\chapter{Implementing Neural Response and JND evaluation}

\label{sec:anr-to-jnd}

\phantomsection

\section{Neural Response Calculation}

\label{sec:neural\_response\_calculation}

Here we implement a massively parallel computing process to calculate the neural response from the results of \cref{sec:parrallelizing-the-algorithm} for the BMV solution.

\begin{figure}

\centering

\includegraphics[width=0.45\textwidth,keepaspectratio=true]{figs/tikzlambda}

\vspace{.2in}

\caption{Flow chart for calculating ANR.}

\label{fig:ANResponse}

\end{figure}

As shown in \cref{sec:anr-calculation-model}, the neural response can be described as a homogeneous Poisson response when no signal is present or as a nonhomogeneous Poisson (NHHP) response when a signal is present.

To calculate the neural response, the IHCs must first be calculated by parallelizing the equation \cref{eq:psi-ihc}.

The quantity $h\_{ihc}$ can be any linear filter; the parameters chosen for the tests are given in \cref{tab:Lambda-parameters}.

This is paralleled by first calculating the IIR filter coefficients locally, and then solving on the GPU. Because IIR is recursive, parallelization is done over the longitude dimension and over multiple power levels (JND section) but not over the time domain. The calculation is done by convolving the output for each section by $b$ and then recursively convolving the result with negative $a$ coefficients. Implementation of the \ac{ac} part from \cref{eq:psi-ihc} takes the form

\begin{equation}

\label{eq:gpu-ac-response}

\begin{aligned}

AC\_{response}(x,t) =& \sum\limits\_{i=0}^{n}\left[b(i)\*BM\_{velocity}(x,t-i)\right]\\

& - \sum\limits\_{i=1}^{n}\left[a(i)\*AC\_{response}(x,t-i)\right],

\end{aligned}

\end{equation}

where $n$ is the number of coefficients in $h\_{ihc}$, $t$ is time index (i.e., the result of $t T\_s$), $x$ is the \cochposition, $BM\_{velocity}$ is the sampled BMV. The result is $\dxi\_{bm}$ from \cref{eq:dxi-ihc} and $AC\_{response}$. The program also supports the use of far-infrared filters for this calculation. If the far-infrared filter is chosen, parallelization is also done over the time domain.

We use the AC part of the IHC response to calculate the DC part, $\{\dot{\xi}\_{ihc}(x,t)[1-h\_{ihc}(t)]\}^{2}$,

by calculating the intermediate result

\begin{equation}

\label{eq:ds-intermediate}

dS\_{high}(x,t)= \left\{BM\_{velocity}(x,t)- AC\_{response}(x,t)\right\}^{2}.\label{ds-high-equation}

\end{equation}

\Cref{eq:ds-intermediate} depends on the results of \cref{eq:gpu-ac-response,eq:dxi-ihc} at a single time index, so it can be done without synchronization. We approximate the DC part of the IHC voltage, $\int\_{t-\delta}^{t}{ \{\dot{\xi}\_{ihc}(x,t) [1-h\_{ihc}(t)] \}^{2}}\mathrm{d}t$, from \cref{eq:psi-ihc}.

We obtain the DC response by substituting $dS\_{high}$ from \cref{ds-high-equation } and synchronizing before summing:

\begin{equation} \label{eq:dc\_response}

\begin{aligned}

DC\_{response}(x,t) = \frac{1}{F\_s\*\delta}\*\sum\limits\_{t\_1=t-F\_s\*\delta}^{t}dS\_{high}(x,t\_1),

\end{aligned}

\end{equation}

because \cref{eq:dc\_response} relies on multiple time indexes. Calculating $\log\_{10}{\bm(\psi\_{ihc}(x,t)\bm)}$ as

\begin{equation} \label{eq:combined\_response}

\voltihc(x,t) = \eta\_{AC} AC\_{response}(x,t) + \eta\_{DC} DC\_{response}(x,t)

\end{equation}

and substituting \cref{eq:combined\_response} into \cref{eq:psi-ihc}, we obtain

\begin{equation} \label{eq:calc\_ihc}

\begin{split}

\MoveEqLeft

\log\psi\_{ihc}(x,t) = \log\_{10}{\big(\psi\_{ihc}(x,t)\big)}\\

& =\log\_{10}\big(\epsilon+\max\{0,({10}^{\gamma\_{ihc}(x)}\voltihc)\}\big).

\end{split}

\end{equation}

The stages of computation are divided to ensure both data integrity (i.e., avoid reading result that is not yet calculated) and maximum speed. Because results that depend on another time sample occur only in filters (i.e., calculating the $AC\_{response}$) and in the stages of \cref{eq:dc\_response}, all calculations that can be done with a single time coordinate are unified into a single function, such as the result of\cref{eq:ds-intermediate} or the \cref{eq:calc\_ihc} stage [implementing $\ln\{1+u[\psi\_{ihc}(x,t)]\}$ from \cref{eq:an-response}]. This situation arises because lambda has three sets for the different types of neurons \(see cref{sec:anr-response-theory}), as found by \cite{odedst2017}. $A\_{ihc}$ thus depends both on the nerve type and on spatial position, and the solution for \cref{eq:an-response} is

\begin{eqnarray}

\lambda\_{AN}^{H}(x,t)=\min\left\{\lambda\_{sat},max\left\{\lambda\_{spont}^{(\ac{hsr})},A\_{ihc}^{(HSR)}(x) \log\psi\_{ihc}\right\}\right\}, \\

\lambda\_{AN}^{M}(x,t)=\min\left\{\lambda\_{sat},max\left\{\lambda\_{spont}^{(\ac{msr})},A\_{ihc}^{(MSR)}(x) \log\psi\_{ihc}\right\}\right\}, \\

\lambda\_{AN}^{L}(x,t)=\min\left\{\lambda\_{sat},max\left\{\lambda\_{spont}^{(\ac{hsr})},A\_{ihc}^{(LSR)}(x) \log\psi\_{ihc}\right\}\right\}.

\end{eqnarray}

\section{JND Calculation}

\label{sec:jnd\_calculation}

\begin{figure}

\centering

\includegraphics[width=0.85\textwidth,keepaspectratio=true]{figs/tikzjndcalculation}

\vspace{.2in}

\caption{Flow chart for calculation of JND.}

\label{fig:jndcalculation}

\end{figure}

Here we implement the \ac{jnd} calculation by using the Rate Mean Square and All Information methods. We solve \cref{eq:ai-crlb,eq:ra-crlb},

where $\alpha$ is the signal amplitude and both inputs signal plus noise and noise only are calculated together. The computation of the JND is ordered so that all dependence on another time interval is first solved. This is done because CUDA cannot synchronize between blocks on the same kernel run \cite{NVIDIA\_Programming\_guide\_Block\_Structure}.

\phantomsection

\subsection{Calculation of Fisher information}

\label{sec:cal-fisher-information}

\phantomsection

\paragraph{Average neural response over time.}

\label{par:average-neural-response-time}

The first stage is to do the calculation for each signal and for every type of lambda:

\begin{equation}

\label{eq:lambda-time-average}

\begin{aligned}

&&\mathsmaller{AN} & \in & (HSR,MSR,LSR) ,\\

&&\alpha & \in & \{\stalpha\_{NL}+\Delta\alpha\_{SL},\stalpha\_{NL}\}, \\

&&\overline{\lambda}\_{AN,\ac{rms}}(x,\alpha) & = & \inv{T} \sum\_{t=0}^T \lambda\_{\ac{an}}(x,t,\alpha). \\

\end{aligned}

\end{equation}

Because the calculation is done in parallel, most stages process data that did not come from another \ac{cuda} block in the JND calculation. However, the \ac{rms} of \cref{eq:lambda-time-average} needs to subtract the average of the reference signal (silence or noise only) because each signal is calculated in separate CUDA blocks,

so the averaging is done in different kernels to synchronize all the data \cite{NVIDIA\_Programming\_guide}.

From this point on, all threads only calculate their own data (or data processed in previous kernels).

\paragraph{Calculate Fisher formula for each neural group.}

\label{par:calculate-fisher-pergroup}

To evaluate \ac{crlb} in \cref{eq:ai-crlb,eq:ra-crlb}, we substitute $\lambda(\alpha,x)$

into \cref{eq:lambda-time-average} to obtain

\begin{equation}

\label{eq:delta-lambda-ra}

\begin{aligned}

\Delta\lambda\_{AN,RA}(x,\stalpha\_{NL},\Delta\alpha\_{SL}) = \frac{\overline{\lambda}\_{AN,\ac{rms}}(x,\stalpha\_{NL}+\Delta\alpha\_{SL}) - \overline{\lambda}\_{AN,\ac{rms}}(x,\stalpha\_{NL})}{\Delta\alpha\_{SL}}.

\end{aligned}

\end{equation}

To obtain AI we implement \cref{eq:crlb-approx} for each longitudinal section $x$ by substituting $\alpha$ with $\stalpha\_{NL}$ and $\Delta\alpha$ with $\Delta\alpha\_{SL}$ to determine the signal-to-noise ratio. Since \ac{anr} is calculated in a different kernel, data synchronization is guaranteed. We thus obtain

\begin{equation}

\label{eq:delta-lambda-ai}

\begin{aligned}

\Delta\lambda\_{AN,\ac{ai}}(x,t,\Delta\alpha\_{SL},\stalpha\_{NL}) & = && \frac{\lambda\_{AN}(x,t,\stalpha\_{NL}+\Delta\alpha\_{SL}) - \lambda\_{AN}(x,t,\stalpha\_{NL})}{\Delta\alpha\_{SL}}.

\end{aligned}

\end{equation}

Each different $\lambda$ from the same longitudinal and temporal coordinates from nerve response to noise only input (or silence), as noted, parallelization utilized to compute response for different $\alpha$ in this case amplitude in parallel, so response is available.

We then integrate \cref{eq:delta-lambda-ai} over time to obtain

\begin{equation}

\label{eq:fisher-unweighted-ai}

\centering

\begin{aligned}

Fisher\_{AN,\ac{ai}}^{Unweighted}(x,t,\Delta\alpha\_{SL},\stalpha\_{NL}) & = && \frac{\Delta\lambda\_{AN,\ac{ai}}^2(x,t,\Delta\alpha\_{SL},\stalpha\_{NL})}{\lambda\_{AN}(x,t,\stalpha\_{NL})}.

\end{aligned}

\end{equation}

We calculate the Fisher formula for AI by first aggregating the results of each longitudinal section \cref{eq:fisher-unweighted-ai} over time to obtain

\begin{equation}

\label{eq:fisher-spaced-ai}

\begin{aligned}

Fisher\_{AN,\ac{ai}}^{Spaced}(x,\Delta\alpha\_{SL},\stalpha\_{NL})=\frac{M(x)}{T} \cdot \sum\_{t=0}^{T} Fisher\_{AN,\ac{ai}}^{Unweighted}(x,t,\Delta\alpha\_{SL},\stalpha\_{NL}).

\end{aligned}

\end{equation}

We substitute \cref{eq:lambda-time-average} into \cref{eq:ra-crlb} to get the inverse square of $CRLB\_{RA}$ per spatial section:

\begin{equation}

\label{eq:fisher-unweighted-rms}

\begin{aligned}

Fisher\_{AN,\ac{rms}}^{Unweighted}(x,\Delta\alpha\_{SL},\stalpha\_{NL}) = \frac{\overline{\lambda}\_{AN,\ac{rms}}(x,\stalpha\_{NL})}{T\cdot \Delta\lambda\_{AN,\ac{rms}}^2(x,\stalpha\_{NL},\Delta\alpha\_{SL})}.

\end{aligned}

\end{equation}

We than substitute \cref{eq:fisher-unweighted-rms} into \cref{eq:ra-crlb} and multiply by the number of nerves per section to get $CRLB\_{RA}$:

\begin{equation}

\begin{aligned}

Fisher\_{AN,RA}^{Spaced}(x,\Delta\alpha\_{SL},\stalpha\_{NL})=\frac{M(x)}{Fisher\_{AN,\ac{rms}}^{Unweighted}(x,\Delta\alpha\_{SL},\stalpha\_{NL})}.

\end{aligned}

\end{equation}

\ac{crlb} uses the nerve-density function $M(x)$ as per \cite{odedst2017} to calculate the Fisher information in both methods (\RA and \AI) by averaging over the longitudinal dimension.

\paragraph{Aggregate Fisher information over space.}

\label{par:aggregate-fisher-space}

\begin{equation}

\begin{aligned}

Fisher\_{AN}(\Delta\alpha\_{SL},\stalpha\_{NL})=\inv{Sections} \sumsections Fisher\_{AN}^{Spaced}(x,\Delta\alpha\_{SL},\stalpha\_{NL}),

\end{aligned}

\end{equation}

because $Sections=256$, so we parallelize the process by using the modified parallel reduction \cref{parallel-reduction} to obtain $signals \times AN$, which contains a few hundred to a few thousand points.

\begin{lstlisting}[caption={CUDA implementation of parallel reduction algorithm page 113 of \cite{Cuda\_C\_Programming}},label={parallel-reduction}]

unsigned int tid=threadIdx.x;

unsigned int i=blockIdx.x\*(blockDim.x\*2)+threadIdx.x;

sm[tid] = d[i]+d[i+blockDim.x];

\_\_syncthreads();

for (stride=blockDim.x/2;stride>=1;stride>>=1)

{

if (tid<stride) sm[tid]+=sm[tid+stride];

\_\_syncthreads();

}

\end{lstlisting}

\paragraph{Aggregate Fisher information over neural groups.}

\label{par:aggregate-fisher-grups}

Because the last stage requires summing over $AN$, the use of GPU is unnecessary because the run time is a small fraction of a percent of the program, so optimizing it will not noticeably accelerate the calculation. Thus, from this stage forward, the same calculation is done for both \RA and \AI.

Solving \cref{eq:fisher-sum}, we obtain

\begin{equation}

\begin{aligned}

JND(\Delta\alpha\_{SL},\stalpha\_{NL})=\rsqrt{\sumfibers Fisher\_{AN}(\Delta\alpha\_{SL},\stalpha\_{NL})}.

\end{aligned}

\end{equation}

\paragraph{Calculate Just Noticeable Difference per noise level.}

\label{par:calc-jnd-per-level}

Because JND depends on $\alpha$, the optimal alpha must be found. To do this, we use the gradient-decent method, because the JND presents a low barrier. As shown by \cite{odedst2017}, two possible methods exist to find the optimal alpha. We do the calculation from the signal processed multiple times in parallel, as shown in \cref{sec:parallel\_input\_generation}:

\begin{enumerate} \label{subpar:jnd-types}

\item \minJND - solve by finding \(\stalpha+\dlalpha\), such that

\begin{equation\*}

\begin{aligned}

JND(\stalpha\_{NL})= & \max\_{SL} & JND(\Delta\alpha\_{\ac{sl}},\stalpha\_{NL}) \ge JND(\Delta\alpha\_{SL-1},\stalpha\_{NL}) \\ & \wedge & JND(\Delta\alpha\_{SL},\stalpha\_{\ac{nl}}) \le JND(\Delta\alpha\_{SL+1},\stalpha\_{NL}).

\end{aligned}

\end{equation\*}

\item \wantedJND - solve by finding \(\stalpha+\dlalpha\), such that

\begin{equation\*}

\begin{aligned}

JND(\stalpha + \dlalpha[1]) - JND(\stalpha + \dlalpha) > \epsilon(\Delta\alpha),

\end{aligned}

\end{equation\*}

with $\epsilon$ being a function of $\Delta\alpha$, we approximate \cref{eq:crlb-approx}.

\end{enumerate}