD is applied to this spike dataset and the first K columns of the left-singular matrix are used for the dictionary. In testing our implementation we discovered that the algorithm is sensitive to improper alignment of spikes. Misalignment results in a dictionary that is less representative of the underlying neuronal population and during sorting, this leads to spurious and inordinately high number of putative neurons being identified. This has important implications for real time performance: the parameter C (which controls the number of iterations of the likelihood loop) increments steadily slowing down the the algorithm and leading to neurons being misidentified and mislabeled. In the multi channel, multi-electrode case, with little leeway w.r.t. execution time, this can quickly lead to sub real time performance. The OPASS algorithm also requires the predetermination of the

$C = 1$;
$\mathbf{x_w} : L \times (L - P)$ matrix of sliding batch segments;
while *DAQ active* **do**
  Enqueue new sample $x(n)$ in circ. buffer $\mathbf{x_w}$
  Update prior $\mathbf{l}_\theta(1:C)$ using the CRP
  Update noise likelihood:
  $\mathbf{l_{noise}} = -\frac{P}{2}\log(2\pi) + \frac{1}{2}\log(|\Sigma^{-1}|) - \frac{1}{2}\mathbf{x_w}^T\Sigma^{-1}\mathbf{x_w}$
  for $c = 1 : C - 1$ **do**
    $\bar{\mathbf{x}}_\mathbf{w} = \mathbf{x_w} - \mathbf{A}\mu(:, c)$
    Update likelihood of neuron c;
    $\mathbf{l_{on}}(c) =$
    $-\frac{P}{2} - \frac{1}{2}\log(|\mathbf{Q}^{-1}(c)|) - \frac{1}{2}\bar{\mathbf{x}}_\mathbf{w}\mathbf{Q}^{-1}(c)\bar{\mathbf{x}}_\mathbf{w}$
  end
  Update likelihood of new neuron C;
  $\mathbf{l_{on}}(C) =$
  $-\frac{P}{2}\log(2\pi) - \frac{1}{2}\log(|\mathbf{Q}^{-1}(C)|) - \frac{1}{2}\mathbf{x_w}\mathbf{Q}^{-1}(C)\mathbf{x_w}$
  $\mathbf{l_{on}} = \mathbf{l_{on}} + \mathbf{l}_\theta$
  $\mathbf{H} = \mathbf{l_{on}} - \max(\mathbf{l_{on}})$
  $\mathbf{l_{thr}} = \mathbf{l_{noise}} - \max(\mathbf{l_{on}}) - \log\sum(\exp(\mathbf{H}))$
  if $l_{thr} < T$ **then**
    `spikeDetect` = **true**
    `search` = **true**, `i`= 0
  end
  if *spikeDetect* = ***true*** **then**
    if *search* = ***true and*** $i <$ *range* **then**
      store $\mathbf{l_{thr}}$, $\mathtt{i} = \mathtt{i} + 1$
      if $i ==$ *range* **then**
        `search` = **false**
      end
    end
    if *search* = ***false*** **then**
      `idx` = `find` $(\min(l_{thr}))$
      $C_{new} = \max(l_{on}(\mathtt{idx}))$
      if $C_{new} > C - 1$ **then**
        $C = C_{new}$
      end
      Update $\mathbf{R}(C_{new})$, $\lambda_{\mathbf{clus}}(C_{new})$ and others
      Update $\mathbf{Q}(C_{new})$, calculate and store
      $\mathbf{Q}^{-1}(C_{new}), |\mathbf{Q}^{-1}(C_{new})|$
      Calculate projection on dict. space $\hat{\mathbf{y}}$
    end
  end
  Dequeue sample at head of circ. buffer
end

Fig. 2. Revised `OPASS` algorithm (condensed)

noise autocorrelation matrix Σ if the noise is being modeled as an AR(1) process. This is also implemented in the pre-processing stage with dictionary estimation. Care must be taken in inverting the matrix as there is a risk of close to not being full rank. However, most matrix libraries provide methods to check for this. After a spike is detected and sorted, the algorithm provides us with the following outputs the timestamp for the spike peak, the spike waveform itself, the K coefficients of the spike in dictionary-space, and the neuron ID assigned to the spike. This meta-data is encapsulated and forwarded to further processing steps for decoding and/or display.

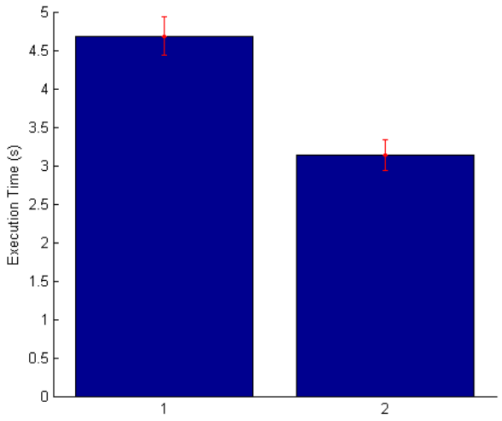

Fig. 3. Execution time of the original OPASS (1) and the optimized OPASS (2).

III. SOFTWARE IMPLEMENTATION

The `OPASS` algorithm was chosen with the view of using a non-/minimally supervised algorithm for fast online detection and sorting during recording experiments. Our aim was to add the implementation as a module to `open-ephys`, a set of collaborative, open-sourced software modules for extra-cellular recordings with a focus towards enabling low cost high quality multichannel data acquisition and processing. The open-ephys acquisition board is used in conjunction with its open-source GUI to display, process, record and save neural activity. For linear algebra functionality, we turned to `Eigen`, because of several attractive features it is header based, has an easy to use API, open-source license, and performance rivaling or exceeding most competitors like LAPACK or Armadillo.

IV. SPIKE SORTING AND RIPPLE DETECTION

In a previous work, we have identified and tested low latency algorithms for ripple detection and proposed a variant of the CUSUM algorithm for ripple detection. Towards the end of decoding replay during ripples, we implemented spike sorting in a software framework that synchronizes between online ripple detection and sorting to identify and label spikes that occur during ripples with minimal latency. Specifically, a downstream module was developed that reconciles spike events and ripple events to accurately mark spikes occuring during ripples in real time. The output from this novel coprocessing approach can be used by processing further modules downstream for online decoding of place cell sequences, calculating spatial trajectories etc. The modular nature of open-ephys ensures that algorithmic improvements can be made in future without affecting the overarching framework.

V. RESULTS

We tested the effect of structural changes to the algorithm on an Intel Core i5 3.0 Ghz processor and 4 GB RAM. MATLAB simulations run over 500 iterations for a simulated

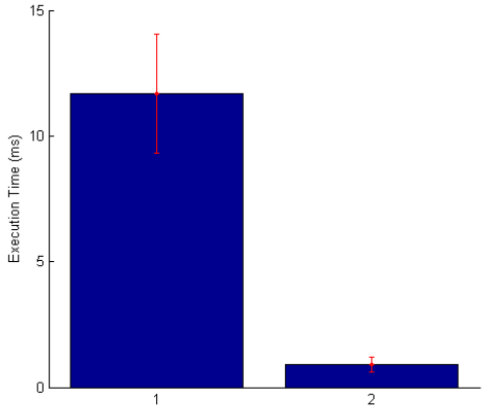

Fig. 4. Execution time of 45 ms of spike data in C++ implementation in original (1) and fully optimized (2) form.

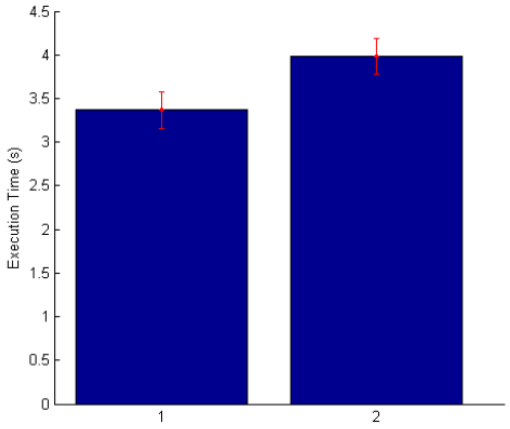

Fig. 5. Execution time of an OPASS run with 5 (1) and 7 (2) putative neurons detected.

dataset showed a drop in execution time (Figure 3). The mean reduction in execution time is 32.9%. Figure 4 shows a comparison in execution time for 45 ms of data in the open-ephys C++ implementation. Here we compared the speed of the original implementation with the our fully optimized version. Apart from the structural changes, matrix optimizations specific to Eigen were also incorporated. The final version achieves a 92.3% decrease in mean execution time.

VI. DISCUSSION

One of the issues identified over the course of this project has been the increase in execution time when spurious neurons are detected. This may be caused by several issues sharp bursts of noise, improper alignment in preliminary spike detection and a high setting for α, the CRP parameter that controls the prior probability of detecting a new neuron. Since the algorithm updates and learns the posterior distributions of each identified neurons shape, a few spurious and transient detections are expected to be observed at the beginning of the sporting. If the total neuron types steadily accumulates, however, the overhead is liable to be high since the algorithm then checks for these neuron types in every time window. In Figure 5 we profiled the performance drop when a change in yields 5 and 7 detected neurons, respectively. Two extra detections ends up increasing the mean execution time by 18%. Some design considerations that are left unsettled in [4] are the best way to calculate the dictionary and auto-correlation matrices for tetrodes. Separate tetrodes are likely to be recording a different signal from each other, but the channels of a tetrode should be highly correlated. We estimated the dictionary using the channel with the best signal quality but it is possible that the signal from all channels could be optimally combined for the purposes of this preliminary estimation.

VII. CONCLUSIONS

The improvements in speed achieved enable up to 5 tetrodes to be spike-sorted in real time on standard desktop hardware. A promising direction for future improvements is creating a sub-algorithm that monitors the statistics of detected neurons to identify and de-activate detection of rarely spiking or spuriously detected neuron patterns. We united an optimized spike sorting algorithm with a previously developed ripple detection algorithm. A downstream module combines event data from both processors to mark sorted spikes occurring during ripples; this can then be used for online experiments investigating the content and significance of replay.

REFERENCES

[1] Buhry, Laure, Amir H. Azizi, and Sen Cheng. “Reactivation, replay, and preplay: how it might all fit together.” Neural plasticity 2011 (2011).

[2] Carr, Margaret F., Shantanu P. Jadhav, and Loren M. Frank. “Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval.” Nature neuroscience 14, no. 2 (2011): 147-153.

[3] Nakashiba, Toshiaki, Derek L. Buhl, Thomas J. McHugh, and Susumu Tonegawa. “Hippocampal CA3 output is crucial for ripple-associated reactivation and consolidation of memory.” Neuron 62, no. 6 (2009): 781-787.

[4] Carlson, David, Vinayak Rao, Joshua T. Vogelstein, and Lawrence Carin. “Real-Time Inference for a Gamma Process Model of Neural Spiking.” In Advances in Neural Information Processing Systems, pp. 2805-2813. 2013.

[5] Bernyi, Antal, Zoltn Somogyvri, Anett J. Nagy, Lisa Roux, John D. Long, Shigeyoshi Fujisawa, Eran Stark, Anthony Leonardo, Timothy D. Harris, and Gyrgy Buzski. “Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals.” Journal of neurophysiology 111, no. 5 (2014): 1132-1149.

[6] Davidson, Thomas J., Fabian Kloosterman, and Matthew A. Wilson. “Hippocampal replay of extended experience.” Neuron 63, no. 4 (2009): 497-507.

[7] Jadhav, Shantanu P., Caleb Kemere, P. Walter German, and Loren M. Frank. “Awake hippocampal sharp-wave ripples support spatial memory.” Science 336, no. 6087 (2012): 1454-1458.

[8] Sethi, Ankit, and Caleb Kemere. “Real time algorithms for sharp wave ripple detection.” In Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE, pp. 2637-2640. IEEE, 2014.